1995-09

Real-time articulation of the upper body for simulated humans in virtual environments

Waldrop, Marianne Susan

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

REAL-TIME ARTICULATION OF THE UPPER BODY FOR SIMULATED HUMANS IN VIRTUAL ENVIRONMENTS

by

Marianne Susan Waldrop

September 1995

Thesis Co-Advisors: Robert B. McGhee
John S. Falby

Second Reader: Shirley M. Pratt

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1995	Master's Thesis

4. TITLE AND SUBTITLE
REAL-TIME ARTICULATION OF THE UPPER BODY FOR SIMULATED HUMANS IN VIRTUAL ENVIRONMENTS

5. FUNDING NUMBERS

6. AUTHOR(S)
Waldrop, Marianne S.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The problem addressed in this thesis is that most large-scale networked virtual environments (VE) do not possess an interface to produce dynamic, real-time interactive simulated human motion. In order to attain a high level of realism in the virtual world, the user must be able to dynamically interact with his environment. For the lower body, we find that scripted locomotive motion is adequate. However, the same is not true for upper body motion because humans by their nature interact with their environment largely with their hands.

The approach taken in this thesis is to develop an interactive interface which achieves dynamic real-time upper body motion while not encumbering the user. The interface is based on inexpensive and commercially available six degree of freedom (DOF) magnetic sensor technology and fast kinematic algorithms.

The result of this work is the creation of a human upper body interface which can be extended for use in any large-scale networked interactive VE, such as NPSNET. Three sensors are strapped onto each arm of the user, which read their position and orientation and transmit this information to the software, which in turn produces the same motion of the computer human icon in real-time. An interface such as this enables participants in networked VE to more naturally interact with the environment in real-time.

14. SUBJECT TERMS	15. NUMBER OF PAGES
human interface, virtual environment, articulated humans, human modeling, magnetic sensors, Polhemus, kinematics, NPSNET, Jack, networked simulations	103
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

REAL-TIME ARTICULATION OF THE UPPER BODY FOR SIMULATED HUMANS IN VIRTUAL ENVIRONMENTS

Marianne Susan Waldrop
Captain, United States Marine Corps
B.A., University of North Carolina, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1995

Author: [redacted]

Marianne Susan Waldrop

Approved by: [redacted]

Robert B. McGhee, Thesis Co-Advisor

[redacted]

John S. Falby, Thesis Co-Advisor

[redacted]

Shirley M. Pratt, Second Reader

[redacted]

Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The problem addressed in this thesis is that most large-scale networked virtual environments (VE) do not possess an interface to produce dynamic, real-time interactive simulated human motion. In order to attain a high level of realism in the virtual world, the user must be able to dynamically interact with his environment. For the lower body, we find that scripted locomotive motion is adequate. However, the same is not true for upper body motion because humans by their nature interact with their environment largely with their hands.

The approach taken in this thesis is to develop an interactive interface which achieves dynamic real-time upper body motion while not encumbering the user. The interface is based on inexpensive and commercially available six degree of freedom (DOF) magnetic sensor technology and fast kinematic algorithms.

The result of this work is the creation of a human upper body interface which can be extended for use in any large-scale networked interactive VE, such as NPSNET. Three sensors are strapped onto each arm of the user, which read their position and orientation and transmit this information to the software, which in turn produces the same motion of the computer human icon in real-time. An interface such as this enables participants in networked VE to more naturally interact with the environment in real-time.

TABLE OF CONTENTS

I. INTRODUCTION 1
A. MOTIVATION 1
B. GOALS 1
C. ORGANIZATION 2
II. SURVEY OF PREVIOUS WORK 5
A. INTRODUCTION 5
B. HUMAN INTERFACE SENSING TECHNOLOGY 6
1. Mechanical Tracker 6
2. Magnetic Trackers 9
3. Previous Sensor Implementations 9
C. SUMMARY 11
III. MODELING THE HUMAN ARM 13
A. INTRODUCTION 13
B. MODELING ARTICULATED BODIES 13
1. Robotics Terminology 14
a. Degrees of Freedom (DOF) 14
b. Joint 14
c. Link 14
d. Position 15
e. Orientation 15
f. World Coordinate System 15
2. Kinematics Terminology 16
a. Index-Free Parameters and Coordinate Systems 16
b. Forward Kinematics (FK) 18
c. Inverse Kinematics (IK) 18
3. Methods Available 20
a. Peabody 20
b. Denavit-Hartenberg Notation (DH) 20
c. Modified Denavit-Hartenberg Notation (MDH) 21
C. HUMAN ARM PROTOTYPE 22
1. Jack Arm Kinematics 22
a. Jack Arm Parameters 23
b. Transformation Matrices 26
2. Jack Arm Forward Kinematics 28
3. Jack Arm Inverse Kinematics 29
4. Summary 30
IV. C++ PROTOTYPING TOOLS 35
A. MOTIVATION 35
B. DESCRIPTION 36
1. Overview 36
2. Design and Implementation 37

a. Motif Interface 37
b. Performer 39
3. Jack Motion Library (JackML) 40
a. Overview 40
b. Overriding Joint Angles 41
4. Forward Kinematics 41
5. Inverse Kinematics 41
6. Sensor Implementation 45
C. RESULTS and CONCLUSIONS 46
V. CONCLUSION 49
A. RESULTS 49
B. LESSONS LEARNED 50
C. EXTENSIBILITY OF SOFTWARE TO OTHER INTERFACES 51
D. FUTURE TRACKING ALTERNATIVES 51
E. FUTURE WORK 52
F. SUMMARY 52
APPENDIX A. LISP CODE 55
APPENDIX B. EDITED LISP EXECUTION SCRIPT 81
LIST OF REFERENCES 87
INITIAL DISTRIBUTION LIST 89

LIST OF FIGURES

1: IPORT Human Sensing Technology [ZYDA95] 8
2: A Minimally Sensed Human [BADL93b] 11
3: Upper Body Joints with Respective DOFs 14
4: Description of Orientation 15
5: The Forward and Inverse Kinematics Problem [FU87] 19
6: Indexed MDH Kinematic Parameters [CRAI89] 21
7: Right Arm Coordinate Systems for Each Link 23
8: MDH Parameters for Right Arm 25
9: Prototype CLOS Human 31
10: CLOS Class and Object Hierarchy 32
11: GUI Prototyping Tool Window 38
12: GUI Prototyping Tool Menus 38
13: Jack Human Model in the Performer Window 39
14: Jack Hand Signal Demonstration 42
15: C++ Class and Object Hierarchy 44
16: Sensor Position on the Arms 45
17: GUI/Non-GUI System Diagram 48

ACKNOWLEDGMENTS

I would like to express my sincere thanks to the people who helped me prepare and implement this thesis. Many thanks and sincere appreciation to Shirley Pratt for unselfishly devoting her time and resources to assist me in my research. Her guidance, patience and insight enabled me to learn a tremendous amount in a very short period of time. Dr Robert McGhee's unsurpassed enthusiasm, encouragement, and tutelage throughout this research, enabled me to fulfill academic goals that may not have otherwise been realized. To John Falby, I owe much gratitude. He ensured that I was organized, on time, and aware of my goals, as they became quite dynamic. He was frequently effective in allaying my nerves when the academic pressures seemed to build. Finally, I would like to thank Scott McMillan for his insight and time spent in enhancing my understanding in this subject area and Dave Pratt for his encouragement in taking on this research project. From all of these people I have learned a spectacular amount.

I. INTRODUCTION

A. MOTIVATION

With the growing requirement for realistic interactive virtual environments (VE), there exists a tremendous need for interactive interfaces for networked computer entities within the world. Due to technological advances in computer power and speed, highly articulated entities have recently been introduced into real-time networked VE [GRAN95]. However, interfaces available to manipulate these entities are not widely used or available beyond the research community. Currently, articulated entity motion is primarily scripted and non-interactive [PRAT95]. We seek to change this for the upper body of human entities because the upper body is an extremely important and dynamic source of interaction between entities. Without a high degree of flexibility with hand and arm movements, a human cannot effectively interact in the world. We see the need for a practical and intuitive, yet comfortable interface for human entities in the networked VE.

B. GOALS

The goal of this thesis is to instrument a user with sensors and allow the user to interact with a virtual world via a virtual human whose motions are driven by natural human motions of the user. We have focused our initial efforts on representing real-time arm motions of humans. This significantly reduces the number of degrees of freedom (DOF) in the problem. We utilize scripted motion to drive the lower body of our human icon. Focusing on interactive arm motion does not result in a significant loss of realism because interactions with the virtual world are largely carried out with the hands.

With this focus, we have four major research goals. First is the desire for real-time, interactive human articulation. The second goal is realistic human movement, a necessity in providing a natural, immersive virtual world in which users will feel comfortable and will participate for extended periods of time. The third objective is to develop the most

efficient interface possible within our technical and financial means. First, we aim to use an inexpensive and preferably the least obtrusive motion sensing technology with which we can produce accurate real-time human motion data. For example, when the interface consists of body-mounted sensors, a minimal number of sensors is desirable. We must also ensure that our methods of computation are accurate and efficient to produce fast and natural human motion. The last research goal is to make the interface intuitive for both the user and the programmer. The physical interface must be comfortable and easy to use with minimal constraints on, and rules for, the user of the VE. Likewise, the interface software is developed in a well-defined library approach so that it is adaptable and extendable to a variety of interfaces.

As can be seen, there are trade-offs in achieving our goals when contending with so many competing objectives. Design decisions are based on the advantages and disadvantages which result from trying to achieve the most of each one of our objectives. The research goals have been accomplished if we have maximized realism and minimized delay in the interface.

C. ORGANIZATION

Chapter II of this thesis reviews previous work in the area of interactive human interfaces for large-scale virtual environments and surveys existing sensing technology. Chapter III provides an overview of kinematics modeling of a human arm possessing multiple degree of freedom joints and of the methods used to calculate link parameters for such articulated figures. The last part of this chapter provides the specific kinematic parameters for a human arm as utilized in a prototype version written in Franz Common Lisp (CLOS) using Allegro Common Windows. Chapter IV describes the C++ prototyping tools, both the graphical user interface (GUI) and a non-GUI version, developed as a part of this research, to more easily manipulate and prototype the computer human's upper body outside of a networked virtual world, both with forward and inverse kinematics. These tools are particularly helpful because they serve as a testbed for interface code which could be

ultimately implemented in a networked virtual world. [ZYDA92] Also covered in this chapter is the partial implementation of magnetic sensors tracking a human upper body. The last chapter, Chapter V, presents some conclusions about the work described. This is followed by recommendations for possible future work with magnetic sensors or other human interfaces which can be used in a large-scale networked VE.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

Recently, the computer power available to the VE researcher has progressed to a point where realistic environments and dynamic entities can be portrayed in real-time. This has been particularly important to the pursuit of one of the grandest challenges in VE research, insertion of humans into large-scale, realistic, interactive virtual worlds. While we are still far from achieving all the goals of virtual human interactions, major strides have been taken. [PRAT95] The virtual reality (VR) community is in continual search for effective three-dimensional position-tracking technology that is affordable and serves as an intuitive interface for the user. To date, available trackers have been used and evaluated by virtual reality researchers, but performances were evaluated with mixed results [MEYE92].

Determination of a "suitable" position tracker can be made through the use of a few important measures of performance. These measures were selected by researchers in the virtual reality community based on results and observations of operational tests [MEYE92]. Evaluation is based on the following five performance measures: resolution and accuracy, responsiveness, robustness, registration, and sociability. Resolution is defined by the smallest change the system can detect. Similarly, accuracy is the range within which a detected position is correct. Responsiveness is determined by the measures of sample rate, data rate, update rate and latency. Latency is the most important of all measures of responsiveness, because it is the measure of delay between the movement of the remotely sensed object and the report of the new position. The third performance criterion is robustness. It is the measure of resistance to noise and other forms of signal interference that may affect the position tracker in its operational environment. Registration is the correspondence between actual position and orientation and reported position and orientation. The final measure is that of sociability. This is the measure of the ability of the

interface to track multiple objects and the range of operation at which it can successfully function. Through these measures, the suitability of a position tracker can be determined. [DURL95]

B. HUMAN INTERFACE SENSING TECHNOLOGY

In recent VE applications, exciting and extremely dynamic worlds have been made possible through the use of position trackers controlling a variety of computer generated entities. Tracking technology can take on a variety of principles of operation. There are four basic principles that are in use today: mechanical, optical, magnetic and acoustic. All of these tracking principles have trade-offs, making not one of these technologies optimal for human tracking in VE. [MEYE92] In order to achieve the research objectives of this thesis, mechanical and magnetic sensors are the most viable technologies currently available. A comparison of these two technologies can be seen in Table 1. Other sensor technologies, like optical and acoustic tracking, are either still evolving or are inappropriate for our needs. Despite the potential benefits of the optical and acoustic technology, the magnetic and mechanical technology are currently the most appropriate for this research.

1. Mechanical Tracker

Mechanical trackers have been in use since Sutherland incorporated them into his first head-mounted display in 1968. Although he was the first to utilize a six DOF goniometer to perform tracking, this technology has been used in a wide variety of applications since. These trackers measure change in position by physically connecting the remote object to a point of reference with jointed linkages. In VR applications, the cumbersome character of mechanical trackers makes them undesirable and impractical. Although there are obvious drawbacks in using mechanical sensing technology for our application, mechanical positioners tend to be accurate, responsive, and robust, (see Table 1). However, this type of sensor has poor sociability due to its inability to track multiple users simultaneously and its limited range of operation. [MEYE92]

	Mechanical	Magnetic
Accuracy and resolution	Good.	Good in small working volumes. Accuracy tends to diminish as emitter-sensor distance increases. Accuracy adversely affected by ferromagnetic objects in working volume.
Responsiveness	Good.	Relatively low data rates. Filtering required for distortions in emitter field can introduce lag.
Robustness	Good. Not sensitive to errors introduced from the environment.	Ferromagnetic objects create eddy currents that distort the emitted field causing ranging errors.
Registration	No reports.	No reports.
Sociability	Limited range, two systems cannot effectively occupy the same working volume.	Most effective for small working volumes. Some implementations improve working volume by augmenting emitted field strength. However, distortions from induced eddy currents increase with field strength. Configurations available for allowing sensors to share emitters in same work space. Magnetic systems are unaffected by non-ferromagnetic occlusion.
Comments	Cumbersome. Well suited to force feedback. Successful applications in Telerobotics.	Available off-the-shelf. Relatively inexpensive. Most commonly used in current VR research. Successfully used in simulated cockpits.

Table 1. Mechanical and Magnetic Tracking Technologies Compared [MEYE92]

The Individual Portal (IPORT) system developed by SARCOS Inc., Army Research Laboratory - Human Research and Engineering Directorate (ARL-HRED), University of Pennsylvania, and the Naval Postgraduate School (NPS) in 1993, is currently one of the few force feedback motion devices able to insert a human into a large scale networked VE (see Figure 1) [PRAT94, PRAT95]. Interactivity across the network is primarily achieved by means of Distributed Interactive Simulation (DIS) Protocols [GRAN95]. IPORT has three major hardware components: IPORT base mechanical display (from two to 18 degrees of freedom (DOF) with force feedback for lower body), a Kaiser VIM head-mounted display, a VME Motorola 680X0-based Real-time Controller with VX Works real-time operating system, and a hand held designator (6 DOF pointer into virtual space).

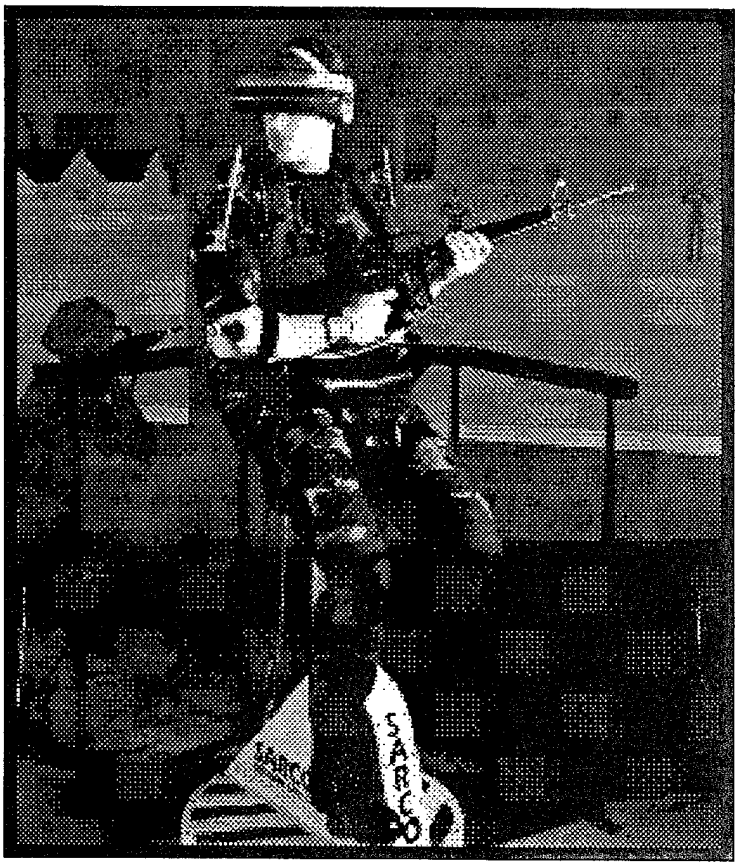


Figure 1: IPORT Human Sensing Technology [ZYDA95]

An optional component to the IPORT is the SARCOS Sensuit (TM). The suit utilizes mechanical trackers to obtain the upper body movement measurements. When this suit is used, the user's limb and joint motions are captured and fed into the simulation system to drive the limbs and joints of the virtual human. Although it requires some involved calibration and is relatively cumbersome for the user, this suit provides fairly accurate position and orientation tracking in real-time.

2. Magnetic Trackers

Magnetic trackers are the most widely used tracker in the VR community today. This is attributed to low cost, modest but acceptable accuracy, and easy implementation. Sociability for this type of sensor is high in a small working volume free from ferromagnetic objects. With these factors considered, this type of sensor can be an excellent candidate for body tracking in the VE. However, the presence of excessive iron and increasing emitter-sensor distance can degrade accuracy ratings. There are two companies that widely market magnetic sensors: Polhemus and Ascension. Polhemus sensors work using alternating current (ac) while Ascension *Flock of Birds* utilizes direct current (dc) [ASCE95]. Testing and implementation of magnetic sensors is the only way to ensure that this technology is appropriate for a particular application. [MEYE92]

3. Previous Sensor Implementations

During Individual Combatant Modeling and Simulation Symposium 1994 (INCOMSS 94) at Ft. Benning, GA in February 1994, the first demonstration of the IPORT was presented. *Jack*, a program developed by University of Pennsylvania's Center for Human Modeling and Simulation to manipulate articulated figures, was used to provide body joint angles [BADL93a]. While this system proved successful, there was significant system overhead involved communicating with *Jack*. Joint information was transferred to a separate machine via a specified port, strongly impacting performance. Also at the same demonstration, the SARCOS sensor suit was used. It allowed the user to control the upper body of the icon in real-time by overriding the Jack joint angles. However, the suit was felt

by some to be cumbersome and difficult to adjust (the suit had to be recalibrated for each individual), and measurements were often accompanied by high noise levels, resulting in jerky motion. [PRAT94]

During May and October 1994, the I-PORT was shown again. The May 1994 demonstration utilized *Jack* as before. However, the October 1994 demonstration used Jack Motion Library (JackML), a library of joint angles linked to the main application, which was able to be queried in real-time on a single machine. Due to the number of people at the demonstration, it was impractical to fit them with sensor suits. To solve the problem of specifying upper body joint angles, the icon kept his hands locked on the rifle and inverse kinematics was used to compute joint angles for the arms. While this was visually acceptable when the soldier was engaged in combat, he was never able to put down his rifle or to hold it with one hand, which for this environment, is impractical. [PRAT94]

Efforts at the University of Pennsylvania have been made to realistically recreate a human posture and position with minimal sensors for a less encumbered operator. They accomplished this by using four six DOF magnetic sensors: one on each palm, on the waist, and on the head (shown in Figure 2). Clearly, the objectives are different in that their paradigm requires more intensive mathematical calculations than those desired to meet the demands of a real-time networked environment. [BADL93b]

The above considerations led to the pursuit of a solution similar to the Reality Built for Two (RB2) system [BLAN90]. In it, VPL Research, Inc. used fiber optic cables to measure joint angles. The result was a fully specified set of angles. Fully specified means that complex mathematical joint angle estimation algorithms are avoided by sensing all the needed position and orientation information. Rather than using a suit containing fiber optics, we propose to use a set of Polhemus sensors which can be configured to be less cumbersome and more durable than the body suit described earlier.

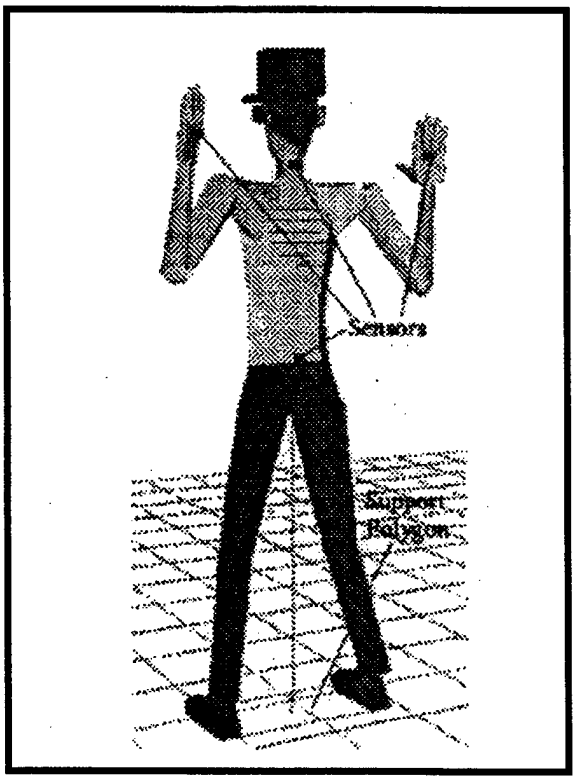


Figure 2: A Minimally Sensed Human [BADL93b]

C. SUMMARY

As seen in this discussion of position tracking technologies, five performance factors should be considered in evaluating each type of tracker:

- resolution and accuracy
- responsiveness
- robustness
- registration
- sociability.

Depending on the application and requirements/conditions of a particular environment, sensors can be largely chosen to meet system specifications. Currently, as seen with the IPORT, a human can be tracked in a networked virtual environment. However, such systems have their limitations with respect to performance factors.

Research is on-going in developing sensors that will be comfortable and effective in real-time, large-scale networked VEs.

III. MODELING THE HUMAN ARM

A. INTRODUCTION

Because the goal of this research is to manipulate a simulated human's upper body in a large-scale virtual environment, it is crucial to know how the human model in the target application is constructed: what joints are available and how many degrees of freedom each has. For this research, the human model used in our target virtual world is called *Jack* and was created by personnel from the Center of Human Modeling and Simulation at the University of Pennsylvania [BADL93a]. Although, this model has a total of 39 DOF in 17 separate joints, we are only concerned with the model's upper body structure, and in particular, the arms [PRAT95]. The arms of the model consist of a total of eight joints, but the two clavicles will not be articulated in this application. Articulating the remaining six joints of the two arms of the upper body still results in realistic arm movements. Each joint has one or more rotational DOFs, as noted by the parenthesized numbers in Figure 3. With this information about the human model, a design can be developed and prototyping can begin. There is, however, some ground work which must be presented to aid in the understanding of our prototype design.

B. MODELING ARTICULATED BODIES

In the field of robotics, objects are described based on how they are represented and how they move. Here important terminology is explained that is helpful in understanding the composition of articulated bodies - entities that have moveable parts, like the arm of a human.

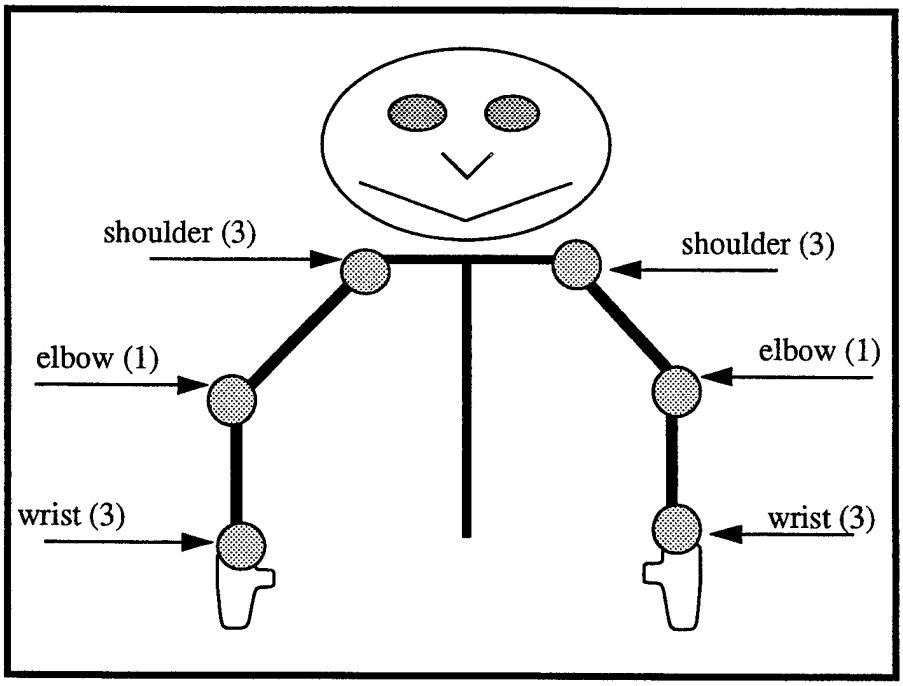


Figure 3: Upper Body Joints with Respective DOFs

1. Robotics Terminology

a. Degrees of Freedom (DOF)

The *degrees of freedom* are the number of independent position variables necessary to specify the location of all parts of an articulated entity. [WATT92]

b. Joint

A *joint* can be one of two types: revolute or prismatic. Revolute joints exhibit rotary motion about a joint axis. A prismatic or sliding joint exhibits linear motion along an axis. In this research, all joints in the arm model are rotary joints. Joints are the points of dynamic motion of an articulated body. [SCHI90]

c. Link

A *link* is a rigid body which connects joints, a critical component in modeling articulated rigid bodies. The arm is modeled as a series of links joined by revolute joints.

Every adjacent pair of links is joined by a revolute joint and there exists a link between successive joints [SCHI90]. Links may have zero length when modeling humans with multiple DOF joints [CRAI89].

d. Position

Position is given in x, y and z dimensions for any point in 3D space describing its location with a 3 x 1 position vector. X, Y and Z location is measured relative to a predetermined reference coordinate space. These quantities are linear measurements along the three respective axes. [CRAI89]

e. Orientation

Orientation is given as the angular rotation about the respective axes, known as azimuth (yaw), elevation (pitch) and roll, describing any point in 3D space with the symbols Ψ, Θ, and Φ, as seen in Figure 4. Movement about axes is described with a 3x3 rotation matrix (R) for a point in space. In this case, every link has a coordinate system from which the link references its position. [FU87]

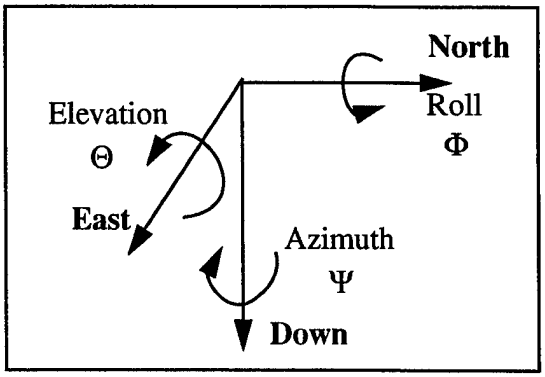


Figure 4: Description of Orientation

f. World Coordinate System

The *world coordinate system* is an established frame of reference or base frame from which position and orientation can be measured. There are many frames of reference that can be chosen for a body. [CRAI89]

2. Kinematics Terminology

a. Index-Free Parameters and Coordinate Systems

In robotics literature there is some inconsistency and confusion as to the terminology, definitions, and labeling that exist for kinematic parameters and coordinate systems. These notions can be described in general terms without discussing one particular indexing convention, for instance Denavit-Hartenberg (DH) or Modified Denavit-Hartenberg (MDH) notation, to be discussed later. In order to develop a better understanding of kinematic terminology, definitions will be given without regard to indices of links or joints.

(1) Definitions

An *axis* is a line in space. An *oriented axis* has a designated positive sense. A *coordinate axis* is an oriented axis with a specified origin. A *link* is a rigid body with two holes drilled in it (they may intersect, but they are distinct). The lines through the center of these holes are called *motion axes*. Each link must have a *name*. This may be (but need not be) simply a numerical index starting at the base and proceeding outward. The first moveable link could be called Link1. The last link in the arm chain could be LinkN. Likewise, every joint has a scientific name and a common (anatomical) name to denote the same thing, but possibly with sign reversal and sometimes with a different definition of zero angle.

Generally, each link has an *inboard motion axis* and an *outboard motion axis*. The inboard axis is the one closest to the base when the links are assembled in an open serial chain. When in a chain, a link can be *rotated* and/or *translated* about its inboard motion axis. The unique shortest distance between the motion axes of a given link lies along the *common normal axis*. If motion axes intersect, the common normal axis passes through the intersection and is perpendicular to the plane defined by motion axes. On the other hand, if the motion axes are parallel, common normal axis is not unique. It must therefore be defined as a *specific* common normal axis.

There are peculiarities with the base link (first link) and the last link in a chain of links. They each have only one hole drilled in them and must be treated differently than any other link. The base link must have a *base coordinate system* assigned to it. On the other hand, in certain notation conventions, the last link could have a second motion axis, or an *approach axis* artificially introduced. [SCHI90]

(2) Parameters

The above definitions allow naming and identification of all links, joints and axes in a serial chain linkage mechanism. Additionally, each link of an articulated body has four parameters associated with it. However, the signs of each of these parameters associated with a link are ambiguous. To resolve this, a positive sense must be associated with each link common normal axis and with each joint axis. There are no generally agreed upon rules for assigning this sense. However, once a sense has been assigned to each joint axis and each common normal axis, the parameters can be uniquely defined.

The four parameters of a link can be categorized into two groups: *joint parameters* and *link parameters*. First, every joint in a serial chain connects its inboard link to its outboard link through a common motion axis which joins the links. The two joint parameters are *joint displacement*, **d**, and the *joint angle*, Θ. The angle Θ is the (signed) angle measured from the inboard common normal axis to the outboard common normal axis about the joint axis. Likewise, **d** is the (signed) distance measured along the joint axis, from the inboard common normal axis to the outboard common normal axis.

The two link parameters are *link length*, **a**, and the *twist angle*, α, and can be determined by the relative position and orientation of the inboard and outboard axes of the link. Each link has a value for **a**, which is the (signed) distance along its oriented common normal axis from the inboard motion axis to the outboard motion axis. Each link also has a twist angle, α, which is the (signed) angle from its oriented inboard motion axis to its outboard motion axis, measured in a right-handed screw sense about the link common normal axis. The right-hand rule (RHR) is a common method of figuring proper orientation

of coordinate system axes. In employing this rule, the thumb of the right hand is oriented along the positive sense of a given rotation axis. Curling the fingers then indicates the positive sense of rotation.

Notice that the above parameter definitions uniquely specify all parameters for all links in an open chain mechanism after a positive sense has been assigned to all link axes. Note also that, at this level of abstraction, it is not possible to perform any computations regarding the position in space of any point attached to any link. This is because no coordinate system has been defined for any link. While this will be discussed later, it is important to recognize that the link parameter table can be completely filled out for all links, except the base link and the last link discussed earlier, without defining any link coordinate systems. Recall that link coordinate systems require that an origin be specified.

b. Forward Kinematics (FK)

Forward Kinematics is very basic to the study of mechanical manipulation [CRAI89]. It is the formulation of the relationship between the joint variables and the position and the orientation of the tool or "end effector" [SCHI90]. This study of motion ignores all underlying forces that cause it. In FK, the movement of each joint is specified explicitly by the animator, as indicated in Figure 5. The motion of the end effector is determined indirectly by the accumulation of all transformations down the entire series of links that leads to that end effector. [WATT92]

c. Inverse Kinematics (IK)

A more difficult problem than FK is IK. Its complexity lies in the fact that there is no system comparable to the existing systematic robotics algorithms for FK (to be discussed in the next section) and that each solution for solving this type of motion is unique. In particular, human motion applications that accommodate dynamic motion simulation are beginning to rely on using feedback from external sensors. Sensors can supply information about link location and orientation. From this information, it is

necessary to determine appropriate values for the joint variables to properly articulate the body part. So, the IK problem presents itself as follows: Given a desired position P and orientation R for the end effector of a body part, find the values for the joint variables q which satisfy the transformation matrix (see Figure 5) [SCHI90]. Inverse kinematics is also known as "goal-directed motion" [WATT92]. In otherwords, IK is when the animator defines the position of the end effector, like the hand of an arm, only. IK determines the position and the orientation of all joints in the link hierarchy that lead to the hand. Because IK becomes computationally more expensive as the complexity of the articulation increases, and is very difficult to use when specifying a particular animation, IK has only been useful in a limited subset of computer animation applications. Human motion simulation is one area that is appropriate for IK because leg and arm motion drives the animation part of the hierarchy. Trying to animate humans by FK is counterintuitive and tedious, thus ensuring IK a place in future computer animation. [WATT92]

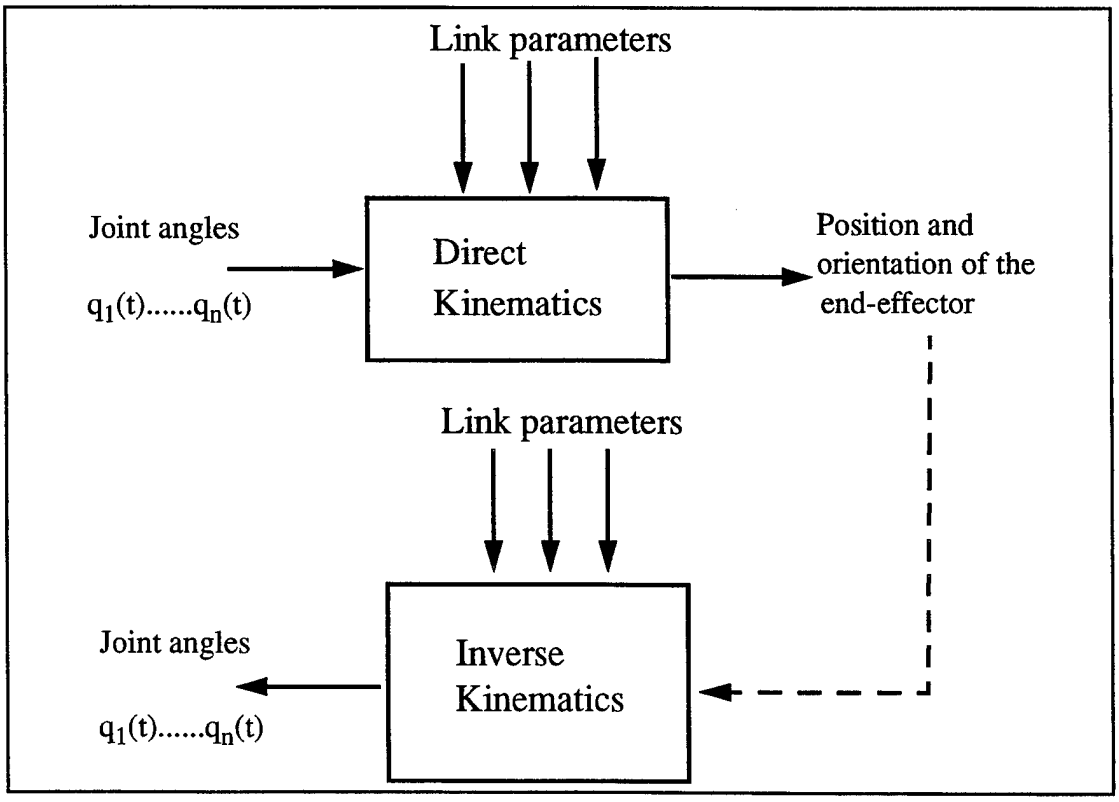


Figure 5: The Forward and Inverse Kinematics Problem [FU87]

3. Methods Available

a. Peabody

The Peabody model is a representation of articulated figures which was developed at the Center of Human Modeling and Simulation, University of Pennsylvania [BADL93a]. It is the notation used in the *Jack* software discussed earlier. Peabody is composed of segments, rather than links, which are connected by multiple DOF joints. It is a data structure which maintains geometric information about segment dimensions and joint angles, and efficiently computes, stores and accesses various kinds of geometric information. Specifically, Peabody maps segment dimensions and joint angles into global coordinates for the end effectors. Peabody was developed to achieve four objectives:

- to be general purpose
- to be a well-developed notation of articulation
- to represent tree-structured objects in a hierarchy
- to be simple to use.

For these reasons, Peabody is used in the *Jack* software instead of standard existing robotics notations which do not possess inherent tree-structured representations, and are not as flexible [BADL93a]. However, Peabody is specific to *Jack* software, not a universal notation like the next two notations to be discussed.

b. Denavit-Hartenberg Notation (DH)

DH notation is the most common kinematics representation in robotics for systematically assigning right-handed orthonormal coordinate systems or frames, one to each link in an open kinematic chain of links [SCHI90]. This notation derives a set of the four kinematic parameters introduced earlier, which describes a link based on measurements between the coordinate frames (axes) of an articulated body [BADL93a]. In this system of notation, the base joint in the articulated system is named Joint0 and the link attached to it is named Link1, thus producing a numbering system in which the link and the link's outboard joint both have the same index values. The most important principle in DH notation is that the link origin for the coordinate system is on the *outboard* motion axis.

Additionally, a transformation matrix (or A-matrix) exists specifically for the DH notation. For a more thorough explanation of this system, see [DAVI93].

c. Modified Denavit-Hartenberg Notation (MDH)

MDH notation utilizes the same system as DH notation, except for one significant difference. The link origin in MDH is on the *inboard* motion axis (seen in Figure 6). In this method, the base joint and the first link, or the base joint's outboard link,

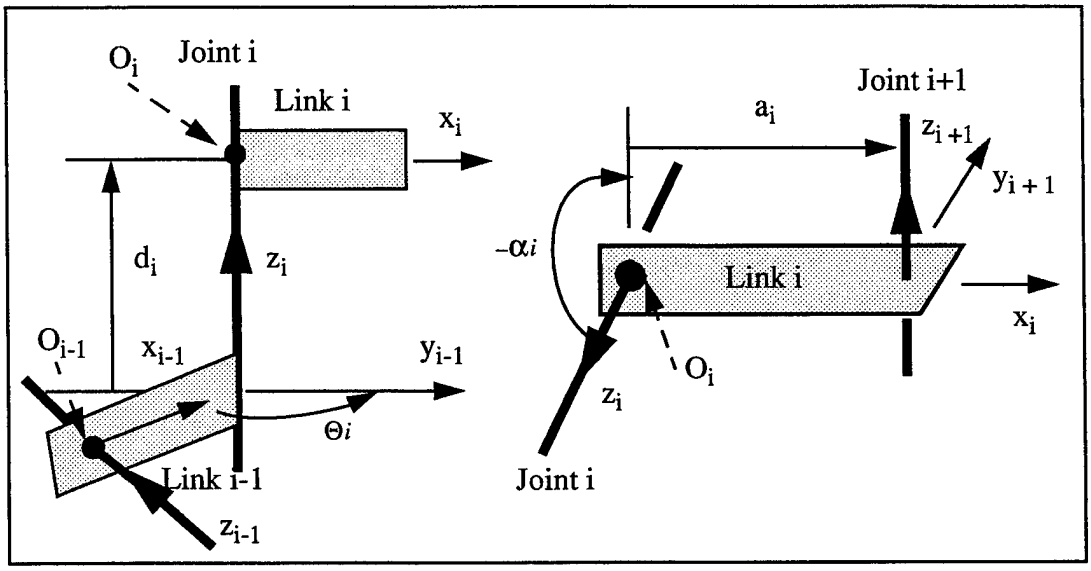


Figure 6: Indexed MDH Kinematic Parameters [CRAI89]

have the same index value, JointN and LinkN. This is the notation that has been utilized in this research and that is covered by John Craig and Sandra Davidson in their respective works [CRAI89][DAVI93]. Below (Equation 3.1) is the transformational matrix (or T-matrix) for the MDH method for assigning right-handed orthonormal coordinate systems for each link in the articulated structure. The appearance of the index i-1 in the matrix reflects the fact that motion for any joint, from the inboard link coordinate system to the outboard link coordinate system involves first a translation of a_{i-1} followed by a rotation of α_{i-1}, both of which are fixed numbers. Θ_i and/or d_i are the parameters which describe the motion of link i.

$$ {}^{i-1}T_i = \begin{bmatrix} \cos\Theta_i & -\sin\Theta_i & 0 & a_{i-1} \\ \sin\Theta_i \cos(\alpha_{i-1}) & \cos\Theta_i \cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1}) d_i \\ \sin\Theta_i \sin(\alpha_{i-1}) & \cos\Theta_i \sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1}) d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq 3.1)} $$

C. HUMAN ARM PROTOTYPE

1. Jack Arm Kinematics

The prototype of the human and its associated arm kinematics were coded in Common Lisp Object System (CLOS). For some problems CLOS is considered to be an exploratory language--one which is effective for prototyping solutions for complex software design problems for ultimate implementation in another language. There are three particularly beneficial features of CLOS for prototyping:

- It is a weakly typed language, meaning that decisions about how variables will be used can be delayed until code is actually used.
- Memory management is automatically handled.
- It is normally interpreted. Can be compiled after debugging.

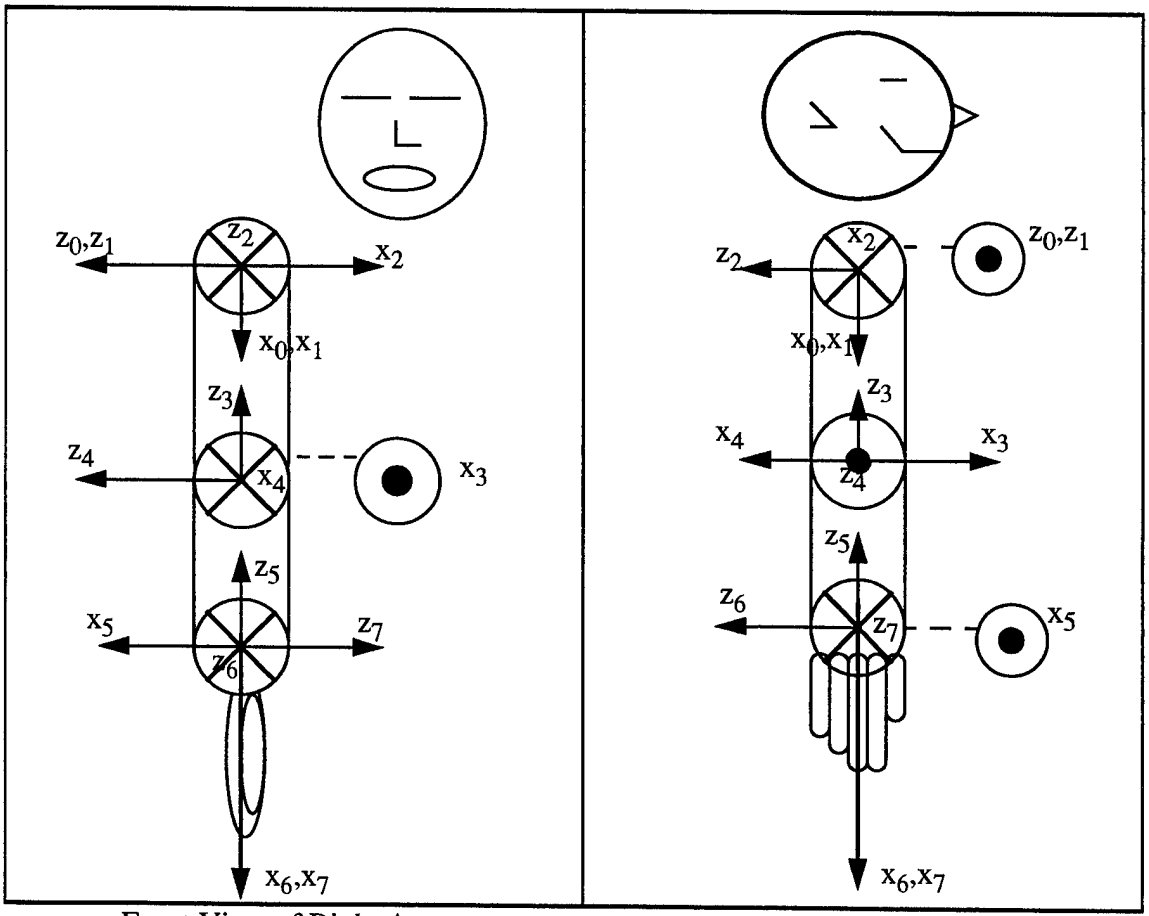
On the other hand, CLOS is not a good language for computationally demanding real-time applications. So, in light of our research objectives, once an adequate prototype is developed in CLOS, it must be translated into a language that exhibits greater run-time efficiency. C++ is such a language. The arm was constructed strictly adhering to object oriented programming style which both CLOS and C++ support. Code translation from the prototype in CLOS to the target application, NPSNET in C++, is straightforward based on object oriented programming.

The prototyped human in this research has only one arm, the right arm; however, the two human arms are identical articulated bodies except for their angle off of the body's forward axis. In order to simplify the arm parameters of the first link, an imaginary link was

constructed from the human's right clavicle joint to the point where the arm joins the body. This makes the body's stationary clavicle the base joint.

a. Jack Arm Parameters

The kinematic parameters for Jack's right arm can be determined if the coordinate systems are known. Figure 7 shows the coordinate systems for all seven DOF plus the imaginary arm link for the right arm of Jack, the way it will be modeled in graphics. It should be noted that Figure 3 reflects the actual/physical makeup of the human arm, indicating three DOF in the shoulder, one in the elbow, and three in the wrist. However, when modeling the arm in robotics and in graphics, a design decision was made to reflect the motion of shoulder roll at the elbow. This is because the visual, actual manifestation of



Front View of Right Arm Side View of Right Arm

Figure 7: Right Arm Coordinate Systems for Each Link

that motion is not evident in the upper arm, but is in the lower arm and hand. The artificiality that exists in the robotics and graphics environment is the lack of muscular tissue on the skeletal frame which causes computer modeling to deviate from reality. As seen in Figure 7, the computer model reflects the shoulder with two DOF, plus the imaginary link, and the elbow with two DOF. The wrist remains unaffected. In this model, the last link does possess an approach axis, x_7. Normally, the approach axis is a z-axis that is aligned with the end effector's roll axis and points away from the wrist [SCHI90]. Here, the last link is not the roll axis, so the approach axis does not conform to the normal approach axis convention.

To establish the coordinate systems in Figure 7, these basic rules were used:

1. Establish the motion axes (z-axis) for each DOF desired in the arm at each joint location. This axis is the oriented line about which a segment moves to produce the desired motion, i.e. azimuth, elevation or roll.
2. Once all the z-axes are established, perform a cross-product of $z_{inboard}$ to $z_{outboard}$ for each z-axis in the entire arm chain. This can physically be performed by the RHR, described earlier. The thumb produces the $x_{inboard}$ axis for the inboard coordinate frame.
3. Although not pictured in Figure 7, it is straightforward to determine the y-axis for each of the coordinate frames by performing the cross-product of z-axis to x-axis of the same coordinate system by the RHR. The y-axis is generally needed to draw a representation of a given limb segment. The general rules, given two of the three axes of a coordinate frame, are to perform the RHR to determine the third axis follow:

- from oriented z-axis to oriented x-axis to get the oriented y-axis
- from oriented x-axis to oriented y-axis to get the oriented z-axis
- from oriented y-axis to oriented z-axis to get the oriented x-axis.

The notation used to denote an oriented axis pointing straight out of the page is an encircled black dot, signifying the point of an arrow. The encircled X, denotes an oriented axis pointing straight into the page, signifying the tail of an arrow. Using the

index-free definitions of the four kinematic parameters explained earlier in section B.2.a of this chapter, and the following MDH index conventions for the parameter values for Jack's right arm, the parameter values can be determined (as shown in Figure 8).

index #	link name	inboard link length (a_{i-1})	inboard twist angle (α_{i-1})	inboard joint offset (d_i)	inboard joint angle (Θ_i)	inboard axis name
0	clavicle	0	0	0	0	body
1	outer shoulder gimbal ring	0	0	0	0	azimuth
2	middle shoulder gimbal ring	0	90	0	-90	elevation
3	upper arm (inner shoulder gimbal ring)	0	90	-2.0cm	-90	roll
4	inner forearm	0	90	0	180	elbow
5	outer forearm	0	90	-2.0 cm	90	roll
6	middle wrist gimbal ring	0	90	0	-90	azimuth
7	hand (inner wrist gimbal ring)	0	90	0	0	elevation

Figure 8: MDH Parameters for Right Arm
(Inboard joint angles are for the position shown in Figure 7)

- **Inboard Link Length (a_{i-1})** is the length or distance between Z_{i-1} and Z_i measured along X_{i-1}, otherwise referred to as the length of the common normal.
- **Inboard Twist Angle(α_{i-1})** is the angle from Z_{i-1} to Z_i measured about X_{i-1}, otherwise referred to as the angle about the common normal.
- **Joint Displacement (d_i)** is the length from X_{i-1} to X_i along Z_i, otherwise referred to as the distance between common normals.
- **Joint Angle(Θ_i)** is the angle from X_{i-1} to X_i about Z_i, or the angle of rotation.

Here, again, it is evident that this particular robotics representation of an arm does not follow natural human intuition in representing an arm. It is peculiar to have a parameter called "link length", in Figure 8, equal to zero. It seems natural to think of the upper arm, forearm, and hand as links of an arm, and these should have lengths that are non-zero. However, this is not how this arm is represented. The joint offset is the parameter which allows for the physical arm representation. This point can cause considerable confusion for beginners in this research field.

b. Transformation Matrices

The transformation matrices, or T-matrices, for Jack are constructed using the link parameters listed in Figure 8 and the MDH transformation matrix template in Equation 3.1. The computed link T-matrices are included in this discussion as Equations 3.2-3.8. In order to move the human arm which is made up of rotary links, the value for Θ, or joint angle, is the parameter which changes to move a particular DOF. Furthermore, the other three parameters, a, α, and d are constants. Each row of parameters is assigned to one specific index, identified in the first column of the row, and substituted into Equation 3.1 to get the T-matrices in Equations 3.2-3.8. Note that because many of the complex matrix terms are eliminated before beginning the calculations, the outcome is more manageable T-matrices.

$$ {}^{0}T_{1} = \begin{bmatrix} \cos\Theta 1 & -\sin\Theta 1 & 0 & 0 \\ \sin\Theta 1 & \cos\Theta 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq 3.2)} $$

$$ {}^{1}T_{2} = \begin{bmatrix} \cos\Theta 2 & -\sin\Theta 2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\Theta 2 & \cos\Theta 2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq 3.3)} $$

$$ {}^{2}T_{3} = \begin{bmatrix} \cos\Theta 3 & -\sin\Theta 3 & 0 & 0 \\ 0 & 0 & -1 & 2.0 \\ \sin\Theta 3 & \cos\Theta 3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq 3.4)} $$

$$ {}^{3}T_{4} = \begin{bmatrix} \cos\Theta 4 & -\sin\Theta 4 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\Theta 4 & \cos\Theta 4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq 3.5)} $$

$$ {}^{4}T_{5} = \begin{bmatrix} \cos\Theta 5 & -\sin\Theta 5 & 0 & 0 \\ 0 & 0 & -1 & 2.0 \\ \sin\Theta 5 & \cos\Theta 5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq 3.6)} $$

$$ {}^{5}T_{6} = \begin{bmatrix} \cos\Theta 6 & -\sin\Theta 6 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\Theta 6 & \cos\Theta 6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq 3.7)} $$

$$ {}^{6}T_{7} = \begin{bmatrix} \cos\Theta 7 & -\sin\Theta 7 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\Theta 7 & \cos\Theta 7 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq 3.8)} $$

2. Jack Arm Forward Kinematics

Forward kinematics manipulation of the links of the upper arm will result in the end effector being positioned by applying desired joint angles to the links in the chain. Equations 3.9-3.11 illustrate the mathematical formulas involved in locating the end effector. The result of the theoretic equation manipulation is the homogeneous transformation matrix (H-matrix) of the ending link, which in the case of the upper arm, is Link2, seen in Equation 3.12. An H-matrix is a 4 x 4 matrix which is used to capture the rotation and translation (orientation and position) of the general transform into a single matrix [CRAI89]. Orientation, alone, can be represented in a rotation matrix or (R-matrix). A real-world example of an H-matrix is found in Equation 3.13 after moves of joint angles $\Theta_1 = 0.1$ radians and $\Theta_2 = 0.5$ radians were executed for Link1 and Link2 respectively.

$$H_1 = H_{Body}({}^0T_1) \qquad \text{(Eq 3.9)}$$

$$H_2 = (H_1)({}^1T_2) \qquad \text{(Eq 3.10)}$$

$$H_2 = (H_{Body})({}^0T_1)({}^1T_2) \qquad \text{(Eq 3.11)}$$

H_{Body} is a 4 x 4 identity matrix and $({}^0T_1)$ and$({}^1T_2)$ are defined in Equations 3.2 and 3.3. The product of the two T-matrices ${}^0T_1$ and ${}^1T_2$ will be referred to henceforth as $T_{upperarm}$:

$$H_2= \begin{bmatrix} \cos\Theta_1\cos\Theta_2 & -\sin\Theta_2\cos\Theta_1 & \sin\Theta_1 & 0 \\ \sin\Theta_1\cos\Theta_2 & -\sin\Theta_2\sin\Theta_1 & -\cos\Theta_1 & 0 \\ \sin\Theta_2 & \cos\Theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eq 3.12)}$$

$$H_2= \begin{bmatrix} 0.477 & 0.873 & 0.099 & 0 \\ 0.048 & 0.088 & -0.995 & 0 \\ -0.878 & 0.479 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eq 3.13)}$$

3. Jack Arm Inverse Kinematics

Inverse kinematics refers to the process of starting with a known position and orientation of the end effector and computing backwards the joint angles for the links which affect the end effector's position and orientation. Imagine if the sensor on the upper arm, measuring the shoulder orientation, produced the H-matrix in Equation 3.13. In the CLOS simulation, a sensor class, called Polhemus class, was written in order to model the IK as it would happen with real sensors attached to the arm. The code takes the last H-matrix generated by the forward kinematics manipulation, in this case H_2, and provides it to the appropriate sensor of the Polhemus class, as if it were directly sensed information. IK requires software to work backwards, doing the multiplication on the right side of Equation 3.14, to derive the $T_{upperarm}$ matrix. In this case, $T_{upperarm}$ matrix is the same matrix as H_2 because the inverse of H_{Body} is an identity matrix. From $T_{upperarm}$ the joint angles, Θ_1 and Θ_2 can be solved through a series of non-linear simultaneous equations, seen in Equations 3.16 -3.17, referring to the matrix values in the reference positions noted in Equation 3.15.

$$T_{upperarm} = H_{Body}^{-1} H_2 \qquad \text{(Eq 3.14)}$$

The positions in the T- matrix will be referred to with the following designators:

$$\begin{bmatrix} a1 & b1 & c1 & d1 \\ a2 & b2 & c2 & d2 \\ a3 & b3 & c3 & d3 \\ a4 & b4 & c4 & d4 \end{bmatrix} \qquad \text{(Eq 3.15)}$$

Comparing Equation 3.15 with Equation 3.12 and through the use of a four quadrant *atan* function, evidently

$$\Theta_1 = (\text{atan } c1\ (-\ c2)) \qquad \text{(Eq 3.16)}$$

$$\Theta_2 = (\text{atan } a3\ b3) \qquad \text{(Eq 3.17)}$$

Generally speaking, non-linear simultaneous equations return the actual joint angles for each consecutive link in the body being articulated. In this example, the upper arm is moving and it is modeled with two segments. Assuming perfectly aligned coordinate systems between the sensor and the link, the exercise above simulates what would happen in code when the sensor returned a matrix like the one found in Equation 3.13. The joint angles solved from the non-linear simultaneous equations will correctly articulate the intermediate joints for the end effector to be in its goal state. The joint angles that result from this inverse kinematic algorithm will be relative to the original joint angles specified for that link in Figure 8. For instance, Equation 3.18 illustrates that the initial joint angle (in degrees) for Link 2 is -90, convert degrees to radians, and add the desired movement for that link in radians and the result is the new desired joint angle.

$$\Theta_2 = -90 = -1.5707 \text{ radians} + 0.5 \text{ radians} = -1.0707 \text{ radians} = \text{new } \Theta_2 \qquad \text{(Eq 3.18)}$$

Therefore, all computed joint angles are measured from the originally specified joint angle parameters. The rest of the arm's IK can be derived similarly to the example just shown.

4. Summary

The human arm prototype in CLOS provides an excellent example of code reuse and object oriented programming. Many of the classes and objects used in this simulation are similar to those used in the Aquarobot example developed by Sandra Davidson in her thesis [DAVI93]. The major difference between the prototype of the aquarobot and the human arm is that the aquarobot's legs are constructed with single DOF joints, whereas the arm has joints which have multiple DOFs. This is a challenging problem, particularly from the robotics perspective, because the task when modeling human limbs is to accurately model the mobility allowed by human ball-and-socket joints in graphical representations. The notion of DOFs being represented by links and some of these links having zero length can

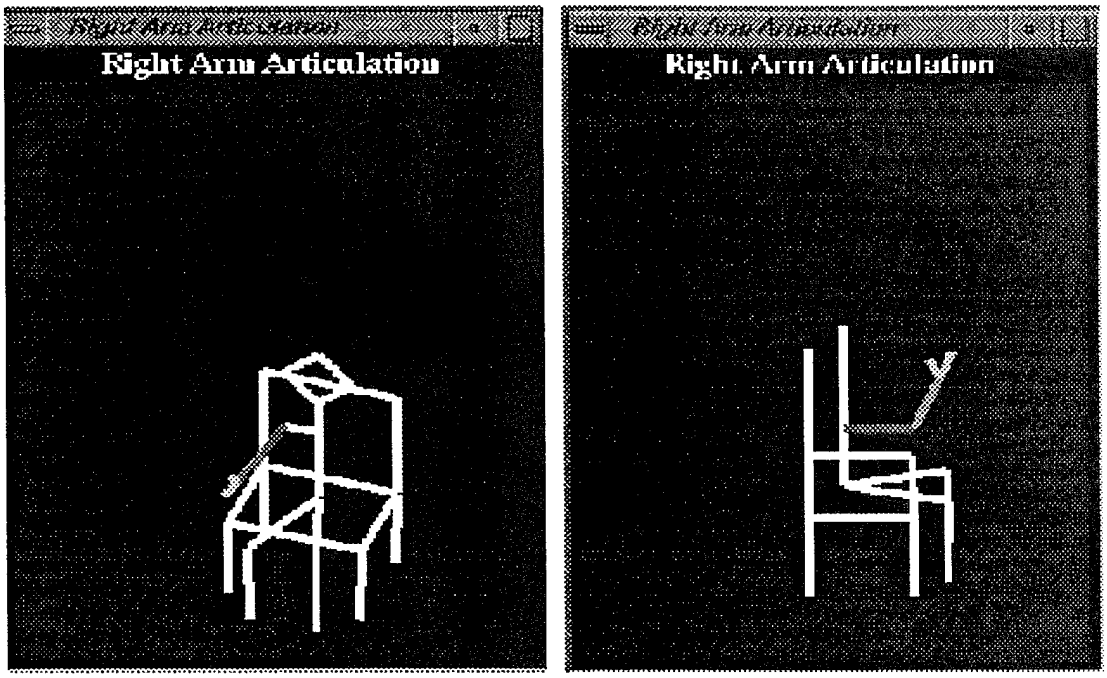


Figure 9: Prototype CLOS Human

be hard to comprehend. However, after grappling with this representation technique, design decisions were made on how the arm would be modeled. Through the reuse/adaptation of some of the classes in Davidson's thesis and standard robotics articulation methods, a human arm can be accurately represented, as seen in Figure 9. The CLOS representation of the human on the screen is done through the use of node lists and polygon lists. Node lists are lists of the node coordinates (X, Y, Z, 1) which are connected by the polygon lists to produce the desired graphical representation. Recall the arm coordinate systems (Figure 7) are structured such that the y-axis or z-axis is the axis which is rendered on the screen through node lists. This result is based on coordinate system design decisions. There is more than one way which these can be chosen. Figure 10 represents the class and object hierarchies of the structure of the code which was written for the human arm motion prototype and is contained in Appendix A. The hierarchy of the arm was modified from that of the Aquarobot leg, adding an additional layer of abstraction in order to more realistically add simulated sensor functionality with polhemus class, in the prototype.

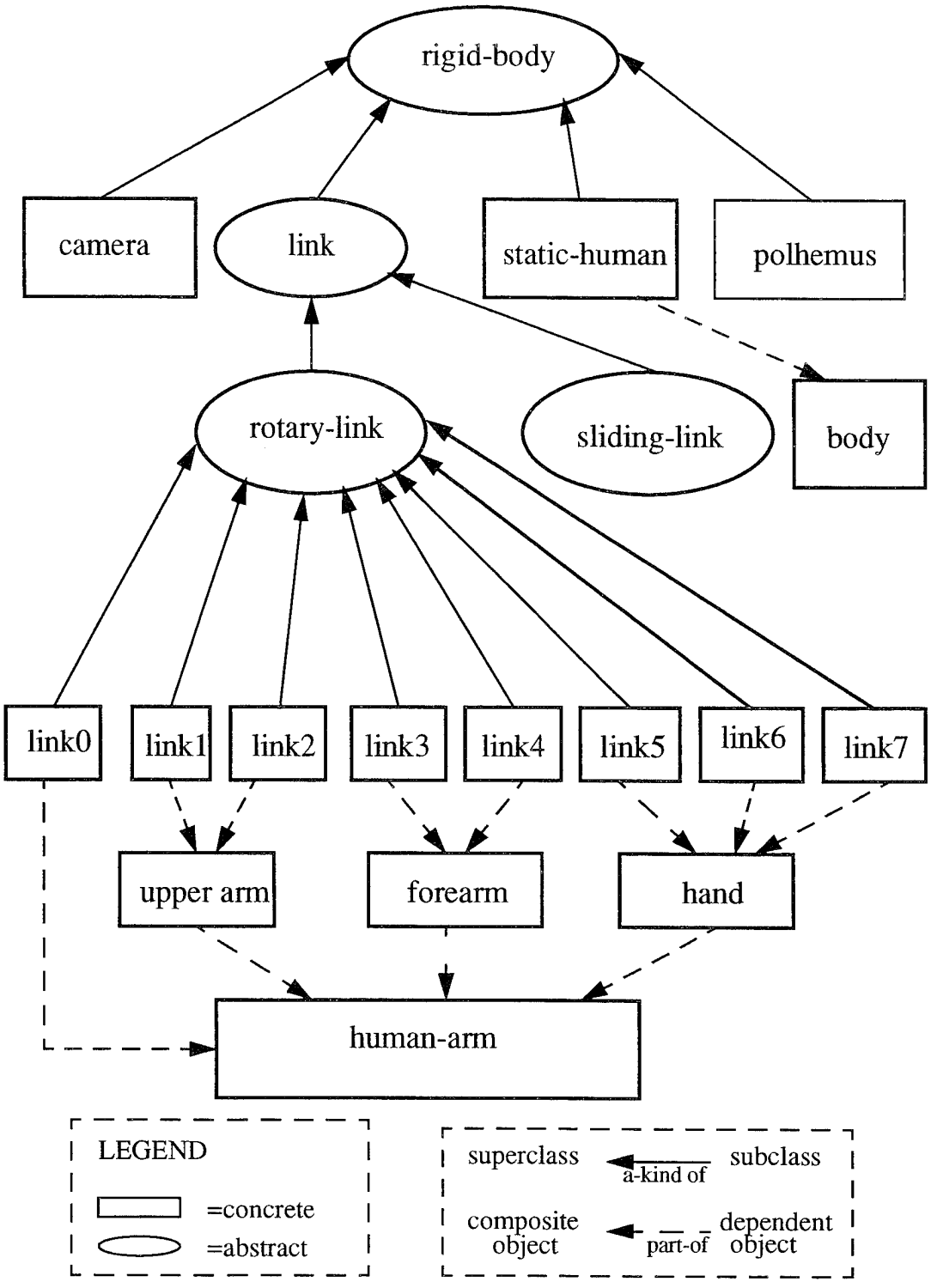


Figure 10: CLOS Class and Object Hierarchy

Now that classes have been developed and testing has been conducted by articulating the human right arm (Appendix B), it is appropriate, at this stage, to develop this functionality in a language that not only accommodates object oriented programming principles, like CLOS, but that can perform articulations of a human arm more rapidly. Because of the anticipated use of this code within a particular networked virtual environment which is written in C++, it is essential that it be translated into C++. The goal here is to adhere as closely to the already constructed CLOS classes and object hierarchy as possible and still be eventually implemented into a networked virtual environment.

IV. C++ PROTOTYPING TOOLS

A. MOTIVATION

Through the development of kinematic algorithms and a model of the upper arm in CLOS, an articulated human arm was constructed. As discussed in Chapter III, an exploratory programming language like CLOS is not the best choice for computationally intensive applications. In order to incorporate this developmental code into an extensible, "next-generation" prototyping tool, there are three basic requirements. First, it is desirable to implement this, initially, into an environment which is non-networked. This is useful to minimize additional complexity and overhead of a fully networked VR system. Second, since the prototype was written in an object oriented style, it is very beneficial for the real-time computer language to also be adaptable to an object oriented style. Last, the environment must utilize the same computer language as the foreseen target application.

To meet these needs, two versions of a prototyping tool were developed, enabling manipulation of a human model's upper body in a C++, object-oriented language environment that is stand-alone and easily extensible to a networked virtual environment application. The first tool developed was an SGI Performer application with a graphical user interface (GUI) which enables manipulation of the human model in order to determine joint construction and coordinate frame orientations. The GUI is particularly desirable when scripting arm movements by forward kinematic methods. Later, a non-GUI version of the tool was developed to aid in the study of the inverse kinematics problem using magnetic sensors attached to the user's arms. Both tools facilitate the development of user interfaces to control the human model.

B. DESCRIPTION

1. Overview

The human model to be manipulated is from University of Pennsylvania, Center for Human Modeling and Simulation, *Jack* software. It is an entity currently used in a networked real-time VE known as NPSNET. This networked VE has been developed and maintained by the staff, faculty, and students at the Naval Postgraduate School in Monterey, CA. The VE currently has the capability of sustaining up to 100 battlefield entities such as helicopters, tanks, airplanes, and most recently, humans [BARH94][PRAT95]. Human motion is controlled by the JackML which provides various postures; however, the upper body largely remains static. Because this networked VE demands dynamic humans who interact with their world, it was suggested that new hand and upper body motion be scripted so that the humans could appear to behave more realistically. But, each time a change was made in the code for upper body motion, the expansive networked virtual environment would have to be exited and recompiled which could take several minutes each time. To do this more rapidly, we found it beneficial to incorporate the required JackML software from the networked application into a stand-alone framework to isolate the development of scripted motion for this networked application.

The code involving a human model was then incorporated into a GUI program which requires much less time to debug and test. This aided in the development of scripted hand and arm signals which were ultimately implemented in the networked application by issuing keyboard commands. As discussed before, this scripted method utilizes the principles of FK and keyboard input which is less intuitive for the user to control. Relative to a generally static human upper body, however, it is an improvement to have user control of the upper body at all. The ultimate goal is to incorporate more dynamic, real-time upper body movement by tracking the user through the use of body-mounted sensors.

2. Design and Implementation

The original prototyping tool is a GUI which required careful design considerations in order to merge two stand-alone applications, Motif and SGI Performer, into one system. The successful integration of the two resulted in an easy-to-manipulate interface which allows basic upper body motion to be developed and performed for the human model, while otherwise keeping the human static in the environment. The scripts for hand and arm signals are coded in the tool software and can be demonstrated through the use of window menu selections. A series of official military hand and arm signals (see Table 2) can be

UNIT SIZE	FORMATION	SPACING	MOVEMENT
Platoon	Line	Close Up	Forward
Squad	Wedge	Open Up	Halt
Fire team	Vee	Disperse	Get Down
	Column		Faster
	Echelon Right		Slower
	Echelon Left		Move Right
			Move Left

Table 2: Tactical Military Hand and Arm Signals

performed upon request [MCIO15]. The second version of the prototyping tool was generated from this GUI to accommodate the use of magnetic sensors and inverse kinematic algorithms. The interface for dynamic human upper body, in this tool version, comes from actual user-directed movement rather than GUI buttons which activate pre-designed, scripted menu-driven upper body movement.

a. Motif Interface

The development of the GUI tool occurred in two basic stages: the development of the windows and menus, and then inclusion of Performer 3D graphics. A Motif window design tool, BX Builder Toolkit, was used to create the layouts of windows, labels, button

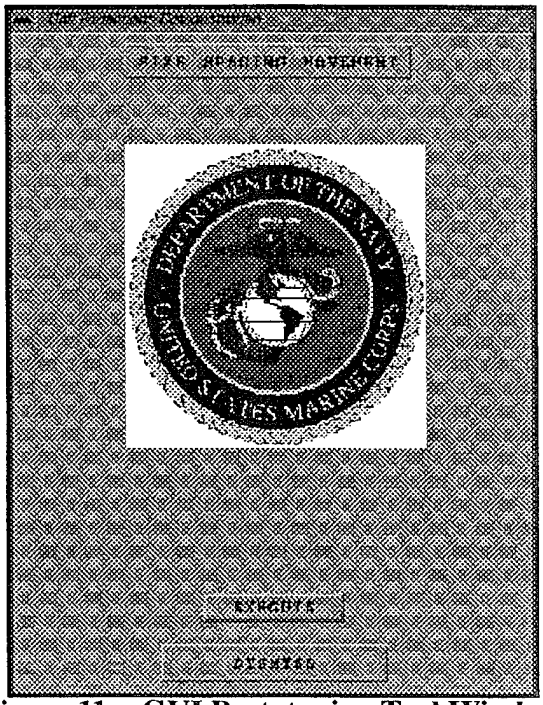


Figure 11: GUI Prototyping Tool Window

types, and locations and types of windows, as seen in Figure 11. It permits easy placement of widgets and creation/editing of callbacks. The GUI menu options (some of which are seen in Figure 12) are the mechanism by which the user can request demonstration of the various hand and arm signals. Once basic design and functionality of the windows and menus were achieved, three windows, two 2-dimensional (2D) graphics and one 3-

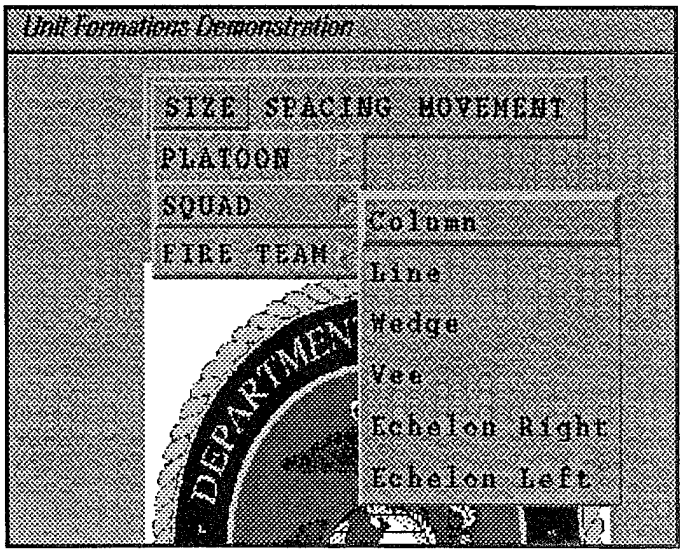


Figure 12: GUI Prototyping Tool Menus

dimensional (3D) Performer window, were incorporated into the GUI. The most significant window in this tool is the 3D graphics window which utilizes Performer and renders the human model executing the hand and arm signals.

b. Performer

In the 3D Performer window, seen in Figure 13, the human's body position is static and is oriented in the middle of the view directly facing the user. The scene is the basic Performer environment scene with ground and sky models. Note that in order to successfully run Performer within the Motif GUI, it is required to specify a single threaded execution mode for Performer. When the GUI application is launched, a function call is made after all the necessary GUI parts have been created to setup Performer. This function does all the standard Performer initializations and setup as well as specific application initializations for the human model. When the GUI is launched, the 3D Performer graphics window appears and is refreshed continuously until the GUI Exit button is pushed.

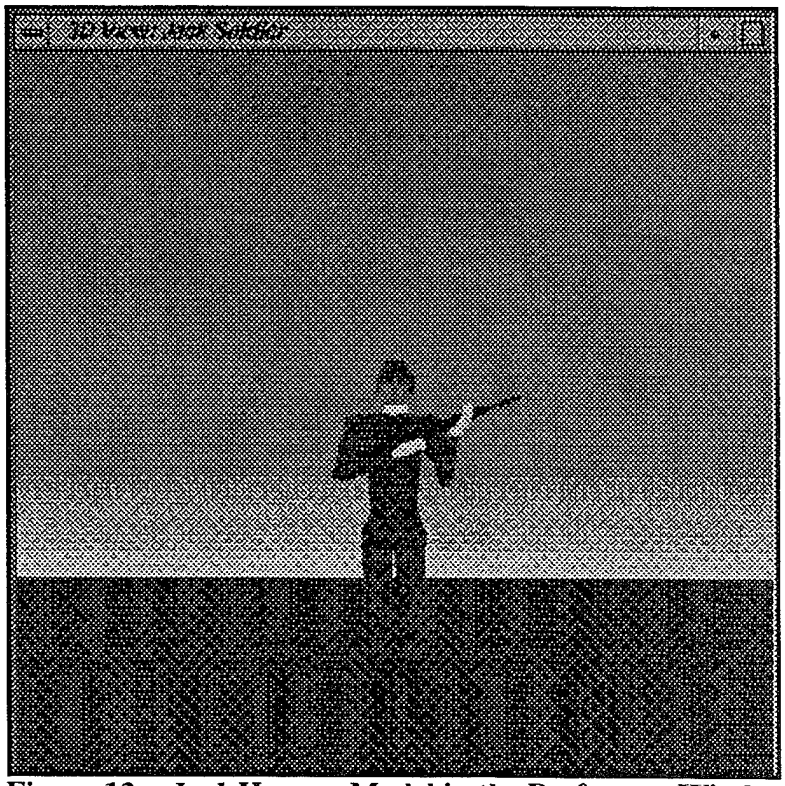


Figure 13: *Jack* Human Model in the Performer Window

3. Jack Motion Library (JackML)

a. Overview

JackML is a real-time module of University of Pennsylvania's *Jack* software that controls smooth motion of the human model. It includes over-sampled, scripted animations for various static and dynamic postures and posture transitions (see Table 3). Leg locomotion, although not of interest in this research, is also handled by this motion library. The library has limited scripted animations which can be played back. Until this research work, the extent of upper body motion was limited to the deployment or non-deployment of the weapon that the human model holds. An important ability with respect to this thesis is being able to override joint angles in real-time, particularly to increase the human's upper body motion capability, interactively. [PRAT95]

STATIC POSTURES
Standing Upright Weapon Deployed
Standing Upright Weapon Not Deployed
Kneeling Weapon Deployed
Kneeling Weapon Not Deployed
Prone Weapon Deployed
Prone Weapon Not Deployed
Dead

DYNAMIC POSTURES
Walking
Running
Crawling

Table 3: JackML Scripted Animations

b. Overriding Joint Angles

The GUI tool can execute programmer scripted motion for the arms by using JackML's vital capability of overriding, in real-time, the default angles normally provided for the human model by JackML. Similarly, the non-GUI tool can dynamically, and in real-time, override motion library joint angles to allow the user to manipulate the model through input from arm-mounted magnetic sensors. Here, instead of programmer-defined joint angles overriding the motion library angles, as demonstrated by the hand and arm signals, joint angles are overridden by user-driven joint angles which are fed into the application through sensor information.

4. Forward Kinematics

As discussed in Chapter III, forward kinematics can be used by the programmer to create more dynamic movement for the computer human, but it is not completely intuitive. For example, the programmer knows the goal position, but very rarely must consider at what angles he will have to move his joints in order to achieve that particular goal position. The GUI tool is very helpful when developing specific scripted arm motions quickly. The joint angles for each DOF in a particular arm segment are directly specified, overriding joint angles in JackML, to produce a desired motion. This GUI tool is able to be tested and debugged quickly and allows for swift development of accurate upper body motions. Upon selecting a hand and arm signal to execute from the menu and pressing the Execute button, the scripted motion is replayed. Figure 14 is a snapshot of a frame from a scripted hand signal, echelon right formation, in the GUI prototyping tool.

5. Inverse Kinematics

The non-GUI tool version was adapted and improved by Scott McMillan from the GUI tool. The tool no longer allows user interaction with the human model through the use of menu buttons. Instead, the user interfaces through magnetic sensors placed on the user's body. In order to allow accurate tracking of a user's upper body in real-time, the Polhemus *3Space* Fastrak sensor system is used [POLH93]. Each sensor returns six DOFs: the

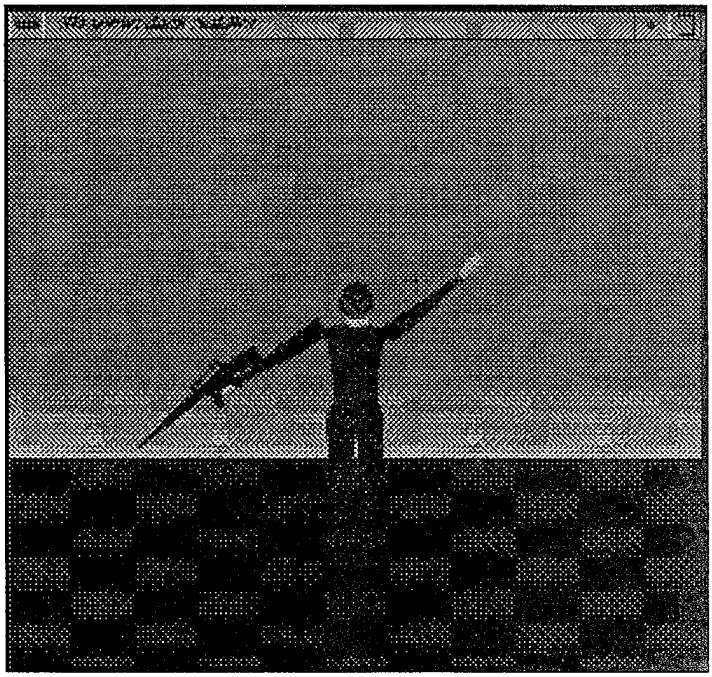


Figure 14: Jack Hand Signal Demonstration

position (X, Y and Z) and orientation (H, P and R) of that part of the body on which the sensor is mounted. The small body sensor receivers in conjunction with the "Long Ranger", a ceiling-mounted transmitter with a range of 15 feet, provides information to drive human upper body motion. The update rates of this sensor system divides its maximum update capability of 60 updates per second (60 updates/sec) between the number of senors used. If one sensor is used it will update at 60 updates/sec. Otherwise, 60 updates/sec will be divided amongst the number of sensors used. Table 4 shows parameters involved in

Joint	Arm Segment	Joint DOF	Cumulative DOF	Number of Sensors for Complete Specification	Number of Sensors for Full Specification
Shoulder	Upper Arm	3	3	1	1
Elbow	Forearm	1	4	1	2
Wrist	Hand	3	7	2	3

TABLE 4. Joint degrees of freedom and sensor requirements

articulating a seven DOF per arm: three in the shoulder, one in the elbow, and three in the wrist. The advantages and disadvantages of the number of sensors to use are as follows:

1. One sensor is not sufficient to uniquely define the seven DOF angles that we need with only six DOF sensor data.
2. Two sensors, however, gives us 12 DOF data elements for seven DOF per arm. This is adequate, giving us what we refer to as a *completely specified solution*[1].
3. Three sensors give us 18 DOF data from which 7 DOF can be computed by eliminating all position data, simplifying the IK problem. This is referred to as the *fully specified problem*[2].

With our real-time requirement, the more sensor information we can get, the less calculations we have to perform, therefore avoiding time-consuming mathematical computations. Up to a certain point, more sensor information allows generation of faster articulations because we can avoid complex inverse kinematic algorithms. Past that point, too much data can also result in a degradation of speed, which can result when not enough data is present. With the fully specified solution, we overcome any need for complex inverse kinematics computation, while disregarding minimal data from sensors. Due to the anticipated types of motions in our target VE, it is not critical to articulate the two DOF clavicle for either the left or right arm. Therefore, both clavicles will remain fixed for our application. In order to try to achieve real-time human upper body articulation, several Polhemus sensor configurations and associated mathematical solutions were explored. Ultimately, the fully specified method, discussed above, was chosen because it appeared simple and able to meet the objectives in a real-time virtual environment.

The C++ software in the non-GUI prototyping tool requires classes which represent the articulated arm and implements the kinematic functions which will produce angles which can override the JackML angles to produced desired movement. A complete

---

1. For the purpose of this paper, when the joint angles can be uniquely determined, the system is said to be "completely specified."
2. A system is "fully specified" when all joint angles are able to be directly measured.

translation of CLOS code into C++ is not required because of the robust nature of the JackML and Performer application. The forward kinematics functionality is not required from the CLOS code because the use of JackML already handles the forward kinematics for the human model. Additional CLOS functionality that is no longer required is the rendering of the human body and the body movement functions as well as some of the basic kinematics mathematical functions. The C++ class hierarchy that was adapted from the CLOS prototype and used in the non-GUI prototyping tool is shown in Figure 15.

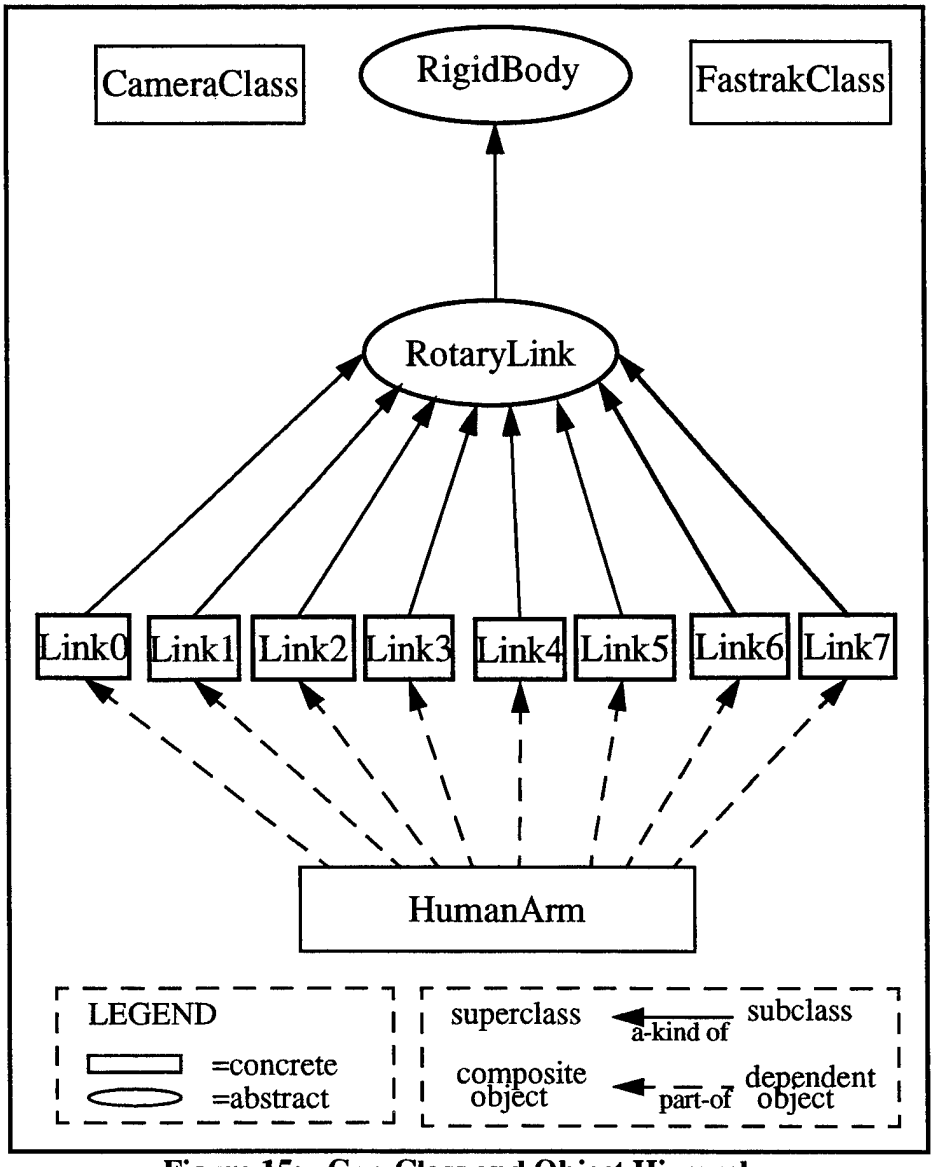


Figure 15: C++ Class and Object Hierarchy

6. Sensor Implementation

The Polhemus *3Space* Fastrak system is controlled by an external control unit which is attached to an RS-232 port on the computer. The Fastrak system is only capable of managing four sensors at one time and all four sensors will be used in this research. There are particular settings on the face of the control unit which are critical to the correct functioning of the system. For this research, the four switches must be in the down position for the sensors to be activated. For more details on the Fastrak system, see the Fastrak User's Manual [POLH93]. The VIM HMD Polhemus sensor will sense head motions and act as the reference location for the human body, while the three arm-mounted sensors are placed on the upper arm, on the forearm, and on the back of the hand, outboard of the wrist. The right arm sensor configuration is shown in Figure 16.

Mounting sensors on the body is a particularly important aspect in sensor tracking. The sensors in this research are mounted on the upper arm and the forearm with two inch wide elastic webbing which is wrapped around the limb snugly with velcro attachment strips. As discussed in Chapter II, the performance of magnetic sensors is degraded by the presence of ferromagnetic material (metals) in the working volume. As a result, it is particularly

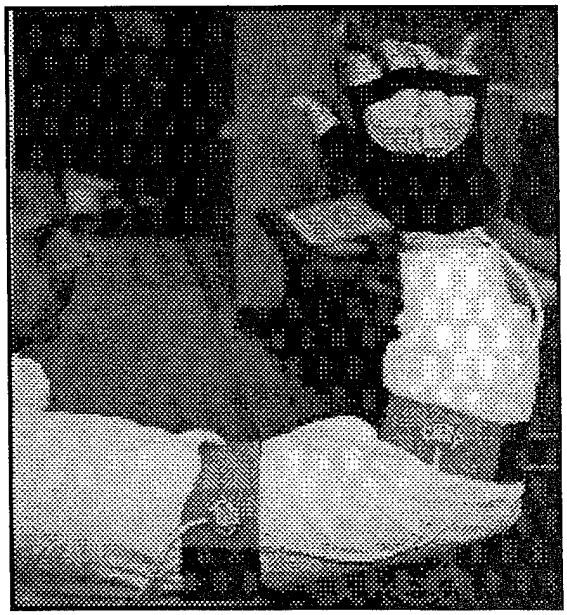


Figure 16: Sensor Position on the Arms

important that the sensor straps be free of any metal connectors in order to avoid any possible ferromagnetic interference. The sensors are sewn on to the elastic bands so that they do not move about on the straps. The hand sensor is mounted on a neoprene glove, which again has an elastic band on the back of the hand. This glove, without enclosed fingers, is "one-size-fits-all" allowing for a snug fit for a wide variety of hand sizes.

Ordering is important when placing the sensors on the right arm. The sensor that is attached to sensor station two on the Fastrak control unit must be attached to the upper arm. The sensor attached to sensor station three is attached to the forearm, and the sensor attached to sensor station four is worn when donning the right-hand glove. Sensors two and three should be placed as close as possible to the nearby joints, i.e. shoulder and elbow respectively, without impeding full mobility of the joints. The sensor mounts are adequate for positioning the sensors on the proper part of the body and for ensuring that they do not drift or move unexpectedly. When attaching the sensors, the glove should be placed on the right hand first, ensuring that when the forearm sensor is attached that the cord from the sensor on the hand is bound under the elastic of the forearm sensor. The same mounting technique applies to the shoulder sensor so that at the shoulder joint, all three cables are gathered and can comfortably trail down the back of the user without interfering with free arm motion.

C. RESULTS and CONCLUSIONS

The four major research objectives which contributed to the development of the prototyping tools described in this chapter are real-time interaction, realism, efficiency and simplicity. First is the desire for real-time, interactive human articulation, particularly for future implementation in a large-scale, networked, interactive virtual environment. The second objective is realistic human movement which is driven by the desire is to create a natural, immersive virtual world in which users will feel comfortable and will participate for extended periods of time. The third objective is efficiency. We must ensure that methods of computation are the most accurate and efficient to produce fast and natural

human motion. Lastly, the fourth research objective is to make the interface simple and intuitive for both the user and the programmer. The physical interface must be comfortable and easy to use with minimal constraints on, and rules for, the user of the virtual environment. The interface software must be a modular library so that it is adaptable and extendable to a variety of interfaces and applications.

There are trade-offs in achieving these goals when contending with many competing objectives. Design decisions are based on the advantages and disadvantages which result from trying to maximize each of our objectives. The key challenge has been balancing the trade-offs between the real-time requirements of the virtual environment, which are critical in a networked environment, while achieving realistic articulated human movements. Lag, or latency, is the delay between the movement of the remotely sensed object and the report of the new position. In graphics applications the report of a new position is manifested as a new frame update. Inherent delays in the tracking system can interfere with achieving real-time performance in the VE to the extent of causing motion sickness [MEYE92]. This research uses the most advanced available sensing technology to produce acceptable real-time results.

Through the use of the two C++ prototyping tools, we are able to manipulate a human model's upper body effectively in real-time (see Figures 14 and 17). The prototyping tools are stand-alone and written in an object oriented, real-time programming language, in order to be able to prototype rapidly and to produce a robust interface that would be compatible with potential environments. To meet such challenges, a number of critical design decisions were made. First, the JackML was used to perform many of the scripted motions of articulated humans in the application and to provide a framework into which joint angles from the sensors are inserted to perform more dynamic upper body motion in real-time. Second, the moveable clavicle was disregarded to eliminate additional articulations that were not critical to motion in our VE. Third, we utilize three sensors per arm, which ultimately eliminates some time-consuming computation and allows the use of only the orientation (H, P, and R) information. With dependence on orientation information, the

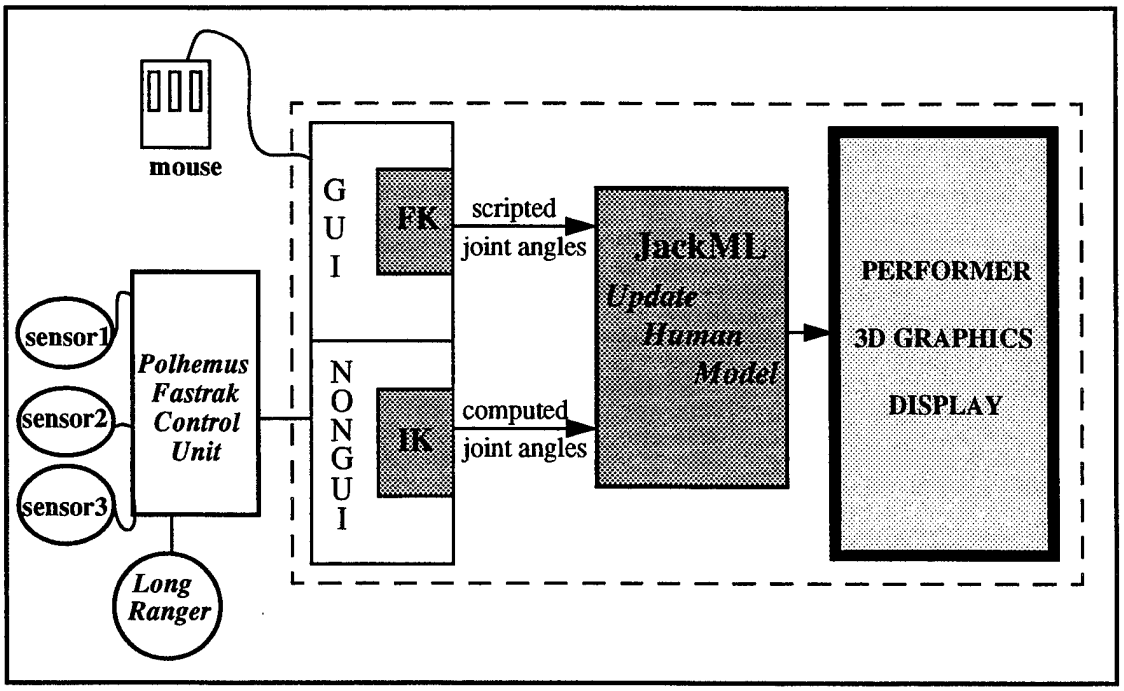


Figure 17: GUI/Non-GUI System Diagram

system is more adaptable for a variety of body sizes without prohibitive calibration routines. The result is an effective human interface to manipulate the upper body of the human model which may be easily integrated into a large-scale networked interactive environment.

V. CONCLUSION

A. RESULTS

Virtual environments have a critical requirement for dynamic articulation of human models. This can be accomplished by two primary methods: forward kinematics and inverse kinematics. Forward kinematics provides the VE user with scripted motion which can be requested by input to some device in order to have the model perform that movement. Alternatively, inverse kinematics gives the user the opportunity to intuitively control the human model's movement through his own motion. This is done through the use of magnetic sensors which provide position and orientation information which can be used to compute the appropriate joint angles for that movement.

The development of human upper body articulation was accomplished in two steps. First was the creation of a experimental prototype in CLOS. It contained classes which model the human arm and its components. These were closely derived from the class structure of Aquarobot in Sandra Davidson's thesis [DAVI93]. Furthermore, this experimental prototype was developed into two stand-alone, real-time, SGI Performer-based prototyping tools. The first was a GUI tool which expedites the creation of scripted upper body motion for replay by keyboard or mouse input. The other tool was a non-GUI prototyping tool which facilitates the development of inverse kinematic software through the use of an interface which tracks body position using sensors. In this case, Polhemus *3Space* Fastrak sensors were used. It should be noted that this interface software is developed without regard for a particular sensor and the interface can be easily modified to work with other types of sensors which provide orientation information.

An entire right arm of a human was designed and articulated in CLOS through forward kinematic methods. This led to the development of the C++ GUI for forward kinematic scripting of upper body motion. The upper portion of the right arm was created and

implemented in both CLOS and the non-GUI C++ tool. The class structure for an arm was inserted into this Performer-based application in order to manipulate the right arm of the human model through the use of magnetic sensors. By reading sensor matrix information, movement of links is initiated and can be viewed on the screen. Although this magnetic sensor system produces six DOF data (three position, three orientation), this interface prototype design, using three sensors per arm, only requires the use of orientation data, resulting in easier and less complicated calibration and the ability to disregard limb lengths of each user. The benefit of these prototype applications is that the code is easily integratable into a VE which requires dynamic human upper body motion.

B. LESSONS LEARNED

Three important lessons have been learned in the process of this research project. The primary lesson is the understanding of strengths and weaknesses of computer programming languages. This research used two different languages, CLOS (LISP) and C++ in two distinct phases of this research. CLOS was used in initial prototyping for its benefits when developing experimental systems. It is easy to debug and can be executed quickly. Both CLOS and C++ support object oriented programming. This is a particular advantage when trying to maintain abstraction away from a particular interface for easy adaptation to other types of tracking interfaces. C++, however, has its advantage of handling computationally intensive algorithms that need to run in real-time. It is also the language most frequently used with graphics applications. This lesson is one that was vividly illustrated during the conduct of this research.

The second is the design of the sensor configuration which uses three sensors per arm to articulate seven DOFs. It has the distinct advantage of not requiring the use of position information. With the availability of 18 pieces of sensor data and a requirement for only seven, all the position information can be disregarded. This eliminates complicated calibration and measuring requirements of human limbs. The third lesson is the importance of mounting the sensors to achieve both physical comfort and reliable data. The sensor must

be fastened as securely as possible to the body in order to achieve accurate motion readings, while maintaining comfort to the user. Additionally, the magnetic sensor has problems with ferromagnetic interference, so the mechanism by which the sensors are attached must not include metal connectors or clips for fastening.

C. EXTENSIBILITY OF SOFTWARE TO OTHER INTERFACES

The code produced during the research for this thesis is strongly affiliated with object oriented programming. The classes developed for the arm structure in both CLOS and C++ can be easily implemented with any other interface for human articulation. These classes are abstracted from any particular interface mechanism, or software, to ensure extensibility of basic human modeling and kinematic algorithms without regard to the type of tracking device. For this research, Polhemus *3Space* Fastrak sensor system is the preferred interface, but as the interface technology changes, the human arm classes and kinematic algorithms that were developed in this thesis can be easily adapted to accommodate a variety of other articulation interfaces. Furthermore, with the configuration of sensors in this research, it is conceivable that a tracking system which only senses three DOF (orientation data) could be an excellent alternative to the current tracking technology used in this research.

D. FUTURE TRACKING ALTERNATIVES

Because magnetic sensors are readily available and affordable, they are currently an excellent option for a human upper body articulation interface in VEs. A variety of human tracking interfaces are in the research and development stages, and may in the future be more efficient, easier to use options, relative to the current magnetic sensor systems. Even though magnetic tracking technology has some drawbacks, it clearly meets the current research requirements in terms of capability and price. Alternatives that are being explored include optical and acoustic sensor systems.

E. FUTURE WORK

This research provides the ground work for future upper body articulations in a CLOS prototyping environment as well as a C++ application. The following is a list of topics for future work in this research area.

- Instantiate and articulate two entire arms in the non-GUI prototyping tool.
- Test and evaluate the articulations of the upper body of the human model in the non-GUI protoyping tool.
- Analyze and compare various inverse kinematic methods and sensor configurations with the ones presented in this thesis.
- Articulate other upper body DOFs that may be useful in the VE.
- Implement interactive human upper body articulation in a large-scale networked VE, like NPSNET.
- Communicate articulation information over a network efficiently using DIS protocol and the least bandwidth possible in order to sustain and articulate from four to upwards of 160 human entities in the world.

F. SUMMARY

This thesis has explored the development of an interface for VEs which articulates a virtual human's upper body possessing seven DOFs per arm with off-the-shelf magnetic sensors in real-time. At the outset, CLOS was used to prototype kinematic algorithms to articulate human arms and to establish and test classes and kinematic algorithms in order to produce modular code (Appendix A) which can be easily extended and reused for other types of upper body articulation interfaces. This code simulates the sensor functionality which would be used in the C++ prototyping environment. The polhemus class enables testing of inverse kinematics functions to ensure that correct joint angles are being calculated. Appendix B is the script of the results produced from the CLOS inverse kinematics routine described in Chapter III.

The CLOS code was translated into C++ which enabled straightfoward class reuse in a real-time GUI prototyping tool, an application based on SGI Performer. With three Polhemus sensors attached to the arm of a user, the articulations of the human are realistic and efficient. The advantage with this sensor configuration is that only orientation

information is used in the computation, thus alleviating any complicated calibration and tedious body measurements to ensure accurate motion is achieved. This proof of concept was developed to begin to evaluate the efficiency and comfort of this sensor tracking configuration relative to the few that exist currently.

The C++ prototyping tools enabled development of both the scripted, keyboard or mouse executed movement and the user-directed upper body movement through the use of sensors attached to the upper body. Although not intuitive, scripts developed through forward kinematic methods are fast and contribute significant real-time dynamics to a VE inhabited with humans and other entities. The more interesting and desirable method for upper body articulations in VEs is through inverse kinematic methods which are driven through sensor-obtained body information. With the user moving his own body to direct the motion of his virtual human entity, he is able to more realistically and intuitively drive the interactions of the human in the world. With the integration of the real-time articulation software developed in this thesis, networked VEs, like NPSNET, will be more realistic and of significant practical use for entertainment and training purposes.

APPENDIX A. LISP CODE

This appendix contains the LISP (CLOS) code for all results pertaining to the simulated Human generated in this thesis.

LOAD-FILES.CL

```
;*************************************************************
;LOAD-FILES.CL loads in all files associated with and necessary for the
;kinematic simulation of the simulated Human as outlined in this thesis.
;*************************************************************

(defun load-jack()

; Abstract class files
  (load "rigid-body.cl")
  (load "link.cl")

; Specific Graphics and Display files
  (load "camera.cl")
  (load "video-camera.cl")


; Specific robot files
  (load "kinematics.cl")
  (load "human.cl")
  (load "human-arm.cl")
  (load "jack-link.cl")
  (load "polhemus.cl")

; Specific demo files
  (load "demo-jack.cl")
)
```

DEMO-JACK.CL

```
;*************************************************************
;DEMO-JACK.CL contains the code necessary to demonstrate the
;kinematics of the simulated Human arm.
;This code was written by Shirley M. Pratt and modified by the author.
;*************************************************************

(defun demo-jack()

; video-camera demo of jack
(jack-video *black*)

(new-picture camera-1 jack-1 0)

(zoom-camera camera-1 10)

(tilt-camera camera-1 -30)

(rotate-camera camera-1 180)
(new-picture camera-1 jack-1 0)

(move-incremental jack-1 '((0 0 0 0 0 0) (0.1 0.5)))
(new-picture camera-1 jack-1 0)
(H-to-jack (sensor (link2 (upperarm (rightarm jack-1)))) jack-1)

(move-incremental jack-1 '((0 0 0 0 0 0) (0.2 0.6)))
(new-picture camera-1 jack-1 0)
(H-to-jack (sensor (link2 (upperarm (rightarm jack-1)))) jack-1)

(move-incremental jack-1 '((0 0 0 0 0 0) (0.3 0.7)))
(new-picture camera-1 jack-1 0)
(H-to-jack (sensor (link2 (upperarm (rightarm jack-1)))) jack-1)

(move-incremental jack-1 '((0 0 0 0 0 0) (0.4 0.8)))
(new-picture camera-1 jack-1 0)
(H-to-jack (sensor (link2 (upperarm (rightarm jack-1)))) jack-1)

(move-incremental jack-1 '((0 0 0 0 0 0) (0.9 0.9)))
(new-picture camera-1 jack-1 0)
(H-to-jack (sensor (link2 (upperarm (rightarm jack-1)))) jack-1)

(move-incremental jack-1 '((0 0 0 0 0 0) (1.5 1.5)))
(new-picture camera-1 jack-1 0)
(H-to-jack (sensor (link2 (upperarm (rightarm jack-1)))) jack-1)

(dotimes (i 30)
(new-picture camera-1 jack-1 0))

;the end of the demonstration
(format t "    Jack is DONE!  ~%~%")
(reset-windows))
```

RIGID-BODY.CL

```
;*************************************************************
;RIGID-BODY.CL defines the class rigid-body associated with the
;simulated Human Arm kinematics model.
;This code was written by Dr. McGhee and modified by the author.
;*************************************************************

(defclass rigid-body()
 ((location          ;The three-vector (x y z) in world coordinates.
   :initarg :location
   :accessor location)
  (velocity          ;The six-vector (u v w p q r) in body coordinates.
   :initform '(1 1 1 .1 .1 .1)
   :initarg :velocity
   :accessor velocity)
  (velocity-growth-rate  ;The vector (u-dot v-dot w-dot p-dot q-dot r-dot).
   :accessor velocity-growth-rate)
  (forces-and-torques    ;The vector (Fx Fy Fz L M N) in body coordinates.
   :initform '(0 0 0 0 0 0)
   :accessor forces-and-torques)
  (moments-of-inertia    ;The vector (Ix Iy Iz) in principal axis coordinates.
   :initform '(1 1 1)
   :initarg :moments-of-inertia
   :accessor moments-of-inertia)
  (mass
   :initform 1
   :initarg :mass
   :accessor mass)
  (node-list    ;(x y z 1) in body coord for each node. Starts with (0 0 0 1).
   :initarg :node-list
   :accessor node-list)
  (polygon-list
   :initarg :polygon-list
   :accessor polygon-list)
  (transformed-node-list ;(x y z 1) in earth coord for each node in node-list.
   :accessor transformed-node-list)
  (H-matrix
   :initform (unit-matrix 4)
   :accessor H-matrix)
  (current-time
   :accessor current-time)))

(defmethod move-incremental ((body rigid-body) increment-list)
 (setf (H-matrix body)
  (matrix-multiply (H-matrix body)
                   (homogeneous-transform
                    (first increment-list)     ;body z rotation
                    (second increment-list)    ;body y rotation
                    (third increment-list)     ;body x rotation
                    (fourth increment-list)    ;body x translation
                    (fifth increment-list)     ;body y translation
                    (sixth increment-list))))  ;body z translation
 (transform-node-list body)
 (update-position body))
```

```
(defmethod transform-node-list ((body rigid-body))
  (setf (transformed-node-list body)
  (mapcar #'(lambda (node-location)
        (post-multiply (H-matrix body) node-location))
        (node-list body)))
  (format t "transformed node list for link ~d is ~a ~% "
        body (transformed-node-list body)))

(defmethod update-position ((body rigid-body))
  (setf (location body) (firstn 3 (first (transformed-node-list body)))))
```

HUMAN.CL

```
;*************************************************************
;HUMAN.CL defines the classes static-human and human associated
;with the simulated Human kinematic model.
;This code was adapted by the author from Aqua.cl which was originally
;written by Dr. McGhee.
;*************************************************************

(defclass static-human (rigid-body)
  ((node-list
  :initform '((0 -1 2.75 1) (-1 -3 -1.25 1) (-1 1 -1.25 1) (-1 1 2.75 1)
        (-1 -3 2.75 1) (2 -3 2.75 1) (2 1 2.75 1) (-1 -3 5.75 1)
        (2 -3 5.75 1) (2 1 5.75 1)
        (-1 1 5.75 1) (0 -1 -3.25 1) (0 0 -2.25 1) (0 -1 -1.25 1)
        (0 -2 -2.25 1) (3 0 2.75 1)
        (3 -2 2.75 1) (3 0 5.75 1) (3 -2 5.75 1) (0 -1 0 1) (0 0 0 1)))
  (polygon-list
  :initform '((1 2 3 4) (3 4 5 6) (4 7) (5 8) (6 9) (3 10) (11 12 13 14)
         (13 0)(0 15) (0 16) (16 18) (15 17) (19 20)))
  (H-matrix
  :initform (homogeneous-transform 0 0 0 0 0 0))))

(defclass human ()
   ((body
    :initform (make-instance 'static-human)
    :accessor body)
   (rightarm
    :initform (make-instance 'human-arm :arm-attachment-angle (deg-to-rad 0))
    :accessor rightarm)
  ;(leftarm
  ; :initform (make-instance 'human-arm :arm-attachment-angle (deg-to-rad 180))  ;
:accessor leftarm)
))

(defmethod initialize ((person human))
  (transform-node-list (body person))
  (initialize-arm (rightarm person)(body person)))

(defun jack-video (draw-color)
  (setf jack-1 (make-instance 'human))
  (initialize jack-1)
  (create-video-camera-1)
  (setf jack-color draw-color))

; this is the new-picture method for the "video-camera", Not "camera"
(defmethod new-picture ((camera video-camera) (person human) draw-color)
  (erase-image-window camera)
  (take-picture camera person draw-color)
  (expose-image camera))

(defmethod take-picture ((camera camera) (person human) draw-color)
  (take-picture camera (body person) draw-color)
  (take-picture-rightarm camera person draw-color))

(defmethod move-incremental ((person human) increment-list)
  (move-incremental (body person) (first increment-list))
  (move-incremental  (rightarm person) (second increment-list)))
```

HUMAN-ARM.CL

```
;*****************************************************************
;HUMAN-ARM.CL defines the class human-arm and its associated
;dependent classes associated with the simulated Human kinematic model.
;This code was adapted by the author from Aqua-leg.cl which was originally
; written by Dr. McGhee.
;*****************************************************************

(defclass human-arm ()
  ((arm-attachment-angle
    :initarg :arm-attachment-angle
    :accessor arm-attachment-angle)
   (link0
    :initform (make-instance 'link0)
    :accessor link0)
   (upperarm
    :initform (make-instance 'upperarm)
    :accessor upperarm)
  ; (forearm
  ;  :initform (make-instance 'forearm)
  ;  :accessor forearm)
  ; (hand
  ;  :initform (make-instance 'hand)
  ;  :accessor hand)
))

(defclass upperarm ()
  ((link1                    ;inboard axis is shoulder azimuth axis
    :initform (make-instance 'link1)
    :accessor link1)
   (link2
    :initform (make-instance 'link2)  ;inboard axis is shoulder elevation axis
    :accessor link2)))

;(defclass forearm ()
;  ((link3
;    :initform (make-instance 'link3) ;inboard axis is elbow axis
;    :accessor link3)
;   (link4
;    :initform (make-instance 'link4) ;inboard axis is forearm roll axis
;    :accessor link4)))

;(defclass hand ()
;  ((link5
;    :initform (make-instance 'link5) ;inboard axis is hand azimuth axis
;    :accessor link5)
;   (link6
;    :initform (make-instance 'link6) ;inboard axis is hand elevation axis
;    :accessor link6)
;   (link7
;    :initform (make-instance 'link7) ;inboard axis is hand roll axis
;    :accessor link7)))
```

```
(defmethod initialize-arm ((arm human-arm)(body static-human))
   (format t "Entering initialize ARM/human-arm.cl ~%")
   (setf (inboard-link (link0 arm)) body)
   (setf (inboard-link (link1 (upperarm arm)))
            (link0 arm))
   (setf (inboard-link (link2 (upperarm arm)))
            (link1 (upperarm arm)))
  ;(setf (inboard-link (link3 (forearm arm)))
  ;         (link2 (upperarm arm)))
  ;(setf (inboard-link (link4 (forearm arm)))
  ;         (link3 (forearm arm)))
  ;(setf (inboard-link (link5 (hand arm)))
  ;         (link4 (forearm arm)))
  ;(setf (inboard-link (link6 (hand arm)))
  ;         (link5 (hand arm)))
  ;(setf (inboard-link (link7 (hand arm)))
  ;         (link6 (hand arm)))

   (setf (reference-link (sensor (link2 (upperarm arm))))
            (link2 (upperarm arm)))

   (rotate-link (link0 arm)
            (arm-attachment-angle arm))
   (rotate-link (link1 (upperarm arm))
            (joint-angle (link1 (upperarm arm))))
   (rotate-link (link2 (upperarm arm))
            (joint-angle (link2 (upperarm arm))))
  ;(rotate-link (link3 (forearm arm))
  ;         (joint-angle (link3 (forearm arm))))
  ;(rotate-link (link4 (forearm arm))
  ;         (joint-angle (link4 (forearm arm))))
  ;(rotate-link (link5 (hand arm))
  ;         (joint-angle (link5 (hand arm))))
  ;(rotate-link (link6 (hand arm))
  ;         (joint-angle (link6 (hand arm))))
  ;(rotate-link (link7 (hand arm))
  ;         (joint-angle (link7 (hand arm))))

   (format t "DONE initialize ARM/human-arm.cl ~%"))

(defmethod take-picture-rightarm ((camera camera) (person human) draw-color)
  (take-picture camera (link0 (rightarm person)) draw-color)
  (take-picture camera (link1 (upperarm (rightarm person))) draw-color)
  (take-picture camera (link2 (upperarm (rightarm person))) *magenta*)
   ;(take-picture camera (link2 arm) draw-color)
   ;(take-picture camera (link3 arm) draw-color)
   ;(take-picture camera (link3 arm) *magenta*)
   ;(take-picture camera (link4 arm) *green*)
   ;(take-picture camera (link4 arm) draw-color)
   ;(take-picture camera (link5 arm) draw-color)
   ;(take-picture camera (link5 arm) *green*)
   ;(take-picture camera (link6 arm) draw-color)
   ;(take-picture camera (link7 arm) *yellow*)
)
```

```
(defmethod move-incremental ((arm human-arm) increment-list)
  (rotate-link (link0 arm) (arm-attachment-angle arm))
  (rotate-link (link1 (upperarm arm))
          (+ (first increment-list) (joint-angle
              (link1 (upperarm arm)))))
  (rotate-link (link2 (upperarm arm))
          (+ (second increment-list) (joint-angle
              (link2 (upperarm arm)))))
 ;(rotate-link (link3 (forearm arm))
 ;          (+ (third increment-list) (joint-angle
 ;              (link3 (forearm arm)))))
 ;(rotate-link (link4 (forearm arm))
 ;          (+ (fourth increment-list) (joint-angle
 ;              (link4 (forearm arm)))))
 ;(rotate-link (link5 (hand arm))
 ;          (+ (fifth increment-list) (joint-angle
 ;              (link5 (hand arm)))))
 ;(rotate-link (link6 (hand arm))
 ;          (+ (sixth increment-list) (joint-angle
 ;              (link6 (hand arm)))))
 ;(rotate-link (link7 (hand arm))
 ;          (+ (seventh increment-list) (joint-angle
 ;              (link7 (hand arm)))))
)
```

LINK.CL

```
;*************************************************************
;
;LINK.CL contains the code necessary to define a link, rotary link, and
;sliding link associated with the simulated Human kinematic model.
;This code was written by Dr. McGhee.
;*************************************************************
;

(defclass link (rigid-body)
  ((motion-limit-flag
    :initform nil
    :accessor motion-limit-flag)
   (inboard-link-length
    :initarg :inboard-link-length
    :accessor inboard-link-length)
   (inboard-twist-angle
    :initarg :inboard-twist-angle
    :accessor inboard-twist-angle)
   (joint-displacement
    :initarg :joint-displacement
    :accessor joint-displacement)
   (joint-angle
    :initarg :joint-angle
    :accessor joint-angle)
   (inboard-link
    :initarg :inboard-link
    :accessor inboard-link)
   (A-matrix
    :accessor A-matrix)))

(defclass rotary-link (link)
  ((min-joint-angle
    :initarg :min-joint-angle
    :accessor min-joint-angle)
   (max-joint-angle
    :initarg :max-joint-angle
    :accessor max-joint-angle)))

(defclass sliding-link (link)
  ((min-joint-displacement
    :initarg :min-joint-displacement
    :accessor min-joint-displacement)
   (max-joint-displacement
    :initarg :max-joint-displacement
    :accessor max-joint-displacement)))
```

JACK-LINK.CL

```
;*************************************************************
;
;JACK-LINK.CL defines the classes Link0 through Link7 associated with
;the arm of the simulated Human kinematic model.
;This code was adapted by the author from Aqua-link.cl which was originally
;written by Dr. McGhee.
;*************************************************************
;

;these link parameters are based on Modified D-H Notation

;imaginary link
(defclass link0 (rotary-link)
  ((inboard-link-length :initform 0)
   (inboard-twist-angle :initform (deg-to-rad 0))
   (joint-displacement :initform 0)
   (joint-angle :initform (deg-to-rad 0))
   (min-joint-angle :initform (deg-to-rad -360))
   (max-joint-angle :initform (deg-to-rad 360))
   (node-list :initform '((0 0 0 1) (0 0 0 1) (0 0 0 1)))
   (polygon-list :initform '((1 2)))))

; shoulder elevation
(defclass link1 (rotary-link)
  ((inboard-link-length :initform 0)
   (inboard-twist-angle :initform (deg-to-rad 0))
   (joint-displacement :initform 0)
   (joint-angle :initform (deg-to-rad 0))
   (min-joint-angle :initform (deg-to-rad -80))
   (max-joint-angle :initform (deg-to-rad 180))
   (node-list :initform '((0 0 0 1) (0 0 0 1) (0 0 0 1)))
   (polygon-list :initform '((1 2)))))

; shoulder azimuth
(defclass link2 (rotary-link)
  ((inboard-link-length :initform 0)
   (inboard-twist-angle :initform (deg-to-rad 90))
   (joint-displacement :initform 0)
   (joint-angle :initform (deg-to-rad -90))
   (min-joint-angle :initform (deg-to-rad -90))
   (max-joint-angle :initform (deg-to-rad 160))
   (node-list :initform '((0 0 0 1) (0 0 0 1) (0 2.0 0 1)))
   (polygon-list :initform '((1 2)))
   (sensor
    :initform (make-instance 'polhemus)
    :accessor sensor)))

; forearm roll
;(defclass link3 (rotary-link)
;  ((inboard-link-length :initform 0)
;   (inboard-twist-angle :initform (deg-to-rad 90))
;   (joint-displacement :initform -2.0)
;   (joint-angle :initform (deg-to-rad -90))
;   (min-joint-angle :initform (deg-to-rad -180))
;   (max-joint-angle :initform (deg-to-rad 0))
;   (node-list :initform '((0 0 0 1) (0 0 0 1) (0 0 0 1)))
;   (polygon-list :initform '((1 2)))))
```

```
; elbow
;(defclass link4 (rotary-link)
;  ((inboard-link-length :initform 0)
;   (inboard-twist-angle :initform (deg-to-rad 90))
;   (joint-displacement :initform 0)
;   (joint-angle :initform (deg-to-rad 180))
;   (min-joint-angle :initform (deg-to-rad 180))
;   (max-joint-angle :initform (deg-to-rad 350))
;   (node-list :initform '((0 0 0 1) (0 0 0 1) (0 0 -2.0 1)))
;   (polygon-list :initform '((1 2)))))

; hand elevation
;(defclass link5 (rotary-link)
;  ((inboard-link-length :initform 0)
;   (inboard-twist-angle :initform (deg-to-rad 90))
;   (joint-displacement :initform -2.0)
;   (joint-angle :initform (deg-to-rad 90))
;   (min-joint-angle :initform (deg-to-rad 0))
;   (max-joint-angle :initform (deg-to-rad 180))
;   (node-list :initform '((0 0 0 1) (0 0 0 1) (0 0 0 1)))
;   (polygon-list :initform '((1 2)))))

; hand roll
;(defclass link6 (rotary-link)
;  ((inboard-link-length :initform 0)
;   (inboard-twist-angle :initform (deg-to-rad 90))
;   (joint-displacement :initform 0)
;   (joint-angle :initform (deg-to-rad -90))
;   (min-joint-angle :initform (deg-to-rad -180))
;   (max-joint-angle :initform (deg-to-rad 0))
;   (node-list :initform '((0 0 0 1) (0 0 0 1) (1 0 0 1) (0.5 -0.5 0 1)))
;   (polygon-list :initform '((1 2) (1 3)))))

; hand azimuth
;(defclass link7 (rotary-link)
;  ((inboard-link-length :initform 0)
;   (inboard-twist-angle :initform (deg-to-rad 90))
;   (joint-displacement :initform 0)
;   (joint-angle :initform (deg-to-rad 0))
;   (min-joint-angle :initform (deg-to-rad -30))
;   (max-joint-angle :initform (deg-to-rad 70))
;   (node-list :initform '((0 0 0 1) (0 0 0 1) (0 0 -2.0 1)))
;   (polygon-list :initform '((1 2)))))

(defmethod update-A-matrix ((link link))
  (with-slots (inboard-link-length inboard-twist-angle
               joint-displacement  joint-angle A-matrix) link
    (setf A-matrix (mdh-matrix
                    inboard-link-length
                    (cos inboard-twist-angle) (sin inboard-twist-angle)
                    joint-displacement
                    (cos joint-angle) (sin joint-angle)))
    (format t "A-matrix for ~a is : ~d ~%" link (A-matrix link))))
```

```
(defmethod rotate ((link rotary-link) angle)
 (setf (joint-angle link) angle)
 (update-A-matrix link)
 (format t "H-matrix for ~d is : ~d ~%" (inboard-link link)
                          (H-matrix (inboard-link link)))
 (setf (H-matrix link) (matrix-multiply (H-matrix (inboard-link link))
                          (A-matrix link)))
 (format t "H-matrix for ~d is : ~d ~%" link (H-matrix link))
 (transform-node-list link)
 (format t "joint angle is ~a ~% " (joint-angle link)))

(defmethod rotate-link ((link rotary-link) angle)
 (cond ((> angle (max-joint-angle link))
                          (rotate link (max-joint-angle link))
                          (setf (motion-limit-flag link) t))
     ((< angle (min-joint-angle link))
                          (rotate link (min-joint-angle link))
                          (setf (motion-limit-flag link) t))
     (t (rotate link angle) (setf (motion-limit-flag link) nil))))
```

POLHEMUS.CL

```
;*****************************************************************
;POLHEMUS.CL defines the polhemus class which simulates the functionality
;of real body-mounted sensors. This class includes functions which perform
;inverse kinematics for the arm of the Human kinematics model.
;This code was written by the author.
;*****************************************************************

(defclass polhemus (rigid-body)
  ((posorient
     :initform '(0 0 0 0 0 0)
     :initarg : posorient
     :accessor posorient)
   (reference-link
     :initarg :reference-link
     :accessor reference-link)
   (quaternion-flag
     :initform 0
     :initarg :quaternion-flag
     :accessor quaternion-flag)
   (position-flag
     :initform 1
     :initarg :position-flag
     :accessor position-flag)
   (euler-angle-flag
     :initform 0
     :initarg :euler-angle-flag
     :accessor euler-angle-flag)
   (direction-cosine-flag
     :initform 1
     :initarg :direction-cosine-flag
     :accessor direction-cosine-flag)))

(defmethod read-sensor (sensor polhemus)
   (format t " H-matrix for the sensor : ~d ~%"
             (H-matrix sensor))
   (format t " H-matrix for ~d : ~d ~%"
             (reference-link sensor)
             (H-matrix (reference-link sensor)))
   (setf (H-matrix sensor) (H-matrix (reference-link sensor)))
   (format t " H-matrix for the sensor : ~d ~%"
             (H-matrix sensor)))
```

```
(defmethod H-to-euler (sensor polhemus)
   (format t "I am in H-to-Euler function.~%")
   ;extract a1,a2,a3,b3,c3 from homogeneous matrix
   (let* ((a1 (first (first H-matrix)))
          (a2 (first (second H-matrix)))
          (a3 (first (third H-matrix)))
          (b3 (second (third H-matrix)))
          (c3 (third (third H-matrix))))
   (format t "H-matrix = ~a ~%" H-matrix)
   (format t "a1 = ~a ~%" a1)
   (format t "a2 = ~a ~%" a2)
   (format t "a3 = ~a ~%" a3)
   (format t "b3 = ~a ~%" b3)
   (format t "c3 = ~a ~%" c3)
   (setf new-elevation (asin (- a3)))
   (setf new-azimuth (*(acos(/ a1 (cos new-elevation))) (signum a2)))
   (setf new-roll (*(acos(/ c3 (cos new-elevation))) (signum b3))))
   (format t "azimuth = ~a ~%" new-azimuth)
   (format t "elevation = ~a ~%" new-elevation)
   (format t "roll = ~a ~%" new-roll)
   (setf orientation (list new-azimuth new-elevation new-roll))
   (format t "Euler angles are ~a ~%" orientation))

(defmethod H-to-jack ((sensor polhemus) (person human))
   (format t "I am in H-to-jack function.~%")
   (format t "The sensor is on ~a ~%" (reference-link sensor))
   (read-sensor sensor)
   (setf inv-hmatrix (inverse-H (H-matrix (body person))))
   (format t "inverse body H matrix = ~a ~%" inv-hmatrix)
   (setf T-matrix (matrix-multiply inv-hmatrix (H-matrix sensor)))
   (format t "T matrix = ~a ~%" T-matrix)

  ;extract c1, b2 from T matrix, during IK
  (let* ((c1 (third (first T-matrix)))
         (c2 (third (second T-matrix)))
         (a3 (first (third T-matrix)))
         (b3 (second (third T-matrix))))

   (format t "c1 = ~a ~%" c1)
   (format t "c2 = ~a ~%" c2)
   (format t "a3 = ~a ~%" a3)
   (format t "b3 = ~a ~%" b3)

   (setf theta1 (atan c1 (- c2)))
   (setf theta2 (atan a3 b3))
   (setf theta1a (mod (atan c1 (- c2)) (* 2 pi)))
   (setf theta2a (mod (atan a3 b3)(* 2 pi))))

   (format t "joint angle for link1 = ~a ~%" theta1)
   (format t "joint angle for link2 = ~a ~%" theta2)
   (format t "joint angle for link1 = ~a ~%" theta1a)
   (format t "joint angle for link2 = ~a ~%" theta2a))
```

KINEMATICS.CL

```
;*****************************************************************
;
;KINEMATICS.CL defines the transformation and mathematical functions
;associated with the Human kinematic model.
;This code was written by Prof. McGhee (robot-kinematics.cl) with added
; functionality contributed by the author.
;*****************************************************************
;

(defun transpose (matrix)          ;A matrix is a list of row vectors.
  (cond ((null (cdr matrix)) (mapcar 'list (car matrix)))
        (t (mapcar 'cons (car matrix) (transpose (cdr matrix))))))

(defun dot-product (vector-1 vector-2) ;A vector is a list of numerical atoms.  (apply
'+ (mapcar '* vector-1 vector-2)))

(defun vector-magnitude (vector) (sqrt (dot-product vector vector)))

(defun post-multiply (matrix vector)
  (cond ((null (rest matrix)) (list (dot-product (first matrix) vector)))
        (t (cons (dot-product (first matrix) vector)
                 (post-multiply (rest matrix) vector)))))

(defun pre-multiply (vector matrix)
  (post-multiply (transpose matrix) vector))

(defun matrix-multiply (A B)     ;A and B are conformable matrices.
  (cond ((null (cdr A)) (list (pre-multiply (car A) B)))
        (t (cons (pre-multiply (car A) B) (matrix-multiply (cdr A) B)))))

(defun chain-multiply (L)        ;L is a list of names of conformable matrices.
  (cond ((null (cddr L)) (matrix-multiply (eval (car L)) (eval (cadr L))))
        (t (matrix-multiply (eval (car L)) (chain-multiply (cdr L))))))

(defun cycle-left (matrix) (mapcar 'row-cycle-left matrix))

(defun row-cycle-left (row) (append (cdr row) (list (car row))))

(defun cycle-up (matrix) (append (cdr matrix) (list (car matrix))))

(defun unit-vector (one-column length)        ;Column count starts at 1.
  (do ((n length (1- n))
       (vector nil (cons (cond ((= one-column n) 1) (t 0)) vector)))
      ((zerop n) vector)))

(defun unit-matrix (size)
  (do ((row-number size (1- row-number))
  (I nil (cons (unit-vector row-number size) I)))
      ((zerop row-number) I)))

(defun concat-matrix (A B)  ;A and B are matrices with equal number of rows.
  (cond ((null A) B)
        (t (cons (append (car A) (car B)) (concat-matrix (cdr A) (cdr B))))))

(defun augment (matrix)
  (concat-matrix matrix (unit-matrix (length matrix))))

(defun normalize-row (row) (scalar-multiply (/ 1.0 (car row)) row))
```

```
(defun scalar-multiply (scalar vector)
 (cond ((null vector) nil)
     (t (cons (* scalar (car vector))
           (scalar-multiply scalar (cdr vector))))))

(defun solve-first-column (matrix)      ;Reduces first column to (1 0 ... 0).
 (do* ((remaining-row-list matrix (rest remaining-row-list))
     (first-row (normalize-row (first matrix)))
     (answer (list first-row)
          (cons (vector-add (first remaining-row-list)
                     (scalar-multiply (- (caar remaining-row-
list))                                   first-row))
           answer)))
    ((null (rest remaining-row-list)) (reverse answer))))

(defun vector-add (vector-1 vector-2) (mapcar '+ vector-1 vector-2))

(defun vector-subtract (vector-1 vector-2) (mapcar '- vector-1 vector-2))

(defun first-square (matrix) ;Returns leftmost square matrix from argument.
(do ((size (length matrix))
     (remainder matrix (rest remainder))
     (answer nil (cons (firstn size (first remainder)) answer)))
    ((null remainder) (reverse answer))))

(defun firstn (n list)
 (cond ((zerop n) nil)
      (t (cons (first list) (firstn (1- n) (rest list))))))

(defun max-car-firstn (n list)
 (append (max-car-first (firstn n list)) (nthcdr n list)))

(defun matrix-inverse (M)
 (do ((M1 (max-car-first (augment M))
       (cond ((null M1) nil)
            (t (max-car-firstn n (cycle-left (cycle-up M1))))))
    (n (1- (length M)) (1- n)))
   ((or (minusp n) (null M1)) (cond ((null M1) nil) (t (first-square M1))))
   (setq M1 (cond ((zerop (caar M1)) nil) (t (solve-first-column M1))))))

(defun max-car-first (L)  ;L is a list of lists. This function finds list with
 (cond ((null (cdr L)) L) ;largest car and moves it to head of list of lists.
     (t (if (> (abs (caar L)) (abs (caar (max-car-first (cdr L)))) L
          (append (max-car-first (cdr L)) (list (car L)))))))

(defun mdh-matrix (length costwist sintwist translate cosrotate sinrotate)
 (list (list cosrotate (- sinrotate) 0. length)
     (list (* sinrotate costwist) (* cosrotate costwist) (- sintwist)
        (- (* translate sintwist)))
     (list (* sinrotate sintwist) (* cosrotate sintwist) costwist
        (* translate costwist))
     (list 0. 0. 0. 1.)))

(defun homogeneous-transform (azimuth elevation roll x y z)
 (rotation-and-translation (sin azimuth) (cos azimuth) (sin elevation)
 (cos elevation) (sin roll) (cos roll) x y z))
```

```
(defun rotation-and-translation (spsi cpsi sth cth sphi cphi x y z)
 (list (list (* cpsi cth) (- (* cpsi sth sphi) (* spsi cphi))
         (+ (* cpsi sth cphi) (* spsi sphi)) x)
      (list (* spsi cth) (+ (* cpsi cphi) (* spsi sth sphi))
         (- (* spsi sth cphi) (* cpsi sphi)) y)
      (list (- sth) (* cth sphi) (* cth cphi) z)
      (list 0. 0. 0. 1.)))

(defun inverse-H (H)        ;H is a 4x4 homogeneous transformation matrix.
 (let* ((minus-P (list (- (fourth (first H)))
                  (- (fourth (second H)))
                  (- (fourth (third H)))))
       (inverse-R (transpose (first-square (reverse (rest (reverse H))))))
       (inverse-P (post-multiply inverse-R minus-P)))
      (append (concat-matrix inverse-R (transpose (list inverse-P)))
           (list (list 0 0 0 1)))))

(defun law_cosines (adjside1 adjside2 opposite)
  (setf angle (acos (/(+(sqr adjside1)(sqr adjside2)(-(sqr opposite)))
                (* 2 adjside1 adjside2))))
  (format t "Angle is ~a ~%" angle))

(defun euclid_dist (coord1 coord2)
  (setf x1 (first coord1))
  (setf y1 (second coord1))
  (setf z1 (third coord1))
  (setf x2 (first coord2))
  (setf y2 (second coord2))
  (setf z2 (third coord2))
  (setf hyp (sqrt(+(sqr(- x1 x2))(sqr(- y1 y2))(sqr(- z1 z2)))))
  (format t "Hypotenuse length is ~a ~%" hyp))

(defun sqr (x) (* x x))
```

CAMERA.CL

```
;*********************************************************************
; File: camera.cl                          Franz Common LISP
;
; ** CAMERA CLASS DEFINITION **
; A Camera "takes a picture" of rigid-body class objects
; and displays the image.  A sequence of images may be
; displayed by superimposing them or by first erasing the display
; window and then creating and displaying the next image.
;
; Requires: rigid-body.cl
;
; by Shirley Isakari  CS4314 Winter 1994  Final Project
; Modifications & enhancements to Prof. McGhee's Strobe-Camera CLOS code
;*********************************************************************

(require :xcw)

(use-package :cw)  ; Note that this is required for use of mouse and color.
                   ; This forced renaming of some original functions, i.e.
                   ; move and translate. Causes some problem when compiling.

(cw:initialize-common-windows)


(defclass camera (rigid-body)
 ((focal-length
   :accessor focal-length
   :initform 6)
  (posture
   :accessor posture   ; azim elev roll x y z
   :initform (list 0 0 0 -300 0 0))
  (camera-window
   :accessor camera-window
   :initform (cw:make-window-stream :borders 5
                                    :left 300
                                    :bottom 300
                                    :width 300
                                    :height 400
                                    :title "Right Arm Articulation"
                                    :background-color blue
                                    :foreground-color white
                                    :activate-p t))
  (H-matrix
   :initform (homogeneous-transform 0 0 0 -300 0 0))
  (inverse-H-matrix
   :accessor inverse-H-matrix
   :initform (inverse-H (homogeneous-transform 0 0 0 -300 0 0)))
  (enlargement-factor
   :accessor enlargement-factor
   :initform 900)))

(defun create-camera-1 ()
 (setf camera-1 (make-instance 'camera))
 (queue-mouse camera-1))
```

```
(defmethod queue-mouse ((camera camera))
  (cw:modify-window-stream-method (camera-window camera) :left-button-down
     :after 'mouse-handler)
  (cw:modify-window-stream-method (camera-window camera) :middle-button-
down
     :after 'mouse-handler)
  (cw:modify-window-stream-method (camera-window camera) :right-button-down
     :after 'mouse-handler))

; Note that mouse-handler requires names of instantiated objects:
; camera-1 jack-1. Unable to modify argument list of this event-handler.
(defun mouse-handler (wstream cw:mouse-state &optional event)
  ;(format t "In mouse-handler button: ~a~%" (mouse-button-state))
  (cond ((eql (cw:mouse-button-state) 128)       ; Left-click
         (rotate-camera camera-1 -10)
       ;  (format t "Mouse Event: Left-click => rotate-camera~%")
        )
        ((eql (cw:mouse-button-state) 129)       ; Left-click & CNTRL key
          (rotate-camera camera-1 10)
       ;   (format t "Mouse Event: CNTRL+Left-click => rotate-camera~%")
        )
        ((eql (cw:mouse-button-state) 64)            ; Middle-click
          (zoom-camera camera-1 10)
       ;  (format t "Mouse Event: Middle-click => zoom-camera~%")
        )
        ((eql (cw:mouse-button-state) 65)        ; Middle-click & CNTRL key
          (zoom-camera camera-1 -10)
       ;  (format t "Mouse Event: CNTRL+Middle-click => zoom-camera~%")
        )
        ((eql (cw:mouse-button-state) 32)            ; Right-click
          (tilt-camera camera-1 -10)
       ;   (format t "Mouse Event: Right-click => tilt-camera~%")
        )
        ((eql (cw:mouse-button-state) 33)        ; Right-click & CNTRL key
          (tilt-camera camera-1 10)
       ;    (format t "Mouse Event: CNTRL+Right-click => tilt-camera~%")
        )
        (t nil))
  (new-picture camera-1 jack-1 jack-color))

; *** Defined global color constants ************************************
; To be used as the draw-color argument in take-picture and new-picture
; functionss (and also jack-picture, jack-video, jack-movie functions)

(defconstant *white* 0)
(defconstant *yellow* 1)
(defconstant *red* 2)
(defconstant *green* 3)
(defconstant *black* 4)
(defconstant *cyan* 5)
(defconstant *magenta* 6)
(defconstant *blue* 7)

; *** Draw picture functions ************************************************
(defmethod take-picture ((camera camera) (body rigid-body) draw-color)
  (let ((camera-space-node-list (mapcar #'(lambda (node-location)
       (post-multiply (inverse-H-matrix camera) node-location))
              (transformed-node-list body))))
    (dolist (polygon (polygon-list body))
      (clip-and-draw-polygon camera polygon camera-space-node-list draw-color))))
```

```
(defmethod erase-camera-window ((camera camera))
  (cw:clear (camera-window camera)))
(defmethod new-picture ((camera camera) (body rigid-body) draw-color)
   (erase-camera-window camera)
   (take-picture camera body draw-color))

(defmethod clip-and-draw-polygon
  ((camera camera) polygon node-coord-list draw-color)
  (do* ((initial-point (nth (first polygon) node-coord-list))
        (from-point initial-point to-point)
        (remaining-nodes (rest polygon) (rest remaining-nodes))
        (to-point (nth (first remaining-nodes) node-coord-list)
                  (if (not (null (first remaining-nodes)))
                      (nth (first remaining-nodes) node-coord-list))))
       ((null to-point)
            (draw-clipped-projection camera from-point initial-point draw-color))
       (draw-clipped-projection camera from-point to-point draw-color)))

(defmethod draw-clipped-projection ((camera camera)
       from-point to-point draw-color)
  (cond ((and (<= (first from-point) (focal-length camera))
              (<= (first to-point) (focal-length camera))) nil)
        ((<= (first from-point) (focal-length camera))
         (draw-line-in-window camera
           (perspective-transform camera (from-clip camera from-point to-point))
(perspective-transform camera to-point) draw-color))
        ((<= (first to-point) (focal-length camera))
         (draw-line-in-window camera
           (perspective-transform camera from-point)
           (perspective-transform camera (to-clip camera from-point to-point))
            draw-color))
        (t (draw-line-in-window camera
           (perspective-transform camera from-point)
           (perspective-transform camera to-point) draw-color))))

(defmethod from-clip ((camera camera) from-point to-point)
  (let ((scale-factor (/ (- (focal-length camera) (first from-point))
                         (- (first to-point) (first from-point)))))
       (list (+ (first from-point)
                (* scale-factor (- (first to-point) (first from-point))))
             (+ (second from-point)
               (* scale-factor (- (second to-point) (second from-point))))
             (+ (third from-point)
               (* scale-factor (- (third to-point) (third from-point)))) 1)))

(defmethod to-clip ((camera camera) from-point to-point)
  (from-clip camera to-point from-point))
```

```
(defmethod draw-line-in-window ((camera camera) start end draw-color)
  (cond ((= 0 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color white))
        ((= 1 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color yellow))
        ((= 2 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color magenta))
        ((= 3 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color green))
        ((= 4 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color red))
        ((= 5 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color cyan))
        ((= 6 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color black))
        ((= 7 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color blue))))

(defmethod perspective-transform ((camera camera) point-in-camera-space)
  (let* ((enlargement-factor (enlargement-factor camera))
         (focal-length (focal-length camera))
         (x (first point-in-camera-space))  ;x axis is along optical axis
         (y (second point-in-camera-space)) ;y is out right side of camera
         (z (third point-in-camera-space))) ;z is out bottom of camera
        (list (+ (round (* enlargement-factor (/ (* focal-length y) x)))
                 150)                ;to right in camera window
              (+ 150 (round (* enlargement-factor (/ (* focal-length (- z)) x))
                            )))))  ;up in camera window

; *** Position camera functions ****************************************
(defmethod move-camera ((camera camera) azimuth elevation roll x y z)
  (setf (H-matrix camera) (homogeneous-transform azimuth elevation roll x y z))
  (setf (inverse-H-matrix camera) (inverse-H (H-matrix camera)))
  (format t "camera: ~a " (posture camera)) )

(defmethod zoom-camera ((camera camera) zoom-amount)
  (setf (slot-value camera 'enlargement-factor)
        (+ (slot-value camera 'enlargement-factor) zoom-amount)))
```

```
; Rotation in x-y plane about origin
(defmethod rotate-camera ((camera camera) angle-increment) ; in degrees
 (let* ((new-position (posture camera))
        (radius (sqrt (+ (* (fourth new-position) (fourth new-position))
                         (* (fifth new-position) (fifth new-position)))))
        (heading (atan (fourth new-position)
          (fifth new-position)))
        (angle (deg-to-rad angle-increment))
        (new-heading (+ heading angle)))
        (setf (first new-position) (- (first new-position) angle)
            (fourth new-position) (* radius (sin new-heading))
            (fifth new-position) (* radius (cos new-heading))
            (posture camera) new-position
            (H-matrix camera) (homogeneous-transform (first new-position)
            (second new-position) (third new-position) (fourth new-position)
(fifth new-position) (sixth new-position))
            (inverse-H-matrix camera) (inverse-H (H-matrix camera)))))

; Vertical tilting about origin in a plane perpendicular to x-y plane
; Max tilt (90 or -90 deg) when top or bottom view of x-y plane is achieved
(defmethod tilt-camera ((camera camera) angle-increment) ; in degrees
 (let* ((new-position (posture camera))
        (radius (sqrt (+ (* (fourth new-position) (fourth new-position))
                         (* (fifth new-position) (fifth new-position))
                         (* (sixth new-position) (sixth new-position)))))
        (tilt (atan (sixth new-position)
            (sqrt (+  (* (fourth new-position) (fourth new-position))
            (* (fifth new-position) (fifth new-position))))))
        (heading (atan (fourth new-position)
            (fifth new-position)))
        (angle (deg-to-rad angle-increment))
        (new-tilt (cond ((< (abs (+ tilt angle)) tilt-limit) (+ tilt angle))
                  (t (cond ((minusp (+ tilt angle)) (* -1 tilt-limit))
                     (t tilt-limit))))))
      (setf (second new-position) new-tilt
        (fourth new-position)
          (cond ((= (abs tilt) (abs new-tilt) tilt-limit)
            (fourth new-position))
            (t (* radius (sin heading) (cos new-tilt))))
        (fifth new-position)
          (cond ((= (abs tilt) (abs new-tilt) tilt-limit)
            (fifth new-position))
(t (* radius (cos heading) (cos new-tilt))))
        (sixth new-position)
          (cond ((= (abs tilt) (abs new-tilt) tilt-limit)
            (sixth new-position))
            (t (* radius (sin new-tilt))))
        (posture camera) new-position
        (H-matrix camera) (homogeneous-transform (first new-position)
        (second new-position) (third new-position) (fourth new-position)
        (fifth new-position) (sixth new-position))
        (inverse-H-matrix camera) (inverse-H (H-matrix camera)))))

(defun deg-to-rad (angle) (* .017453292519943295 angle))
(defconstant tilt-limit (deg-to-rad 89.9))
```

```
; *** Auxiliary functions ************************************
(defun kill ()
  (cw:kill-common-windows))

(defun reset-windows ()
  (kill)
  (cw:initialize-common-windows))
```

VIDEO-CAMERA.CL

```
;*****************************************************************
;
; File: video-camera.cl                    Franz Common LISP
;
; ** VIDEO-CAMERA CLASS DEFINITION **
; A Video-camera is a camera which uses double buffering in order to
; display a sequence of images without flicker.
;
; by Shirley Isakari  CS4314 Winter 1994  Final Project
;
; Requires: camera.cl
;
; by Shirley Isakari  CS4314 Winter 1994  Final Project
; Adapted from Prof. Kwak's Movie Camera flavor.
; *****************************************************************
;

(defclass video-camera (camera)
  ((image-window
   :accessor image-window
   :initform (cw:make-bitmap-stream :borders 5
                    :width 300
                    :height 400
                    :title "Right Arm Articulation"
                    :background-color blue
                    :foreground-color white
                    :activate-p nil))))

(defun create-video-camera-1 ()
  (setf camera-1 (make-instance 'video-camera))
  (queue-mouse camera-1))

(defmethod new-picture ((camera video-camera) (body rigid-body) draw-color)
  (erase-image-window camera)
  (take-picture camera body draw-color)
  (expose-image camera))
```

```
(defmethod draw-line-in-window ((camera video-camera) start end draw-color)
  (cond ((= 0 draw-color) (cw:draw-line (image-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color white))
        ((= 1 draw-color) (cw:draw-line (image-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color yellow))
        ((= 2 draw-color) (cw:draw-line (image-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color red))
        ((= 3 draw-color) (cw:draw-line (image-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color green))
        ((= 4 draw-color) (cw:draw-line (image-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color black))
        ((= 5 draw-color) (cw:draw-line (image-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color cyan))
        ((= 6 draw-color) (cw:draw-line (image-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color magenta))
        ((= 7 draw-color) (cw:draw-line (image-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color blue))))

(defmethod expose-image ((camera video-camera))
  (cw:bitblt (image-window camera) 0 0 (camera-window camera) 0 0))

(defmethod erase-image-window ((camera video-camera))
  (cw:clear (image-window camera)))
```

APPENDIX B. EDITED LISP EXECUTION SCRIPT

```
Script started on Wed Aug 30 23:02:26 1995

;; Copyright Franz Inc., Berkeley, CA, USA
;; Unpublished.  All rights reserved under the copyright laws
;; of the United States.


user(1): (load "load-files.cl")
  (load-jack)
(jack-video  *black*)    ;  Loading  /workd/waldrop/cs4314.dir/polhemus.dir/load-
files.cl.
t
user(2): ; Loading /workd/waldrop/cs4314.dir/polhemus.dir/rigid-body.cl.
; Loading /workd/waldrop/cs4314.dir/polhemus.dir/link.cl.
; Loading /workd/waldrop/cs4314.dir/polhemus.dir/camera.cl.
; Loading /workd/waldrop/cs4314.dir/polhemus.dir/video-camera.cl.
; Loading /workd/waldrop/cs4314.dir/polhemus.dir/kinematics.cl.
; Loading /workd/waldrop/cs4314.dir/polhemus.dir/human.cl.
; Loading /workd/waldrop/cs4314.dir/polhemus.dir/human-arm.cl.
; Loading /workd/waldrop/cs4314.dir/polhemus.dir/jack-link.cl.
; Loading /workd/waldrop/cs4314.dir/polhemus.dir/polhemus.cl.
; Loading /workd/waldrop/cs4314.dir/polhemus.dir/demo-jack.cl.
t
user(3): (new-picture camera-1 jack-1 0)
         (zoom-camera camera-1 10)
         (tilt-camera camera-1 -30)
         (rotate-camera camera-1 180)
         (new-picture camera-1 jack-1 0)

user(9): (move-incremental jack-1 '((0 0 0 0 0 0) (0.1 0.5)))

A-matrix for #<link1> is : ((0.9950042 -0.09983342 0 0)
                  (0.09983342 0.9950042 0.0 0.0)
                  (0.0 0.0 1.0 0.0) (0 0 0 1))
H-matrix for #<link0> is : ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0)
                  (0.0 0.0 1.0 0.0)(0.0 0.0 0.0 1.0))
H-matrix for #<link1> is : ((0.9950042 -0.09983342 0.0 0.0)
                  (0.09983342 0.9950042 0.0 0.0)
```

(0.0 0.0 1.0 0.0) (0.0 0.0 0.0 1.0))

joint angle is 0.1
A-matrix for #<link2> is : ((0.4794255 0.8775826 0 0)
(3.8360354e-8 -2.0956352e-8 -1.0 0.0)
(-0.8775826 0.4794255 -4.371139e-8 0.0)
(0 0 0 1))
H-matrix for #<link1> is : ((0.9950042 -0.09983342 0.0 0.0)
(0.09983342 0.9950042 0.0 0.0)
(0.0 0.0 1.0 0.0)
(0.0 0.0 0.0 1.0))
H-matrix for #<link2> is : ((0.47703037 0.8731984 0.09983342 0.0)
(0.047862723 0.087612055 -0.9950042 0.0)
(-0.8775826 0.4794255 -4.371139e-8 0.0)
(0.0 0.0 0.0 1.0))

joint angle is -1.0707964

user(11): (H-to-jack (sensor (link2 (upperarm (rightarm jack-1)))) jack-1)

The sensor is on #<link2>
H-matrix for #<link2> : ((0.47703037 0.8731984 0.09983342 0.0)
(0.047862723 0.087612055 -0.9950042 0.0)
(-0.8775826 0.4794255 -4.371139e-8 0.0)
(0.0 0.0 0.0 1.0))
H-matrix for the sensor : ((0.47703037 0.8731984 0.09983342 0.0)
(0.047862723 0.087612055 -0.9950042 0.0)
(-0.8775826 0.4794255 -4.371139e-8 0.0)
(0.0 0.0 0.0 1.0))
inverse body H matrix = ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0)
(0.0 0.0 1.0 0.0) (0 0 0 1))
T matrix = ((0.47703037 0.8731984 0.09983342 0.0)
(0.047862723 0.087612055 -0.9950042 0.0)
(-0.8775826 0.4794255 -4.371139e-8 0.0)
(0.0 0.0 0.0 1.0))
c1 = 0.09983342
c2 = -0.9950042
a3 = -0.8775826
b3 = 0.4794255
joint angle for link1 = 0.1
joint angle for link2 = -1.0707964

ser(12): (move-incremental jack-1 '((0 0 0 0 0 0) (0.2 0.6)))

A-matrix for #<link0> is : ((1.0 0.0 0 0) (0.0 1.0 0.0 0.0)
(0.0 0.0 1.0 0.0) (0 0 0 1))
H-matrix for #<static-human> is : ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0)
(0.0 0.0 1.0 0.0)(0.0 0.0 0.0 1.0))
H-matrix for #<link0> is : ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0)
(0.0 0.0 1.0 0.0) (0.0 0.0 0.0 1.0))

joint angle is 0.0
A-matrix for #<link1> is : ((0.9553365 -0.29552022 0 0)
(0.29552022 0.9553365 0.0 0.0)
(0.0 0.0 1.0 0.0) (0 0 0 1))
H-matrix for #<link0> is : ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0)
(0.0 0.0 1.0 0.0)(0.0 0.0 0.0 1.0))
H-matrix for #<link1> is : ((0.9553365 -0.29552022 0.0 0.0)
(0.29552022 0.9553365 0.0 0.0)
(0.0 0.0 1.0 0.0) (0.0 0.0 0.0 1.0))

joint angle is 0.3

A-matrix for #<link2> is : ((0.89120734 0.45359614 0 0)
(1.9827317e-8 -3.895591e-8 -1.0 0.0)
(-0.45359614 0.89120734 -4.371139e-8 0.0)
(0 0 0 1))
H-matrix for #<link1> is : ((0.9553365 -0.29552022 0.0 0.0)
(0.29552022 0.9553365 0.0 0.0)
(0.0 0.0 1.0 0.0) (0.0 0.0 0.0 1.0))
H-matrix for #<link2> is : ((0.8514029 0.43333697 0.29552022 0.0)
(0.26336983 0.13404681 -0.9553365 0.0)
(-0.45359614 0.89120734 -4.371139e-8 0.0)
(0.0 0.0 0.0 1.0))

joint angle is -0.47079635

user(14): (H-to-jack (sensor (link2 (upperarm (rightarm jack-1)))) jack-1)

The sensor is on #<link2>
H-matrix for #<link2> : ((0.8514029 0.43333697 0.29552022 0.0)
(0.26336983 0.13404681 -0.9553365 0.0)
(-0.45359614 0.89120734 -4.371139e-8 0.0)
(0.0 0.0 0.0 1.0))
H-matrix for the sensor : ((0.8514029 0.43333697 0.29552022 0.0)
(0.26336983 0.13404681 -0.9553365 0.0)

(-0.45359614 0.89120734 -4.371139e-8 0.0)
(0.0 0.0 0.0 1.0))

inverse body H matrix = ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0)
(0.0 0.0 1.0 0.0) (0 0 0 1))
T matrix = ((0.8514029 0.43333697 0.29552022 0.0)
(0.26336983 0.13404681 -0.9553365 0.0)
(-0.45359614 0.89120734 -4.371139e-8 0.0)
(0.0 0.0 0.0 1.0))
c1 = 0.29552022
c2 = -0.9553365
a3 = -0.45359614
b3 = 0.89120734
joint angle for link1 = 0.3
joint angle for link2 = -0.47079635

user(15): (move-incremental jack-1 '((0 0 0 0 0 0) (0.3 0.7)))

A-matrix for #<link0> is : ((1.0 0.0 0 0) (0.0 1.0 0.0 0.0)
(0.0 0.0 1.0 0.0) (0 0 0 1))
H-matrix for #<static-human> is : ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0)
(0.0 0.0 1.0 0.0)(0.0 0.0 0.0 1.0))
H-matrix for #<link0> is : ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0)
(0.0 0.0 1.0 0.0)(0.0 0.0 0.0 1.0))
joint angle is 0.0
A-matrix for #<link1> is : ((0.8253356 -0.5646425 0 0)
(0.5646425 0.8253356 0.0 0.0)
(0.0 0.0 1.0 0.0) (0 0 0 1))
H-matrix for #<link0> is : ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0)
(0.0 0.0 1.0 0.0) (0.0 0.0 0.0 1.0))
H-matrix for #<link1> is : ((0.8253356 -0.5646425 0.0 0.0)
(0.5646425 0.8253356 0.0 0.0)
(0.0 0.0 1.0 0.0) (0.0 0.0 0.0 1.0))

joint angle is 0.6
A-matrix for #<link2> is : ((0.9738476 -0.22720206 0 0)
(-9.931317e-9 -4.256823e-8 -1.0 0.0)
(0.22720206 0.9738476 -4.371139e-8 0.0)
(0 0 0 1))
H-matrix for #<link1> is : ((0.8253356 -0.5646425 0.0 0.0)
(0.5646425 0.8253356 0.0 0.0)
(0.0 0.0 1.0 0.0) (0.0 0.0 0.0 1.0))
H-matrix for #<link2> is : ((0.8037511 -0.18751793 0.5646425 0.0)

(0.54987574 -0.12828797 -0.8253356 0.0)
(0.22720206 0.9738476 -4.371139e-8 0.0)
(0.0 0.0 0.0 1.0))
joint angle is 0.22920364

user(17): (H-to-jack (sensor (link2 (upperarm (rightarm jack-1)))) jack-1)

The sensor is on #<link2>
H-matrix for #<link2> : ((0.8037511 -0.18751793 0.5646425 0.0)
(0.54987574 -0.12828797 -0.8253356 0.0)
(0.22720206 0.9738476 -4.371139e-8 0.0)
(0.0 0.0 0.0 1.0))
H-matrix for the sensor : ((0.8037511 -0.18751793 0.5646425 0.0)
(0.54987574 -0.12828797 -0.8253356 0.0)
(0.22720206 0.9738476 -4.371139e-8 0.0)
(0.0 0.0 0.0 1.0))
inverse body H matrix = ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0)
(0.0 0.0 1.0 0.0) (0 0 0 1))
T matrix = ((0.8037511 -0.18751793 0.5646425 0.0)
(0.54987574 -0.12828797 -0.8253356 0.0)
(0.22720206 0.9738476 -4.371139e-8 0.0) (0.0 0.0 0.0 1.0))
c1 = 0.5646425
c2 = -0.8253356
a3 = 0.22720206
b3 = 0.9738476
joint angle for link1 = 0.6
joint angle for link2 = 0.22920364

user(18): (exit)
; killing "Default Window Stream Event Handler"
; killing "X11 event dispatcher"
; killing "Initial Lisp Listener"
; Exiting Lisp
[1;7mlike:/workd/waldrop/cs4314.dir/polhemus.dir>>exit
script done on Wed Aug 30 23:06:36 1995

LIST OF REFERENCES

ASCE95 Ascension Technology, http://www.oz.is/OZ/Misc/Ascension.html, Internet.

BADL93a Badler, N. I., Phillips, C. B. and Webber, B. L., *Simulating Humans: Computer Graphics Animation and Control,* Oxford University Press, New York, 1993.

BADL93b Badler, N. I., Hollick, M. J. and Granieri, J. P., "Real-Time Control of a Virtual Human Using Minimal Sensors," *Presence: Teleoperators and Virtual Environments*, Winter 1993, Volume 2, Number 1, MIT, pp. 82-86.

BARH94 Barham, P. T., Pratt, D. R., Zyda, M. J., Locke, J. and Falby, J., "NPS-NET-IV: A DIS-Compatible Object Oriented Software Architecture for Virtual Environments," Naval Postgraduate School, Monterey, California, October 1994.

BLAN90 Blanchard, C., Burgess, S., Harvill, Y., Lanier, J., Lasko, A., Oberman, M. and Teitel, M., "Reality Built for Two: A Virtual Reality Tool," *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, March, 1990, Volume 24, Number 2, pp. 35-36.

CRAI89 Craig, J., *Introduction to Robotics: Mechanics and Control*, Second Edition, Addison-Wesley Publishing Company, Inc., Menlo Park, California, 1989.

DAVI93 Davidson, S., *An Experimental Comparison of CLOS and C++ Implementations of an Object-Oriented Graphical Simulation of Robot Kinematics*, Master's Thesis, Naval Postgraduate School, Monterey, California, March, 1993.

DURL95 Durlach, N. I. and Mavor, A. S., National Research Council, *Virtual Reality: Scientific and Technological Challenges*, National Academy Press, Washington, D.C., 1995, pp. 188-204, 306-317.

FU87 Fu, K. S., Gonzalez, R. C. and Lee, C. S. G., *Robotics: Control, Sensing, Vision and Intelligence*, McGraw-Hill Book Company, New York, 1987.

GRAN95 Granieri, J. P. and Badler, N. I.,"Simulating Humans in VR," *Applications of Virtual Reality*, Academic Press, 1995. To appear.

MCIO15 Marine Corps Institute Order, P1500.44B, *Battleskills Training/Essential Subjects Handbook*, The United States Marine Corps.

MEYE92 Meyer, K., Applewhite, H. L. and Biocca, F. A., "A Survey of Position Trackers," *Presence: Teleoperators and Virtual Environments*, Spring, 1992, Volume 1, Number 2, MIT, pp. 173-200.

POLH93 Polhemus, *3Space Fastrak User's Manual Revision F*, OPM3609-002C, November 1993.

PRAT94 Pratt, D. R., Barham, P. T., Locke, J., Zyda, M. J., Eastman, B., Moore, T., Biggers, K., Douglass, R., Jacobsen, S., Hollick, M., Granieri, J., Ko, H. and Badler, N. I., "Insertion of an Articulated Human into a Networked Virtual Environment," *Proceedings of the Fifth Annual Conference on AI, Simulation and Planning in High Autonomy Systems: Distributed Interactive Simulation Environments*, IEEE Computer Society Press, Gainesville, Florida, December 7-9,1994, pp. 84-90.

PRAT95 Pratt, S. M., Pratt, D. R., Waldrop, M. S., Barham, P. T., Ehlert, J. F. and Chrislip, C. A., "Humans in a Large-Scale, Real Time, Networked Virtual Environments," submitted to *Presence*, 1995.

SCHI90 Schilling, R. J., *Fundamentals of Robotics: Analysis and Control*, Prentice Hall, Englewood Cliffs, New Jersey, 1990.

WATT92 Watt, A. and Watt, M., *Advanced Animation and Rendering Techniques: Theory and Practice*, Addison-Wesley Publishing Company, Inc., New York, 1992, pp. 369-394.

ZYDA92 Zyda, M. J., Pratt, D. R., Monahan, J. G. and Wilson, K. P., "NPSNET: Constructing a 3D Virtual World," *1992 Proceedings of Symposium on Interactive 3D Graphics*, pp. 147-156.

ZYDA95 Zyda, M. J., Image of IPORT Soldier Station.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Library, Code 013 — 2
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Director, Training and Education — 1
   MCCDC, Code C46
   1019 Elliot Rd.
   Quantico, VA 22134-5027

4. Chairman, Code CS — 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

5. Dr. Robert B. McGhee, Code CS/Mz — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

6. Mr. John S. Falby, Code CS/Fa — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

7. Ms. Shirley M. Pratt, Code CS — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

8. Dr. Michael Zyda, Code CS/Zk — 10
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

9. Dr. David R. Pratt, Code CS/Pr 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

10. Mr. Paul T. Barham, Code CS/Barham 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

11. Dr. Donald Brutzman, Code UW/Br 1
Undersea Warfare Department
Naval Postgraduate School
Monterey, CA 93943

12. Captain Marianne S. Waldrop, USMC 2
103 Sand Castle Drive
Emerald Isle, NC 28594