

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI[®]

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

**Development of Animated Finger Movements via a Neural Network for
Tendon Tension Control**

by

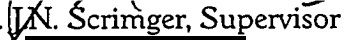
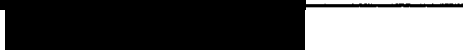
Olga Anna Kuchar

A Thesis Submitted to the
Faculty of Computer Science
in Partial Fulfilment of the Requirements
for the Degree of


DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

APPROVED: 

Dr.  Scrimger, Supervisor 

Dr. D. Riordan 

Dr. M. El-Hawary 

Dr. T. Calvert, Technical University of British Columbia, External Examiner

DALHOUSIE UNIVERSITY - DALTECH

Halifax, Nova Scotia

©

1999



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*

Our file *Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-39322-4

Canada

DALTECH LIBRARY

"AUTHORITY TO DISTRIBUTE MANUSCRIPT THESIS"

TITLE:

Development of Animated Finger Movements via a Neural Network for Tendon
Tension Control

The above library may make available or authorize another library to make available
individual photo/microfilm copies of this thesis without restrictions.

Full Name of Author: Olga Anna Kuchar

Signature of Author:

Date: 04/12/1999

Keep in touch with your dreams...

Table of Contents

	Page
List of Tables	viii
List of Figures	ix
List of Abbreviations	xi
Glossary	xii
Acknowledgments	xiv
Abstract	xv
1. Introduction	1
1.1 Problem Statement	1
1.2 Goal	2
1.3 Organization	3
1.4 Summary	3
2. Background	4
2.1 Computer Animation	4
2.1.1 Geometric Methods	5
2.1.1.1 Performance Animation and Motion Capture	5
2.1.1.2 Keyframing	7
2.1.1.3 Inverse Kinematics	8
2.1.1.4 Procedural Animation	8
2.1.2 Physical Methods	8
2.1.2.1 Dynamic Analysis	9
2.1.3 Behavioural Methods	9
2.1.3.1 Task-level Animation	10
2.1.3.2 Artificial and Virtual Life	10
2.1.4 Computer Animation Summary	11

	Page
4.4.4 Muscles in the Thumb	42
4.4.4.1 Extrinsic Muscles	42
4.4.4.1.1 Abductor Pollicis Longus	42
4.4.4.1.2 Extensor Pollicis Brevis	43
4.4.4.1.3 Extensor Pollicis Longus	43
4.4.4.1.4 Flexor Pollicis Longus	44
4.4.4.2 Intrinsic Muscles	44
4.4.4.2.1 Flexor Pollicis Brevis	45
4.4.4.2.2 Abductor Pollicis Brevis	45
4.4.4.2.3 Adductor Pollicis	45
4.4.4.2.4 Opponens Pollicis	46
4.5 Mathematical Model	46
4.5.1 The Basic Finger Model	46
4.5.2 The FDP Mathematical Model	47
4.6 Computer Implementation	52
4.6.1 The Input File	52
4.6.2 Start-up Screen	53
4.6.3 Mouse Movements and Rotations	54
4.6.4 The Sliders	54
4.7 Chapter Summary	55
5. Artificial Neural Networks Controlling Tendons in an Animated Hand	56
5.1 Hierarchical Hand Movement Control Scheme	56
5.2 Hand Movements	57
5.2.1 Classification of Hand Movements	57
5.2.2 Databases	61
5.3 Artificial Neural Networks in this Research	62
5.3.1 Artificial Neural Network Architecture	62
5.3.2 Error Back-propagation Training Algorithm	64
5.4 Artificial Hand Neural Network	65
5.4.1 Finger Flexion and Extension Neural Network	66
5.4.1.1 FFENN Architecture	66
5.4.1.2 Input-Output Data	67
5.4.1.3 Training the FFENN	68
5.4.1.4 FFENN Results	69
5.4.1.5 Test Data Performance	69
5.4.1.6 Discussion and Future Direction	71

	Page
5.4.2 Finger Abduction and Adduction Neural Network	71
5.4.2.1 FAANN Architecture	71
5.4.2.2 Input-Output Data	72
5.4.2.3 Training the FAANN	72
5.4.2.4 FAANN Results	72
5.4.2.5 Test Data Performance	73
5.4.2.6 Discussion and Future Direction	73
5.4.3 Summary of the AHNN	75
5.5 Computer Implementation	75
5.5.1 Modified Hand-Muscle Programme	75
5.5.2 Signing Programme	77
5.6 Chapter Summary	78
6. Conclusions and Future Work	79
7. References	83
8. Appendices	89
Appendix A	90
Appendix B	97
Appendix C	106

List of Tables

	Page
Table 4-1. Summary of Modelled Muscles and their Functions	32
Table 4-2. A Sample Input File	52
Table 5-1. 25 Cases of Finger Flexion and Extension	59
Table 5-2. 2 Cases of Finger Abduction and Adduction	61
Table 5-3. Excerpt from the Finger Commands Database	62
Table 5-4. Excerpt from the Finger Tendon-Length Database	62

List of Figures

	Page
Figure 2-1. A Biological Neuron	13
Figure 2-2. Computer Model of a Biological Neuron	14
Figure 2-3. A Taxonomy of Network Architectures	15
Figure 2-4. Architecture for using a Neural Network	18
Figure 4-1. The Carpal Hand	26
Figure 4-2. A Hand Skeleton	27
Figure 4-3. Range of Flexion in the PIP Joints	29
Figure 4-4. Range of Flexion in the DIP Joints	30
Figure 4-5. Sagittal Plane	31
Figure 4-6. Muscle Model for the Flexor Digitorum Sublimis	35
Figure 4-7. The Flexor Digitorum Sublimis	36
Figure 4-8. Muscle Model for the Flexor Digitorum Profundus	36
Figure 4-9. The Flexor Digitorum Profundus	37
Figure 4-10. The Extensor Apparatus	38
Figure 4-11. The Extensor Expansion	39
Figure 4-12. The Interosseous	41
Figure 4-13. The Muscle Model for the Abductor Pollicis Longus	43
Figure 4-14. The Muscle Model for the Extensor Pollicis Brevis	44
Figure 4-15. The Muscle Model for the Abductor Pollicis Brevis	45
Figure 4-16. The Finger Model	46
Figure 4-17. DIP Joint is Flexed	48
Figure 4-18. PIP Joint is Flexed	49
Figure 4-19. MCP Joint is Flexed	50
Figure 4-20. Muscle Program	53
Figure 5-1. Finger Flexion and Extension Neural Network	67
Figure 5-2. Performance Error of FEENN	70
Figure 5-3. Finger Abduction and Adduction Neural Network	72
Figure 5-4. Performance Error of FAANN	74

List of Figures (continued..)

	Page
Figure 5-5. Artificial Hand Neural Network	75
Figure 5-6. Modified Hand-muscle Programme	76
Figure 5-7. Sign Language Programme	77

List of Abbreviations

ADM	Abductor Digiti Minimi
AHNN	Artificial Hand Neural Network
AIVH	Artificially Intelligent Virtual Human
ANN	artificial neural network
AP	Adductor Pollicis
APB	Abductor Pollicis Brevis
APL	Abductor Pollicis Longus
CMC	carpometacarpal joint
DI	dorsal interossei
DIP	distal interphalangeal joint
DOF	degrees of freedom
EDC	Extensor Digitorum Communis
EMG	electromyographic
EPB	Extensor Pollicis Brevis
EPL	Extensor Pollicis Longus
FAANN	Finger Abduction and Adduction Neural Network
FDM	Flexor Digiti Minimi
FDP	Flexor Digitorum Profundus
FDS	Flexor Digitorum Sublimis
FFENN	Finger Flexion and Extension Neural Network
FPB	Flexor Pollicis Brevis
FPL	Flexor Pollicis Longus
IP	interphalangeal joint
MCP	metacarpophalangeal joint
PI	palmar interossei
PIP	proximal interphalangeal joint

Glossary

Abductor Pollicis Brevis (APB) The muscle that causes adduction of the thumb and flexes the MCP joint.

Abductor Pollicis Longus (APL) The muscle that causes abduction and flexion of the first metacarpal bone.

Adductor Pollicis (AP) A muscle that causes adduction of the thumb.

Aponeurosis Any one of the thicker and denser of the deep fasciae that cover, invest, and terminate and attach many muscles. They often differ from tendons only in being flat and thin.

Articulated Figure An articulated figure is a structure that consists of a series of rigid links connected at rotary joints.

Degrees of Freedom (DOF) The number of degrees of freedom of an articulated structure is the number of independent position variables necessary to specify the state of a structure.

Distal Interphalangeal Joint (DIP) The joint located between the distal and proximal phalanges in the fingers.

End Effector Most industrial manipulations are so-called open chains and the free end of such a chain of links is called the end effector. In a human model, an end effector may be the hand or a foot.

Epicondyle A projection on the inner side of the distal end of the humerus.

Extensor Digitorum Communis (EDC) The muscle that allows a finger to extend.

Extensor Pollicis Brevis (EPB) A thumb muscle that produces extension at the proximal joint and is the true abductor of the thumb.

Extensor Pollicis Longus (EPL) A thumb muscle that extends the proximal phalanx.

Glossary (continued...)

Flexor Digiti Minimi (FDM) A flexor muscle located in the pinky.

Flexor Digitorum Profundus (FDP) Primarily a muscle that produces flexion of the DIP joint in the fingers.

Flexor Digitorum Sublimis (FDS) Primarily a muscle that produces flexion of the PIP joint in the fingers.

Flexor Pollicis Brevis (FPB) Primarily a muscle that produces flexion and adduction of the thumb.

Flexor Pollicis Longus (FPL) Primarily a muscle that produces flexion at the IP joint in the thumb.

Interphalangeal Joint (IP) A term that relates to the DIP and PIP joints.

Metacarpophalangeal Joint (MCP) The joint between the proximal phalanx and the metacarpal bone in the fingers.

Proximal Interphalangeal Joint (PIP) The joint located between the middle and proximal phalanges in the fingers.

Acknowledgments

First and foremost, this thesis was funded by the Natural Sciences and Engineering Research Council (NSERC) and by the Izaak Walton Killam Memorial Scholarship. Other support was received from the Faculty of Computer Science, Dalhousie University - DalTech.

I would like to extend my deepest gratitude to my supervisor, Dr. Norm Scrimger, for his continuous support and guidance through my Ph.D. program. I would also like to thank my committee for taking the time to read my thesis and for providing helpful comments.

I would like to especially thank Dr. Hamdullahpur, Dean of Graduate Studies, for his constant support in my research and for helping me to attend conferences to present my work.

I would like to thank my parents for their love and encouragement.

I thank God for helping me to attain one of my life's dreams.

Abstract

Adding virtual humans into virtual reality environments requires expensive hardware and, for the animator, time-consuming tasks. An ideal situation in populating virtual worlds would be to create an artificially intelligent virtual human (AIVH) that is capable of moving and interacting in its environment with little intervention from the animator. This research focuses on the preliminary developments of creating such an AIVH. Before an AIVH can move completely, an AIVH needs to learn how to move its body. Thus, the body needs to be divided into separate areas and the AIVH needs to learn about movements in these areas.

This thesis focuses on free hand motion, and more specifically on thumb and finger postures. The author incorporates the use of feedforward artificial neural networks (ANNs) to manipulate an underlying model in real time and determine postures of the thumb and fingers to allow for dynamic hand animation. The underlying model of the computerized human hand is based on a biological model involving tendons. The ANNs trigger any tendons that need to be manipulated to achieve an animated goal. The prescribed animated goals involve different flexion, extension, abduction, and adduction movements. There are several drawbacks associated with using a multilayer feedforward network. First, since the architecture is designed by trial-and-error, it is very difficult to create an effective 'a priori' architecture. Secondly, ANNs are not trained on an entire input space, but there is a supposition that the ANN will work correctly when presented with any possible input within this problem space.

In this ongoing research, the ability for ANNs to determine tendon control in the thumb and fingers has been developed and tested, producing positive results. Now that ANNs are seen to be successfully applied to hands, the next step is to apply this technique to other parts of the body.

1. Introduction

*The beginning is the most important
part of the work.
- Plato*

An ideal virtual world would be filled with people, animals, nature, and many other objects that we encounter in our real world. Real actors can enter a virtual world and can suspend reality for a few moments since they immerse themselves completely into that world. In order for a person to feel that they are immersed in a different reality, the person must not suspect that the virtual actors are computer generated. A virtual actor must produce the desired motion effect, whether this is in movement, speech, or behaviour. Thus, the main goal of computer animation is to synthesize the desired motion effect by mixing natural phenomena, perception, and imagination. An animator needs different tools to create a virtual world populated with many objects. An animator designs an object's dynamic behaviour based on the animator's interpretation of reality within the simulated environment. Thus, an animation system has to provide an animator with motion control tools such that the animator can create their virtual world easily. Computer animation methods may also help to understand physical laws by adding motion control to data in order to show their evolution over time.

In the next generation of animation systems, motion control will tend to be performed automatically using artificial intelligence, and motion will be planned at a task level and computed using physical laws.

1.1 Problem Statement

Films such as *Rendezvous à Montréal* (Thalmann et al. [1987]) were created from the desire to use human beings as synthetic actors in 3D computer animation. The difference between film and virtual reality is the dynamic behaviour required for the synthetic or virtual human. Cavazza et al. [1998] indicated some current research directions in virtual humans and their control. Some of these directions include: generating and controlling virtual humans in real time; creating characters with individuality and personality who react to and interact with other real or virtual people; and

enabling virtual humans to perform complex tasks. Controlling body motion involves animating a skeleton that consists of connected segments corresponding to limbs and joints. Animators control the skeleton locally and define it in terms of coordinates, angles, velocities, and accelerations. Motion control techniques from computer animation can be applied to virtual humans. Magnenat-Thalmann et al. [1996] proposed such a classification scheme based on three motion control methods: geometric; physical; and behavioural. Geometric methods correspond to methods driven by geometric data, and some of these techniques are performance animation, keyframing, and inverse kinematics. Physical methods control an object's motion by describing the behaviour of the object in terms of the interaction of internal and external forces, and one of these techniques is dynamic analysis. In dynamic analysis, an animator calculates the forces and torques acting upon masses to predict motion. Dynamic analysis has two main disadvantages: it is necessary to first determine the torques and forces required to produce a particular human motion; and the amount of computer time required to calculate human motion is based on the complexity of the model. Behavioural methods drive motion based on relationships between objects. A problem with this approach is the need to define the behavioural model completely for all types of interaction between objects. Reynolds [1987] uses a distributed behavioural model simulating interaction within flocks of birds, herds of land animals, and schools of fish.

Several of the techniques in these methods are time consuming and not acceptable when applying them to artificially intelligent virtual humans (AIVHs). A method is needed to create motion in real time such that a virtual human can react to its environment dynamically. In this thesis we propose the application of artificial neural networks to allow control of animated human thumb and fingers. A biological model uses tendon tensions to control hand movements; however, in this research, artificial neural networks (ANNs) manipulate tendon lengths to determine hand postures.

1.2 Goal

This thesis develops a first step in creating one aspect of an AIVH. Animating human movement has usually been divided into animation of three separate body sections: the face; the hands; and the rest of the body. This thesis focuses on determining hand postures in real time to allow for dynamic human hand animation. Since a

complex model of the thumb and fingers of an animated hand was developed based on tendon control (Kuchar [1996]), back-propagation neural networks were created to learn how to manipulate the tendons to achieve thumb and finger postures. A database of finger end-positions was created on which to train and test the neural networks. Once the networks had been trained, they were incorporated into an animation system for visual verification of finger postures, since humans are better at detecting problems visually. The neural networks were able to manipulate the tendons successfully to allow for real-time motion.

1.3 Organization

This thesis is organized into several chapters. Chapter 2 summarizes some of the important information required for this thesis: motion control in articulated figure animation; ANNs; and muscle control. Chapter 3 provides the rationale for this thesis and its application within VR. Chapter 4 describes the underlying tendon model and the corresponding computer implementation that is used in this thesis. Chapter 5 describes the creation of back-propagation neural networks to manipulate an animated human hand based on tendon control, including the corresponding computer implementation. Chapter 6 completes this thesis with conclusions and future work.

1.4 Summary

Virtual human modelling must include the structure needed for virtual human animation. When linking modelling and animation, human figure animation is complex because the human body is capable of making vast and innumerable combinations of shape positions and interacting with its environment in many different ways. A computer model for a synthetic character could emulate a human by creating ANNs that are trained to accomplish a movement. The hypothesis of this research is to create ANNs to control tendons in an animated human hand. The hand model developed is based on muscle control for the thumb and fingers of a human hand. Using this model, back-propagation neural networks were successfully developed and trained to manipulate thumb and finger postures. Since ANNs were applied to an intricate part of the body, such as hands, then in theory this technique can be applied to the entire body. Thus, ANNs may be an important aspect in the future creation of AIVHs.

2. Background

*Study the past if you would
divine the future.
- Confucius*

Adding virtual humans into virtual reality environments requires expensive hardware and, for the animator, time-consuming tasks. An ideal situation in populating virtual worlds would be to create an artificially intelligent virtual human (AIVH) that is capable of moving and interacting in its environment with little intervention from the animator. As a first step in understanding the development of an AIVH, background research in several areas must be considered: computer animation; artificial neural networks; and muscle control. This chapter gives a brief overview of these areas as related to this thesis.

2.1 Computer Animation

Zeltzer [1985] classifies animation systems as being either:

- *guiding systems*: behaviour of animated objects is explicitly described.
- *animator-level systems*: behaviour of animated objects is algorithmically specified.
- *task-level systems*: behaviour of animated objects is specified in terms of events and relationships.

Guiding and animator-level systems are described in detail in Thalmann [1990]. General purpose task-level systems are not available now, but HUMANOID (Boulic et al. [1995]) and JACK (Badler et al. [1993]) are steps towards task-level animation.

This thesis research is based on the application of a new motion control method for animating human hands; thus when involving synthetic actors, a classification of computer animation scenes according to motion control methods would be more appropriate. Magnenat-Thalmann et al. [1991] proposed such a classification

scheme based on three motion control methods: geometric; physical; and behavioural. Each of these categories is described in the following sections.

2.1.1 Geometric Methods

Geometric methods correspond to methods driven by geometric data. Objects are locally controlled. For example, if an animated object is a human finger and an animator is using a geometric method, then the finger's motion is described at every joint. The animator provides detailed geometric data corresponding to a local definition of the motion. Motion may be defined in terms of coordinates, angles, and other shape characteristics, or it may be specified using velocities and accelerations, but no forces are involved. A detailed description of geometric methods can be found in Delaney [1998] and Magnenat-Thalmann et al. [1996]. The following section describes briefly some geometric techniques, such as performance animation, keyframing, inverse kinematics, and procedural animation.

2.1.1.1 Performance Animation and Motion Capture

Performance animation or motion capture consists of measuring and recording motions and applying those motions to animate different characters. For example, the people strolling on the sun deck in the 1998 movie *Titanic* were created in part using motion capture techniques (Delaney [1998]). Traditional cell animation requires the animator to animate different characters frame-by-frame for a particular scene. Motion capture helps to lessen the rigors of traditional cell animation, and thus reduces the time needed for animation development.

There are three kinds of motion capture systems: mechanical; magnetic; and optical. Mechanical systems or digital puppetry require the use of one or more real-time input devices, such as a mouse, a joystick, or datagloves, to be able to animate 3D characters. The information provided by these devices is used to control the variation of parameters over time for every animated feature of the character.

Magnetic motion capture systems require a real actor to wear a set of receivers. One or more transmitters, located in the room with the actors, generate three orthogonal electromagnetic fields each. When the receivers pick up the signals, the computer determines the distance from the transmitters by the time elapsed and the orientation

of the receiver by the changes in the signal caused by the tilting of the magnetic fields. The position and orientation of each receiver is then used to drive an animated character. One difficulty is the need for synchronizing receivers. As noted in Magnenat-Thalmann et al. [1996], human body motion generally requires eleven receivers: one on the head; one on each upper arm; one on each hand; one in the center of the chest; one on the lower back; one on each ankle; and one on each foot. Inverse kinematics is used to calculate the remainder of the information. The two most popular magnetic systems are Polhemus Fastrak® (Polhemus [1997]) and Ascension Flock of Birds® (Ascension Technology Corporation [1998]). The benefits of using magnetic motion capture systems can be attained by animators who do not require high accuracy in motion, since the motion data can be applied to animated characters in real time. For example, if we use data gathered from the eleven receivers indicated for human body motion, the resultant coarse motion may be acceptable for an animated human that is far in the background of an animated scene since small peculiarities cannot be easily detected; however, if this data was applied to an animated human whose hand motion was forming an American Sign Language letter, then obviously the motion data provided would not be of sufficient resolution. These systems are fast at providing data to use in animation at the cost of accuracy in motion. The disadvantage of using magnetic motion capture systems for hand movements is that a hand has a small surface area and a vast amount of dexterity. Depending on the size of the receivers placed on each finger, a real hand would become cumbersome to move, and full dexterity would not be accomplished.

Optical motion capture systems are based on cameras visually tracking small reflective markers attached to real performers at key points. Freedom of movement is the main advantage of this method over most magnetic systems that are hardwired with wires attached from the real actors to a host computer. For motion with lots of action, wires limit the range of motion and, if more than one real actor is being tracked (such as a dancing couple), the wires could become entangled. The two main disadvantages of optical motion capture systems are occlusion and inter-reflection. First, occlusion occurs when there is missing data resulting from hidden markers. For example, an animator requires motion data for a human to get out of bed. Some of the markers on a real actor will be placed on the actor's back, but when an actor lies on

his/her back the optical motion capture system cannot track this sensor since it is blocked by the actor. Secondly, when markers are too close to each other during motion, inter-reflection causes a problem for optical motion capturing systems. For example, when two real actors are dancing together in a tango, some markers between the two actors will be very close to each other and the capturing system will not be able to distinguish which marker is on which actor. These problems may be minimized by adding more cameras, but result in higher production costs since more equipment and sophisticated software are required. Generally, most optical systems operate with four or six cameras, as is exemplified by the VICON® system (Vicon [1998]).

Overall, motion capture systems provide good motion data that comes directly from reality; however, for every new motion, it is necessary to record that motion data again, directly from reality. Humans produce a wide variety of movements, and to store motion data for all possible movements would be intractable. Motion capture is not appropriate in real-time simulations since the accuracy needed for motion data would require greater sampling and processing time for a computer, and a contemporary computer would not be able to derive the necessary motion for these simulations. Even for a small, but dextrous part of a human body like a hand, capturing all possible hand movements would be impractical, if not impossible.

2.1.1.2 Keyframing

Keyframing is still one of the most common interactive motion control methods. The method for keyframing involves an animator specifying a sequence of positions and the times when they occur, and a computer interpolates between these positions to produce the animation. The advantage to keyframing is that an animator can see the total configuration of the system at certain given times. The disadvantages are that keyframing is a low level control method that requires an animator to specifically control the motion of each degree of freedom, and does not allow easy visualization of the motion between keyframes.

2.1.1.3 Inverse Kinematics

Inverse kinematics requires an animator to define only the position of the end effectors, such as a hand or leg. The skeleton of an animated human is defined as a hierarchical structure in which each level is affected from above and responds to feedback from levels below. Inverse kinematics determines the position and the orientation of all joints in the hierarchy that lead to the end effector. In inverse kinematics, as the number of limbs in an articulated figure increases, the problem of finding a solution for a given position of the end effector becomes undefined. To avoid this problem, animators use constraints, such as energy minimization and momentum conservation, to limit the number of possible solutions. Inverse kinematics, or goal-directed motion, is used for such tasks as animating human walking since an animator may require an animated human to move within a scene to attain a goal. A more intense study of goal-directed motion is described in Korein et al. [1982].

2.1.1.4 Procedural Animation

Procedural animation corresponds to the generation of motion using a series of preprogrammed instructions (procedures) with minimal user input concerning actual motion. The user may suggest parameters controlling the algorithm, but does not control the specific motion of individual degrees of freedom. Simple algorithmic control is mainly applied to such rigid objects as spinning cubes and planets following elliptical orbits, rather than human motion. Procedural animation is used when a motion is well-defined and can be incorporated into an algorithm. For example, animating human walking may involve: lift left foot; swing left leg; place left foot. The Extensible Director-Oriented Systems MIRANIM (Magnenat-Thalman et al. [1990]) is an example of procedural animation in an interactive system.

2.1.2 Physical Methods

Physical methods control an object's motion by describing the behaviour of the object in terms of the interaction of internal and external forces. For example, a description of the shape of a rubber ball bouncing on a wooden floor is obtained by considering the gravitational effect on the rubber ball and the interaction between the ball

and the floor. The following section describes briefly one physical method – dynamic analysis.

2.1.2.1 Dynamic Analysis

An approach to animating human motion is dynamic analysis; it gives greater physical reality to the animation by simulating the effects of forces and torques acting upon masses to predict motion. Human motion is governed by forces and torques applied to the limbs. Some of these forces are generated by muscles or by interaction of objects within the environment. An example of dynamics in animating articulated figures are crash studies that include dynamic simulations involving animated humans. In these studies, the environmental effects of the forces and torques being applied to the crash dummy during a crash test are simulated.

The dynamics equations for articulated bodies are complex. In order to provide greater realism, each segment of the articulated body would be described by dynamics equations. Since the connected segments interact with each other, dynamics equations are coupled and must be solved as a system of simultaneous equations, one equation for each degree of freedom. As an animated human model becomes more complex, a dynamic analysis system must solve a larger number of simultaneous equations. Some forces and torques for certain types of complex motion, such as falling, striking the floor, or bouncing against other objects, can be automatically calculated and realistically rendered with no user input. When forces and torques cannot be automatically calculated, an animator must explicitly describe them in an animation. This is a problem since producing controlled, coordinated motion described in terms of force and torque control is nonintuitive. Dynamic analysis has two main disadvantages: it is necessary to first determine the torques and forces required to produce a particular human motion; and the amount of computer time required to calculate human motion is based on the complexity of the model. A more detailed study of dynamic analysis is given in Wilhelms [1987].

2.1.3 Behavioural Methods

Behavioural methods drive motion based on relationships between objects. Behavioural motion control methods drive the behaviour of objects by providing high-

level directives indicating a specific behaviour without any other stimulus. For example, if an animated human hand is required to make a fist, the command *FIST* would be an appropriate high-level directive to indicate the required behaviour. The following section describes briefly two examples of behavioural methods: task-level animation; and artificial life.

2.1.3.1 Task-level Animation

In task-level animation, an action is specified only by its effects on objects. Task-level commands are translated into low-level instructions, such as a script for algorithmic animation. For example, when an animated hand is given the command *FIST*, this command would be translated into a script that closes all the fingers.

Task-level animation has been used for animating grasping hands. Magnenat-Thalmann et al. [1988] describe a system that facilitates the task of animating human interaction with an environment; however, an animator has to position a hand and determine the contact points between the hand and the object to be grasped. A method for producing an automatic grasp was developed by Rijpkema et al. [1991]. These authors give a full description of a grasping system and approximate objects in an environment with simple primitives. A knowledge database is constructed that contains the mechanisms to grasp all known primitives in the environment. Recently, Mas et al. [1994] have presented a hand control and automatic grasping system that can decide to use a pinch when the object is too small to be grasped by more than two fingers, or to use a two-handed grasp when the object is too large to be grasped by one hand. The disadvantage of task-level animation applied to virtual animated humans is the lack of dynamic interaction with an environment.

2.1.3.2 Artificial and Virtual Life

Research from artificial intelligence has been applied to behavioural animation. Reynolds [1987] simulated flocks of birds, herds of land animals, and schools of fish using a distributed behavioural model. For example, flocking behaviour in birds consists of two opposing factors: a desire for the birds to stay close to the flock; and a desire for the birds to avoid collisions within the flock. This flocking behaviour was implemented as a set of rules with decreasing precedence:

1. *collision avoidance*: avoid collisions with nearby flockmates.
2. *velocity matching*: attempt to match the velocity of nearby flockmates.
3. *flock centering*: attempt to stay close to nearby flockmates.

The simulated flock is based on particle systems (Hearn et al. [1994]), with the birds simulated as particles. The animator can control global parameters by adjusting the "importance" or weights of the rules mentioned above, and can change the maximum velocity, maximum acceleration, and minimum distance between the birds.

One of the problems of using artificial intelligence for animated humans is that the rules or equations of motion cannot be explicitly defined since the equations are complex and cannot be determined absolutely. For example, when a human rises from a chair, many muscles coordinate in the back, legs, and other areas to create this motion. To define explicitly all of these relationships and dependencies amongst the muscles cannot be done since researchers in biomechanics and physiology still have not determined all coordination of movements.

2.1.4 Computer Animation Summary

Motion control for animating humans has been a main research area in computer graphics. A wide variety of techniques are used and combined to produce required animated movements. As human models increase in complexity, new motion control algorithms will be created to attain realistic human movement. The goal of realistic human movement is hard to achieve in animation, but the goal of realistic real-time animated human simulations is even harder to achieve and more work is required in this area.

2.2 Artificial Intelligence and Neural Networks

Artificial intelligence is a research area in computer science where scientists try to emulate human cognitive skills through the design and implementation of computer programs. Within the area of artificial intelligence, artificial neural networks (ANNs) provide an alternative approach to many problems, such as pattern recognition, optimization, and control. As noted in section 2.1.3, artificial intelligence has

been applied in behavioural methods. For example, artificial neural networks that guide lower-level motor skills have been proposed by Ridsdale [1990] to determine animated handball serves. The main focus of this thesis is on applying ANNs to animating human finger movements. In order to understand the work accomplished in this thesis, an overview of ANNs is provided in this section, including a biological neuron, an artificial neuron, and ANN architectures.

2.2.1 Biological Neural Networks

The human brain is a complex non-linear parallel computer. It has the capability of organizing neurons to process certain tasks. A neuron, or nerve cell, is depicted in Figure 2-1 and is the fundamental cellular unit of the brain's nervous system. The function of the neuron is to process information. Neurons are constructed from the same basic parts, independent of their size and shape. These are the soma, the dendrites, and the axon. The soma is the cell body; it has a nucleus that contains information about hereditary traits and a plasma that holds the molecular equipment for producing material needed by the neuron. A neuron receives and combines signals from other neurons through input paths called dendrites and transmits signals along its output path called the axon. Neurons do not perform identically since different neurons can give different responses. Each neuron has an activation function that dictates whether the neuron is excited or inhibited. The axon branches and connects to other dendrites of other neurons through synapses. Synapses are junctions that contain neurotransmitter fluid. This fluid mitigates the flow of electrical signals. The strength or conductance of the synaptic junction is modified as the brain learns. The synapse's effectiveness can be adjusted by the signals passing through it so that the synapses can learn from the activities in which they participate. This dependence on history acts as a memory that may be responsible for human memory.

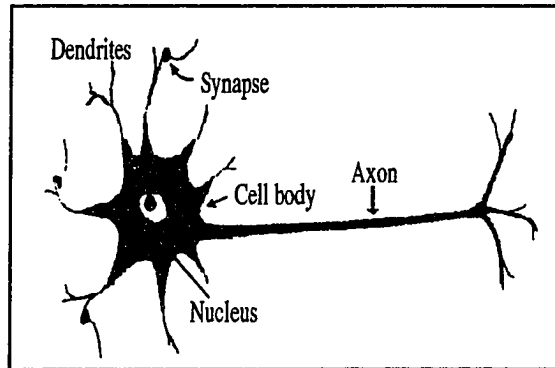


Figure 2-1. A biological neuron. Adapted from Jain et al. [1996].

2.2.2 ANNs Definition

The computer model of a biological neural network takes the form of an ANN that contains artificial neurons, and is depicted in Figure 2-2. An artificial neural system models the neural network and contains algorithms for cognitive tasks, such as learning and optimization, modelled in mathematical terms. Müller et al. [1995] describe an ANN model in mathematical terms as a directed graph with the following properties:

1. A state variable n_i is associated with each node i .
2. A real-valued weight w_{ik} is associated with each link (ik) between two nodes i and k .
3. A real-valued bias v_i is associated with each node i .
4. A transfer function $f_i[n_k, w_{ik}, v_k, (k \neq i)]$ is defined for each node i , that determines the state of the node as a function of its bias, of the weights of its incoming links, and of the states of the nodes connected to it by these links.

To associate the biological neuron with the artificial neuron, the nodes of the ANN are called neurons, the links are called synapses, and the bias is known as the activation threshold. The transfer function is either a discontinuous step function or a sigmoidal function. Nodes without links toward them are called input neurons; nodes without links leading away from them are called output neurons. A neuron can have many inputs, analogous to the dendrites, and can combine the values of the inputs. The output of the neuron can branch to many other neurons, analogous to the axon. The out-

put is then modified by connection weights that correspond to the synaptic strength of the neural connections before being input to the receiving neuron.

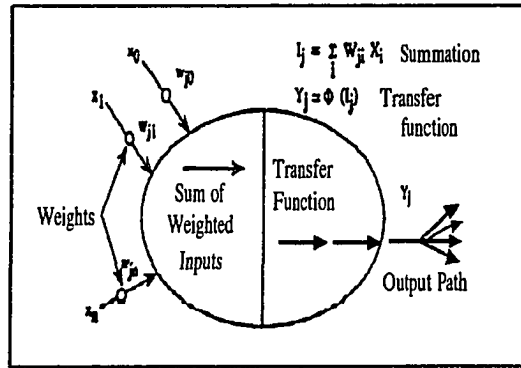


Figure 2-2. Computer model of a biological neuron. Adapted from Uhrig [1995].

2.2.3 Neural Network Architectures

An ANN consists of many nodes joined with an adjustable weighting function for each input. The nodes are usually organized into a series of layers that may contain full or random connections between them. There are at least three layers in an ANN: an input layer; one or more hidden layers; and an output layer.

There exist many different types of ANNs, as depicted in Figure 2-3. ANNs can be grouped into two categories based on their connection pattern: feedforward neural networks; and recurrent or feedback neural networks. A feedforward network is a graph with no loops whereas a recurrent neural network is a graph with loops.

Different connectivities result in different network behaviours. Feedforward networks are static for they only produce one set of output values, and are thus memory-less for they are not dependent on the state of the network. Recurrent neural networks are dynamic since they have loops due to the feedback connections, and thus the inputs to each neuron are modified leading the network to enter a new state.

One of the major advantages of ANNs is their ability to learn. The operation of an ANN involves the two processes of learning and recall. Learning is the process of adapting connection weights in response to input values. The neural network "learns" based on a learning rule that governs how the connection weights are modi-

fied as a result of the input values. Recall is the process of taking the input and producing a response based on what the network has learned.

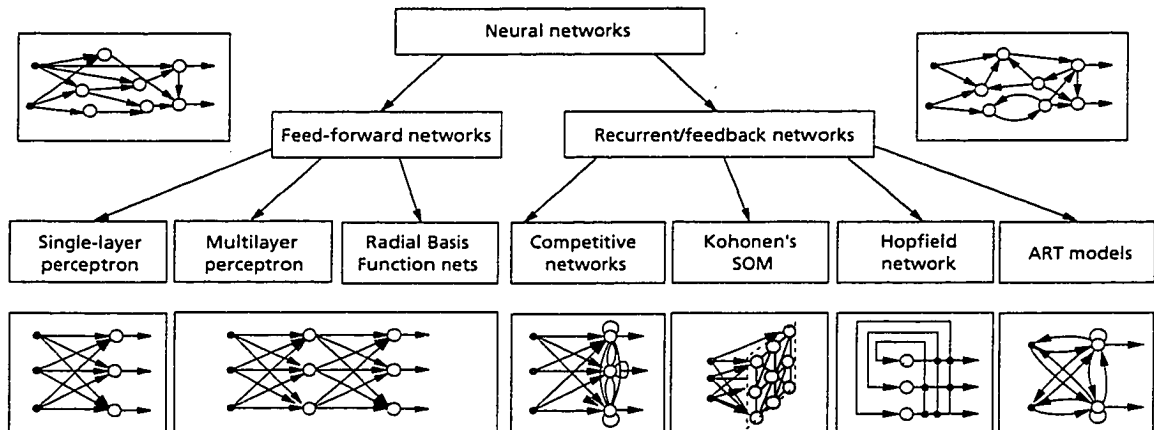


Figure 2-3. A taxonomy of feedforward and recurrent network architectures. Adapted from Jain et al. [1996].

There are three main learning paradigms: supervised; unsupervised; and hybrid. Supervised learning is the most common and an ANN is provided with a correct output for every input pattern. It uses a corrective algorithm to convert the difference between the output patterns of the neural network and the desired output into an adjustment of the connection weights. Unsupervised learning tries to deduce correlations between patterns in the data and then organizes these correlations into categories without providing information about the actual categories. Hybrid learning combines the two learning algorithms, where part of the weights is determined through supervised learning.

The common learning algorithms are: Hebbian learning; Delta-rule learning; and competitive learning. The Hebbian learning algorithm increments weights if both the input and the desired output are high; it is analogous to the biological process in

that a neural pathway is strengthened each time it is used. The Delta-rule learning algorithm modifies the weights based on the difference between the desired output and actual output by minimizing this difference using a least-squares process. The competitive learning algorithm modifies the weights of the neuron that gives the strongest response to a given input by calculating the difference between the input values and these weights. The final value of the weights constitutes the "memory" of the network.

2.2.4 Why Neural Networks?

ANNs are powerful because they are distributed in a parallel structure, and they can learn and generalize based on their learning algorithms. Their ability to generalize makes it possible for ANNs to solve complex problems. Some other features of ANNs are:

- *Nonlinearity:* An ANN is a non-linear, distributed network. Nonlinearity is an important property, for the underlying physical mechanism responsible for the generation of an input signal is inherently non-linear.
- *Input-Output Mapping:* As noted from section 2.2.3, an ANN learns the relationships between the input values and the output values.
- *Adaptivity:* As noted from section 2.2.3, an ANN can adapt its weights based on the learning algorithm, and these weights can be modified in real time. An ANN trained to operate in a specific environment can be easily retrained to deal with minor changes in the operating environmental conditions.
- *Evidential Response:* An ANN can be designed to provide information about the confidence in a decision that it made regarding a selected pattern.
- *Contextual Information:* Every neuron in an ANN can be affected by the global activity of all other neurons in the network, and thus contextual information is dealt with naturally.

Fault Tolerance: An important advantage of ANNs is their high degree of error resistivity. A normal computer may completely fail in its operation if only a single bit of stored information or a single programme statement is incorrect. In contrast, the operation of an ANN often remains almost unaffected if a single neuron fails, or if a few synaptic connections collapse.

2.2.5 Further Readings

Section 2.2 provided an overview of both biological and artificial neural networks, architectures, and learning algorithms. For further detailed information, the reader is directed to the following textbooks on ANNs: Müller et al. [1995]; Haykin [1994]; and Zurada [1992].

2.3 ANNs and Muscle Control

Another aspect of this thesis is applying tendon control to animated human fingers, with ANNs manipulating the tendons. Only recently have ANNs been applied to predict muscle activity. As stated by Nussbaum et al. [1997], accurate prediction of muscle activity, and an understanding of the systems responsible for coordinating these activities, remain open problems. Work accomplished by Nussbaum et al. [1995] and Sepulveda et al. [1993] required a set of known electromyographic (EMG) patterns to be able to generalize and predict muscle activity and apply ANNs to derive a mapping function from an external moment to a set of muscle activations given an existing database. The database of EMG patterns did not exist and thus had to be created by the authors; however, Nussbaum et al. [1997] presented a model that could be developed without a specific database by having elements representing specific muscle groups placed 'within' the model, and their behaviour emerged through a training process requiring moment equilibrium. Also, new elements representing competitive (inhibitory) interactions among the muscles were incorporated, their values determined by comparing model output with a pre-existing EMG data set, and are hypothesized to reflect interactions that occur within a motor control system. These elements are nodes in a back-propagation neural network model used by Nussbaum et al. [1997]. As depicted in Figure 2-4, this architecture included an ANN that would re-

ceive as input the magnitudes of applied right lateral, left lateral, flexion, and extension moments. The eight output nodes correspond to four pairs of muscles and the ANN would predict normalized muscle activity for each muscle. The ANN architecture was composed of a multilayer, fully connected, feedforward network and the authors used error back-propagation for training the ANN. The training algorithm used a gradient descent technique to traverse the error surface by moving downhill in the steepest direction. The training varied from 20,000 to 44,000 cycles, depending on the number of hidden nodes incorporated into the ANN. This model was compared to two optimization-based muscle force prediction models. The first optimization model (Bean et al. [1988]) is a two-step linear programme that minimizes first the maximum muscle intensity and second the spinal compressive force. The second optimization model (Hughes [1991] and Crowninshield et al. [1981]) is a nonlinear programme that minimizes the sum of the cubes of muscle force intensities. Once trained, the ANN model predictions were better correlated with observed data than the two optimization-based models.

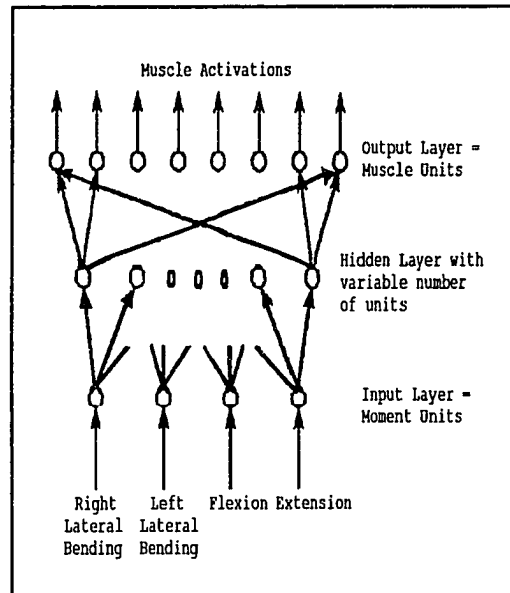


Figure 2-4. Architecture for using a neural network to predict lumbar muscle activity. Adapted from Nussbaum et al. [1997].

2.4 Chapter Summary

This background chapter summarizes some of the important information required for this thesis: motion control in articulated figure animation; ANNs; and muscle control. A wide variety of motion control techniques are described in section 2.1. These techniques are used and combined to produce required animated movements. As human models increase in complexity, new motion control algorithms will be created to attain realistic human movement. Section 2.2 provides an overview of ANNs. ANNs are an alternative approach to solving many problems and may be able to provide a new approach in animation motion control. Section 2.3 provides an overview of how ANNs have been applied for muscle activity, indicating that an ANN could learn to predict muscle activity for certain movements. These are the three main areas that apply to this thesis research, and are combined to create a new approach that is described in the next chapter.

3. Thesis Scope and Motivation

*The only limit to our realization of tomorrow
will be our doubts of today.
- Franklin D. Roosevelt*

Many advances in virtual reality (VR) have occurred during the past ten years, in different research areas and on focused topics, such as multiresolution techniques for displaying millions of polygons (Foley et al. [1990]), and the use of robotics hardware as force-feedback interfaces (Rosenblum et al. [1997]). Within these different research areas, one of the focused topics deals with creating and populating virtual worlds. Chapter 2 provided an overview of animation techniques and artificial neural networks; two disciplines that have the potential to create artificially intelligent virtual humans. This chapter describes the focused area of virtual human motion control addressed by this thesis, and its place and importance in VR research.

3.1 Virtual Reality: A Hierarchy

VR incorporates difficult research problems involving many disciplines since VR requires synthesizing numerous techniques in both software and hardware development. Sometimes, the next advance in VR depends on progress by non-VR researchers, such as a new robotics device or a new natural language technique. This thesis research lies within the scope of the first aspect – software development.

Researchers have explored new models that are required in a virtual world to realistically emulate a broad variety of living and non-living things, such as rivers, houses, plants, animals, and humans. Typically, these models inhabiting the virtual worlds are modelled using physical law-based modelling techniques. As VR hardware develops, graphics researchers create animated worlds that are filled with objects that exhibit greater complexity than is typically accessible through physical modelling alone – objects that are *alive* (animated plants, animals, and humans). Artificial life transcends the traditional boundaries of engineering, computer science, and biological science, and the natural synergy between computer graphics and artificial life can be potentially beneficial to both disciplines. Animated living objects must simulate many

of the natural processes that uniquely characterize living systems, such as birth and death, growth, natural selection, evolution, perception, locomotion, manipulation, adaptive behaviour, intelligence, and learning. The challenge in graphics research is to develop sophisticated graphics models that are self-replicating, self-evolving, self-controlling, and self-animating, by simulating the natural mechanisms fundamental to life.

There are several areas in computer graphics that have started to incorporate artificial life phenomena, such as:

- *artificial plants*: Modelling the response of plants to its environment, including light, nutrients, pruning, and mechanical obstacles.
- *artificial animals*: Modelling involves a physics-based model of an animal in its world, the animal's use of physics for locomotion, and its ability to link perception to action through adaptive behaviour.
- *interactive synthetic characters*: Modelling full-body interaction between human participants and graphical worlds inhabited by artificial life forms that people find interesting. These life forms have their own goals, and can sense and interpret the actions of real actors and respond to them in real time.
- *artificial life of virtual humans*: Modelling human beings that include perception, and tactile and auditory sensors. These sensors provide information to support human behaviour, such as visually directed locomotion, manipulation of objects, and response to sounds and utterances. This area also includes communication between virtual humans, behaviour of crowds of virtual humans, and communication between real and virtual humans.

Each of the above areas contain complex research problems in both animation and VR. The author is specifically interested in the last area – artificial life of virtual humans.

Animated human beings require a complex model that contains human senses, behaviour, communication, and movement. Each of these is a major research topic,

and thus the scope of this thesis was narrowed to modelling human movement. Modelling and deformation of 3D human bodies during the animation process is an important but difficult problem. Researchers have devoted significant effort to the representation and deformation of the human body shape, resulting in two body model categories: the surface model; and the multi-layered model. The surface model proposed by Thalmann et al. [1987] contains a skeleton and outer skin layer. The skin layer is composed of planar or curved patches. There are two main problems with this model. First, this model requires a detailed surface mesh to approximate skin. Secondly, in this model it is hard to control the realistic deformations of the surface across joints. Surface singularities or anomalies can easily be produced, but simple observation of human skin in motion reveals that the deformation of the skin results from many other factors besides the skeleton configuration, such as muscle and fat tissue. The multi-layered model proposed by Chadwick et al. [1989] contains a skeleton layer, intermediate layers, and a skin layer. The intermediate layers may consist of muscle, fat tissue, and other physical attributes since deformation of the skin layer is influenced by such structures of the intermediate layer. The key advantage of the layered model is that once the layered character is constructed, only the underlying skeleton needs to be scripted for an animation and the skin deformations are generated automatically.

This thesis is concerned with the underlying skeleton that needs to be manipulated to allow for real-time human movement. Improving motion realism requires many degrees of freedom (DOF) in the body linkages and that increases the difficulty of control. The variability and complexity of human movement requires complex adaptive motion generation algorithms for animating such movement. Even though algorithms have addressed greater animation power with kinematics, dynamics, inverse kinematics, available torque, locomotion, gestural and directional control, the human models themselves tended to be rather simplified versions of a real human. Increased realism in the human models would demand more accurate and complicated motion control. A virtual character must move convincingly enough for a real actor to believe that they are interacting with another real human in VR. Virtual characters need to respond *dynamically* to their environment, and facilitating interesting and engaging responses that reflect the way real humans behave, in real time, remains a major challenge in the animation field.

3.2 Human Hand Movement

As indicated in the previous section, this thesis is concerned with manipulating an underlying skeleton to allow for real-time human movement. Animating human movement has usually been divided into animation of three separate body sections: the head; the hands; and the rest of the body. This is mainly due to the differences in modelling and animation applied to these different sections.

Hands represent a very small part of the whole body, but they are the most flexible part of the body and they are essential in our dealings with the environment. Hand positioning varies greatly and allows the hand to compose many different movements. Hands are capable of conforming to the shape of objects to be grasped or studied, and of emphasizing an idea being expressed. This is accomplished because of the unique structure of human hands, consisting of 19 bones, 17 articulations, and 19 muscles situated entirely within the hand, and about the same number of tendons activated by the forearm muscles. Each hand contains a sum of many relative DOF, and the hand's importance and particularities require a dedicated model. The hand is the most difficult part of the human body to model because of its complex structure, and therefore the hand requires a complex motion algorithm. As noted in Kuchar [1996], a complex model of the thumb and fingers of a human hand includes the bones, motor muscles, and tendons. Kuchar [1996] proposed a hand-simulation model well suited for real-time animation, to be used in conjunction with traditional approaches.

3.3 Applying Artificial Neural Networks to Hand Animation

Humans move their hands using a biological neural network that has learned to manipulate the fingers. A synthetic character could use an artificial neural network (ANN) that has been trained to manipulate the synthetic fingers. A survey of recent technical journals in ANNs and computer graphics suggests that ANNs have not yet been applied to movement animation of synthetic actors in virtual environments.

Virtual human modelling must include the structure needed for virtual human animation. When linking modelling and animation, human figure animation is complex because the human body is capable of achieving vast and innumerable combinations of shape positions and interacts with its environment in many different ways. A

computer model for a synthetic character could try to mimic a human by creating ANNs that are trained to accomplish a particular movement. In robotics, ANNs have been developed that have learned such behaviour patterns as walking and grasping. For example, Rao et al. [1996] used a single stage recurrent neural network to simulate a central pattern generator that produces rhythmic motion for actions such as locomotion and respiration. Another example is found in Srinivasan et al. [1992] where a modified Jordan's sequential network was used to generate several bipedal gaits at different frequencies. A final example is found in Tascillo et al. [1993] where a modified genetic back-propagation neural network controller was used to determine a best first grasp for a robotic hand. Since ANNs have been successful in controlling robotic manipulators, a natural progression would be to apply ANNs to control animated virtual manipulators.

This thesis develops a first step in creating an artificially intelligent virtual human. As stated in section 3.2, animating human movement has usually been divided into animation of three separate body sections; this thesis focuses on animation of human hands. Since the complex model of the thumb and fingers of an animated hand was developed based on tendon control (Kuchar [1996]), back-propagation neural networks were created to learn how to manipulate the tendons to control the postures of the thumb and fingers. A database of end positions was created on which to train and test the neural networks. Once the networks had been trained, they were incorporated into an animation system for visual verification of finger postures, since humans are better at detecting problems visually. The neural networks were able to manipulate the tendons successfully to allow for dynamic hand animation.

3.4 Chapter Summary

This chapter provides the rationale for this thesis and its application within VR. The purpose of this thesis is to determine end positions of animated human hand motions in real time using ANNs. The remainder of this thesis describes the application of the ANNs to tendon control of an animated human hand.

4. Hand Control Subsystem

*A journey of a thousand miles begins
with a single step.
- Confucius*

The human hand is a complex mechanical structure comprised of bones, ligaments loosely connecting bones, muscles serving as tension controllers, tendons acting as cables connecting muscles to bone, and a covering of protective soft tissue and skin. The human hand is an articulated structure with a sum of about 30 relative degrees of freedom (DOF) and changes shape in various ways as a result of its joint movements. The bones are linked at the joints and do not change in size. There are many muscles and tendons in the hand, but only a subset apply to movement. This chapter describes the bones and joints of the hand, the muscles and tendons that produce movement in the thumb and fingers, the underlying mathematical model applied to the muscle effect on the fingers, and the hand-muscle programme that allows the user to manually manipulate the muscles and tendons of the thumb and fingers.

4.1 A General Overview of the Bones of the Hand

The skeleton of the hand consists of three segments: the carpal bones; the bones of the palm; and the bones of the fingers (phalanges). The carpal bones are arranged in two rows. The row closest to the ulna and radius includes the pisiform, triquetral, lunate, and scaphoid. The distal row is arranged with the hamate, capitate, trapezoid, and trapezium. A diagram of the carpal hand is shown in Figure 4-1. There are five metacarpal bones in the palm of the hand, numbered from the thumb side as I through V. As depicted in Figure 4-2, each finger has three phalanges – a proximal, a middle, and a distal. The thumb has only two phalanges.

4.2 Description of Joint Movements

The movement of a particular segment of the hand is produced by rotations of the bone around one or more of three mutually orthogonal axes. A joint can be described as uniaxial, biaxial, and triaxial. When a movement of a bone at a joint is lim-

ited to rotation about a single axis, the joint is termed uniaxial and it possesses one DOF, such as a knee joint. When completely independent movements can occur around two axes, the joint is termed biaxial and it possesses two DOF, such as a metacarpophalangeal (MCP) joint. When completely independent movements can occur around three axes, the joint is termed triaxial and it possesses three DOF, such as a shoulder joint. A closer examination of joint classification can be found in Warwick et al. [1973].

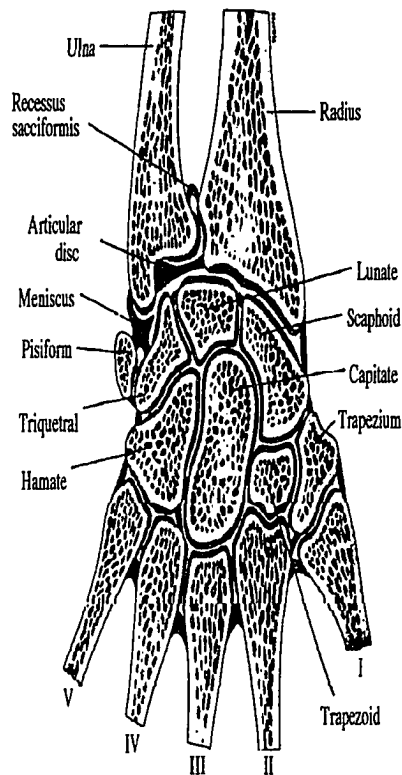


Figure 4-1. A coronal section of the dorsal right hand through the distal ends of the radius and ulna, the carpus and the proximal ends of the metacarpals, showing the general form of the articular surfaces, synovial cavities, interosseous ligaments, and fibrocartilages. Adapted from Warwick et al. [1973].

Figure 4-2 depicts the hand model. Each finger (II - V) has a sum of four relative DOF: two DOF at the MCP joint; one DOF at the proximal interphalangeal (PIP) joint; and one DOF at the distal interphalangeal (DIP) joint. Similarly, the thumb (I) has a sum of five relative DOF: two DOF at the carpometacarpal (CMC) joint; two DOF at the MCP joint; and one DOF at the interphalangeal (IP) joint. The wrist's movements have not been included in this model.

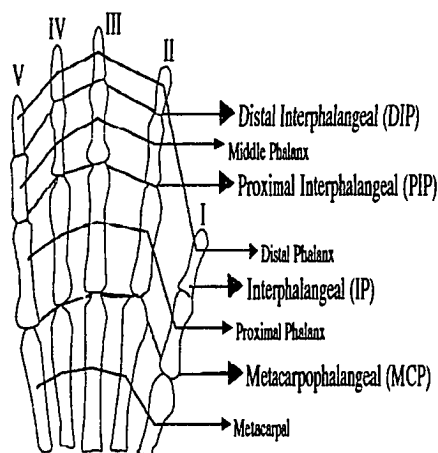


Figure 4-2. A right hand skeleton observed from the palmar side. Adapted from Lee et al. [1995].

Angular movement refers to the change in the angle between adjoining bones. Two common angular movements occur around axes set at right angles to each other and are named:

1. flexion (bending), with extension (straightening) as its opposite.
2. abduction (movement away from the midline of the body), with adduction (towards the midline of the body) as its opposite.

Limitation of movement is effected by a number of different factors, of which the tension of ligaments is very important. In life, however, the tension of the muscles that are antagonistic to the movement is equally important as a limiting factor. The latter involves both the passive elastic component of the muscles, including other soft

tissues surrounding the joint, and the reflex contraction of the appropriate musculature that occurs in the tissues. In synovial joints, where the bones concerned are connected by ligaments and muscles only, the articular surfaces are in constant opposition in all positions of the joint.

4.2.1 The Metacarpophalangeal Joints

The MCP joints are biaxial ellipsoid joints. The heads of the metacarpals, fitting into shallow concavities on the bases of the proximal phalanges, are not regularly convex since they are partially divided on their palmar aspect to resemble condyles.

The active movements of the MCP joints are flexion, extension, adduction, abduction, circumduction (derived compounded movement of flexion, extension, abduction, and adduction), and some limited rotation.

As noted by Warwick et al. [1973], flexion is freer than extension since individual joint manipulation is greater in flexion than in extension, and both movements are limited by the tension of the opposing muscles. Abduction and adduction are less free and cannot be performed actively when the fingers are flexed. The flexion and extension of the MCP joint of the thumb seldom exceed 60° (Seireg et al. [1989]). The side-to-side movements of the thumb are very much restricted at the MCP joint.

The range of active flexion and extension varies between humans. The range of flexion is nearly 90° for the index finger, but increases progressively to the fifth finger. Flexion of the middle finger is limited by tension developed in the deep transverse ligaments of the palm. The range of flexion can reach 30° to 40° .

Since the insertion of the collateral ligaments into the metacarpal head is slightly posterior to its centre of curvature, they become lax in extension and taut in flexion. Thus, abduction and adduction become difficult, if not impossible, when the MCP joint is flexed. When the MCP joint is extended, abduction and adduction are easier and their range is 20° to 30° on either side of the finger midline (Seireg et al. [1989]). Of all the fingers, except the thumb, the index finger has the greatest range of side-to-side movements (30°).

4.2.2 The Interphalangeal Joints

The IP joints are uniaxial hinge joints. The only active movements in the IP joints are flexion and extension. The movements of the IP joints are greater in range between the proximal and middle phalanges than between the middle and distal phalanges. The amount of flexion is very considerable, but extension is limited by the tension of the digital flexors and by the palmar ligaments. Finally, during finger flexion and extension, the movements are accompanied by a small amount of conjunct rotation. During flexion, as noted in Warwick et al. [1973], the rotation turns the pulp of a finger slightly laterally to face more fully the pulp of the opposed thumb. An opposite rotation occurs during finger extension.

The range of flexion in the PIP joints is greater than 90° , so that in flexion the proximal phalanx and the middle phalanx form an acute angle. As in the case of the MCP joints, flexion increases in range from the second to the fifth finger to reach a maximum of 135° with the latter. In Figure 4-3 the phalanges are seen obliquely from the side so that the angles appear obtuse.

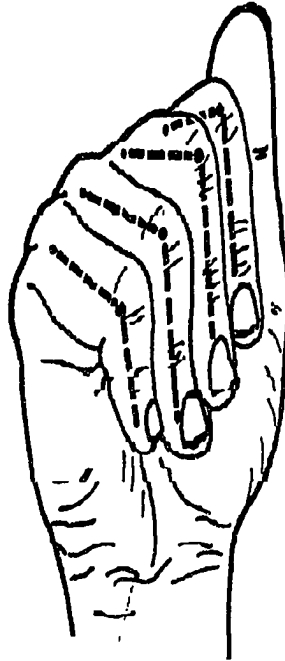


Figure 4-3. The range of flexion in the PIP joints. Adapted from Kapandji [1970].

The range of flexion at the DIP joints is slightly less than 90° so that the angle between the middle phalanx and the distal phalanx remains obtuse. As with the PIP joints, the range increases from the second to the fifth finger to attain a maximum of 90° with the latter. Figure 4-4 depicts the range of flexion at the DIP joints.



Figure 4-4. The range of flexion in the DIP joints. Adapted from Kapandji [1970].

Flexion of the index moves in a sagittal plane (P) towards the base of the thenar eminence (long white arrow shown in Figure 4-5). The axes of the fingers during flexion all converge to a point corresponding to the radial pulse. This can only occur if the other fingers are flexed in an increasingly oblique plane. The pinky shows maximal obliquity to the plane of flexion (small white arrow shown in Figure 4-5). The significance of this oblique flexion lies in the fact that it allows not only the index to oppose the thumb, but all other fingers also.

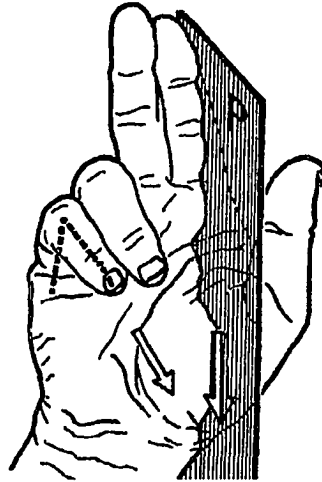


Figure 4-5. Plane of movement of flexion of the four fingers. Adapted from Karpandji [1970].

4.3 Model of the Hand

There are many muscles and tendons in the hand, but only a subset apply to movement. More information on muscle physiology can be found in Sherwood [1992]. The following sections will examine the muscles and tendons that produce movement in the thumb and fingers.

For this research, the thumb and fingers were modelled. The model includes 19 bones and the muscles that produce movements of these bones. Table 4-1 summarizes some of the muscles and tendons of the thumb and fingers and their functions. Since the tendons of the thumb and finger muscles usually are inserted into the distal end of the ulna-radius, an alternate location for the origins of these muscles is selected for the implementation. The new origin is established at the base of the metacarpal bone since, in reality, most tendons must pass through this point and our model of the thumb and fingers begins at the metacarpal bone. The muscles and tendons are, in general, modelled by lines joining suitable origin points to the corresponding insertion points on the phalanges.

Table 4-1. Summary of Modelled Muscles and their Functions.

Abbreviation	Name	Function
FDS	Flexor Digitorum Superficialis	- flexes the fingers
FDP	Flexor Digitorum Profundus	- flexes the fingers
	Flexor Digiti Minimi	- located in the pinky - flexion of the MCP joint
FPL	Flexor Pollicis Longus	- located in the thumb - flexes at the IP joint
FPB	Flexor Pollicis Brevis	- located in the thumb - flexes, adducts
EDC	Extensor Digitorum Communis	- extends the fingers
	Extensor Indicis	- extends the index
	Extensor Digiti Minimi	- extends the pinky
EPL	Extensor Pollicis Longus	- located in the thumb - extends the distal phalanx
EPB	Extensor Pollicis Brevis	- located in the thumb - extends the proximal phalanx - abducts the hand
AP	Adductor Pollicis	- adducts the thumb
ADM	Abductor Digiti Minimi	- abducts the pinky - flexes the proximal phalanx
APB	Abductor Pollicis Brevis	- adduction of the thumb - flexes the MCP joint
APL	Abductor Pollicis Longus	- abducts the thumb - flexes the metacarpal bone

4.3.1 Muscles Producing Movement at the MCP Joints

The muscles that produce the flexion of the MCP joint are the Flexor Digitorum Superficialis (FDS) and the Flexor Digitorum Profundus (FDP). The Flexor Digiti Minimi produces flexion of the MCP joint in the pinky. The thumb muscles that produce flexion of the MCP joint are the Flexor Pollicis Longus (FPL) and the Flexor Pollicis Brevis (FPB), along with the first palmar interosseous. Slight lateral rotation accompanies finger flexion.

The muscle that produces the extension of the MCP joint in the third and ring fingers is the Extensor Digitorum Communis (EDC), assisted, in the index and pinky, by Extensor Indicis and Extensor Digiti Minimi respectively. In the thumb, the Extensor Pollicis Longus (EPL) and Pollicis Brevis (EPB) affect the MCP joint.

The muscles that produce adduction are the palmar interossei. During flexion the long flexors of the fingers play the principal part. The slight degree of this movement in the thumb is attributable to the Adductor Pollicis (AP) and the first palmar interosseous.

The muscles that produce abduction are the dorsal interossei, assisted by the long extensors, except in the case of the middle finger. In the pinky, the Abductor Digiti Minimi (ADM) produces abduction. The Abductor Pollicis Brevis (APB) produces the slight movement possible in the thumb that contributes towards opposition. When the fingers are in the flexed position, abduction cannot be performed actively. The inability to perform abduction actively in this position may be because the dorsal interossei and the ADM are so shortened by flexion that they are unable to function.

4.3.2 Muscles Producing Movement at the IP Joints

Flexion of the PIP joint is produced by the FDS and FDP. Flexion of the DIP joint is produced by the FDP only. At the IP joint of the thumb, the FPL is the only flexor. Extension of the IP joints is accomplished by the EDC and the EPL.

The combined movements of flexion at the MCP joint and extension at the IP joints can be carried out simultaneously, and are of importance in such fine movements of writing, drawing, and threading a needle. When the lumbricales and interossei flex the MCP joints, the balance between the tone of the digital flexors and extensors is altered in favor of the extensors, and this factor alone is responsible for the

extension of the IP joints. However, both the lumbricales and interossei can individually extend these joints.

4.4 Description of the Motor Muscles and the Modelled Counterpart

In the next section, only the muscles that help in producing movement of the thumb and fingers are explained in detail, along with a description of how the muscles were incorporated into the programme.

4.4.1 Tendons of the Flexors

There are two flexors in each finger: the FDS and the FDP. The fleshy bellies of the digital flexors lie in the anterior compartment of the forearm and so can be considered as extrinsic muscles. The course of the flexors across the wrist and the palm of the hand, including their insertions and actions will be described in the following section.

4.4.1.1 Flexor Digitorum Sublimis

The FDS muscle arises from three heads of origin – humeral, ulnar, and radial, and divides into two planes of muscular fibers – the superficial plane carrying tendons for insertion into the middle and ring fingers, and the deep plane carrying tendons for insertion into the index and pinky finger. The FDS has its insertion proximal to that of the FDP muscle, thus the FDS splits into two slips. The two slips reunite and finally insert into the sides around the middle of the second phalanx. The tendons, during their course of insertion, enter into the fibrous tendon sheaths beginning proximally over the heads of the metacarpal bones, which prevent their "bowstringing". It is assumed that the tendons are held to the digits by the fibrous sheaths at approximately the middle of each segment. The muscle therefore can be modelled by four components – each component responsible for the flexion of an individual finger. The flexor is then inserted into the middle phalanx on the palmar side as shown in Figure 4-6.

The action of the FDS flexes the PIP joint as depicted in Figure 4-7. It has no effect on the DIP joint and is a weak flexor of the MCP joint, and only when the PIP joint is fully flexed.

4.4.1.2 Flexor Digitorum Profundus

The FDP arises from the ulna and ends in four tendons that pass through the openings in the tendons of the FDS and insert into the base of the distal phalanges. The model of this muscle is therefore similar to that of the FDS – four components, each going to an individual finger. Each component is modelled by a line similar to the FDS with the line connecting the middle phalanx to the distal phalanx to represent the final insertion on that segment, as shown in Figure 4-8.

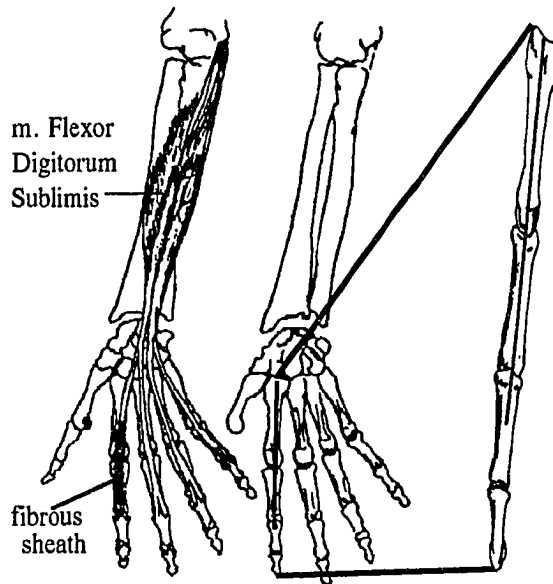


Figure 4-6. The muscle model for the Flexor Digitorum Sublimis. Adapted from Seireg et al. [1989].

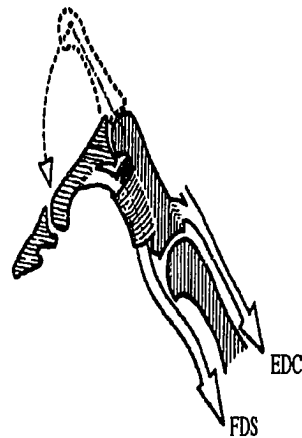


Figure 4-7. The Flexor Digitorum Sublimis. Adapted from Kapandji [1970].

As depicted in Figure 4-9, the FDP is primarily a flexor of the DIP joint but flexion of this joint is soon followed by flexion of the PIP joint, which has no special extensor to antagonize this action. When the MCP and PIP joints are flexed passively to 90° the FDP cannot flex the DIP joint because it has become too slack for any useful contraction.

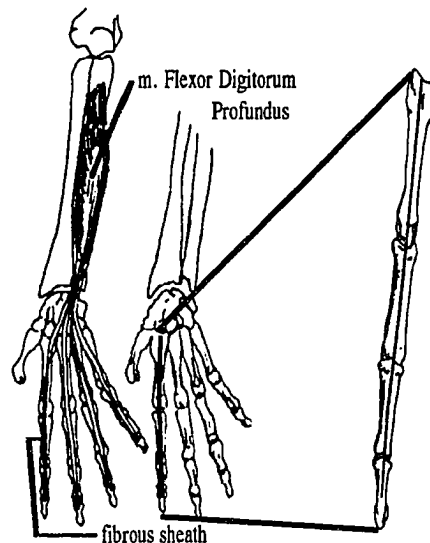


Figure 4-8. The muscle model for the Flexor Digitorum Profundus. Adapted from Seireg et al. [1989]

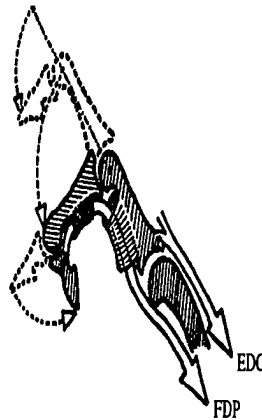


Figure 4-9. The Flexor Digitorum Profundus. Adapted from Kapandji [1970].

4.4.2 Tendons of the Extensor

The extensor is also an extrinsic muscle of the hand and is described in the following section.

4.4.2.1 Extensor Digitorum Communis

The EDC originates on the lateral epicondyle of the humerus, passes across the wrist and divides into four tendons that insert into the second and third phalanges of the fingers. The manner of insertion of the tendons is rather complex, forming the extensor apparatus for each finger as shown in Figure 4-10. After crossing the MCP articulation the tendon is joined by the tendons of the interossei and lumbricales, spreading into a broad aponeurosis covering the dorsal surface of the first phalanx. Opposite the first IP joint, this aponeurosis divides into three bands. The central band is inserted into the base of the second phalanx, while the two collateral bands continue onward along the side of the second phalanx, unite across the DIP and insert into the dorsal surface of the distal phalanx. When the fingers are flexed, the tendon will wrap around the digits and consequently the muscle is modelled by several lines, with appropriate reactions at assumed points of contact.

The muscle model for the EDC consists of four major parts, each part representing the part of the muscle responsible for the extension of an individual finger. Each part in turn consists of three components to represent the three bands -- the central, the medial and the lateral that form the individual extensor mechanism. Hence,

the central band of the extensor apparatus of each finger consists of a line connecting the base of the metacarpal to a point on the dorsal surface of the proximal phalanx. The collateral bands (medial and lateral) are modelled as lines connecting from the middle of the proximal phalanx and attaching to the midpoint of the middle and distal phalanges simulating the appropriate movement.

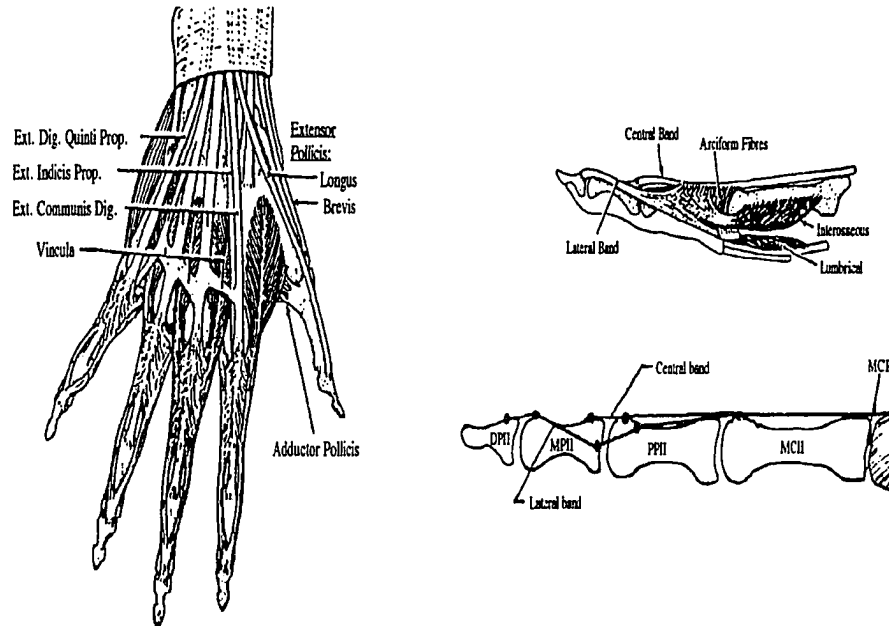


Figure 4-10. The model of the extensor apparatus. Adapted from Seireg et al. [1989].

Functionally, the EDC is essentially an extensor of the MCP joint. The EDC is a powerful extensor and active in all positions of the wrist. The EDC action of the PIP joint, by means of the medial band (number 2 on Figure 4-11) and on the DIP joint, by means of the two lateral bands (number 3 on Figure 4-11), depends on the amount of tension in the tendon and on the position of the wrist.

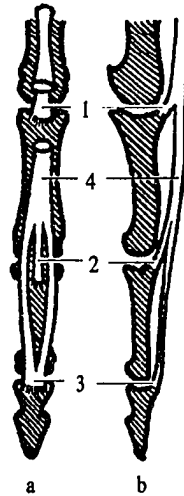


Figure 4-11. The extensor expansion. (1) tendon has been partially resected to show the deep expansion. (2) median band. (3) lateral band. (4) aponeurosis. Adapted from Kapandji [1970].

The action of the EDC on the DIP and PIP joints depends on the amount of tension in the digital flexors. If the digital flexors are taut because the wrist or the MCP joint is extended, the EDC by itself cannot extend these interphalangeal joints. If the flexors are relaxed by flexion of the wrist or the MCP joint or are sectioned, the EDC can easily extend these IP joints.

There has been debate among researchers about finger extension and the role of the EDC and intrinsic muscles. Long et al. [1964] stated that the interossei reinforce the function of the lumbricales when the EDC cannot assist the lumbricales in extension of the distal joints. The EDC always seems to cooperate with the lumbricales when extending interphalangeal extension. Long et al. [1964] based these conclusions on hand experiments performed using EMG signals. Tubiana et al. [1996] stated that the interosseous muscles are dependent on the position of the MCP joint when extending the interphalangeal joints. Tubiana et al. [1996] also stated that the lumbrical muscles are able to extend the two distal phalanges whether the MCP joint is in extension or flexion. This debate is on-going because the interossei and the lumbrical muscles are connected with the middle band of the EDC, and it is difficult to determine their functions independently. The intrinsic muscles will be discussed in greater detail in section 4.4.3.

4.4.2.2 Extensor Indicis Proprius and Extensor Digiti Minimi

The tendons of the Extensor Indicis and Extensor Digiti Minimi behave in the same way as those of the EDC with which they blend. They allow the index and pinky finger to be extended singly. These muscles are not modelled as extra muscles since they only add to the flexibility and power of the finger.

4.4.3 The Intrinsic Muscles

The intrinsic muscles are composed of the interossei and lumbricales. The interossei have two actions: adduction and abduction; and flexion and extension. There are two different interossei: dorsal and palmar. The lumbricales flex the MCP joint and extend the IP joints. These muscles are described in this section.

4.4.3.1 Dorsal Interosseous

The four dorsal interosseous occupy the spaces between the metacarpal bones, depicted in Figure 4-12a. The dorsal interosseous originates by two heads from the adjacent sides of the metacarpal bones. The dorsal interosseous inserts into the bases of the proximal phalanges and into the lateral bands of the aponeurosis of the tendons of the extensor digitorum. Thus, the four dorsal interossei are:

1. The first dorsal interosseous is located between MC I and MC II and arises from the same two bones, and inserts into the base of PPII and into the lateral band (radial) of the extensor apparatus of the index finger.
2. The second dorsal interosseous arises from the MC II and MC III and inserts into the base of the PPIII and into the lateral band of the extensor apparatus of the middle finger.
3. The third dorsal interosseous arises from the MC III and MC IV and inserts into the base of the PPIII and into the medial band (ulna) of the extensor apparatus of the middle finger.
4. The fourth interosseous arises from the MC IV and MC V and inserts in to the base of the PPIV and into the medial band of the extensor apparatus of the ring finger.

Each of the four dorsal interossei is modelled by two components – one that inserts directly into the proximal phalanx and the other that represents the insertion into the medial or the lateral bands of the extensor apparatus. The latter are modelled similar to the bands described in the EDC model.

4.4.3.2 Palmar Interosseous

The three palmar interossei (Figure 4-12b) lie on the palmar surfaces of the metacarpal bones. The palmar interossei arise from the metacarpal bone. The interossei insert into the side of the base of the first phalanx and into the lateral band of the aponeurosis of the tendon of the EDC. Each of the three interossei thus can be modelled by a line connecting the metacarpal bone of the finger to the proximal phalanx of the same finger.

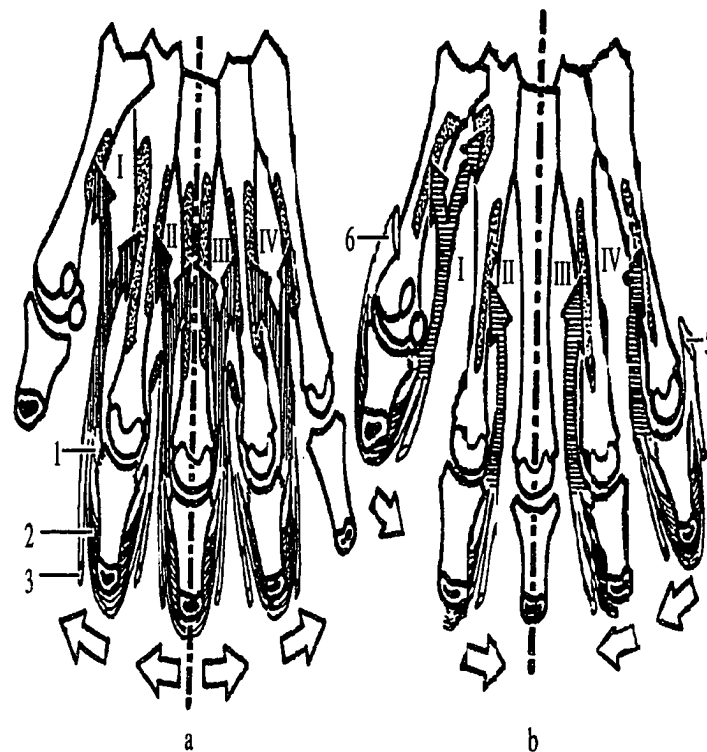


Figure 4-12. The Interossei. (a) Dorsal Interossei. (b) Palmar Interossei. Adapted from Kapandji [1970].

4.4.3.3 Lumbricales

The lumbricales are four small fleshy muscles associated with the FDP and the EDC. The lumbrical muscles give the balance between the flexors and extensors. Tubiana [1981] stated that the motor function of the lumbricales remains ill understood. Generally, the lumbricales extend the interphalangeal joints, do not interact in flexion of the MCP joint, and are partially involved in side-to-side and rotation movements of the fingers. The lumbrical muscle was included in the programme but as a part of the modelled intrinsic muscle that supports extension of the interphalangeal joints. Thus, these muscles are modelled similarly to the collateral bands of the EDC.

4.4.4 Muscles in the Thumb

The thumb is unique because there are nine motor muscles located in the thumb. The thumb motor muscles are grouped as extrinsic (long) and intrinsic (short) muscles. The motor muscles of the thumb are described in this section.

4.4.4.1 Extrinsic Muscles

The extrinsic muscles are the long muscles of the thumb. There are four extrinsic muscles: Abductor Pollicis Longus (APL), Extensor Pollicis Brevis (EPB), Extensor Pollicis Longus (EPL), and Flexor Pollicis Longus (FPL).

4.4.4.1.1 Abductor Pollicis Longus

The APL tendon arises from the dorsal surface of the ulna and the radius and from the interosseous membrane between them. The APL tendon passes through a groove on the lateral side of the distal end of the radius. The APL tendon inserts into the radial side of the base of the first metacarpal bone. The tendon can be modelled by a line connecting to a point on the base of the first metacarpal bone. The APL abducts and flexes the first metacarpal bone as depicted in Figure 4-13.

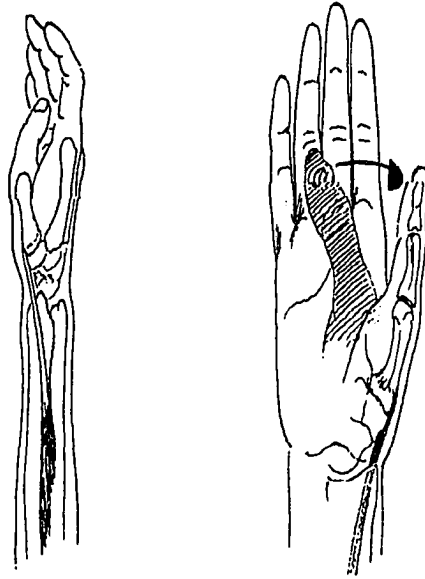


Figure 4-13. The muscle model for the Abductor Pollicis Longus. Adapted from Chase [1973].

4.4.4.1.2 Extensor Pollicis Brevis

This muscle arises from the radius and the interosseous membrane and inserts into the base of the first phalanx of the thumb, as depicted in Figure 4-14. It is therefore modelled as a line connecting the base of the metacarpal bone to an insertion point on the proximal phalanx of the thumb. This muscle is modelled similar to the FDS. The EPB extends the MCP and abducts the first metacarpal bone.

4.4.4.1.3 Extensor Pollicis Longus

The EPL arises from the ulna and from the interosseous membrane, and inserts into the base of the last phalanx of the thumb. The muscle can be modelled by a similar line as the FDP that inserts into the distal phalanx. The EPL extends the IP joint and the MCP joint. The EPL also adducts the first metacarpal bone.

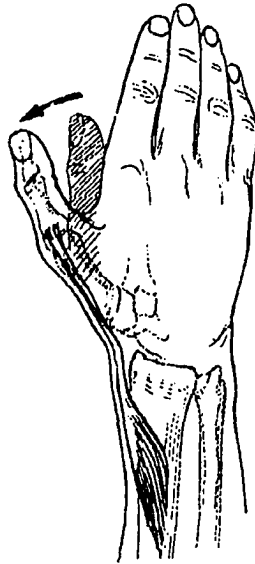


Figure 4-14. The muscle model for the Extensor Pollicis Brevis. Adapted from Chase [1973].

4.4.4.1.4 Flexor Pollicis Longus

The FPL has its origin on the radius, the interosseous membrane, and the medial epicondyle of the humerus. The FPL fibers converge into a flattened tendon that passes under the flexor retinaculum, enters into an osseo-aponeurotic tunnel similar to those for the flexor tendons of the fingers, and finally inserts into the base on the distal phalanx of the thumb. It is assumed that the restraint to bowstringing provided by the fibrous sheaths is at the center of the phalanx. The muscle is modelled similar to the FDP. The FPL primarily flexes the IP joint and secondarily flexes the MCP joint.

4.4.4.2 Intrinsic Muscles

The intrinsic muscles are the muscles lying within the thenar eminence and first interosseous space. There are five extrinsic muscles: Flexor Pollicis Brevis (FPB); Abductor Pollicis Brevis (APB); Adductor Pollicis (AP); Opponens Pollicis (OP); and the first interosseous. The FPB, APB, AP, and OP are described in this section. The first interosseous is modelled similarly to the other interossei described in section 4.4.3.1.

4.4.4.2.1 Flexor Pollicis Brevis

The FPB arises from the flexor retinaculum and the distal part of the ridge on the trapezium, and inserts into the base of the proximal phalanx of the thumb. The muscle is modelled similarly to the FDS. The FPB is primarily an adductor and brings the ball of the thumb into opposition to the last two digits.

4.4.4.2.2 Abductor Pollicis Brevis

The APB originates on the scaphoid and the trapezium and inserts into the radial side of the base of the first phalanx of the thumb, as shown in Figure 4-15. The muscle is modelled by a line connecting a point on the metacarpal to a point on the base of the proximal with a reaction on the head of the first metacarpal. The APB abducts the first metacarpal bone while flexing the MCP joint.

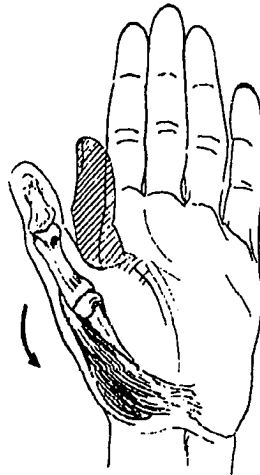


Figure 4-15. The muscle model for the Abductor Pollicis Brevis. Adapted from Chase [1973].

4.4.4.2.3 Adductor Pollicis

The AP has two main parts – an oblique part and a transverse part. The oblique head arises from the capitate bone and the bases of the second and third metacarpal bones; the transverse head arises from the palmar surface of the third metacarpal bone. Both heads converge towards the thumb to insert into the ulnar side of the base of the proximal phalanx. The muscle is modelled by four lines – from the capi-

tum; the base of index metacarpal bone; the base of the third metacarpal bone; and the palmar surface of the third metacarpal bone; all inserting into the proximal phalanx of the thumb. The AP adducts the thumb toward the palm.

4.4.4.2.4 Opponens Pollicis

The OP originates on the anterior surface of the flexor retinaculum, deep to the fibers of the APB and proximal to the origins of the superficial head of the FPB. The OP inserts into the lateral border of the shaft of the first metacarpal and into the lateral aspect of its neck. The OP has three actions: flexion of the first CMC; adduction (the first metacarpal is drawn towards the second metacarpal); and axial rotation (pronation). This muscle is essential for opposition of the thumb with the fingers.

4.5 Mathematical Model

As discussed in the previous section, several muscles were modelled according to their original counterparts. This section gives a flavour for the underlying mathematical model (presented in Kuchar [1996]) applied to the muscle effect on the fingers. Only one example of the mathematical model is provided – the FDP tendon. Other examples of the mathematical model can be found in Kuchar [1996].

4.5.1 The Basic Finger Model

The finger model is shown in Figure 4-16. The finger is reduced to a line structure with four segments and three joints that correspond to the bones and joints of a human finger. A "puppet string" is used to represent a tendon. The puppet string is attached to the bone segment similar to the original tendon insertion. The puppet string is the only part of the model that the user manipulates with the sliders in the graphical interface. The sliders are described in section 4.6.5.

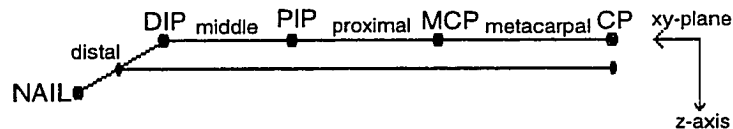


Figure 4-16. The Finger Model.

4.5.2 The FDP Mathematical Model

The FDP is primarily a flexor of the DIP joint but flexion of this joint is soon followed by flexion of the PIP joint, which has no special extensor to antagonize this action. As stated in section 4.4.1.2, the FDP tendon is modelled by a line that connects the base of the metacarpal bone to the middle of the distal phalanx. For computational reasons, the middle of the distal phalanx was chosen instead of the base of the distal phalanx.

Since a tendon affects all joints that it passes, the FDP affects the DIP, PIP, and MCP joints. When modelling the effects of the FDP tendon on the joints, three cases occur:

1. The DIP joint is flexed by the pull of the puppet string. If this joint is flexed past the maximum constraint limit of the DIP angle specified for the finger, then the DIP joint is placed at the maximum angle and flexion of the PIP joint begins.
2. The PIP joint is flexed by the pull of the puppet string. In this case, the DIP joint has been fully flexed. If the PIP joint is flexed past the maximum limit of the angle specified for the finger, then this joint is placed at the maximum constraint angle and flexion of the MCP joint begins.
3. The MCP joint is flexed by the pull of the puppet string. In this case, the IP joints have been fully flexed. If the MCP joint is flexed past the maximum limit of the angle specified for the finger, then this joint is placed at the maximum angle and the finger is fully flexed.

Each case mentioned above is described by the accompanying mathematical model.

CASE 1: Flexion at the DIP Joint

The algorithm that calculates the angle at the DIP joint, indicated as φ in Figure 4-17, is now presented.

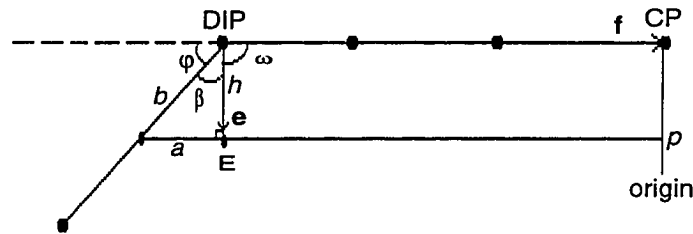


Figure 4-17. DIP joint is flexed.

The following constraints exist in this algorithm:

1. p is the slider value passed from the interface to the FDP procedure. p is the current length of the puppet string.
2. b is one-half the length of the distal phalanx.

The algorithm to calculate φ is:

1. Calculate β .
Calculate a as the remainder of the puppet string when the lengths of the middle, proximal, and metacarpal bones are subtracted from p . Thus, β is calculated as the arcsine of (a / b) .
2. Calculate ω .
First, point E needs to be determined. The (x, y) values of E are obtained using the (x, y) coordinates of the DIP joint; however, the z coordinate needs to be calculated. Thus, using the Pythagorean theory, the variables a and b can be used to calculate h . The z coordinate of E is $(-h)$. Secondly, a vector \mathbf{e} is constructed from the points E and the DIP joint. Thirdly, a vector \mathbf{f} is constructed to the base of the metacarpal bone from the current flexing joint. Finally, ω can be calculated using the dot product of the two vectors \mathbf{e} and \mathbf{f} .

3. Calculate φ .
To calculate φ , the angle at the DIP joint, subtract the sum of β and ω from 180° . This is the flexion of the finger at the DIP joint. If φ is flexed past the maximum constraint limit of the angle specified in the input file for the finger, then φ is placed at the maximum angle and flexion of the PIP joint begins.

CASE 2: Flexion at the PIP Joint

The algorithm that calculates the angle at the PIP joint, indicated as γ in Figure 4-18, is now presented.

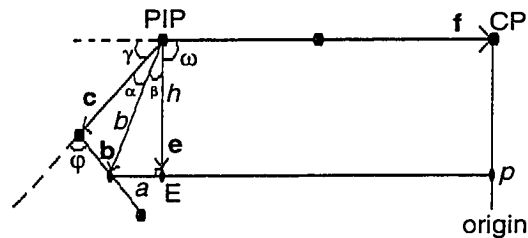


Figure 4-18. PIP joint is flexed.

The following constraints exist in this algorithm:

1. p is the slider value passed from the interface to the FDP procedure. p is the current length of the puppet string.
2. β is the constant length between the midpoint of the distal phalanx to the PIP joint.
3. φ is the maximum angle of the DIP joint from case 1.

The algorithm to calculate γ is:

1. Calculate β .
Calculate a as the remainder of the puppet string when the lengths of the proximal and metacarpal bones are subtracted from p . Thus, β is calculated as the arcsine of (a / b) .

2. Calculate ω .
First, point **E** needs to be determined. The (x, y) values of **E** are obtained using the (x, y) coordinates of the PIP joint; however, the z coordinate needs to be calculated. Thus, using the Pythagorean theory, the variables a and b can be used to calculate h . The z coordinate of **E** is $(-h)$. Secondly, a vector **e** is constructed from the points **E** and the PIP joint. Thirdly, a vector **f** is constructed to the base of the metacarpal bone from the current flexing joint. Finally, ω can be calculated using the dot product of the two vectors **e** and **f**.
3. Calculate α .
First determine two vectors **b** and **c**. Vector **b** is constructed from the PIP joint to the midpoint of the distal phalanx. Vector **c** is constructed to the DIP joint from the PIP joint. Thus α is calculated using the dot product of **b** and **c**.
4. Calculate γ .
To calculate γ , the angle at the PIP joint, subtract the sum of α , β , and ω from 180° . This is the flexion of the finger at the PIP joint. If γ is flexed past the maximum constraint limit of the angle specified in the input file for the finger, then γ is placed at the maximum angle and flexion of the MCP joint begins.

CASE 3: Flexion at the MCP Joint

The algorithm that calculates the angle at the MCP joint, indicated as θ in Figure 4-19, is now presented.

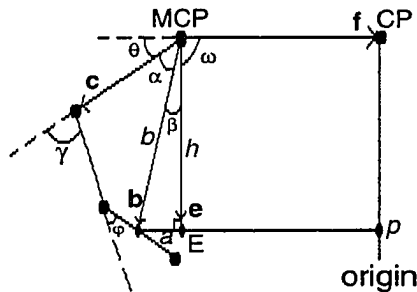


Figure 4-19. MCP joint is flexed.

The following constraints exist in this algorithm:

1. p is the slider value passed from the interface to the FDP procedure. p is the current length of the puppet string.
2. β is the constant length between the midpoint of the distal phalanx to the MCP joint.
3. φ is the maximum angle of the DIP joint from case 1.
4. γ is the maximum angle of the PIP joint from case 2.

The algorithm to calculate θ is:

1. Calculate β .
Calculate a as the remainder of the puppet string when the length of the metacarpal bone is subtracted from p . Thus, β is calculated as the arcsine of (a / b) .
2. Calculate ω .
First, point **E** needs to be determined. The (x, y) values of **E** are obtained using the (x, y) coordinates of the MCP joint; however, the z coordinate needs to be calculated. Thus, using the Pythagorean theory, the variables a and b can be used to calculate h . The z coordinate of **E** is $(-h)$. Secondly, a vector **e** is constructed from the points **E** and the MCP joint. Thirdly, a vector **f** is constructed to the base of the metacarpal bone from the current flexing joint. Finally, ω can be calculated using the dot product of the two vectors **e** and **f**.
3. Calculate α .
First determine two vectors **b** and **c**. Vector **b** is constructed from the MCP joint to the midpoint of the distal phalanx. Vector **c** is constructed to the PIP joint from the MCP joint. Thus α is calculated using the dot product of **b** and **c**.

4. Calculate θ .
To calculate θ , the angle at the MCP joint, subtract the sum of α , β , and ω from 180° . This is the flexion of the finger at the MCP joint. If θ is flexed past the maximum constraint limit of the angle specified in the input file for the finger, then γ is placed at the maximum angle and the finger is fully flexed.

The corresponding computer procedure for the FDP tendon is located in Appendix A.

4.6 Computer Implementation

Several programmes were developed for this thesis. The first programme is a hand muscle interface that allows the user to manipulate tendons manually. This section describes the finger movement interface and the rationale for the interface.

4.6.1 The Input File

A file can be created to contain data specific to a hand or data to a generic hand. This file is used as input to the programme, and a sample file is shown in Table 4-2.

Table 4-2. A Sample Input File.

	Dis.	Mid.	Prox.	Carp.	DIP	PIP	MCP	Dor.
Index	23.5	26.0	39.0	55.0	63.3	102.0	71.2	41.9
Third	26.0	28.5	43.5	55.0	55.0	100.0	53.6	33/32
Ring	25.2	27.0	37.5	55.0	71.0	96.0	76.8	32.0
Pinky	21.0	20.0	32.0	45.0	52.7	94.9	87.9	42.0
Thumb	25.0		40.0	35.0	95.0	80.5	21.2	

The format of the file is as follows:

1. four measurements for the distal phalanx, middle phalanx, proximal phalanx, and carpal bones measured in millimeters.
2. three angle measurements which give the maximum flexion of the DIP, PIP, and MCP joints measured in degrees.
3. one angle measurement which gives the maximum abduction of the finger measured in degrees.

The information for the hand is entered in the following sequence: index; third; ring; pinky; and thumb. The thumb has a different format from the above since it has a different structure. The following is the description of the input for the thumb:

1. three measurements for the distal phalanx, proximal phalanx, and carpal bones measured in millimeters.
2. three angle measurements which give the maximum flexion of the DIP, PIP, and MCP joints measured in degrees.
3. one angle measurement which gives the maximum abduction of the thumb measured in degrees.

4.6.2 Start-up Screen

As the programme begins executing, an interface appears in which the user can manipulate the fingers. The initial screen is shown Figure 4-20. A hand is generated on the left side of the screen. The hand can be rotated in the 3D environment using the mouse buttons for x-axis, y-axis, and z-axis rotation. Section 4.6.4 describes the mouse buttons and movements in greater detail.

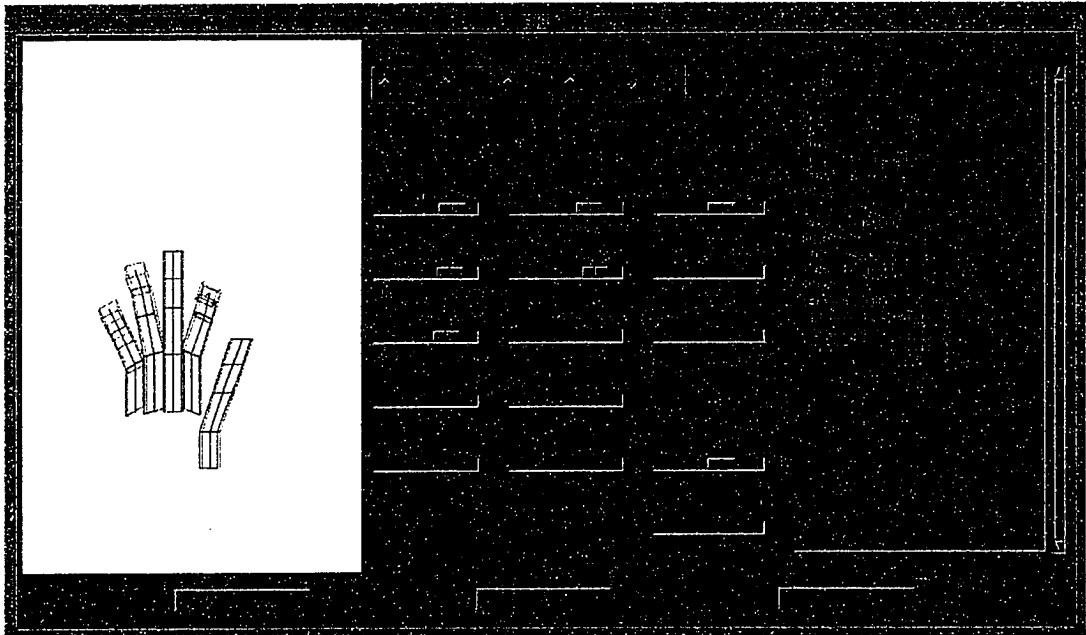


Figure 4-20. Initial screen of the muscle programme.

The largest area of the interface contains the buttons and sliders for manipulating the fingers. At the top of this area, the user may select which finger to manipulate. The sliders which represent the different tendons and muscles in the fingers are set appropriately for each finger. If a tendon or muscle does not exist for that finger, then the slider is de-activated. The slider value indicates the current length of the tendon.

To the right of the user interface is a log which keeps track of all the sliders and fingers which are manipulated during the session. This helps the user recognize how a finger reaches its final state, since moving tendons in different orders create a different end position for the finger.

There are three buttons at the bottom of the user interface: reset finger; reset hand; and quit. The reset finger button resets the currently selected finger to a straight finger. The reset hand button resets all of the fingers simultaneously. The quit button terminates the programme.

4.6.3 Mouse Movements and Rotations

The hand can be rotated in the 3D environment using the mouse buttons for x-axis, y-axis, and z-axis rotation. The rotations are done around 3 orthogonal-axes corresponding to the right-hand coordinate system. For the user to rotate the hand, the user presses and holds one of the mouse buttons while moving the mouse pointer horizontally across the view area. The effects of the rotation are shown on the left side of the interface. For x-axis rotations, the user must press and hold the left mouse button. Similarly for y-axis and z-axis rotations, the user must press and hold the middle and right mouse buttons respectively.

The rotations of the hand in the view area are advantageous to the user. The ability for the user to rotate the hand may provide the user with a better view of the finger movement that is being accomplished.

4.6.4 The Sliders

The sliders that are present in the interface relate to the muscles and tendons described previously in this chapter. The values for the sliders are calculated based on the length of the phalanges and on the insertion point of the tendon or muscle. As an example, the maximum value of the FDP slider indicates that the finger is fully ex-

tended. As noted in section 4.4.1.2, the FDP inserts into the middle of the distal phalanx. The maximum length of the FDP is then calculated from this middle position on the distal phalanx to the end of the finger at the base of the carpal bone. The finger is then placed in the flexed position for each phalanx based on the angular constraints for the finger's joints. The minimum length of the FDP is calculated as the distance from the middle of the distal phalanx to the end of the finger at the base of the carpal bone. A similar approach is followed to calculate the other sliders' minimum and maximum values for each tendon associated with the finger. All of this data is then stored in a linked-list structure.

4.7 Chapter Summary

This chapter described the bones and joints of the hand, the muscles and tendons that produce movement in the thumb and fingers, the underlying mathematical model applied to the muscle effect on the fingers, and the hand-muscle programme that allows the user to manually manipulate the muscles and tendons of the thumb and fingers. Before any artificial neural network could be developed for controlling tendons, this underlying tendon model needed to be developed, implemented, and tested.

5. Artificial Neural Networks Controlling Tendons in an Animated Hand

*Practically all great artists accept the information of others.
But... the artist with vision sees his material, chooses, changes, and by integrating
what he has learned with his own experiments, finally molds something distinctly personal.
- Romare Howard Bearden*

Humans move their hands using a biological neural network that has learned to manipulate the fingers. A synthetic character could use an artificial neural network (ANN) that has been trained to manipulate the synthetic fingers. In chapter 4, the underlying tendon model for a hand was described and implemented. In the hand-muscle programme, an animator could manually manipulate the thumb and fingers by controlling the tendons. In this chapter, ANNs are developed and tested for controlling tendons in the hand to achieve the desired motion.

5.1 Hierarchical Hand Movement Control Scheme

Virtual human modelling must include the structure needed for virtual human animation. Section 4.6 describes a hand-muscle programme that allows an animator to manually manipulate the thumb and fingers by controlling the tendons. A virtual human would require the knowledge of how to move its thumb and fingers by learning to control the tendons in its hands. The goal of this thesis is to develop ANNs that control the tendons described in chapter 4.

The human's biological network triggers the appropriate tendons to create the desired motion. For example, if a human wants to make a fist, a human has learned that a fist is created by fully flexing all fingers and then flexing the thumb. Applying this ideology to virtual humans, several aspects needed to be created: a database of hand commands such as *FIST*; a database of finger commands such as *CLOSE_INDEX*; a database of normalized tendon values for a finger; a hand neural network; and an underlying animation control system. Each of these aspects is described in detail in the remainder of this chapter.

5.2 Hand Movements

An understanding of hand movements was needed before developing ANNs to control thumb and finger postures. This section describes a classification of hand movements and the databases developed here for these movements.

5.2.1 Classification of Hand Movements

Tubiana [1981] stated that for basic kinesiological descriptions, it is possible to classify most hand motions in an inclusive system. Once such a system has been defined, any complex motions of the hand are combinations of the basic motions classified in the inclusive system. This thesis deals with free motion. In free motion, the hand moves freely in space, unencumbered by external resistance without significant compression of the digits against each other or against the palm. The basic free motions possible to the hand are:

- *open*: fully extending the thumb and the fingers until the hand is fully open.
- *close*: fully flexing the thumb and the fingers until the hand is closed in a fist with the thumb overlapping the index and middle fingers.
- *claw*: the movement that reaches the terminal position of MCP extension and IP flexion.
- *reciprocal*: the movement that reaches a terminal position of MCP flexion and IP extensions.

Twelve variations of these motions are observed if the terminal position of each motion is the starting point of each other motion. For example, from the open position the hand can move into closing, clawing, or reciprocal motions. Landsmeer et al. [1965] and Long et al. [1964] used such categories to classify finger control, and deduced fourteen patterns of finger motion:

1. Movement from a fully-flexed finger, keeping the IP joints flexed, to the position of MCP extension.
2. Movement from the position of IP joints flexed and MCP joints extended to the position of all-joint extension.

3. Movement of IP joints into flexion from the position of all-joint extension, while MCP joints are held straight.
4. Movement from the position of IP flexion, MCP extension to the position of all-joint flexion.
5. Movement from the position of all-joint flexion to the position of IP extension with the MCP joint held in flexion.
6. Movement from the position of MCP joint flexed and IP joints extended to MCP joint extended and IP joints extended, keeping IP joints extended throughout the motion.
7. Movement from the position of all-joint extension to the position of all-joint flexion.
8. Movement from the position of all-joint flexion to the position of all-joint extension.
9. Movement from the position of all-joint extension to the position of MCP flexion, IP extension, keeping the IP joints extended throughout the motion.
10. Movement from the position of IP extension and MCP flexion to the position of IP flexion and MCP extension.
11. Movement from the position of IP flexion and MCP extension to the position of IP extension and MCP flexion.
12. Movement from the position of IP extension and MCP flexion to the position of all-joint flexion.
13. Movement from all-joint extension to the position of PIP and MCP flexion, keeping the DIP extended throughout the motion.
14. Movement from the position of PIP and MCP flexion to the position of all-joint extension.

Table 5-1 depicts all possible finger movement combinations. The first 14 cases depicted in Table 5-1 were created using the above classification of finger control as a guide. The table incorporates only six different starting and finishing positions, specifically:


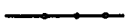






















































- CLOSE (): fully flexed finger;
- OPEN (): fully extended finger;

Table 5-1. 25 cases of finger flexion and extension.

CASE NUMBER	POSITION CHANGE FROM	POSITION CHANGE TO	ACTIVE TENDON
1			EDC
2			INTR
3			FDP
4			FDP
5			INTR
6			EDC
7			FDP
8			EDC+INTR
9			DOR
10			EDC+FDP
11			DOR+INTR
12			FDP
13			FDS
14			EDC+INTR
15			INTR
16			FDS
17			EDC
18			FDS
19			FDP
20			INTR
21			FDS
22			FDP
23			EDC+FDS
24			EDC+FDP
25			DOR+INTR

- FF-DIP-STR (): fully flexed finger with the DIP joint remaining extended;
- HOOK (): fully flexed finger with the MCP joint fully extended;
- V-SHAPE (): fully flexed finger with the DIP and MCP joints fully extended;
- POINT-DOWN (): fully extended finger with the MCP joint fully flexed.

30 variations of the above positions can be created if the terminal position of each finger is the starting point of each other position. Five cases have been eliminated by the author after an in-depth study of human finger movements from Tubiana [1981], Kapandiji [1970], and Seireg et al. [1989]:

- From CLOSE to FF-DIP-EXT;
- From CLOSE to V-SHAPE;
- From HOOK to FF-DIP-EXT;
- From HOOK to V-SHAPE;
- From FF-DIP-EXT to CLOSE.





The first four cases were eliminated because no tendon exists in the finger that can extend the DIP joint before the PIP joint. The last case was eliminated because when the MCP and PIP joints are flexed to 90°, the FDP cannot flex the DIP joint for it has become too slack for any useful contraction, as noted in Kapandiji [1970]. The fourth column in Table 5-1 indicates which tendons are required to be triggered from the starting position to create the desired finishing position.

The cases in Table 5-1 depict finger flexion and extension; however, the finger also abducts and adducts. Two cases were included to reflect these movements:

- ABDUCT: a finger is moved away from the midline;
- ADDUCT: a finger is moved towards the midline.

These two cases are depicted in Table 5-2.

Table 5-2. Two cases of finger abduction and adduction.

Case Number	Position Change From	Position Change To	Active Tendon
26			DI
27			PI

5.2.2 Databases

Based on hand motion classification and finger motion patterns, there were three databases developed for this research: a Hand Commands Database; a Finger Commands Database; and a Finger Tendon-Length Database.

The first database mentioned was the Hand Commands Database. As noted in section 5.1, any complex motions of the hand are combinations of the basic motions classified in the inclusive system. Since the hand motions are described as combinations of finger commands, this database is dynamic for an animator can edit this database to define new postures as required in terms of finger commands. Currently, this database only contains the following hand motions: flat; fist; claw; spread; hook; and a subset of the American Sign Language.

The second database mentioned was the Finger Commands Database. Each hand command can be described in terms of finger commands. For example, a hand command *FIST* would require the following finger commands: *CLOSE_INDEX*; *CLOSE_THIRD*; *CLOSE_RING*; *CLOSE_PINKY*; and *CLOSE_THUMB*. The finger commands are the eight positions described in section 5.1: *CLOSE*; *OPEN*; *FF-DIP-STR*; *HOOK*; *V-SHAPE*; *POINT-DOWN*; *ABDUCT*; and *ADDUCT*. The specific finger that this command is modifying is appended as an argument to the finger command. An excerpt from the Finger Commands Database is depicted in Table 5-3.

The third database mentioned was the Finger Tendon-Length Database. This database describes the eight finger commands in terms of normalized tendon lengths. To develop this database, the author used the hand-muscle programme described in section 4.6 to determine the length of all tendons to create the specified finger command. These lengths were normalized to provide generality to the tendons in all fingers. When a tendon is at its rest position (open hand), the length value is 100%; when

a tendon is completely taut, the length value is 0%. For example, a command CLOSE would require the following tendon lengths: FDP at 4.9%; FDS at 0%; DOR at 0%; EDC at 4.9%; and INTR at 4.9%. Noting that the FDP, EDC, and INTR lengths are set to 4.9% because the range on the sliders was increased to accommodate the FDS-only flexion which would set the slider for the FDP to 0%. Using this interface (see Figure 4-20), the author fully flexed a finger (by setting the slider value to the FDP minimum) and recorded the lengths indicated by the sliders for each tendon. An excerpt from the finger tendon-length database is depicted in Table 5-4.

Table 5-3. Excerpt from the Finger Commands Database.

HAND COMMAND	FINGER COMMANDS
FIST	CLOSE_INDEX, CLOSE_MIDDLE, CLOSE_RING, CLOSE_PINKY, CLOSE_THUMB
POINT	OPEN_INDEX, CLOSE_MIDDLE, CLOSE_RING, CLOSE_PINKY, CLOSE_THUMB
PEACE	CLOSE_RING, CLOSE_PINKY, CLOSE_THUMB, OPEN_INDEX, OPEN_MIDDLE, ABD_INDEX, ABD_MIDDLE

Table 5-4. Excerpt from the Finger Tendon-Length Database.

FINGER COMMAND	FINGER TENDON LENGTHS
CLOSE_INDEX	0.0490 0.0 0.0 0.0490 0.0490
OPEN_INDEX	1.0 1.0 1.0 1.0 1.0

5.3 Artificial Neural Networks in this Research

There were several ANNs developed and tested within this research. This section provides a general overview of the type of ANNs adopted, and the training algorithm applied to these ANNs.

5.3.1 Artificial Neural Network Architecture

As discussed in section 2.2.3, there exist many different types of ANNs. The ANNs developed for this research are feedforward neural networks. A feedforward network is a graph with no loops. Feedforward networks are static for they only pro-

duce one set of output values, and are memory-less for they are not dependent on the state of the network. This architecture was chosen based on several reasons:

- Masters [1993] states that the multilayer feedforward neural network is a universal function approximator and can thus, theoretically at least, teach anything learnable to the network;
- Masters [1993] states that if real-time processing is needed, a multilayer feedforward network may be the only practical choice for its execution speed is among the fastest of all models currently in use;
- Ridsdale [1990] stated that a feedforward ANN is thought by many to be a good model for how learning actually takes place in the nervous system.

A disadvantage of a multilayer feedforward network is the lack of a fast and reliable training algorithm. The error back-propagation learning algorithm presented in the next section usually converges to the nearest minimum, but still can be slow. There is no guarantee that the achieved minimum is global. While the learning algorithm can be slow during training, it does not affect execution performance during operation. In general, the training time is orders of magnitude less than the projected operation time, and can thus be considered negligible.

Once a feedforward ANN architecture is chosen, a difficulty arises in finding a good match between an ANN layout and the corresponding research problem. Researchers are still trying to develop a general-purpose ANN architecture (Barhen et al. [1989] and McClelland et al. [1988]). The most commonly used general-purpose layout is called the back-propagation ANN. This network is composed of three layers: an input layer; zero or more hidden layers; and an output layer. Neurons in one layer receive input from all the neurons in the previous layer, and send their outputs to all the neurons in the next layer. All the neurons in the back-propagation ANN have the same activation function.

The purpose of a back-propagation ANN is to learn to generate an arbitrary function given a set of examples. The first step in designing a back-propagation ANN is to determine the form of the function to be learned; thus, the number of inputs and the number of outputs needs to be determined. The input layer has one element for

each input; the output layer has one element for each output. The number of hidden layers, and the number of elements in these layers, both depend on the complexity of the function to be learned. In general, more layers with more elements will allow the ANN to learn a more complex function to a desired degree of accuracy. More layers with more elements also cause slower learning and slower retrieval. Beyond these generalizations, little is yet known. In practice, considerable experimentation is needed to find the optimal layout.

There are several drawbacks associated with using an ANN. First, given the specifications of a problem, it is very difficult to specify an effective 'a priori' architecture. Secondly, after an ANN has been trained, intuitive understanding of how it works may be difficult. Thirdly, there is a supposition that the ANN will work correctly when presented with any possible input within a problem space. Techniques for strict mathematical verification of an ANN's performance are still being developed by many researchers. An ANN that has been well trained for a specific task and verified with a reasonable testing set will perform well in practice. Performance quality is simply difficult to prove at this time.

5.3.2 Error Back-propagation Training Algorithm

The operation of an ANN involves two processes: learning and recall. Learning is the process of adapting the connection weights in response to the input values. The ANN "learns" based on a learning rule that governs how the connection weights (\mathbf{w}) are modified because of the input values (\mathbf{z}) (see section 2.2). Recall is the process of taking an input and producing a response based on what the ANN learned.

The learning algorithm applied in this research is a general delta-rule learning algorithm, also known as the error back-propagation training algorithm (Rumelhart et al. [1986]). Training involves multiple cycles through the training data during which the weights are modified to reduce the error between the ANN output and the desired output. The cumulative error is calculated using a squared-error cost function, and this function is defined as the difference between the desired output and actual output, squared and summed over all patterns. The standard error back-propagation training algorithm can be summarized as follows:

1. One pattern is presented from the training data to the ANN and the ANN computes the output.
2. The output from the ANN is compared to the desired output.
3. An error signal is calculated using a negative gradient descent technique.
4. The error signal is propagated to both the input and hidden layers, and connection weights are modified.

In this research, the author modified the standard error back-propagation algorithm by dividing the last part into two steps:

4. The error signal is propagated to the input layer and connection weights are modified.
5. Parts 1-3 are repeated for the same pattern, and the error signal is propagated to the hidden weight layer and connection weights are modified.

As noted in Samad [1988] and tested on a set of ANN architectures in this research, this method speeds convergence greatly. Since the essence of the learning algorithm is to evaluate the contribution of each particular weight to the output error, the above modification allows the algorithm to approximate the contribution of the layers separately. This modification decreases the total number of passes through all the cases.

Once all cases have been presented to the ANN, a cycle has been completed and the summed error generated by this cycle is compared to the user's desired maximum error. If the summed error is still greater than the desired error, the ANN continues to cycle through the training data until the desired error is attained. The corresponding computer procedure is located in Appendix B.

5.4 Artificial Hand Neural Network

Once the databases had been developed, attention centered towards creating and testing ANNs to control tendons. The artificial hand neural network (AHNN) was divided into two ANNs: a finger flexion and extension neural network (FFENN); and a finger abduction and adduction neural network (FAANN). This section describes in detail these neural networks, including current results.

5.4.1 Finger Flexion and Extension Neural Network

The FFENN controls finger flexion and extension. To create a specified movement or animated goal, the output of the FFENN calculates which tendon(s) must be activated, and the amount of tendon-length change needed by the flexors and extensors in the finger model. The development, training, and testing of the FFENN is described next.

5.4.1.1 FFENN Architecture

The FFENN is a multilayer feedforward, fully connected network consisting of a set of neurons that are arranged into three layers consisting of: ten neurons in the input layer corresponding to two sets of input data; nine neurons in the hidden layer; and five neurons in the output layer. Figure 5-1 depicts the FFENN. The first set of the input data corresponds to the desired resultant tendon lengths of the FDP, FDS, DOR, EDC, and INTR when the movement will have been completed (Table 5-4). The second set of the input data corresponds to the current tendon lengths of the FDP, FDS, DOR, EDC, and INTR at the time of the requested command. The hidden layer, consisting of nine neurons not directly corresponding to any tendons, is required for the FFENN to make the necessary correlations between the input and desired output of the FFENN. Several configurations ranging from 6 to 15 hidden nodes were tested to determine an optimal size. If the ANN fails to converge to a solution, then more hidden nodes would be required. If the ANN does converge, fewer hidden nodes should be tested for they may suffice for convergence. The output layer consists of five neurons and corresponds to the tendon activation layer. The activation function used was a modified unipolar continuous function:

$$F(\text{net}) = 1.02 / (1 + e^{-\text{net}})$$

where $\text{net} = \sum w_i z_i$ (see section 2.2). The activation function was modified because it was noted during testing that the FFENN had problems reaching either end of the unipolar continuous range of (0, 1) within an acceptable limit. Thus for FFENN to reach (0, 1) within an acceptable range, the activation function was increased to a range of (0, 1.02). A sigmoidal activation function was chosen to ensure a desirable

normalization, since the output layer computed the amount of tendon-length change as a percentage. In Zurada [1992], the unipolar continuous activation function contains a steepness factor λ . λ was chosen to be linear and set equal to 1.0 since the range of the desired outputs of the ANN are evenly distributed in finger postures ranging from fully flexed to fully open.

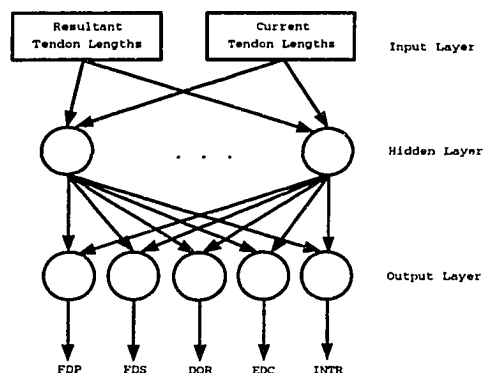


Figure 5-1. Finger flexion and extension artificial neural network.

5.4.1.2 Input-Output Data

Finger flexion and extension cases, listed in Table 5-1, were used to train the FFENN. Two batches of input-output sets were created. The first input-output set for the FFENN only included cases 1-7, 9, 12-13, and 15-22 because these were the cases where only one specific tendon trigger was required for a specified movement. The other cases (8, 10, 14, 23-25) were not included in either the training or testing sets since these cases can be derived from a combination of two or more other cases. Once the FFENN was trained and tested with this input-output set, it was noted by the author that the FFENN could not create certain finger end-positions that required opposing tendons to be manipulated. Thus, a second input-output set was created to include all cases listed in Table 5-1 since some of these cases required manipulating opposing tendons simultaneously. The gathering of empirical data for cases where more than one tendon is required for a specified movement was problematic. Since one tendon affects all other tendons, it was difficult to measure the amount of tendon-length change in just one tendon in isolation. An assumption was made that re-

quires flexors always to move before extensors, and thus resolves the difficulties of gathering empirical data when opposing tendons are involved.

The FFENN's inputs and outputs have been normalized between 0.0 and 1.0 to allow for the normalized manipulation of different finger lengths. The hand-muscle programme described in section 4.6 was modified to aid in gathering data based on the cases listed in Table 5-1. The author used the hand-muscle programme to place fingers in postures denoted under the Position Change From column; the automated process was then used to gather tendon-length information on different postures between this position and the posture denoted under the Position Change To column in Table 5-1. This allowed for an automated process of data collection for the training and testing sets; however, once the data was collected, the author imported this data into a spreadsheet programme to reformat the data according to the specifications of the FFENN. Current tendon lengths were gathered from the automated process of the hand-muscle programme based on case number. The resultant lengths were replicated for each current tendon length, and the desired output of the FFENN was calculated. The desired output included which tendon(s) must be activated, and the amount of length change needed by the flexors and extensors in the finger model. Once the FFENN data was in the appropriate format, duplicate patterns were eliminated from the testing data. From the testing data, patterns were chosen to create the training data. The first set of input-output data contains 826 testing patterns, of which 317 patterns comprised the training data. The second set of input-output data contains 1634 patterns, of which 756 patterns comprised the training data. Appendix C1 depicts an excerpt from the input-output data set.

5.4.1.3 Training the FFENN

All data may be used to train the FFENN; however, a small subset is often all that is needed to train a network successfully. The previous section discussed the creation of the training and testing data. The testing data is used to verify the training of the FFENN on input data that it may not have processed during training. The training data should cover the entire expected input space. The training data used for the FFENN contains 317 patterns for flexion and extension of an animated finger.

Weights were initialized to random values between -0.5 and +0.5. An error back-propagation training algorithm was implemented as stated in section 5.3.2.

5.4.1.4 FFENN Results

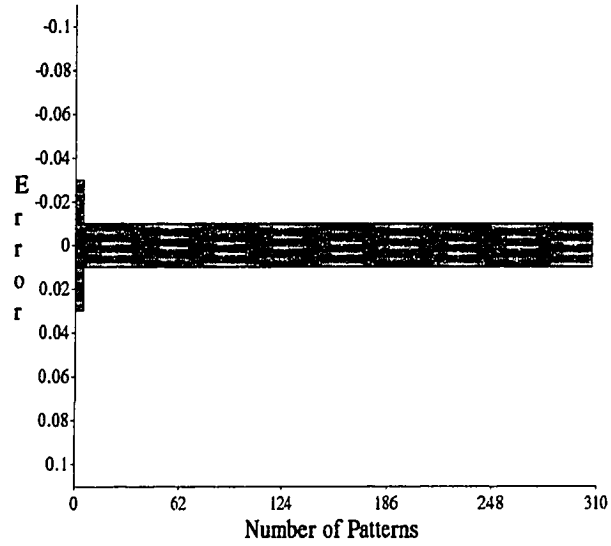
There are three variables needed in the training of the FFENN: the number of nodes in the hidden layer (H); the learning rate parameter (N); and the maximum error desired by the trainer (EMAX). The number of nodes in the hidden layer must be set in constructing the FFENN. As noted previously (Section 5.4.1.1), a hidden layer of nine nodes performed efficiently.

The selection of a value for the learning rate parameter affects the network performance. Experiments to determine a range for N were conducted on the FFENN, and it was found that N should be a number between 0.01 and 0.2 to ensure that the network will converge to a solution. A small value of N means that the network may have to make a large number of iterations, and any large N value may prevent the FFENN from converging. From the experiments, smaller values of N resulted in a convergent performance, noting that $N=0.1$ seemed efficient. As noted by Zurada [1992], N may be modified during training to speed convergence of the ANN; however, in this training algorithm, N remained constant.

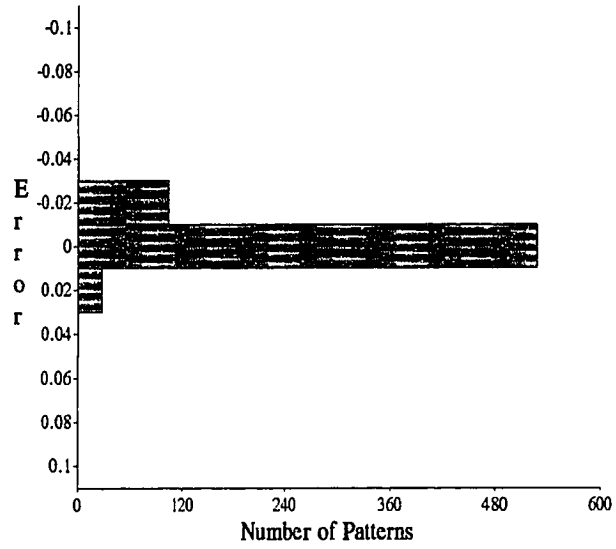
Finally, the maximum error desired by the trainer was set to 0.002 since an error of 0.2% in the extension or flexion of a finger cannot be visually detected during animation and the FFENN converged to this error.

5.4.1.5 Test Data Performance

The error between the desired and actual output was computed for all patterns in the testing data, and the errors were plotted in histograms depicted in Figure 5-2. Figure 5-2a represents the overall performance error on the training data. The FFENN computed the proper tendon-length change for 307 patterns (from a total of 317 patterns) within an error range of $\pm 1\%$. All 317 patterns fell inside the $\pm 3\%$ error range. Figure 5-2b represents the overall performance error on the test data. The FFENN computed the proper tendon length change for 527 patterns (from a total of 826 patterns) within an error range of $\pm 1\%$. Only four patterns were beyond the $\pm 3\%$ error range, but were within the $\pm 5\%$ error range.



a.)



b.)

Figure 5-2. Overall performance error on the a.) training set. b.) testing set.

5.4.1.6 Discussion and Future Direction

From Figure 5-2, we were able to attain information on the accuracy of the FFENN on both training and testing data. The testing set allowed the author to notice certain input patterns with which the network had difficulties and to address those problems individually. Such problems included finger patterns that required manipulations of opposing tendons. The author wanted to be able to recognize what the FFENN still could not solve. Usually, this required adding the problematic finger patterns into the training set.

5.4.2 Finger Abduction and Adduction Neural Network

The FAANN controls finger abduction and adduction. To create a specified movement or animated goal, the output of the FAANN calculates which tendon must be activated, and the amount of length change needed by the palmar and dorsal interosseous in the finger model. This section describes the development, training, and testing of the FAANN.

5.4.2.1 FAANN Architecture

The FAANN architecture is similar to the FFENN architecture discussed in section 5.4.1.1. The FAANN is a multilayer feedforward, fully connected network consisting of a set of neurons that are arranged into three layers, consisting of: four neurons in the input layer corresponding to two sets of input data; five neurons in the hidden layer; and two neurons in the output layer. Figure 5-3 depicts the FAANN. The first set of the input data corresponds to the desired resultant tendon lengths of the PI and DI when the movement will have been completed (Table 5-2). The second set of the input data corresponds to the current tendon lengths of the PI and DI at the time of the requested command. The hidden layer, consisting of five neurons not directly corresponding to any tendons, is required for the FAANN to make the necessary correlations between the input and desired output of the FAANN. Similar to the FFENN, several experiments were conducted to evaluate the size of the hidden layer and experiments of hidden nodes ranging in number from 2 to 7 were conducted to determine an acceptable size of the hidden layer. The output layer consists of two neurons

and corresponds to the tendon activation layer. The activation function used was a modified unipolar continuous function discussed in section 5.4.1.1.

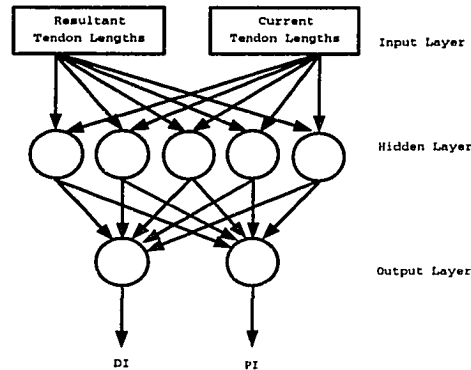


Figure 5-3. Finger abduction and adduction artificial neural network.

5.4.2.2 Input-Output Data

Similar to the input-output data sets that were created for the FFENN, data sets were also created for the FAANN in a comparable manner as discussed in section 5.4.1.2. Finger abduction and adduction cases, listed in Table 5-2, were used to train FAANN. The input-output set for the FAANN contains 1308 testing patterns, of which 328 patterns comprised the training data. Appendix C2 depicts an excerpt from the input-output data set.

5.4.2.3 Training the FAANN

Similar to the training data for the FFENN, the training data for the FAANN should cover the entire expected input space. The training data used for the FAANN contains 328 patterns for abduction and adduction of an animated finger. Weights were initialized to random values between -0.5 and +0.5. An error back-propagation training algorithm was implemented as stated in section 5.3.2.

5.4.2.4 FAANN Results

There are three variables in training the FAANN: the number of nodes in the hidden layer (H); the learning rate parameter (N); and the maximum error desired by the trainer (EMAX). These variables were discussed in section 5.4.1.4. For the

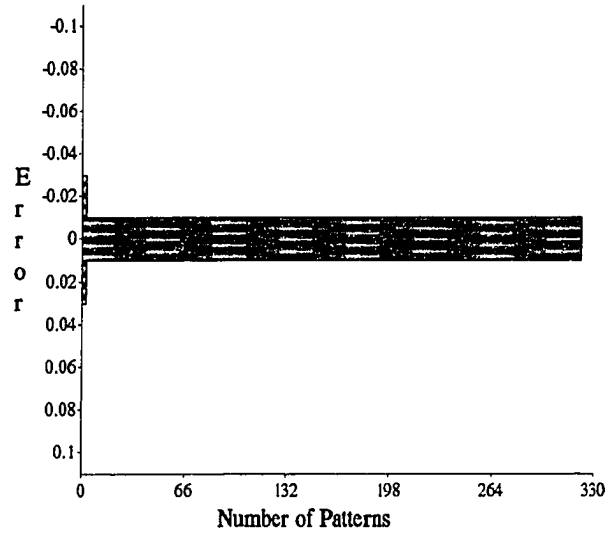
FAANN, these variables were set to the following values: $H = 5$; $N = 0.1$; and $EMAX=0.001$. $EMAX$ was set to 0.1% since an error of this size in the abduction or adduction of a finger cannot be visually detected during animation, and the FAANN was able to train to a lower error rate than the FFENN in the same length of time.

5.4.2.5 Test Data Performance

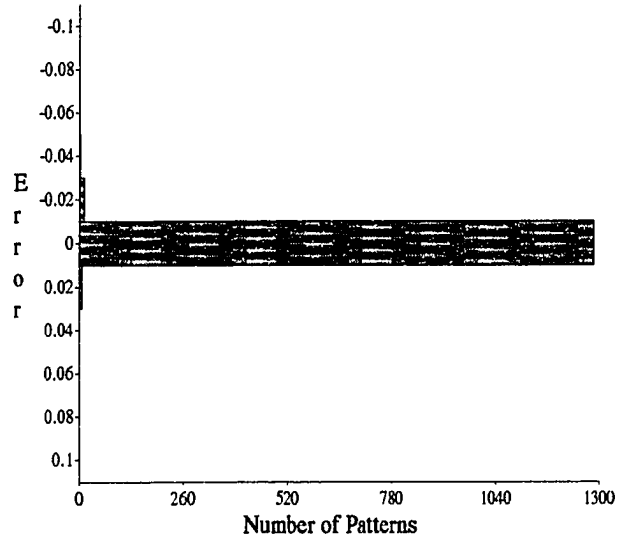
The error between the desired and actual output was computed for all patterns in the testing data, and the errors were plotted in histograms depicted in Figure 5-4. Figure 5-4a represents the overall performance error on the training data. The FAANN computed the proper tendon-length change for 322 patterns (from a total of 328 patterns) within an error range of $\pm 1\%$. All 328 patterns fell inside the $\pm 3\%$ error range. Figure 5-4b represents the overall performance error on the testing data. The FAANN computed the proper tendon length change for 1294 patterns (from a total of 1308 patterns) within an error range of $\pm 1\%$. No patterns were outside the $\pm 3\%$ error range.

5.4.2.6 Discussion and Future Direction

From Figure 5-4, we were able to attain information on the accuracy of the FAANN on both training and testing data. The testing set allowed the author to notice certain input patterns with which the network had difficulties and to address those problems individually. No such difficulties occurred in the FAANN as in the FFENN.



a.)



b.)

Figure 5-4. Overall performance error on the a.) training set. b.) testing set.

5.4.3 Summary of the AHNN

In this section, the FFENN and FAANN were developed for finger movement. Once these neural networks were trained and tested, they were reproduced for each finger and placed within a large hierarchy, as depicted in Figure 5-5.

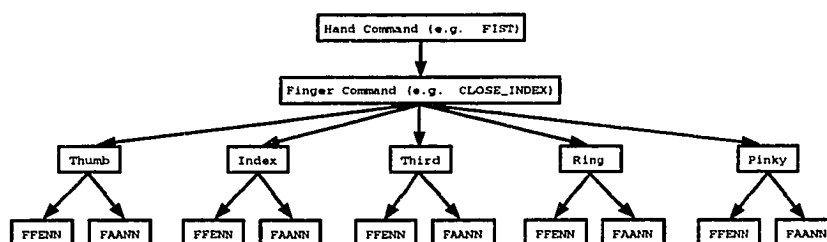


Figure 5-5. Architecture of the Artificial Hand Neural Network.

5.5 Computer Implementation

Two computer programmes were created to incorporate the AHNN. The first computer programme is a modified version of the hand-muscle programme described in section 4.6. This modified programme allowed the author to test more finger patterns that can be processed by the FFENN and FAANN, since the testing data only contained a subset of the entire expected input space. The second computer programme incorporates an animated hand that has learned a subset of the American Sign Language. Both of these programmes are described in the following section.

5.5.1 Modified Hand-Muscle Programme

A testing set should cover the entire input space, but for the FFENN and FAANN this is impossible since the input space is vast. Thus to test more situations and visually detect problems with the networks, the AHNN was incorporated into an animation programme to view the end-position results. When problems were detected (as mentioned in sections 5.4.1.6 and 5.4.2.6), the input patterns were incorporated into both training and testing data. Once these new problematic patterns have been incorporated into the training data, the networks were retrained and tested to note if the networks improved in their performance on these patterns.

A menu option was created to allow a user to send individual hand and finger commands to the AHNN. The animator could manually manipulate the tendons as depicted in section 4.6.5 and then send either a hand command or finger command to the AHNN. This allows the animator to test visually the results of the AHNN for a vast number of finger tendon lengths. An example screen shot of this programme is depicted in Figure 5-6.

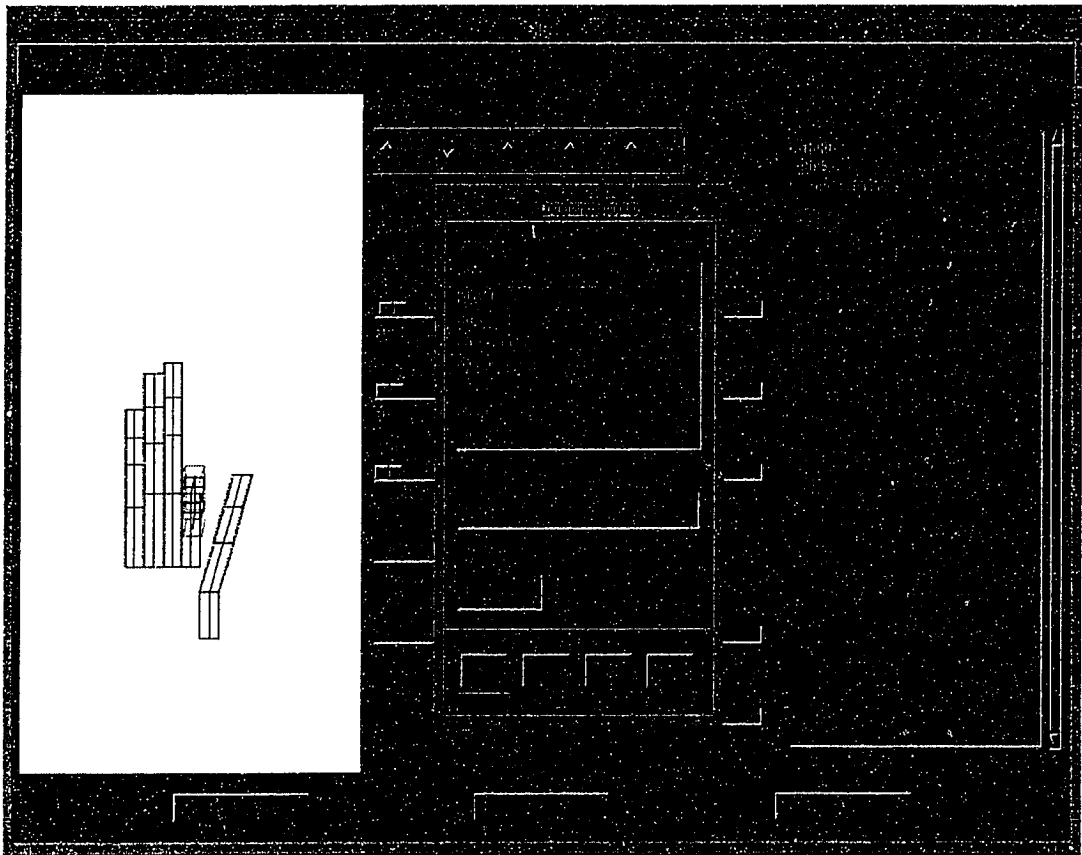


Figure 5-6. Screenshot of the modified hand-muscle programme.

5.5.2 Signing Programme

The ability for an ANN to manipulate the muscles and tendons of the human hand is a very important step in the development process of dynamically displaying finger movements. As a simple example, the author has presented an American Sign Language (signing fingers) demonstration displaying the capabilities of the AHNN in determining the end positions of the thumb and fingers.

As the programme begins executing, an initial screen appears as shown in Figure 5-7. The hand is generated in the top part of the screen. The scene can be rotated in the 3D environment using the mouse buttons for x-axis, y-axis, and z-axis rotation. Mouse movements and rotations were described in detail in section 4.6.4.

The user can type letters in the input space provided at the bottom of the screen. Upon pressing the return key, the letters that were typed by the user will be signed by the fingers.

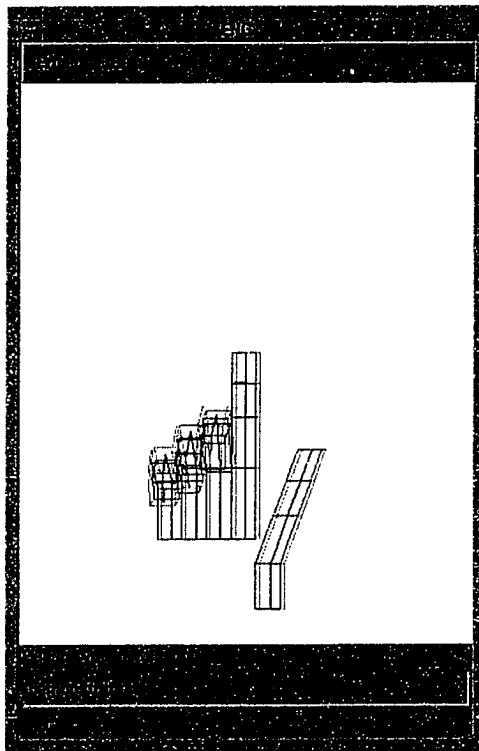


Figure 5-7. The initial screen of the ASL program.

5.6 Chapter Summary

In this research, an approach for creating ANNs to manipulate an animated human hand based on tendon control is described and tested. Three databases were created: a Hand Commands Database; a Finger Commands Database; and a Finger Tendon-Length Database. The Hand Commands Database is a dynamic database that contains hand commands. The Finger Commands Database contains the eight positions described in section 5.1: CLOSE; OPEN; FF-DIP-STR; HOOK; V-SHAPE; POINT-DOWN; ABDUCT; and ADDUCT. The Finger Tendon-Length Database describes the eight finger commands in terms of normalized tendon lengths. The AHNN contains five sets of ANNs, with each set containing a FFENN and a FAANN. The FFENN was developed and tested to flex and extend an animated finger. The FAANN was developed and tested to abduct and adduct an animated finger. The results of these ANNs produced correct tendon control and this could be viewed in our animation programme. Using the cases described in Table 5-1 and Table 5-2, entire hand motion can be achieved and this is an important step in AIVH development.

6. Conclusions and Future Work

He who chooses the beginning of a road chooses the place it leads to. It is the means that determines the end.
- Harry Emerson Fosdick

In this research, the ability for artificial neural networks (ANNs) to determine tendon control in the hand has been developed and tested, producing positive results. Feedforward ANNs manipulated an underlying tendon hand model in real time and determined end positions of the thumb and fingers to allow for free hand animation.

A wide variety of motion control techniques were described in section 2.1. These techniques are used and combined to produce required animated movements. As human models increase in complexity, new motion control algorithms will be created to attain realistic human movement. Section 2.2 provided an overview of ANNs. ANNs provide an alternative approach to solving many problems and may be able to provide a new approach in animation motion control. Section 2.3 provided an overview of how ANNs have been applied for muscle activity, indicating that an ANN could learn how to trigger tendons and muscles to produce certain movements. In this research, the author applied ANNs to trigger tendons to allow for animation of the thumb and finger movements.

As discussed in the first part of chapter 2, a classification scheme based on motion control methods was presented, and included the following categories: geometric; physical; and behavioural. The model created in this thesis would be included in the behavioural methods category, and the model also would be included in the task-level method since an action is specified only by its effects on objects, in this case the thumb and fingers. For example, when an animated hand is given the command *FIST*, this command is translated into low-level instructions. These instructions correspond to finger commands, such as *CLOSE_INDEX*, *CLOSE_THIRD*, *CLOSE_RING*, *CLOSE_PINKY*, and *CLOSE_THUMB*. Each of these commands is further translated into tendon control.

As discussed in the second part of chapter 2, ANNs provide an alternative approach to many problems and may be able to provide a new approach in animation

motion control. ANNs were incorporated into an animation programme to determine in real time the final positions of human hand postures. ANNs have not been applied to animate human movement, but their potential is promising in this area.

As discussed in the last part of chapter 2, ANNs have been applied for muscle activity, indicating that an ANN could learn to predict muscle activity for certain movements. In this thesis, ANNs were trained to control tendons of an animated hand.

In chapter 4, a detailed description of a human hand was given. There are many muscles and tendons in a hand; however, only a subset of these muscles were modelled. These motor muscles were implemented in a programme (section 4.6) for a user to manually manipulate, but there are other muscles that were implemented within the programme code that the user does not manipulate. Also provided in this chapter is a flavour of the mathematical calculations required to appropriately affect the joints of the thumb and fingers.

Once the underlying tendon model was developed and tested, ANNs could be created to manipulate tendon control. This is the main focus of chapter 5 and the goal of this thesis. An understanding of hand movements was needed before developing ANNs to control the thumb and fingers, resulting in Table 5-1. The first 14 cases depicted in this table were extracted from work done by Landsmeer et al. [1965] and Long et al. [1964]. The remainder of the cases in Table 5-1 and the two cases in Table 5-2 were determined by the author to complete all possible finger movement combinations. Based on hand motion classification and finger motion patterns, there were three databases developed for this research: a Hand Commands Database; a Finger Commands Database; and a Finger Tendon-Length Database. These databases were created by the author since such databases do not exist in other related works or research areas. In particular, the Finger Tendon-Length Database that describes the eight finger commands in terms of normalized tendon lengths was important in allowing ANNs to determine tendon control. Using the modified hand-muscle programme described in section 4.6 and 5.5.1, finger patterns were created on which to train and test the neural networks. Once the networks had been trained, they were incorporated into an animation system for further visual verification of finger postures, since humans are better at detecting problems visually. The results of the networks produced

appropriate end positions. In chapter 5, a demonstration of an American Sign Language programme was created to exhibit the neural networks' ability to manipulate the tendons successfully that could allow for real-time motion and would display finger movements.

There are several accomplishments in this thesis. First, the most complex and versatile part of the human body was modelled and controlled. Secondly, ANNs were developed, trained, and tested to learn the correlation between finger postures and tendon activations. This allows ANNs to manipulate in real time the animated thumb and fingers. Now that ANNs are seen to be successfully applied to hands, then the next step is to apply this technique to the entire body. The aspect of applying ANNs is new to the field of computer animation, but promising results can be obtained by applying ANNs in this field. Thirdly, this research points to further developments in fields of biomechanics, orthopedics, and medicine. In biomechanics and orthopedics, this research can be applied as a tool for developing and testing of robotic manipulators and prosthetics. In the field of medicine, researchers can use this approach as a basis to question human motion problems. Finally, this model can be integrated with existing models for animated human movement. Since it manipulates an underlying skeleton model, it can be integrated into existing hand animations that also manipulate a skeleton model. The execution speed of ANNs can be increased if the ANNs are integrated into hardware components, thus real-time performance in virtual reality environments can be achieved.

In general, sophisticated animation techniques are required to create and control virtual humans. Virtual human modelling must include the structure needed for virtual human animation. When linking modelling and animation, human figure animation is complex because the human body is capable of making vast and innumerable combinations of shape positions and interacts with its environment in many different ways. A computer model for a synthetic character could try to mimic a human by creating ANNs that are trained to accomplish a set of movements. The goal of this research was to create ANNs to control tendons in an animated human hand. The hand model developed is based on a human hand, and the movement control of the thumb and fingers is based on muscle control. Using this model, back-propagation neural networks were successfully developed and trained to manipulate free thumb

and finger patterns, and this is an important step in AIVH development.

This research is an ongoing endeavour. The next step to take within this area is to create an ANN that can learn the relationships between the hand commands and the finger commands, thus eliminating the need for these databases as stated in section 5.2.2. This idea would be used for an AIVH to interact with its environment in such ways as touching or grasping. The next logical step would be to include resisted motion, such as power gripping, precision handling, and pinching. This step involves many different aspects of computer animation, such as collision detection, object identification, and environment awareness.

7. References

- Ascension Technology Corporation, (1998), <http://www.ascension-tech.com>, January 24, 1999
- Badler, N.I., Phillips, C.B., and Webber, B.L., (1993) Simulating Humans: Computer Graphics Animation and Control, Oxford University Press, New York, N.Y.
- Barhen, J., Gulati, S., and Zak, M., (1989) Neural Learning of Constrained Nonlinear Transformations, *IEEE Computer*, Volume 22, Issue 6, pp. 67-76
- Bean, J.C., Chaffin, D.B., and Schultz, A.B., (1988) Biomechanical model calculation of muscle contraction forces: a double linear programming method, *Journal of Biomechanics*, Volume 21, pp. 59-66
- Boulic, R., Capin, T., Huang, Z., Mocozet, L., Molet, T., Kalra, P., Lintermann, B., Magnenat-Thalmann, N., Pandzic, I., Saar, K., Schmitt, A., Shen, J., and Thalmann, D., (1995) The HUMANOID Environment for Interactive Animation of Multiple Deformable Human Characters, *Proc. Eurographics*, August, pp. 337-348
- Boulic, R., Thalmann, D., and Magnenat-Thalmann, N., (1990) A Global Human Walking Model with Real Time Kinematic Personification, *The Visual Computer*, Volume 6, Number 6, pp. 344-358
- Cavazza, M., Earnshaw, R., Magnenat-Thalmann, N., and Thalmann, D., (1998) Motion Control of Virtual Humans, *IEEE Computer Graphics and Applications*, Volume 18, Number 5, pp. 24-31
- Chadwick, J.E., Haumann, D.R., and Parent, R.E., (1989) Layered construction for deformable animated character, *Computer Graphics*, Volume 23, Number 3, pp. 243-252
- Chase, R.T., (1973) Atlas of Hand Surgery, W.B. Saunders Co., Philadelphia, USA

- Crowninshield, R.D., and Brand, R.A., (1981) A physiologically based criterion of muscle force prediction in locomotion, *Journal of Biomechanics*, Volume 14, pp. 793-801
- Delaney, B., (1998) *The Mystery of Motion Capture*, ed: Michael J. Potel, *IEEE Computer Graphics and Applications*, Volume 18, Number 5, pp. 14-19
- Foley, J.D., van Dam, A., Feiner, S.K., and Hughes, J.F., (1990) Computer Graphics Principles and Practice, Second Edition, Addison-Wesley Publishing Company, Don Mills, Ontario
- Haykin, S., (1994) Neural Networks A Comprehensive Foundation, Macmillan College Publishing Company, New Jersey
- Hearn, D., and Baker, M.P., (1994) Computer Graphics, Second Edition, Prentice Hall, Englewood Cliffs, New Jersey
- Hughes, R.E., (1991) Empirical evaluation of optimization-based lumbar muscle force prediction models, Ph.D. Dissertation, University of Michigan, Ann Arbor, MI, USA
- Jain, A.K., Mao, J., and Mohiuddin, K.M., (1996) Artificial Neural Networks: A Tutorial, *IEEE Computer*, Volume 29, Number 3, March, pp. 31-44
- Kapandji, I.A., (1970) The Physiology of the Joints. Annotated diagrams of the mechanics of the human joints., Volume 1, Longman Group Ltd., London, G.B.
- Korein, J.U., and Badler, N.I., (1982) Techniques for Generating the Goal-Directed Motion of Articulated Structures, *IEEE Computer Graphics and Applications*, November, pp. 71-81
- Kuchar, O.A., (1996) *Finger Movements on a Keyboard: An Animated Simulation of the Biomechanics of the Hand*, Master thesis, Technical University of Nova Scotia, Halifax, N.S., Canada
- Landsmeer, J.M.F., and Long, C., (1965) The Mechanism of Finger Control, Based on Electromyograms and Location Analysis, *Acta anat*, Volume 60, pp. 330-347

- Lee, J., and Tosiyasu, L.K., (1995) Model-Based Analysis of Hand Posture, IEEE Computer Graphics and Applications, Volume 15, Issue 5, pp. 77-86
- Long, C., and Brown, M.E., (1964) Electromyographic kinesiology of the hand: Pt. III. Lumbricalis and flexor digitorum profundus to the long finger., Arch. phys. med., Volume 43, pp. 450-460
- Magnenat-Thalmann, N., Laperrière, R., and Thalmann, D., (1988) Joint-Dependent Local Deformations for Hand Animation and Object Grasping, Proc. Graphics Interface, Edmonton, pp. 26-33
- Magnenat-Thalmann, N., and Thalmann, D., (1996) Computer Animation, Handbook of Computer Science, CRC Press, pp. 1300-1318
- Magnenat-Thalmann, N., and Thalmann, D., (1991) Complex Models for Animating Synthetic Actors, IEEE Computer Graphics and Applications, Volume 11, Issue 5, pp. 32-44
- Magnenat-Thalmann, N., and Thalmann, D., (1990) Computer Animation: Theory and Practice, 2nd Edition, Springer-Verlag Berlin Heidelberg, New York
- Mas, R., and Thalmann, D., (1994) A Hand Control and Automatic Grasping System for Synthetic Actors, Proc. Eurographic, pp. 167-178
- Masters, T., (1993) Practical Neural Network Recipes in C++, Academic Press, Inc. Toronto, Canada
- McClelland, J.L., and Rumelhart, D.E., (1988) Explorations in Parallel Distributed Processing, Bradford Books, Cambridge, Mass.
- Müller, B., Reinhardt, J., and Strickland, M.T., (1995) Neural Networks An Introduction, Second Updated and Corrected Edition, Springer-Verlag Berlin Heidelberg, New York
- Nussbaum, M.A., Chaffin, D.B., and Martin, B.J., (1995) A Back-propagation Neural Network Model of Lumbar Muscle Recruitment During Moderate Static Exertions, Journal of Biomechanics, Volume 28, Number 9, pp. 1015-1024

- Nussbaum, M.A., Martin, B.J., and Chaffin, D.B., (1997) A Neural Network Model for Simulation of Torso Muscle Coordination, *Journal of Biomechanics*, Volume 30, Number 3, pp. 251-258
- Polhemus, (1997), <http://www.polhemus.com>, January 24, 1999
- Rao, D.H., and Kamat, H.V., (1996) Artificial Neural Networks for the Emulation of Human Locomotion Patterns, *Proceedings of the First Regional Conference, IEEE Engineering in Medicine and Biomechanics Society and 14th Conference of the Biomedical Engineering Society of India. An International Meeting.*, pp. 2/80-2/81
- Reynolds, C., (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, *Proc. SIGGRAPH 1987, Computer Graphics*, Volume 21, Number 4, pp. 25-34
- Ridsdale, G., (1990) Connectionist Modelling of Skill Dynamics, *Journal of Visualization and Computer Animation*, Volume 1, pp. 66-72
- Rijkema, H., and Girard, M., (1991) Computer Animation of Knowledge-Based Human Grasping, *ACM Computer Graphics*, Volume 25, Number 4, pp. 339-348
- Rosenblum, L.J., and Macedonia, M.R., (1997) Phantom-Based Haptic Interaction with Virtual Objects, *IEEE Computer Graphics and Applications*, Volume 17, Number 5, pp. 6-10
- Rumelhart, D.E., Hinton, G.E., and Williams, R.J., (1986) Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1, pp. 318-362
- Samad, T., (1988) Backpropagation is Significantly Faster if the Expected Value of the Source Unit is Used for Update, *Conference of the International Neural Network Society*
- Seireg, A., and Arvikar, R., (1989) Biomechanical Analysis of the Musculoskeletal Structure for Medicine and Sports, Hemisphere Publishing Corp., USA
- Sepulveda, F., Wessa, D.M., and Vaughan, C.L., (1993) A neural network representation of electromyography and joint dynamics in human gait, *Journal of Biomechanics*, Volume 26, pp. 101-110

- Sherwood, L., (1992) Muscle Physiology, Human Physiology From Cells to Systems, 2nd Edition, Chapter 8, West Publisher, Minnesota, USA
- Srinivasan, S., Gander, R.E., and Wood, H.C., (1992) A Movement Pattern Generator Model Using Artificial Neural Networks, IEEE Transactions on Biomedical Engineering, Volume 39, Number 7, pp. 716-722
- Tascillo, A., Skormin, V., and Bourbakis, N., (1993) Neurofuzzy Grasp Control of a Robotic Hand, IEEE, pp. 507-516
- Thalmann, D., (1990) Robotics Methods for Task-level and Behavioral Animation, in: Thalmann, D. (ed) Scientific Visualization and Graphics Simulation, John Wiley and Sons Ltd., New York, USA, pp. 129-147
- Thalmann, N.M., and Thalmann, D., (1987) The Directions of Synthetic Actors in the Film Rendezvous à Montréal, IEEE Computer Graphics and Applications, Volume 7, Number 12, pp. 9-19
- Tubiana, R., (1981) The Hand, Volume 1, W.B. Saunders Company, Toronto, Canada
- Tubiana, R., Thomine, J., and Mackin, E., (1996) Examination of the Hand and Wrist, Martin Dunitz Ltd., London, England
- Uhrig, R.E., (1995) Introduction to Artificial Neural Networks, Proceedings of the IEEE IECON 21st International Conference on Industrial Electronics, Control, and Instrumentation, Volume 1, pp. 33-37
- VICON, (1998), <http://www.vicon.com>, January 24, 1999
- Warwick, R., Williams, P.L., and assoc., (1973) Gray's Anatomy, 35th Edition, Longman Group Ltd., Harlow, England
- Wilhelms, J., (1987) Using Dynamic Analysis for Realistic Animation of Articulated Bodies, IEEE Computer Graphics and Applications, June, pp. 12-27
- Zeltzer, D., (1985) Towards an Integrated View of 3D Computer Animation, The Visual Computer, Volume 1, Number 4, pp. 249-259

Zurada, J.M., (1992) An Introduction to Artificial Neural Systems, West Publishing Company, St. Paul, MN

8. Appendices

These appendices provide code and data fragments for this thesis. All programmes in this thesis were written in the C language. All programmes execute on a DEC ALPHA, using Motif for the user interface and Mesa for the computer graphics. This annex contains the following appendices:

- A. Code fragment for the FDP tendon;
- B. Code fragment for the error back-propagation training algorithm.
- C. Excerpts from the input-output data sets for:
 - C1. Finger Flexion and Extension Neural Network
 - C2. Finger Abduction and Adduction Neural Network

Appendix A

Flexor Digitorum Profundus Program Module

```

void FDP_tension (double new_length, int fin_num)
{
    /* calculate the angles at the joints for the tension given
       (new_length) */
    int i;          /* counter for joint */

    double opp, hyp, alpha, beta, theta, omega, height;
    /* opp - opposite in a triangle; hyp - hypotenuse, angles - alpha,
       beta, theta, omega; height - height of triangle */

    struct Vector distal_midpt, middle_midpt, temp, temp1, temp2,
        temp3;
    /* distal_midpt - midpoint on distal phalanx; middle_midpt -
       mid-point on middle phalanx, temporary variable - temp, temp1,
       temp2, temp3 */

    int axis;          /* axis of rotation */
    double sag_angle; /* saggital plane */
    int MCPflex = FALSE; /* flag for flexion of the MCP joint */

    if (fin_num == THUMB)
        axis = 3;      /* different rotation than the fingers */
    else
        axis = 2;

    for (i=deg_DIP; i<=deg_MCP; i++)
    {
        if (finger[fin_num].joints[i].curr_angle !=
            finger[fin_num].joints[i].max_angle)
        {
            if ((i == deg_DIP) &&
                (finger[fin_num].joints[deg_PIP].curr_angle >= 90.0)
                &&
                (finger[fin_num].joints[deg_MCP].curr_angle >= 90.0))
            {
                /* cannot flex DIP if PIP and MCP joints are
                   flexed */
                fprintf(stderr, "NOT flex DIP if PIP and MCP are
                    bigger than 90\n");
            }
        }
    }
}

```

```

        else
            break;
    } /* if curr_angle != max_angle */
} /* for */

if ((i <= deg_MCP) &&
    (new_length < finger[fin_num].fdp.max_tension)&&
    (new_length < finger[fin_num].fdp.tension))
{
    if (fin_num != THUMB)
    {
        /* check if finger is in it's saggital plane */
        sag_angle = sag_plane(fin_num);

        /* move finger to saggital plane */
        if (sag_angle != 0.0)
            change_duct(fin_num, sag_angle);
    }

    /* find hypotenuese */
    switch (i)
    {
        case deg_DIP:
        {
            hyp = finger[fin_num].phalanx_length[DISTAL] / 2;
            /* midpt of distal */
            alpha = 0.0;

            /* calculate the length of the opposite */
            opp = new_length -
                ((finger[fin_num].curr_pos[PIP].x -
                  finger[fin_num].curr_pos[DIP].x) +
                 (finger[fin_num].curr_pos[MCP].x -
                  finger[fin_num].curr_pos[PIP].x) +
                 (finger[fin_num].curr_pos[CP].x -
                  finger[fin_num].curr_pos[MCP].x));

            /* calculate the height */
            height = sqrt((hyp*hyp) - (opp*opp));

            temp3.x = finger[fin_num].curr_pos[DIP].x;
            temp3.y = finger[fin_num].curr_pos[DIP].y;
            temp3.z = finger[fin_num].curr_pos[DIP].z -
                height;
        }
    }
}

```

```

        temp1 = vsub(temp3,
                    finger[fin_num].curr_pos[DIP]);
        temp2 = vsub(finger[fin_num].curr_pos[PIP],
                    finger[fin_num].curr_pos[DIP]);

        break;
    }
    case deg_PIP:
    {
        /* find midpt of distal */
        distal_midpt =
            midpt(finger[fin_num].curr_pos[NAIL],
                finger[fin_num].curr_pos[DIP]);

        /* find length of hypotenuese */
        temp = vsub(finger[fin_num].curr_pos[PIP],
                    distal_midpt);
        hyp = mag(temp);

        /* find alpha */
        temp1 = vsub(finger[fin_num].curr_pos[DIP],
                    finger[fin_num].curr_pos[PIP]);
        temp2 = vsub(distal_midpt,
                    finger[fin_num].curr_pos[PIP]);
        alpha = joint_angle(temp1, temp2);

        /* calculate the length of the opposite */
        opp = new_length -
            ((finger[fin_num].curr_pos[MCP].x -
              finger[fin_num].curr_pos[PIP].x) +
             (finger[fin_num].curr_pos[CP].x -
              finger[fin_num].curr_pos[MCP].x));

        /* calculate the height */
        height = sqrt((hyp*hyp) - (opp*opp));

        /* calculate omega variables */
        temp3.x = finger[fin_num].curr_pos[PIP].x;
        temp3.y = finger[fin_num].curr_pos[PIP].y;
        temp3.z = finger[fin_num].curr_pos[PIP].z -
                    height;

        temp1 = vsub(temp3,
                    finger[fin_num].curr_pos[PIP]);
        temp2 = vsub(finger[fin_num].curr_pos[MCP],
                    finger[fin_num].curr_pos[PIP]);

        break;
    }
}

```

```

case deg_MCP:
{
/* find midpt of distal */
    distal_midpt =
        midpt(finger[fin_num].curr_pos[NAIL],
              finger[fin_num].curr_pos[DIP]);

/* find length of hypotenuese */
    temp = vsub(finger[fin_num].curr_pos[MCP],
                distal_midpt);
    hyp = mag(temp);

/* find alpha */
    temp1 = vsub(finger[fin_num].curr_pos[PIP],
                 finger[fin_num].curr_pos[MCP]);
    temp2 = vsub(distal_midpt,
                 finger[fin_num].curr_pos[MCP]);
    alpha = joint_angle(temp1, temp2);

    opp = new_length -
          finger[fin_num].phalanx_length[CARPO];

/* calculate the height */
    height = sqrt((hyp*hyp) - (opp*opp));

/* calculate variables for omega */
    temp3.x = finger[fin_num].curr_pos[MCP].x;
    temp3.y = finger[fin_num].curr_pos[MCP].y;
    temp3.z = finger[fin_num].curr_pos[MCP].z -
              height;

    temp1 = vsub(temp3,
                 finger[fin_num].curr_pos[MCP]);
    temp2 = vsub(finger[fin_num].curr_pos[CP],
                 finger[fin_num].curr_pos[MCP]);

/* note that the MCP is going to flex */
    MCPflex = TRUE;

    break;
}
} /* switch */

/* step 2: calculate omega */
omega = joint_angle(temp1, temp2);

```

```

/* step 3: calculate beta */
beta = ((double)(opp))/hyp;

if ((beta <= 1.0) && (beta >= -1.0))
{
    beta = (asin(beta)) * 180 / PI;
    /* step 4: calculate theta */
    theta = 180.0 - (alpha + beta + omega);
} /* if */
else
    theta = 180.0;

/* check the constraints on angles */
if (theta > finger[fin_num].joints[i].max_angle)
{
    /* change the phalanx position counterclockwise */
    change_phalanx(fin_num, i,
        -(finger[fin_num].joints[i].max_angle -
            finger[fin_num].joints[i].curr_angle), axis);

    /* position this angle at max and recalculate */
    finger[fin_num].joints[i].curr_angle =
        finger[fin_num].joints[i].max_angle;

    /* move finger back if necessary */
    if (sag_angle != 0.0)
        change_duct(fin_num, -sag_angle);

    /* call this routine again */
    FDP_tension(new_length, fin_num);

    /* need continue with this version of the procedure */
    /* check if finger is in it's saggital plane */
    sag_angle = sag_plane(fin_num);

    /* move finger to saggital plane */
    if (sag_angle != 0.0)
        change_duct(fin_num, sag_angle);
} /* beta >= max */
else
{
    /* need change phalanx position counterclockwise */
    change_phalanx(fin_num, i,
        -(theta-finger[fin_num].joints[i].curr_angle),
        axis);

    /* current angle set to new angle */
    finger[fin_num].joints[i].curr_angle = theta;
}

```



```

/* need to reset the current muscle tension */
distal_midpt = midpt(finger[fin_num].curr_pos[NAIL],
                    finger[fin_num].curr_pos[DIP]);
finger[fin_num].fdp.tension =
    finger[fin_num].curr_pos[CP].x - distal_midpt.x;

/* EDC muscle is proportional to FDP */
finger[fin_num].edc.tension = finger[fin_num].fdp.tension;

/* need to reset the FDS muscle tension */
middle_midpt = midpt(finger[fin_num].curr_pos[DIP],
                    finger[fin_num].curr_pos[PIP]);
finger[fin_num].fds.tension =
    finger[fin_num].curr_pos[CP].x - middle_midpt.x;

if (fin_num != THUMB)
{
    /* EDL muscle is proportional to FDS */
    finger[fin_num].edl.tension =
        finger[fin_num].fdp.tension;

    /* reset the IO dorsal flexor */
    middle_midpt = midpt(finger[fin_num].curr_pos[PIP],
                        finger[fin_num].curr_pos[MCP]);
    finger[fin_num].io.d.tension =
        finger[fin_num].curr_pos[CP].x - middle_midpt.x;

    /* move finger back if necessary */
    if (sag_angle != 0.0)
        change_duct(fin_num, -sag_angle);

    /* if flexed MCP, abduct the finger by a percentage */
    if (MCPflex)
        ligament_mod(fin_num);
}

if (fin_num == THUMB)
{
    /* need to reset the EPB muscle tension */
    finger[fin_num].epb.tension =
        finger[fin_num].fds.tension;

    /* need to reset the APL muscle tension */
    distal_midpt = midpt(finger[fin_num].curr_pos[PIP],
                        finger[fin_num].curr_pos[MCP]);
    finger[fin_num].apl.tension =
        finger[fin_num].curr_pos[CP].x - distal_midpt.x;
}
} /* if */

```

```
    return;  
} /* FDP_tension */
```

Appendix B

Error Back-propagation Training Algorithm

```

/* This is the file for the activation function and its derivative.
   The function has been tailored to give answers between (0, 1.02).
   The derivative of the activation function was calculated by hand.
*/

```

```

double func_cont (double lambda, double net)
{
    return (1.02 / (1.0 + exp(-1.0 * lambda * net)));
}

```

```

double func_cont_der (double lambda, double net)
{
    return ((1.02 * lambda * exp(-1.0 * lambda * net)) /
            ((1.0 + exp(-1.0 * lambda * net)) *
             (1.0 + exp(-1.0 * lambda * net))));
}

```

```

/* This is the program that implements an error back-propagation
   training algorithm. The modifications are:
   1. calculating the errors - first do hidden weights, then
      recalculate the output of the neural network, then modify
      input weights.
   2. activation function is tailored for (0, 1.02)
*/

```

```

void feedforward (double lambda)
{
    int i, k; /* counters */

    /* output of hidden layer */
    for (i = 0; i < J; i++)
    {
        for (k = 0, nethid[i] = 0.0; k < I; k++)
            nethid[i] += (V[i][k] * z[k]);
    }
}

```

```

        /* apply the activation function */
        Y[i] = func_cont(lambda, nethid[i]);
    }

    /* output of output layer */
    for (i = 0; i < K; i++)
    {
        for (k = 0, netout[i] = 0.0; k < J; k++)
            netout[i] += (W[i][k] * Y[k]);

        /* apply the activation function */
        O[i] = func_cont(lambda, netout[i]);
    }

    return;
} /* feedforward */

void main (int argc, char *argv[])
{
    double Emax, N, ESFO[ARRAY_SIZE], ESFH[ARRAY_SIZE];
    double minW[ARRAY_SIZE][ARRAY_SIZE], minV[ARRAY_SIZE][ARRAY_SIZE];
    double E, d[ARRAY_SIZE], temp, lambda;
    double PATT[NUM_PATTERNS][15];
    int p, P;
    int i, j, k;
    FILE *ow, *hw, *infile;
    unsigned long int q, COUNT=0;
    double min_error;
    char filehid[20], fileout[20], cathid[]="-hid.weights",
        catout[]="-out.weights";
    int changed = FALSE, min_COUNT=0;

    /* change for input/output redirection */
    if ((argc < 3))
    {
        printf("The program must be started with:\ntrain <training file>
        <case name> <OPTIONAL initial weights file>\n");
        return;
    }

    /* STEP 0 - Get all user-defined values */

    /* get the values for P, I, J, K, N, and Emax */
    printf("Please enter the number of training patterns.\n");
    scanf("%d", &P);

```

```

printf("Please enter I (number of inputs), J (number of hidden
      nodes), and K (number of outputs) values for a single hidden layer
      network.\n");
scanf("%d %d %d", &I, &J, &K);
printf("Please enter the learning constant\n");
scanf("%lf", &N);

/* make sure that n is > 0 */
while (N <= 0)
{
    printf("Learning constant must be greater than 0. Please re-enter
          the learning constant.\n");
    scanf("%lf", &N);
}

printf("Please enter the maximum error value.\n");
scanf("%lf", &E_max);

printf("Please enter the value for lambda for the continuous bipolar
      function.\n");
scanf("%lf", &lambda);

/* STEP 1 - initialize W, V, q, p, E */
/* If more than 3 arguments in the startup line, then no randomize
   weights */

if (argc == 4)
{
    /* read from initial weights file */
    ow = fopen(argv[3], "r");

    for (i = 0; i < K; i++)
        for (j = 0; j < J; j++)
            fscanf(ow, "%lf", &W[i][j]);

    for (i = 0; i < J; i++)
        for (j = 0; j < I; j++)
            fscanf(ow, "%lf", &V[i][j]);

    printf("\nAll done with reading weights.\n");

    fclose(ow);
} /* read weights */
else
{
    for (i = 0; i < K; i++)
        for (j = 0; j < J; j++)
            W[i][j] = (double)((rand() % 11) - 5) / 10.0;
}

```

```

    for (i = 0; i < J; i++)
        for (j = 0; j < I; j++)
            V[i][j] = (double)((rand() % 11) - 5) / 10.0;

    printf("\nAll done with randomizing weights.\n");
} /* else randomize */

/* make outfile names */
strcpy(filehid, argv[2]);
strcpy(fileout, argv[2]);
strcat(filehid, cathid);
strcat(fileout, catout);

printf("OUTPUT FILE NAMES:  %s %s\n", filehid, fileout);

/* initialize variables */
q = 0;  p = 0;
E = 0.0;
min_error = 100000.0;

/* get all the patterns */
infile = fopen(argv[1], "r");
if ((infile == NULL) || (P > NUM_PATTERNS))
{
    printf("Can't open file      OR\nNumber of training patterns is
    greater than %d!\n", NUM_PATTERNS);
    return;
}
else
{
    for (i = 0; i < P; i++)
        for (j = 0; j < (I + K); j++)
            fscanf(infile, "%lf", &PATT[i][j]);

    printf("First pattern is:\n");
    for (j = 0; j < 15; j++)
        printf("%f ", PATT[0][j]);
    printf("\n");

    /* all done reading patterns */
    fclose(infile);
}
while (p < P)
{
    p += 1;
    q += 1;
}

```

```

/* STEP 2 - training starts here */
/* get z and d for this training case p */
for (i = 0; i < I; i++)
    z[i] = PATT[p-1][i];

for (i = 0; i < K; i++)
    d[i] = PATT[p-1][i+I];

/* process the pattern through the neural network */
feedforward(lambda);

/* STEP 3 - compute error (don't need this until second
pass-through) */

/* STEP 4 - error signal vectors of both layers are computed */

for (i = 0; i < K; i++)
{
    ESFO[i] = (d[i] - O[i]) *
              func_cont_der(lambda, netout[i]);
}

for (i = 0; i < J; i++)
{
    for (k = 0, temp = 0.0; k < K; k++)
        temp += (ESFO[k] * W[k][i]);
    ESFH[i] = func_cont_der(lambda, nethid[i]) * temp;
}

/* This is where we changed steps 5 and 6 from Zurada:
do step 6 first, then rerun the network, compute
error signals, and then change the output layer
weights. This is to speed convergence of the
network.
*/

/* STEP 6 - hidden layer weights are adjusted */
for (j = 0; j < J; j++)
{
    for (i = 0; i < I; i++)
        V[j][i] += (N * ESFH[j] * z[i]);
}
/* now rerun the network, computer errors */
feedforward(lambda);

```

```

/* STEP 3 - now compute error */
for (i = 0; i < K; i++)
{
    E += (0.5 * ((d[i] - O[i]) * (d[i] - O[i]))));
}

/* STEP 4 - error signal vector computed for output layer only */
for (i = 0; i < K; i++)
{
    ESFO[i] = (d[i] - O[i]) *
               func_cont_der(lambda, netout[i]);
}

/* STEP 5 - output layer weights are adjusted */
for (k = 0; k < K; k++)
{
    for (j = 0; j < J; j++)
    {
        W[k][j] += (N * ESFO[k] * Y[j]);
    }
}

/* STEP 7 - if p < P then p++, q++ and go back to step 2;
           otherwise, go to step 8 */

/* STEP 8 - training cycle is complete. */
if (p >= P)
{
    COUNT++;
    printf("Training cycle %ld had ERROR %lf\n", COUNT, E);

    /* if error decreases, then temp values need to be updated */
    if (min_error > E)
    {
        /* update temp weights */
        for (i = 0; i < K; i++)
        {
            for (j = 0; j < J; j++)
            {
                minW[i][j] = W[i][j];
            } /* for j */
        } /* for i */
        for (i = 0; i < J; i++)
        {
            for (j = 0; j < I; j++)
            {
                minV[i][j] = V[i][j];
            } /* for j */
        } /* for i */
    }
}

```



```

/* weights have been changed and so has mincount*/
changed = TRUE;
min_COUNT = COUNT;

/* update minerror */
min_error = E;

} /* min_error */

if (((COUNT % CYCLE_NUM) == 0) && (changed) && (E >= Emax))
{
    printf("TEMP Print output layer weights to %s:\n", fileout);

    ow = fopen(fileout, "w");
    if (ow != NULL)
    {
        fprintf(ow, "TEMP\n\n");
        fprintf(ow, "COUNT = %ld and ERROR = %lf\n\n", min_COUNT,
            min_error);
        for (i = 0; i < K; i++)
        {
            for (j = 0; j < J; j++)
                fprintf(ow, "%f ", minW[i][j]);
            fprintf(ow, "\n");
        }
        fclose(ow);
    } /* if */
    printf("\n\nTEMP Print hidden layer weights to %s:\n",
        filehid);

    hw = fopen(filehid, "w");
    if (hw != NULL)
    {
        for (i = 0; i < J; i++)
        {
            for (j = 0; j < I; j++)
                fprintf(hw, "%f ", minV[i][j]);
            fprintf(hw, "\n");
        }
        fclose(hw);
    } /* if */
    changed = FALSE;

} /* temp out */

```

```

if (E < Emax)
{
    printf("Print output layer weights to %s:\n", fileout);

    ow = fopen(fileout, "w");
    if (ow != NULL)
    {
        fprintf(ow, "COUNT = %ld and ERROR = %lf\n\n", COUNT, E);
        for (i = 0; i < K; i++)
        {
            for (j = 0; j < J; j++)
                fprintf(ow, "%f ", W[i][j]);
            fprintf(ow, "\n");
        }
        fclose(ow);
    } /* if */
    else
    {
        printf("File %s could NOT be opened for writing!!\n",
            fileout);
        printf("Output to screen.\n");

        for (i = 0; i < K; i++)
        {
            for (j = 0; j < J; j++)
                printf("%f ", W[i][j]);
            printf("\n");
        }
    } /* else */

    printf("\n\nPrint hidden layer weights to %s:\n", filehid);

    hw = fopen(filehid, "w");
    if (hw != NULL)
    {
        for (i = 0; i < J; i++)
        {
            for (j = 0; j < I; j++)
                fprintf(hw, "%f ", V[i][j]);
            fprintf(hw, "\n");
        }
        fclose(hw);
    } /* if */

```

```
else
{
    printf("File %s could NOT be opened for writing!!\n",
        filehid);
    printf("Output to screen.\n");
    for (i = 0; i < J; i++)
    {
        for (j = 0; j < I; j++)
            printf("%f ", V[i][j]);
        printf("\n");
    }
} /* else */

printf("\n\n q = %ld      E = %lf\n", q, E);
break;
} /* if */
else
{
    /* initiate new training cycle */
    E = 0.0;
    p = 0;
}
} /* if p >= P */
} /* while */
} /* main */
```

Appendix C

Excerpts from the input-output data sets for:

- C1. Finger Flexion and Extension Neural Network
- C2. Finger Abduction and Adduction Neural Network

C1. Excerpt from the Input-output Data Set for the Finger Flexion and Extension Neural Network

Resultant Tendon Lengths				Current Tendon Lengths				Desired ANN Output						
FDP	FDS	DOR	EDC	INTR	FDP	FDS	DOR	EDC	INTR	FDP	FDS	DOR	EDC	INTR
0.551490	0.703585	1.000000	1.000000	0.551490	0.764041	0.444151	1.000000	1.000000	0.764041	0.000000	0.140566	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.576178	0.720000	0.720000	1.000000	1.000000	0.576178	0.000000	0.000000	0.000000	0.000000	0.000000
0.050949	0.012264	0.000000	0.000000	0.077211	0.077211	0.118003	1.000000	1.000000	0.077211	0.000000	0.000000	0.000000	0.000000	0.000000
0.463657	0.703585	1.000000	1.000000	0.488345	0.135094	0.000000	0.000000	0.000000	0.488345	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.276778	0.012453	0.000000	0.000000	0.000000	0.276778	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.566814	0.428491	0.153355	1.000000	1.000000	0.566814	0.000000	0.000000	0.000000	0.000000	0.000000
0.488345	0.334906	0.000000	0.000000	0.199771	0.148515	0.000000	0.000000	0.000000	0.199771	0.000000	0.000000	0.000000	0.000000	0.000000
0.551490	0.703585	1.000000	1.000000	0.862300	0.211173	0.112129	1.000000	1.000000	0.862300	0.000000	0.000000	0.000000	0.000000	0.000000
0.463657	0.703585	1.000000	1.000000	0.077801	0.077801	0.077801	1.000000	1.000000	0.077801	0.000000	0.000000	0.000000	0.000000	0.000000
0.050949	0.012264	0.000000	0.000000	0.441133	0.681830	0.935461	1.000000	1.000000	0.441133	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.488345	0.315283	0.000000	0.000000	0.000000	0.488345	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.693912	0.797736	1.000000	1.000000	1.000000	0.693912	0.000000	0.000000	0.000000	0.000000	0.000000
0.463657	0.703585	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.551490	0.703585	1.000000	1.000000	0.463657	0.115570	0.160189	0.274584	1.000000	0.463657	0.000000	0.000000	0.000000	0.000000	0.000000
0.050949	0.012264	0.000000	0.000000	0.606669	0.740189	1.000000	1.000000	1.000000	0.606669	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.870643	0.380000	1.000000	1.000000	1.000000	0.870643	0.000000	0.000000	0.000000	0.000000	0.000000
0.463657	0.703585	1.000000	1.000000	0.451567	0.655283	0.980333	1.000000	1.000000	0.451567	0.000000	0.000000	0.000000	0.000000	0.000000
0.551490	0.703585	1.000000	1.000000	0.050949	0.050949	0.163405	0.255283	1.000000	0.050949	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.463657	0.703774	0.384906	0.648260	1.000000	0.463657	0.000000	0.000000	0.000000	0.000000	0.000000
0.050949	0.012264	0.000000	0.000000	1.000000	0.505459	0.738679	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.050949	0.051048	0.034453	0.000000	0.000000	0.050949	0.000000	0.000000	0.000000	0.000000	0.000000
0.463657	0.703585	1.000000	1.000000	0.303108	0.335472	0.571104	0.571104	1.000000	0.303108	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.576178	0.794151	1.000000	1.000000	1.000000	0.576178	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.065539	0.148260	0.148260	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.50949	0.012264	0.000000	0.000000	0.717419	0.090943	1.000000	1.000000	1.000000	0.717419	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.050949	0.480083	0.335094	0.000000	0.000000	0.050949	0.000000	0.000000	0.000000	0.000000	0.000000
0.463657	0.703585	1.000000	1.000000	0.488345	0.138750	0.097547	0.000000	0.000000	0.488345	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.463657	0.399823	0.637547	0.966717	1.000000	0.463657	0.000000	0.000000	0.000000	0.000000	0.000000
0.463657	0.703585	1.000000	1.000000	1.000000	0.776335	0.330000	1.000000	1.000000	0.776335	0.000000	0.000000	0.000000	0.000000	0.000000
0.551490	0.703585	1.000000	1.000000	0.463657	0.584046	0.800000	1.000000	1.000000	0.463657	0.000000	0.000000	0.000000	0.000000	0.000000
0.050949	0.012264	0.000000	0.000000	0.232205	0.232205	0.232205	1.000000	1.000000	0.232205	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.788040	0.316792	1.000000	1.000000	1.000000	0.788040	0.000000	0.000000	0.000000	0.000000	0.000000
0.050949	0.012264	0.000000	0.000000	0.259172	0.434340	0.721634	0.821483	1.000000	0.259172	0.000000	0.000000	0.000000	0.000000	0.000000
0.488345	0.334906	0.000000	0.000000	0.488345	0.584222	0.488345	1.000000	1.000000	0.488345	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.551000	0.703585	1.000000	1.000000	0.463657	0.139046	0.246189	0.000000	0.000000	0.463657	0.000000	0.000000	0.000000	0.000000	0.000000
0.463657	0.703585	1.000000	1.000000	0.050949	0.463657	0.118454	0.000000	0.000000	0.050949	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.050949	0.246189	0.174562	1.000000	1.000000	0.050949	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	1.000000	0.811846	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.463657	0.703585	1.000000	1.000000	0.463657	0.961050	0.000000	0.000000	0.000000	0.463657	0.000000	0.000000	0.000000	0.000000	0.000000
0.050949	0.012264	0.000000	0.000000	0.050949	0.155208	0.248330	0.418306	1.000000	0.050949	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.050641	0.050641	0.050641	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.488345	0.334906	0.000000	0.000000	0.488345	0.962821	0.951698	0.927383	1.000000	0.488345	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.085270	0.267020	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.849556	0.568868	1.000000	1.000000	1.000000	0.849556	0.000000	0.000000	0.000000	0.000000	0.000000
0.050949	0.012264	0.000000	0.000000	0.050949	0.932279	0.703245	1.000000	1.000000	0.050949	0.000000	0.000000	0.000000	0.000000	0.000000
0.050949	0.012264	0.000000	0.000000	0.463657	0.922072	0.303245	1.000000	1.000000	0.463657	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	0.050949	0.727058	0.896623	1.000000	1.000000	0.050949	0.000000	0.000000	0.000000	0.000000	0.000000
0.551490	0.703585	1.000000	1.000000	1.000000	0.952087	0.968068	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.551490	0.803482	0.870189	1.000000	1.000000	0.551490	0.000000	0.000000	0.000000	0.000000	0.000000
0.050949	0.012264	0.000000	0.000000	0.000000	0.481263	0.648434	0.975794	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.463657	0.703585	1.000000	1.000000	0.463657	0.181174	0.185283	0.000000	0.000000	0.463657	0.000000	0.000000	0.000000	0.000000	0.000000
					0.675027	0.675027	1.000000	1.000000	0.675027	0.000000	0.000000	0.000000	0.000000	0.000000

C2. Excerpt from the Input-output Data Set for the Finger Abduction and Adduction Neural Network

Tendon Lengths					
Resultant		Current		Desired ANN Output	
DI	PI	DI	PI	DI	PI
0.000000000	0.000000000	0.225115000	0.270541000	0.225115000	0.000000000
1.000000000	1.000000000	0.635528000	0.727455000	0.000000000	0.272545000
0.000000000	0.000000000	0.702910000	0.795591000	0.702910000	0.000000000
1.000000000	1.000000000	0.341501000	0.406814000	0.000000000	0.593186000
1.000000000	1.000000000	0.751914000	0.841683000	0.000000000	0.158317000
0.000000000	0.000000000	0.733538000	0.823647000	0.733538000	0.000000000
1.000000000	1.000000000	0.078101000	0.096192000	0.000000000	0.903808000
1.000000000	1.000000000	0.139357000	0.168337000	0.000000000	0.831663000
0.000000000	0.000000000	0.010720000	0.014028000	0.010720000	0.000000000
1.000000000	1.000000000	0.151608000	0.184369000	0.000000000	0.815631000
1.000000000	1.000000000	0.555896000	0.645291000	0.000000000	0.354709000
0.000000000	0.000000000	0.947933000	0.985972000	0.947933000	0.000000000
1.000000000	1.000000000	0.574273000	0.665331000	0.000000000	0.334669000
0.000000000	0.000000000	0.249617000	0.300601000	0.249617000	0.000000000
0.000000000	0.000000000	0.059724000	0.072144000	0.059724000	0.000000000
0.000000000	0.000000000	0.114855000	0.140281000	0.114855000	0.000000000
1.000000000	1.000000000	0.837672000	0.913828000	0.000000000	0.086172000
0.000000000	0.000000000	0.194487000	0.234469000	0.194487000	0.000000000
1.000000000	1.000000000	0.445636000	0.525050000	0.000000000	0.474950000
0.000000000	0.000000000	0.604900000	0.697395000	0.604900000	0.000000000
1.000000000	1.000000000	0.996937000	1.000000000	0.000000000	0.000000000
1.000000000	1.000000000	0.402757000	0.476954000	0.000000000	0.523046000
1.000000000	1.000000000	0.004594000	0.006012000	0.000000000	0.993988000
1.000000000	1.000000000	0.892802000	0.953908000	0.000000000	0.046092000
1.000000000	1.000000000	0.537519000	0.625251000	0.000000000	0.374749000
0.000000000	0.000000000	0.837672000	0.913828000	0.837672000	0.000000000
1.000000000	1.000000000	0.274119000	0.328657000	0.000000000	0.671343000
0.000000000	0.000000000	0.482389000	0.565130000	0.482389000	0.000000000
1.000000000	1.000000000	0.304747000	0.364729000	0.000000000	0.635271000
1.000000000	1.000000000	0.176110000	0.212425000	0.000000000	0.787575000
0.000000000	0.000000000	0.182236000	0.220441000	0.182236000	0.000000000
0.000000000	0.000000000	0.917305000	0.969940000	0.917305000	0.000000000
1.000000000	1.000000000	0.114855000	0.140281000	0.000000000	0.859719000
1.000000000	1.000000000	0.010720000	0.014028000	0.000000000	0.985972000
1.000000000	1.000000000	0.415008000	0.490982000	0.000000000	0.509018000
0.000000000	0.000000000	0.898928000	0.957916000	0.898928000	0.000000000
0.000000000	0.000000000	0.984686000	0.997996000	0.984686000	0.000000000
0.000000000	0.000000000	0.745789000	0.835671000	0.745789000	0.000000000
0.000000000	0.000000000	0.200613000	0.242485000	0.200613000	0.000000000
0.000000000	0.000000000	0.886677000	0.949900000	0.886677000	0.000000000
1.000000000	1.000000000	0.476263000	0.559118000	0.000000000	0.440882000