ABSTRACT

A HAPTIC CONTROL SYSTEM FOR FUNCTIONAL ELECTRICAL STIMULATION OF PARAPLEGIC LEGS

by
Mark R. Shaker

Functional electrical stimulation (FES) is a means by which paraplegic men and women can use their natural legs for walking. In FES the impaired muscles are stimulated with electricity in a proper cycle to cause the legs to move in a walking pattern. It can be greatly beneficial for paraplegics however, current systems are not widely used because they are difficult to control in a useful manner.

The system proposed here uses a haptic interface, one that utilizes the sense of touch, attached to a user’s index and middle fingers. The haptic device allows the wearer to feel with the fingers what would normally be felt by the feet. Movement of the fingers is monitored and the positions of the two fingertips can be used to dictate the appropriate positions for the feet to be moved to using FES. Therefore, by moving the fingers in a cyclic pattern similar to that of walking, a stimulation pattern needed for activation of leg muscles to allow walking can be generated. Further, by having the sense of feeling for the feet translated to the fingers a person could have improved control over their legs.

To test the feasibility of this system a virtual simulation was developed. The simulation navigated a virtual environment using the finger walking technique. The trajectory and velocity of the movements of the subjects was compared to normal human gait and it was found that finger walking greatly resembles natural human gait. Further, it was determined that control was enhanced by haptic feedback. These results show that FES walking can benefit from a controller that incorporates haptics.

A HAPTIC CONTROL SYSTEM FOR FUNCTIONAL ELECTRICAL STIMULATION OF PARAPLEGIC LEGS

by
Mark R. Shaker

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Biomedical Engineering

Department of Biomedical Engineering

August 2010

APPROVAL PAGE

A HAPTIC CONTROL SYSTEM FOR FUNCTIONAL ELECTRICAL STIMULATION OF PARAPLEGIC LEGS

Mark R. Shaker

Dr. Richard A. Foulds, Thesis Advisor Date
Associate Professor of Biomedical Engineering, NJIT

Dr. Sergei Adamovich, Committee Member Date
Associate Professor of Biomedical Engineering, NJIT

Dr. Mesut Sahin, Committee Member Date
Associate Professor of Biomedical Engineering, NJIT

BIOGRAPHICAL SKETCH

Author:	Mark R. Shaker
Degree:	Master of Science
Date:	May 2010

Undergraduate and Graduate Education:

- Master of Science in Biomedical Engineering
  New Jersey Institute of Technology, Newark, NJ, 2010

- Bachelor of Science in Biomedical Engineering
  New Jersey Institute of Technology, Newark, NJ, 2009

Major:	Biomedical Engineering

This work is dedicated to my family and friends who believed in me when I did not.

ACKNOWLEDGMENT

I wish to acknowledge the assistance of my advisor Dr. Richard Foulds. I would like to thank Dr. Sergei Adamovich and Dr. Mesut Sahin for serving on my thesis committee.

I wish to thank the Gustavus and Louise Pfeiffer Research Foundation for their generous grant which helped to fund my research.

I also wish to acknowledge all the students working on their theses and dissertations in Dr. Foulds' and Dr. Adamovich's labs who gave their input and advice along the way. Without the technical assistance of Diego Ramirez and Qinyin Qiu I would not have been able to complete this project. I would like to thank John Weimer, Brook Odle and John Hoinowski for their generous computer donations.

Finally I wish to acknowledge my family and friends for their enduring support throughout my education.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Objective

The objective of this research is to prove the theory that FES walking can be better controlled by using the finger walking technique with haptic feedback. In current FES systems locomotion is synthesized by iteratively modifying a basic time-varying stimulation pattern to improve the gait of each subject. A single stimulation pattern can not be implemented for every subject due in large part to the differences in the inertia of limbs from person to person. This comes about as a result of the difference in the size and shape of limb segments [1]. Stimulation patterns are adapted to each individual's needs in order to create a gait pattern similar to natural walking. Adaptive feed forward control systems allow more accurately controlled movements by monitoring joint and limb segment angles throughout the movement and changing the stimulation intensity and timing appropriately to optimize the movement to the desired trajectory. One such system of adaptive control developed by Ou, Riess, and Abbas produced movements with minimal errors between the desired input limb segment movements and the observed output movements [2]. Due to the necessity for synchronization of upper limb movement to lower limb movement and the level of control required to walk across rough or uneven terrain or to navigate obstacles, FES must be controlled largely under the patient's voluntary control [3]. Voluntary control systems include switches and joysticks however no existing systems incorporate haptic feedback as this system does.

When proven thoroughly this technique can be applied to FES systems with the purpose of enabling walking to people with paraplegia. Given the current technology in FES which can produce a highly accurate movement given a desired limb trajectory, this system can enhance control further in order to allow more versatility to FES systems. If FES walking can be used by people with paraplegia throughout their day to day life instead of a wheelchair, it can allow more mobility and better health [4]. This basic science research is a preliminary step towards the final goal of an FES system for paraplegic walking.

1.2 Spinal Cord Injury

The spinal cord is an extension of the central nervous system that runs down the back and is surrounded by the vertebrae. The spinal cord transmits signals between the brain and the rest of the body. The spine has four sections. The highest eight vertebrae are called the cervical vertebrae. Moving down the spine, the next twelve vertebrae are called the thoracic vertebrae. Below the thoracic region there are five lumbar vertebrae. The lowest five vertebrae are fused together and form what is called the sacrum. Signals from the brain move through the spinal column and nerve roots branch out to the different regions of the body from the spine at different levels. Therefore, all the nerves of the spinal cord pass through the first cervical vertebra (C1) and the fewest number of nerves pass through the lowest sacral vertebra (S5) [5].

Approximately 259,000 persons are currently living in the United States with a spinal cord injury (SCI). It is estimated that an additional 12,000 new SCI cases occur each year [6]. Causes for SCI include but are not limited to; falls, acts of violence, sports injuries, and automobile accidents. Automobile accidents are the most prevalent if the age of the person is less than 45, after that age falls become the most prevalent cause for SCI. Of these injuries slightly more than half result in quadriplegia [7].

Injuries to the spinal cord have different effects based upon the level of the injury and the completeness of the injury. An injury can be complete or incomplete. A complete injury leads to no function below the level of the injury. An incomplete injury can still retain some function below the level of the injury. The level of the injury can predict what parts of the body may lose function. Injuries to the cervical vertebrae generally lead to quadriplegia, complete paralysis of the body from the neck down, and injuries to the thoracic vertebrae generally lead to paraplegia, paralysis of the lower part of the body including the legs. If an injury is higher on the spine it can affect more parts of the body than if the injury is lower on the spine. High level injuries can cause impairments to the extent that a person would require a ventilator to breathe. Injuries to the lumbar region do less harm, these can cause a person to be unable to move their legs but still have the ability to sit upright and control their abdominal muscles, chest and arms. People with lower level injuries can attain mobility with manual or powered wheelchairs but, higher level injuries generally require a powered wheelchair [8].

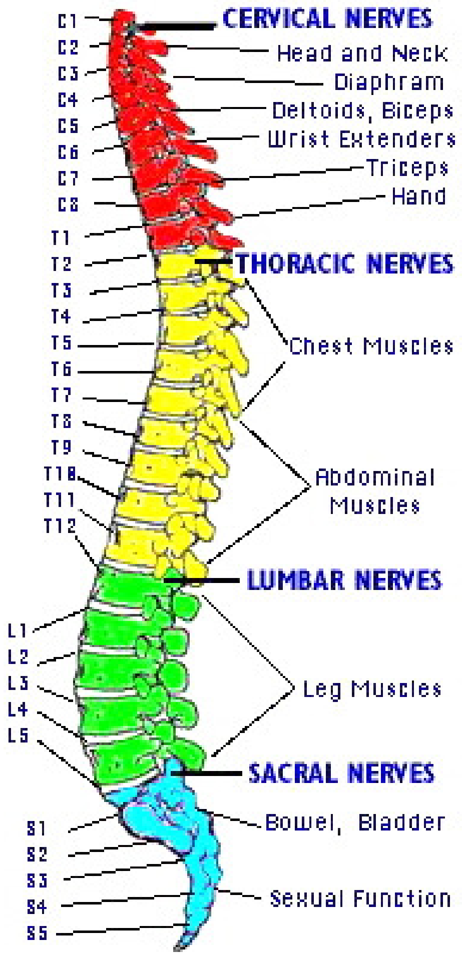

Figure 1.1 A diagram representing the different levels of the spine and what is controlled at each level [9].

1.3 Functional Electrical Stimulation

1.3.1 Introduction to FES

For people who have received a spinal cord injury and experienced paralysis FES is sometimes used as a treatment. FES is a method of restoring or improving body function by applying low level currents to the affected region [4]. Depending on the level of the injury and severity of the SCI, different methods of FES can be used to benefit the person with the injury. If an injury is above the C4 level of the spine, a person may need assistance to breathe, in which case FES can be used. If an injury is at the C5 level, a person can retain control of their chest and biceps however, hand function and triceps control can be lost, in which case FES can be applied to assist with arm extension and some hand control. A lumbar level injury can lead to control of the upper body but loss of voluntary movement in the legs, in this situation FES may be applied for stimulation of the muscles to increase blood flow, for standing to prevent osteoporosis and for walking. Other uses for FES include; cardiovascular exercise, coughing assistance, improving bladder and bowel control, prevention and treatment of pressure sores, controlling spasticity or tremor, and in some cases it can be used to regain voluntary control [4, 8].

Walking with the use of FES has many benefits for those who can use it. The paralyzed muscles do not receive the signals from the spinal cord that cause movement. Therefore, these do not get used. The lack of use causes the muscles to break down over time, this is called atrophy. Lack of use also causes poor circulation of blood in the region which can lead to blood clots. The lack of mechanical loading of the bones in the legs leads to their weakening, this is called osteoporosis. Paraplegics who use a system which can enable them to move and load their legs reduce their likelihood to experience blood clots and broken bones in their legs. This can also improve their general cardiovascular health [4].

In order to walk using FES, a minimum of four channels is required for stimulation [11]. These would stimulate the quadriceps and the common peroneal nerve of each leg. By stimulating the quadriceps a person could stand and lock their legs in place. By stimulating the peroneal nerve without stimulating the quadriceps a leg can be brought into flexion. Cyclic stimulation of the appropriate muscle groups can lead to walking. Systems are available with up to 48 channels [21]. The higher number of channels leads to a greater extent of control over the legs; however, it requires a more complex system to control the greater number of channels. Usually the users control the FES with switches or a joystick. The users also generally require a walker or a cane to help maintain balance and support with their arms [4].

1.3.2 The Human Gait Cycle

The human gait cycle is composed of two main phases [10]. These are swing phase and stance phase. In stance phase the foot is in contact with the ground. In the swing phase the foot is progressing forward and is not in contact with the ground. The stance phase can be broken down into three subsets. These are: the first double support phase, the single support phase, and the second double support phase. The double support phase refers to the part of a walking cycle in which both feet are on the ground at the same time. The single support phase refers to when only one foot is in contact with the ground while the other foot is in swing phase. The gait cycle describes the motion of one leg. Each leg moves through its own gait cycle a half cycle out of sync from the other leg. Therefore, while one leg is in the swing phase the other leg is in the single support phase. In a normal healthy gait, approximately 60% of the cycle is spent in stance phase and the remaining 40% is in swing phase. For describing the gait cycle it is standard to start when one leg makes contact with the heel, in Figure 1.2 you can see that the cycle starts when the boy’s right heel makes contact with the ground.

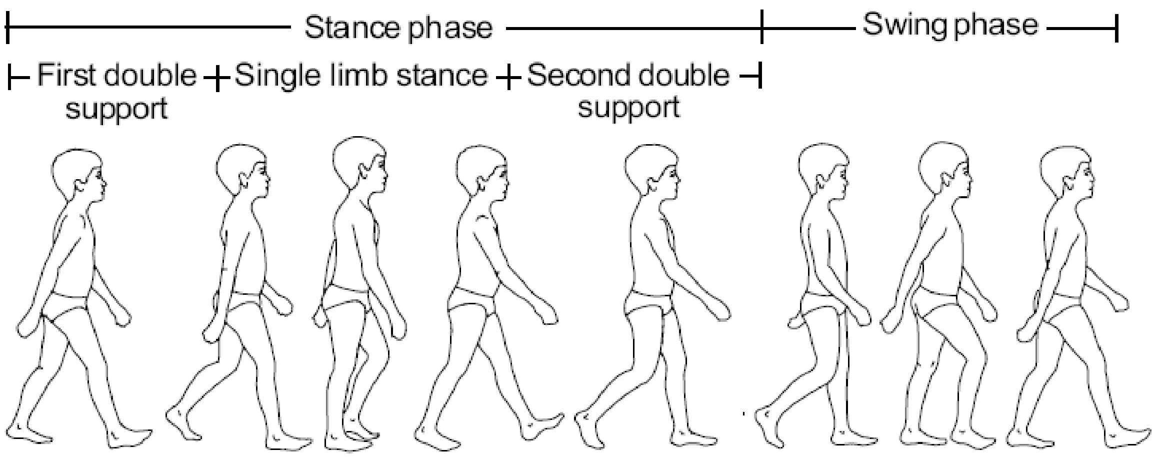

Figure 1.2 A diagram of the phases of the gait cycle [10].

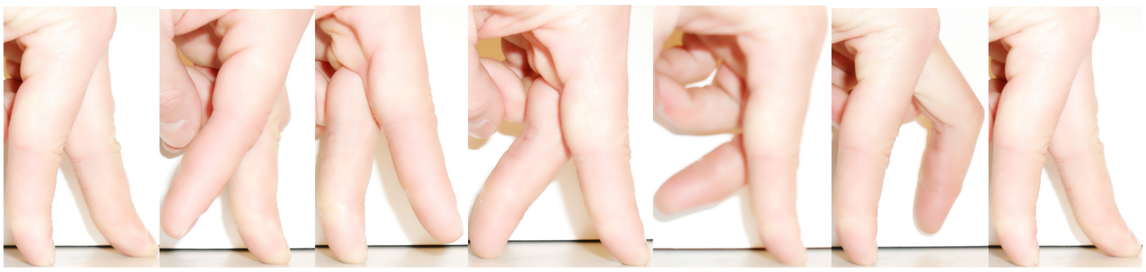
Figure 1.3 A picture of the phases of the gait cycle when finger walking.

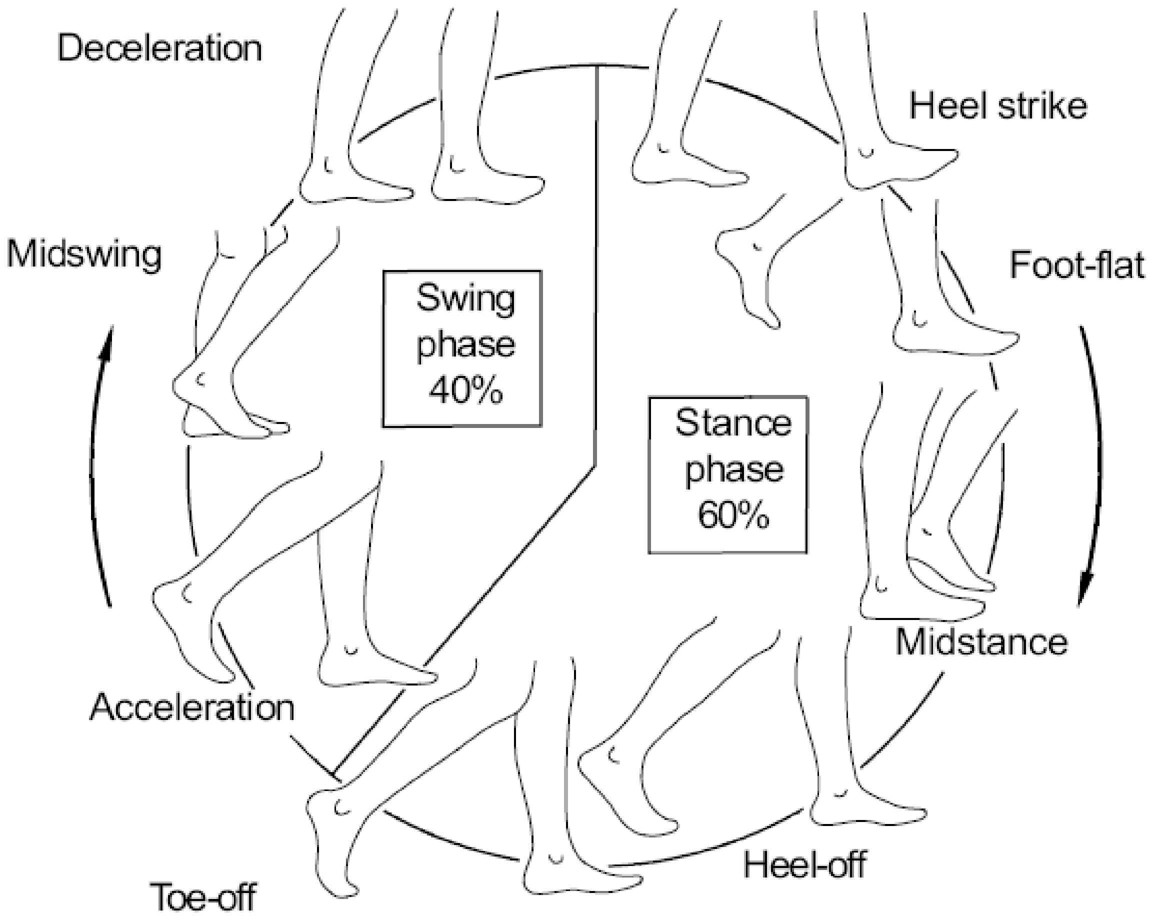

Figure 1.4 A diagram of the gait cycle and breakdown of the phases [10].

1.3.3 FES Walking

FES walking can be achieved with a minimum of four channels of stimulation. One such study of this was published by Badj et al. in 1983 [11]. In this study stimulation of the knee extensor muscles was performed first in order to prepare the muscles for standing and walking. This exercise was done until the knee joint torque generated by the electrical stimulation exceeded 30 – 50 Nm. When the subjects were able to generate the appropriate torque, they were trained to stand by stimulating the knee extensor muscles. In order to do this, a stimulation voltage of 100 V was applied to the knee extensor muscles, and the voltage was then lessened to the lowest level at which the subject could maintain knee extension. Subjects were then trained to stand for an hour or more using this electrical stimulation system.

Once subjects were appropriately conditioned to FES, and their muscles could generate the required forces, the ambulation training was begun. In order to walk, a minimum of four channels of stimulation is required. Two of the channels are used to stimulate the knee extensor muscles of each leg. The remaining two channels are used to stimulate the knee flexor muscles of each leg. Simultaneous flexion of the hip and knee, and dorsiflexion of the ankle was achieved by stimulating one of four afferent nerves. These nerves were: superficial peroneal, common peroneal, sural, or saphenous. Appropriate electrode placement for each subject was determined by trial and error adjustment of the electrodes until the desired response was achieved upon stimulation.

Double stance phase can be achieved by simultaneous stimulation of the knee extensors of both legs. In order to generate a swing phase, the extensor muscles of one leg are stimulated while the flexor muscles of the opposite leg are stimulated. By cyclically stimulating the extensors of one leg and the flexors of the other for alternating legs a gait cycle was developed. In order to control this, in the experiment, two switches were used. One switch would cause a swing phase of the right leg. The second switch would cause a swing phase of the left leg. If neither switch is pressed, the knee extensors for both legs would be stimulated and the subject would remain in stance phase. These switches are mounted on a walker, crutches or some other form of upper body support to maintain balance and for safety.

Using this system, one subject was able to walk distances of approximately one kilometer each day in the area outside of his home. Other subjects were able to stand without any assistance other than the FES device and walk short distances with a walker. Some of the other benefits experienced by participants in the program were; enhanced bladder and bowel control, regular bowel movements, increased strength in trunk muscles, and in one patient there was a decrease in high blood pressure.

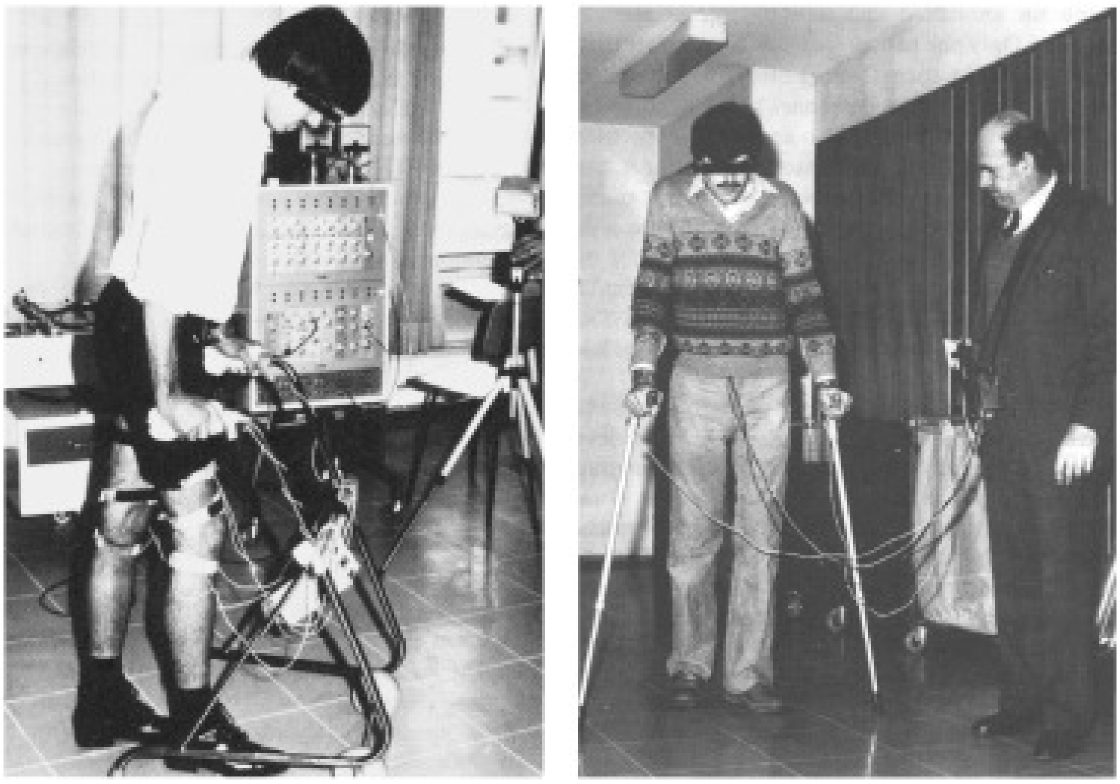

Figure 1.5 A man walking with four channel FES and a walker (left) and two canes (right) [11].

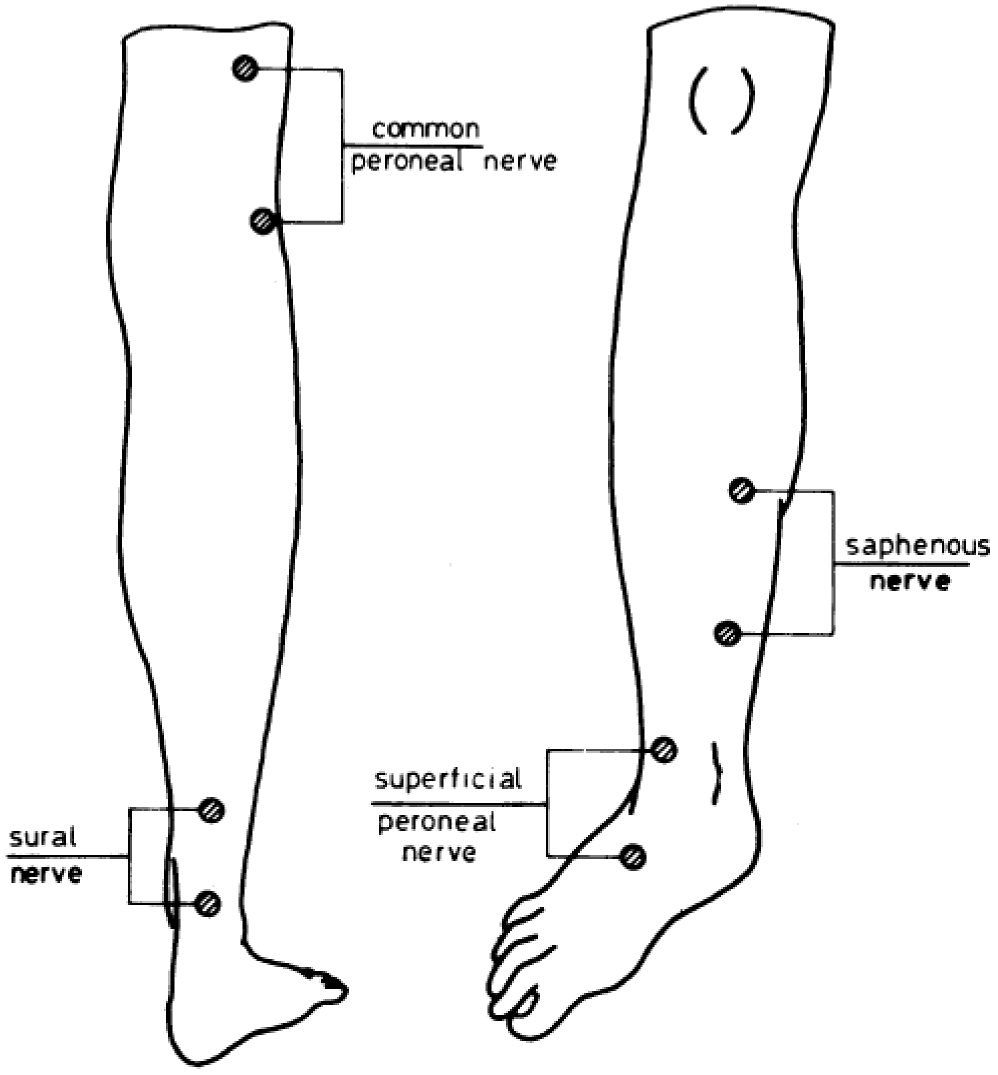

Figure 1.6 Approximate stimulation points for eliciting the synergistic flexion response using FES [11].

More control can be obtained with more channels of stimulation to additional muscle groups. In some studies intramuscular electrodes are used instead of surface electrodes to achieve a better signal and muscle response [12]. Other systems exist that combine leg braces with FES, these are called hybrid systems [13]. Hybrid systems provide more stability of the legs while stimulating fewer muscles which makes them less energy consuming and more stable however, their use is dependent on the level and extent of the injury.

1.4 Finger Walking

Finger walking refers to moving fingers in a reciprocating pattern similar to that of the human gait cycle. If the index and middle fingers of the right hand are used, the index finger would represent the left leg and the middle finger would represent the right leg. A thesis was conducted by Matthew Noesner at The New Jersey Institute of Technology which investigated finger walking [14]. There gait cycle timing, finger trajectories and ground reaction forces were measured for subjects while they performed the finger motions. These were compared to published data for normal walking and strong similarities were seen. Through this work it was demonstrated that human walking can be mimicked through finger movements.

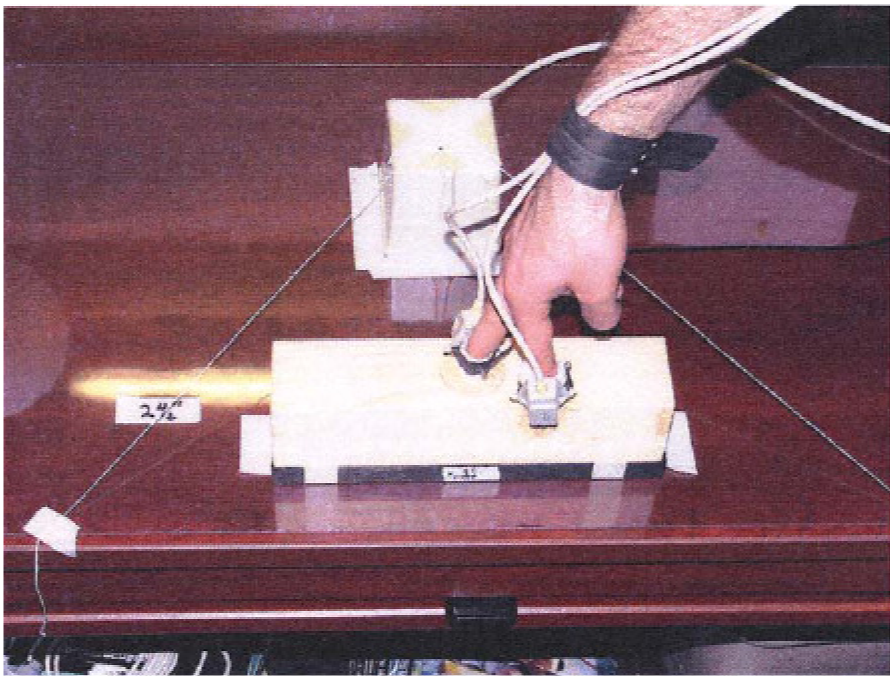

Figure 1.7 A demonstration of finger walking [14].

Additional work was done in which finger walking in place was used as a means to navigate virtual environments. In a research study conducted by Kim, Gracanin, Matkovic, and Quek, Subjects were able to navigate 2-D virtual environments by sliding their bare fingers across a touch sensitive surface [15]. For this application, two touch screen computer monitors were used in order to allow a subject to walk in a virtual environment with one hand and turn with the other hand. This was only a study into 2-D navigation however; the authors believe it can be extended into a 3-D environment.

1.5 Haptics

Haptics is the science of incorporating the sense of touch into computer programs by providing kinesthetic or tactile feedback [16]. Haptic technology has a wide number of uses which include but are not limited to; surgical simulations, medical training, painting, computer aided design, video games, and rehabilitation therapy [16, 17]. In order to use haptics in a computer program a special device is required to generate the forces, this is called a haptic device. In this research the haptic device used was the Sensable PHANToM Premium 1.0A. To use this haptic device a finger is inserted in the end effector of the robot's arm and by moving the hand or finger a computer can be controlled. The sensation of feeling that makes this a haptic device is caused by motors. This robot can generate forces in three degrees of freedom by using motors attached to its three joints. By appropriately powering the motors, forces up to 1.9 lbf can be generated by this device.

CHAPTER 2

INSTRUMENTATION

2.1 System Overview

The system developed to test finger walking with haptic feed back was a virtual simulation. The simulation consists of a walkway on which test subjects would travel by finger walking using the two haptic devices. The walkway has obstacles that need to be navigated which include; a set of three stairs, an inclined slope and a declined slope. The person playing the simulation can feel the virtual floor of the trail with their fingers. The sensation of touch is generated by the two haptic devices. The haptic devices chosen for this system were both the PHANToM Premium 1.0A by Sensable Technologies Inc. of Cambridge Massachusetts. Stereo rendering techniques were implemented to make the simulation 3-D. This was done to enhance the sense of depth and therefore, make the walkway easier to navigate. The 3-D image was generated using Crystal Eyes Workstation glasses by Stereographics. The simulation was developed in C++ using the OpenHaptics 2.0 and OpenGL libraries. OpenHaptics is a set of C++ libraries used by Sensable to control the haptic devices. OpenGL is a set of C++ libraries that are used for creating computer images, usually for videogames or movie special effects.

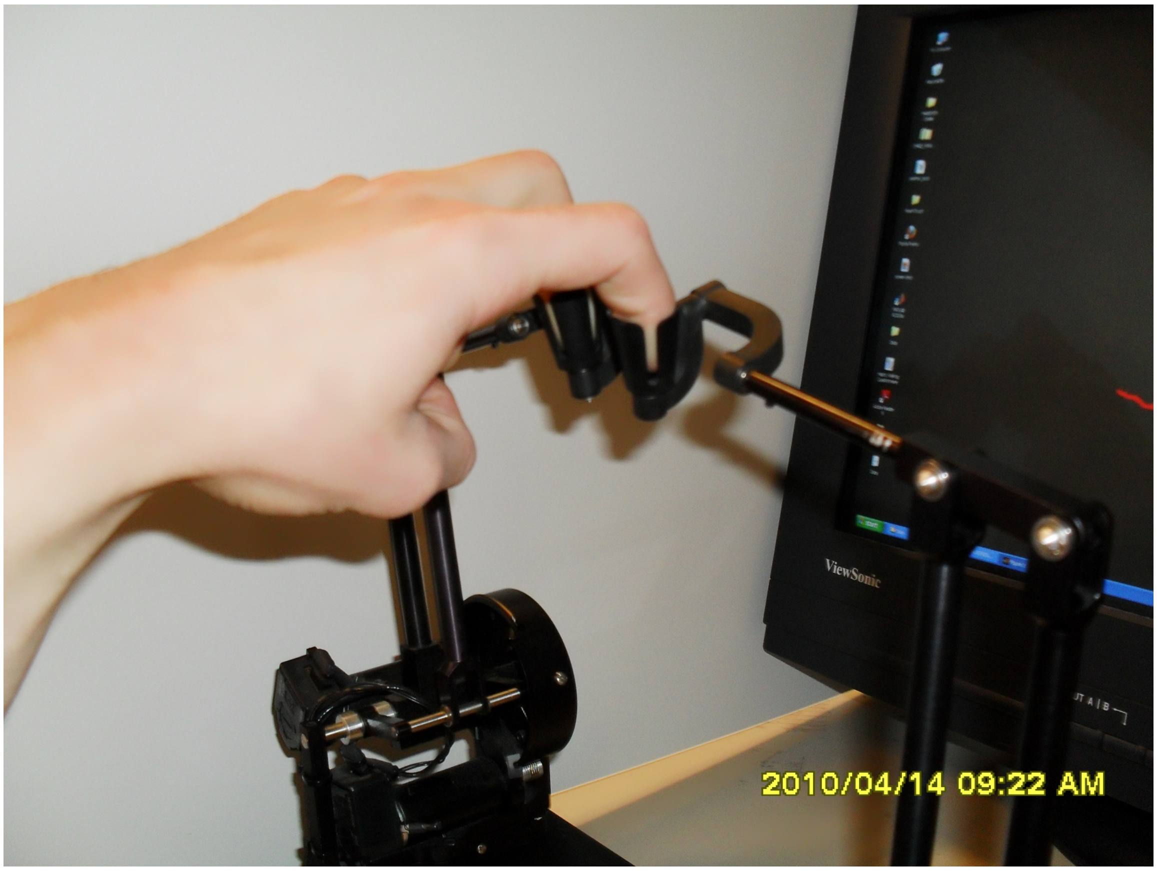

Figure 2.1 A picture of how a subject operates the game.

2.2 PHANToM Haptic Devices

The haptic devices chosen for this system were the PHANToM Premium 1.0A by Sensable Technologies Inc. of Cambridge Massachusetts. These devices were selected because of the size of their operating space, their ability to render forces, and the gimbal end effector that each has.

The size of the workspace of the device is 5 inches by 7 inches by 10 inches. Finger walking requires only a small volume to perform. Other devices manufactured by Sensable have ranges of motion which can accommodate whole arm movements. Those devices were not selected because their workspaces are superfluous to the needs of this system. The PHANToM Premium 1.0A has a workspace volume which is large enough to easily navigate the virtual trail using finger walking and not so large that there is excess space in the workspace.

The model chosen can render forces better than models of lower cost. The device’s PCI interface and large external amplifiers allow it to render forces more smoothly than other devices by the same manufacturer. Compared to lower cost models, the Premium 1.0A can also generate greater forces. The Premium 1.0A can generate 1.9 lbf while the Omni can only generate 0.75 lbf. The ability to render forces smoothly and generate greater forces enhanced the functionality of the system. The smoothness of the force rendering allows the virtual environment to have a viscous damping effect. This effect prevents the device from shaking when it makes contact with an object in the virtual environment by slowing high velocity movements. The high force generation is also necessary for proper game play. If the maximum force generated is too low, the person playing the game will experience their finger falling through the virtual surface if they were to apply a force in excess of the maximum. This would lead to difficulty in navigation of the virtual environment.

The end effector of this device is a thimble gimbal. The term thimble in the name refers to the thimble shaped cup in which the subject places his or her finger. A gimbal is a device that allows three degrees of freedom for rotations. With this thimble gimbal a subject can place his or her fingertip in the thimble and move about the environment without regard for the orientation of their finger. Other haptic devices use a pen style control arm that a user holds in their hand. The thimble gimbal did not need any instrumentation or adaptation to work for the application of finger walking.

The devices were arranged so that they were facing each other. This was necessary for two fingers of the same hand to be used. It this had not been done, the devices would have collided with each other during use. To accommodate for this rotation, the program was written to rotate the workspace of the devices so that what is normally the Z axis for the device is now the X axis. Also, the device on the left side has its end effector oriented into the positive X direction while the device on the right side is oriented into the negative X direction.

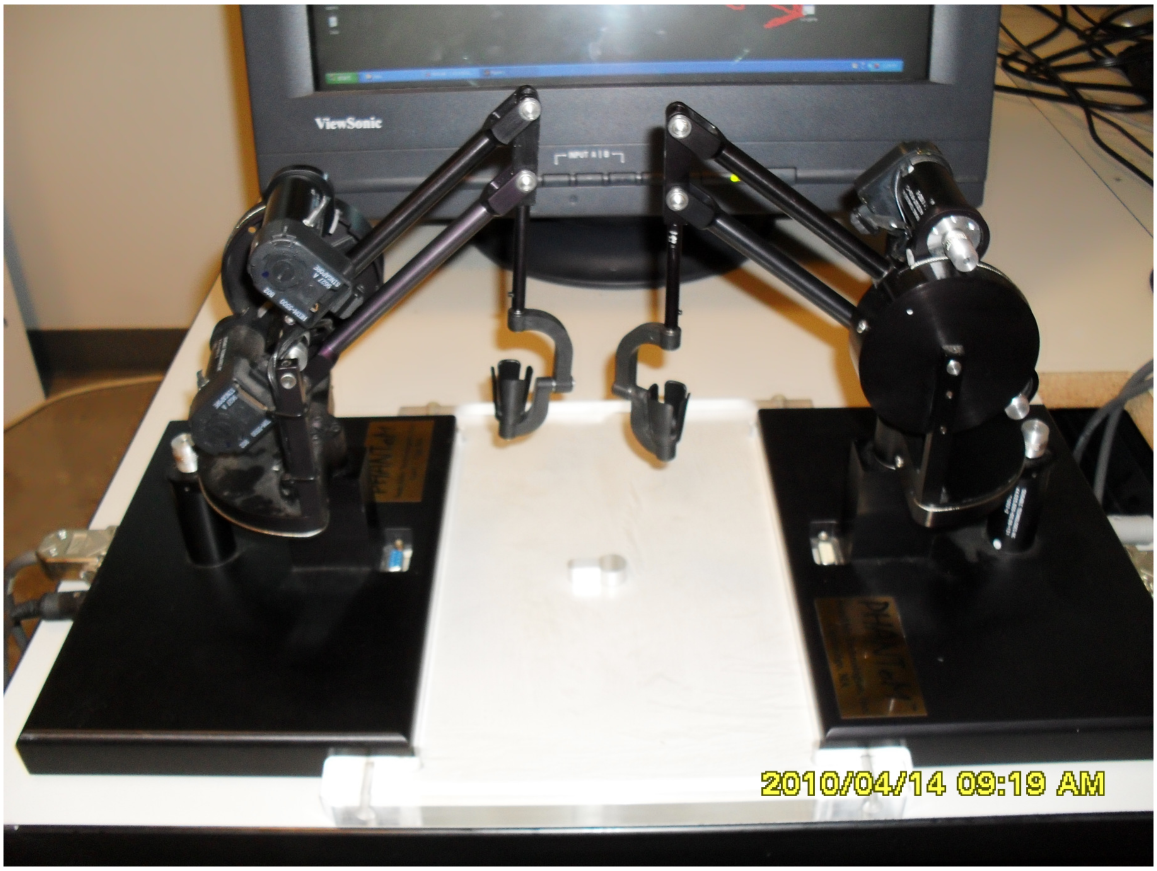

Figure 2.2 A picture of the phantoms in their appropriate configuration for game play.

The two devices interface with the computer via PCI card. One of the PCI cards was inserted into the highest PCI slot on the computer's motherboard and the remaining card was inserted into the lowest slot. This was essential to prevent computer errors. For the operating system and BIOS revision of the computer used, interrupt request settings are automatically assigned to devices depending on their type and position on the motherboard. For use in this computer, if the PCI cards are in any position other than the one described, the computer can not function properly. The system will either crash unexpectedly, or the haptic devices will not be able to properly communicate with the scheduler and therefore, will not render forces correctly.

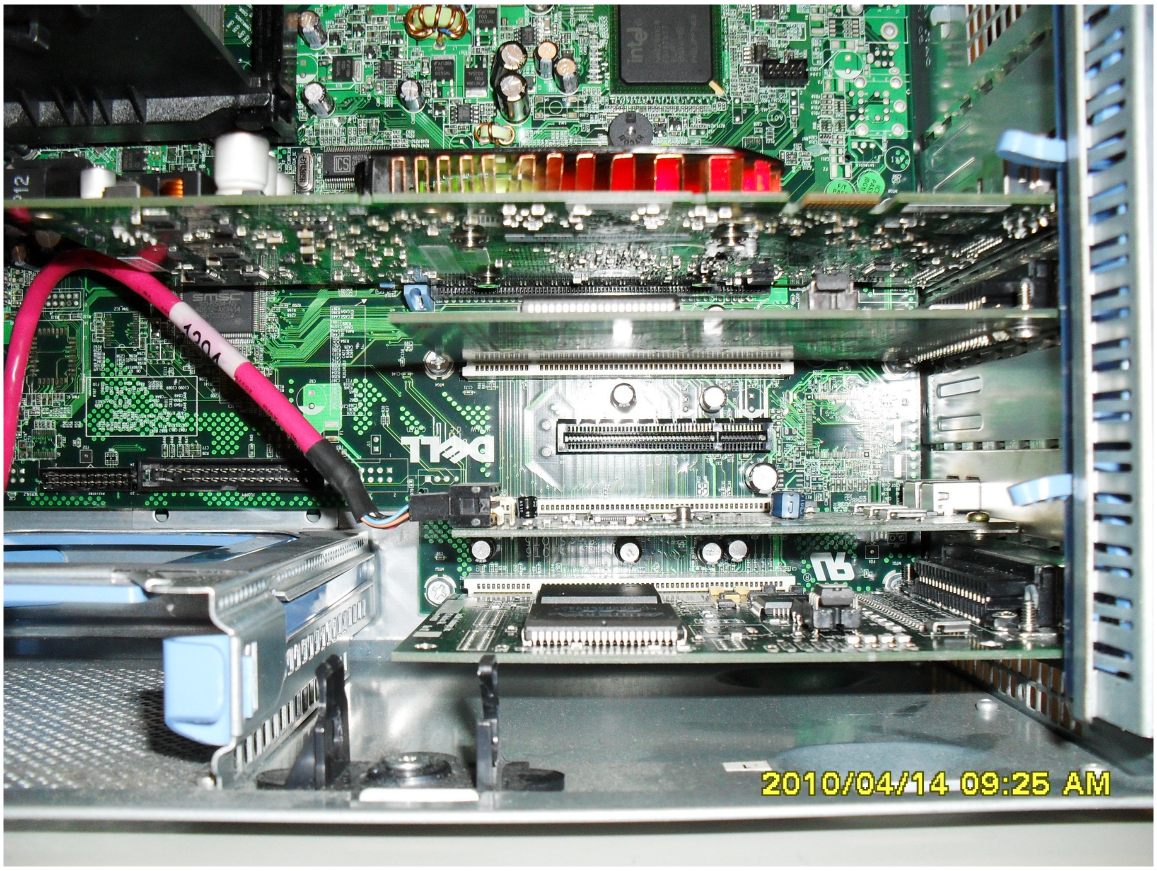

Figure 2.3 A picture of the proper positions for the PHANToM's PCI cards in the computer's motherboard. The highest slot is an unoccupied PCI express slot, the second slot is occupied by the graphics card, the third slot contains the first PHANToM PCI card, the fourth slot is an unoccupied PCI express slot, the fifth slot contains the fire wire card, and the sixth slot contains the second PHANToM PCI card.

2.3 OpenGL

OpenGL is a set of libraries for use in C++ which enables the generation of computer images. By linking these libraries to a C++ application, a software developer can use a set of simple functions to render images. These images are generally rendered by providing points in three dimensional space to a function which will link these points in an appropriate manner. OpenGL is the computer graphics industry's most widely used and supported 3-D graphics application programming interface.

The virtual environment was rendered using OpenGL. The environment was simple and was made using three main functions. These were quads, triangles, and quad strips. These are three methods of drawing a shape. Quads accepts sets of four points in 3-D space. The function then connects the four points appropriately to form a four sided shape. Quad strip accepts sets of two vertices for four sided faces of a shape. It then connects these faces together at common sides to create a strip of four sided shapes. Triangles accepts sets of three vertices and draws a triangle from each set. Using these three basic functions a trail was drawn and trees were placed beside the trail. The trees were included to provide a motion reference. This allowed users to recognize that forward progress was being made by seeing objects move closer relative to their motions.

The representation of the feet in the virtual environment was a pair of boots. This image was too difficult to generate, so it was acquired through the website TurboSquid.com, which is a site where 3-D graphics images can be purchased and downloaded. The file obtained from the website was of a form other than OpenGL code, so it was converted using the Okino Computer Graphics PolyTrans software. This software accepts 3-D images rendered with various formats and can convert them to various other formats. A piece of C++ code was written by PolyTrans, which was inserted into the application and rendered the boots in the game. Because the free evaluation version of PolyTrans was used, the boots have triangles missing from their mesh.

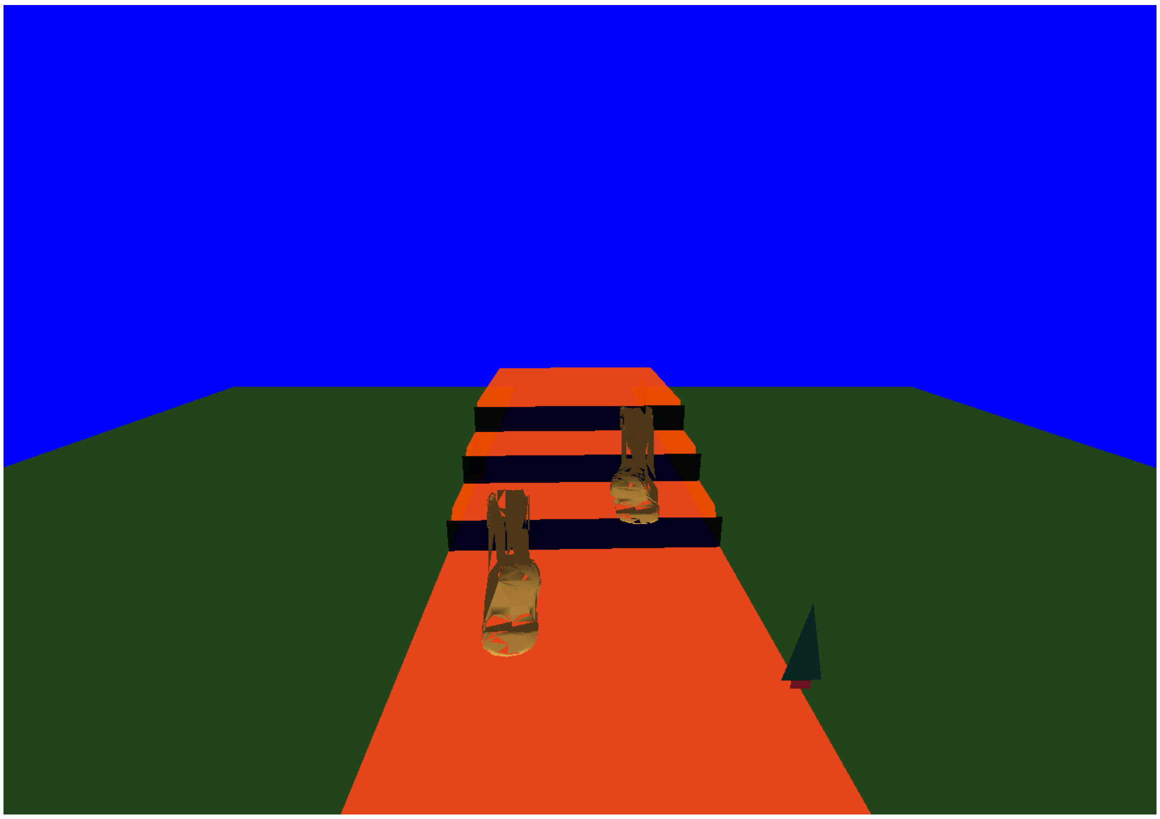

Figure 2.4 A screenshot of the virtual environment rendered in OpenGL.

2.4 OpenHaptics

OpenHaptics is the application programming interface utilized by Sensable for developing applications with their haptic devices. This interface builds off of OpenGL to render the haptics. In general, an image is drawn using OpenGL functions and it is sent to the haptic device with slight additions to the code. OpenHaptics was designed so that a programmer with experience in OpenGL can easily create high level graphical programs in OpenHaptics.

OpenHaptics 2.0 was used instead of OpenHaptics 3.0, which is the most up to date version of the software. The reason for this was because the haptic devices used were not recently purchased and the newest version of the software no longer supports the device used. If a newer device is used OpenHaptics 3.0 may then be used.

2.5 Stereo Rendering

Stereo rendering was used to create a three dimensional image. When rendering 3-D images, the image is drawn on the screen twice, once for the left eye and once for the right eye. Various systems exist to selectively allow the images to be viewed by only the appropriate eye. Special glasses, manufactured by Stereographics, were used to achieve this. The glasses can selectively prevent an image from entering either eye at a very high speed by shuttering the lenses open and closed. The glasses synchronize with the computer monitor's refresh rate so that when an image is being projected for the left eye only the left eye can see that image and vice versa for the right eye. Using these glasses, and appropriately drawing the image for either eye, presents the user with the illusion that the images are coming out of the screen.

To appropriately render the images so that the 3-D affect can be achieved, the view port was moved for each eye. When creating images in OpenGL, a viewpoint must be chosen. The point from which each eye views the image was set up along the same horizontal line and looked at the same point in space but, the two eye positions were slightly separated in the X direction. The left eye was placed slightly to the left of the point X = 0 and the right eye was placed the same distance to the right of X = 0. By doing this, binocular vision can be simulated because, one image is being projected for the left eye and a different image is projected for the right eye.

2.6 Computer Specifications

The computer used was a Dell Dimension 9100. In order to create images in 3-D using the Stereographics shutter glasses, a graphics card capable of quad buffering was necessary. The card used was an Nvidia Quadro FX-3500. Also, it was necessary to use a monitor with a high refresh rate in order to project images for either eye without creating discontinuities when the objects move. A Viewsonic Graphics series G225f was used for this. Using this graphics card and monitor combination a refresh rate of 100Hz was achieved.

CHAPTER 3

METHODS

3.1 Experimental Design

The first step of the experiment is the tutorial. This is an instruction period which lasts 5 to 10 minutes. During the tutorial, it is demonstrated to the subject how to perform finger walking and subjects are allowed to try the simulation in a free play mode. The free play mode is a continuous version of the simulation during which no data is collected. In addition, if a subject fails to have at least one finger in contact with the surface at any given time, the screen will turn red as a warning of a double stance error. This is because slow walking is defined as having a period of double stance. For this experiment we are only interested in slow walking.

After the tutorial is concluded, three versions of the simulation are run. The first simulation includes walking over flat terrain. The second simulation includes walking up and down three hills which have a 9° slope. The third simulation has four sets of stairs that the subject must navigate. When the end of the trail is reached the simulation ends automatically and the program terminates. During the simulation, data are recorded and stored for offline analysis. The data recorded are the positions of the fingers and the forces being sent to the haptic devices. Additionally, a binary variable is recorded which indicates if and when a double stance error has occurred.

After the game is concluded, the subjects are asked to fill out a questionnaire. The questionnaire asks; if any discomfort was felt, if the game was difficult to play, and if they have any suggestions for improvements.

All experimental protocols were approved by the NJIT IRB committee.

3.2 Data Analysis

Five primary variables were analyzed. These were; number of double stance errors, percentages of time spent in swing and stance phase during level and hill walking, the number of errors which occurred during stair walking, the shapes of the velocity and acceleration curves, and the results of the questionnaire. A double stance error refers to a moment when both fingers leave the surface simultaneously. A stair walking error is when a subject misses a step or slips off of a step.

In order to analyze this data, it was imported to Matlab. A code was written for analysis which determined the onset of each movement by analyzing the forces in the Y direction which were acting on each finger. The moment the force went from an upward value to a downward value, it marked the rise of the finger to initiate a movement. The end of each movement was marked by a stopping of the movement in the Y direction as observed from the position data. Once the onset and offset of each movement was determined, the percentage of time spent in stance and swing was analyzed by observing the time at which each movement began and ended.

To determine the velocities and accelerations of the fingers during the movements, the data were filtered and differentiated using a central difference function. The data were sampled at a rate around 50Hz. In order to obtain velocity and acceleration data that is meaningful, the position data must be filtered. The most effective filtering technique for this data was found to be interpolating the data to bring the sample rate up to 100Hz and then applying a second order low pass Butterworth filter with a cutoff frequency of 3Hz.

Additionally, stride length was analyzed for uphill and downhill finger walking. In order to do this, a technique was taken from treadmill walking gait analysis [18]. To determine stride length, the difference between the positions of one foot before and after a step has been taken are analyzed. This difference is considered to be the step length. The onset of movement and end of movement times have already been determined in order to find step timing. The onset and offset times were then paired with the horizontal position data in order to determine step length.

CHAPTER 4

RESULTS

4.1 Double Stance Errors

Double stance errors are defined as an instant during finger walking when neither finger is in contact with the surface. During the training session, the subjects were given an indication when a double stance error occurred however, during the testing period, no such warning was given.

Table 4.1 Number of Double Stance Errors for Each Subject Under Each Condition

Subject	**Level**	**Hills**	**Stairs**
AN	0	0	1
DS	5	5	0
GA	1	0	2
IL	2	3	14
KC	0	0	0

The minimum number of errors for a subject was 0 and the maximum number of errors for a subject was 14 in a single trial. The overall average number of errors for the entire experiment was 2.2 errors in a single trial.

4.2 Percentage of Time in Stance and Swing

For normal healthy walking, it is usually expected to observe a 60% stance and 40% swing distribution. In addition, the gait cycle of the left leg should be 180° out of phase with the right leg. For the five subjects, this was analyzed during the level and hill walking trials.

Table 4.2 Percentage Stance and Swing During Level Finger Walking

		Left Finger		Right Finger	
Subject	Hand used	Percent Stance	Percent Swing	Percent Stance	Percent Swing
AN	Left	63.0411	36.9589	57.737	42.263
DS	Right	54.9744	45.0256	59.5641	40.4359
GA	Right	67.1155	32.8845	72.7049	27.2951
IL	Left	69.8964	30.1036	61.9966	38.0034
KC	Right	63.8864	36.1136	58.8962	41.1038

Table 4.3 Percentage Stance and Swing During Hill Walking

		Left Finger		Right Finger	
Subject	Hand used	Percent Stance	Percent Swing	Percent Stance	Percent Swing
AN	Left	65.9287	34.0722	64.8696	35.1304
DS	Right	52.174	47.8253	53.551	46.449
GA	Right	66.8774	33.1226	67.5458	32.4542
IL	Left	67.8175	32.1825	56.9619	43.0381
KC	Right	57.1592	42.8408	50.8716	49.1284

During finger walking, the subjects show a gait cycle similar to that of a subject with an impaired limb. This is observed through the asymmetry in the percentage of time spent during stance and swing. For all subjects during level walking, the distributions were nearly 60:40 for one leg and a slightly longer time spent in stance for the other leg. This is similar to what would be observed for a patient with damage to one leg [10]. This is brought about because the index finger is shorter than the middle finger. In order to help counteract this effect, during the experiment, a virtual lengthening of the index finger was done by changing the end position of the index finger in the virtual environment to effectively equalize the lengths of the two fingers. However, the asymmetry was still observed even with this correction.

4.3 Step Errors

When walking on the stairs, all subjects were able to successfully navigate all four flights of stairs. However, each subject performed missed step errors. A Missed step error was defined as either landing on the wrong step or slipping off of a step.

Table 4.4 Number of Stair Walking Errors

	Number of errors	
Subject	Left finger	Right finger
AN	7	2
DS	6	9
GA	4	5
IL	2	3
KC	2	1

4.4 Position, Velocity and Acceleration

For all subjects, the velocity and acceleration curves had similar trends. Movement was similar to ankle data during walking except, there is no notch in the movement that would be attributed to the roll from heel to toe. Presented here are the vertical Y position, velocity and acceleration curves for level, hill, and stair walking.

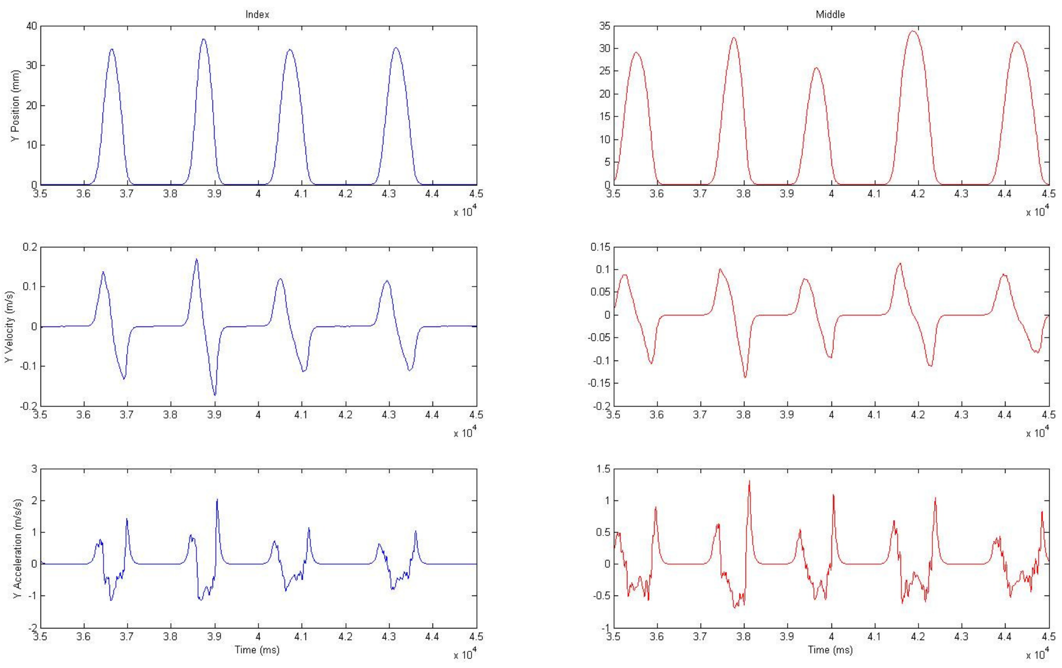

Figure 4.1 Position, velocity and acceleration of index and middle finger of one subject during level finger walking. The left finger is the index finger and the right finger is the middle finger of the subject's right hand.

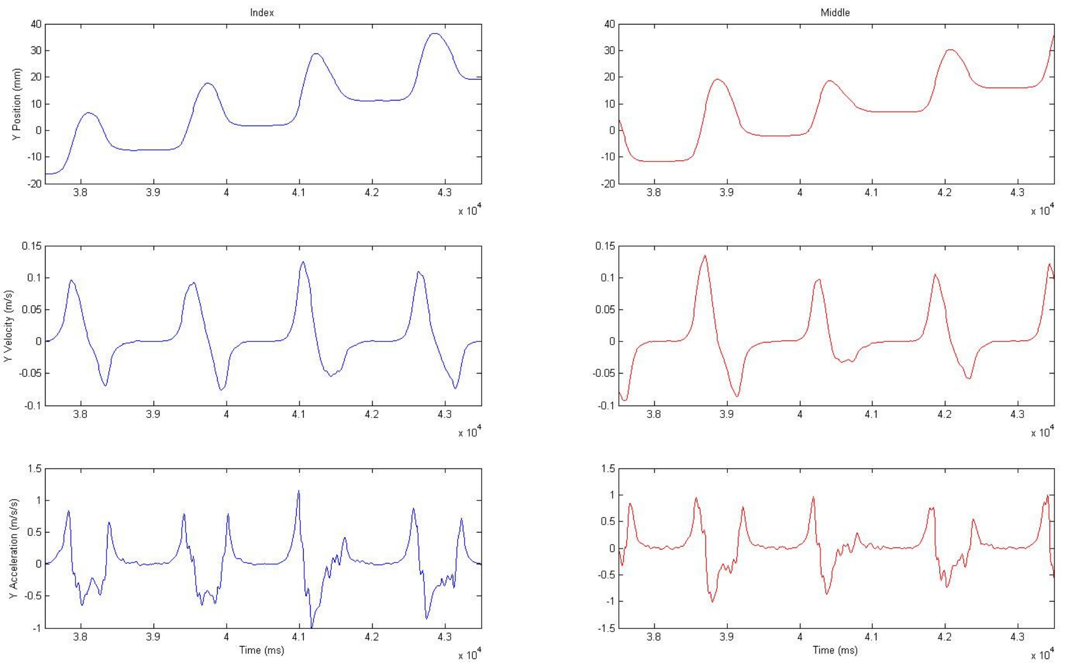

Figure 4.2 Position, velocity and acceleration plots of a single subject during uphill finger walking.

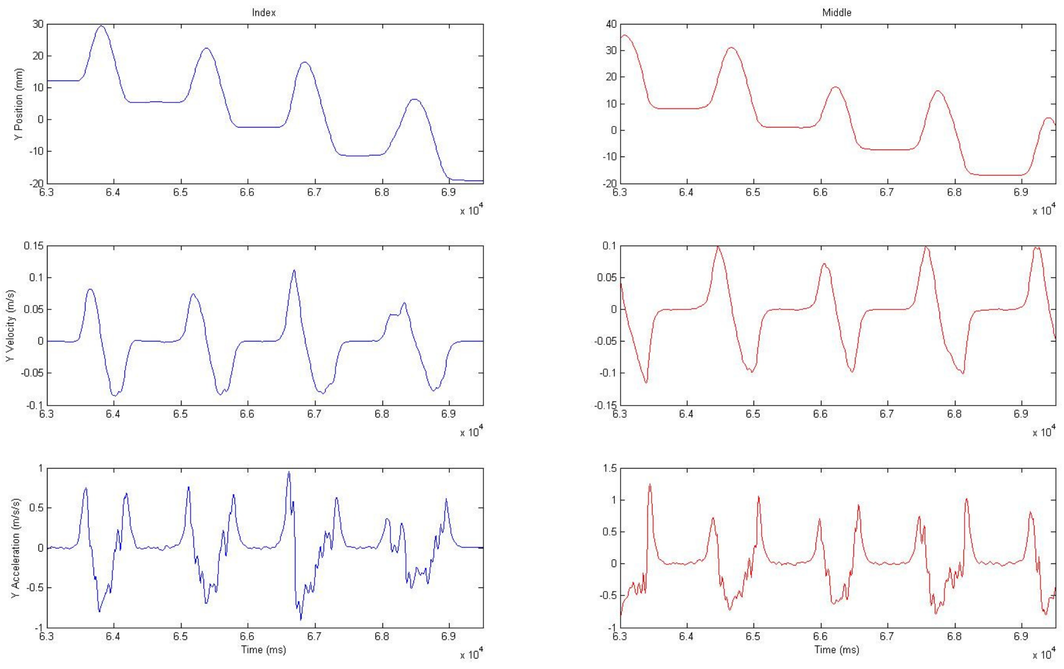

Figure 4.3 Position, velocity and acceleration plots of a single subject during downhill walking

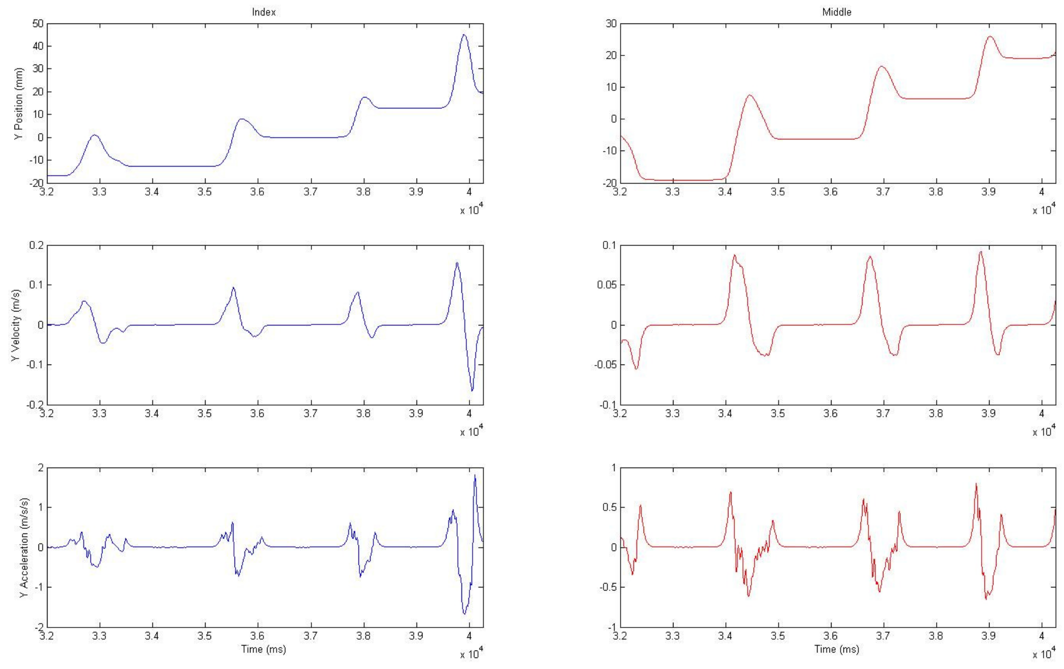

Figure 4.4 Position, velocity and acceleration of a single subject during finger walking up stairs.

If the data is normalized, by using a 20:3 ratio, then the movements can be more easily compared to normal human walking. The 20:3 ratio was a rough estimate of the vertical displacement of the foot during walking versus that of the finger during finger walking. The position data was therefore multiplied by 20/3 and then differentiated to obtain normalized position, velocity, and accelerations in the Y direction.

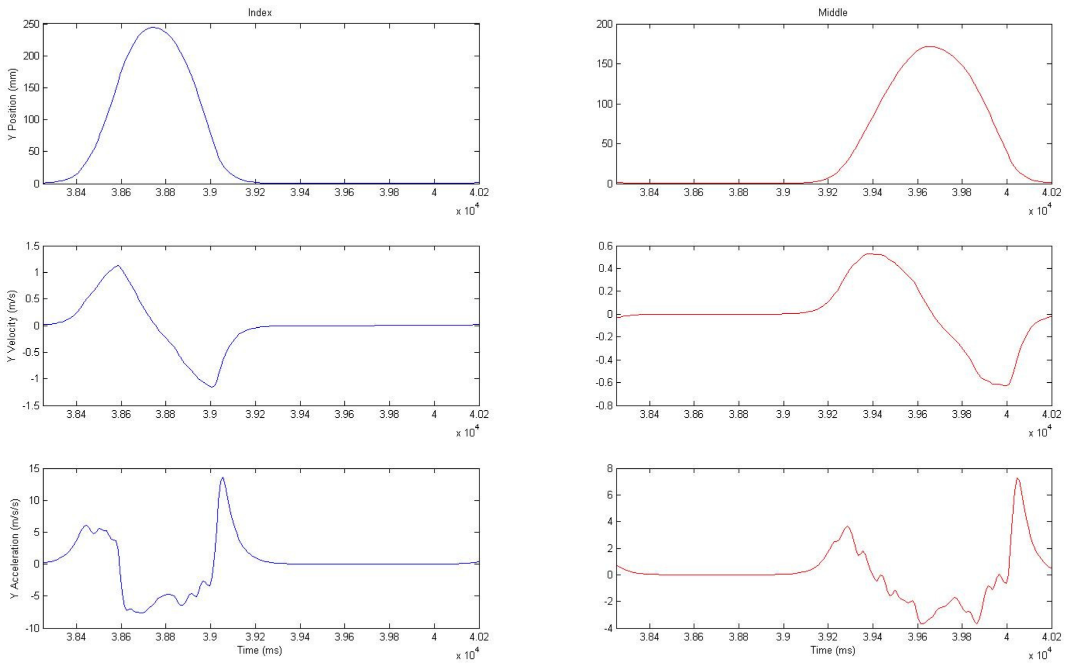

Figure 4.5 Normalized data for a single gait cycle of the fingers while moving over level terrain.

4.5 Questionnaire Results

The questionnaire asked if any discomfort was experienced and if the simulation was difficult to operate. If either was responded to with a yes answer, subjects were asked to elaborate as to why.

Table 4.5 Questionnaire Results

	Yes	No
Discomfort	0	5
Difficult	1	4

Of the five subjects, none experienced any pain or discomfort. Four of the five subjects did not find the simulation difficult to play. The remaining subject, who experienced difficulty, cited trouble with the viewing perspective used in the simulation.

4.6 Stride Length

For three subjects, stride length was measured during both uphill and downhill finger walking. The data did not show statistical significance nor did it follow the expected trend for any of the subjects analyzed.

Table 4.6 Average Uphill Stride Length Data

Subject	**Left Finger**	**Right Finger**
AN	**0.512**	**0.962**
DS	**0.431**	**0.987**
IL	**1.352**	**0.599**

Table 4.7 Average Downhill Stride Length Data

Subject	**Left Finger**	**Right Finger**
AN	**1.063**	**1.189**
DS	**0.563**	**1.219**
IL	**1.338**	**0.428**

CHAPTER 5

DISCUSSION OF RESULTS

5.1 Double Stance Errors

The number of double stance errors varied from subject to subject. The least number of errors for a single subject was 0 and the maximum number of errors observed in a single subject was 19. During walking, double stance is seen during the period between initial contact of one foot and toe off of the opposite foot. Movement without double stance can be observed if the subject is running. During finger walking, the weight of the hand is not supported by the fingers as the weight of the body is by the feet during normal walking. The weight of the hand is supported by the wrist; therefore, it is possible to move through the simulation without exhibiting double stance. This error was observed upon analysis of data from earlier trials. To help correct for this error, the addition of the warning indicator was made to the training program however, double stance errors were still seen in 4 of the 5 subjects. It can be seen from the results of subject KC that actively forcing double stance during walking can be done and walking can be performed without the errors. If more training is given to the subjects, with a specific emphasis on the double stance errors, subjects can learn to not make the errors.

5.2 Percentage of Time in Stance and Swing

All five subjects demonstrated dominance toward one side in their stride pattern. A healthy subject would exhibit 60% of their stride in stance and 40% in swing during normal walking. In general, all five subjects showed timing that tended toward 60:40

with a range from 53:47 to 73:27. However, none showed a precisely symmetrical gait pattern for both fingers. In general, one finger is established as the dominant finger and more time is spent in stance on that finger. This is a result of the difference in lengths of the two fingers. Subjects are exhibiting a gait pattern similar to that of a person with a shortened leg. In order to help correct for this, a virtual lift was added to the shortened finger; however, it does not perfectly correct the deficit. Most likely, the deficit is due more to the differences in the positions of the proximal interphalangeal joints of the index and middle fingers than to the total lengths of the fingers.

5.3 Step Errors

A step error was defined as missing the target step or slipping off of the step once on it. All subjects made step errors, the most being 15 and the least being 3. The majority of these errors were from slipping off of the surface once on it. This is more a deficiency in the simulation than the subject, in order to prevent unexpected kicking, which can occur if the virtual representation of the foot becomes stuck on the stairs, a very low coefficient of friction was used. If a higher friction coefficient was used, it would have made the stairs more easily climbable however, it risked injury to the user and damage to the equipment.

With the flaw of the slippery staircase, one subject was able to navigate a flight of six consecutive stairs flawlessly. This demonstrates that climbing stairs is achievable with finger walking. If subjects trained for a more extensive period of time, it is more likely that stair navigation could be more easily done.

5.4 Position, Velocity and Acceleration

In order to obtain smooth acceleration curves, the position data had to be filtered with a 3Hz cut off frequency. The motions of the fingers are less than 1 cycle per second, so none of the important motions are being filtered however, with any filtering data distortion can be expected.

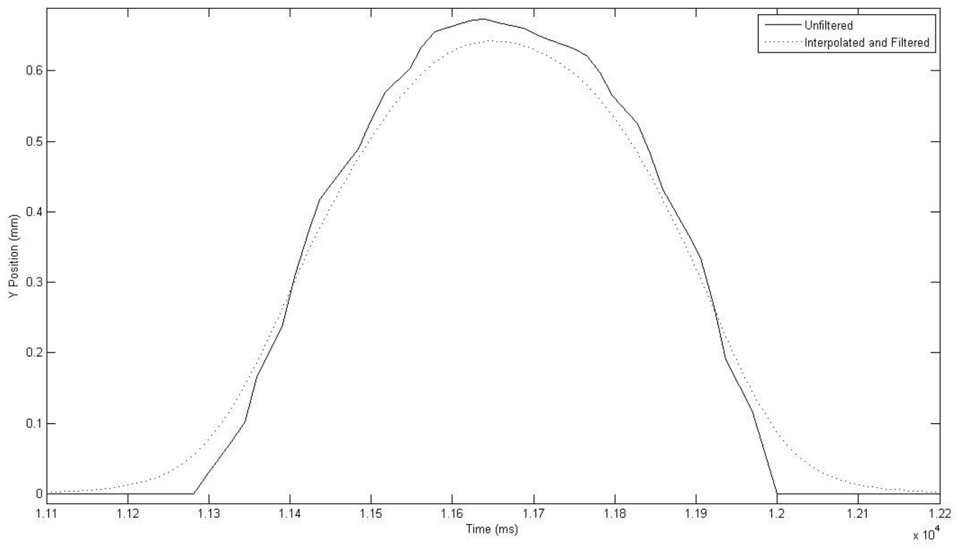

Figure 5.1 Example of distortion due to filtering.

The filtering at the low frequency distorted the values however it allowed for determination of velocity and acceleration. Due to the method of differentiation, which measured the differences between the values before and after each point and divided by the time for the movement, any noise is amplified when velocity is calculated and it is further amplified in the acceleration. By filtering out any sharp changes in the position data, noise can be eliminated in the velocity and acceleration curves, however, the amplitudes are affected.

The shapes of the vertical movement trajectory, the velocity, and the acceleration profile, show similarities to walking. The positions move in a manner similar to that of the ankle during normal walking, with the exception of the peak at the end of the step which is attributed to the foot rolling from the heel to the toe. Since the finger has a rounded end, there is no increase in the elevation, like the ankle has during walking. The velocity profiles resemble those of normal walking in which a movement has a positive velocity for the first half of the movement and a negative velocity for the second half of the movement. The approximate symmetry of the positive velocity and the negative velocity demonstrates a planning in the movement. The subjects move their fingers to the point they are trying to reach and then slow down until impact. This type of motion is further demonstrated by the acceleration curves, in which the motion begins with a positive acceleration that climbs to a point a quarter of the way through the movement before it starts to slow the positive acceleration. At a point mid way through the movement the acceleration becomes negative as the finger moves downward, it reaches a peak negative acceleration and then the negative acceleration decreases until impact with the surface, at which time there is a large positive acceleration for a moment before the finger comes to rest.

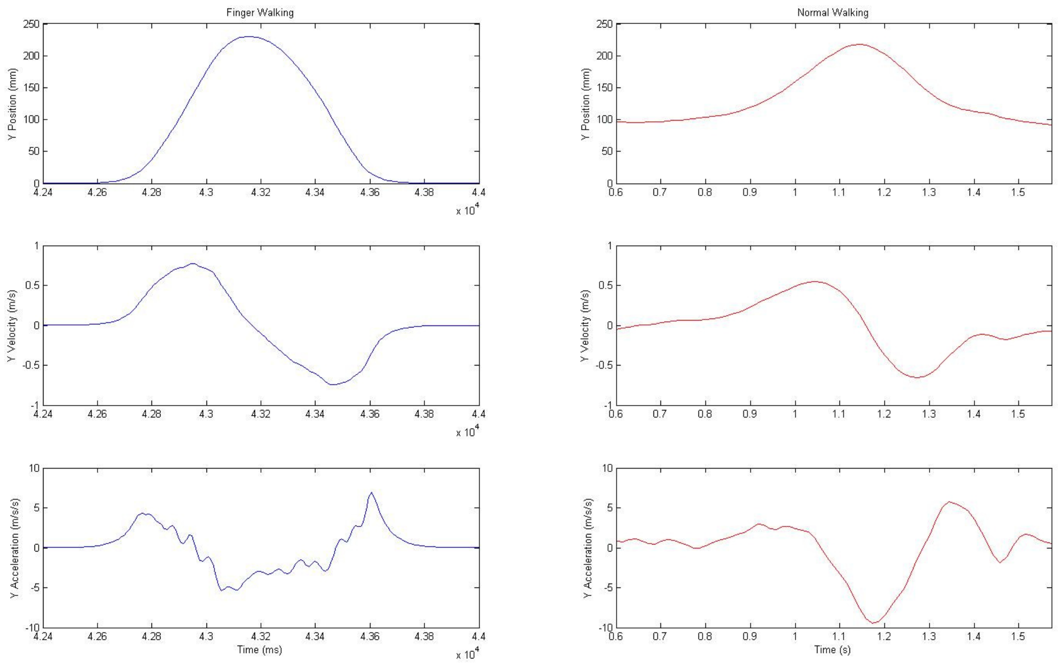

Figure 5.2 Normalized finger movements compared to ankle walking data.

Figure 5.2 was created by multiplying the finger positions by a 20:3 scaling factor, this was determined to be an appropriate approximation to the scaling factor between finger movements and foot movements during walking. The movements were then differentiated to determine the velocities and accelerations. The right column of the figure comes from analysis of published ankle position data during walking from Winter's text [20]. The ankle position data was filtered using the same filter as the finger positions, appropriately changed to accommodate for the different sampling frequency. It was then differentiated to obtain velocity and acceleration. Looking at this figure, you can see great similarity between normal walking and finger walking. When multiplied by the scaling factor, the values of peak velocity and acceleration come out to be very close to one another. In normal walking, the peak negative acceleration is almost double the peak positive acceleration, but this is not observed in finger walking. However, the finger walking data did not filter and differentiate as smoothly as the normal walking data and it may be that the same peak values are met but noise has distorted them sufficiently.

In stair climbing, the results that demonstrated control and planning are also evident. In order to effectively navigate the stairs a greater level of control is required. Generally a slowing of the finger can be seen just before impact with the stair. This results in a smaller value for the positive acceleration at the end of the movement because the movement is being brought to a stop over a greater period of time.

In the first step of the left finger in Figure 4.4, a slight positive acceleration can be seen before a negative acceleration and the final positive acceleration that ends the movement. This movement was made as a last minute correction. The subject decided late in the movement to advance his finger more forward, so he slowed, made the adjustment, and then finished the movement. This demonstrates an ability to adapt with finger walking just as can be seen with normal walking.

In addition, it should be noted that the subject was able to learn the positions of the stairs after the first step. There was an obvious error when approaching the first step that the subject was able to compensate for, but each step after the first showed more control and less hesitation. The fastest movement was the final step up, which shows that subjects were able to learn the height of the step quickly and adapt to that height by the sixth stair.

The finger movements closely mimic ankle positions. Using these data, inverse kinematics can be used to determine virtual trajectories for the hip and knee joints. These virtual trajectories, and the timing obtained from the finger movements, can be used in order to enable FES walking.

5.5 Questionnaire Results

The questionnaire asked two yes or no answer questions. Did you experience any discomfort during the simulation? Did you find the simulation difficult? If either of these questions were answered yes then the subject was asked to further elaborate. Of the five subjects only one answer of yes was recorded. The subject found it difficult to lift one foot higher than the other when moving uphill and became stuck momentarily. In addition, the subject had difficulty with the viewpoint. The subject's difficulty in lifting up his fingers to navigate the hill could be attributed to fatigue. He had already successfully navigated two hills before that, so it was a feat that the subject had accomplished previously. The viewpoint issue comes about because the point through which the simulation is perceived is fixed, but the virtual representation of the feet can move out of the range of this fixed field of view. Four of the five subjects did not have a difficulty with this however, in future research, it would be beneficial to come up with a different viewpoint.

5.6 Stride Length

According to various papers, stride length should lengthen going uphill and shorten going downhill [19]. The data did not show statistical significance and they did not follow the expected trend. This can be attributed to the lack of weight supported by the finger tips. When walking up hill stride length increases while cadence decreases. This is done to accommodate for having to step higher to overcome an obstacle. When walking down hill stride length decreases while cadence increases. This is done because more control is needed to safely lower the center of gravity of the body. Because the hand is not actually supporting any weight, the steps do not significantly change in length or cadence from flat finger walking.

CHAPTER 6

CONCLUSIONS

Of the five subjects tested, all were able to successfully navigate each of the three types of terrain. One subject in particular did exceptionally well in that he had no double stance errors and was able to navigate the stairs with only three miss steps. This demonstrates that finger walking with haptic feedback is a feasible means of navigating virtual environments. The level walking did closely resemble normal walking in the velocity and acceleration profiles. However, finger walking on hills did not resemble normal walking. Stair climbing was achieved and with practice can be improved. In order to extend finger walking to natural environments, certain conditions must be met. The fingers need to be in support of the mass they are moving. The hand can travel freely in space without the fingers because the weight of the hand is supported by the arm. Either the subject needs to train extensively in order to not misstep and fall or, some form of algorithm must be written which can restrict the movements of the fingers to those that can be achieved by the legs. Also, the stride timing resembles that of a person with unilateral leg damage. This is because one finger is shorter than the other which, effectively, makes the person walk as if one leg is shorter than the other. Lengthening of the finger in the virtual environment was not enough to correct this issue. In order to more thoroughly enable natural walking with fingers, the proximal interphalangeal joint of the index finger must be effectively brought to the same position as that of the middle finger. If motion sensors are placed at the interphalangeal joints of each finger, and the haptic devices remain at the end of each finger, an algorithm can be written to determine the appropriate end position of the foot. This could improve the cadence to that of a healthy subject.

This research has demonstrated that significant work needs to be done in order to safely navigate hills and uneven terrain using the finger walking technique. However, this research has also supported the findings of both Mathew Noesner's thesis and the work of Ji-Sun Kim and colleagues; it demonstrates that finger movements can mimic leg movements during walking and that this technique can be used to navigate virtual environments. This research also advances the navigation of virtual environments beyond what has been done previously because it allows for navigation of 3-D and not just 2-D environments.

APPENDIX A

HAPTIC WALKING CODE

The code for the haptic walking game implemented in C++ is provided below.

```
/**********************************************************************
Mark Shaker's code for a haptic environment. with stereovision
This was adapted from an openhaptics example called hellospheredual
**********************************************************************/

// 2/23/2010

#include <math.h>
#include <assert.h>
#include "C:\Documents and Settings\Mark\Desktop\PHANToM
Codes\boots\Moving Stairs\boot.c"


#if defined(WIN32)
# include <windows.h>
#endif

#if defined(WIN32) || defined(linux)
# include <GL/glut.h>
#elif defined(__APPLE__)
# include <GLUT/glut.h>
#endif

#include <HL/hl.h>
#include <HDU/hduMatrix.h>
#include <HDU/hduError.h>

#include <HLU/hlu.h>

#include <conio.h>
#include <stdio.h>
//#include <time.h>

#define           EYESEP            0.30
#define           FOCALLENGTH 3.0
static HHD hHD1 = HD_INVALID_HANDLE;
static HHD hHD2 = HD_INVALID_HANDLE;
static HHLRC hHLRC1 = 0;
static HHLRC hHLRC2 = 0;
static double alf=0.90;

FILE *pFile;


// shape id for shape we will render haptically
HLuint objectShapeId1;
HLuint objectShapeId2;
HLuint effectId1;
HLuint effectId2;
HLdouble zTranslation;

HLdouble device1ProxyPosition[3];
HLdouble device2ProxyPosition[3];
HLdouble device1Force[3];
HLdouble device2Force[3];
HLboolean device1Contact;
HLboolean device2Contact;
HLdouble ZProxy1;
HLdouble ZProxy2;
HLdouble lastProxy1;
HLdouble lastProxy2;
HLdouble defaultPhantomTransform = -90;
HLdouble Phantom2Transform = 90;
HLdouble defaultxtransform = -7;
HLdouble secondxtransform = 7;
long int initial_time;

#define CURSOR_SIZE_PIXELS 20
static double gCursorScale;
static GLuint gCursorDisplayList = 0;
bool stereo = true;

/* Function prototypes. */
void glutDisplay(void);
void glutReshape(int width, int height);
void glutIdle(void);
void glutMenu(int);

void exitHandler(void);

void initEffects();
void initGL();
void initHD(HDstring pConfigName, HHD &hHD);
void initHL(HHD hHD, HHLRC &hHLRC, HLuint &shapeId);
void initScene();
void drawObjectHaptics(HLuint shapeId);
void drawSceneGraphics();
void ObjectGraphics();
void realativeMotionObjects();
void drawCursor(HLfloat angle);
void drawSceneDamping(HHLRC &hHLRC, HLuint &effectId);
void updateWorkspace(HLdouble transform,HLdouble xtransform);
void MoveIt();
void Draw();
void DataCollection();

/**********************************************************************
*********
Initializes GLUT for displaying a simple haptic scene
**********************************************************************
********/
int main(int argc, char *argv[])
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH |
GLUT_STEREO);

      glutInitWindowSize(500, 500);
      glutCreateWindow("Mark's Stairs");

      /* Set glut callback functions. */
      glutDisplayFunc(glutDisplay);
      glutReshapeFunc(glutReshape);
      glutIdleFunc(glutIdle);

      glutCreateMenu(glutMenu);
      glutAddMenuEntry("Quit", 0);
      glutAttachMenu(GLUT_RIGHT_BUTTON);

      /* Provide a cleanup routine for handling application exit. */
      atexit(exitHandler);

      initScene();

      //effect
      drawSceneDamping(hHLRC1, effectId1);
      drawSceneDamping(hHLRC2, effectId2);

      //open the file for data collection
      pFile = fopen("C:/Documents and Settings/Mark/Desktop/Data.txt",
"w");

      initial_time=GetTickCount();

      glutMainLoop();

      //close the data file
      fclose(pFile);

      return 0;
}

/*********************************************************************
*********
GLUT callback for redrawing the view
**********************************************************************
********/
void glutDisplay()
{
      MoveIt();

      DataCollection();

      hlMakeCurrent(hHLRC1);
      drawObjectHaptics(objectShapeId1);

      hlMakeCurrent(hHLRC2);
      drawObjectHaptics(objectShapeId2);

      //drawSceneGraphics();
      Draw();

      glutSwapBuffers();
}

/******************************************************************
*********
GLUT callback for reshaping the window. This is the main place where
the
viewing and workspace transforms get initialized.
*******************************************************************
********/
void glutReshape(int width, int height)
{
      static const double kPI = 3.1415926535897932384626433832795;
      static const double kFovY = 100;

      double nearDist, farDist, aspect;

      glViewport(0, 0, width, height);

      /* Compute the viewing parameters based on a fixed fov and
viewing
      * a canonical box centered at the origin */

      nearDist = 1.0 / tan((kFovY / 2.0) * kPI / 180.0);
      farDist = nearDist + 6;
      aspect = (double) width / height;

      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(kFovY, aspect, nearDist, farDist);
      //gluPerspective(50, aspect, 10, 1700);

      /* Place the camera down the Z axis looking at the origin */
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();

      gluLookAt(0.0, 2.0, farDist,    //places camera
                  0.0, 0.5, 0.0,  //aims camera lens towards this point
                  0.0, 1.0, 0.0); //defines which way is up

      hlMakeCurrent(hHLRC1);
      updateWorkspace(defaultPhantomTransform, defaultxtransform);
      hlMakeCurrent(hHLRC2);
      updateWorkspace(Phantom2Transform, secondxtransform);
}
/******************************************************************
*******
 Draw stereo

*******************************************************************
****/
void Draw()
            // Draw Our Scene
{
      static const double kPI = 3.1415926535897932384626433832795;
      static const double kFovY = 100;

      double nearDist, farDist;
      nearDist = 1.0 / tan((kFovY / 2.0) * kPI / 180.0);
      farDist = nearDist + 6;

      /* Compute the viewing parameters based on a fixed fov and
viewing
      * a canonical box centered at the origin */

      glFlush();
      glMatrixMode(GL_MODELVIEW);

      glDrawBuffer(GL_BACK_LEFT);

      glPushMatrix();

      glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);        //
Clear Screen And Depth Buffer
      glLoadIdentity();
            // Reset The Modelview Matrix


      gluLookAt(-EYESEP/2, 2.0, farDist,
                  0.0, 0.5, 0.0,
                  0.0, 1.0, 0.0);

      glEnable (GL_BLEND); glBlendFunc (GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA);

      drawSceneGraphics();
      glPopMatrix();


    // Draw the right eye view
    glFlush();

      glDrawBuffer(GL_BACK_RIGHT);

      glPushMatrix();

      glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);        //
Clear Screen And Depth Buffer
      glLoadIdentity();
            // Reset The Modelview Matrix

      gluLookAt(EYESEP/2, 2.0, farDist,
                  0.0, 0.5, 0.0,
                  0.0, 1.0, 0.0);

      glEnable (GL_BLEND); glBlendFunc (GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA);

      drawSceneGraphics();

      glPopMatrix();
            // Restore The Old Projection Matrix
```

```
  //glFlush();
                    // Flush The GL Rendering Pipeline
      glutSwapBuffers();

}
/*******************************************************************
*********
GLUT callback for idle state. Use this as an opportunity to request a
redraw.
********************************************************************
********/
void glutIdle()
{
      glutPostRedisplay();
}

/*******************************************************************
********
Popup menu handler
********************************************************************
*******/
void glutMenu(int ID)
{
      switch(ID) {
            case 0:
                  exit(0);
                  break;
      }
}

/*******************************************************************
*********
Initialize the scene. Handle initializing both OpenGL and HL
********************************************************************
********/
void initScene()
{
      initGL();

      // Initialize HDAPI first, so that the device instances exist in
the system
      // All device instances need to exist before starting the
scheduler,
      // which gets started automatically by the first created context
      initHD("Default PHANToM", hHD1);
      initHD("PHANToM 2", hHD2);

      // Initialize the contexts and give each one a handle to a device
instance
      initHL(hHD1, hHLRC1, objectShapeId1);
      initHL(hHD2, hHLRC2, objectShapeId2);
}

/*******************************************************************
*********
Setup general OpenGL rendering properties, like lights, depth
buffering, etc.
```

```
**********************************************************************
********/
void initGL()
{
      static const GLfloat light_model_ambient[] = {0.3f, 0.3f, 0.3f,
1.0f};
      static const GLfloat light0_diffuse[] = {0.9f, 0.9f, 0.9f, 0.9f};
      static const GLfloat light0_direction[] = {0.0f, -0.4f, 1.0f,
0.0f};

      /* Enable depth buffering for hidden surface removal. */
      glDepthFunc(GL_LEQUAL);
      glEnable(GL_DEPTH_TEST);

      /* Cull back faces. */
      glCullFace(GL_BACK);
      glEnable(GL_CULL_FACE);

      /* Other misc features. */
      glEnable(GL_LIGHTING);
      glEnable(GL_NORMALIZE);
      glShadeModel(GL_SMOOTH);

      glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE);
      glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE);
      glLightModelfv(GL_LIGHT_MODEL_AMBIENT, light_model_ambient);
      glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
      glLightfv(GL_LIGHT0, GL_POSITION, light0_direction);
      glEnable(GL_LIGHT0);
}

/*********************************************************************
*********
Initialize an HD device instance
**********************************************************************
********/
void initHD(HDstring pConfigName, HHD &hHD)
{
      HDErrorInfo error;

      hHD = hdInitDevice(pConfigName);
      if (HD_DEVICE_ERROR(error = hdGetError()))
      {
            hduPrintError(stderr, &error, "Failed to initialize haptic
device");
            fprintf(stderr, "Press any key to exit");
            getchar();
            exit(-1);
      }

      printf("Found device model: %s / serial number: %s.\n\n",
            hdGetString(HD_DEVICE_MODEL_TYPE),
hdGetString(HD_DEVICE_SERIAL_NUMBER));
}


/*********************************************************************
```

```
*********
Initialize an HL rendering context for a particular device instance
**********************************************************************
********/
void initHL(HHD hHD, HHLRC &hHLRC, HLuint &shapeId)
{
      hHLRC = hlCreateContext(hHD);
      hlMakeCurrent(hHLRC);

      // Enable optimization of the viewing parameters when rendering
      // geometry for OpenHaptics
      hlEnable(HL_HAPTIC_CAMERA_VIEW);

      // Specify front face touchability only
      hlTouchableFace(HL_FRONT);

      // generate id's for the three shapes
      shapeId = hlGenShapes(1);
}

/*********************************************************************
*********
This handler will get called when the application is exiting.
Deallocates any state and cleans up.
**********************************************************************
********/
void exitHandler()
{
      // deallocate the shape id we reserved in in initHL
      hlDeleteShapes(objectShapeId1, 1);
      hlDeleteShapes(objectShapeId2, 1);

      // free up the haptic rendering context
      hlMakeCurrent(NULL);

      if (hHLRC1 != NULL)
      {
            hlDeleteContext(hHLRC1);
      }

      if (hHLRC2 != NULL)
      {
            hlDeleteContext(hHLRC2);
      }

      // free up the haptic device
      if (hHD1 != HD_INVALID_HANDLE)
      {
            hdDisableDevice(hHD1);
      }

      if (hHD2 != HD_INVALID_HANDLE)
      {
            hdDisableDevice(hHD2);
      }
}
```

```
/*******************************************************************
*********
Use the current OpenGL viewing transforms to initialize a transform for
the
haptic device workspace so that it's properly mapped to world
coordinates.
********************************************************************
********/
void updateWorkspace(HLdouble transform,HLdouble xtransform)
{
      GLdouble modelview[16];
      GLdouble projection[16];
      GLint viewport[4];

      glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
      glGetDoublev(GL_PROJECTION_MATRIX, projection);
      glGetIntegerv(GL_VIEWPORT, viewport);

      hlMatrixMode(HL_TOUCHWORKSPACE);
      hlLoadIdentity();
      hlRotated(transform,0,1,0);
      hlTranslated(xtransform,200,0);

      /* fit haptic workspace to view volume */
      hluFitWorkspace(projection);

      /* compute cursor scale */
      gCursorScale = hluScreenToModelScale(modelview, projection,
viewport);
      gCursorScale *= CURSOR_SIZE_PIXELS;
}
/*******************************************************************
*********
The main routine for displaying the scene. Get the latest snapshot of
state
from the haptic thread and use it for displaying a 3D cursor.
********************************************************************
********/
void drawSceneGraphics()
{
      glClearColor(0.0,0.0,1.0,0.0);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

      // draw 3D cursor at haptic device position
      hlMakeCurrent(hHLRC1);
      drawCursor(-90);
      hlMakeCurrent(hHLRC2);
      drawCursor(90);

      //draws the non haptic scene
      glPushMatrix();
      glTranslated(0,0,zTranslation);
      realativeMotionObjects();
      glPopMatrix();
      glEnable(GL_COLOR_MATERIAL);

      //draws the object specified in draw object graphics
      glColor4d(1.0,0.255,0.0,alf);
      glPushMatrix();
      glTranslated(0,0,zTranslation);
      ObjectGraphics();
      glPopMatrix();
      glEnable(GL_COLOR_MATERIAL);
}
/*********************************************************************
*********
A function to draw an object graphically that can be plugged into
the draw object haptics function to save space rewriting it
*********************************************************************
********/
void ObjectGraphics()
{
      glBegin(GL_QUAD_STRIP);
      //opening stretch
      glVertex3f(-1,0.0,15);
      glVertex3f(1,0.0,15);
      glVertex3f(-1,0,-10);
      glVertex3f(1,0,-10);
      //stairs with sloped backside
      glVertex3f(-1,0.25,-10.15);
      glVertex3f(1,0.25,-10.15);
      glVertex3f(-1,0.25,-10.5);
      glVertex3f(1,0.25,-10.5);
      glVertex3f(-1,0.5,-10.65);
      glVertex3f(1,0.5,-10.65);
      glVertex3f(-1,0.5,-11);
      glVertex3f(1,0.5,-11);
      glVertex3f(-1,0.75,-11.15);
      glVertex3f(1,0.75,-11.15);
      glVertex3f(-1,0.75,-12.5);
      glVertex3f(1,0.75,-12.5);
      //second stretch
      glVertex3f(-1,0,-14.5);
      glVertex3f(1,0,-14.5);
      //hill
      glVertex3f(-1,0,-20);
      glVertex3f(1,0,-20);
      glVertex3f(-1,0.25,-25);
      glVertex3f(1,0.25,-25);
      glVertex3f(-1,0.25,-27);
      glVertex3f(1,0.25,-27);
      glVertex3f(-1,0,-29);
      glVertex3f(1,0,-29);
      //third stretch
      glVertex3f(-1,0,-35.0);
      glVertex3f(1,0,-35.0);
      //chasm
      glVertex3f(-1,-0.2,-40);
      glVertex3f(1,-0.2,-40);
      glVertex3f(-1,-0.2,-42);
      glVertex3f(1,-0.2,-42);
      glVertex3f(-1,0,-47);
      glVertex3f(1,0,-47);
      //fourth stretch
      glVertex3f(-1,0,-67);
      glVertex3f(1,0,-67);
      glEnd();
}
void realativeMotionObjects()
{
      //green trianlges for the tops of trees
      glColor4d(0.034,0.139,0.034,alf);
      glBegin(GL_TRIANGLES);
      glVertex3f(-1.1,.05,1.0);glVertex3f(-0.9,0.05,1.0);glVertex3f(-
1,0.5,1.0);
      glVertex3f(0.9,0.05,-1.0);glVertex3f(1.1,0.05,-
1.0);glVertex3f(1.0,0.5,-1.0);
      glVertex3f(-1.1,0.05,-3.0);glVertex3f(-0.9,0.05,-
3.0);glVertex3f(-1,0.5,-3.0);
      glVertex3f(0.9,0.05,-5.0);glVertex3f(1.1,0.05,-
5.0);glVertex3f(1.0,0.5,-5.0);
      glVertex3f(-1.1,0.05,-7.0);glVertex3f(-0.9,0.05,-
7.0);glVertex3f(-1,0.5,-7.0);
      glVertex3f(0.9,0.05,-9.0);glVertex3f(1.1,0.05,-
9.0);glVertex3f(1.0,0.5,-9.0);
      glVertex3f(-1.1,0.05,-11.0);glVertex3f(-0.9,0.05,-
11.0);glVertex3f(-1,0.5,-11.0);
      glVertex3f(0.9,0.05,-13.0);glVertex3f(1.1,0.05,-
13.0);glVertex3f(1.0,0.5,-13.0);
      glVertex3f(-1.1,0.05,-15);glVertex3f(-0.9,0.05,-15);glVertex3f(-
1,0.5,-15);
      glVertex3f(0.9,0.05,-17);glVertex3f(1.1,0.05,-
17);glVertex3f(1.0,0.5,-17);
      glVertex3f(-1.1,0.05,-19);glVertex3f(-0.9,0.05,-19);glVertex3f(-
1,0.5,-19);
      glVertex3f(0.9,0.05,-21);glVertex3f(1.1,0.05,-
21);glVertex3f(1.0,0.5,-21);
      glVertex3f(-1.1,0.05,-23);glVertex3f(-0.9,0.05,-23);glVertex3f(-
1,0.5,-23);
      glVertex3f(0.9,0.05,-25);glVertex3f(1.1,0.05,-
25);glVertex3f(1.0,0.5,-25);
      glVertex3f(-1.1,0.05,-27);glVertex3f(-0.9,0.05,-27);glVertex3f(-
1,0.5,-27);
      glVertex3f(0.9,0.05,-29);glVertex3f(1.1,0.05,-
29);glVertex3f(1.0,0.5,-29);
      glVertex3f(-1.1,0.05,-31);glVertex3f(-0.9,0.05,-31);glVertex3f(-
1,0.5,-31);
      glVertex3f(0.9,0.05,-33);glVertex3f(1.1,0.05,-
33);glVertex3f(1.0,0.5,-33);
      glVertex3f(-1.1,0.05,-35);glVertex3f(-0.9,0.05,-35);glVertex3f(-
1,0.5,-35);
      glVertex3f(0.9,0.05,-37);glVertex3f(1.1,0.05,-
37);glVertex3f(1.0,0.5,-37);
      glVertex3f(-1.1,0.05,-39);glVertex3f(-0.9,0.05,-39);glVertex3f(-
1,0.5,-39);
      glVertex3f(0.9,0.05,-41);glVertex3f(1.1,0.05,-
41);glVertex3f(1.0,0.5,-41);
      glVertex3f(-1.1,0.05,-43);glVertex3f(-0.9,0.05,-43);glVertex3f(-
1,0.5,-43);
      glVertex3f(0.9,0.05,-45);glVertex3f(1.1,0.05,-
45);glVertex3f(1.0,0.5,-45);
```

```
      glVertex3f(-1.1,0.05,-47);glVertex3f(-0.9,0.05,-47);glVertex3f(-
1,0.5,-47);
      glVertex3f(0.9,0.05,-49);glVertex3f(1.1,0.05,-
49);glVertex3f(1.0,0.5,-49);
      glVertex3f(-1.1,0.05,-51);glVertex3f(-0.9,0.05,-51);glVertex3f(-
1,0.5,-51);
      glVertex3f(0.9,0.05,-53);glVertex3f(1.1,0.05,-
53);glVertex3f(1.0,0.5,-53);
      glVertex3f(-1.1,0.05,-55);glVertex3f(-0.9,0.05,-55);glVertex3f(-
1,0.5,-55);
      glVertex3f(0.9,0.05,-57);glVertex3f(1.1,0.05,-
57);glVertex3f(1.0,0.5,-57);
      glVertex3f(-1.1,0.05,-59);glVertex3f(-0.9,0.05,-59);glVertex3f(-
1,0.5,-59);
      glVertex3f(0.9,0.05,-61);glVertex3f(1.1,0.05,-
61);glVertex3f(1.0,0.5,-61);
      glVertex3f(-1.1,0.05,-63);glVertex3f(-0.9,0.05,-63);glVertex3f(-
1,0.5,-63);
      glVertex3f(0.9,0.05,-65);glVertex3f(1.1,0.05,-
65);glVertex3f(1.0,0.5,-65);
      glEnd();

      //brown rectangles for the tree trunks
      glColor4d(0.39,0.069,0.019,alf);
      glBegin(GL_QUADS);
      glVertex3f(-1.05,0,1.0);glVertex3f(-0.95,0,1.0);glVertex3f(-
0.95,0.05,1.0);glVertex3f(-1.05,0.05,1.0);
      glVertex3f(0.95,0,-1.0);glVertex3f(1.05,0,-
1.0);glVertex3f(1.05,0.05,-1.0);glVertex3f(0.95,0.05,-1.0);
      glVertex3f(-1.05,0,-3);glVertex3f(-0.95,0,-3);glVertex3f(-
0.95,0.05,-3);glVertex3f(-1.05,0.05,-3);
      glVertex3f(0.95,0,-5);glVertex3f(1.05,0,-
5);glVertex3f(1.05,0.05,-5);glVertex3f(0.95,0.05,-5);
      glVertex3f(-1.05,0,-7);glVertex3f(-0.95,0,-7);glVertex3f(-
0.95,0.05,-7);glVertex3f(-1.05,0.05,-7);
      glVertex3f(0.95,0,-9);glVertex3f(1.05,0,-
9);glVertex3f(1.05,0.05,-9);glVertex3f(0.95,0.05,-9);
      glVertex3f(-1.05,0,-11);glVertex3f(-0.95,0,-11);glVertex3f(-
0.95,0.05,-11);glVertex3f(-1.05,0.05,-11);
      glVertex3f(0.95,0,-13);glVertex3f(1.05,0,-
13);glVertex3f(1.05,0.05,-13);glVertex3f(0.95,0.05,-13);
      glVertex3f(-1.05,0,-15);glVertex3f(-0.95,0,-15);glVertex3f(-
0.95,0.05,-15);glVertex3f(-1.05,0.05,-15);
      glVertex3f(0.95,0,-17);glVertex3f(1.05,0,-
17);glVertex3f(1.05,0.05,-17);glVertex3f(0.95,0.05,-17);
      glVertex3f(-1.05,0,-19);glVertex3f(-0.95,0,-19);glVertex3f(-
0.95,0.05,-19);glVertex3f(-1.05,0.05,-19);
      glVertex3f(0.95,0,-21);glVertex3f(1.05,0,-
21);glVertex3f(1.05,0.05,-21);glVertex3f(0.95,0.05,-21);
      glVertex3f(-1.05,0,-23);glVertex3f(-0.95,0,-23);glVertex3f(-
0.95,0.05,-23);glVertex3f(-1.05,0.05,-23);
      glVertex3f(0.95,0,-25);glVertex3f(1.05,0,-
25);glVertex3f(1.05,0.05,-25);glVertex3f(0.95,0.05,-25);
      glVertex3f(-1.05,0,-27);glVertex3f(-0.95,0,-27);glVertex3f(-
0.95,0.05,-27);glVertex3f(-1.05,0.05,-27);
      glVertex3f(0.95,0,-29);glVertex3f(1.05,0,-
29);glVertex3f(1.05,0.05,-29);glVertex3f(0.95,0.05,-29);
```

```
	glVertex3f(-1.05,0,-31);glVertex3f(-0.95,0,-31);glVertex3f(-
0.95,0.05,-31);glVertex3f(-1.05,0.05,-31);
	glVertex3f(0.95,0,-33);glVertex3f(1.05,0,-
33);glVertex3f(1.05,0.05,-33);glVertex3f(0.95,0.05,-33);
	glVertex3f(-1.05,0,-35);glVertex3f(-0.95,0,-35);glVertex3f(-
0.95,0.05,-35);glVertex3f(-1.05,0.05,-35);
	glVertex3f(0.95,0,-37);glVertex3f(1.05,0,-
37);glVertex3f(1.05,0.05,-37);glVertex3f(0.95,0.05,-37);
	glVertex3f(-1.05,0,-39);glVertex3f(-0.95,0,-39);glVertex3f(-
0.95,0.05,-39);glVertex3f(-1.05,0.05,-39);
	glVertex3f(0.95,0,-41);glVertex3f(1.05,0,-
41);glVertex3f(1.05,0.05,-41);glVertex3f(0.95,0.05,-41);
	glVertex3f(-1.05,0,-43);glVertex3f(-0.95,0,-43);glVertex3f(-
0.95,0.05,-43);glVertex3f(-1.05,0.05,-43);
	glVertex3f(0.95,0,-45);glVertex3f(1.05,0,-
45);glVertex3f(1.05,0.05,-45);glVertex3f(0.95,0.05,-45);
	glVertex3f(-1.05,0,-47);glVertex3f(-0.95,0,-47);glVertex3f(-
0.95,0.05,-47);glVertex3f(-1.05,0.05,-47);
	glVertex3f(0.95,0,-49);glVertex3f(1.05,0,-
49);glVertex3f(1.05,0.05,-49);glVertex3f(0.95,0.05,-49);
	glVertex3f(-1.05,0,-51);glVertex3f(-0.95,0,-51);glVertex3f(-
0.95,0.05,-51);glVertex3f(-1.05,0.05,-51);
	glVertex3f(0.95,0,-53);glVertex3f(1.05,0,-
53);glVertex3f(1.05,0.05,-53);glVertex3f(0.95,0.05,-53);
	glVertex3f(-1.05,0,-55);glVertex3f(-0.95,0,-55);glVertex3f(-
0.95,0.05,-55);glVertex3f(-1.05,0.05,-55);
	glVertex3f(0.95,0,-57);glVertex3f(1.05,0,-
57);glVertex3f(1.05,0.05,-57);glVertex3f(0.95,0.05,-57);
	glVertex3f(-1.05,0,-59);glVertex3f(-0.95,0,-59);glVertex3f(-
0.95,0.05,-59);glVertex3f(-1.05,0.05,-59);
	glVertex3f(0.95,0,-61);glVertex3f(1.05,0,-
61);glVertex3f(1.05,0.05,-61);glVertex3f(0.95,0.05,-61);
	glVertex3f(-1.05,0,-63);glVertex3f(-0.95,0,-63);glVertex3f(-
0.95,0.05,-63);glVertex3f(-1.05,0.05,-63);
	glVertex3f(0.95,0,-65);glVertex3f(1.05,0,-
65);glVertex3f(1.05,0.05,-65);glVertex3f(0.95,0.05,-65);
	glEnd();

	//green lawn
	glColor4d(0.124,0.252,0,alf);
	glBegin(GL_QUADS);
	glVertex3f(-6,0,15);glVertex3f(-1,0,15);glVertex3f(-1,0,-
67);glVertex3f(-6,0,-67);
	glVertex3f(1,0,15);glVertex3f(6,0,15);glVertex3f(6,0,-
67);glVertex3f(1,0,-67);
	glEnd();

	//sides in chasm and vertical stair faces
	glColor4d(0,0,0,alf);
	glBegin(GL_QUADS);
	glVertex3f(-1,0,-35);glVertex3f(-1,-0.2,-35);glVertex3f(-1,-0.2,-
47);glVertex3f(-1,0,-47);
	glVertex3f(1,0,-47);glVertex3f(1,-0.2,-47);glVertex3f(1,-0.2,-
35);glVertex3f(1,0,-35);
	glVertex3f(-1,0,-9.999);glVertex3f(1,0,-
9.999);glVertex3f(1,0.25,-9.999);glVertex3f(-1,0.25,-9.999);
	glVertex3f(-1,0.25,-10.499);glVertex3f(1,0.25,-
10.499);glVertex3f(1,0.5,-10.499);glVertex3f(-1,0.5,-10.499);
      glVertex3f(-1,0.75,-10.999);glVertex3f(-1,0.5,-
10.999);glVertex3f(1,0.5,-10.999);glVertex3f(1,0.75,-10.999);
      glEnd();
}
/*******************************************************************
*********
The main routine for rendering scene haptics.
Renders the plane haptically.
draw object haptics for the shape defined in draw object graphics
********************************************************************
********/
void drawObjectHaptics(HLuint shapeId)
{
      // Clear the depth buffer when using a depth buffer shape
      glClear(GL_DEPTH_BUFFER_BIT);

      // start haptic frame - must do this before rendering any haptic
shapes
      hlBeginFrame();

      glPushMatrix();
      glTranslated(0,0,zTranslation);
      hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, shapeId);

      //assigning haptic properties to the shape
      hlMaterialf(HL_FRONT_AND_BACK, HL_STIFFNESS, 0.4);
      hlMaterialf(HL_FRONT_AND_BACK, HL_STATIC_FRICTION, 0.25);
      hlMaterialf(HL_FRONT_AND_BACK, HL_DYNAMIC_FRICTION, 0.15);

      ObjectGraphics();
      //haptic rendering of the lawn
      glBegin(GL_QUADS);
      glVertex3f(-6,0,15);glVertex3f(-1,0,15);glVertex3f(-1,0,-
67);glVertex3f(-6,0,-67);
      glVertex3f(1,0,15);glVertex3f(6,0,15);glVertex3f(6,0,-
67);glVertex3f(1,0,-67);
      glEnd();

      glPopMatrix();

      hlEndShape();

      hlEndFrame();
}
/*******************************************************************
*********
Draw a 3D cursor for the haptic device using the current local
transform,
the workspace to world transform and the screen coordinate scale.
********************************************************************
********/
void drawCursor(HLfloat angle)
{
      static const double kCursorRadius = 1;
      static const double kCursorHeight = 2;
      static const int kCursorTess = 15;
```

```
	HLdouble proxyxform[16];

	GLUquadricObj *qobj = 0;


	glPushAttrib(GL_CURRENT_BIT | GL_ENABLE_BIT | GL_LIGHTING_BIT);
	glPushMatrix();

	if (!gCursorDisplayList)
	{
		gCursorDisplayList = glGenLists(1);
		glNewList(gCursorDisplayList, GL_COMPILE);
		glTranslated(0,54,30);
		Box02();
		gluDeleteQuadric(qobj);

		glEndList();
	}

	/* Get the proxy transform in world coordinates */
	hlGetDoublev(HL_PROXY_TRANSFORM, proxyxform);
	glMultMatrixd(proxyxform);

	/* Apply the local cursor scale factor. */
	glScaled(gCursorScale, gCursorScale, gCursorScale);

	glEnable(GL_COLOR_MATERIAL);
	glColor4d(1, 0.75, 0.5,alf);
	glRotatef(angle,0,1,0);
	glCallList(gCursorDisplayList);
	glPopMatrix();
	glPopAttrib();
}
/*******************************************************
Ambient viscous damping.
Sets the current haptic rendering context and generates an effect ID
for the instance then starts the damping force
*******************************************************/
void drawSceneDamping(HHLRC &hHLRC,HLuint &effectId)
{
	hlMakeCurrent(hHLRC);
	effectId = hlGenEffects(1);
	hlBeginFrame();
	hlEffectd(HL_EFFECT_PROPERTY_GAIN, 0.75);
	hlEffectd(HL_EFFECT_PROPERTY_MAGNITUDE, 1);
	hlStartEffect(HL_EFFECT_VISCOUS, effectId);
	hlEndFrame();
}
/****************************************************************
the routine for moving the object.
****************************************************************/
void MoveIt()
{
	hlMakeCurrent(hHLRC1);
	hlGetDoublev(HL_PROXY_POSITION, device1ProxyPosition);
	hlGetBooleanv(HL_PROXY_IS_TOUCHING, &device1Contact);
	hlGetDoublev(HL_DEVICE_FORCE, device1Force);
```

```
      hlMakeCurrent(hHLRC2);
      hlGetDoublev(HL_PROXY_POSITION, device2ProxyPosition);
      hlGetBooleanv(HL_PROXY_IS_TOUCHING, &device2Contact);
      hlGetDoublev(HL_DEVICE_FORCE, device2Force);
      ZProxy1 = device1ProxyPosition[2];
      ZProxy2 = device2ProxyPosition[2];
      if (device1Contact == 1 && device2Contact == 0)
      {
            zTranslation = zTranslation + (ZProxy1 - lastProxy1);

      }
      if (device2Contact == 1 && device1Contact == 0)
      {
            zTranslation = zTranslation + (ZProxy2 - lastProxy2);

      }
      if (zTranslation >= 65)
      {
            zTranslation = 5;
      }
      lastProxy1 = ZProxy1;
      lastProxy2 = ZProxy2;
}

// this is the section of the code that takes the data for position
// and forces which were collected in the MoveIt function
//and writes it to a file for offline analysis
//the format of the file is a thirteen column data sheet
//time pos1X pos1Y pos1Z F1X F1Y F1Z pos2X pos2Y pos2Z F2X F2Y F2Z
void DataCollection()
{
      long int Time;

      Time=GetTickCount() - initial_time;
      fprintf(pFile, "%i %g %g %g %g %g %g %g %g %g %g %g %g\n",
            /*time is in ms units of force are newtons units of
position are mm*/
            Time,
            device1ProxyPosition[0],
            device1ProxyPosition[1],
            device1ProxyPosition[2],
            device1Force[0],
            device1Force[1],
            device1Force[2],
            device2ProxyPosition[0],
            device2ProxyPosition[1],
            device2ProxyPosition[2],
            device2Force[0],
            device2Force[1],
            device2Force[2]);
}
```

APPENDIX B

MATLAB CODE FOR ANALYSIS

The following is the Matlab code used to analyze the position and force data collected from the haptic devices during game play.

```
%Haptic Walking Data Analyzer
Time=Data(:,1); %in milliseconds
FirstXpos=Data(:,2)*25.4; %converted to mm
FirstYpos=Data(:,3)*25.4;
FirstZpos=Data(:,4)*25.4;
FirstXforce=Data(:,5); % in Newtons
FirstYforce=Data(:,6);
FirstZforce=Data(:,7);
SecondXpos=Data(:,8)*25.4;
SecondYpos=Data(:,9)*25.4;
SecondZpos=Data(:,10)*25.4;
SecondXforce=Data(:,11);
SecondYforce=Data(:,12);
SecondZforce=Data(:,13);
doublestance=Data(:,14);
%apply an interpolation and filter
desiredrate=100; %desired sample rate in hz
TI=(Time(1):1000/desiredrate:Time(end))';
Y1I=interp1(Time,FirstYpos,TI,'cubic');
Y2I=interp1(Time,SecondYpos,TI,'cubic');
Z1I=interp1(Time,FirstZpos,TI,'cubic');
Z2I=interp1(Time,SecondZpos,TI,'cubic');
cutoff=3;
[b,a]=butter(1,(2*cutoff)/desiredrate);
Y1IF=filtfilt(b,a,Y1I);
Y2IF=filtfilt(b,a,Y2I);
Z1IF=filtfilt(b,a,Z1I);
Z2IF=filtfilt(b,a,Z2I);
%calculate velocities
[VY1,VT]=CDF(Y1IF,TI);
[VY2,VT]=CDF(Y2IF,TI);
[VZ1,VT]=CDF(Z1IF,TI);
[VZ2,VT]=CDF(Z2IF,TI);
%calculate accelerations
[AY1,AT]=CDF(VY1,VT);
[AY2,AT]=CDF(VY2,VT);
[AZ1,AT]=CDF(VZ1,VT);
```

```
[AZ2,AT]=CDF(VZ2,VT);
%all accelerations are in m/s/ms so multiply by 1000 to
make m/s/s
AY1=1000*AY1;
AY2=1000*AY2;
AZ1=1000*AZ1;
AZ2=1000*AZ2;

%determine stride lengths
%from unfiltered data

%plot vertical positions
plot(FirstYpos);
hold on
plot(SecondYpos,'r');
plot(doublestance,'k--');
%make the selection of the data
point1=input('start:');
point2=input('finish:');
%assign new variables to the appropriate selection
newTime=Time(point1:point2);
Y1=FirstYpos(point1:point2,:);
Z1=FirstZpos(point1:point2,:);
FY1=FirstYforce(point1:point2,:);
Y2=SecondYpos(point1:point2,:);
Z2=SecondZpos(point1:point2,:);
FY2=SecondYforce(point1:point2,:);
%determination of step beginnings
n=2;
Y1Starts=[];
while n<=length(FY1)-1
    if FY1(n)<=0 && FY1(n-1)>=0 && FY1(n+1)<=0;
        Y1Starts=[Y1Starts; n];
    end
    n=n+1;
end

n=2;
Y2Starts=[];
while n<=length(FY2)-1
    if FY2(n)<=0 && FY2(n-1)>=0 && FY2(n+1)<=0;
        Y2Starts=[Y2Starts; n];
    end
    n=n+1;
end
%determination of step ends
Y1Ends=[];
```

```
n=3;
while n<=length(Y1)-2
    if Y1(n-1)<Y1(n-2)...
            && Y1(n)<Y1(n-1)...
            && Y1(n)<=Y1(n+1)+0.001...
            && Y1(n+1)<=Y1(n+2)+0.001;
        Y1Ends=[Y1Ends; n];
    end
    n=n+1;
end

Y2Ends=[];
n=3;
while n<=length(Y2)-2
    if Y2(n-1)<Y2(n-2)...
            && Y2(n)<Y2(n-1)...
            && Y2(n)<=Y2(n+1)+0.001...
            && Y2(n+1)<=Y2(n+2)+0.001;
        Y2Ends=[Y2Ends; n];
    end
    n=n+1;
end
%pairing beginings and ends
StartsAndEndsLeft=[1 1];
j=1;
k=1;
l=1;
onset=1;
offset=1;
while l<=length(Y1Starts)-1 && l<=length(Y1Ends)-1
   j=1;
    while onset==StartsAndEndsLeft(l,1)...
            && j<=length(Y1Starts)...
            && k<=length(Y1Ends)
        if Y1Starts(j)>StartsAndEndsLeft(l,2)...
                &&
Y1(Y1Starts(j))>=(Y1(StartsAndEndsLeft(l,2))-0.25)...
                &&
Y1(Y1Starts(j))<=(Y1(StartsAndEndsLeft(l,2))+0.25)
            onset=Y1Starts(j);
        else
            j=j+1;
        end
    end
    k=1;
    while offset==StartsAndEndsLeft(l,2)...
            && j<=length(Y1Starts)...
            && k<=length(Y1Ends)
        if Y1Ends(k)>onset...
                && Y1(Y1Ends(k))<=(Y1(onset)+0.75)...
                && Y1(Y1Ends(k))>=(Y1(onset)-0.75)...
                && Y1Ends(k)<onset+75
            offset=Y1Ends(k);
        else
            k=k+1;
        end
        if Y1Ends(k)>=onset+75
            onset=Y1Starts(j+1);
        end
    end
    StartsAndEndsLeft=[StartsAndEndsLeft; onset offset];
    l=l+1;
end

StartsAndEndsRight=[1 1];
j=1;
k=1;
l=1;
onset=1;
offset=1;
while l<=length(Y2Starts)-1 && l<=length(Y2Ends)-1
   j=1;
    while onset==StartsAndEndsRight(l,1)...
            && j<=length(Y2Starts)...
            && k<=length(Y2Ends)
        if Y2Starts(j)>StartsAndEndsRight(l,2)...
                &&
Y2(Y2Starts(j))>=(Y2(StartsAndEndsRight(l,2))-0.25)...
                &&
Y2(Y2Starts(j))<=(Y2(StartsAndEndsRight(l,2))+0.25)
            onset=Y2Starts(j);
        else
            j=j+1;
        end
    end
    k=1;
    while offset==StartsAndEndsRight(l,2)...
            && j<=length(Y2Starts)...
            && k<=length(Y2Ends)
        if Y2Ends(k)>onset...
                && Y2(Y2Ends(k))<=(Y2(onset)+0.75)...
                && Y2(Y2Ends(k))>=(Y2(onset)-0.75)...
                && Y2Ends(k)<onset+75
            offset=Y2Ends(k);
        else
            k=k+1;
        end
        if Y2Ends(k)>=onset+75
            onset=Y2Starts(j+1);
        end
    end
    StartsAndEndsRight=[StartsAndEndsRight; onset offset];
    l=l+1;
end
%stride length calculations
FirstStrideLengths=-1*(Z1(StartsAndEndsLeft(:,2))-
Z1(StartsAndEndsLeft(:,1)));
SecondStrideLengths=-1*(Z2(StartsAndEndsRight(:,2))-
Z2(StartsAndEndsRight(:,1)));
%percent stride and swing calculations
LeftTimes=[newTime(StartsAndEndsLeft(:,1)),
newTime(StartsAndEndsLeft(:,2))];
LeftSwing=LeftTimes(:,2)-LeftTimes(:,1);
percentLeftSwing=(sum(LeftSwing(1:end-
1))/(LeftTimes(end,1)-LeftTimes(1,1)))*100;
percentLeftStance=100-percentLeftSwing;
RightTimes=[newTime(StartsAndEndsRight(:,1)),
newTime(StartsAndEndsRight(:,2))];
RightSwing=RightTimes(:,2)-RightTimes(:,1);
percentRightSwing=(sum(RightSwing(1:end-
1))/(RightTimes(end,1)-RightTimes(1,1)))*100;
percentRightStance=100-percentRightSwing;
%visualizations
clf
plot(Y1)
hold on
plot(Y1Starts,Y1(Y1Starts),'kx')
plot(Y1Ends,Y1(Y1Ends),'gx')
plot(StartsAndEndsLeft(:,1),Y1(StartsAndEndsLeft(:,1)),'ok'
)
plot(StartsAndEndsLeft(:,2),Y1(StartsAndEndsLeft(:,2)),'og'
)
figure()
plot(Y2,'r')
hold on
plot(Y2Starts,Y2(Y2Starts),'kx')
plot(Y2Ends,Y2(Y2Ends),'gx')
plot(StartsAndEndsRight(:,1),Y2(StartsAndEndsRight(:,1)),'o
k')
plot(StartsAndEndsRight(:,2),Y2(StartsAndEndsRight(:,2)),'o
g')
```

```
figure()
plot(Time,FirstYpos)
hold on
plot(TI,Y1IF,'c')
plot(Time,SecondYpos,'r')
plot(TI,Y2IF,'m')
figure()
subplot(3,2,1)
plot(TI,Y1IF)
subplot(3,2,2)
plot(TI,Y2IF,'r')
subplot(3,2,3)
plot(VT,VY1)
subplot(3,2,4)
plot(VT,VY2,'r')
subplot(3,3,5)
plot(AT,AY2)
subplot(3,2,3)
plot(VT,VY1)
subplot(3,2,4)
plot(VT,VY2,'r')
subplot(3,2,5)
plot(AT,AY1)
subplot(3,2,6)
plot(AT,AY2,'r')
```

APPENDIX C

MATLAB CENTRAL DIFFERENCE FUNCTION

The following is the central difference function that was written to perform calculations of derivatives so that velocity and acceleration could be obtained from position data. This function was implemented in the Matlab code in Appendix B.

```
%a central diference function for derivative calculations
function [diff, difftime] = CDF(undiff, undifftime)
X=length(undiff)-1;
N=2;
diff=zeros(length(undiff)-2,1);
while N<X
    diff(N)=(undiff(N+1)-undiff(N-1))/(undifftime(N+1)-undifftime(N-
1));
    N=N+1;
end
difftime=undifftime(2:end-1);
```

REFERENCES

[1] Crago, Patrick E. et al. "New control strategies for neuroprosthetic systems." Journal of Rehabilitation Research and Development. 33.2 (1996): 158-172.

[2] Ou, Junli et al. "Adaptive control of cyclic movements in a multi-segment system." ifess.org. 15 April 2010. < http://www.ifess.org/ifess01/oral5/ouJ.pdf>.

[3] Badj, T. et al. "Voluntary commands for FES assisted walking in incomplete SCI subjects." Medical and Biological Engineering and Computing. 33 (1995): 334-337.

[4] O'Malley Teeter, Jeanne et.al. Functional Electrical Stimulation Resource Guide for Persons with Spinal Cord Injury or Multiple Sclerosis. Cleveland: FES Information Center, 1995. 23 March. 2010. <http://www.thestim.org/FESG/FESRG.pdf>.

[5] "Spinal Cord 101." spinalinjury.net. 15 April 2010. <http://www.spinalinjury.net/html/_spinal_cord_101.html>.

[6] "Facts and Figures at a Glance 2009." nscisc.uab.edu. January 2010. 15 April 2010. <https://www.nscisc.uab.edu/public_content/facts_figures_2009.aspx>.

[7] "Facts and Figures about Spinal Cord Injury." spinalcord.org. 29 July 2007. 15 April 2010. <http://www.spinalcord.org/news.php?dep=17&page=94&list=1191>.

[8] "Common Questions about Spinal Cord Injury." spinalcord.org. 29 July 2007. 15 April 2010. <http://www.spinalcord.org/news.php?dep=17&page=94&list=1190>.

[9] Home page. 15 April 2010. <http://www.spinalinjury.net/index.html>.

[10] Vaughan, Christopher L. et al. Dynamics of Human Gait 2nd Edition. Cape Town: Kiboho Publishers, 1992.

[11] Badj, Tadej et al. "The use of a four-channel electrical stimulator as an ambulatory aid for paraplegic patients" Physical Therapy 63.7 (1983): 1116-1120.

[12] Marsolais, E.B. and Rudi Kobetic. "Functional electrical stimulation for walking in paraplegia" The Journal of Bone and Joint Surgery 69 (1987): 728-733.

[13] Shimada, Yoichi et al. "Hybrid functional electrical stimulation with medial linkage knee-ankle-foot orthoses in complete paraplegics" Tohoku Journal of Experimental Medicine 209 (2006): 117-123.

[14] Noesner, Mathew Stephen. An Investigation of Position and Force During Gait-Mimicking Finger Motions. M.S. thesis, Dept. of Biomedical Engineering. New Jersey Institute of Technology, Newark, NJ, May 2004.

[15] Kim, Ji-Sun et al. "Finger Walking In Place (FWIP): A Traveling Technique in Virtual Environments" Lecture Notes in Computer Science 5166 (2008): 58-69.

[16] Sensable Technologies Inc. OpenHaptics Toolkit version 2.0 Programmer's Guide. Woburn: Sensable Technologies Inc., 2005.

[17] Qiu, Qinyin et al. "The New Jersey Institute of Technology Robot-Assisted Virtual Rehabilitation (NJIT-RAVR) system for children with cerebral palsy: a feasibility study" Journal of NeuroEngineering and Rehabilitation 6.40 (2009).

[18] Zeni, JA Jr. et al. "Two simple methods for determining gait events during treadmill and overground walking using kinematic data" Gait Posture 27(4): 710-714.

[19] Kawamura, Kenji et al. "Gait analysis of slope walking: a study on step length, stride width, time factors, and deviation in the center of pressure." Acta Medica Okayama 45(3) article 8.

[20] Winter, David A. Biomechanics and Motor Control of Human Movement Third Edition. Hoboken: John Wiley and Sons, Inc., 2005.

[21] Kralj, Alojz and Tadej Bajd. Functional Electrical Stimulation: Standing and Walking After Spinal Cord Injury. Boca Raton: CRC Press, 1989.

[22] Xue, Zhaojun et al. "New gait recognition technique used in functional electrical stimulation system control" Proceedings of the 6th World Congress on Intelligent Control and Automation. June 21-23, Dalian, China: 9421-9424.

[23] Sensable Technologies Inc. OpenHaptics Toolkit version 2.0 API Reference. Woburn: Sensable Technologies Inc., 2005.

[24] Sensable Technologies Inc. PHANToM Premium User Guide. Woburn: Sensable Technologies Inc., 2004.

[25] Everaert, Dirk G. et al. "Does functional electrical stimulation for foot drop strengthen corticospinal connections?" Neurorehabilitation and Neural Repair 24.2 (2010): 168-177.

[26] Popovic, Dejan et al. "Optimal control of walking with functional electrical stimulation: a computer simulation study" IEEE Transactions on Rehabilitation Engineering 7.1 (1999): 69-80.

[27] Ming, Dong et al. “A gait stability investigation into FES-assited paraplegic walking based on the walker tipping index” Journal of Neural Engineering 6 (2009).

[28] Thrasher, T. Adam and Milos R. Popovic. “FES-assisted walking for rehabilitation of incomplete spinal cord injury.” ifess.org. 15 April 2010. <http://www.ifess.org/ifess03/Oral%20Session%205%20-%20FES%20for%20Gait%20and%20FES%20for%20Respiratory%20Control/T%20Adam%20Thrasher.pdf>.

[29] Kostov, Aleksandar et al. “Integrated control system for FES assisted locomotion after spinal cord injury.” 1995 IEEE-EMBC and CMBEC. Theme 5: Neuromuscular SystemsBiomechanics. 1147-1148.