Scholar Commons
USF University of South Florida

University of South Florida
Scholar Commons

Graduate Theses and Dissertations | Graduate School

2007

Assistive force feedback for path following in 3D space for upper limb rehabilitation applications

Ramya Swaminathan
University of South Florida

Follow this and additional works at: http://scholarcommons.usf.edu/etd

 Part of the American Studies Commons

Scholar Commons Citation
Swaminathan, Ramya, "Assistive force feedback for path following in 3D space for upper limb rehabilitation applications" (2007). *Graduate Theses and Dissertations.*
http://scholarcommons.usf.edu/etd/2381

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.

Assistive Force Feedback for Path Following in 3D Space for Upper Limb Rehabilitation Applications

by

Ramya Swaminathan

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Biomedical Engineering
Department of Chemical Engineering
College of Engineering
University of South Florida

Major Professor: Rajiv V. Dubey, Ph.D.
Craig Lusk, Ph.D.
Shuh Jing Benjamin Ying, Ph.D.
Kathryn J. De Laurentis, Ph.D.

Date of Approval:
November 8, 2007

Keywords: Spring Force, Haptic, Fitts' Task, Scaling, Repetitive Motion, Phantom

© Copyright 2008, Ramya Swaminathan

Acknowledgments

First of all, I take this opportunity to thank my advisor Dr. Rajiv Dubey for his valuable guidance and support in completing my thesis. I would like to specially thank my good friend and colleague, Eduardo J Veras for his constant help, guidance and encouragement during my entire Master's research tenure. I would also like to thank Dr. Kathryn J. De Laurentis and all my colleagues at the Robotics and Rehabilitation group for their help. I also thank the members of my committee Dr. Shuh-Jing Ying, Dr. Craig Lusk and Dr. Kathryn J. De Laurentis for their valuable comments to this research. My friends are a huge pillar of support to me and i would like to acknowledge their valuable help and encouragement. Last, but not the least, all this wouldn't have been possible without my family. I would like to thank them for their unconditional love, support and encouragement during all phases of my life.

Table of Contents

List of Tables

List of Figures

Assistive Force Feedback for Path Following in 3D Space for Upper Limb Rehabilitation Applications

Ramya Swaminathan

ABSTRACT

The primary objective of this research was the design of an easy to use C++ Graphical User Interface (GUI) which helps the user to choose the task that he/she wants to perform. This C++ application provides a platform intended for upper arm rehabilitation applications. The user can choose from different tasks such as:

- Assistive Function in 3D Space to Traverse a Linear Trajectory
- User Controlled Velocity Based Scaling
- Fitts' Task in X, Y, Z Directions

According to a study conducted by the scientific journal of the American Academy of Neurology, stroke patients aided by robotic rehabilitation devices gain significant improvement in movement [1]. They also indicate that both initial and long term recovery are greater for patients assisted by robots during rehabilitation. This research aims to provide a haptic interface C++ platform for clinicians and therapists to study human arm motion and also to provide assistance to the user. The user would get to choose and

perform repetitive tasks aimed at improving his/her muscle memory. About eight healthy volunteers were chosen to perform a set of preliminary experiments on this haptic integrated C++ platform. These experiments were performed to get an indication of the effectiveness of the assistance functions provided in this C++ application. The eight volunteers performed the Fitts' Task in X, Y and Z directions. The subjects were divided into two groups, where one of the groups was given training without assistance and the other was given training with assistance. The execution time for both the groups was compared and analyzed. The experiments performed were preliminary, however some trends were observed: the people who received training with assistive force feedback took less execution time compared to those who were given training without any assistance. The path following error was also analyzed. These preliminary tests were performed to demonstrate the haptic platform's use as a therapeutic assessment application, a rehabilitation tool and a data collection system for clinicians and researchers.

1. Introduction

1.1 Motivation

The concept of applying robotics to rehabilitation has come a long way. Earlier, robotics research emphasized robot motion control and then focus shifted to the force control of a robot and the dynamics of robot interaction with the object it manipulates [1]. This research aims at developing a user friendly platform that has multiple applications for a robotic rehabilitation tool. Robot assisted devices are increasingly being used in stroke rehabilitation. The robotic tools help in the study of functional adaptation after a stroke. The greatest impact of the application of robotics to rehabilitation is not just the devices themselves, but also the infrastructure supporting rehabilitation. The concept of machines guiding people who are partially abled is not new, but what we present here is a useful modification in upper arm rehabilitation to help perform Activities of Daily Living (ADL). The main advantage of robot assisted therapy is that, they allow semi-autonomous practice of therapeutic tasks. Now, when we apply haptic technology to robotics, we provide another dimension to upper arm rehabilitation. This project emphasizes the use of assistive technology using haptics, to help people identify the direction of arm movement which causes tremors and undergo rehabilitative

training accordingly. This project is user friendly and cost effective, and hence becomes suitable for providing a platform for rehabilitative training. The unique idea about this project is that, it combines different assistance functions in one platform from which the user can choose the task that he or she wants to perform.

1.2 Problem Definition

There are a lot of robot assisted devices currently available to help people with arm disorder perform any of the simple ADL. This includes force assistance functions to help the user perform activities such as:

- Moving an Object from, Position A to Position B
- Grasping Objects

But there are not many options given to a user in terms of the types of assist functions or even in terms of the functionality of the interface device. The current arm rehabilitation tools for stroke, recoveries, etc, unfortunately do not provide a good quantitative measure of the recovery process. Secondly, standard interface devices like the keyboard or mouse often pose problems to motion impaired users [5]. These devices are not appropriate to meet the diverse needs of people with varying physical capabilities. Moreover, the devices may not perform consistently for extended computer usage. These standard input devices rely only on visual feedback which maybe supported by sound. The motion impaired users may not find visual feedback sufficient enough to perform teleoperation with their already reduced motor control and muscle strength. This would increase the error output in terms of position precision along with a huge delay in the time taken for completing the task. Studies have also shown that the fewer the number of

degrees of freedom of input devices, fewer the interactions rates. This emphasizes the importance of incorporating more degrees of freedom like finger flexion, to improve the reaction time [6]. This shows the significance of input devices for rehabilitation purposes.

To be more specific, there is no user friendly software to help identify and quantify the hand disability motion in any particular direction. The haptic device has been identified as a better and effective tool aimed at rehabilitation purposes. One such device called the Phantom is being widely used for the same. Moreover, the C++ application provides an immediate feedback on the quantitative measure of parameters which helps the clinician assess user performance during rehabilitation. The assist functions applied here also provide path assistance to facilitate the rehabilitation process. However, not many researchers have put the Phantom to its maximum use in terms of assistance based force feedback. The scaling of forces depending on factors such as the arm mobility of each individual or the type of teleoperation based task facilitates the user control and task performance. Even though there are a lot of haptic based rehabilitation application tools, not all have applied the concept of Multithreading to their applications. These haptic based rehabilitation tools will be discussed in the next chapter. The current scenario of haptic teleoperation with assistive techniques offers very few choices in terms of ease of operation, time of task execution, type of assistive functions, and extent of user control. That is the reason, why this area of research presents a high scope for advanced haptic teleoperation and rehabilitation.

1.3 Objectives

The objective is to create a haptic integrated PC based platform in C++ that unites different kinds of assistive functions intended for upper arm rehabilitation applications. Then, different types of assistive functions were studied and applied.

This application was designed to provide different types of virtual tasks for the user to perform. Also, the user can perform these tasks either with or without any assistance. Three virtual tasks in this application were designed and modeled based on a well known model of human psychomotor behavior called the Fitts' Task. It is very important to design a virtual task based on a mathematical model so that we can validate the accuracy of the task. Now, the human arm has seven degrees of freedom, three degrees of freedom in the shoulder, three in the wrist and one degree of freedom in the elbow. We try to replicate at least three translational movements, in other words, three degrees of freedom of the human arm based for a 3 degree of freedom Fitts' Task. Finally, the goal was to conduct experiments to validate the assistance concept and position accuracy among a sample population of eight subjects. The Fitts' Task implemented with force feedback aim at giving training to improve the speed-accuracy ratio of the arm disabled users apart from quantifying human dexterity [7]. The Fitts' Task test preliminary results later show a trend that the assist functions help reduce the task execution time and maintain the accuracy of user motion.

The objectives are summarized in points as shown below:

- To create a haptic integrated PC based platform in C++ that unites different kinds of assistive functions intended for upper arm rehabilitation applications.
- To study and implement some of the different types of assistive forces.
- To implement the well known model of human psychomotor behavior called the Fitts' Task in a virtual environment.
- To replicate three translational movements (3 DOF) of the human arm based with the Fitts' Task.
- To validate the assistance concept and position accuracy.

The GUI application in C++ also provides various test options to the user, in a very easy to use menu. This project intends to enable a robust haptic control with assistive technology primarily designed for motion impaired users. The various options presented in this GUI are as follows:

- Linear Constraint Motion 3D
- Velocity Scaling Assistance (User Controlled Scaling)
- Fitts' Task Implemented in X, with Assistance
- Fitts' Task Implemented in Y, with Assistance
- Fitts' Task Implemented in Z, with Assistance
- No Assistance Applied to any of the Above Tasks

The GUI is displayed in a later part of this thesis.

1.4 Thesis Outline

The current chapter is an introduction to robotics in rehabilitation, and a small outline of the research work. The related work or background in haptics and non haptics based upper arm rehabilitation, forms the crux of the second chapter. The third chapter presents a background on the different kinds of assistance concepts. The fourth chapter talks about the background of the Fitts' Law. The fifth chapter presents the experimental set-up with an introduction to the hardware, software part used in this research. The sixth chapter presents two virtual tasks that were designed along with a descriptive explanation of the concepts applied here. The seventh chapter describes the three Fitts' Tasks designed for the application and gives an introduction on the assistive concepts applied here. The eighth chapter presents the results and the conclusions. The Fitts' experiments were performed on a group of eight people. All the results with the graphs are provided in the eighth chapter. The last chapter talks about the future work that is integrated to this research.

2. Background

The Background work for this research was done in four different areas of interest such as:

- Haptics Based Therapy
- Non Haptics Based Therapy
- Assistance Concepts
- Fitts' Law

In this chapter, the background details of haptics based therapy and non haptics based therapy will be discussed.

2.1 Introduction

As far as rehabilitation is concerned, robotics has come a long way. Stroke rehabilitation is one of the main areas where robot assisted devices are extensively used. They help in the study of functional adaptation after a stroke. About 10% of the world population is affected with arm disabilities like stroke, tremors and disabled hand motor functions. Tremor is the most common cause of movement disorder. Tremors can be triggered by normal aging, drugs, Parkinson's disease, and Multiple Sclerosis or

excessive stress. The severity of the hand disability increases in the case of a spinal cord injury leading to neurological deficit conditions. In such a condition, a human would face difficulties in accomplishing simple ADL. Tremors are classified into Rest and Action types. Action tremors are further divided into Postural, Isometric and Kinetic. The tremor syndromes are further classified into Physiological, Essential, drug induced Parkinson's, Cerebellar and Pschycogenic. There exist a number of tremor syndrome diagnostic tools like Surface Electromyography, Accelerometers, Potentiometers and Handwriting Tremor Analysis but these are useable only in research or in some specialty centers. They require more technical knowledge to be operated upon and hence may not be user friendly. Even new technologies like Positron Emission Tomography and Single Photon Emission have limited applications and require more investigation. They can hardly be used on a daily basis. Here, we propose an efficient way to find the direction of motion in which the user faces difficulty in moving the arm. This research work aims at providing a platform for the clinicians and therapists to study arm motion and also to provide assistance to a user. This research helps to provide more precise, objective and detailed data on what actually happens during recovery. This in turn, would provide a better understanding of the key biomechanical and neurological factors required for successful rehabilitation.

2.2 Haptics Based Upper Arm Rehabilitation

The term haptic is derived from the Greek word haptesthai, meaning "to touch". The haptic sensory system employs both Cutaneous and Kinesthetic Receptors when

engaged in an active procedure. Basically, any kind of touch becomes “active”, when the sensory inputs are combined with controlled body motion. Haptic rendering is defined as the process of computing and generating forces in response to user interactions with virtual objects. They offer important applicability in engineering and medical training tasks. The past research [8] in haptic interface implemented several forms of assistance functions designed to augment human performance. The test bed used for these tasks consisted of a six degree of freedom force reflecting haptic interface device called the Phantom, with the GHOST SDK software produced by Sensable Tech. More recently, this company developed the Omni Phantom which is a more affordable haptic interface and it uses the OpenHaptics toolbox for programming. Pernalete [8] demonstrated that for a set of chosen tasks, the assistance functions significantly reduced execution times and enhanced the individual’s performance. Arsenault et al [9] implemented a haptic device interface to test eye-hand coordination during the manipulation of any 3D object in the virtual world. They proved that haptic rendering of virtual objects improved the eye-hand coordination for user interactions.

In teleoperation applications, a user is able to perform complex tasks in a remote environment. For example removal of bombs, mines and inspection of underwater structures require the intervention of a remote operator. The visual feedback plays an important role for these task executions and Morris [10] proved that the use of a haptic Interface with force feedback assistance increases the user’s perception of the virtual environment. Yu [12] developed a telemanipulation system using intelligent mapping from a haptic user interface to a remote manipulator. The mapping is referred to as an

assistance function, and was derived on the basis of user's motion intention. He applied the Hidden Markov Model for classifying the user's motion intention in a teleoperation task. In order to characterize the skill of the upper limb motion, Yu chose a simple task such as human movement along a virtual labyrinth. Although this is an excellent example, he implements the motion only in the X-Y plane, which limits the arm movement. We have developed a platform where the user can get haptic assistance in X, Y and Z directions.

One of the simplest of rehabilitation applications using haptics was the upper limb rehabilitation program using a basic motion training program [14]. The users were made to execute a simple task called the POINT task where time, viscosity and friction were measured when the user moved and pointed to 9 small circles displayed on the screen. The same task was performed with assistance. However, the entire task takes place only in the X and Y directions completely ignoring the Z axis.

Another interesting haptic exercise designed for post stroke arm rehabilitation is an application called the "The Labyrinth" which combines virtual environment with haptics. This [13] task was found to be very encouraging and motivating to the stroke patients. The users had to guide a stylus through a virtual maze using the Phantom device from one panel to the other without touching the walls as shown in Figure 2.1. Though this is very good exercise for the affected upper arm, the lack of a suitable model to quantify human motion performance might leave the clinician with very less information.

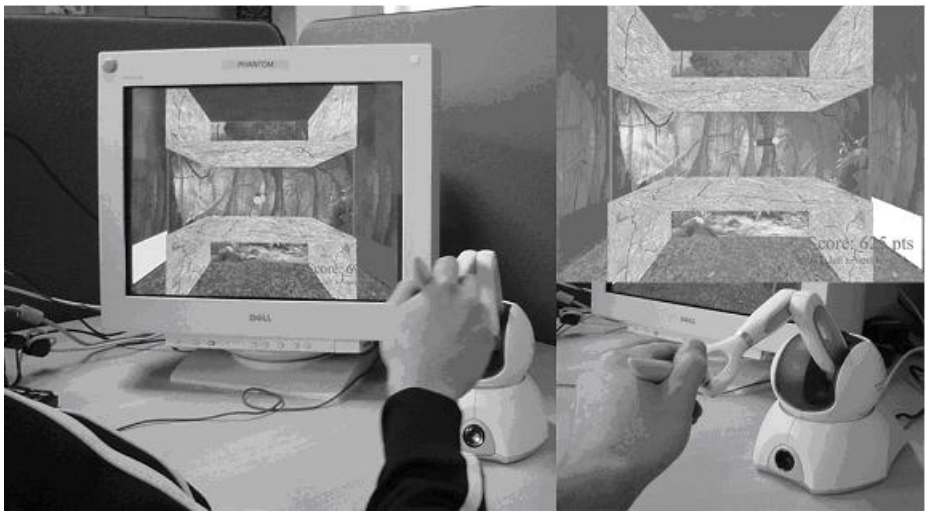


Figure 2.1: The Phantom Omni with “THE LABYRINTH”(Ref:[10])

The Rutgers Master II (RMII) glove [14] is another interesting rehabilitation device which provides resistive forces to the user wearing it. This glove, as shown in Figure 2.2, is integrated to a novel multipurpose haptic interface along with a tracker. This device helps the user perform a set of physical therapy and rehabilitation exercises.

Figure 2.2: The Prototype Developed at Rutgers, The State University of New Jersey, Rutger’s Glove (Ref:[14])

The MIT Manus [18] robot is a direct drive five bar linkage Selective Compliance Assembly Robot Arm (SCARA) mechanism which provides two translational degrees of freedom for the elbow and forearm motion. It also provides wrist rotation movements.

The Manus needs an occupational therapist to physically guide the patient initially in executing simple tasks. The Manus can be seen in Figure 2.3.

Figure 2.3: Commercial Version of the MIT Manus [Ref:Journal of Neuroengineering and Rehabilitation 2004]

Our project, also guides the patient in executing simple tasks like, approaching a trajectory path, following a trajectory path, Fitts' X, Fitts' Y, Fitts' Z and user controlled force scaling. Another significant difference between the Manus project and the rehabilitation tool presented here is that, there is no need of a therapist to guide the user to perform the above mentioned tasks. The Phantom device used as a haptic input device is very user friendly.

2.3 Non Haptics Based Upper Arm Rehabilitation

The development of rehabilitation robot manipulators began in the late 1960's itself. One of the first successful rehabilitation robot manipulators is the Rancho "Golden arm" developed in California [16]. One of the earliest examples of the workstation based approach to implement robotic systems was the Heidelberg Manipulator [16]. In line with the workstation based robotic device, the most commonly used system was the Robot for Assisting the Integration of the Disabled (RAID) [17]. This involved a set of preprogrammed tasks like moving books from one shelf to another using the robot controlled by a joystick. Another such robot used for occupational therapy is the Assisted Rehabilitation and Measurement (ARM) Guide device.

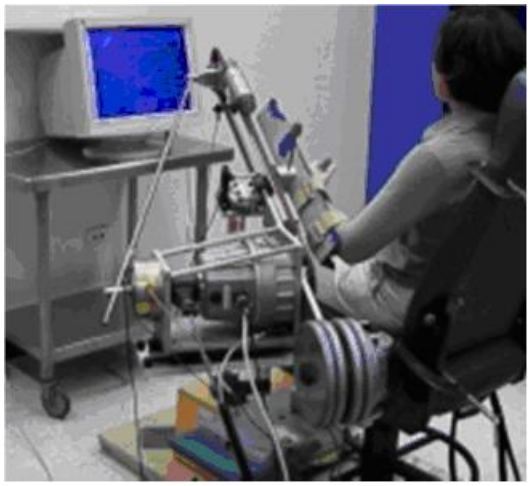


Figure 2.4: Photo of the ARM Guide (Ref:[11])

The ARM Guide [19] is a four degree of freedom robotic device which consists of a hand piece attached to an oriented linear track as seen in Figure 2.4. A force sensor can record the forces and torques at the interface between the device and the subject. This

device was primarily used to measure hand movements. Another interesting robotic device for arm rehabilitation is the Robotic Assisted Upper Extremity Repetitive Therapy (RUPERT) [20]. This arm is powered by four pneumatic muscles and has four actuated degrees of freedom. It can be adjusted to fit the patient body size. The arm is used to assist patients wearing them, to perform simple ADL and to move the arm in 3D space. The RUPERT arm can be seen in Figure 2.5.

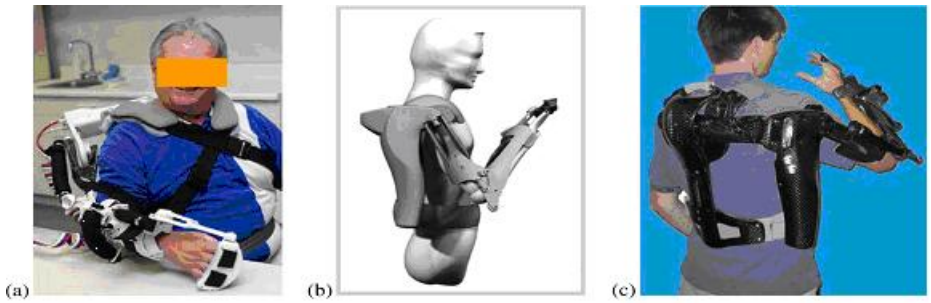


Figure 2.5: a)Version II of RUPERT Device b) Computer Simulation of Version III c) Version III of RUPERT Device (Ref:[12])

The Armin rehabilitation robotic arm designed for therapy [21] had a target population consisting of patients with orthopedic and neurological conditions. This is a six degree of freedom robotic arm designed like a human arm, fitted with position and force sensors. This arm did not provide any kind of force assistance to guide the user. So, the training sessions lasted longer. These robotic rehabilitation arms were either bulky or complicated to be operated upon. These are the major limitations of the established rehabilitation arms. That is why this research concentrates on the creation of a system that is simple and user friendly.The non haptic therapy devices do not provide a sense of touch to the user while performing tasks.

3. Assistance Concepts

3.1 Introduction

This chapter is an overview of the various types of forces that can be generated using the Phantom device and each of them is explained briefly here. In the later part of the research, the assistance concepts discussed here is eventually applied in the Fitts' experiments. The force feedback device used in this research is the three degree of freedom device called the phantom. The Assistance function concept is a way of assisting a user in task execution without overriding his or her command to the operator. The following chapter presents an introduction on the types of assistance concepts. In order to make our haptic tasks and teleoperation more accessible to people with arm disabilities, it was decided that assistance functions would be incorporated into the proposed simulations experiments.

The importance of assistance functions is emphasized in the following lines. In his study, J. Schuyler [22] concluded that even a slight increase in manipulation ability and strength would improve the job scope of the disabled individual. T. Kesavadas and Hari Subramanium [24] developed a system of virtual tools that one could use to interactively perform tasks in 3D with just click of a mouse on a computer screen. They performed experiments with tools with and without attributes. Here the ones with

attributes had pre-defined guide planes which would control the motion of the end-effector depending on the path. The task execution time was computed using Fitts' Task algorithm as it involved the tapping of blocks in 3D space. The Fitts' Task is used to model human movement based on speed and accuracy. The experiment was performed without any kind of assistance and the execution time was noted. They inferred from their results that the user performance improved when toll guided features were used. The importance of haptic assistance was further emphasized by K. Maclean [25] who determined that force reflection, sense of touch, made the haptic devices most suitable for applications in human augmentation, filtering and for manual activities performed by the differently abled. Thus, the assistance function concepts play a very important role in robotic tele-operation. The position and velocity mappings between the slave and manipulator are the basis of assistance function concepts [27]**.**

The assistance functions can be classified as:

- Force Assistance Function or Regulation of Contact Forces
- Regulation of Velocity
- Regulation of Position

3.2 Force Assistance Function/Force Feedback Design

This helps to augment the user's hand dexterity by imposing some constraints based on attractive or repulsive potential fields. The attractive or repulsive forces are implemented as virtual constraints in the master's workspace so that user cannot move in any undesired direction.

The force vector is a unit of output for a haptic device. The three different ways of simulating forces are motion dependent, time dependent and motion and time dependent.

3.2.1 Motion Dependent

This is computed based on the motion of the haptic device. The motion dependent forces are classified as follows:

3.2.1.1 Spring Force

The spring force is the simplest of force calculation techniques and is easily applied. The spring force is calculated based on the Hooke's Law which can be described as follows,

$$\vec{F} = k * \vec{x} \tag{3.1}$$

Where:

- $\vec{F}$ = Spring Force Vector
- k = Stiffness Constant
- $\vec{x}$ = Displacement Vector = $\vec{x}_0 - \vec{x}_1$
- $\vec{x}_0$ = Fixed Anchor Position
- $\vec{x}_1$ = Device Position

The virtual spring is attached between the fixed anchor position p0 and device position p1. The surface of the object that the user is touching usually forms the fixed anchor position.

The spring tries to restore itself to its original length from the displaced length and this mathematical displacement value is used to compute the *restoring force* of the spring

which is conveniently called as *spring force*. This force will always be directed towards the fixed anchor position. The stiffness constant k, determines how aggressively the spring would try to restore itself to its rest length. The behavior of this type of force is easy to predict and understand as it is based on the physical analogy of a spring. This force could be applied to either push two points in the virtual world from each other or just attract them towards each other which made this force suitable for our project as it could help the user to move towards any point on the trajectory or move away from it. Moreover, the data obtained from spring type force displayed aesthetic criteria like uniform edge length, uniform vertex distribution and symmetry.

3.2.1.2 Force Based on Exponential Law

This force is computed based on an exponential relationship between the haptic end-effector and the target object or trajectory. The relationship can be explained as,

$$F1 = k * e^{-a1Dist} \tag{3.2}$$

$$F2 = k * e^{-a2Dist} \tag{3.3}$$

$$F3 = k * e^{-a2Dist} \tag{3.4}$$

Where:

- $F1, F2, F3 =$ Force Components in X, Y, Z Direction
- $a1, a2, a3$ = Scaling Factors in the X, Y, Z Directions
- $k =$ Scaling Factor
- *Dist* = Distance Between Haptic End-Effector and Target Object

Hence, as the exponential force increases with decrease in distance and vice versa. The incorporation of a negative sign would change the direction of force vector which depends on the location of the end-effector and target on 3D screen. This force is considered to be an ideal force during trajectory approach because; the force is high only when the user is near the trajectory. This would prevent any sudden high force rendering in the beginning of trajectory approach itself. The user would not feel any jerks or unwanted pulling forces at the beginning itself. This slow increasing force would create a very smooth motion during trajectory or target approach.

3.2.1.3 Force Based on Projected Velocity

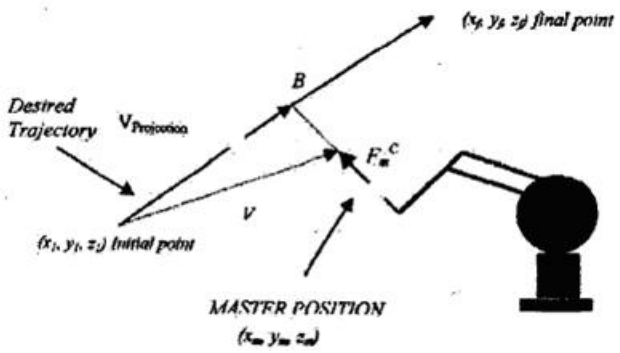


Figure 3.1: Representation of Force Assistance Function [13]

These forces are calculated based on the projection vector of the Cartesian position of the end-effector on the desired trajectory. They are important set of forces because they allow the user to stay on a perfect line of trajectory. The attractive force is computed as:

$$\vec{F} = k(\overline{V} - \vec{V}_{\text{Projection}}) \tag{3.5}$$

Where:

- k = Scaling Factor
- $\vec{V} = VelocityVector$ of the End-Effector
- $\vec{V}_{\text{Projection}} = VelocityVector$ Projected on to the Trajectory Vector.

This is then multiplied by a scaling factor depending upon the direction of motion. That is, the scaling value is higher for the component of the force (F) vector which is in the direction of required motion of master. The lesser scaling values applied in the other two direction constraint the force in those directions respectively. Thus, while the attractive force enhances the motion in one or two directions, the master will have to fight high torques on the constraint directions.

3.2.1.4 Force Assistance Based on Constant Force Projection

As indicated above, a force based on exponential law is exerted on the user to assist his or her motion towards the trajectory. Once the user is on the trajectory, a constant force is projected in the direction of the trajectory as follows:

$$\vec{F}_P = -(\vec{F}_D / \left|\vec{T}\right|^2) * \vec{T} \tag{3.6}$$

Where:

- $\vec{F}_P$ = Projected Force
- $\vec{F}_D = \vec{F} \bullet \vec{T}$, Dot product of a Constant Force Vector with the Trajectory Vector
- $\left|\vec{T}\right|^2$ = Square of the Magnitude of Trajectory Vector
- $\vec{T}$ = Vector along the Desired Trajectory Path

3.2.1.5 Force Assistance Based on Velocity Based Force Projection

The exponential based force is provided to the user to assist trajectory approach as in the above two cases. A force based on the haptic velocity of the approaching sphere is determined. The haptic velocity is obtained using a haptic command and projected in the direction of trajectory as shown in the next page.

$$\vec{F}_P = -(\vec{V}_D / \left|\vec{T}\right|^2) * \vec{T} \tag{3.7}$$

Where:

- $\vec{V}_D = \vec{V} * \vec{T}$, Dot Product of Haptic Velocity Vector with the Trajectory Vector
- $\left|\vec{T}\right|^2$ = Square of the Magnitude of Trajectory Vector
- $\vec{T}$ = Vector along the Desired Trajectory Path

This force projection assists the user in moving on the trajectory path.

3.2.1.6 Damper

The Damper, as the name suggests damps or opposes motion. The strength of Damper is proportional to end-effector velocity. The Damper equation is,

$$\vec{F} = -b * \vec{V} \tag{3.8}$$

Where:

- b = Damping Constant
- $\vec{V}$ = Velocity of End-Effector

This force always points in the opposite direction of motion. This force can be very useful when the user wants to control the scaling of forces sent to him during

teleoperation. Basically, user controlled scaling requires a factor which defines user motion to compute forces for every frame per second. The velocity of the user is the best factor which can determine user intention and scale up or scale down the forces accordingly. If the user velocity is low, then the Damper force applied is also low, based on the equation so that user controlled motion is not hindered. If the user velocity is high, then to prevent him or her from exceeding the velocity limits, a higher damping force is computed accordingly and sent to the user.

3.2.2 Friction

The haptic device can be used to simulate the following kinds of frictional forces: Columbic Friction, Viscous Friction, Static Friction, Dynamic Friction, and Inertia. The Frictional forces play an important role in rehabilitation based training tasks using haptic technology for people with hand disabilities. They can especially help haptic users with hand tremors to move in 3D space smoothly in any direction. Here is a description of the different types of frictional forces:

3.2.2.1 Columbic Friction

This basic kind of force opposes the direction of motion with a force that is computed by:

$$\vec{F} = -C \times sign(\vec{V}) \tag{3.9}$$

Where:

- $\vec{V}$ = Velocity of End-Effector
- C = Frictional Constant
- $sign(\vec{V})$ =Direction of Vector $\vec{V}$

This frictional force helps to create a smooth transition when changing directions, since friction will be proportional to velocity for slow movement. Here the damping constant is high, with a small constant force clamp.

3.2.2.2 Viscous Friction

This force is very similar to the Columbic force. This is also computed with damping constant and constant force clamp. The only difference here is that, the damping constant is low with a high clamp value. The Static and Dynamic Friction is also referred to as the stick-slip friction as the friction model switches between no relative motion and resisted relative motion. This force opposes lateral motion along a surface, and the magnitude of the frictional force is always proportional to the perpendicular (normal) force of contact.

3.2.3 Inertia

This force is associated with any moving mass. In a given trajectory, one can calculate the force during that motion using Newton's second law of motion:

$$\vec{F} = m * \vec{a} \tag{3.10}$$

Where:

- $\vec{F}$ = Force in Newton
- m= Mass
- $\vec{a}$ = Acceleration

3.2.4 Time Dependent

Any force that is computed as a function of time is called Time Dependent. The following are the different ways of rendering Time Dependent forces:

3.2.4.1 Constant Force

This is a force with fixed magnitude and direction. This force is used for gravity compensation so that the end-effector feels weightless. This force can also be used to make the end-effector feel heavier than normal.

3.2.4.2 Periodic Force

This force is produced by applying a pattern that repeats over time. The patterns vary from a Saw Tooth Wave, Sinusoidal Wave or Square Wave. The Periodic force is

defined by a time constant and amplitude. The time constant controls the period of the patterns cycle whereas the amplitude determines how strong the force will be at the peak of the cycle.

3.2.5 Impulses

These are forces which are applied instantaneously. This force is more effective because, nerves are more sensitive to discontinuities in force rather than steady-state force. This is why, a larger derivative of force with lower magnitude is more compelling than a smaller force derivative with higher magnitude. The impulse force can be very useful when it comes to computing average impact forces during collision. Given, the end-effector velocity of the Phantom, and the assumed mass of the end-effector, the impulse force could be calculated as follows:

$$\vec{F} = m * (\Delta \vec{V} / \Delta t) \tag{3.11}$$

Where:

- $m =$ Mass of the End-Effector
- $\Delta \vec{V} =$ Change in End-Effector Velocity
- $\Delta t =$ Change in Time

3.3 Velocity Assistance Function

The Velocity assistance function serves two purposes: assistance in approach to a target and the assistance in avoidance of an obstacle. The velocity scaling is varied

according to whether the motion in that particular direction is achieving the desired effect of the motion. The scaling is applied in two different ways such as:

- Constant Scaling Factors of 0 and 1**:** The scaling was performed only if it assisted the user in the desired direction of travel.
- Variable Scaling Factor**:** Here, the scaling factor is computed for every new position of the haptic end-effector and multiplies the velocity vector. According to [26], an estimate of the desired scaling can be obtained as follows:

The scaling can be applied in two different ways:

$$S_s \max = V_S(X) / V_m Max \tag{3.12}$$

Where:

- $S_s \max$ = Scaling Factor
- $V_S(X)$ = Desired Velocity of Approach
- $V_m Max$ = Maximum Expected Master Velocity

The velocity assistance with scaling is given by:

$$\vec{V}(t+1)_{SCALED} = K * \vec{V}(t) \tag{3.13}$$

Where:

- $\vec{V}_{SCALED} = ScaledVelocity$ vector
- $\vec{V} = VelocityVector$
- $K = ScaleFactor$

3.4 Position Assistance Function

The Position Assistance Function also known as Position scaling, involves the enlargement or reduction of slave workspace as compared to master workspace.

There are two types of Position Assistance Functions:

- Planar Assistance Function
- Linear Assistance Function

In the above cases, the motion constraint was designed to lie along a line or in a plane. This was designed for people with disabilities so that they could operate in a more stable manner. In Planar Assistance Function, the whole workspace is scaled up or down depending on the master workspace requirements. In Linear Assistance Function, just the linear path end-effector points are scaled up or down with respect to the master workspace.

The force assistance concepts used in this project are:

- Spring Type Force Feedback
- Exponential Type Force Feedback
- Constant Force Projection
- Velocity Scaling Based Assistance

These different concepts are explained in detail in a later chapter in this document. The concepts are also accompanied by detailed diagrams.

.

4. Implementation of Fitts' Task

The reason to choose the Fitts' Task as the rehabilitation task is because its one of the most commonly used mathematical model of human movement. This experiment is simple in nature but yet provides a large amount of information regarding the user performance. This is the important phase of the project where three Fitts' experiments were designed on the haptic interfaced C++ application platform. The Fitts' experiments were used as a validation tool for this system which also illustrates the use of the system for rehabilitation applications. This application is very versatile because the Fitts' is designed in 3 degrees of freedom, which means in all three directions of motion, X, Y and Z. Three Fitts' experiments were designed based on the direction of motion. Eight volunteers were asked to perform experiments based on these three Fitts' Tasks. The motive behind designing these Fitts' Tasks was to measure the user performance after training with and without assistance.

4.1 Introduction

The Fitts' Task was proposed by Paul M. Fitts' (1912-1965) who was a Psychologist at Ohio State University. This widely known human model of movement is based on rapid, aimed movement and is a well studied mathematical model of human

motion [7]. The Fitts' model is a formal relationship which describes speed/accuracy tradeoffs in aimed one dimensional translational movement in upper extremity tasks.

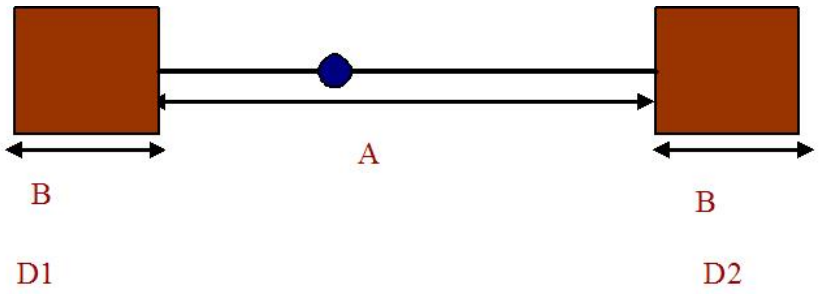


Figure 4.1: Descriptive Representation of Fitts' Law

Mathematically, Fitts' Task is expressed as follows:

$$T_{move} = \alpha + \beta \log_2(2A/B) \tag{4.1}$$

Where:

- T_{move} = Time for a Given Number of Traversals between the Two Goals
- α, β = Constants Fit to the Experimental Data
- A = Distance between the Two Goals
- B = Width of Each Goal

As seen in the above equation, the Fitts' model can predict the time required to rapidly move from a starting position to a final target area, as a function of the distance to the target and the size of the target.

Another important aspect of Fitts' law is the Index of difficulty (ID) which provides a measure of the difficulty of a motor task. The ID is given as follows:

$$\mathrm{ID} = \log_2(2A/B) \tag{4.2}$$

This law is applied to human movements involving pointing both in the real world and in simulation tasks. This relationship means that at increased difficulty, more accurate movements require longer aiming time [28]. Initially, proposed as a method of quantifying human dexterity [35], Fitts' Task has found wide applications since then, in the area of robotics and rehabilitation.

4.1.1 Existing Models of Predicting Human Movement

Now, let us see why the Fitts' Task was chosen among the existing mathematical models of human movement. The existing models of human movement are classified into Predictive models and Descriptive models. The Predictive models, also known as engineering models or performance models, determine metrics of human performance analytically. On the other hand, the Descriptive models help to provide a framework for describing a problem or a situation. The Predictive models are classified into Hick-Hymann Law, Key Stroke Level Model and Fitts' Law. The Descriptive models are Key-Action Model, Three State Model of Graphical Input, Model for Mapping Degrees of Freedom to Dimensions and Guiards Model of Bimanual Skill.

The effect of spatial uncertainty on motor planning was studied using a queing method in a reaching task [29]. The results showed that the relationship between average reaction time and number of cues was poorly described by the Hick-Hymann Law [29]. The Key Stroke Level Model [30] did not require any specialized psychological

knowledge. However, this model had lot of limitations. They could predict only one aspect of user computer interaction namely, time of interaction. Moreover, this model required a complete error free performance time on routine tasks and also a high user expertise. The Descriptive models do not provide any empirical or analytical data as mentioned above. However, they are used to understand any problem or situation better. With all these considerations, the significance of the Fitts' Task in human computer interactions is discussed next.

4.1.2 Significance of the Fitts' Task

The technological advancement in human-computer interfaces has evoked an interest in developing a reliable prediction model of movement time in computer input tasks [31]. In our study, the human interfaces like cursor keys and function keys are replaced by GUI. The Fitts' Task satisfies the need to measure and quantify human movement in virtual space. Previous experiments have shown that kinematic patterns of human movement change in a systematic manner. These kinematic patterns displayed a relationship between simple oscillatory motion and informational flow in task space during rhythmic pointing tasks [32].

This Task has found prominence in many experiments to validate speed, accuracy and time coordination involving movement time. Movement time is defined as the time taken to complete a task. Sommer Gentry, Eric Feron and Roderick Murray Smith [33] report that two person teams, also known as dyads can achieve lower movement time (MT) for cyclical, continuous aiming movements'. One of their experiments was based

on a cyclic task formulated with Fitts' model. In the cyclic task, the user had to aim successively at each pair of targets as rapidly as possible without overshooting or undershooting. This was as opposed to the discrete task where the person was asked to aim at and come to a stop within a given target. The haptically coupled cyclic Fitts' Task involved two people to control the motion of the pointer and target respectively. The experimental setup involved a standard computer driving wheel fixed to a desk, with a 4 foot long wooden dowel attached to create a lever. The screen displayed two targets as sectors of a circle and the current position as a pointer. The authors did two experiments, one with dyads and one with solo subjects.

While the solo subjects individually tried to move the pointer to the target, the dyads performed the cyclic task together with one person on either side of the rotating handle. Here, each person had his own target and pointer display. The data analysis led them to conclude that dyads performed significantly better at a minimum time cyclical aiming task than individuals. This because, the dyads could concentrate on their individual targets better.

Steven Edward Everett [34] performed some experiments like the surface impact and Fitts' Task to demonstrate the efficacy of his assistance algorithms in performing radioactive waste tank cleanup. Everett's test bed consisted of two dark rectangles representing the goals on a white table. He measured the transit time in moving from the left goal to the right and back again for different values of distance (D) between the goals and width (W) of the goal regions. Then, he experimentally determined the best fit theoretical lines for the time versus (D/W) ratio. This was done for both constant and

variably scaled velocity mappings. The findings of his research suggested that the assistance functions resulted in smaller task execution times as compared to the performance of tasks without assistance functions.

4.2 A Unique Implementation of Fitts' Task

The implemented Fitts' Task is designed to detect tremors in any direction in 3D space. Fitts' model describes a relationship developed by applying the information theory of physical communication systems to the sensory-motor system [7]. The reason why Fitts' Task was chosen to be a part of this project is two-fold. One, to validate the assistance functions which were developed previously and secondly, to provide parameters like Movement Time (MT), Index of Difficulty (ID), Index of Performance (IP) which provides very useful information to the human factors engineers and biomedical engineers [35]. The Human movement has been shown to display a tradeoff between speed and accuracy in target directed movements [36, 37]. This trade-off could be traced back to pschycomotor delay.

Sensory transduction, latencies in central processing and motor output could be the reason for motor circuit delays. This is due to a synaptic delay between two single neurons which may range between 1-2 ms. The conduction along an axon could also be the cause of the delay in motor circuit. The factors which determine axon conduction would be length of axon, and if the axon is myelinated or non-myelinated. This explains one of the possible reasons of time delay in task execution.

The Fitts' Law [35] has been shown to provide an indirect estimate of the delay within the motor circuit. They applied the natural relationship developed between the

underlying physiology and (α, β), the coefficients of Fitts' Law to observe human motor performance and also to provide an indirect estimate of delay within the motor circuit. Thus, researchers [35] have shown that the coefficients of Fitts' Law can be used to study human-motor behavior. This understanding of the human motor behavior through Fitts' Law could also lead to improved robotic aids for teleoperation and rehabilitation applications. Fitts' Law has many applications in Human Computer Interactions (HCI) like developing methods for target acquisition in virtual worlds, conceptual extension of Fitts' model with the help of multiscale pointing in zoomable surfaces [36]. With all these factors in mind, the Fitts' Task was incorporated in the GUI. In the user interface, the user could choose from any of the three Fitts' Tasks provided. These tasks were classified as Fitts' Task X, Fitts' Task Y, Fitts' Task Z, three dimensional force scaling and user controlled velocity based scaling technique. Each of these tasks were subdivided into two options. One option did not provide any kind of assistance to the user while the other one provided force assistance in performing the same task. All these options were provided in the C++ GUI.

The Fitts' Task can be classified into discrete and cyclical motion types. Given, two targets in the virtual space, when the user moves towards a target and comes to a stop within the same target, then the motion is called a discrete type motion. On the other hand, if the user has to move towards each of the pair of targets successively in a rapid motion, the task becomes a cyclical kind of motion.

Researchers [33] have proved that cyclic aiming reduces the Movement Time (MT) compared to the discrete aiming. This was due to the sinusoidal motion of the

cyclic Fitts' Task which permits storage and reuse of kinetic energy that a user has generated. Hence, a cyclic task puts the user at an advantage physically. G. P. Van Galen and J. Duysens [39] conducted a study to compare discrete versus cyclic movements for different target widths. The sample population used for the study consisted of 24 healthy participants. They found that the Index of Performance (IP) and movement velocity were almost twice as large in cyclic compared to discrete movements. The predicted Index of Performance (IP) constant was found not to hold for rapid cyclic movements. Their studies clearly indicated that cyclic movements exploit the energetic and physiological properties of the neuromotor system. This is the reason why cyclic Fitts' Task was implemented for the rehabilitation part of this project.

4.2.1 Fitts' Task X

In Fitts' Task X, the OpenGL scene consisted of two goal regions represented by two rectangular blocks drawn with a pre-defined distance between them. The linear trajectory connecting the two goal points was drawn in the X direction. The user could move the Phantom end-effector over the line from one goal point to another to first determine if he or she had any difficulty in that particular direction for hand motion. In other words, the user could detect motion abnormalities in the horizontal direction. In Fitts' X, the user moved in the X-Y plane while traveling on the trajectory. The flowchart in Figure 4.2 describes the assistive function applied in Fitts' Task X.

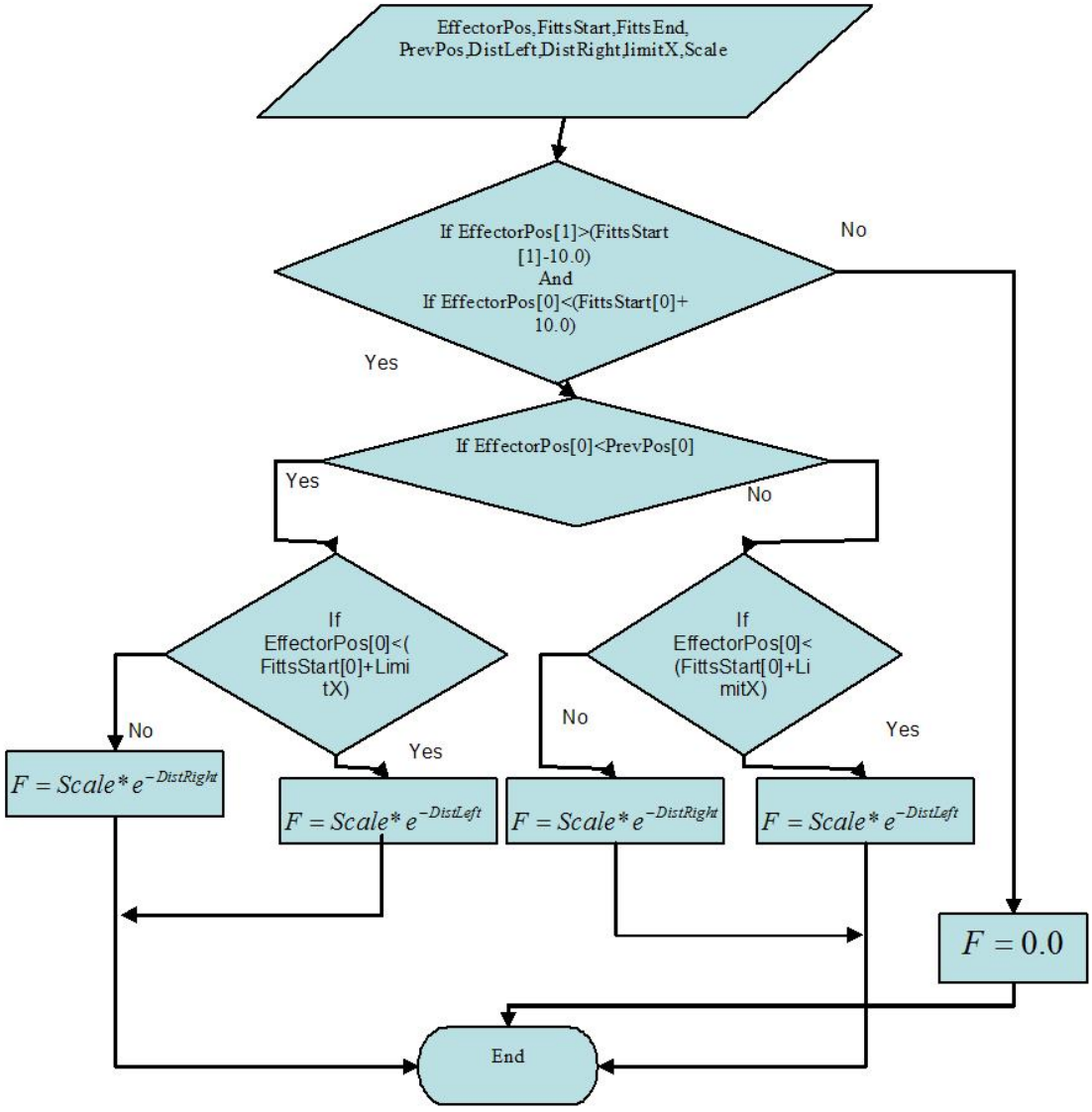


Figure 4.2: Flowchart Depicting Assistive Function in Fitts' Task X

In order to understand the flowchart, we should first understand the terminology applied here.

4.2.1.1 Terminology Used in Fitts' X

EffectorPos is a vector representing the current user position in a virtual space. Fitts'Start is a vector representing the point at the start of the trajectory. Fitts'End is a vector representing the point at the end of the trajectory. PrevPos is a vector representing the previous position of the user. DistRight is the distance between the EffectorPos and the Fitts'End when the user is on the right side of the centre of the trajectory. DistLeft is the distance between the EffectorPos and the Fitts'Start when the user is on the left side of the centre of the trajectory. LimitX is a point which defines the limits of the middle space on the trajectory where null force exists.

4.2.1.2 Explanation of the Flowchart in Fitts' X

The steps involved are described as follows:

- Step 1: First, the code checks if the effector X and Y coordinate lies in a range of 10 units on either side of the trajectory.
- Step 2: Then it is determined if the effector position is greater or lesser than the previous position. This determines the direction of motion of the user. The Phantom provides assistance in both directions of motion.
- Step 3: This code does not generate any force in a small region in the middle of the trajectory in Fitts' Task. That is why, the code calculates whether the user is beyond the limits of the middle region.

- Step 4: If yes, then an exponential force is provided to assist the user in the direction of motion which is totally controlled by the user.
- Step 5: If no, that is if the user is in the middle region, then no force is given so that the user can easily disengage from the task.

Any irregularity like tremors in the hand in this particular direction of motion qualified for assistance. Then the user could choose the assistance option from GUI. The user was given force assistance to move from the center of the trajectory to the left goal point and vice versa. Similarly, a spring type force was given when the user began to approach the right hand side goal point. This resulted in smooth acceleration during transit, lessening any deviations from the trajectory or any unwanted vibrations.

4.2.2 Fitts' Task Y

The Fitts' Task Y had a graphical scene similar to the previous one except for the fact that the trajectory connecting the two goal points was drawn in the direction of the Y axis. Here also, the movement was made in the X-Y plane. This experiment was designed to test the effectiveness of the human hand motion in the Y direction or up-down motion. People with hand motor dysfunctions in this direction would have difficulty in performing this Fitts' Task. The assistance option along with this task provided assistive force to the user to move towards the goal point in an upward motion and also to come back to the starting point from the goal. These forces were also made effective when the user wanted to travel to the goal point in the lower end of the trajectory in a smooth motion.

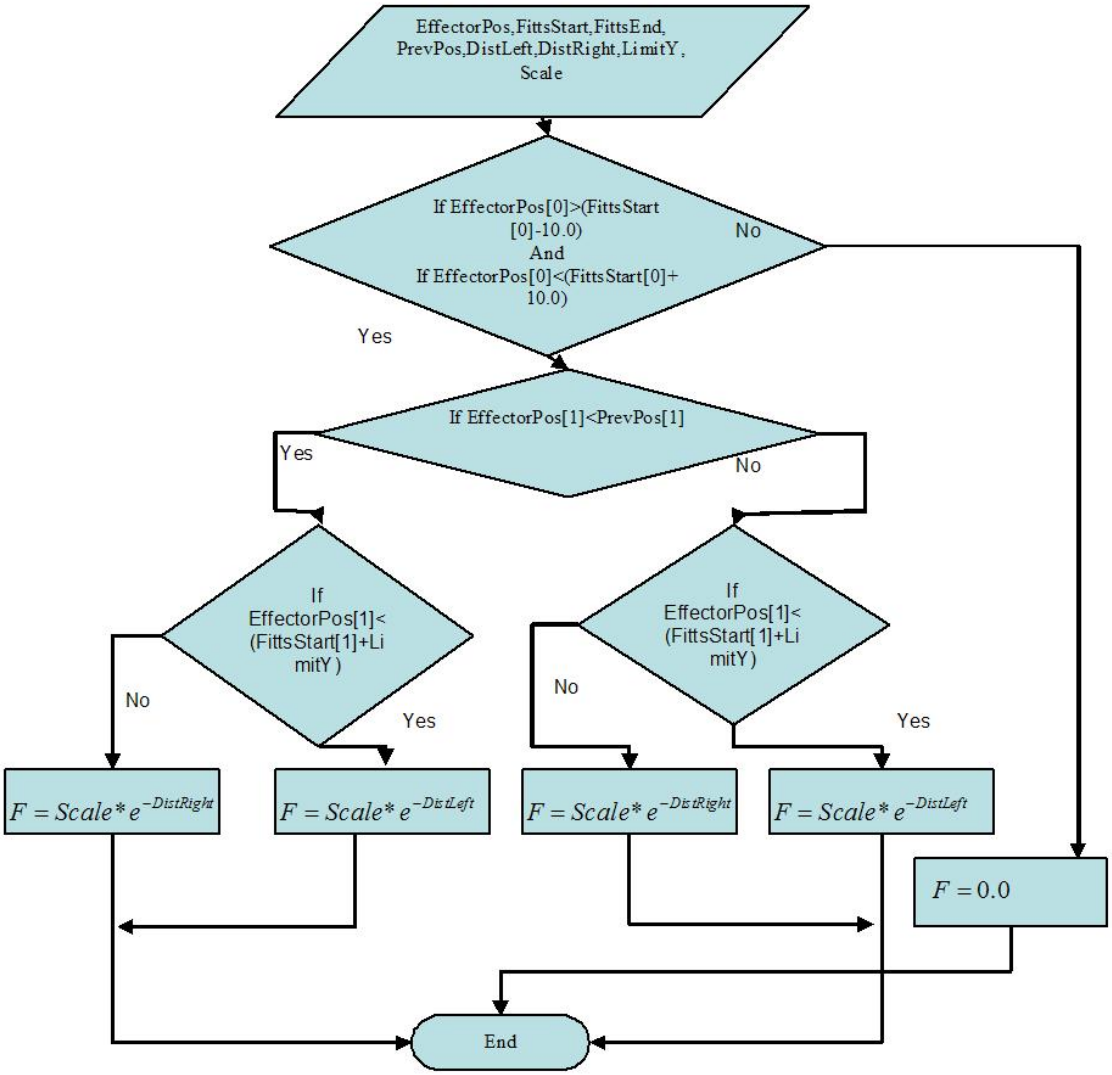


Figure 4.3: Flowchart Describing Assistance Function in Fitts' Task Y

4.2.3 Fitts' Task Z

The final one, Fitts' Task Z was implemented to help the user identify any kind of hand motion difficulties in the Z direction. This meant that the user could detect motion abnormalities for movement towards the screen. The trajectory vector was drawn in the Z direction. The assistance option for this task was also provided in the GUI. The user could overcome the hand motor difficulties in approaching a virtual object in space in the Z direction with the help of the assistive forces. Here the plane of motion is the Y-Z plane. The terminology for this task remains the same as the previous two experiments. The flowchart representing the sequence of events in Fitts' Y is shown in Figure 4.4.

Here, the algorithm differs from that of Fitts' X and Fitts' Y as three conditions are checked instead of just two. That is, the algorithm in Fitts' Z checks for the location of the end–effector with respect to the X, Y and Z directions.

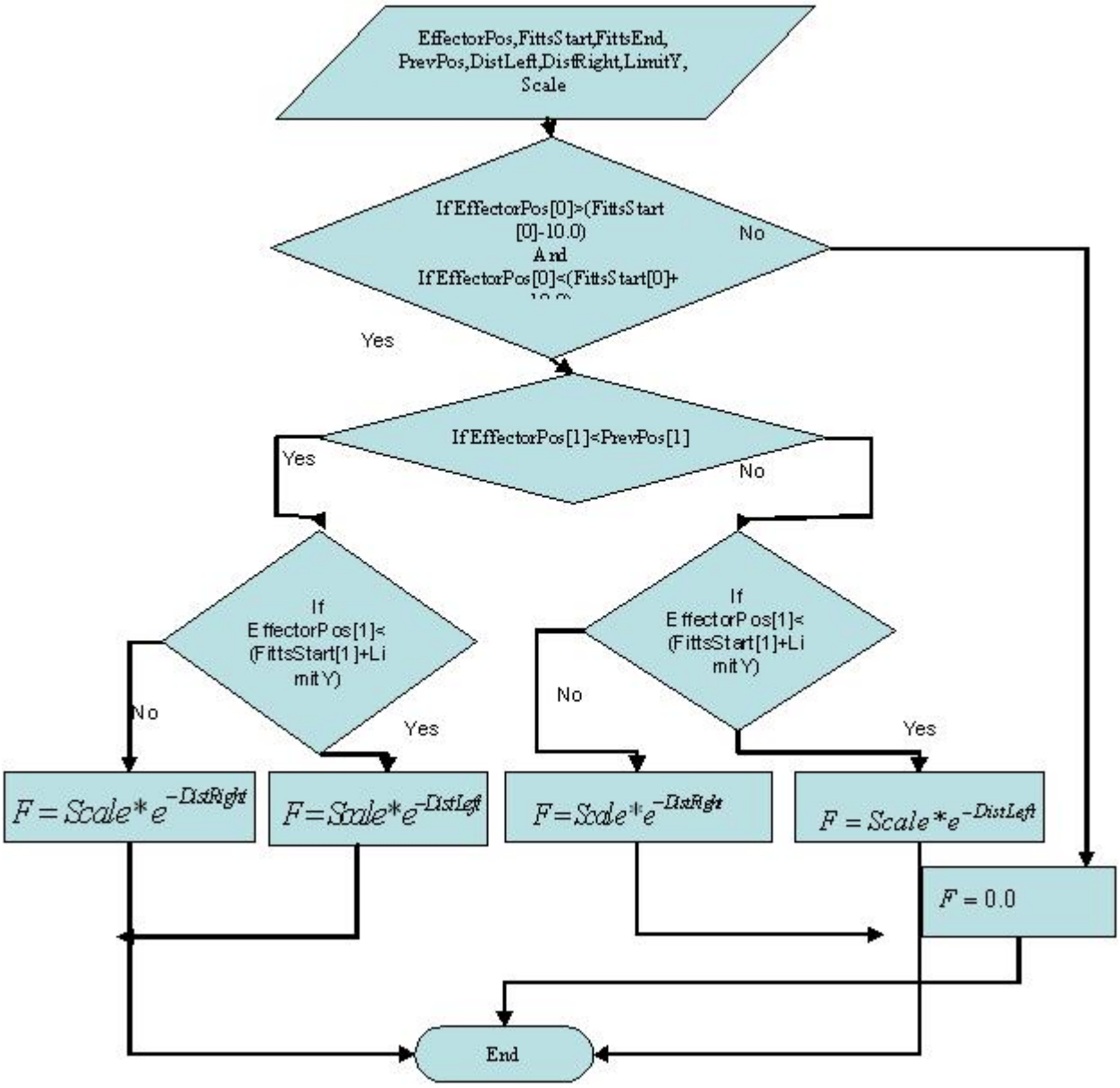


Figure 4.4: Flowchart Describing Assistance Function in Fitts' Task Z

5. Experimental Test Set-Up

5.1 Introduction

The experimental set up consists of a C++ based application integrated with a haptic device called the Phantom from Sensable Technology. The Phantom is an ideal rehabilitation tool for this research because it provides force feedback in X, Y and Z directions. This force can be used to our advantage to provide constraint or assistance. Thus, the Phantom plays a very important role in this project and it becomes essential to learn about its libraries and capabilities. The rehabilitation tasks are simulated using OpenGL graphical software tools. A well know mathematical model of human movement called the Fitts' Task was simulated in X, Y and Z directions separately. A stopwatch is used to measure the execution time of the eight subjects in completing all the Fitts' Tasks. A C++ code is written to store the haptic position data of every user systematically.

5.2 Hardware

5.2.1 Phantom 3D-Touch Enabled Modeling System

Earlier, haptic feedback was used mainly for flight simulator applications and Master-Slave tele-robotic application. These systems had mechanical linkages between the master and the slave. Then in 1954, Goertz and Thomson developed an electrical servomechanism that received feedback signals from sensors mounted on the slave. This servomechanism applied forces to the master. The founding stone for haptics tele-operations was hence made. In the period of 1967-1990, the GROPE project at University of North Carolina created another breakthrough in haptic research [30]. This project initiated force feedback from simulated interactions. Here, the slave robot was substituted by a simulated system, in which forces were computed using physically based simulations. Thus, the concept of force feedback was implemented from a very long time back. Engineers just needed to develop a better user friendly device to incorporate these force feedback concepts.

In 1980, Bejczy and Salisbury devised a computer based Cartesian control for tele-operator systems. This helped to develop separate kinematic configurations for the master and the slave. By 1991, Cartesian control was being used to manipulate simulated slave robots. Eventually, the research was directed towards interaction of forces with objects with rich geometrical information. Massie and Salisbury in 1994, designed the Phantom, a stylus based haptic device which was later commercialized.

5.2.2 Haptic Process Flow

Figure 5.1: Phantom from Sensable Technologies [www.sensable.com]

The concept of combining robotic and haptic technologies to touch and manipulate 3D data developed into a full fledged project in the early 1990s. This project evolved into the first ever haptic device called “Phantom”, a force feedback device. The development of Phantom since then has opened a new avenue for computer interaction techniques for visually impaired people and people with physical disabilities. Haptic technology makes it possible to extend the range of touch from the length of an arm to a virtually unlimited distance. The Phantom enables the user to control the robot with small movements of one finger and also feel some of the tasks that the robot performs. The Phantom interacts with the computer to interpret the user’s finger position in three dimensional spaces and applies an appropriate and variable resisting force.

Three sensors located in the Phantom tracks the position of the user’s fingertip and send them to the computer. The computer calculates the necessary force and sends

them to the three DC motors which generate the force that could be felt by the user. The haptic process can be well described in the diagram that follows:

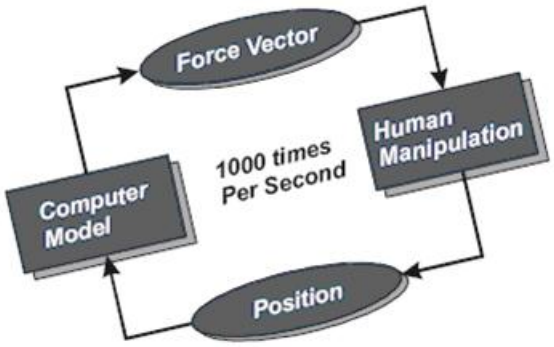


Figure 5.2: Haptic Process Flow [www.sensable.com]

This process is carried out 1000 times per second. The high frequency along with high resolution of the encoders makes it possible to feel almost any shape very realistically with a device like the Phantom.

5.3 Software

The General Haptic Open Software (GHOST) SDK toolkit is a powerful C++ software toolkit that eases the task of developing touch-enabled applications. This software essentially takes care of the complex computations and helps the users deal with simple, high level objects and physical properties like location, mass, friction and stiffness. The GHOST SDK also consists of libraries of 3D prismatic objects, touch effects etc. Hence, all the computations necessary to simulate physical interaction with graphical objects is provided by these libraries.

The Sensable OpenHaptics toolkit enables software developers to add haptics and true 3D navigation to a broad range of applications. This software allows the user to add functionality to support new types of shapes. They can also integrate third party libraries such as physics/dynamics and collision detection engines.

5.3.1 Open Haptic Overview

The OpenHaptics toolkit mainly consists of the following libraries:

- Haptic Device API (HDAPI)
- Haptic Library API (HLAPI)

5.3.1.1 HDAPI

This provides low-level access to the haptic device renders direct force, controls run time behavior of the drivers. The HDAPI consists of two components, the device and the scheduler. The device allows any supported 3D haptic mechanism to be used with the HDAPI. The scheduler callbacks meanwhile, allows the user to enter commands that will be performed within the servo loop thread. The HDAPI is generally used to initialize a device, create scheduler callbacks to define force effects, enable forces and start the scheduler. An example of force effect is the query of position device at every scheduler tick and the calculation of force based on that. The device routines can be classified into device initialization, device safety and device state. The scheduler's main purpose is the rendering of forces and retrieval of state information from the device. In order to create compelling and stable force feedback, the force updates need to be sent at 1000 Hz frequency. The scheduler interface allows the application to communicate effectively

performed in the servo loop thread in a thread safe manner. This also allows operations to be performed in the servo loop thread.

5.3.1.2 HLAPI

The HLAPI allows rendering of geometric primitives along with haptic material properties. The haptic rendering machine uses this information along with data read from haptic device to calculate the appropriate forces to be sent to the haptic device. The HLAPI commands can modify the rendering state of haptic device and store important information like position and orientation. The API also has the ability to set event callback functions which the rendering engine can call whenever any event like, touching a shape or pressing the stylus button on the haptic device occurs.

5.3.2 OpenGL Graphical Software

OpenGL is a graphical software tool that can be used to develop interactive 2D and 3D graphical applications. We used the OpenGL graphical software to simulate the environment for the Fitts' experiments in all X, Y and Z axis respectively. Moreover, walls and floor were also graphically drawn to give the user a sense of being inside a room. This made the whole experiment very real and visually very guiding.

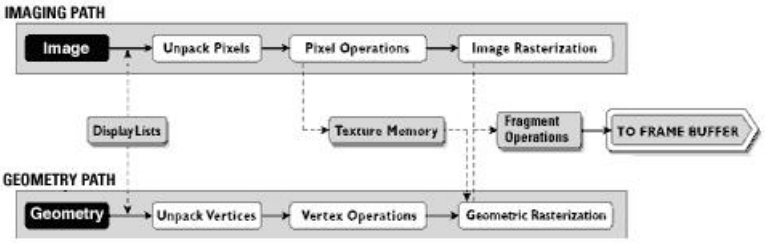


Figure 5.3: OpenGL Visualization Programming Pipeline

The graphical scene was simulated using OpenGL with very simple commands like:

gluSphere(quadObj, VISITOR_SPHERE_RADIUS, 20, 20);

This command generated a virtual sphere with a radius represented by VISITOR_SPHERE_RADIUS, and the remaining two arguments referred to slices and stacks which made the sphere visually realistic.

```
glBegin(GL_QUADS);
glLineWidth(10.0);
  glVertex3d(pointstart.x-10.0,pointstart.y+40.0,pointstart.z);
  glVertex3d(pointstart.x+10.0,pointstart.y+40.0,pointstart.z);
  glVertex3d(pointstart.x+10.0,pointstart.y,pointstart.z);
  glVertex3d(pointstart.x-10.0,pointstart.y,pointstart.z);
glEnd;
```

These OpenGL commands were used to draw each of the rectangular goal regions at the two ends of the linear path. The command glVertex3d is used to represent a point in a 2D or 3D Space.

5.3.3 Multithreading

The concept of Multithreading is implemented here to enable real time generation of haptic data and to enable simultaneous running of the haptic and graphic loop. Multithreading is the concept of integrating the Phantom haptics thread with the graphics thread to work concurrently. The haptics needs to be updated more frequently than the graphics and hence the HLAPI creates two additional threads called the collision thread and servo thread. While the servo thread handles direct communication with the haptic device, the collision thread determines which geometric primitives are in contact with the proxy.

The servo thread runs at a frequency of 1000 Hz while the collision thread runs at 100 Hz. In this particular application, a synchronizer structure is created which stores variables that can be modified by the haptics thread and simultaneously used by the graphics thread. The synchronizer basically gets a snapshot of data from the haptics thread in a thread safe fashion. The synchronizer uses the same pointer to the haptic state as that used in the haptics loop. The graphics loop however, access the haptic state in a thread safe manner by using a synchronous callback. This callback is executed in a thread-safe and realtime fashion. This can be seen in the realtime force generated from Phantom.

6. Description of the Different Assistive Functions

We designed six types of tasks provided in a GUI from which the user can choose the one he or she wants to perform. The assistance concepts described before, are applied to the tasks described here in this chapter.

6.1 Introduction

This chapter describes the application of the various assistance concepts for 2D and 3D virtual tasks in a haptic environment. This is the basis of designing repetitive tasks like the Fitts' Task with assistance functions for upper arm rehabilitation purposes.

6.2 Trajectory Approach and Traversal

One of the basic haptic experiments is where the user approaches a trajectory and moves along the same to reach the target. This system is designed to assist the user in trajectory approach and traversal with effective force feedback. The system is also designed in a way that human motion can control the movement when necessary. In this case, the user intentions are always considered. Here, the assistive functions have been used to enable the user to perform with ease without any unnecessary buzzing or jittery motion.

6.2.1 Experimental Set-Up

The experimental set-up consists of a display window depicting a linear trajectory and the end-effector represented by the blue sphere on a user interface (UI), as shown in Figure 6.1.

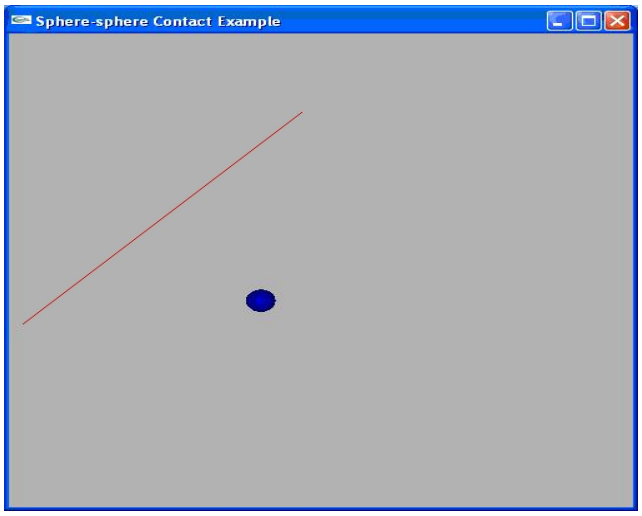


Figure 6.1: UI with the Desired Trajectory and End-Effector

This sphere follows the motion of the end-effector of the Phantom Omni device. The user controls the motion of the sphere with the haptic stylus. The goal of the user is to move the sphere towards the trajectory and move it along the linear path. The user traverses the trajectory path as shown in Figure 6.2. Here all the directions of motion is in the X-Y plane.

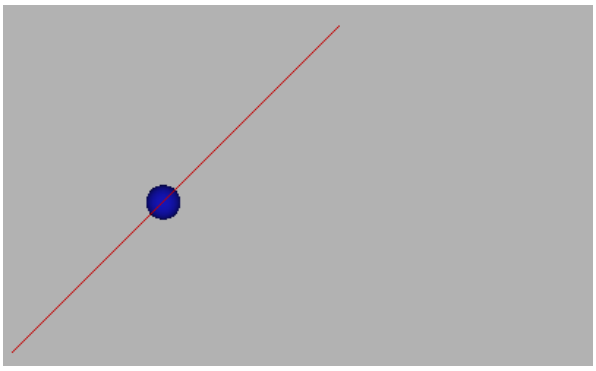


Figure 6.2: UI Showing the End-Effector on the Trajectory Path

6.2.2 Terminology Used in Trajectory Approach Task

Here is the terminology which clearly explains the logic behind this assistance concept:

- $(x1, y1, z1)$ = Start Point of the User Defined Trajectory
- (x2, y2, z2) = End Point of the User Defined Trajectory
- (X_n, Y_n, Z_n) = Point on the Trajectory Which is Closest to the User Controlled Phantom End – Effector Position.
- (U_1, U_2, U_3) = User Controlled Phantom End-Effector Position
- $\vec{F}$ = Force Vector at the User Controlled Effector Position
- F_1, F_2, F_3 = Force Components in the X, Y, and Z Directions Respectively
- Line_Mag = Magnitude of the Length of the Trajectory Path
- (I_1, I_2, I_3) = Closest Point on the Trajectory

6.2.3 Stiffness

This is the resistance of an elastic body to deflection or deformation by an applied force. This is defined as,

$$Stiffness = P / \delta \tag{6.1}$$

This is the stiffness of a body that deflects a distance δ, under an applied force P.

6.2.4 Distance

This is the distance between the user's position and the closest point on the trajectory to the user. The closest distance is computed based on a point on the trajectory which lies on the normal to the trajectory path.

If (X_n, Y_n, Z_n) is the closest point on the trajectory to the user, and if (U_1, U_2, U_3) is the user's end-effector position, then the distance can be represented as:

$$Dist = \sqrt{(U_1 - X_n)^2 + (U_2 - Y_n)^2 + (U_3 - Z_n)^2} \tag{6.2}$$

6.2.5 Trajectory Vector

The trajectory vector is a vector defined in the direction of the trajectory path. It is given as follows:

$$\vec{t} = (x_2 - x_1)\vec{i} + (y_2 - y_1)\vec{j} + (z_2 - z_1)\vec{k} \tag{6.3}$$

6.2.6 Normal Vector

The Normal vector is the vector which is in the direction of the normal to the closest point on the trajectory.

$$\vec{n} = ((x_2 - x_1)/Dist)\vec{i} + ((y_2 - y_1)/Dist)\vec{j} + ((z_2 - z_1)/Dist)\vec{k} \quad (6.4)$$

Where:

- K = also known as the scaling factor, this value can be used to Scale up or scale down the force to get the desired effect.
- Y_start = this is the predefined point in the Y axis below which the user is allowed to approach the trajectory easily.
- Y_end = this is the predefined point in the Y axis above which the user is allowed toleave the trajectory easily.

Some of these variables will be described in a later section in this chapter.

The following section describes how the distance between the end-effector and the closest point on the trajectory is determined.

6.2.7 Determination of a Point on the Trajectory Closest to the User's Position

The length of the trajectory line Line_Mag, is determined as follows:

$$\text{Line\_Mag} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad (6.5)$$

This mathematical computation is explained very descriptively in Figure 6.3.

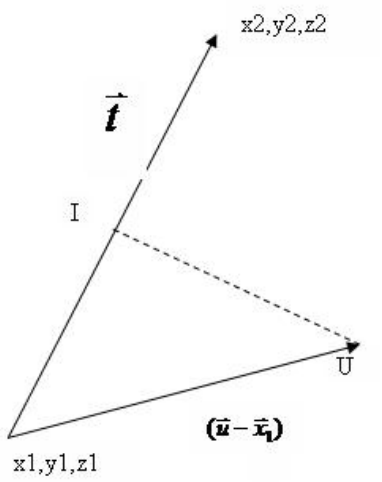


Figure 6.3: Calculation of the Nearest Point on a Line to a Trajectory

Next, the dot product $\vec{P}$ of the trajectory vector $\vec{t}$, and line joining the end-effector position, is computed as shown below.

$$P = ((U_1 - x_1) * (x_2 - x_1) + (U_2 - y_1) * (y_2 - y_1) + (U_3 - z_1) * (z_2 - z_1)) / \text{Line\_Mag} \quad (6.6)$$

Then, the dot product is checked P is checked for perpendicularity.

If ($0 < P < 1$), then the closest point on the trajectory is computed as follows:

$$I_1 = x_1 + P * (x_2 - x_1) \quad (6.7)$$
$$I_2 = y_1 + P * (y_2 - y_1) \quad (6.8)$$
$$I_3 = z_1 + P * (z_2 - z_1) \quad (6.9)$$

If the condition is not true, then the closest point does not lie near the line segment.

6.2.8 Assistance Concepts Applied to this System

In order to understand the application of the assistance concepts, each instance of assistance application is descriptively explained next. The force algorithm applied when the user approaches the trajectory, is first explained here. The corresponding force flow

diagrams are also shown immediately. Next, we are going to explain all the assistance scenarios figuratively in five cases as follows. These five cases represent different scenarios of trajectory approach and traversal. The first scenario is one where the user approaches a trajectory and the forces that influence him or her in performing the same task. The force applied here is an exponential force which increases slowly as the distance between the user represented end-effector and trajectory is reduced.

6.2.8.1 Case a: When the User Approaches the Trajectory

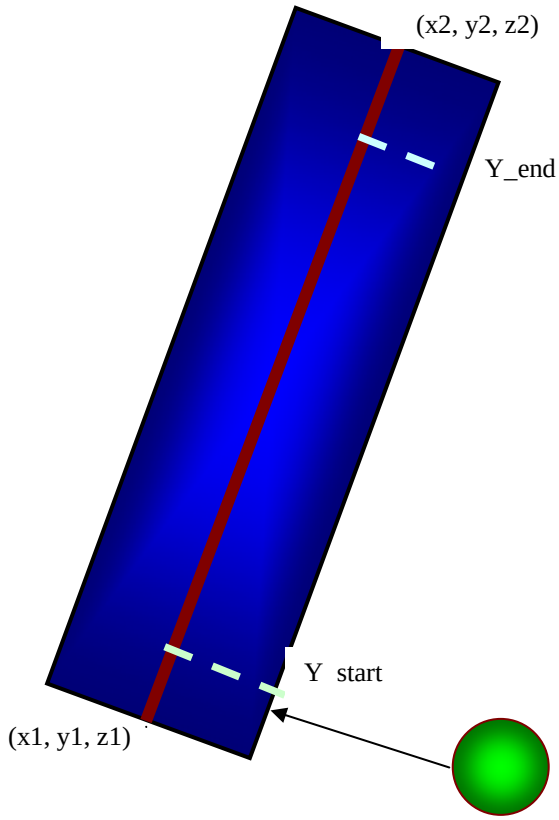


Figure 6.4: Graphical Representation of a User Approaching a Trajectory

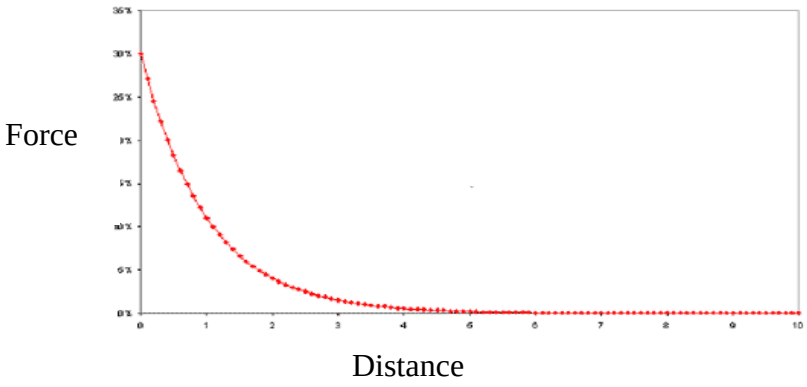


Figure 6.5: Negative Exponential Relationship between Force and Distance

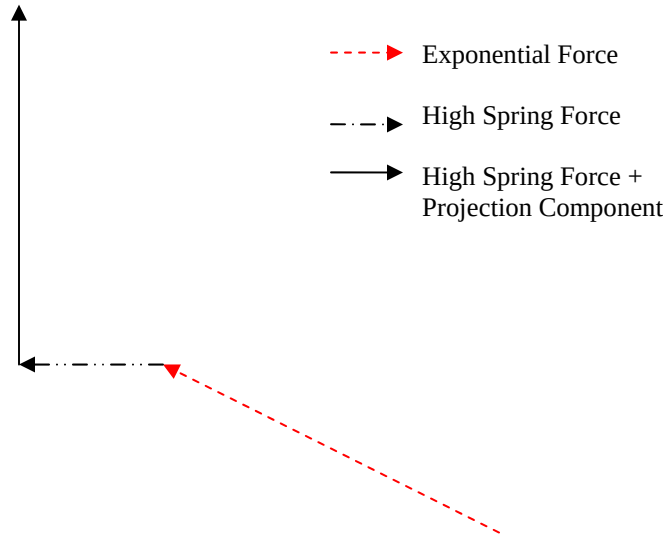


Figure 6.6: Graphical Representation of Force Direction in Case a

$$F1 = K * e^{-a1Dist} \tag{6.10}$$

$$F2 = K * e^{-a2Dist} \tag{6.11}$$

$$F3 = K * e^{-a3Dist} \tag{6.12}$$

Where:

- F1, F2, F3 = Force Components in X, Y and Z Directions
- K = Scaling Factor
- $a1, a2, a3$ = Scaling Factor
- Dist = Distance Between the Current Effector Position and the Closest Point on the Trajectory.

The trajectory approach task has been graphically depicted in Figure 6.4. This exponential force is given to the user when the Y coordinate of the end-effector position falls within the Y-start limit. This force is intended to help the user approach the trajectory at its starting point. The exponential force increases as the user gets closer to the trajectory. This exponential force is represented by the red dashed line as shown in Figure 6.6. The Y-start is represented by a dashed line in the lower end as shown in Figure 6.4. The negative exponential relationship between force and distance is graphically shown in Figure 6.5.

6.2.8.2 Case b: When the User is Very Close to the Trajectory and Under the Y_Start Point.

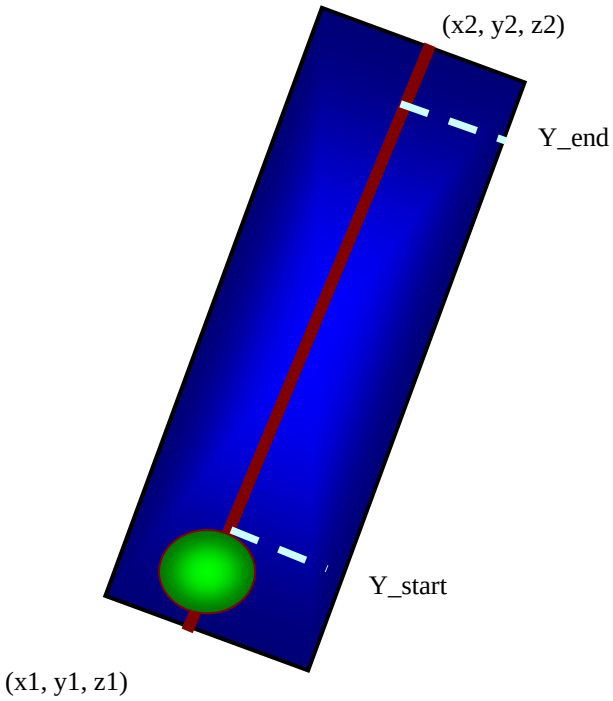


Figure 6.7: Graphical Representation when a User is Very Close to the Trajectory

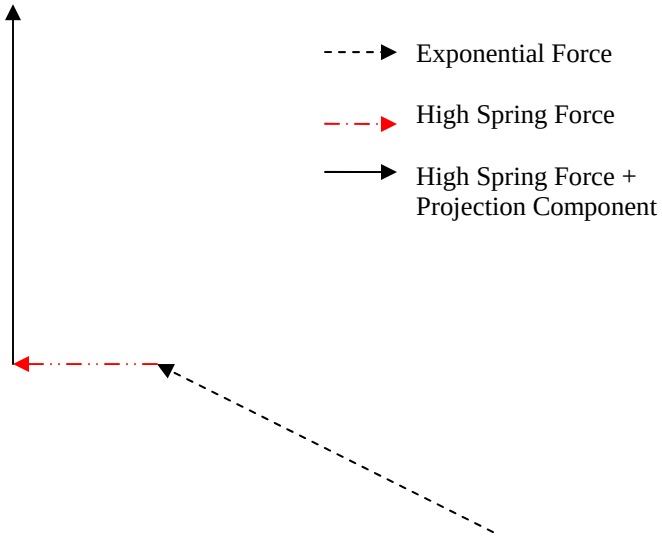


Figure 6.8: Graphical Representation of Force Vector Acting in Case b

$$\vec{F}_1 = 18.0 * STIFFNESS * Dist * \vec{n}_x \ ; \tag{6.11}$$

$$\vec{F}_2 = 18.0 * STIFFNESS * Dist * \vec{n}_y ; \tag{6.12}$$

$$\vec{F}_3 = 18.0 * STIFFNESS * Dist * \vec{n}_z ; \tag{6.13}$$

Where, n_x, n_y, n_y are the three components of the line vector normal to the trajectory. The high spring force attracts the sphere to the trajectory at a very high magnitude such that the user feels a high pulling force to the trajectory. Figure 6.8 contains the force diagram with the high spring force highlighted in red.

6.2.8.3 Case c: When the User is On the Trajectory

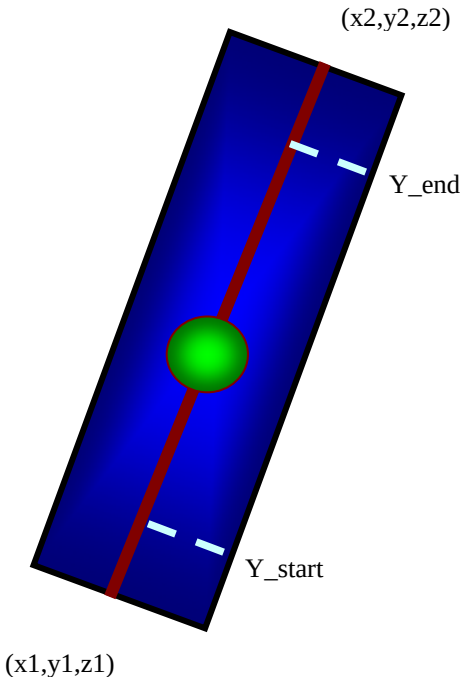


Figure 6.9: Graphical Representation when a User is on the Trajectory and Above Y_start

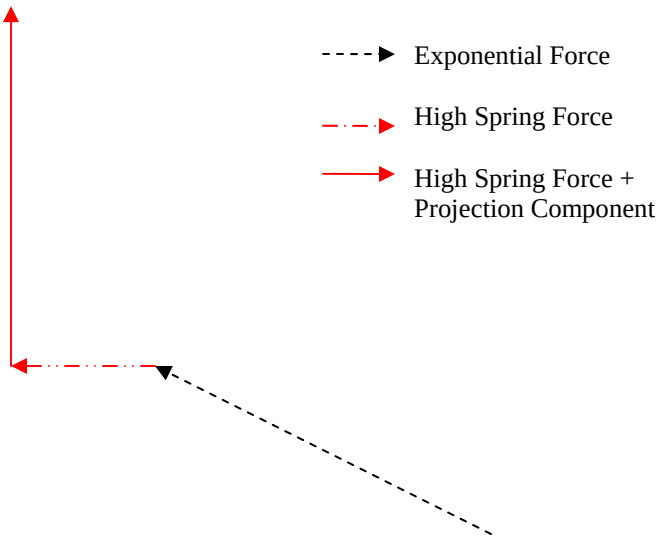


Figure 6.10: Graphical Representation of Forces Acting in Case c

$$\vec{F} = (K * STIFFNESS * Dist * \vec{n}) + (K * \vec{t}) \quad (6.12)$$

Where:

- $(K * STIFFNESS * Dist * \vec{n})$ = Spring Force Component which Assists the User to move towards the Trajectory Path and Stick to it.
- $(K * \vec{t})$ = Projected Force Component which Assists the User to Move Along the Trajectory Path to the End Point of the Specified Trajectory.
- K = Scaling Factor

When the user is on the trajectory, a projection force is applied in the direction of the trajectory so that the user gets assistance to reach the end of the trajectory as shown in Figure 6.9. Here, the spring force is scaled down a bit so that, buzzing is eliminated. The two component forces acting on the end-effector sphere are highlighted in red as shown in the force diagram in Figure 6.10. The next case study talks about the forces acting when the user is at very close proximity to the trajectory. In this scenario, the user is

expected to be above Y_start limit as shown in the Figure 6.11. There is a strong attractive force that pulls the user close to the trajectory path. This force is a high spring force applied in the direction of the path between the user represented end-effector and trajectory.

6.2.8.4 Case d: When the User is at Very Close Proximity to the Trajectory

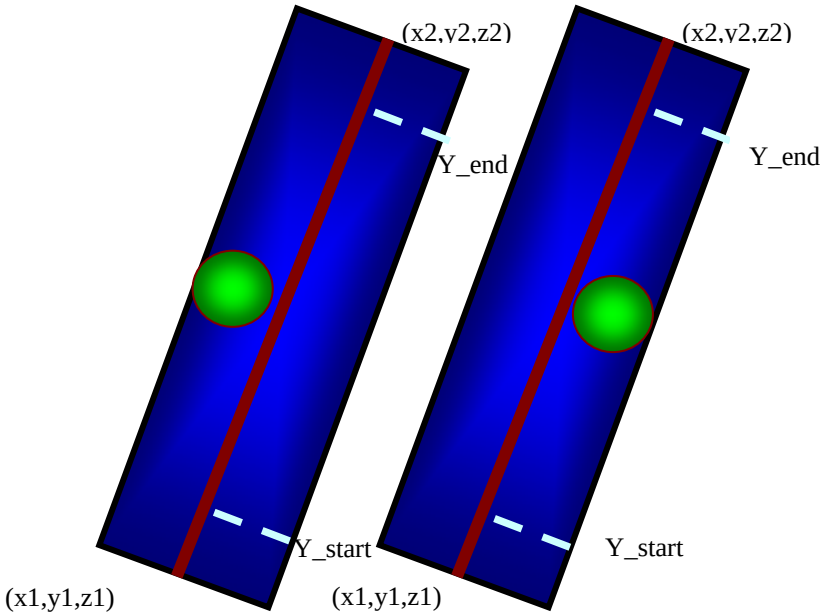


Figure 6.11: Graphical Representation when a User is Near Trajectory and Above Y_start

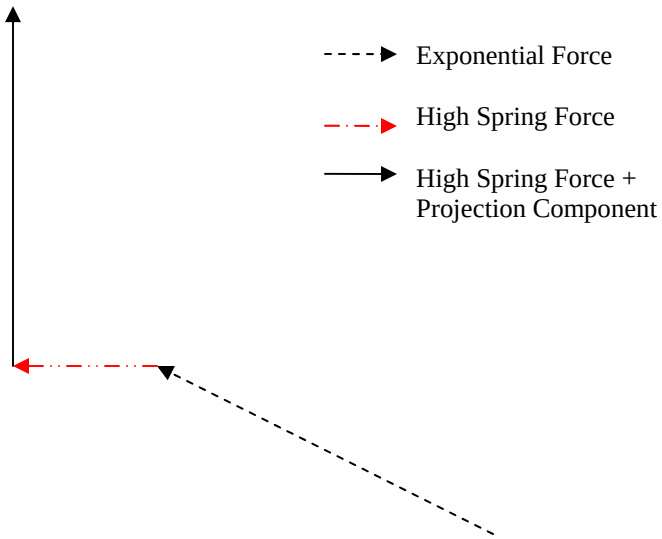


Figure 6.12: Graphical Representation of Force Vector

$$\vec{F} = K * STIFFNESS * Dist * \vec{n} \tag{6.13}$$

$$\vec{n} = ((x_2 - x_1)/Dist)\vec{i} + ((y_2 - y_1)/Dist)\vec{j} + ((z_2 - z_1)/Dist)\vec{k} \tag{6.14}$$

Here, Dist is the distance between the effective sphere and the closest point on the trajectory. This high magnitude spring force is applied on the left and right hand side of the trajectory as shown. These two forces become virtual walls and don't allow the user to move beyond them. These virtual walls are designed at pre-defined distances from the trajectory. The force on the right hand side of the trajectory is given at a magnitude lesser than the left side of virtual wall so that the user can pull away from the trajectory at any position of the end-effector. This force acts within the virtual wall. The virtual wall is clearly visible in Figure 6.11 and they basically prevent the end-effector from moving away from the trajectory path. The force diagram is given in Figure 6.12 where the active

force in this condition is highlighted. The next case describes the scenario where the user has reached the end of the trajectory path and is ready to move away from the same.

6.2.8.5 Case e: When the User Reaches the End of the Trajectory Path

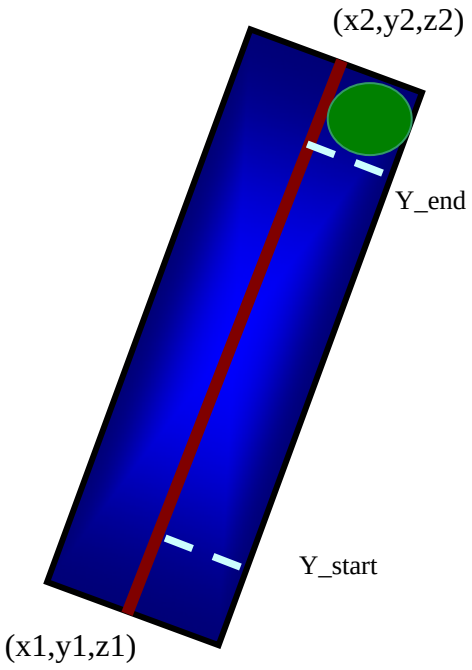


Figure 6.13: Graphical Representation when a User Exceeds Y_end Position

$$F = 0.0 \tag{6.15}$$

When the user exceeds the Y_end position, all forces are made null as shown in equation 6.15. This enables the user to easily get out of the trajectory and back to the rest position. The Y_end point is represented by a dotted line at the upper end of the trajectory path.

6.3 Three Dimensional Force Scaling

6.3.1 Description

The graphical scene of the multidimensional trajectory path is similar to the 2D trajectory path. Here the user travels in X, Y and Z directions in 3D space.

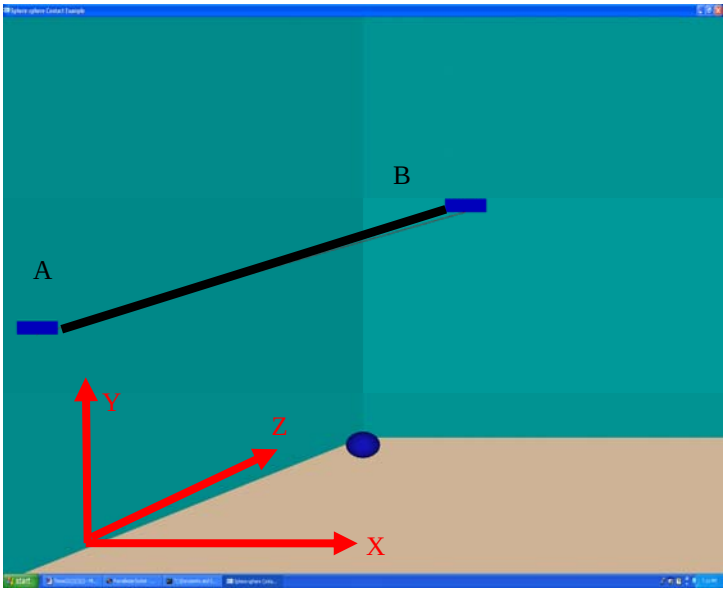


Figure 6.14: UI with End-Effector and a Three Dimensional Trajectory Path

The multidimensional assistance concept was developed in a haptic and OpenGL environment as shown in the above figure. The room environment was simulated graphically using OpenGL commands. The walls were drawn in a way so that the user felt he or she was moving in the corner of a room. The visual enhancement helped in assisting users to respond to assistive forces in a better way. The trajectory path was drawn in this environment with the end coordinates varying in all three directions of X, Y and Z. The user thus would travel in a 3D space when moving on this trajectory path. The

two rectangular blocks A, B were drawn to give the user a better sense of judgment of the start and end points of the trajectory line.

6.3.2 Method

This concept focused on providing assistance to the user for bidirectional movement on the trajectory. The first rectangular region was denoted as A and the second one was represented as B. If the user was near A and was proceeding to move towards the target B, then scaled forces were provided in X, Y and Z directions to push the user smoothly towards B. Similarly, when the user wanted to return to A from B, appropriate forces were provided to assist movement in the reverse direction. In the next page, there is a flowchart presenting the logic behind the 3D assistive concept.

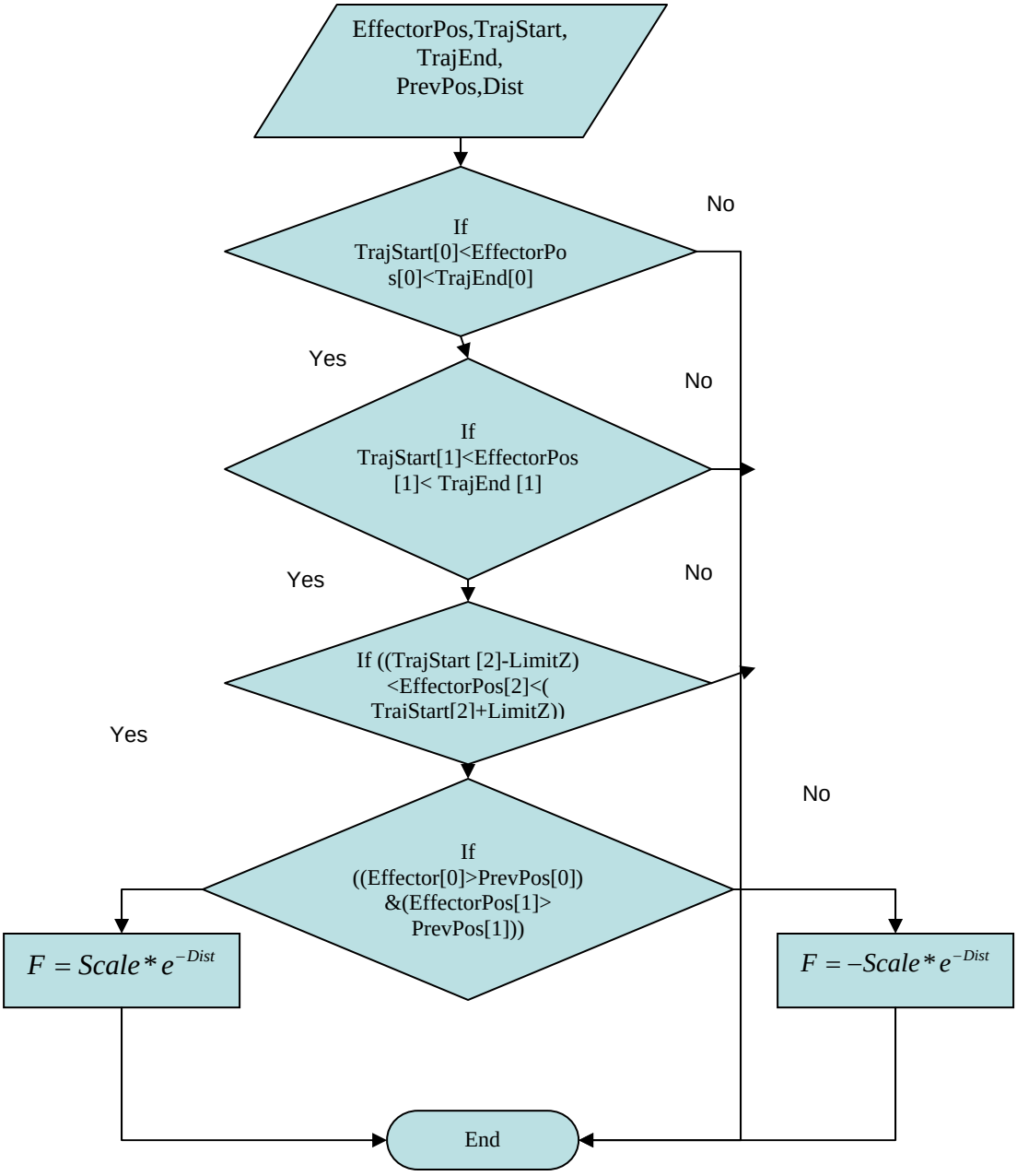


Figure 6.15: Flowchart Describing Assistive Functions for Three Dimensional Trajectory Traversal

6.3.2.1 Terminology for 3D Force Scaling

EffectorPos is a vector which represents the position of the Phantom end-effector in the haptic environment. EffectorPos [0], EffectorPos [1], EffectorPos [2] are the components of the vector in the X, Y, and Z directions respectively. TrajStart is a vector which represents the starting point of the three dimensional trajectory path. TrajStart [0], TrajStart [1], TrajStart [2] are the components of the vector in the X, Y, and Z directions respectively. TrajEnd is a vector which represents the end point of the three dimensional trajectory path. TrajEnd [0], TrajEnd [1], TrajEnd [2] are the components of the vector in the X, Y and Z directions respectively. PrevPos is a vector which represents the previous position of the haptic end -effector for any given current effector position. PrevPos [0], PrevPos [1], PrevPos [2] are the components of the vector in the X, Y and Z directions respectively. LimitZ is the range of Z values for which the assistive forces should be active. This limit in Z is parallel to the Z axis in OpenGL. Dist is the distance between the end-effector position and the closest point on the trajectory.

6.3.2.2 Explanation of Flow Chart

The first step was to check for the position of the end-effector. It was made sure that the effector fell within an agreeable range of X, Y and Z coordinates in the 3D space. If any of these conditions were not satisfied, then the program terminated. Once the user controlled effector sphere was within the assistance range, and going from A to B, the following force pushed him or her ahead.

The force computed here was based on the Exponential Law where:

$$F1 = K * e^{-a1Dist} \tag{6.16}$$

$$F2 = K * e^{-a2Dist} \tag{6.17}$$

$$F3 = K * e^{-a3Dist} \tag{6.18}$$

Where:

- F1, F2, F3 = Force Components in X, Y and Z Directions
- $a1, a2, a3$ = Scaling Factors in X, Y and Z Directions
- Dist = Distance between the User Position and the Closest Point on the Trajectory

If the user was moving from B to A, then a force in the opposite direction was implemented.

$$F1 = -K * e^{-a1Dist} \tag{6.19}$$

$$F2 = -K * e^{-a2Dist} \tag{6.20}$$

$$F3 = -K * e^{-a3Dist} \tag{6.21}$$

The user just had to pull out of the trajectory to get out of the force region.

6.4 User Controlled Velocity Based Force Scaling

6.4.1 Experimental Set-Up

The user-controlled velocity based force scaling has a similar environmental set up as that of the previous experiment.

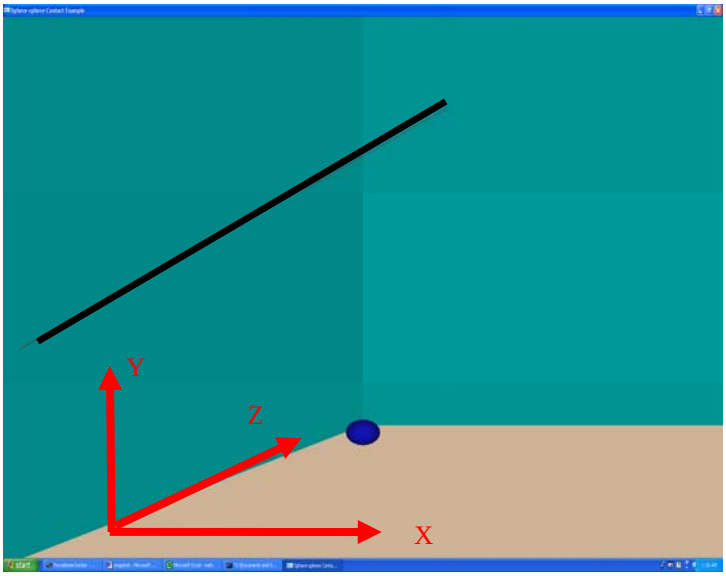


Figure 6.16: User Controlled Velocity Based Force Scaling

6.4.2 Method

The user had to move the Phantom controlled end-effector sphere towards the starting point on the trajectory and move on the line towards the target point or end of the trajectory as shown in Figure 6.16. The force algorithm applied here is similar in concept to the logic applied in the trajectory approach and traversal experiment. The only difference here would be the addition of extra set of code statements to decide the scaling value in the force computation algorithm. The C++ application is designed in a way that the user's haptic velocity can be determined as real time data and displayed simultaneously. The command to determine the user controlled end-effector velocity is given by the following command which falls in the next page.

\

hdGetDoublev(HD_CURRENT_VELOCITY,vel);

Then the software velocity limit of the Phantom was determined using the following command,

hdGetDoublev(HD_SOFTWARE_VELOCITY_LIMIT,Vmax)

Then the user velocity is compared to maximum allowable velocity to determine the level of scaling that would suit the user motion best. The application of user controlled force scaling enables the user to have more control and smoother motion in a haptic process.

7. Experiments Based on Fitts' Task

These experiments are performed to validate the data which may indicate the significance of the assistance concepts applied here. These experiments are more like preliminary experiments which gives an indication of the versatility of application rather than restoring a person's function. This chapter describes two types of experimental designs based on the Fitts' Tasks. The experimental designs are:

- Validation of Assistance Concept
- Position Accuracy

The sample population consists of 8 young, healthy subjects. They were divided into two groups, A and B. One group was given training with assistance, the other group was given rehabilitative training without assistance. The validation of the assistance concept experiment is designed to indicate that the Phantom based rehabilitation training with assistance helps the user to learn the task faster compared to those who were not given training with assistance. In the second experiment, we verify the accuracy of two of the eight volunteers. Among the two, one of them was from Group A, and the other from Group B. Now, this chapter talks in detail about the Fitts' experimental setup.

7.1 Experimental Set-Up

7.1.1 Graphical User Interface

The C++ GUI consists of a list of different types of assistive options such as, user controlled velocity based force scaling, multidimensional force scaling and the Fitts' Task experiments in 3D. The interface is basically a popup menuhandler which is activated on the right click of the mouse on an OpenGL window. The interface is displayed on the screen as shown in the Figure 7.1.

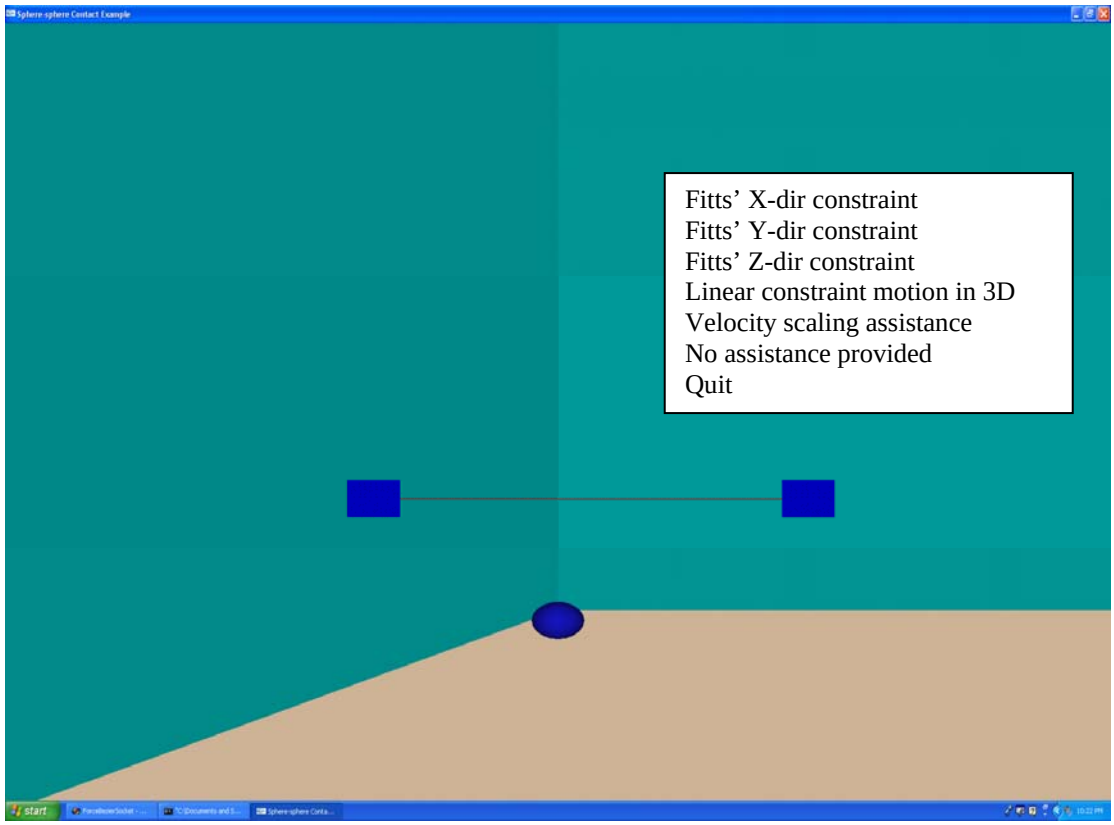


Figure 7.1: Graphical User Interface

We chose to perform the Fitts' experiment which gives a meaningful insight into the speed and accuracy of human motion. The following section describes two experiments which were designed based on the Fitts' Law.

7.1.2 Validation of Assistance Concept

In this experiment, emphasis was laid on the observed experimental execution time when assistance was provided compared to the case when no assistance was provided. The sample population consisted of eight normal, healthy young people, who fell within an age group of 20-30 years. They were divided in an unbiased way into two groups, Group A and Group B. All the members of Group A and Group B were asked to perform Fitts' Task X, Fitts' Task Y, and Fitts' Task Z for a total sum of seven trials in each direction for a total of 21 trials. Thus, each volunteer performed a total of 63 trials. The time was noted for a total of 63 trials. The observed time for all eight volunteers was recorded. Each of the three experiments was performed for three different (A/B) ratios. The (A/B) ratio is the ratio of the distance between the two targets and the width of each target. The Fitts' Law essentially states that the time taken to move from one target to another can be described as a function of distance between the two targets and the width of each target.

7.1.3 Baseline Time Values

In the first trial, all the eight volunteers were asked to perform the Fitts' experiment in all directions for three different A/B values without any assistance. The

time recorded here was noted as the baseline value for each. In the second trial, all the experiments were repeated with assistance provided to each of them. The time recorded here was observed as baseline time value with assistance for each. These time values were designated as t_i.

7.1.4 Fitts' Training

Next, the eight volunteers were divided into groups, Group A and Group B. In the next three trials, Group A was given assistance in executing the same set of tasks again. Group B was not provided assistance in executing the same set of tasks. This is called the training stage where, Group B was given training in Fitts' Task with assistance and Group A was given training in Fitts' Task without assistance, and the average of observed time for both the groups was recorded separately.

7.1.5 Final Task

The last trial involved performing the same set of experiments without any assistance by all the volunteers in Group A and Group B. Again, all the volunteers were asked to perform the same set of experiments with assistance. The time recorded here for each one of the volunteers was designated as t_f.

$$\text{DeltaT} = t_i - t_f \tag{7.1}$$

Where:

- t_i = Time taken at Baseline Task
- t_f =Time taken at the Final Task

The average Delta T was calculated as shown above for each of the eight volunteers in all three directions of X, Y and Z.

7.1.6 Determination of Position Accuracy

This experiment involves just two subjects who were a part of the previous experiment. The first subject performed the Fitts' experiment without assistance for six trials in X, Y and Z directions. The second subject repeated the same experiments but with assistance.

In the next type of position accuracy experiment, time was kept constant and the haptic position data was recorded and plotted for both the subjects. In the last experiment in position accuracy, the haptic position of the subjects was compared to themselves before and after training. Here is a brief description of the Fitts' X, Y and Z Tasks.

7.2 Fitts' Task X-Direction

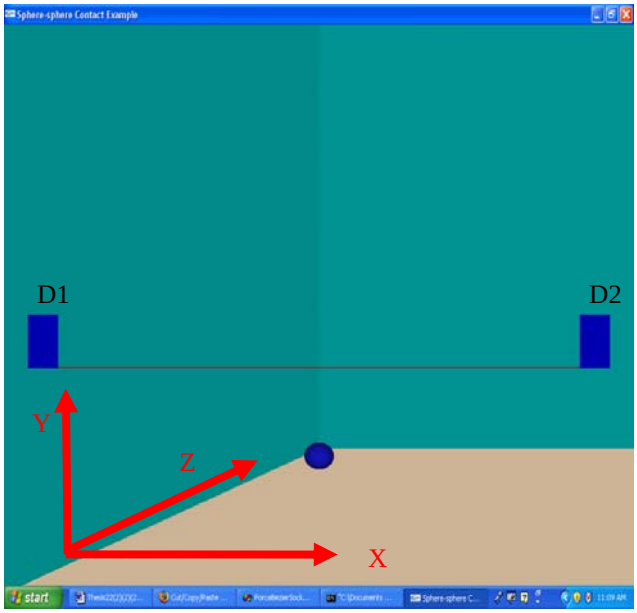


Figure 7.2: Schematic Representation of Fitts' Task X

The Fitts' Task X was the first experiment conducted where the range of motion was constrained in the X direction alone in 3D virtual space. This is clearly depicted in Figure 7.2. The rectangular boxes shown in the above figure represented the goal points at the either end of the path. The blue sphere represented the end-effector of the Phantom which was user controlled. The scene consisted of two walls and a floor. This gave a realistic environment to the user. The user felt as if, he or she was moving a virtual object in a 3D room. The visual feedback has been proved to help the user perform better in virtual space. The distance and goal region width measurements were made. The distance between the centre of the two goal regions D1 and D2 was represented by A. The width of both D1 and D2 was made equal to each other and was represented by B. The Fitts' Task in the X direction was conducted as follows. The user was first asked to haptically

move towards the midpoint of D1. The task involved the user to move from mid point of D1 to the midpoint of D2 and trace the path back from midpoint of D2 to midpoint of D1 without any stopping. About three different values of A/B ratio was computed. While Group A was provided with no kind of assistance, the Group B was given force assistance in performing the Fitts' Task X. For each A/B ratio, the time taken to move from D1 centre to D2 centre and back again was measured using a stop watch. The time values are discussed in a later chapter. Meanwhile the average Delta T values were also computed for each of the eight participants and compared among Group A and Group B members using a histogram chart. This chart will be explained in the results chapter. The second experiment involving position accuracy was further classified into two types:

- Non Constant Time
- Constant Time

7.3 Fitts' Task Y-Direction

The Fitts' Task Y allowed the user to travel in the Y direction alone, as shown in Figure 7.3. The environmental set up for the task was the same as that of the previous experiment. The walls of the room were simulated haptically as before. The rectangular goal regions were displaced in the Y direction. D1 is the rectangular target which was drawn in the lower part of the trajectory and D2, at the upper end of the trajectory. The user was asked to perform the same task of moving from D1 to D2 and vice versa.

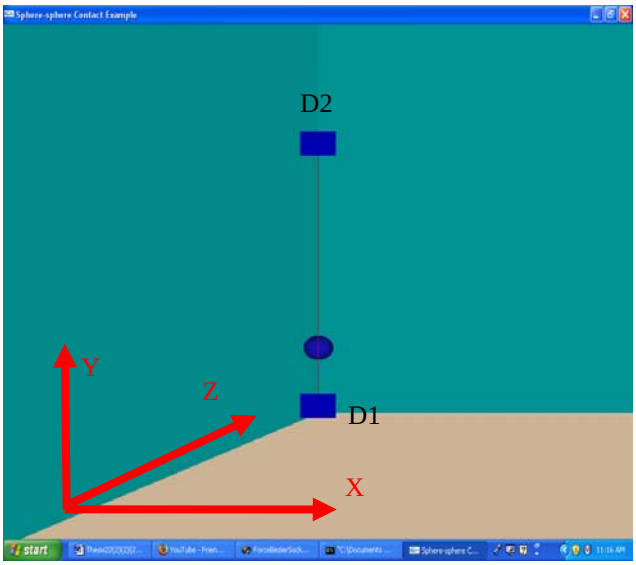


Figure 7.3: Schematic Representation of Fitts' Task Y

While Group A was not given any assistance while performing this experiment, Group B was provided assistance based on the logic explained in the previous chapter. The average Delta T was also computed here as previously described and plotted in the same histogram for all the eight subjects. The determination of position accuracy was performed for Fitts' Task Y also. The real time position values of the two subjects were recorded haptically and compared to the trajectory points in the Y direction from D1 to D2. The position accuracy experiments were also performed in Fitts' Y.

7.4 Fitts' Task Z-Direction

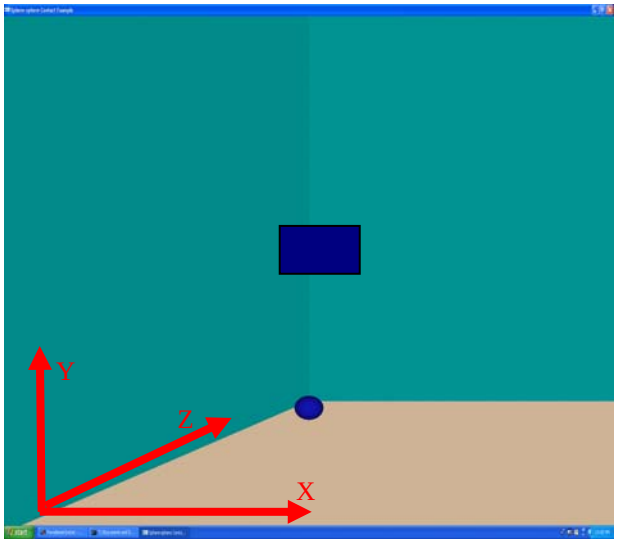


Figure 7.4: Schematic Representation of Fitts' Task Z before OpenGL Camera Rotation

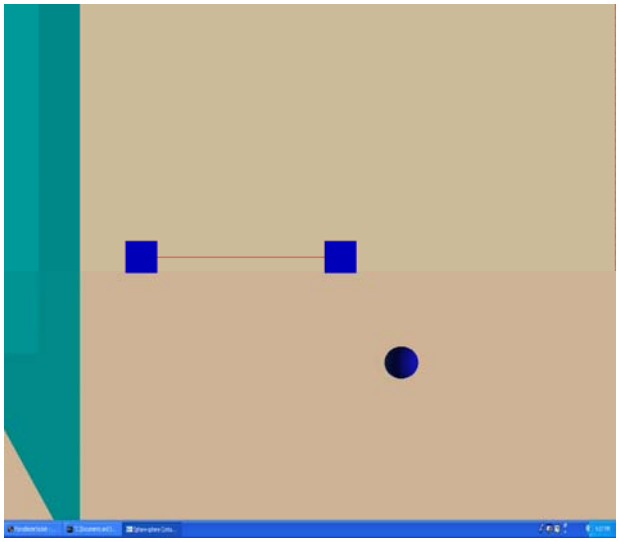


Figure 7.5: Schematic Representation of Fitts' Task Z after OpenGL Camera Rotation

The Fitts' Task Z consists of a trajectory drawn in the Z axis as shown in Figure 7.4. The OpenGL camera location is changed and rotated at an angle such that viewer

looks at the Z axis from a side view in Figure 7.5. This is because, the Z axis is located perpendicular to the OpenGL screen and hence the computer's screen for the default OpenGL camera settings. The camera was thus rotated to visually map the Z axis of the OpenGL screen to the real world Z axis of the Phantom. The environmental set up for the task was the same as that of the previous experiment. The walls of the room were simulated haptically as before. The rectangular boxes were simulated at the two ends of the trajectory path. The lower and upper rectangular boxes were denoted as D1 and D2 respectively. The time taken for the user to move from D1 to D2 and back to D1 was noted as T3. The experiment was repeated for all the eight participants in moving from D1 to D2 and from D2 to D1. The assistance concept was provided depending on which group they fell into. The theoretical time was calculated using the Fitts' Law. The Delta T values were calculated as described in the previous experiments and added to the histogram. Meanwhile, the position accuracy test was also performed in the Z direction for the two subjects. Also, the haptic position data was compared to the trajectory position data both before and after training for both the subjects. The position accuracy experiments will be explained in the next chapter along with detailed graphs and tables.

8. Results

The data obtained from the previous mentioned experiments are preliminary data and provide a basis of information for further testing. These results indicate the advantage of having this C++ based platform for future rehabilitation applications. The data presented in this chapter are indications of the effectiveness of the assistance concepts and intends to validate the assistance concepts. In the first experiment, the following observations were made:

- The performance of assistance functions with respect to the C++ application was verified.
- In the second experiment the position accuracy was tested to see how closely the subjects traveled near the trajectory path. The position accuracy is meanwhile divided into two types.
- In this chapter, we analyze and compare the performance of eight subjects for the previously mentioned experimental procedures.
- Also, the reliability of the C++ application is tested here.
- The experiments were performed to confirm the versatility of the system, in terms of the effectiveness of the assist functions, and the benefit of applying multithreading concepts.

The intention of this research is to show how the PC based system can be used as a rehabilitation tool in future. The following is a detailed explanation about the experimental framework. The Group A and Group B members first perform 18 trials as a part of their baseline task. Here, all of them perform Fitts' X, Y and Z once with assistance and once without assistance. The next phase is the experimental phase where Group A performs 27 trials without assistance and Group B performs the same 27 trials with assistance. The last part is the Final task phase where all subjects again perform Fitts' X, Y and Z once with assistance and once without assistance. This phase is important because it determines the actual performance of the subjects after their training sessions. The Fitts' tasks can be described as follows:

Group A (For one A/B) (Without Assistance Training)

- Baseline Task
    - Fitts' X: wa, woa :2 trials
    - Fitts' Y: wa, woa :2 trials
    - Fitts' Z: wa, woa :2 trials
- Fitts' Training
    - Fitts' X: woa: 3 trials
    - Fitts' Y: woa :3 trials
    - Fitts' Z: woa: 3 trials

Group B (For One A/B) (With Assistance Training)

- Baseline Task
    - Fitts' X: wa, woa :2 trials
    - Fitts' Y: wa, woa :2 trials
    - Fitts' Z: wa, woa :2 trials
- Fitts' Training
    - Fitts' X: wa: 3 trials
    - Fitts' Y: wa: 3 trials
    - Fitts' Z: wa: 3 trials

- Final Task
    - Fitts' X: woa, wa : 2 trials
    - Fitts' Y: woa, wa : 2 trials
    - Fitts' Z: woa, wa : 2 trials

Total No of Trials for 1(A/B): 21 trials

Total No of Trials for 3(A/B): 63 trials

Delta T = $t_i - t_f$ where $t_i =$ Time Taken at Initial Task,

$t_f =$ Time Taken at Final Task

Three A/B values = 8.25, 7.75,7

Hence every subject performed 63 trials including Fitts' X, Y and Z.

8.1 Validation of Assistance Concept

The eight subjects were divided into two groups A and B with four members in each group. All the subjects in Group A were provided training without assistance whereas all the subjects in Group B were provided training with assistive concepts. Each of these eight subjects performed a total of 63 trials in Fitts' X, Y and Z directions. The 63 trials were divided into 21 for the Fitts' X, Y and Z each. These 21 trials were further divided into 7 trials for three different (A/B) ratios each. In each of these 7 trials, the first two were called the baseline trial. Here all the subjects irrespective of their groups were asked to perform the experiment once with and without assistance. Then the next three trials were called the Fitts' training session where Group A was given training without assistance and Group B was given training with assistance. The last two trials aptly called the final task is a repetition of the baseline task with the only difference being that it is

performed at the end of the training session. All these experiments are timed using a stopwatch. Now, the time difference between the baseline and final tasks was noted down as Delta time for Fitts' X, Y, Z separately for each of the subjects. Then, the average Delta time values in Group A for Fitts' X, Y, Z were determined separately. This was again repeated for Group B. The higher the Delta time value, the shorter the execution time at the end of the training session, and hence better performance in terms of speed. Group B was expected to show higher Delta time values signifying that they moved faster for the same number of training trials compared to Group A. For three different (A/B) ratios, the average Delta time values are tabulated as shown below. The average time values are determined separately for Fitts' X, Y and Z. Table 8.1 displays the Delta time value between the baseline and final task when executed without any assistance.

Table 8.1: Comparison of Average Delta Time Values when Performed without Any Assistance for Group A and Group B

	Group A (Average Delta Time WOA)			Group B(Average Delta Time WOA)		
A/B	Fitts' X	Fitts' Y	Fitts' Z	Fitts' X	Fitts' Y	Fitts' Z
165/20	0.49975	-1.272	2.2815	1.3405	1.252	1.34675
155/20	-0.75025	0.095	0.81835	0.4375	0.637	1.4505
140/20	-0.50425	0.47575	0.5145	0.62675	0.68525	0.6565

The Table 8.2 displays the average Delta time (baseline-final) values when performed with assistance for Group A and Group B.

Table 8.2: Comparison of Average Delta Time Values with Assistance for Group A and Group B

	Group A(Average Delta Time(WA)			Group B(Average Delta Time WOA)		
A/B	Fitts' X	Fitts' Y	Fitts' Z	Fitts' X	Fitts' Y	Fitts' Z
165/20	0.44125	-0.46475	1.492	0.612	0.69475	1.130667
155/20	0.07275	0.567	0.06045	0.118	0.1775	0.495775
140/20	-0.05063	0.421375	0.6175	0.22825	0.2365	0.229

From Table 8.1, the average Delta time values (without any assistance) of all the three (A/B) ratios are tabulated below:

Table 8.3**:** Comparison of Average Delta Execution Time when Performed without Assistance

Group A			Group B		
Fitts' X	Fitts' Y	Fitts' Z	Fitts' X	Fitts' Y	Fitts' Z
-0.0217	0.084242	1.1585	0.651908	0.856975	1.151167

A bar chart depicting the above tabulated values is shown below. The values are taken from Table 8.3.

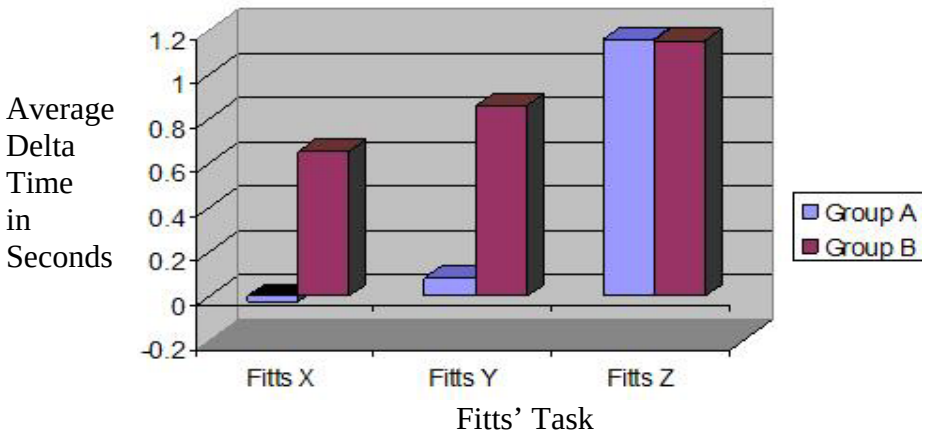


Figure 8.1: Comparison of Average Delta (Baseline-Final) Execution Time when Performed without Assistance

In Figure 8.1, we can see that the average Delta time value is greater for all the subjects in Group B compared to Group A in Fitts' X and Fitts' Y. All the volunteers in Group B received training with assistance. The subjects in Group A received training without assistance. When the final task after training was executed without assistance for all the subjects, Group B took less time to perform the same task except for in the Fitts' Z Task. This is because the subjects in Group B trained with assistance, improved their speed and accuracy in less time compared to Group A. The exception in the Fitts' Z direction can be explained by the visual perception error. The Phantom Z direction is visually mapped as the X axis on the haptic screen.

The next analysis involves the comparison of average Delta (baseline-final) execution time for tasks performed with assistance by Group A to the Fitts' Tasks performed by Group B without assistance. This is a very interesting data analysis because we are going to compare the performance of Group A with assistance after training versus performance of Group B without assistance after training.

Here, the Delta time is defined as:

- Delta time (Group A) = Average of Baseline Time Value (with Assistance) Final (with Assistance) for Fitts' X, Y, Z Each.
- Delta time (Group B) = Average of Baseline Time Value (without Assistance) - Final (without Assistance) for Fitts' X, Y, Z Each.

From Table 8.4, the average Delta time values (with assistance) of all the three (A/B) ratios are tabulated below:

Table 8.4**:** Comparison of Average Delta Execution Time when Group A Performed (WA)
and Group B (WOA)

Group A(WA)			Group B(WOA)		
Fitts' X	Fitts' Y	Fitts' Z	Fitts' X	Fitts' Y	Fitts' Z
0.300709	-0.04687	0.97685	0.651908	0.856975	1.151167

The following is a histogram representation of the above data,

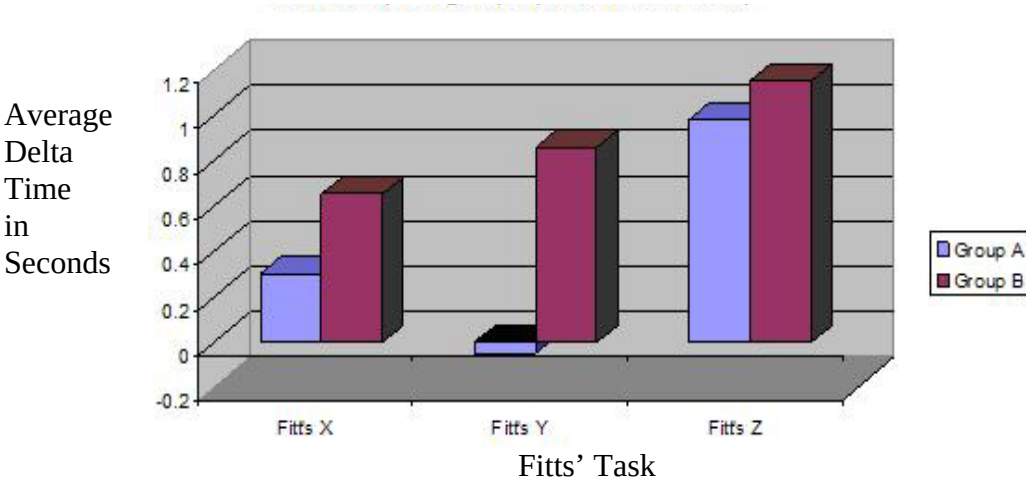


Figure 8.2: Comparison of Average Delta (Baseline-Final) Execution Time when Performed with Assistance

The volunteers in Group B took less time to perform the Fitts' Task without assistance at the end of their training. It is reminded here that the training session involved inclusion of assistance function. The volunteers in Group A meanwhile, were given assistance once before the training and once after the training. Here again, they received training without assistance. Now from the above histogram chart, we learn that the Group B members when given no assistance perform better than Group A (given assistance) at the end of their respective training sessions.

The performance of Group B subjects improves in a way that even when you remove assistance to them at the end of their training, they perform better than Group A when given assistance after training. Though the results are preliminary, this data trend shows that repetitive task helps the subjects to develop speed, which may lead to improved muscle memory sooner than expected. The only exception was found in Fitts'

Task Z where the average Delta time values were almost equal to each other. This could be attributed to the visual perception error discussed previously.

Next we compare the average Delta time values at every (A/B) for Fitts' X, Y and Z. The analysis of performance in Group A and Group B at every (A/B) ratio with assistance in Fitts' X, Y and Z is described here:

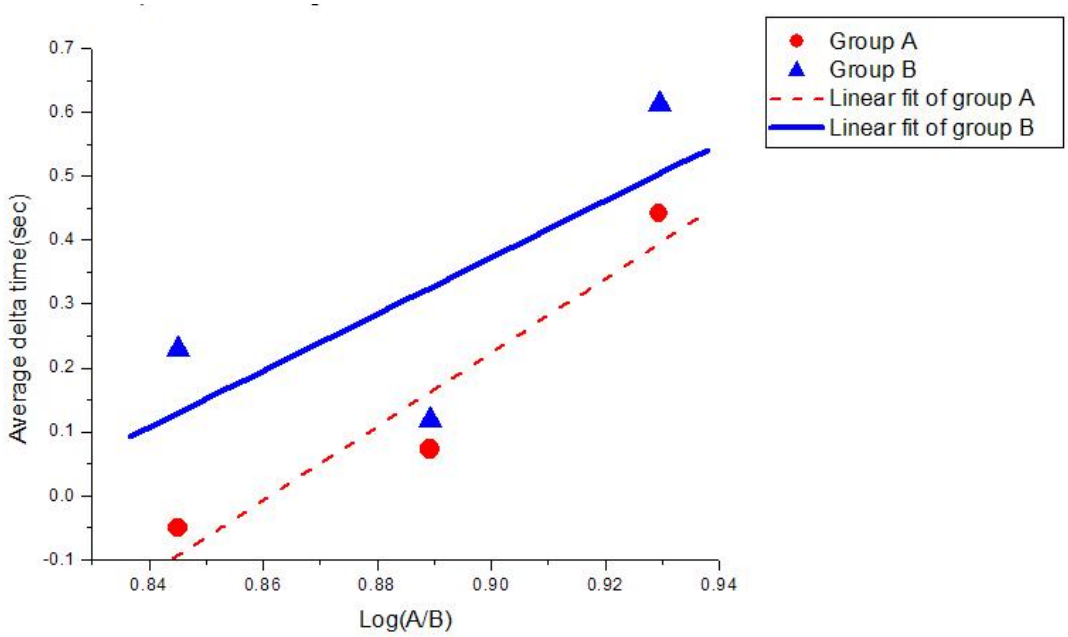


Figure 8.3: Comparison of Average Delta Time with Assistance in Fitts' X

As shown in Figure 8.3, the average Delta time increases for small (A/B) ratio. This Delta time value is the difference between the baseline and final time taken to execute the Fitts' Task with assistance. Now, the speed reduces in shorter distances because the subject tries not to overshoot the target location and hence moves slowly. Group B who received training with assistance displayed greater average Delta time at smaller (A/B) points indicating that they travel faster compared to Group A. Overall,

Group B displays a greater Delta time value compared to Group A for all the three Log(A/B) ratios. The slope and intercept values for the above experiment were -3.61578 and 4.4.3138 respectively. The coefficient of significance for the above task was 0.48572. Figure 8.3 is a sample graph in Fitts' X and the Fitts' graphs in Y and Z directions are given in Appendix B.

The following graphs display useful information with regards to the effectiveness of Fitts' Task over time in a wide range of (A/B) ratios. The average Delta time taken by Group A (baseline-final) with assistance is compared with average Delta time taken by Group B (baseline-final) without assistance for all three (A/B) ratios. The Group B subjects performed better at smaller (A/B) ratios compared to Group A.

In Fitts' Y, the Group B subjects performed better at larger values of (A/B). The lesser increase in average Delta time for smaller values of (A/B) could be attributed to the error in human accuracy. Here is the performance of subjects in Group B.

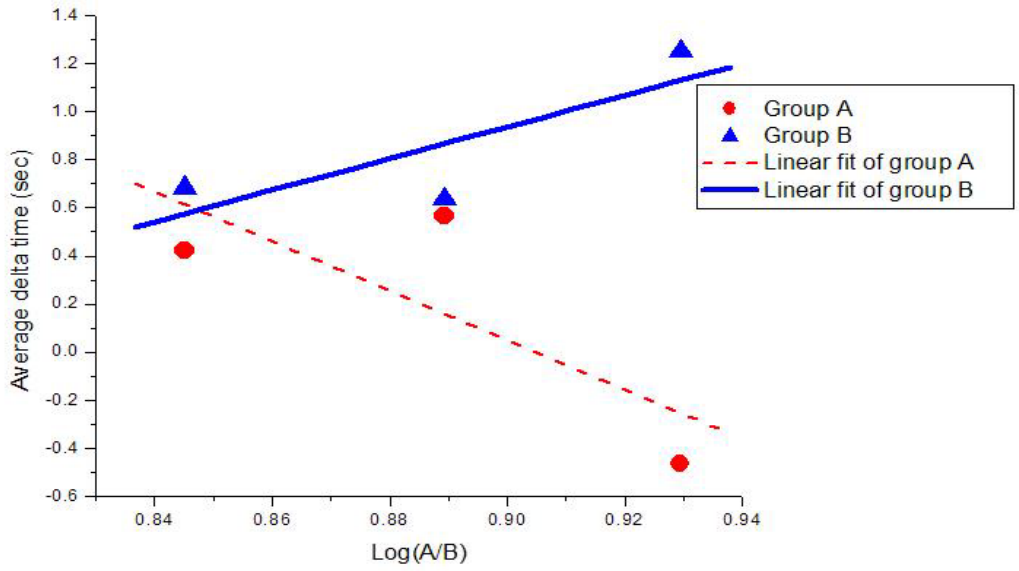


Figure 8.4: Comparison of Average Delta (Baseline–Final) Time between Group A (with Assistance) and Group B (without Assistance) in Fitts' Y

All these results indicate that, the subjects in Group B who received training with assistance performed better when the assistance was removed at the final session. They displayed greater average Delta time or shorter execution time as compared to Group A whose subjects received assistance at the end of their training session. The Group A members received training without any assistance during their training session.

The intercept and slope parameters are computed for Fitts' X, Y and Z experiments from the observed execution times. The observed execution times are fit to straight line in order to get the regression parameters. Now using these constants, the theoretical time values are calculated. The observed and theoretical time values were found to be almost equal in magnitude. Then, the theoretical time values were plotted and fit to get the experimental constants. The error percentage in α, β for observed versus the theoretical values was found to fall within 20-30%.

8.2 Determination of Position Accuracy

The position accuracy experiment involves the tracing of the haptic position of the user with the user defined trajectory points to check how he or she followed the path between the two goal points.

8.2.1 Comparison of Fitts' with and without Assistance or without Time Constant

This experiment involved two healthy subjects who were asked to move from D1 to D2 in the Fitts' Task X, Y and Z respectively. Let S1 denote the subject who was not given assistive force during task execution and let S2 denote the subject who was given assistive force during execution of each of the above tasks. Each of these subjects were not told if they were given assistance or not until they actually performed the task. Both S1 and S2 received rehabilitation training with a total of 63 trials each. While S1 received training without assistance, S2 received training with assistance. The haptic position data for both the subjects was recorded in a text file in the C++ application. This data was compared to the corresponding position data points on the trajectory implemented in the Fitts' Task. The haptic real time data position values are compared to the trajectory path values for both the subjects. Subject S2 was given assistance to perform the experiment in Fitts' X, Y and Z directions whereas subject S1 was not given any assistance to perform the same set of experiments. The real time haptic position and trajectory path value comparisons for the Fitts' Task in all three directions are shown below graphically. First, let's see the performance of S1 and S2 in Fitts' X.

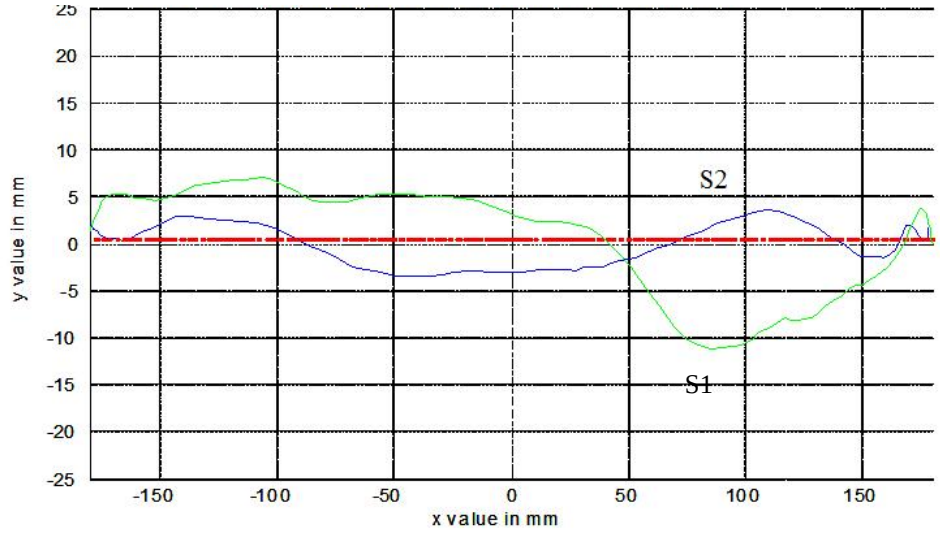


Figure 8.5: Trajectory Path Position Values Versus Haptic Real Time Data Positions in Fitts' X

The trajectory path in the Fitts' X Task is represented by the red dotted lines as shown in the Figure 8.5. While the green line represents the movement of the haptic cursor controlled by subject S1, the blue line represents the position values of the haptic cursor when controlled by subject S2. As seen in the above graph, subject S2 has followed the trajectory path more closely than subject S1. S2 performed with a standard deviation of only 1.008 while S1 showed a higher standard deviation of 1.462. The scaled forces acting in the X direction of motion accelerate the user motion on the desired direction of motion. Exponential forces based on the distance between the starting position and target is applied on the user controlled sphere. The scaling factor is small at the ends of the trajectory compared to those applied at the centre of the trajectory. This is done so that no kind of buzzing noise is generated. Buzzing may be caused by high magnitude forces acting in small distance segments.

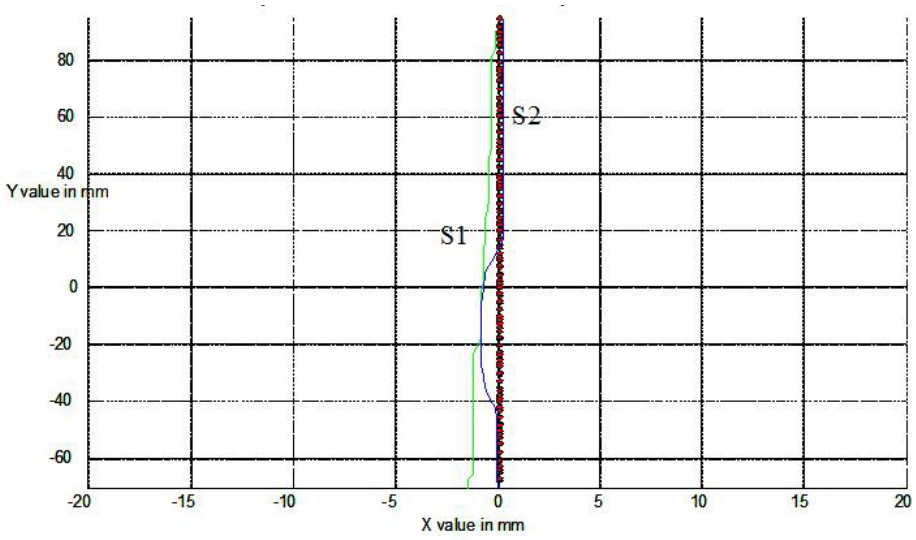


Figure 8.6: Trajectory Path Position Versus Haptic Real Time Data Positions in Fitts' Y

The Fitts' Task in Y direction was comparatively easier for both the subjects. Even then, subject S2 followed the trajectory path in Y direction more closely when compared to subject S1 as shown in Figure 8.6. The red line represents the original trajectory path with the blue line and green line representing subject S2 and S1 respectively. S2 traced the trajectory path with a standard deviation of 0.187, and S1 displayed a standard deviation of 1.468.

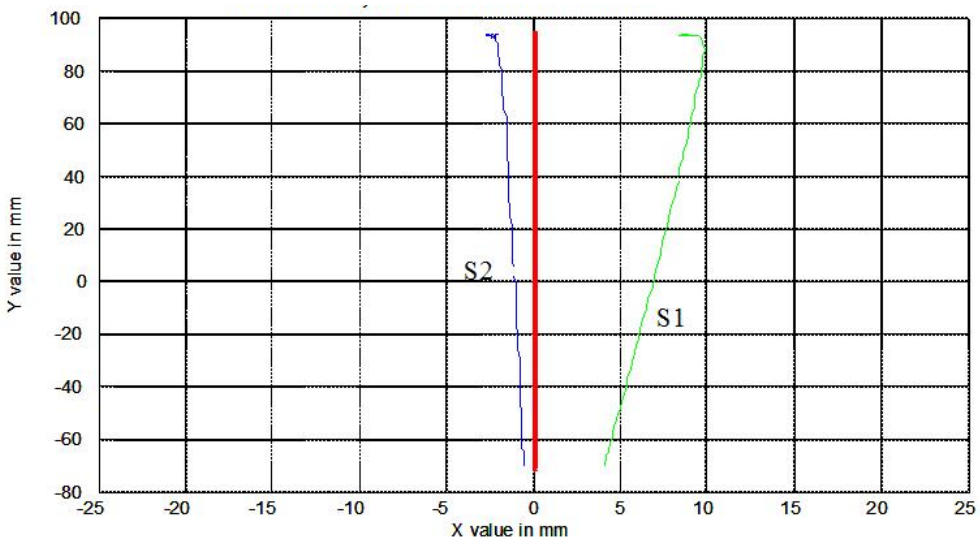


Figure 8.7: Trajectory Path Position Versus Haptic Real Time Data Positions in Fitts' Z

The Fitts' Task in the Z direction of motion was observed to be the most difficult task for all the volunteers. While subject S1 found it very difficult to move in the preset trajectory path in the Z direction without any assistance, subject S2 was guided along the entire trajectory path by the applied force algorithm effectively. This can be very clearly seen in Figure 8.7. While S1 showed a standard deviation of 18.88, S2 traced the trajectory path with a standard deviation of 4.52. The physical limitations of the Phantom here in the Z direction are effectively overcome by the forces generated in the Z direction.

These results show that the C++ application provides an effective force feedback for the different assistance tasks. This shows the system's reliability in terms of the number of times the force feedback has been provided. Moreover, the real-time aspect of the system allows the user's to get immediate force feedback. Moreover, it is seen that the logic of assistance functions in X, Y and Z directions works fine.

8.2.2 Comparison of Accuracy (without Assistance) with Execution Time Kept Constant

In this experiment, a comparison is made between the same two subjects S1 and S2 from Group A and Group B respectively. These are the same groups that we gave training in Fitts' Task previously. These two subjects S1 and S2 were given more training because the performance in Fitts' Task was said to improve over time. The more the number of trials, the better the learning curve while performing Fitts' Task. Hence, we are going to look at the position accuracy of S1 and S2. It is reminded here that, Group A received training without assistance and Group B received training with assistance.

8.2.2.1 Experimental Procedure

Both S1 and S2 had already completed 63 trials each. Now, they agreed to come again for two more sessions to perform 126 trials more. They completed their trials over three days. After their training sessions each, both subjects S1 and S2 had acquired training skills in Fitts' X, Y and Z according to their training type. At the end of training, S1 and S2 were made to perform the Fitts' X, Y and Z in three different distance ranges. The only difference in the experiment procedure was that, time was kept constant. The subjects were asked to move as much as they could in Fitts' X, Y and Z for two seconds. Both the subjects were not given any assistance during these experiments. Their haptic position data in X, Y and Z were recorded and plotted against the trajectory data in Fitts' X, Y and Z respectively. Here is the set of graphs representing the position accuracy of

S2 while performing the Fitts' in X direction without assistance. The subject S2 had to move without assistance and cover as much

as possible in 2 seconds in three ranges of distances. The time was kept constant to compare accuracy between the subjects. As seen in Figure 8.8, 8.9 and 8.10, S2 displayed good accuracy for all the three distance ranges.

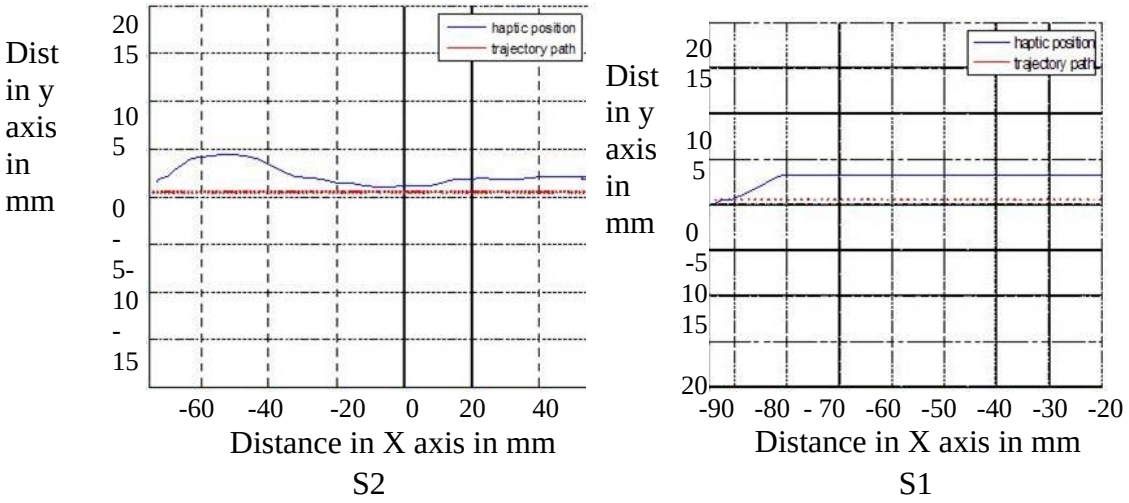


Figure 8.8 : Position Accuracy of Subject S2 and S1 in Fitts' X, Smaller Distance

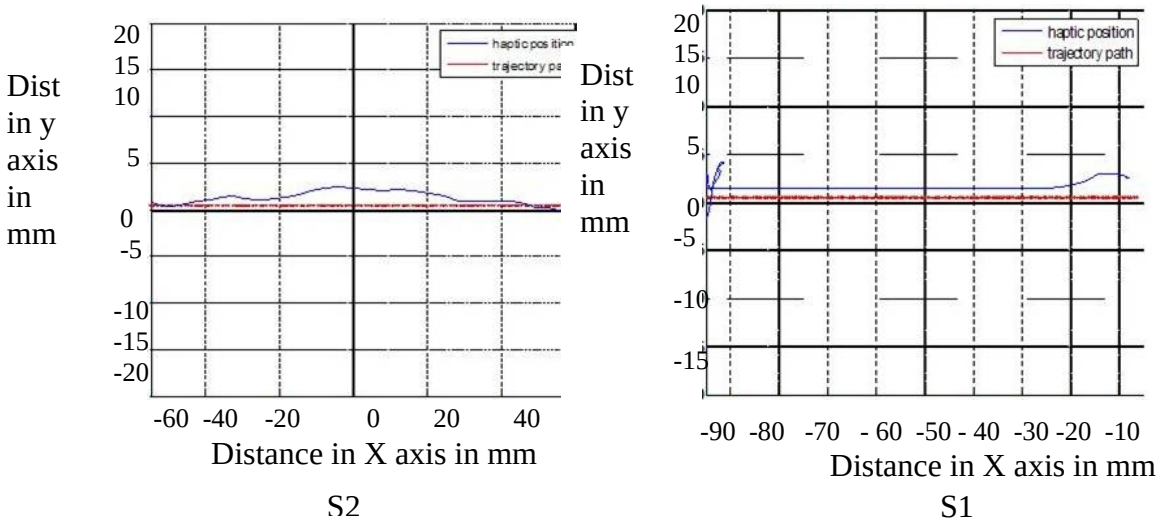
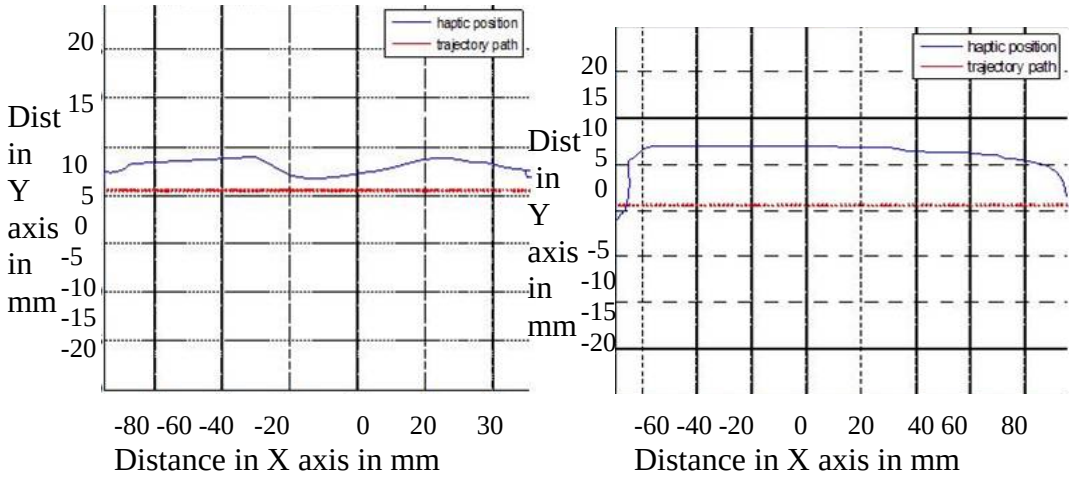


Figure 8.9: Position Accuracy of Subject S2 and S1 in Fitts' X, Medium Distance



S2 S1

Figure 8.10 : Position Accuracy of Subject S2 and S1 in Fitts' X, Larger Distance

In Figure 8.8, while S2 showed a standard deviation value of 1.008, S1 had a standard deviation of 4.007. In Figure 8.9 it is seen that, for medium distance S2 had a standard deviation of 0.97666 while S1 had a standard deviation of 2.005. In Figure 8.10, at a larger distance, S2 travels with a standard deviation of 1.463 whereas S1 showed a greater deviation of 29.9. When we compare the graphs for S1 and S2, it becomes obvious that, accuracy of S2 was as good as and in fact, better in the third distance, when compared to S1. This is attributed by the fact that S2 traveled greater distance when compared to S1 in two seconds. In Figure 8.8, S1 shows less accuracy because the subject tries to cover a greater distance in two seconds. This causes S1 to travel faster compromising on accuracy. Subject S2 performed with better accuracy in this distance.This shows that S2 improved with training session with Fitts' compared to S1 who did not receive any training with Fitts'. The Fitts' improves the performance of the user on a longer period of time.

The subject S2 traveled with a standard deviation of 1.0562 whereas subject S1 showed a standard deviation of 0.2289. In Figure 8.9, the standard deviations are less for S1 and S2 indicating that both performed better at a medium distance. The standard deviation values were 1.056 and 0.187 for S2 and S1 respectively. Here subject S1 showed better accuracy compared to S2 but covered less distance in the same time as compared to S2. At a larger distance, S2 displayed a standard deviation of 1.4653 with S1 moving much more away from the trajectory showed a higher standard deviation value of 16.79. Subject S2 obviously covered greater distance in two seconds and also showed accuracy almost as equal to S1. S2 displayed better accuracy in the larger distance range as shown in Figure 8.10 and decent accuracy for the other two distances. The results were similar for Fitts' Y and Z experiments. The corresponding graphs are given in Appendix B.

8.2.3 Position Accuracy Before and After Training for S1 and S2

The haptic position data for both S1 and S2 was recorded both before and after the training session. The subjects were not provided any kind of assistance during the recording of this haptic position. The haptic position data before the training session is the position data corresponding to the baseline time value when measured without any assistance. The haptic position data after the training session is the position data corresponding to the final task time value also measured without any assistance.

Here is a comparison of the haptic position data of S1 before and after training plotted against the trajectory data points in Fitts' X, Fitts' Y and Fitts' Z respectively.

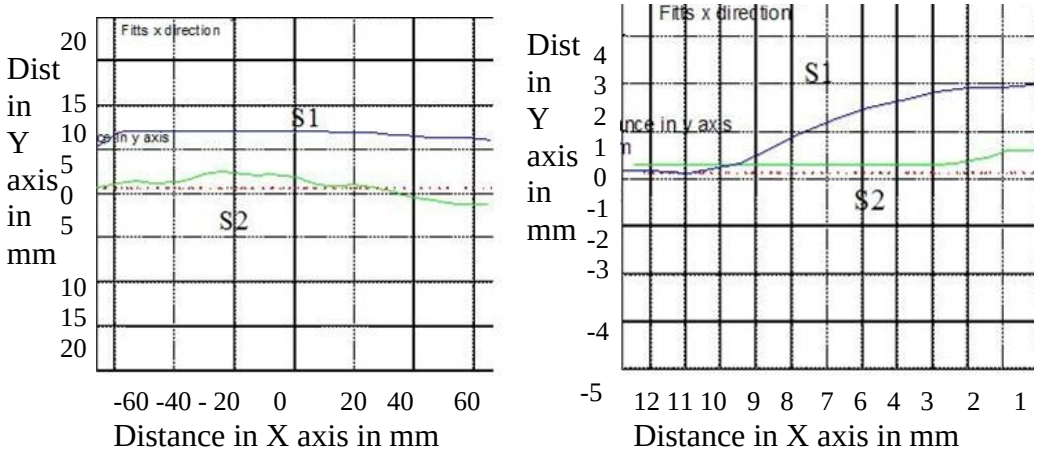


Figure 8.11: Haptic Position Data of Subject S2 and S1 before and after Training in Fitts' X

While the green line represents the haptic position data after training, the blue line denotes the haptic position data before training. As evident in the Figure 8.11, the subject S1 who received training without assistance did perform better after the training. The subject S2 traveled closer to the trajectory path compared to S1 after training. S1 showed a standard deviation of 4.008 which reduced to 2.003 after training. S2 whereas, improved its standard deviation from 3.521 to 1.09. This means that subject S2 traveled almost close to the trajectory path compared to his counterpart S1 after their respective training sessions. This is because subject S2 moved with greater accuracy after the training with assistance compared to subject S1 who received training without assistance. The next set of graphs depicts the performance of S1 and S2 in Fitts' Y Task.

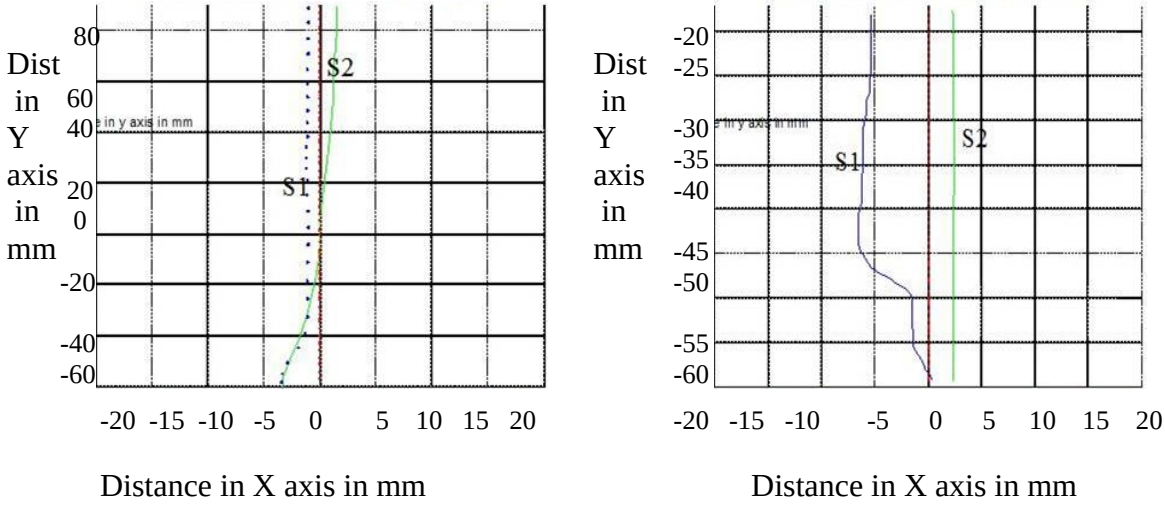


Figure 8.12: Haptic Position Data of Subject S2 and S1 before and after Training in Fitts' Y

In Figure 8.12, the green line represents the movement after the training sessions and the blue one represents the movement before any training session and without any assistance. Let us take a look at the performance of subject S2 and compare to S1 and S2. As shown in Figure 8.12, both the subjects' showed similar improvement both before and after performance. But, it is observed that, subject S2 still showed a tendency to move a accurately on the trajectory path when compared to subject S1. While the standard deviation of S2 improved from 1.056 to 0.187, S1 showed an improvement in standard deviation from 16.79 to 1.553 after training. Well, both S1 and S2 performed better after training here compared to Fitts' X, which may be due to the ease with which the Phantom imitates motion in the Y direction.

The Fitts' Z is one of the most difficult tasks and it would be interesting to see if subject S2 performed more accurately than S1. Here are the graphs for S1 and S2 recorded during the Fitts' Z Task.

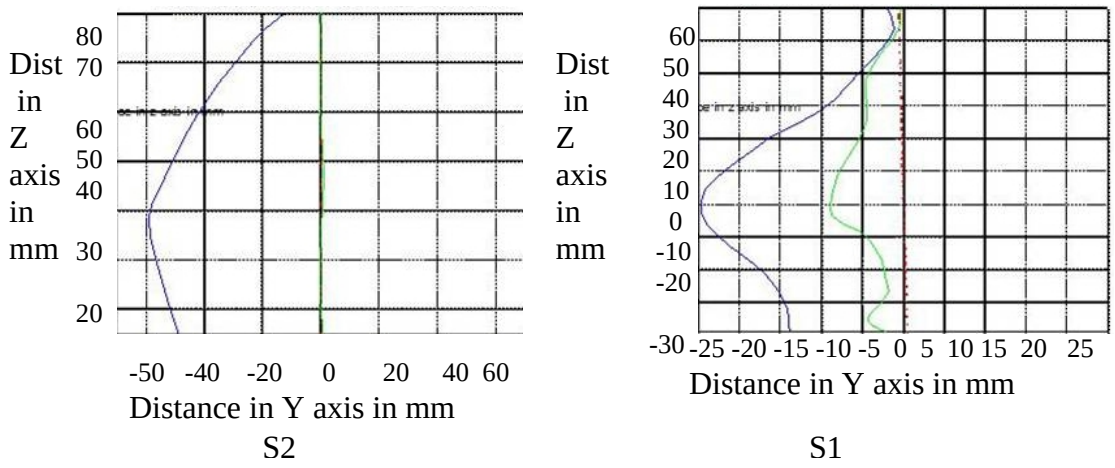


Figure 8.13: Haptic Position Data of Subject S2 and S1 before and after Training in Fitts' Z

While the green line denotes the haptic position data for the subject after training, the blue line presents the haptic position data before the training. Subject S1 did better as seen in Figure 8.13 but was not as accurate as S2 in path following. While S2 showed a standard deviation of 16.8 before training, the standard deviation value improved to 1.778 after training. Meanwhile, S1 showed a standard deviation of 18.88 before training and 6.99 after training. Hence, while position accuracy of S1 improved after training in the Z direction, it was not as good as the position tracking of S2. These graphs show the different ways to compare and analyze data. Hence this system provides an ideal platform for data analysis for any kind of haptic based rehabilitation tasks which could be added on later. Also, though these data are preliminary they show the potential of this system as a therapy tool.

8.2.3.1 Standard Deviation of S1 and S2

It is reminded that subjects S1 and S2 who had performed position accuracy experiments had performed a total of 189 trials each. Excluding the baseline and final experiment, both S1 and S2 performed a total of 27 trials of training each. This means S1 performed 9 trials without assistance in Fitts' X, Y and Z each, while S2 performed 9 trials with assistance in Fitts' X, Y and Z directions each. The average time value of these 9 trials was determined and the standard deviation of the sample set was determined for Fitts' X, Y and Z. The values are tabulated below.

Table 8.5: Standard Deviation of Subject S2 who Received Training with Assistance

	Fitts' X	Fitts' Y	Fitts' Z
	2.9621	4.1877	5.8033
	4.267	4.398	10.14533
	4.4087	4.032	9.0957
Standard Deviation	0.79746	0.1837	2.2655

Table 8.6: Standard Deviation of Subject S1 who Received Training without Assistance

	Fitts' X	Fitts' Y	Fitts' Z
	2.9621	4.1877	5.8033
	4.267	4.398	10.14533
	4.4087	4.032	9.0957
Standard Deviation	0.79746	0.1837	2.2655

The standard deviation values are low enough to show that the subjects were consistent for all the trials that they performed. On a final note, task oriented repetitive experiments may help in improving muscle skill and movement coordination [22] and this project shows that task execution with assistive forces may help people improve their speed and accuracy in task execution in a shorter period of time as compared to those who received training without assistance. The preliminary results help for future analysis in clinical studies.

9. Conclusions and Future Work

9.1 Conclusions

The following are indications of the data trend observed from performing the predefined preliminary experiments. All these inferences basically validate the future use of this haptic integrated C++ application for rehabilitation purposes. Here is a summary of the inferences:

- Group B took less execution time compared to Group A subjects in Fitts' X, Fitts' Y and Fitts' Z Tasks (Delta T was computed for without assistance tasks).
- Group B (without assistance) Delta T was greater than Group A (with assistance) Delta T. Group B learnt the task faster than Group A subjects.
- The position accuracy of Group B subject S2, improved considerably after training and it was as good and in some cases, better than the improvement in Group A subject S1 after training.
- The Group B subject S2, not only improved in position accuracy but also improved in speed maintaining his or her accuracy. The Group A subject S1 showed good improvement in position accuracy but did not improve in speed.

- Improved user performance in smaller distances: Higher Delta T at smaller distances in Fitts' X.
- Position accuracy in Fitts' Z improved after training for subject S2.

These inferences provide a platform for future research for making this application a full fledged rehabilitation tool. These results indicate a positive response with respect to the effectiveness of the assistance functions and the C++ code. The system developed here also provides a good data collection tool for clinicians.

9.2 Future Work

The haptics based assistive technology provides an amazing scope in haptic teleoperation and for use in rehabilitative applications. Here, is a list of suggested future work based on the research work presented here.

9.2.1 3D Simulation of Pre-Set Activities of Daily Living Tasks

The simulated ADL tasks help people with hand disabilities in the following ways:

- To Assist in any ADL task.
- To Rehabilitate the Hand as Well as the Arm.
- To Learn the Movements Involved in any Activities of Daily Living Task before Actually Executing it.

The assistance concepts in this thesis could be applied to any type of Pre-Set ADL tasks. One very good example is a Handwriting Expert software tool. This tool should contain a Graphical User Interface consisting of all the alphabets from A-Z. The OpenGL

frame work could be used to simulate a path which traces the chosen alphabet. Then, appropriate forces could be applied to the path so that the user is assisted while tracing the path of the chosen alphabet. The user hence, learns to move the hand in the shape of an alphabet and thus exercises the hand muscles required to write an alphabet. Thus, electronic handwriting combined with assistive technology would enable users with hand disabilities write coherently. Another good example is the maze, whose path consists of twists and turns. The assistance concepts would be applied to a curvilinear path as against to a straight line path which is described in this thesis. The user receives assistance in tracing the non-linear path of the maze and to finally reach the goal at the end of the maze. The Fitts' Task could also be used to test the accuracy of the above two experiments. The Fitts' Law was applicable only in translational movements but studies [40] have shown that this law could be extended to describe angular motion. Hence, be it a linear or angular motion in a maze, it is going to be possible to verify the accuracy of experiments set up in these non linear paths using the modified Fitts' Law. It would be interesting to design the peg in hole experiment in 3D virtual space like the Fitts' task. The experiment consists of a cylindrical peg which could be moved alternately between the two holes [41]. The time taken by the user to insert a peg would be a reliable measure of the subject's skill and performance. The assistive concepts described in this project would definitely reduce the time taken for this task when applied properly.

9.2.2 Addition to this Research Work

Now, we are going to take a look at the various suggestions that can improve this research work. We validate the system performance, in terms of effective force feedback, real time application and choice in terms of assistance functions. The experimental data are still preliminary and here is a look at the various steps that may lead to proper clinical testing using this haptic integrated C++ application. This project only shows a data trend, and further work is required to conclude on the actual benefit of using this application for people with disabilities. This is a set of preliminary experiments and it requires more experiments to actually get conclusive data regarding human performance. Currently, this application is treated as an indication of the effect of assistance functions on normal healthy users. The experiments must be designed in a way such that there are more trials and practice sessions than the current number of trials. This should be done to ensure that enough practice is given before any conclusive evidence is made on user performance. Moreover, we should have better linear regression Fitts' with acceptable significance values.

9.2.3 Robotic Teleoperation

Now, haptics also plays a very significant role in force feedback in robotic teleoperation. The Phantom provides a user friendly interface device in robotic teleoperation. They generate force feedback based on the distance between the robotic end-effector and the target. The GUI and C++ haptic interface tool can be applied to control a simulation of any degree of freedom robotic arm (simulation or real world) and

provide different types of assistive forces. These assistive concepts could be based on the type of task involved during teleoperation. Thus, this project offers high scope for robotic teleoperation and future upper arm rehabilitative applications. A good example of robotic teleoperation is the control of a simulation of robotic arm called PUMA 560. This is a six degree of freedom robotic arm and it has been used previously to help surgical operations. It would be interesting to provide force assistance for this robotic arm while performing surgical procedures.

References

[1] American Academy of Neurology, 1999, “Robots Improve Movement in Stroke Patients, ” ScienceDaily. Retrieved October 31, 2007, from http://www.sciencedaily.com /releases/1999/11/991112064413.html.

[2] Bamford, J. M., Sandercock, P., and Dennis, M., 1988, “A Prospective Study of Acute Cerebrovascular Disease in the Community: The Oxfordshire Community Stroke Project 1981-1986: Methodology, Demography, and Incident Cases of First Ever Stroke,” J Neurol Neurosurg Psychiatry, pp. 1373-1380.

[3] Zesiewicz, T. A., and Hauser, 2001, “Phenomenology and Treatment of Tremor Disorders,” Neurol Clin 2001, pp. 651-680.

[4] Smaga, S., M. D., 2003, ”Tremor,” American Family Physician, PVP-Vol. 68, Number 8.

[5] Keates, S., Langdon, P., Clarkson, J., and Robinson, P., 2000, “Investigating the Use of Force Feedback for Motion–Impaired Users,” 6th ERCIM Workshop, CNR-IROE, Italy, pp. 25-26.

[6] Zhai, Buxton, W., 1996, ”The Influence of Muscle Groups on Performance of Multiple-Degree-of-Freedom,” Input. In: Proceedings of CHI ‘96(Vancouver, Canada), Addison Wesley, pp. 308-315.

[7] Fitts’, P. M., 1954, “The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement,” Journal of Experimental Psychology, pp. 381-391.

[8] Pernalete, N., Yu, W., Dubey, R., and Moreno, W., 2002, ” Development of a Robotic Haptic Interface to Assist the Performance of Vocational Tasks by People with Disabilities,” Proceedings of the IEEE.

[9] Arsenault, R., and Ware, C., 2000, “Eye-Hand Co-ordination with Force Feedback,” CHI Letters, PVP- Vol. 2, Issue 1, pp. 1-6.

[10] Morris, D., Hong, T., Barbagli, F., Chang, T., and Salisbury, K.," Haptic Feedback Enhances Force Skill Learning," Department of Computer Science, Stanford University, Haptic Interface Research Laboratory, Purdue University.

[11].Otaduy, A., and Lin, C.,"Introduction to Haptic Rendering," Department of Computer Science, University of North Carolina at Chapel Hill.

[12] Yu., W., Dubey, R., and Pernalete, N., 2004, "Telemanipulation Enhancement through User's Motion Intention Recognition and Fixture Assistance," International Conference on Intelligent Robots and Systems, PVP- Vol. 3, pp. 2235-2240.

[13] Lovquist, E., and Dreifaldt, U., 2006, "The Design of a Haptic Exercise for Post Stroke Arm Rehabilitation," Proceedings of the Sixth International Conference of Disability, Virtual Reality and Assoc Tech, Denmark.

[14] Popescu1, V., Burdea1, G., Bouzit, M., Girone1, M., and Hentz, V., " PC-based Telerehabilitation System with Force Feedback," ECE department, Rutgers University, Division of Hand and Upper Extremity Surgery, Stanford University Medical Center.

[15] Leifer, L., 1981, " Rehabilitation Robots," Robotics Age, pp. 4 - 15.

[16] Roesler, H., Kuppers, H. J., and Schmalenbach, E., 1978, "The Medical Manipulator and Its Adapted Environment: A System for the Rehabilitation of Severely Handicapped," in IRIA Proc.Int.Conf.Telemanipulators for the Physically Handicapped, pp.73 - 77.

[17] Dallaway, J. L., and Jackson, R. D., 1992," RAID a Vocational robotic workstation," in ICORR 92-Conference Proceedings.

[18] Hogan, N., Krebs, H. I., Chamnarong, J., Srikrishna, P., and Sharon, A., 1992, " MIT-MANUS . A Workstation for Manual Therapy and Training," Neman Laboratory for Biomechanics and Human Rehabilitation , MIT, Cambridge, MA, USA, June 12.

[19] Reinkensmeyer, D. J., Kahn, L. E., Averbuch, M., McKenna-Cole, A. N., Schmit, B. D., and Rymer, S., 2000, " Understanding and Treating Arm Movement Impairment after Chronic Brain Injury: Progress with The ARM Guide," Journal of Rehabilitation Research and Development, PVP-Vol .37, pp. 653-662.

[20] Sugar, T. G., Koeneman, E. J., Herman, J. B., Schultz, R. S., Herring, D. E., Wanberg, J., Swenson, S., and Ward, P., 2007, "Design and Control of RUPERT, A Device for Robotic Upper Extremity Repetitive Therapy," IEEE Transactions on Rehabilitation Engineering, PVP-Vol 15, Issue 3, pp 336-346.

[21] Nef, T., and Reiner, R., 2005, ”ARM-in Design of a Novel Arm Rehabilitation Robot,” Proceedings of the 2005 IEEE, 9th International Conference on Rehabilitation Robotics, Chicago, Illinois, USA.
[22] Schuyler, R. M., and Mahoney, R., 1995, ”Job Identification and Analysis for Vocational Robotics Application,” Proceedings RESNA.

[23] Takahashi, Y., Terada, T., Inoue, K., Ito, Y., Lee, H., Ikeda, Y., and Komeda, T., 2003, ”Upper-Limb Rehabilitation System Using Haptic Device with Basic Motion Training Program,” Proceedings of the 25th Annual International Conference of the IEEE EMBS, Mexico, September 17-21.

[24] Kesavadas, T., and Subramanium, H., 2003, ”Development and Pilot Testing for Virtual Manufacturing Tools with Intelligent Attributes,” pp. 933-940.

[25] MacLean, K. E., 2000, “Designing with Haptic Feedback,” Robotics and Automation, Proceedings.ICRA, IEEE.

[26] Pernalete, N., Yu, W., Dubey, R., and Moreno, W., 2002, ”Development of A Robotic Haptic Interface to Assist The Performance of Vocational Tasks by People with Disabilities,” Proceedings of the 2002 IEEE, International Conference on Robotics & Automation, Washington, DC.

[27] Everett, S. E., 1998, ”Human-Machine Cooperative Telerobotics Using Uncertain Sensor or Model Data,” Proceedings of the 1998 IEEE, International Conference on Robotics and Automation, PVP-Vol. 2, Issue, pp. 1615-1622.

[28] Gentry, S., Feron, E., and Murray-Smith, R., 2005, ”Human-Human Haptic Collaboration in Cyclical Fitts’ Tasks,” Proceedings of the IEEE, International Conference on Intelligent Robot and Systems, pp. 3402-3407.

[29] Pellizer, G., and Hedges, J., 2003, ”Motor Planning: Effect of Directional Uncertainty with Discrete Spatial Cues,” Biomedical Life Sciences, Springer, PVP-Vol 150 June, pp. 276-289.

[30] Card, K., Moran, P., and Newell, A., 1980, ”The Keystroke Level Model for User Performance Time with Interactive Systems,” Communications of the ACM, PVP-Vol 23, Issue 7, pp. 396-410.

[31] Accott, J., and Zhai, S., 1997, ”Beyond Fitts’ Law: Models Based on Trajectory Based HCI Tasks,” Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, PVP-Vol. 23, Issue 7, pp. 295-302.

[32] Bootsma, R. J., Fernandez, L., and Mottet, D., 2004, ”Behind Fitts’ Law: Kinematic Patterns in Goal Directed Movements,” International Journal of Human Computer Studies, Science Direct, PVP-Vol. 61, Issue 6, pp. 811-821.

[33] Gentry, S., and Murray-Smith, R., 2005,”Human Human Haptic Collaboration in Cyclical Fitts’ Tasks,” Proceedings of the IEEE Conference on Intelligent Robot and Systems.

[34] Everett, S. E., and Dubey, R. V., 1998, ”Human Machine Cooperative Telerobotics using Uncertain Sensor and Model Data,” Proceedings of the IEEE International Conference on Robotics and Automation, PVP-Vol 2, pp. 1615-1622.

[35] Phillips, C. A., and Repperger, D. W., 1997, ”Why Engineers Should Know and Use Fitts’ Law,” Proceedings -19th International Conference, IEEE/EMBS, Oct 3, Chicago, IL.USA.

[36] Beamish, D., Bhatti, S. A., Scott, I., and MacKenzie, Wu. J., 2006, ”Fifty Years Later: A Neurodynamic Explanation of Fitts’ Law,” J.R.Soc.Interface, pp. 649-654, 18.

[37] Woodworth, R. S., 1899, “The Accuracy of Voluntary Movement,” Psychol. Rev. 3 (Monograph Supl.), pp. 1-119.

[38] Guiard, Y., and Beaudouin-Lafon, M., 2004, ”Fitts’ Law 50 Years Later: Applications and Contributions from Human-Computer Interaction,” International Journal of Human-Computer Studies, PVP-Vol 61, Issue 6, pp. 747-750.

[39] Smits-Engelsman, B. C. M., Van Galen, G. P., and Duysens, J., 2002, ”The Breakdown of Fitts’ Law in Rapid, Reciprocal Aiming Movements,” Springer-Verlag.

[40] Kondraske, G. V., 1994, “An Angular Motion Fitts’ Law for Human Performance Modeling and Prediction,” Proceedings of the 16th Annual International Conference of the IEEE, pp. 307-308.

[41] Amirabdollahian, Gomes, F., and Johnson, G. T., 2005, ”The Peg- In -Hole: A VR - Based Haptic Assessment for Quantifying Upper Limb Performance and Skills,” Rehabilitation Robotics, 9th International Conference On, pp. 422-425.

Appendices

Appendix A. Fitts' Coefficients

α, β values for Fitts' X, Fitts' Y and Fitts' Z.

Comparison of Average Delta Time with Assistance in Fitts' X Task

Group B
α = -3.61578
β = 4.43138
Coefficient of Significance: 0.48572

Group A
α = -4.9795
β = 5.78188
Coefficient of Significance: 0.19616

Comparison of Average Delta Time with Assistance in Fitts' Y Task

Group B
α = -4.35429
β = 5.32013
Coefficient of Significance: 0.41733

Group A
α = 9.29844
β = -10.2735
Coefficient of Significance: 0.43439

Comparison of Average Delta Time with Assistance in Fitts' Z Task

Group B
α = -8.80956
β = 10.79132
Coefficient of Significance: 0.19987

Appendix A (Continued)

Group A
α = -8.14036
β = 9.9823
Coefficient of Significance: 0.60341

Comparison of Average Delta (Baseline – Final) Time between Group A (with Assistance) and Group B (without Assistance) in Fitts' X

Group B
α = -3.61578
β = 4.43183
Coefficient of Significance: 0.48572

Group A
α = -4.9795
β = 5.7818
Coefficient of Significance: 0.19616

Comparison of Average Delta (Baseline – Final) Time between Group A (with Assistance) and Group B (without Assistance) in Fitts' Y

Group B
α = -4.9926
β = 6.58912
Coefficient of Significance: 0.39609

Group A
α = 9.29844
β = -10.275
Coefficient of Significance: 0.43439

Comparison of Average Delta (Baseline – Final) Time between Group A (with Assistance) and Group B (without Assistance) in Fitts' Z

Appendix A (Continued)

Group B
α = -6.26433

β = 8.35144
Coefficient of Significance: 0.39223

Group A
α = -8.14036
β = 9.9823
Coefficient of Significance: 0.60341

Appendix B. Fitts' Graphs

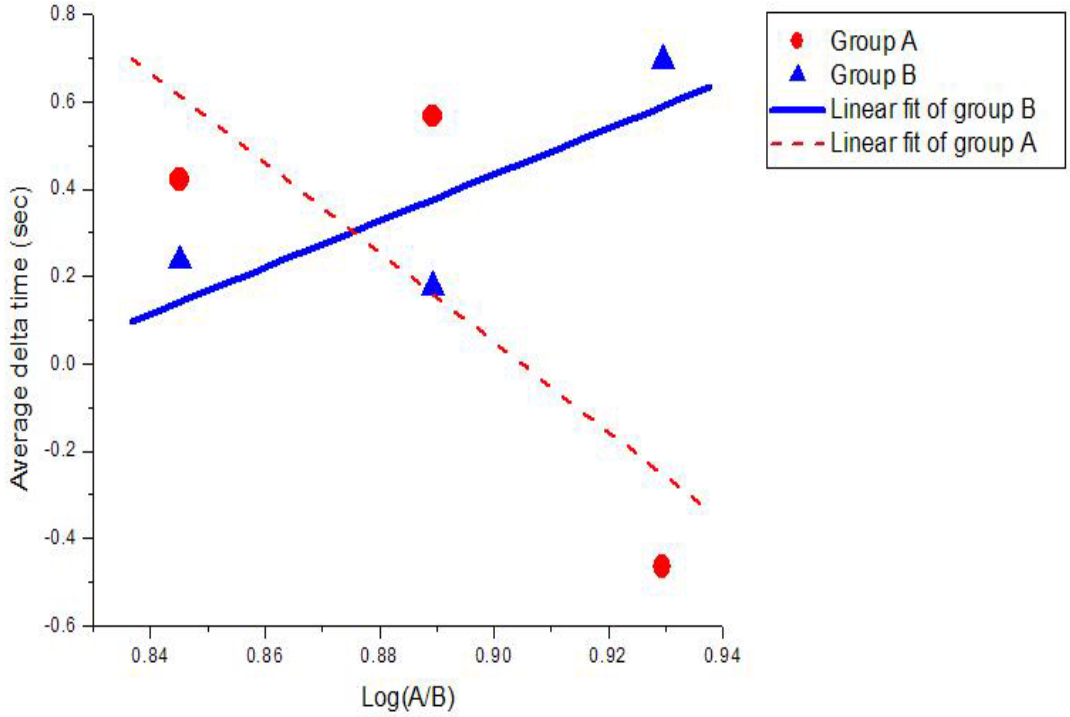


Figure B-1: Comparison of Average Delta Time with Assistance in Fitts' Y

The above graph indicates that execution time has reduced significantly for greater values of (A/B) in Fitts' Y Task. For smaller (A/B), the average Delta time for Group A was greater compared to Group B. This can be attributed to the fact that, in general, all the subjects performed well in Fitts' Y direction.

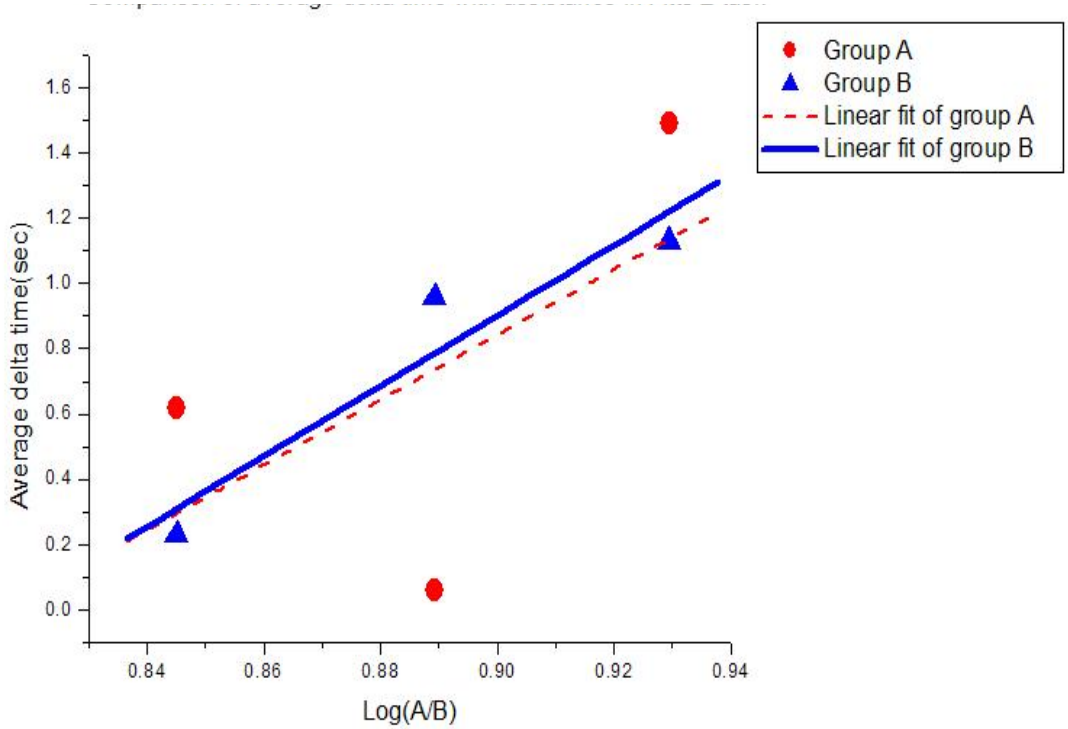


Figure B-2: Comparison of Average Delta Time with Assistance in Fitts' Z

All the subjects in Group A and Group B found the Fitts' Z Task comparatively difficult because of the previously mentioned reasons. Irrespective of this, the subjects in Group B showed greater average Delta time values compared to Group A.

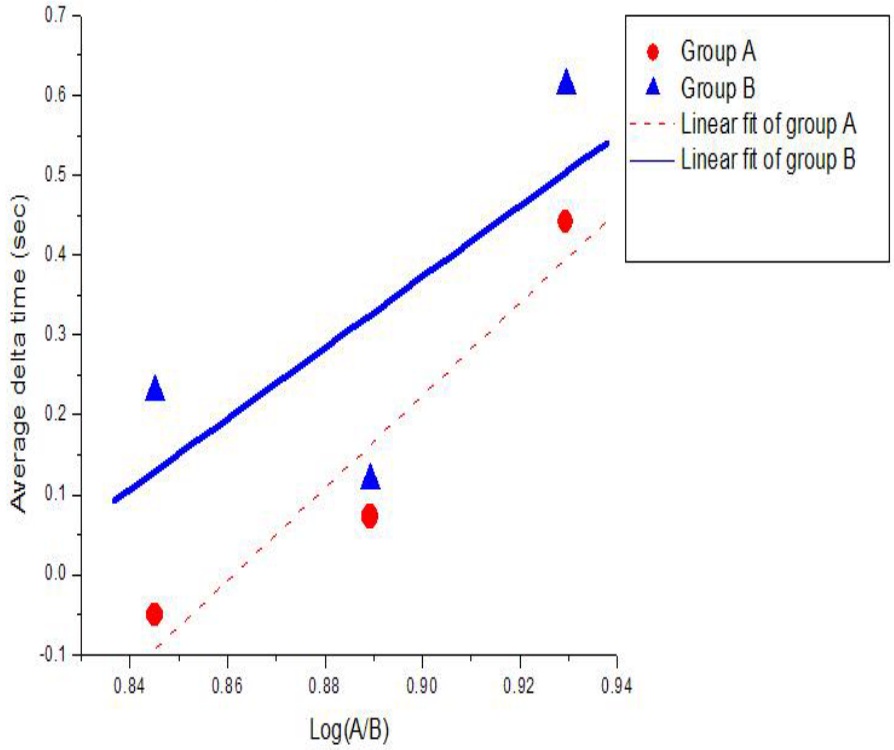


Figure B-3: Comparison of Average Delta (Baseline – Final) Time between Group A (with Assistance) and Group B (without Assistance) in Fitts' X

The Fitts' Z also showed better results at smaller values of (A/B) with group B taking lesser time compared to Group A. This can be seen in Figure B-3.

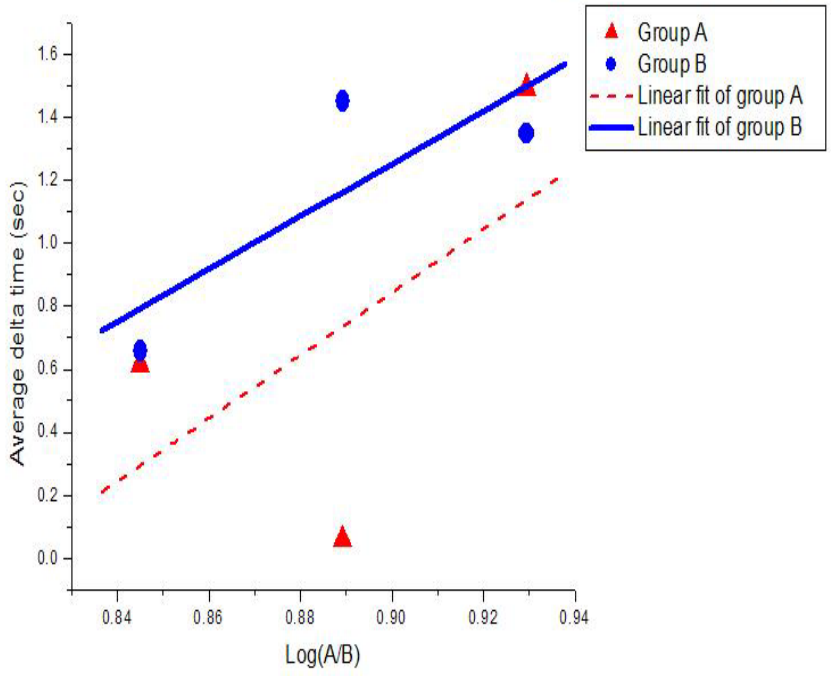
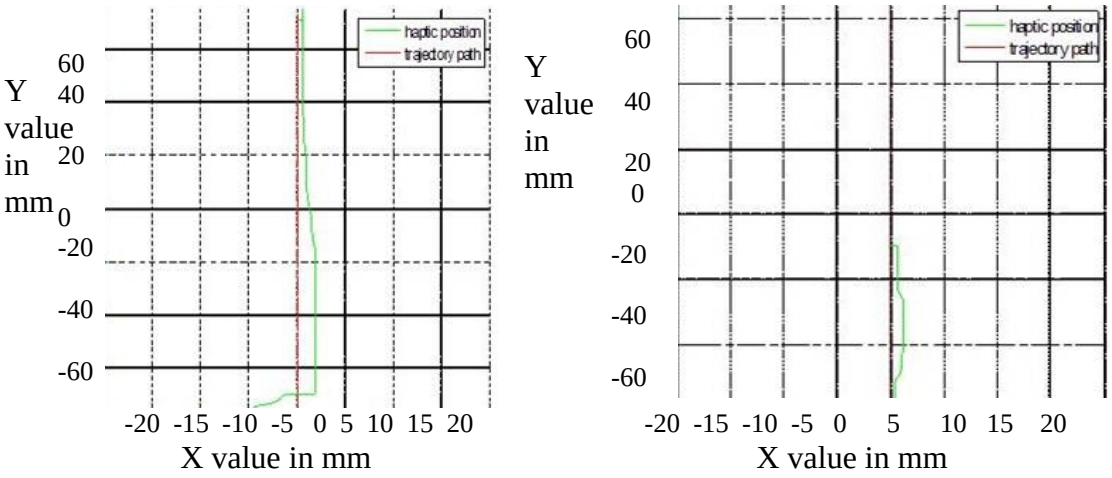


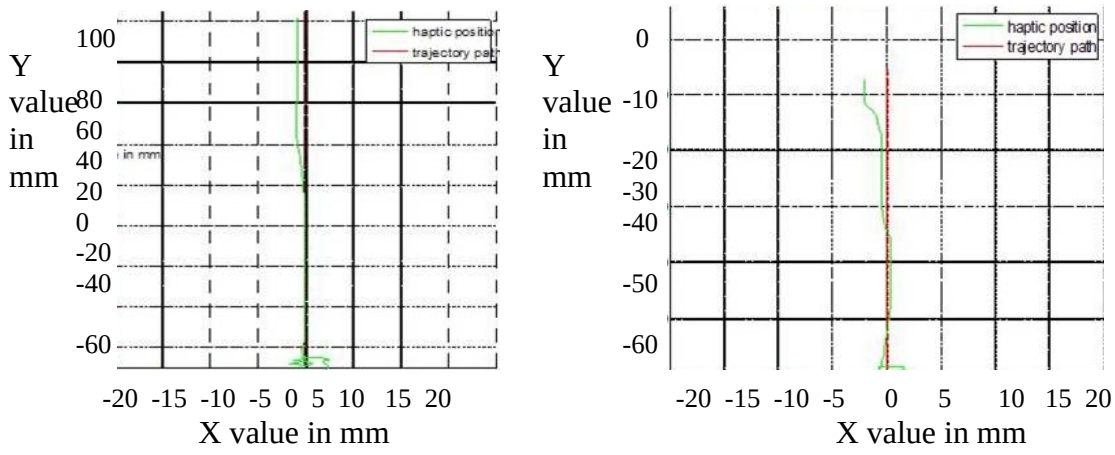
Figure B-4: Comparison of Average Delta (Baseline – Final) Time between Group A (with Assistance) and Group B (without assistance) in Fitts' Z



S2 S1

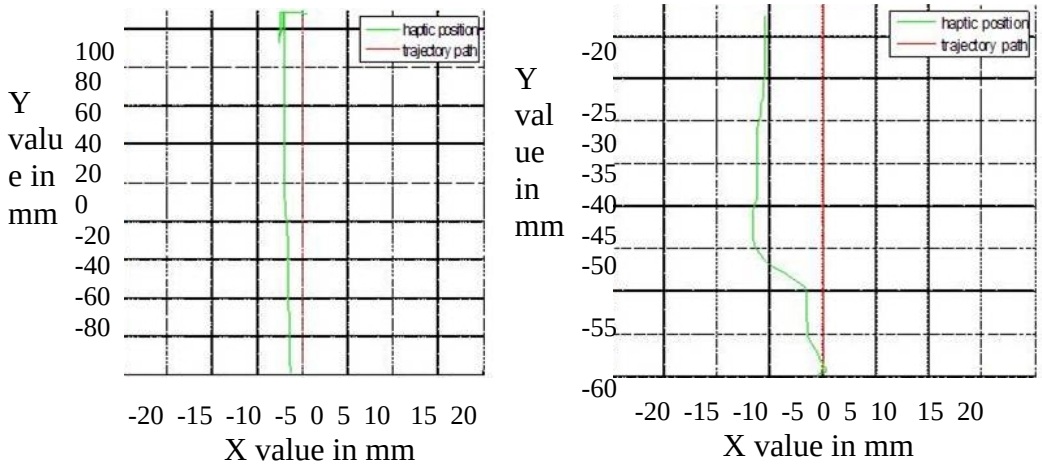
Figure B-5: Performance of S2 and S1 before and after Training

Appendix B (Continued)



S2 S1

Figure B-6: Position Accuracy of Subject S2 and S1 before and after Training in Fitts' Y, Medium Distance



S2 S1

Figure B-7: Position Accuracy of Subject S2 and S1 before and after Training in Fitts' Y, Larger Distance

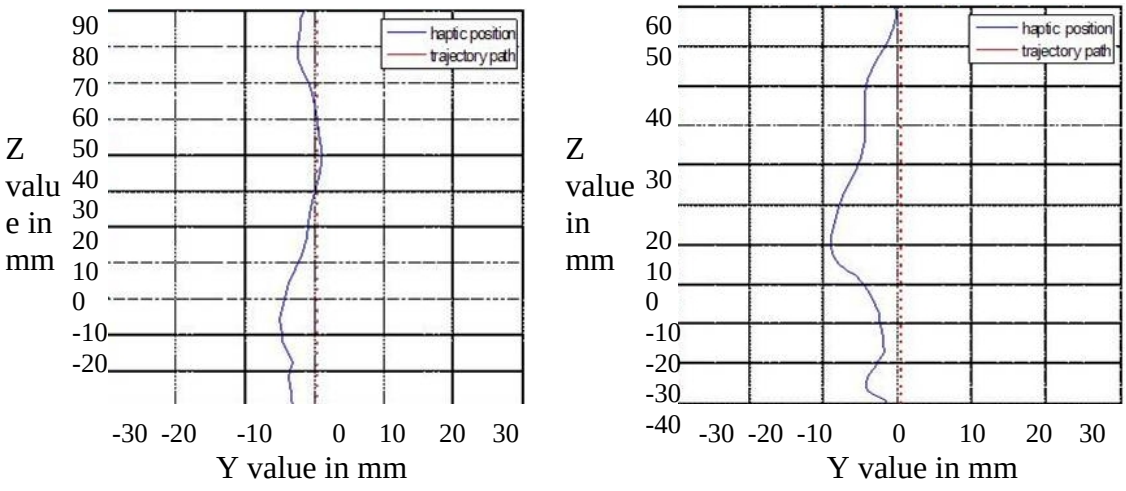


Figure B-8: Position Accuracy of Subject S2 and S1 before and after Training in Fitts' Z, Smaller Distance

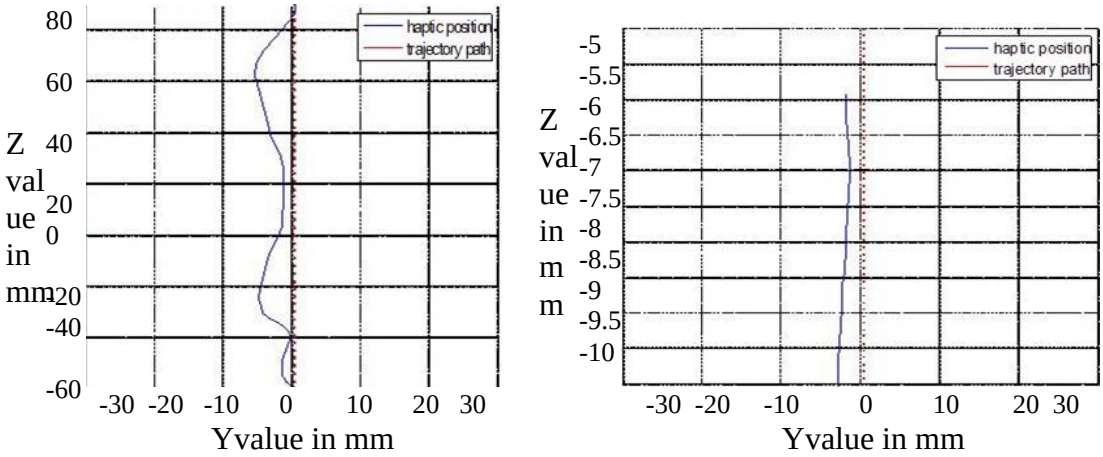


Figure B-9: Position Accuracy of Subject S2 and S1 before and after Training in Fitts' Z, Medium Distance

Appendix B (Continued)

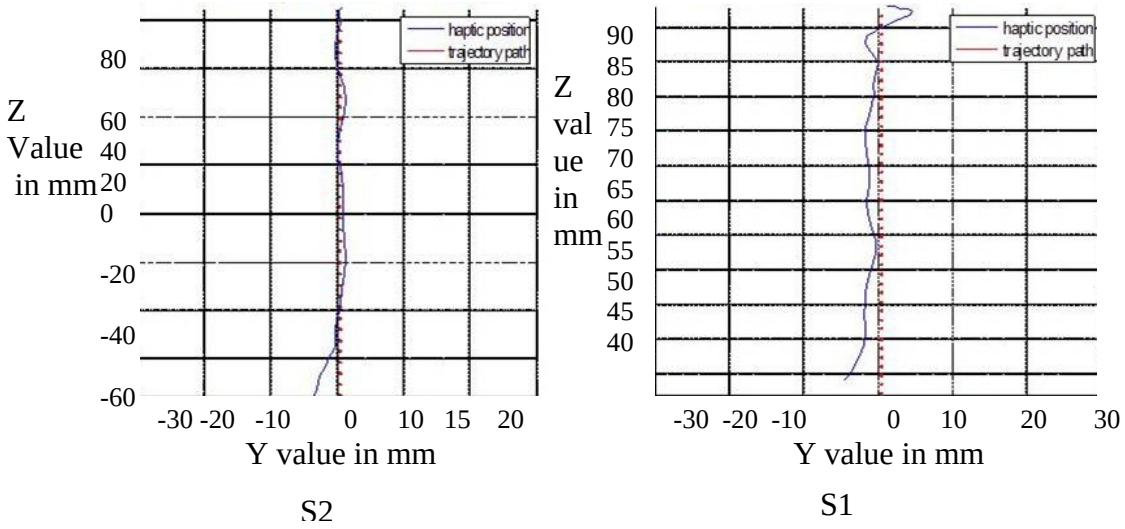


Figure B-10: Position Accuracy of Subject S2 and S1 before and after Training in Fitts' Z, Larger Distance

Appendix C. C++ Code

```
#include <HL/hl.h>
//ADDED NOW//
#include <HDU/hduMatrix.h>
//#include <dos.h>
//#include <time.h>
#include <malloc.h>
#include "ConstantsSock.h"
#include <cstdio>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include "TrajectorySock.h"
//#ifndef DataStruc.h
#include "DataStruc.h"
//#endif DataStruc.h
//#include "udpSocketClass.h"                    //socket class definition

//#include "udpSocketClass.h"
//#include "DataStruc.h"
//#define NP 50
//#define BUFFERSIZE 256
#define NP 2
#define NPT 100
#define  RECTSIZE 80
double p,q,r;
double smallest_dist=13.0;
double px;
int i,v;
int value;
/*USER CONTROLLED FORCE PROJECTION ON THE TRAJECTORY
PATH.PARAMETERS USED.Testing on exp 6*/

double Proj_Scale;
double xExp_Scale;
double yExp_Scale;

//#define FRAMECOUNTER 29

/*typedef struct point_3d {                                    //3D in world coord sys
                double x, y, z;
```

Appendix C (Continued)

```
}POINT_3D;
//POINT_3D *mpoints;*/

typedef struct point_3d {                                   //3D in world coord sys
                    double x, y, z;
                    }POINT_3D;

typedef struct TrajP {
                    double x, y, z;


                    int indx;
}TrajP_3D;

TrajP_3D *TrajPoint;
TrajP_3D ClosestPoint;
TrajP_3D Next_Pt;


POINT_3D pointstart;
POINT_3D pointend;
POINT_3D Intersection;
POINT_3D hapticposition;
POINT_3D nearest_pt;
//POINT_3D TrajPoint;
//ADDED NOW//

//ADDED NOW//
//hduVector3Dd vectorPoint;
//ADDED ON MaRCH 27TH
hduVector3Dd vectorPoint;
hduVector3Dd dcentertoeffector;
//hduVector3Dd Trajpoint;
hduVector3Dd scalepoint;
hduVector3Dd fixedcenter;
//added on march 31st
hduVector3Dd bezierpoint;
hduVector3Dd secondpoint;
hduVector3Dd m_chosenp;
```

Appendix C (Continued)

```
hduVector3Dd m_zeroPosition;
hduVector3Dd VScaled;
hduVector3Dd VProjected;
//Declarations for FITTS' Task
hduVector3Dd Fitts'_Start;
hduVector3Dd Fitts'_End;
hduVector3Dd Fitts'_Delta;
double GoalDist;
//hduVector3Dd m_TrajPoint;
HDint m_gIDMenu;
void displayFunction(void);
void handleMenu(int);
void handleIdle(void);
void handleMouse(int, int, int, int);
void drawLine(void);
void displayWallsHaptically(void);
POINT_3D findClosestPoint(void);
void displayLine(void);
//TrajP_3D displayLine();
double Magnitude(void);
//double a1,b1,c1,a2,b2,c2,a3,b3,c3,a4,b4,c4;
double CP[4][3] = {             {100,100,0 },  //control points
                                                                {50,100,20 },
                                                                { -50,0,0 },
                                                                {-100,50,-
20}};double velocityMag;
//Create checkboard texture//
#define checkImageWidth 100
#define checkImageHeight 100
static GLubyte checkImage[checkImageHeight][checkImageWidth][4];
static GLuint texName;
// haptic device and rendering context handles
static HHD hHD = HD_INVALID_HANDLE;
static HHLRC hHLRC = 0;
// shape id for shape we will render hapticallyuint WallShapeId;
//Fitts'' Task (X-dir Constraint) -> option 1
//#define FITTS'TASK1
//Fitts'' Task (Y-dir Constraint) -> option 1
//Fitts'' Task (Z-dir Constraint) -> option 2
```

Appendix C (Continued)

```
// 3D motion & user controlled velocity scaling
//#define TASK1

#define REMOVEZ
//#define INCLUDEZ

#define NormalView
//#define ZView

#define XYZwalls
//#define Zwalls


#define TransNormal
//#define TransZaxis
//conditional compilation depending on type of assistance

//#define EXP1
//#define EXP2
//#define EXP3
//#define EXP4
//#define EXP5
//#define EXP6
//#define EXP7
//#define EXP8
//#define EXP9
//#define EXP10
//#define EXP11


/*************************************************************************
***************
*************************************************************************
***************/
/*void makeCheckImage(void)
{
        int i,j,c;
        for(i=0;i<checkImageHeight;i++){
                for(j=0;j<checkImageWidth;j++){
```

Appendix C (Continued)

```
                checkImage[i][j][0] = (GLubyte) c;
        checkImage[i][j][1] = (GLubyte) c;
                                        Appendix C (Continued)

        checkImage[i][j][2] = (GLubyte) c;
        checkImage[i][j][3] = (GLubyte) 255;
                                    }
                }
}*/

/*********************************************************************************
********
 GLUT initialization
*********************************************************************************
*******/

void initGlut(int argc, char* argv[])

{
   glutInit(&argc, argv); /* Initialize GLUT. */
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(600, 600);
   glutCreateWindow("Sphere-sphere Contact Example");

   glutDisplayFunc(displayFunction); /* Setup GLUT callbacks. */
   glutMouseFunc(handleMouse);
   glutIdleFunc(handleIdle);
   glutCreateMenu(handleMenu);
   //glutAddMenuEntry("About...", 0);
                glutAddMenuEntry("Fitts'' Task (X-dir Constraint)...", 0);
                glutAddMenuEntry("Fitts'' Task (Y-dir Constraint)...", 1);
                glutAddMenuEntry("Fitts'' Task (Z-dir Constraint)...", 2);
                glutAddMenuEntry("Linear Constraint Motion in 3D...", 3);
                glutAddMenuEntry("Velocity Scaling Assistance...", 4);
                glutAddMenuEntry("No assistance provided...", 5);
   glutAddMenuEntry("Quit", 6);
   glutAttachMenu(GLUT_RIGHT_BUTTON);


}
```

Appendix C (Continued)

```
{
   HDErrorInfo error;

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

   hHLRC = hlCreateContext(hHD);
   hlMakeCurrent(hHLRC);

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

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

   hlTouchableFace(HL_FRONT);
}
/*******************************************************************************
********
 Use the haptic device coordinate space as model space for graphics.
 Define orthographic projection to fit it. LLB: Low, Left, Back point of device workspace
 TRF: Top, Right, Front point of device workspace
*******************************************************************************
*******/
void initGraphics(const HDdouble LLB[3], const HDdouble TRF[3])
{    glMatrixMode(GL_PROJECTION); /* Setup perspective projection. */
   glLoadIdentity();
   HDdouble centerScreen[3];
   centerScreen[0] = (TRF[0] + LLB[0])/2.0;
   centerScreen[1] = (TRF[1] + LLB[1])/2.0;
   centerScreen[2] = (TRF[2] + LLB[2])/2.0;
   HDdouble screenDims[3];
   screenDims[0] = TRF[0] - LLB[0];
```

```
   HDdouble maxDimXY = (screenDims[0] > screenDims[1] ?
screenDims[0]:screenDims[1]);
   HDdouble maxDim = (maxDimXY > screenDims[2] ? maxDimXY:screenDims[2]);
   maxDim /= 2.0;

   glOrtho(centerScreen[0]-maxDim, centerScreen[0]+maxDim,
         centerScreen[1]-maxDim, centerScreen[1]+maxDim,
         centerScreen[2]+maxDim, centerScreen[2]-maxDim);

   printf("glortho %lf %lf %lf %lf %lf %lf\n",
        centerScreen[0]-maxDim, centerScreen[0]+maxDim,
        centerScreen[1]-maxDim, centerScreen[1]+maxDim,
        centerScreen[2]+maxDim, centerScreen[2]-maxDim);

   glShadeModel(GL_SMOOTH);

   glMatrixMode(GL_MODELVIEW); /* Setup model transformations. */
   glLoadIdentity();

   glClearDepth(1.0); /* Setup background colour. */
   //before glClearColor(0.7, 0.7, 0.7, 0); //gray

                  //new
                  glClearColor(205.0/255.0, 179.0/255.0, 149.0/255.0, 0);
                  //238-203-173

                  //glClearColor(1.0, 1.0, 1.0, 0); //gray
   glDisable(GL_DEPTH_TEST);

                  //glClearColor(0.0,0.0,0.0,0.0);
                  /*glShadeModel(GL_FLAT);
                  glEnable(GL_DEPTH_TEST);
                  makeCheckImage();
                  glPixelStorei(GL_UNPACK_ALIGNMENT,1);
                  glGenTextures(1, &texName);
                  glBindTexture(GL_TEXTURE_2D, texName);
                  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_RE
PEAT);
```

Appendix C (Continued)

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
                glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA,checkImageWidth,chec
kImageHeight,0,GL_RGBA,GL_UNSIGNED_BYTE,checkImage);*/
}


/*************************************************************************
********
   Setup graphics pipeline, lights etc.
*************************************************************************
*******/
void doGraphicsState()
{


#ifdef NormalView
                                glMatrixMode(GL_MODELVIEW); /* Setup model
transformations. */
#endif

#ifdef ZView
                                glMatrixMode(GL_MODELVIEW); /* Setup model
transformations. */
      //glRotatef(-3.1416/1.5,0.0,1.0,0.0);
      glRotatef(-3.1416/1.34,0.0,1.0,0.0);
#endif
                //              glMatrixMode(GL_MODELVIEW); // Setup model
transformations.
                                //glRotatef(-3.1416/2,0.0,1.0,0.0);
     //n glRotatef(-3.1416/1.5,0.0,1.0,0.0);
      //gluLookAt(0.0,10.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0);

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glEnable(GL_COLOR_MATERIAL);
                //add texture to the graphics
   //glColor3f(1.0f,0.0f,0.0f);
                //glEnable(GL_TEXTURE_2D);
```

Appendix C (Continued)

```
                        glColor3f(0.0, 0.7, 0.7); //gray
                        glTexCoord2f(0.0,0.0);glVertex3f(-300.0,500.0,0.0);
                        glTexCoord2f(0.0,1.0);glVertex3f(-10.0,500.0,187.0);
                        glTexCoord2f(1.0,1.0);glVertex3f(-10.0,-60.0,187.0);
                        glTexCoord2f(1.0,0.0);glVertex3f(-300.0,-210.0,0.0);
            glEnd();
            //right wall    glBegin(GL_QUADS);
                        glColor3f(0.0, 0.75, 0.75);

                        glTexCoord2f(0.0,0.0);glVertex3f(-10.0,500.0,187.0);
                        glTexCoord2f(0.0,1.0);glVertex3f(310.0,500.0,187.0);
                        glTexCoord2f(1.0,1.0);glVertex3f(310.0,-60.0,187.0);
                        glTexCoord2f(1.0,0.0);glVertex3f(-10.0,-60.0,187.0);
            glEnd();*/
//glShadeModel(GL_SMOOTH);

glEnable(GL_LIGHTING);
glEnable(GL_NORMALIZE);
glEnable(GL_LIGHT_MODEL_TWO_SIDE);
glShadeModel(GL_SMOOTH);

GLfloat lightZeroPosition[] = {10.0, 4.0, 100.0, 0.0};
//GLfloat lightZeroColor[] = {0.6, 0.6, 0.6, 1.0}; /* green-tinted */
        GLfloat lightZeroColor[] = {0.1, 1.0, 0.1, 1.0}; /* green-tinted */
GLfloat lightOnePosition[] = {-1.0, -2.0, -100.0, 0.0};
GLfloat lightOneColor[] = {0.6, 0.6, 0.6, 1.0}; /* red-tinted */


GLfloat light_ambient[] = {0.8, 0.8, 0.8, 1.0}; /* White diffuse light. */
GLfloat light_diffuse[] = {0.0, 0.0, 0.0, 1.0}; /* White diffuse light. */
GLfloat light_position[] = {0.0, 0.0, 100.0, 1.0}; /* Infinite light loc. */

glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, 1);
glLightfv(GL_LIGHT0, GL_POSITION, lightZeroPosition);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor);
glLightfv(GL_LIGHT1, GL_POSITION, lightOnePosition);
glLightfv(GL_LIGHT1, GL_DIFFUSE, lightOneColor);
glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
```

Appendix C (Continued)

```
//glFlush();
                //glDisable(GL_TEXTURE_2D);

}

/*************************************************************************
********
 Draw Slave Sphere, at given position.
*************************************************************************
*******/
void displayVisitorSphere(GLUquadricObj* quadObj, const double position[3])
{
                //char buffer[BUFFERSIZE];
   glMatrixMode(GL_MODELVIEW);
                glPushMatrix();

                /*hlBeginFrame();
   hlCheckEvents();
   HLboolean buttDown = false;
   hlGetBooleanv(HL_BUTTON2_STATE,&buttDown);
                if (buttDown==true)
   {
                                fprintf(stdout, "i am ok\n");
                }
                hlEndFrame();*/

       //Display one sphere to represent the haptic cursor and the dynamic
        //charge.
       #ifdef TransNormal
       glTranslatef(position[0], position[1], position[2]);
       p = position[0];
                q = position[1];
                r = position[2];
       #endif
                                //MODIFIED FOR Z AXIS FITTS' TASK
IMPLEMENTATION
       #ifdef TransZaxis
       glTranslatef(position[0], position[1],position[2]);
       p = position[0];
                q = position[1];
```

Appendix C (Continued)

```
//Modified by Ramya in order to assign the position vector in variables p,q,r.

                                    //Slave object is blue.
                                   glColor4f(0.1, 0.1, 0.9, 1.0);

      // The center sphere.
      gluSphere(quadObj, VISITOR_SPHERE_RADIUS, 20, 20);
      glPopMatrix();

}
/*******************************************************************************
*************
Draw the quadrilaterals haptically so that we can give a sanse of touch to the walls.
*******************************************************************************
**************/
void displayWallsHaptically(void)
{
   hlBeginFrame();

   // set material properties for the shapes to be drawn
   hlMaterialf(HL_FRONT, HL_STIFFNESS, 0.7f);
   hlMaterialf(HL_FRONT, HL_DAMPING, 0.1f);

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   //glEnable(GL_COLOR_MATERIAL);
                  //add texture to the graphics
   //glColor3f(1.0f,0.0f,0.0f);
                  //glEnable(GL_TEXTURE_2D);
                  //glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_R
EPLACE);
                  //glBindTexture(GL_TEXTURE_2D,texName);
                  //left wall

#ifdef XYZwalls

   hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, WallShapeId);
                  glBegin(GL_QUADS);
```

Appendix C (Continued)

```
                                glVertex3f(-0.0f,500.0f,187.0f);
                                glVertex3f(-0.0f,-60.0f,187.0f);
                                glVertex3f(-290.0f,-210.0f,0.0f);


                                glColor3f(0.0, 0.75, 0.75);
                                glVertex3f(-0.0f,500.0f,187.0f);
                                glVertex3f(300.0f,500.0f,187.0f);
                                glVertex3f(300.0f,-60.0f,187.0f);
                                glVertex3f(-0.0f,-60.0f,187.0f);
                        glEnd();
  hlEndShape();

#endif


#ifdef Zwalls


   hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, WallShapeId);
                        glBegin(GL_QUADS);

                                glColor3f(0.0, 0.7, 0.7); //gray
      glVertex3f(-290.0f,500.0f,0.0f);
                                glVertex3f(-0.0f,500.0f,187.0f);
                                glVertex3f(-0.0f,-60.0f,187.0f);
                                glVertex3f(-290.0f,-210.0f,0.0f);


                                glColor3f(0.0, 0.75, 0.75);
                                glVertex3f(-0.0f,500.0f,187.0f);
                                glVertex3f(300.0f,500.0f,187.0f);
                                glVertex3f(300.0f,-60.0f,187.0f);
                                glVertex3f(-0.0f,-60.0f,187.0f);
                        glEnd();
  hlEndShape();
#endif

  hlEndFrame();
```

Appendix C (Continued)

```
/*************************************************************************
*********************************
To compute the Magnitude between two points
*************************************************************************
*********************************/
double Magnitude(POINT_3D pointend,POINT_3D pointstart)
{
                double Magnitude;
                double r,s,t;
                r = pointend.x-pointstart.x;
                s = pointend.y-pointstart.y;
                t = pointend.z-pointstart.z;
   Magnitude = sqrt((r*r)+(s*s)+(t*t));
                return Magnitude;
}

/*************************************************************************
*********************************
Draw a straight line and compute the distenace between haptic point and any point on the
straight line
*************************************************************************
*********************************/
void displayLine(void)
{
  //if (m_gIDMenu == 4) {
  #ifdef TASK1
 //pointstart.x = -200.0;
   pointstart.x = -200.0;
   pointstart.y = 0.0;
   pointstart.z = 0.0;
   //pointend.x = 0.0;
   pointend.x = 200.0;
   pointend.y = 300.0;
   pointend.z = 0.0;
  //to draw a straight line based on the above computed points using OpenGL//
  glMatrixMode(GL_MODELVIEW); //Setup model transformations.
  glPushMatrix();
  glColor3f(1.0,0.0,0.0);
  //hlTouchModel(HL_CONSTRAINT);
```

Appendix C (Continued)

```
 glBegin(GL_LINE_STRIP);
   glVertex3d(pointstart.x,pointstart.y,pointstart.z);
   glVertex3d(pointend.x,pointend.y,pointend.y);
```

Appendix C (Continued)

```
   glEnd();

   #endif

/*************************************************************************
**************************************
Draw a second line for the Fitts' Task...FITTS' TASK IMPLEMENTATION,SETTING
*************************************************************************
**************************/
   //if (m_gIDMenu == 0) {
                  #ifdef FITTS'TASK1
 //pointstart.x = -200.0;
    pointstart.x = -95.0;
    pointstart.y = 0.5;
    pointstart.z = 0.0;
    //pointend.x = 0.0;
    pointend.x = 95.0;
    pointend.y = 0.5;
    pointend.z = 0.0;
   //to draw a straight line based on the above computed points using OpenGL//
   glMatrixMode(GL_MODELVIEW); //Setup model transformations.
   glPushMatrix();
   glColor3f(1.0,0.0,0.0);
   //hlTouchModel(HL_CONSTRAINT);
   //hlTouchModelf(HL_SNAP_DISTANCE,1.5);
   glBegin(GL_LINE_STRIP);
   glLineWidth(10);
   glVertex3d(pointstart.x,pointstart.y,pointstart.z);
   glVertex3d(pointend.x,pointend.y,pointend.z);
   glEnd();
   glColor3f(0.0,0.0,0.9);
   glBegin(GL_QUADS);
   glLineWidth(10.0);
```

```
  glVertex3d(pointstart.x+10.0,pointstart.y+10.0,pointstart.z);
  glVertex3d(pointstart.x+10.0,pointstart.y-10.0,pointstart.z);
  glVertex3d(pointstart.x-10.0,pointstart.y-10.0,pointstart.z);
  //glEnd();
  glColor3f(0.0,0.0,0.9);
  glVertex3d(pointend.x-10.0,pointend.y+10.0,pointend.z);
 glVertex3d(pointend.x+10.0,pointend.y+10.0,pointend.z);

  glVertex3d(pointend.x+10.0,pointend.y-10.0,pointend.z);
  glVertex3d(pointend.x-10.0,pointend.y-10.0,pointend.z);
  glEnd();
  //}//end if gIDMenu == 0
  #endif
/*************************************************************************
***********
  FITTS' TASK with the line implemented in the Y direction/motion.

*************************************************************************
**********/
  #ifdef FITTS'TASK2
 //if (m_gIDMenu == 1) {
   pointstart.x = 0.0;
   pointstart.y = -60.0;
   pointstart.z = 0.0;
   pointend.x = 0.0;
   pointend.y = 130.0;
   pointend.z = 0.0;
  //to draw a straight line based on the above computed points using OpenGL//
  glMatrixMode(GL_MODELVIEW); //Setup model transformations.
  glPushMatrix();
  glColor3f(1.0,0.0,0.0);
  //hlTouchModel(HL_CONSTRAINT);
  //hlTouchModelf(HL_SNAP_DISTANCE,1.5);
  glBegin(GL_LINE_STRIP);
  glLineWidth(10);
  glVertex3d(pointstart.x,pointstart.y,pointstart.z);
  glVertex3d(pointend.x,pointend.y,pointend.z);
  glEnd();
```

```
  glColor3f(0.0,0.0,0.9);
  glBegin(GL_QUADS);
  glLineWidth(10.0);
  glVertex3d(pointstart.x-12.0,pointstart.y+10.0,pointstart.z);
  glVertex3d(pointstart.x+12.0,pointstart.y+10.0,pointstart.z);
  glVertex3d(pointstart.x+12.0,pointstart.y-10.0,pointstart.z);
  glVertex3d(pointstart.x-12.0,pointstart.y-10.0,pointstart.z);
  //glEnd();
  glColor3f(0.0,0.0,0.9);
  glVertex3d(pointend.x-12.0,pointend.y+10.0,pointend.z);
  glVertex3d(pointend.x+12.0,pointend.y+10.0,pointend.z);
```

Appendix C (Continued)

```
  glVertex3d(pointend.x+12.0,pointend.y-10.0,pointend.z);
  glVertex3d(pointend.x-12.0,pointend.y-10.0,pointend.z);
  glEnd();
 //}

  #endif
/*************************************************************************
***********
  FITTS' TASK with the line implemented in the Z direction/motion.

*************************************************************************
**********/
 #ifdef FITTS'TASK3:
 //if (m_gIDMenu == 3) {
    pointstart.x =-190.0;
    //pointstart.x =-100.0;
    pointstart.y = 18.0;
    //pointstart.z = 180.0;
    pointstart.z = -5.0;
    //pointend.x = 20.0;
    pointend.x = 60.0;
    //pointend.y = 80.0;
     pointend.y =110.0;
    //pointend.z = 0.0;
    pointend.z = -40.0;
   /* pointstart.x =0.0;
```

Appendix C (Continued)

```
  glVertex3d(pointend.x,pointend.y,pointend.z);
  glEnd();

  glColor3f(0.0,0.0,0.9);
  glBegin(GL_QUADS);
  glLineWidth(10.0);
  glVertex3d(pointstart.x-12.0,pointstart.y+10.0,pointstart.z+10.0);
  glVertex3d(pointstart.x+12.0,pointstart.y+10.0,pointstart.z+10.0);
  glVertex3d(pointstart.x+12.0,pointstart.y,pointstart.z-10.0);
  glVertex3d(pointstart.x-12.0,pointstart.y,pointstart.z-10.0);
 //glEnd();
  glColor3f(0.0,0.0,0.9);

  glVertex3d(pointend.x-12.0,pointend.y+10.0,pointend.z+10.0);
  glVertex3d(pointend.x+12.0,pointend.y+10.0,pointend.z+10.0);
  glVertex3d(pointend.x+12.0,pointend.y,pointend.z-10.0);
  glVertex3d(pointend.x-12.0,pointend.y,pointend.z-10.0);
  glEnd();

 //}

  #endif



/*************************************************************************
***********
  FITTS' TASK with the line implemented in the Z direction/motion.

*************************************************************************
**********/
//if (m_gIDMenu == 2) {

 #ifdef FITTS'TASK4:

   pointstart.x =0.0;
   //pointstart.x =-100.0;
   pointstart.y =0.0;
   //pointstart.z = 180.0;
   pointstart.z =-50.0;
```

Appendix C (Continued)

```
 //pointend.y = 80.0;
  pointend.y =0.0;
 //pointend.z = 0.0;
 pointend.z =75.0;


/* pointstart.x =0.0;
 pointstart.y = 0.0;
 pointstart.z = -180.0;
// pointstart.z = -150.0;

 pointend.x = 0.0;
 pointend.y = 0.0;
 pointend.z =240.0;
 //pointend.z =300.0;*/
//to draw a straight line based on the above computed points using OpenGL//
glMatrixMode(GL_MODELVIEW); //Setup model transformations.
glPushMatrix();
glColor3f(1.0,0.0,0.0);
//hlTouchModel(HL_CONSTRAINT);
//hlTouchModelf(HL_SNAP_DISTANCE,1.5);
glBegin(GL_LINE_STRIP);
glLineWidth(10);
glVertex3d(pointstart.x,pointstart.y,pointstart.z);
glVertex3d(pointend.x,pointend.y,pointend.z);
glEnd();

glColor3f(0.0,0.0,0.9);
glBegin(GL_QUADS);
glLineWidth(10.0);
glVertex3d(pointstart.x,pointstart.y+10.0,pointstart.z+10.0);
glVertex3d(pointstart.x,pointstart.y+10.0,pointstart.z-10.0);
glVertex3d(pointstart.x,pointstart.y-10.0,pointstart.z-10.0);
glVertex3d(pointstart.x,pointstart.y-10.0,pointstart.z+10.0);

/*glVertex3d(pointstart.x-10.0,pointstart.y,pointstart.z-10.0);
glVertex3d(pointstart.x-10.0,pointstart.y,pointstart.z+10.0);
glVertex3d(pointstart.x+10.0,pointstart.y,pointstart.z+10.0);
glVertex3d(pointstart.x+10.0,pointstart.y,pointstart.z-10.0);*/
```

Appendix C (Continued)

```
 glColor3f(0.0,0.0,0.9);
   glVertex3d(pointend.x,pointend.y+10.0,pointend.z+10.0);
   glVertex3d(pointend.x,pointend.y+10.0,pointend.z-10.0);
   glVertex3d(pointend.x,pointend.y-10.0,pointend.z-10.0);
   glVertex3d(pointend.x,pointend.y-10.0,pointend.z+10.0);


   /*glVertex3d(pointstart.x+10.0,pointstart.y,pointstart.z+10.0);
   glVertex3d(pointstart.x+10.0,pointstart.y,pointstart.z-10.0);
   glVertex3d(pointstart.x-10.0,pointstart.y,pointstart.z-10.0);
   glVertex3d(pointstart.x-10.0,pointstart.y,pointstart.z-10.0);*/


   glEnd();
//}

   #endif
}


/*************************************************************************
**************************************
To compute the point on the line which is closest to the haptic device based on the
perpendicularity condition.
*************************************************************************
**********************************/
//POINT_3D IntersectionPoint(const double position[3],POINT_3D
pointstart,POINT_3D pointend)
POINT_3D IntersectionPoint()
{
                double LineMag;
                double U;



    /*hapticposition.x=position[0];
    hapticposition.y=position[1];
    hapticposition.z=position[2];*/
    hapticposition.x=p;
```

Appendix C (Continued)

```
   hapticposition.z=r;

                LineMag = Magnitude(pointend,pointstart);
                                Appendix C (Continued)

                U= (((hapticposition.x - pointstart.x)*(pointend.x - pointstart.x))+
                                ((hapticposition.y - pointstart.y)*(pointend.y -
pointstart.y))+
                                ((hapticposition.z - pointstart.z)*(pointend.z -
pointstart.z)))/(LineMag*LineMag);

                if(U<0.0 || U>1.0)
                {

                                //printf ("U = %d,x =%d\n", U, hapticposition.x);
                                printf ("closest point does not fall within the line
segment\n");
                }
                else{
                                  Intersection.x = pointstart.x+ (U *(pointend.x-
pointstart.x));
                                   Intersection.y = pointstart.y+ (U *(pointend.y-
pointstart.y));
                                   Intersection.z = pointstart.z+ (U *(pointend.z-
pointstart.z));
                                //printf ("near x = %3.4f, near y = %3.4f \n", Intersection.x,
Intersection.y);
                }


                return Intersection;

}


/*************************************************************************
**********************
This is the  algorithm to divide the trajectory into 50 equal points and store them in
TrajPoint.
```

Appendix C (Continued)

```
   pointstart.x = -200.0;
   pointstart.y = 0.0;
   pointstart.z = 0.0;
   //pointend.x = 0.0;
   pointend.x = 200.0;
   pointend.y = 300.0;
   pointend.z = 0.0;



   //allocate memory for bezier points
                TrajPoint = (TrajP_3D*)malloc(sizeof(TrajP_3D)*(NPT+1));

//////////////////////////start compute trajectory points/////////////////////////////////////

                int i,j,k;
                double t;
                double DeltaX,DeltaY,DeltaZ;
   TrajPoint[0].x = pointstart.x ;
   TrajPoint[0].y = pointstart.y;
                TrajPoint[0].z =0.0;
                TrajPoint[0].indx = 0;

   TrajPoint[NPT].x=pointend.x;
   TrajPoint[NPT].y=pointend.y;
   TrajPoint[NPT].z=0.0;
                TrajPoint[NPT].indx = NPT;

   DeltaX= (pointstart.x-pointend.x)/NPT;
   DeltaY= (pointstart.y-pointend.y)/NPT;
   DeltaZ=0.0;
                nearest_pt = IntersectionPoint();

                for (i= 1;i<NPT;i++)
                {
                   t = (double)i/(NPT-1);

                                TrajPoint[i].x=t*DeltaX;
                    TrajPoint[i].y=t*DeltaY;
```

Appendix C (Continued)

```
                }
  glMatrixMode(GL_MODELVIEW); //Setup model transformations.
  glPushMatrix();
  glColor3f(1,0,0);
  //hlTouchModel(HL_CONSTRAINT);
  //hlTouchModelf(HL_SNAP_DISTANCE,1.5);
  glBegin(GL_LINE_STRIP);
  glVertex3d(pointstart.x,pointstart.y,pointstart.z);
  glVertex3d(pointend.x,pointend.y,pointend.y);
  glEnd();


}*/
//////////////////TESTING/////////////////////////////////////////////
void ContactModel::UpdateEffectorPosition(const hduVector3Dd visitor)
{
                double dx, dy, dz,Dist;
                HDdouble mtime;
  m_currentDistance = 0.0;
  m_effectorPosition = visitor;
                int m_springlength = 20;
                double DAMPING = 0.00225;
                double THRESHOLD = 20.00;
                double SPRING_END = 80.00;
                double DIST;
                double Res;
                int index;
                //int m_springlength = 100;
                //double scale;
  //smallest_dist=13;

                hduVector3Dd inv;
  hduVector3Dd hapticinv;
  hduVector3Dd TrueDist;
  hduVector3Dd VelocityProjected;
  hduVector3Dd TrajVec;
                hduVector3Dd Force;
  hduVector3Dd ForceProjected;
  hduVector3Dd DeltaVel;
  hduVector3Dd mforceproj;
```

```
        hduVector3Dd m_modified;
  hduVector3Dd Normal_Vel;
  hduVector3Dd D1;
  hduVector3Dd Normal_pos;
  hduVector3Dd Num;
  hduVector3Dd Denom;
  hduVector3Dd distance;
        hduVector3Dd Normal;
  hduVector3Dd norm_velocity;
  hduVector3Dd EndPoint;
  hduVector3Dd n_vector;
  hduVector3Dd Deltaxyz;
  hduVector3Dd n_traj;
  hduVector3Dd Delta_Dist;
  hduVector3Dd Delta_rightside;
  hduVector3Dd dx_OtherSide;
  hduVector3Dd second_vector;
  hduVector3Dd third_vector;
  hduVector3Dd Traj_left;
  hduVector3Dd Traj_right;

  hduVector3Dd Dist_Goal;


  hduVector3Dd g_vector;
  hduVector3Dd Dist_Goalleft;
  hduVector3Dd Dist_Goalright;
        hduVector3Dd Previous_Xcoordinate;
  hduVector3Dd Mid_Point;

  double Dist_left;
        double Dist_right;

        double m_mass;
        double m_kStiffness;
        //double m_kDamping;
        double Distance;
        double Distance2;
  double DeltaX,DeltaY,DeltaZ;
```

```
                //HDdouble MaxForce;

   //double ForceDotProd;
   double Square;
                double dt = 0.001;

   //double SCALING = 0.00001;
   double SCALING = 1.0;
                double m_force;
                bool firstTime = true;
   double SpringLimitDistance =20.0;
   double scaleForceFactor = 0.8;
                double dismag;
                double mass = 0.001;//kg
                double m_stiffness = 0.5;
                double m_damping = 2*sqrt(mass*m_stiffness);
                double mDist,oldDist,o_Dist,yDist;
   OmniLocalStruct gOmni;
                int timer = 1 ;
   bool FirstTime = true;
                bool f_Time = true;
   bool firstTrip;

   m_effectorPosition = visitor;
                m_visitorPosition = m_effectorPosition;

                nearest_pt = IntersectionPoint();
   //printf("%3.6f %3.6f\n",nearest_pt.x,nearest_pt.y);

                /*************************************************************
*****************************
                RAMYA:The code where we determine the point on the trajectory which
is closest to the
                effector sphere and the next adjacent point to the closest point.These
points are stored
                in TrajPoint and can read from there by pulling out the proper index
                *************************************************************
*****************************/
```

Appendix C (Continued)

```
                                                    ClosestPoint=TrajPoint[i];

                                    }
                    ClosestPoint=displayLine();
                    index=ClosestPoint.indx;
                    Next_Pt.x=nearest_pt.x+DeltaX;
                    Next_Pt.y=nearest_pt.y+DeltaY;
                    Next_Pt.z=nearest_pt.z+DeltaZ;*/

   //printf("%3.6f\n",Distance2);



                    //printf("%3.4f %3.4f %3.4f
%3.4f\n",ClosestPoint.x,ClosestPoint.y,Next_Pt.x,Next_Pt.y);

   /*Closest3DPoint[0]=nearest_pt.x;
   Closest3DPoint[1]=nearest_pt.y;
   Closest3DPoint[2]=0;*/
#ifdef REMOVEZ

  DeltaX= (pointstart.x-pointend.x)/NPT;
   DeltaY= (pointstart.y-pointend.y)/NPT;
   DeltaZ=0.0;
                    Delta_Dist[0]=DeltaX;
                    Delta_Dist[1]=DeltaY;
                    Delta_Dist[2]=DeltaZ;
                    Distance2=Delta_Dist.magnitude();
                    vectorPoint[0] = nearest_pt.x;
                    vectorPoint[1] = nearest_pt.y;
                    vectorPoint[2] = 0;
   dx =vectorPoint[0]-m_visitorPosition[0];
                    dy =vectorPoint[1]-m_visitorPosition[1];
   dz = 0;
#endif
#ifdef INCLUDEZ
   DeltaX= (pointstart.x-pointend.x)/NPT;
   DeltaY= (pointstart.y-pointend.y)/NPT;
   DeltaZ=(pointstart.z-pointend.z)/NPT;;
                    Delta_Dist[0]=DeltaX;
```

```
                Delta_Dist[1]=DeltaY;
                Delta_Dist[2]=DeltaZ;
                Distance2=Delta_Dist.magnitude();
                vectorPoint[0] = nearest_pt.x;
                vectorPoint[1] = nearest_pt.y;
                vectorPoint[2] = nearest_pt.z;
   dx =vectorPoint[0]-m_visitorPosition[0];
                dy =vectorPoint[1]-m_visitorPosition[1];
   dz =vectorPoint[2]-m_visitorPosition[2];

#endif

                //The algorithm which would store the end-point on the trajectory in a hdu
vector format//
                EndPoint[0] = pointend.x;
   EndPoint[1] = pointend.y;
   EndPoint[2] = pointend.z;



                //test

                Deltaxyz[0]=EndPoint[0]-m_effectorPosition[0];
                Deltaxyz[1]=EndPoint[1]-m_effectorPosition[1];
   Deltaxyz[2]=EndPoint[2]-m_effectorPosition[2];

   //Algorithm to compute the direction vector of the imaginary straight line on the right
                //side of the trajectory path.
                Delta_rightside[0]=(EndPoint[0]+VISITOR_SPHERE_RADIUS)-
(m_visitorPosition[0]);
                Delta_rightside[1]=EndPoint[1]-m_visitorPosition[1];
   Delta_rightside[2]=EndPoint[2]-m_visitorPosition[2];

                Dist_rightside=Delta_rightside.magnitude();

   second_vector[0]=Delta_rightside[0]/Dist_rightside;
                second_vector[1]=Delta_rightside[1]/Dist_rightside;
   second_vector[2]=Delta_rightside[2]/Dist_rightside;
```

```
                dx_OtherSide[0]=(vectorPoint[0]-VISITOR_SPHERE_RADIUS)-
(m_visitorPosition[0]-VISITOR_SPHERE_RADIUS);
                dx_OtherSide[1]=vectorPoint[1]-m_visitorPosition[1];
    dx_OtherSide[2]=vectorPoint[2]-m_visitorPosition[2];
    Dist_leftside=dx_OtherSide.magnitude();
    third_vector[0]=dx_OtherSide[0]/Dist_rightside;
                third_vector[1]=dx_OtherSide[1]/Dist_rightside;
    third_vector[2]=dx_OtherSide[2]/Dist_rightside;

                Distance =Deltaxyz.magnitude();
                n_traj[0]=Deltaxyz[0]/Distance;
                n_traj[1]=Deltaxyz[1]/Distance;
                n_traj[2]=Deltaxyz[2]/Distance;

   TrueDist[0] = dx;
                TrueDist[1] = dy;
                //TrueDist[2] = 0.0;
   TrueDist[2] = dz;
                Dist = TrueDist.magnitude();

                n_vector[0]=TrueDist[0]/Dist;
                n_vector[1]=TrueDist[1]/Dist;
                n_vector[2]=TrueDist[2]/Dist;


                m_currentDistance = Dist;

                TrajVec[0]=pointend.x-pointstart.x;
   TrajVec[1]=pointend.y-pointstart.y;
   TrajVec[2]=pointend.z-pointstart.z;

                //printf("%3.6f\n", gOmni.option);

   //m_currentDistance = 0.0;
   //m_effectorPosition = visitor;
   //_centerToEffector = m_effectorPosition ;//- m_fixedCenter;
                //double SpringLimitDistance = 80;

   //m_currentDistance =  m_centerToEffector.magnitude();
```

```
  //if(m_currentDistance > SpringLimitDistance)
                if (firstTime==true) {
                                    Appendix C (Continued)
                                firstTime = false;
                                mtime = 0.0;
                                oldDist=Dist;
                }
  mDist=(oldDist+Dist)/2;
  //The distance computed based on the extreme right side of the sphere
                //and the point on the trajectory.
                /*if (firstTime==true) {
                                firstTime = false;
                                mtime = 0.0;
                                oldDist=Dist_rightside;
                }
  mDist=(oldDist+Dist_rightside)/2;*/

   if (f_Time==true) {
                                f_Time = false;
                                mtime = 0.0;
                                o_Dist=Dist_leftside;
                }
  yDist=(o_Dist+Dist_leftside)/2;

#ifdef EXP1:

if(m_currentDistance > 130.0)
{
                 m_forceOnVisitor[0]=m_forceOnVisitor[1]=m_forceOnVisitor[2]=0.0;
}
//else if((m_currentDistance>10.0)&&(m_currentDistance10.0))
else if(m_effectorPosition[0]>vectorPoint[0])
{
                 if(m_effectorPosition[0]<(vectorPoint[0]+20.0))
                 {
                                if(m_effectorPosition[0]<(vectorPoint[0]+10.0))
                                {

                if(m_effectorPosition[0]<(vectorPoint[0]+5.0))
```

Appendix C (Continued)

```
        if(m_effectorPosition[1]<Y_Limit)
                                        {
                                                m_forceOnVisitor[0]=
18.0*STIFFNESS*Dist*n_vector[0];
                                                m_forceOnVisitor[1]=
18.0*STIFFNESS*Dist*n_vector[1];
                        Appendix C (Continued)

                                                m_forceOnVisitor[2]=
18.0*STIFFNESS*Dist*n_vector[2];


                                        }
                                        else
                                        {

if(m_effectorPosition[1]>Y_End)
                                                {
            m_forceOnVisitor[0]=
5.0*STIFFNESS*Dist*n_vector[0]+(0.0015*TrajVec[0]);
                                              m_forceOnVisitor[1]=
5.0*STIFFNESS*Dist*n_vector[1]+(0.0015*TrajVec[0]);
                                                m_forceOnVisitor[2]=
5.0*STIFFNESS*Dist*n_vector[2]+(0.0015*TrajVec[0]);
                                                }
                                                else
                                                {
            m_forceOnVisitor[0]=
15.0*STIFFNESS*Dist*n_vector[0]+(0.0015*TrajVec[0]);



                                                m_forceOnVisitor[1]=
15.0*STIFFNESS*Dist*n_vector[1]+(0.0015*TrajVec[0]);
                                                m_forceOnVisitor[2]=
15.0*STIFFNESS*Dist*n_vector[2]+(0.0015*TrajVec[0]);
                                                }


                                        }
```

Appendix C (Continued)

```
        //printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
                                                }
                                                else
                                                {
        if(m_effectorPosition[1]<Y_Limit)
                                                 {
```

Appendix C (Continued)

```
                                                    m_forceOnVisitor.set(-1.2*exp(-
mDist/70.0), 1.2*exp(-mDist/70.0), 0.0);


                                                 }

                                                 else
if((m_effectorPosition[1]>Y_Limit)&&(m_effectorPosition[1]<Y_End))

                                                 {


            m_forceOnVisitor[0]=
0.1*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);

                                        m_forceOnVisitor[1]=
0.8*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);

            m_forceOnVisitor[2]=
scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);

                                                 }
                                                }
                                    }

                                    else
                                    {
                                      if(m_effectorPosition[1]<Y_Limit)
```

Appendix C (Continued)

```
                                    }


                                   }
                    }
                    else
                    {
        if(m_effectorPosition[1]<Y_Limit)
                                   {


                                    m_forceOnVisitor.set(-1.2*exp(-mDist/70.0), 1.2*exp(-
mDist/70.0), 0.0);

                    //m_forceOnVisitor.set(0.0,0.0,0.0);
                                   }
        else if(m_effectorPosition[1]>Y_Limit)
                                   {

          m_forceOnVisitor.set(0.0,0.0,0.0);


                                   }


                    }


  }
  else if(m_effectorPosition[0]<vectorPoint[0])
  {

     m_forceOnVisitor[0]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);
     m_forceOnVisitor[1]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);
     m_forceOnVisitor[2]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);

  }
```

Appendix C (Continued)

```
/**************************************************************************
************
Test 2 with just the projection component on the trajectory without the high spring force.
**************************************************************************
****************/
#ifdef EXP2if(m_currentDistance > 130.0)
 {
                m_forceOnVisitor[0]=m_forceOnVisitor[1]=m_forceOnVisitor[2]=0.0;
 }
 //else if((m_currentDistance>10.0)&&(m_currentDistance10.0))
 else if(m_effectorPosition[0]>vectorPoint[0])



 {
                 if(m_effectorPosition[0]<(vectorPoint[0]+20.0))
                 {
                                 if(m_effectorPosition[0]<(vectorPoint[0]+10.0))
                                 {

                if(m_effectorPosition[0]<(vectorPoint[0]+8.0))
                                                 {

          if(m_effectorPosition[1]<Y_Limit)
                                                  {
                                                                  m_forceOnVisitor[0]=
18.0*STIFFNESS*Dist*n_vector[0];
                                                                  m_forceOnVisitor[1]=
18.0*STIFFNESS*Dist*n_vector[1];
                                                                  m_forceOnVisitor[2]=
18.0*STIFFNESS*Dist*n_vector[2];


                                                  }
                                                  else
                                                  {

            m_forceOnVisitor[0]=
scaleForceFactor*(5.0*STIFFNESS*Dist*n_vector[0])+(0.002*TrajVec[0]);
```

```
                                                    m_forceOnVisitor[1]=
scaleForceFactor*(5.0*STIFFNESS*Dist*n_vector[1])+(0.002*TrajVec[1]);

                                                    m_forceOnVisitor[2]=
scaleForceFactor*(5.0*STIFFNESS*Dist*n_vector[2])+(0.002*TrajVec[2]);


                                                }



        //printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
                                                }
                                                else
                                                {
        if(m_effectorPosition[1]<Y_Limit)
                                                {

                                                    m_forceOnVisitor.set(-1.5*exp(-
mDist/70.0), 1.2*exp(-mDist/70.0), 0.0);


                                                }

                                                else
if((m_effectorPosition[1]>Y_Limit)&&(m_effectorPosition[1]<Y_End))

                                                {


            m_forceOnVisitor[0]=
0.1*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);

                                        m_forceOnVisitor[1]=
0.8*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);

            m_forceOnVisitor[2]=
scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);
```

Appendix C (Continued)

```
                                        }
                                      }
                                }

                                else
                                {
                                  if(m_effectorPosition[1]<Y_Limit)
                                  {

          m_forceOnVisitor.set(-1.5*exp(-mDist/70.0), 1.2*exp(-mDist/70.0), 0.0);
//m_forceOnVisitor.set(0.0,0.0,0.0);

                                  }
        else if(m_effectorPosition[1]>Y_Limit)
                                  {
            m_forceOnVisitor.set(0.0,0.0,0.0);


                                  }


                                }
                    }
                    else
                    {
       if(m_effectorPosition[1]<Y_Limit)
                                  {


                                   m_forceOnVisitor.set(-1.5*exp(-mDist/70.0), 1.2*exp(-
mDist/70.0), 0.0);

                  //m_forceOnVisitor.set(0.0,0.0,0.0);
                                  }
       else if(m_effectorPosition[1]>Y_Limit)
                                  {

         m_forceOnVisitor.set(0.0,0.0,0.0);

             }
```

Appendix C (Continued)

```
 else if(m_effectorPosition[0]<vectorPoint[0])
 {

    m_forceOnVisitor[0]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);
    m_forceOnVisitor[1]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);
    m_forceOnVisitor[2]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);

 }
#endif

/*************************************************************************
Test 3 with just the projection component on the trajectory getting normalized.
*************************************************************************
***************/
#ifdef EXP3
if(m_currentDistance > 130.0)
 {
                m_forceOnVisitor[0]=m_forceOnVisitor[1]=m_forceOnVisitor[2]=0.0;
 }
 //else if((m_currentDistance>10.0)&&(m_currentDistance10.0))
 else if(m_effectorPosition[0]>vectorPoint[0])
 {
                 if(m_effectorPosition[0]<(vectorPoint[0]+20.0))
                 {
                                if(m_effectorPosition[0]<(vectorPoint[0]+10.0))
                                {

               if(m_effectorPosition[0]<(vectorPoint[0]+5.0))
                                                {

          if(m_effectorPosition[1]<Y_Limit)
                                                  {
                                                                m_forceOnVisitor[0]=
18.0*STIFFNESS*Dist*n_vector[0];
                                                                m_forceOnVisitor[1]=
18.0*STIFFNESS*Dist*n_vector[1];
                                                                m_forceOnVisitor[2]=
18.0*STIFFNESS*Dist*n_vector[2];

                                                  }
```

Appendix C (Continued)

```
                                                    else
                                                    {

            m_forceOnVisitor[0]=-0.9;
           m_forceOnVisitor[1]=0.0;
                        m_forceOnVisitor[2]=0.0;

ForceDotProd=(m_forceOnVisitor[0]*TrajVec[0])+(m_forceOnVisitor[1]*TrajVec[1])+(
m_forceOnVisitor[2]*TrajVec[2]);
                        Square=(TrajVec.magnitude())*(TrajVec.magnitude());
                        ForceProjected=-(ForceDotProd/ Square)*TrajVec;
           m_forceOnVisitor[0]=0.5*ForceProjected[0];
           m_forceOnVisitor[1]=ForceProjected[1];


           m_forceOnVisitor[2]=ForceProjected[2];
           printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
                                                     }



        //printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
                                                   }
                                                   else
                                                   {
        if(m_effectorPosition[1]<Y_Limit)
                                                    {

                                                      m_forceOnVisitor.set(-1.2*exp(-
mDist/70.0), 1.2*exp(-mDist/70.0), 0.0);


                                                   }

                                                   else
if((m_effectorPosition[1]>Y_Limit)&&(m_effectorPosition[1]<Y_End))

                                                   {
```

Appendix C (Continued)

```
            m_forceOnVisitor[0]=
0.1*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);

                                        m_forceOnVisitor[1]=
0.8*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);

            m_forceOnVisitor[2]=
scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);

                                                  }
                                                 }
                                    }

                                    else
                                    {
                                      if(m_effectorPosition[1]<Y_Limit)
                                      {

          m_forceOnVisitor.set(-1.2*exp(-mDist/70.0), 1.2*exp(-mDist/70.0), 0.0);
//m_forceOnVisitor.set(0.0,0.0,0.0);

                                      }
        else if(m_effectorPosition[1]>Y_Limit)
                                      {
            m_forceOnVisitor.set(0.0,0.0,0.0);


                                      }

                                    }
                        }
                        else
                        {
       if(m_effectorPosition[1]<Y_Limit)
                                     {
```

Appendix C (Continued)

```
                                    }
        else if(m_effectorPosition[1]>Y_Limit)
                                    {

          m_forceOnVisitor.set(0.0,0.0,0.0);


                                    }


                        }


  }
  else if(m_effectorPosition[0]<vectorPoint[0])
  {

     m_forceOnVisitor[0]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);
     m_forceOnVisitor[1]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);
     m_forceOnVisitor[2]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);

  }
#endif

/*************************************************************************
************
Test 4 with just the high spring force on the trajectory without any projection assistance
*************************************************************************
****************/
#ifdef EXP4:
if(m_currentDistance > 130.0)
  {
                    m_forceOnVisitor[0]=m_forceOnVisitor[1]=m_forceOnVisitor[2]=0.0;
  }
  //else if((m_currentDistance>10.0)&&(m_currentDistance10.0))
  else if(m_effectorPosition[0]>vectorPoint[0])
  {
                     if(m_effectorPosition[0]<(vectorPoint[0]+20.0))
                     {
```

```
if(m_effectorPosition[0]<(vectorPoint[0]+10.0))
```

Appendix C (Continued)

```
{

if(m_effectorPosition[0]<(vectorPoint[0]+5.0))

{

if(m_effectorPosition[1]<Y_End)
{
m_forceOnVisitor[0]=
19.0*STIFFNESS*Dist*n_vector[0];
m_forceOnVisitor[1]=
19.0*STIFFNESS*Dist*n_vector[1];
m_forceOnVisitor[2]=
19.0*STIFFNESS*Dist*n_vector[2];

}
else
{
m_forceOnVisitor.set(0.0,0.0,0.0);

}

//printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
}
else
{
if(m_effectorPosition[1]<Y_Limit)
{
```

Appendix C (Continued)

```
            m_forceOnVisitor[0]=
0.1*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);

                                        m_forceOnVisitor[1]=
0.8*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);

            m_forceOnVisitor[2]=
scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);

                                                  }
                                                }
                                    }

                                    else
                                    {
                                      if(m_effectorPosition[1]<Y_Limit)
                                      {


          m_forceOnVisitor.set(-1.2*exp(-mDist/70.0), 1.2*exp(-mDist/70.0), 0.0);
//m_forceOnVisitor.set(0.0,0.0,0.0);

                                      }
        else if(m_effectorPosition[1]>Y_Limit)
                                      {
            m_forceOnVisitor.set(0.0,0.0,0.0);


                                      }


                                    }
                        }
                        else
                        {
      if(m_effectorPosition[1]<Y_Limit)
                                     {
```

Appendix C (Continued)

```
                    //m_forceOnVisitor.set(0.0,0.0,0.0);
                                        }
        else if(m_effectorPosition[1]>Y_Limit)
                                        {

          m_forceOnVisitor.set(0.0,0.0,0.0);


                                        }


                    }


  }
  else if(m_effectorPosition[0]<vectorPoint[0])
  {

     m_forceOnVisitor[0]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);
     m_forceOnVisitor[1]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);
     m_forceOnVisitor[2]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);

  }
#endif
/**************************************************************************
************
Test 5 with just the projection component on the trajectory without the high spring
force/User
Controlled Velocity based force scaling.
**************************************************************************
***************/
//#ifdef EXP5:
if (m_gIDMenu == 3)

{
  if(m_currentDistance > 130.0)
  {
```

Appendix C (Continued)

```
                m_forceOnVisitor[0]=m_forceOnVisitor[1]=m_forceOnVisitor[2]=0.0;
 }
 //else if((m_currentDistance>10.0)&&(m_currentDistance10.0))
 else if(m_effectorPosition[0]>vectorPoint[0])
 {
    //printf("%3.6f %3.6f\n",m_effectorPosition[0],vectorPoint[0]);
                  if(m_effectorPosition[0]<(vectorPoint[0]+20.0))
                  {
                              if(m_effectorPosition[0]<(vectorPoint[0]+10.0))
                              {

                 if(m_effectorPosition[0]<(vectorPoint[0]+8.0))
                                          {

          if(m_effectorPosition[1]<Y_Limit)
                                            {
                                                        m_forceOnVisitor[0]=
18.0*STIFFNESS*Dist*n_vector[0];
                                                        m_forceOnVisitor[1]=
18.0*STIFFNESS*Dist*n_vector[1];
                                                        m_forceOnVisitor[2]=
18.0*STIFFNESS*Dist*n_vector[2];
                                                 //printf("%3.6f
%3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
                                                        //printf("%3.6f
%3.6f\n",m_effectorPosition[0],vectorPoint[0]);
                                                }
                                                else
                                                {
            m_forceOnVisitor[0]=
scaleForceFactor*(5.0*STIFFNESS*Dist*n_vector[0])+(0.002*TrajVec[0]);

                                                        m_forceOnVisitor[1]=
scaleForceFactor*(5.0*STIFFNESS*Dist*n_vector[1])+(0.002*TrajVec[1]);

                                                        m_forceOnVisitor[2]=
scaleForceFactor*(5.0*STIFFNESS*Dist*n_vector[2])+(0.002*TrajVec[2]);
             //printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
              //printf("%3.6f %3.6f\n",m_effectorPosition[0],vectorPoint[0]);
                    }
```

```
        //printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
                                                }
                                                else
                                                {
        if(m_effectorPosition[1]<Y_Limit)


                                                  {

                                                    m_forceOnVisitor.set(-1.5*exp(-
mDist/70.0), 1.2*exp(-mDist/70.0), 0.0);
                                                                  //printf("%3.6f
%3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
            //printf("%3.6f %3.6f\n",m_effectorPosition[0],vectorPoint[0]);
                                                  }

                                                  else
if((m_effectorPosition[1]>Y_Limit)&&(m_effectorPosition[1]<Y_End))

                                                  {


            m_forceOnVisitor[0]=
0.1*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);

                                        m_forceOnVisitor[1]=
0.8*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);

            m_forceOnVisitor[2]=
scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);
            //printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
            //printf("%3.6f %3.6f\n",m_effectorPosition[0],vectorPoint[0]);

                                                  }
                                                }
                                    }
```

Appendix C (Continued)

```
                                    if(m_effectorPosition[1]<Y_Limit)
                                    {

          m_forceOnVisitor.set(-1.5*exp(-mDist/70.0), 1.2*exp(-mDist/70.0), 0.0);
//m_forceOnVisitor.set(0.0,0.0,0.0);
          //printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
          //printf("%3.6f %3.6f\n",m_effectorPosition[0],vectorPoint[0]);

                                    }
        else if(m_effectorPosition[1]>Y_Limit)
                                    {
            m_forceOnVisitor.set(0.0,0.0,0.0);
                                                    //printf("%3.6f
%3.6f\n",m_effectorPosition[0],vectorPoint[0]);

                                    }


                                   }
                    }
                    else
                    {
       if(m_effectorPosition[1]<Y_Limit)
                                    {



                                     m_forceOnVisitor.set(-1.5*exp(-mDist/70.0), 1.2*exp(-
mDist/70.0), 0.0);

                   //m_forceOnVisitor.set(0.0,0.0,0.0);
                                    }
       else if(m_effectorPosition[1]>Y_Limit)
                                    {

         m_forceOnVisitor.set(0.0,0.0,0.0);


                                    }
```

Appendix C (Continued)

```
 }
 else if(m_effectorPosition[0]<vectorPoint[0])
 {

    m_forceOnVisitor[0]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);
    m_forceOnVisitor[1]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);
    m_forceOnVisitor[2]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);

 }

}
//#endif

/**************************************************************************
************
Test 6 with just the projection component on the trajectory without the high spring force.
**************************************************************************
***************/
//#ifdef EXP6:
if (m_gIDMenu == 4)//velocity scaling assistance//
{

 //code for USER CONTROLLED SCALING//

 if (velocityMag<0.0)
 {
     velocityMag= -(velocityMag);

 }
  if (velocityMag<m_VelocityLimit)
  {
     printf("%3.6f %3.6f\n",velocityMag,m_VelocityLimit);
                    //When the user is travelling at a velocity which is greater value than
90% of the
                     //velocity limit
                      if (velocityMag>(m_VelocityLimit*0.09))

                                        {
        Proj_Scale = 0.002;
        yExp_Scale = 1.1;
```

```
                xExp_Scale = -1.0;
                        }
              else
                        {
//When the user is travelling at a velocity which is greater than 80% of the
                //velocity limit
 if (velocityMag>(m_VelocityLimit*0.08))
                                        {
   //Proj_Scale = 0.001;
     Proj_Scale = 0.002;
     yExp_Scale = 1.1;
                //xExp_Scale = -1.3;
     xExp_Scale = -2.3;
                        }
                        else
                        {

 //When the user is travelling at a velocity which is greater than 70% of the
                //velocity limit
 if (velocityMag>(m_VelocityLimit*0.05))
                                        {
  //Proj_Scale = 0.0015;
 Proj_Scale = 0.002;
  yExp_Scale = 1.1;
                //xExp_Scale = -1.5;
    xExp_Scale = -3.5;
                                        }
                                        else
                                        {

   if (velocityMag>(m_VelocityLimit*0.009))
                                          {
      Proj_Scale = 0.002;
      yExp_Scale = 1.1;
                //xExp_Scale = -2.0;
      xExp_Scale = -4.5;
                                          }
                                          else
                                          {
```

Appendix C (Continued)

```
                Proj_Scale = 0.002;
                 yExp_Scale = 1.2;
                                //xExp_Scale = -2.5;
                  xExp_Scale = -5.5;
                                                        }
                                                      }

                                        }



                            }

 }
 else
 {
   //When the user had crossed the velocity limit//
   //fprintf(stdout,"The user has crossed the velocity limit");

                            Proj_Scale = 0.0;
    yExp_Scale = 0.0;
                            xExp_Scale = 0.0;
 }


 //fprintf(stdout,"%3.3f %3.3f mm/s\n", velocityMag,m_VelocityLimit);

if(m_currentDistance > 130.0)
  {
                            m_forceOnVisitor[0]=m_forceOnVisitor[1]=m_forceOnVisitor[2]=0.0;
  }
  //else if((m_currentDistance>10.0)&&(m_currentDistance10.0))
  else if(m_effectorPosition[0]>vectorPoint[0])
  {
                             if(m_effectorPosition[0]<(vectorPoint[0]+20.0))
                             {
                                                if(m_effectorPosition[0]<(vectorPoint[0]+10.0))
                                                {

                           if(m_effectorPosition[0]<(vectorPoint[0]+8.0))
```

Appendix C (Continued)

```
        if(m_effectorPosition[1]<Y_Limit)
                                                {
                                                        m_forceOnVisitor[0]=
18.0*STIFFNESS*Dist*n_vector[0];
                                                        m_forceOnVisitor[1]=
18.0*STIFFNESS*Dist*n_vector[1];
                                                        m_forceOnVisitor[2]=
18.0*STIFFNESS*Dist*n_vector[2];


                                                }
                                                else
                                                {

          m_forceOnVisitor[0]=
scaleForceFactor*(5.0*STIFFNESS*Dist*n_vector[0])+(Proj_Scale*TrajVec[0]);

                                                        m_forceOnVisitor[1]=
scaleForceFactor*(5.0*STIFFNESS*Dist*n_vector[1])+(Proj_Scale*TrajVec[1]);
```

Appendix C (Continued)

```
                                                        m_forceOnVisitor[2]=
scaleForceFactor*(5.0*STIFFNESS*Dist*n_vector[2])+(Proj_Scale*TrajVec[2]);


                                                }



        //printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
                                                }
                                                else
                                                {
        if(m_effectorPosition[1]<Y_Limit)
                                                 {
```

Appendix C (Continued)

```
                                                    m_forceOnVisitor.set(xExp_Scale*exp(-
mDist/70.0), yExp_Scale*exp(-mDist/70.0), 0.0);




                                                }

                                                else
if((m_effectorPosition[1]>Y_Limit)&&(m_effectorPosition[1]<Y_End))

                                                {


            m_forceOnVisitor[0]=
0.1*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);

                                        m_forceOnVisitor[1]=
0.8*scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);

            m_forceOnVisitor[2]=
scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);

                                                }
                                               }
                                    }

                                    else
                                    {
                                      if(m_effectorPosition[1]<Y_Limit)
                                      {

          m_forceOnVisitor.set(xExp_Scale*exp(-mDist/70.0), yExp_Scale*exp(-
mDist/70.0), 0.0);          //m_forceOnVisitor.set(0.0,0.0,0.0);

                                      }
        else if(m_effectorPosition[1]>Y_Limit)
                                      {
```

Appendix C (Continued)

```
                                    }


                                   }

                    }
                    else
                    {
        if(m_effectorPosition[1]<Y_Limit)
                                   {


                                    m_forceOnVisitor.set(xExp_Scale*exp(-mDist/70.0),
yExp_Scale*exp(-mDist/70.0), 0.0);

                   //m_forceOnVisitor.set(0.0,0.0,0.0);
                                   }
        else if(m_effectorPosition[1]>Y_Limit)
                                   {

          m_forceOnVisitor.set(0.0,0.0,0.0);


                                   }


                    }


  }
  else if(m_effectorPosition[0]<vectorPoint[0])
  {
     m_forceOnVisitor[0]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[0]);
     m_forceOnVisitor[1]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[1]);
     m_forceOnVisitor[2]= scaleForceFactor*(15.0*STIFFNESS*Dist*n_vector[2]);

  }
```

Appendix C (Continued)

```
//#endif
/**************************************************************************
***************Test 7:Fitt's Task/To determine the time taken to execute the task,
with assistance and

**************************************************************************
*****************/
#ifdef EXP7:   double scale=0.1;  double Proj_Scale=0.08;


  double y_limit=10.0;

  Traj_left[0] = ((pointend.x+10.0)-(pointend.x+40.0));

  Traj_left[1] = ((pointend.y+10.0)-(pointend.y+40.0));

  Traj_left[2] = 0.0;


  Traj_right[0] = ((pointstart.x-10.0)-(pointstart.x-40.0));

  Traj_right[1] = ((pointstart.y-10.0)-(pointstart.y-40.0));

  Traj_right[2] = 0.0;

  Fitts'_Start[0] =pointstart.x;
  Fitts'_Start[1]= pointstart.y;
  Fitts'_Start[2]= pointstart.z;

  Fitts'_End[0]=pointend.x;
  Fitts_End[1]= pointend.y;
  Fitts_End[2]= pointend.z;

  //Trajectory vector between the two goal points//
  Fitts_Delta[0]=Fitts_End[0]-Fitts_Start[0];

  Fitts_Delta[1]=Fitts_End[1]-Fitts_Start[1];
```

```
 GoalDist= Fitts_Delta.magnitude();
 //Normal vector along the trajectory path
 /*g_vector[0]=Fitts_Delta[0]/GoalDist;
 g_vector[1]=Fitts_Delta[1]/GoalDist;
 g_vector[2]=Fitts_Delta[2]/GoalDist;*/

 //Distance between effector position and left goal point

 Dist_Goalleft[0]= Fitts_Start[0]-m_effectorPosition[0];
 Dist_Goalleft[1]= Fitts_Start[1]-m_effectorPosition[1];
 Dist_Goalleft[2]= Fitts_Start[2]- m_effectorPosition[2];



g_vector[0]=Dist_Goalleft[0]/Dist_left;
 g_vector[1]=Dist_Goalleft[1]/Dist_left;
 g_vector[2]=Dist_Goalleft[2]/Dist_left;


 Dist_left=  Dist_Goalleft.magnitude();

 //Distance between effector position right goal point



 Dist_Goalright[0]= Fitts_End[0]-m_effectorPosition[0];
 Dist_Goalright[1]=Fitts_End[1]-m_effectorPosition[1];
 Dist_Goalright[2]= Fitts_End[2]-m_effectorPosition[2];
 Dist_right=Dist_Goalright.magnitude();


//if((m_effectorPosition[0]>Fitts_Start[0])&&(m_effectorPosition[0]<Fitts_Start[0]+40.0
)&&(m_effectorPosition[1]>Fitts_Start[1]-
30.0)&&(m_effectorPosition[1]<(Fitts_Start[1]+30.0)))
   if(m_effectorPosition[1]>(Fitts_Start[1]-10.0))
                {
     if(m_effectorPosition[1]<(Fitts_Start[1]+10.0))
                  {
```

Appendix C (Continued)

```
            //printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
                                                }

                                                }
                                                else
                                                {

                  if(m_effectorPosition[0]>(Fitts_End[0]-130.0))
                                                                {

                  if(m_effectorPosition[0]>(Fitts_End[0]-40.0))
                                                                                {
                 //m_forceOnVisitor.set(-0.7*exp(-Dist_right/70.0),0.7*exp(-
Dist_right/70.0),0.0);

                  m_forceOnVisitor.set(0.09*exp(-Dist_right/70.0),0.1*exp(-
Dist_right/70.0),0.0);
                  //printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);

                                                                                }
                                                                else
                                                                                {


                 m_forceOnVisitor.set(2.0*exp(-Dist_right/70.0), 0.1*exp(-Dist_right/70.0),
0.0);
                  //printf("%3.6f %3.6f\n",m_forceOnVisitor[0],m_forceOnVisitor[1]);
                                                                                }

                                                                }
                                                                Else


                                                                {
               m_forceOnVisitor.set(0.0,0.0,0.0);
                                                                }

                                                }
                                }
```

Appendix C (Continued)

```
                    {
        m_forceOnVisitor.set(0.0,0.0,0.0);
                    }
                }
                else
                {
     m_forceOnVisitor.set(0.0,0.0,0.0);
                }
#endif

/*****************************************************************************
**************************
Test 8:FITTS TASK/The execution of FITTS TASK without any kind of assistance
provided to the user
*****************************************************************************
**************************/
//#ifdef EXP8:
if (m_gIDMenu == 5)//no assistance provided
{
                m_forceOnVisitor.set(0.0,0.0,0.0);
                printf("%3.6f %3.6f
%3.6f\n",m_effectorPosition[0],m_effectorPosition[1],m_effectorPosition[2]);

}
//#endif
/*****************************************************************************
**************************
Test 9:FITTS TASK/The execution of FITTS TASK with assistance provided to the user
*****************************************************************************
**************************/
//#ifdef EXP9:
if (m_gIDMenu == 0)
{

  double scale=0.1;
  double Proj_Scale=0.08;
  double y_limit=10.0;
  bool OneTrip = false;
  //HDdouble m_PreX = 0.0;
```

```
Traj_left[0] = ((pointend.x+10.0)-(pointend.x+40.0));

Traj_left[1] = ((pointend.y+10.0)-(pointend.y+40.0));

Traj_left[2] = 0.0;


Traj_right[0] = ((pointstart.x-10.0)-(pointstart.x-40.0));

Traj_right[1] = ((pointstart.y-10.0)-(pointstart.y-40.0));

Traj_right[2] = 0.0;

Fitts_Start[0] =pointstart.x;
Fitts_Start[1]= pointstart.y;
Fitts_Start[2]= pointstart.z;

Fitts_End[0]=pointend.x;
Fitts_End[1]= pointend.y;
Fitts_End[2]= pointend.z;

//Trajectory vector between the two goal points//
Fitts_Delta[0]=Fitts_End[0]-Fitts_Start[0];

Fitts_Delta[1]=Fitts_End[1]-Fitts_Start[1];

Fitts_Delta[2]=Fitts_End[2]-Fitts_Start[2];

GoalDist= Fitts_Delta.magnitude();
//Normal vector along the trajectory path
/*g_vector[0]=Fitts_Delta[0]/GoalDist;
g_vector[1]=Fitts_Delta[1]/GoalDist;
g_vector[2]=Fitts_Delta[2]/GoalDist;*/

//Distance between effector position and left goal point

Dist_Goalleft[0]= Fitts_Start[0]-m_effectorPosition[0];
Dist_Goalleft[1]= Fitts_Start[1]-m_effectorPosition[1];
Dist_Goalleft[2]= Fitts_Start[2]- m_effectorPosition[2];
```

```
g_vector[0]=Dist_Goalleft[0]/Dist_left;
g_vector[1]=Dist_Goalleft[1]/Dist_left;
g_vector[2]=Dist_Goalleft[2]/Dist_left;


Dist_left=  Dist_Goalleft.magnitude();

//Distance between effector position right goal point
Dist_Goalright[0]= Fitts_End[0]-m_effectorPosition[0];

Dist_Goalright[1]=Fitts_End[1]-m_effectorPosition[1];
Dist_Goalright[2]= Fitts_End[2]-m_effectorPosition[2];
Dist_right=Dist_Goalright.magnitude();


/*if (firstTrip==true) {
                    firstTrip = false;
                    Previous_Xcoordinate[0]= 0.0;
                   }*/



if(m_effectorPosition[1]>(Fitts_Start[1]-10.0))
                  {
  if(m_effectorPosition[1]<(Fitts_Start[1]+10.0))
                   {

    if(m_effectorPosition[0]<m_PreX[0])
                     {
       if(m_effectorPosition[0]<(Fitts_Start[0]+180.0))
                                    {

                                       //printf("%3.6f
%3.6f\n",m_effectorPosition[0],m_PreX[0]);
          if(m_effectorPosition[0]<(Fitts_Start[0]+135.0))
                                                  {
            if(m_effectorPosition[0]<(Fitts_Start[0]+40.0))
                                                                {
```

```
            //m_forceOnVisitor.set(0.0,0.0,0.0);
                                                    //m_forceOnVisitor.set(-0.7*exp(-
Dist_left/70.0),0.7*exp(-Dist_left/70.0),0.0);
                                                                m_forceOnVisitor.set(-
0.6*exp(-Dist_left/70.0),0.0,0.0);


                                                                }
                        else
                                                                {
             //m_forceOnVisitor.set(0.0,0.0,0.0);
                                        //m_forceOnVisitor.set(-2.0*exp(-Dist_left/70.0),
1.2*exp(-Dist_left/70.0), 0.0);
             m_forceOnVisitor.set(-1.8*exp(-Dist_left/70.0),0.0, 0.0);


                                                                }

                                                    }
                                    }
                                    else
                                    {
          if(m_effectorPosition[0]>(Fitts_Start[0]+180.0))
                                                    {
                                                                //printf("%3.6f
%3.6f\n",m_effectorPosition[0],m_PreX[0]);
                                     Appendix C (Continued)

            if(m_effectorPosition[0]>(Fitts_End[0]-135.0))
                                                                {
              if(m_effectorPosition[0]>(Fitts_End[0]-40.0))
                                                                  {
                 //m_forceOnVisitor.set(0.0,0.0,0.0);
                                                    //m_forceOnVisitor.set(-0.7*exp(-
Dist_right/70.0),0.8*exp(-Dist_right/70.0),0.0);

m_forceOnVisitor.set(-0.6*exp(-Dist_right/70.0),0.0,0.0);
                                                      //printf("%3.6f
%3.6f\n",Previous_Xcoordinate[0]);
                                                                  }
```

```
                //m_forceOnVisitor.set(0.0,0.0,0.0);
                                        //m_forceOnVisitor.set(-2.0*exp(-
Dist_right/70.0),1.2*exp(-Dist_right/70.0), 0.0);
                 m_forceOnVisitor.set(-1.8*exp(-Dist_right/70.0),0.0, 0.0);
                                        // m_forceOnVisitor.set(-1.8*exp(-
Dist_right/70.0),1.2*exp(-Dist_right/70.0), 0.0);
                 //printf("%3.6f\n",Previous_Xcoordinate[0]);
                                                                }

                                                              }

                                                    }

                                  }
                   }


                   else
                   {
                                 if(m_effectorPosition[0]> m_PreX[0])
                                 {
        if(m_effectorPosition[0]<(Fitts_Start[0]+180.0))
                                   {  //printf("%3.6f
%3.6f\n",m_PreX[0],m_effectorPosition[0]);
          if(m_effectorPosition[0]<(Fitts_Start[0]+135.0))
                                                              {
            if(m_effectorPosition[0]<(Fitts_Start[0]+40.0))
                                                                {
              //m_forceOnVisitor.set(0.0,0.0,0.0);
                                        //m_forceOnVisitor.set(0.5*exp(-
Dist_left/70.0),0.7*exp(-Dist_left/70.0),0.0);

m_forceOnVisitor.set(0.6*exp(-Dist_left/70.0),0.0,0.0);
                //Previous_Xcoordinate[0]= m_effectorPosition[0];
                                                                }
                                 else
                                                                {
                 //m_forceOnVisitor.set(0.0,0.0,0.0);
                                        //m_forceOnVisitor.set(2.0*exp(-Dist_left/70.0),
1.2*exp(-Dist_left/70.0), 0.0);
```

```
                                                                    }

                                                                  }
                                    }
                                    else
                                    {
            if(m_effectorPosition[0]>(Fitts_Start[0]+180.0))
                                                                  {          //printf("%3.6f
%3.6f\n",m_PreX[0],m_effectorPosition[0]);
                  //if(m_effectorPosition[0]<(Fitts_Start[0]+135.0))
                  if(m_effectorPosition[0]>(Fitts_End[0]-135.0))
                                                                                {
                    if(m_effectorPosition[0]>(Fitts_End[0]-40.0))

                    {
                      //m_forceOnVisitor.set(0.0,0.0,0.0);
                                          //m_forceOnVisitor.set(0.5*exp(-
Dist_right/70.0),0.6*exp(-Dist_right/70.0),0.0);

m_forceOnVisitor.set(0.6*exp(-Dist_right/70.0),0.0,0.0);
                      //Previous_Xcoordinate[0]= m_effectorPosition[0];

                    }
                                else

                    {
                      // m_forceOnVisitor.set(0.0,0.0,0.0);
                                          //m_forceOnVisitor.set(2.0*exp(-
Dist_right/70.0), 0.8*exp(-Dist_right/70.0), 0.0);
                      m_forceOnVisitor.set(1.8*exp(-Dist_right/70.0), 0.0, 0.0);
                     //Previous_Xcoordinate[0]= m_effectorPosition[0];
                     //printf("%3.6f\n",m_PreX);

                    }
                                                                                }
                                                                  }
                                    }
                                  }
                     }                          }
   else
```

Appendix C (Continued)

```
 {
   m_forceOnVisitor.set(0.0,0.0,0.0);
 }

}


//Previous_Xcoordinate[0]= m_effectorPosition[0];
//#endif

/*************************************************************************
*************************
Test 10:Fitts' Task/The execution of Fitts' Task implemented in the Y direction with
assistance
*************************************************************************
*************************/

//#ifdef EXP10:
if (m_gIDMenu == 1)
{
  double scale=0.1;
  double Proj_Scale=0.08;
  double y_limit=10.0;
  bool OneTrip = false;
  //HDdouble m_PreX = 0.0;

  Traj_left[0] = ((pointend.x+10.0)-(pointend.x+40.0));

  Traj_left[1] = ((pointend.y+10.0)-(pointend.y+40.0));

  Traj_left[2] = 0.0;


  Traj_right[0] = ((pointstart.x-10.0)-(pointstart.x-40.0));

  Traj_right[1] = ((pointstart.y-10.0)-(pointstart.y-40.0));

  Traj_right[2] = 0.0;
```

```
Fitts_Start[2]= pointstart.z;

Fitts_End[0]=pointend.x;
Fitts_End[1]= pointend.y;
Fitts_End[2]= pointend.z;

//Trajectory vector between the two goal points//
Fitts_Delta[0]=Fitts_End[0]-Fitts_Start[0];

Fitts_Delta[1]=Fitts_End[1]-Fitts_Start[1];

Fitts_Delta[2]=Fitts_End[2]-Fitts_Start[2];
GoalDist= Fitts_Delta.magnitude();
//Normal vector along the trajectory path
/*g_vector[0]=Fitts_Delta[0]/GoalDist;
g_vector[1]=Fitts_Delta[1]/GoalDist;
g_vector[2]=Fitts_Delta[2]/GoalDist;*/

//Distance between effector position and left goal point

Dist_Goalleft[0]= Fitts_Start[0]-m_effectorPosition[0];
Dist_Goalleft[1]= Fitts_Start[1]-m_effectorPosition[1];
Dist_Goalleft[2]= Fitts_Start[2]- m_effectorPosition[2];

g_vector[0]=Dist_Goalleft[0]/Dist_left;
g_vector[1]=Dist_Goalleft[1]/Dist_left;
g_vector[2]=Dist_Goalleft[2]/Dist_left;



Dist_left=  Dist_Goalleft.magnitude();

//Distance between effector position right goal point
Dist_Goalright[0]= Fitts_End[0]-m_effectorPosition[0];
Dist_Goalright[1]=Fitts_End[1]-m_effectorPosition[1];
Dist_Goalright[2]= Fitts_End[2]-m_effectorPosition[2];
Dist_right=Dist_Goalright.magnitude();
```

Appendix C (Continued)

```
                    Previous_Xcoordinate[0]= 0.0;
                  }*/


 if(m_effectorPosition[0]>(Fitts_Start[0]-5.0))
                  {
   if(m_effectorPosition[0]<(Fitts_Start[0]+5.0))
                   {

     if(m_effectorPosition[1]<m_PreX[1])
                     {

        if(m_effectorPosition[1]<(Fitts_Start[1]+175.0))
                                    {
          //printf("%3.6f %3.6f\n",m_effectorPosition[1],m_PreX[1]);

                                        //printf("%3.6f
%3.6f\n",m_effectorPosition[0],m_PreX[0]);
           if(m_effectorPosition[1]<(Fitts_Start[1]+ 155.0))
                                                  {
            //printf("%3.6f %3.6f\n",m_effectorPosition[1],m_PreX[1]);
            if(m_effectorPosition[1]<(Fitts_Start[1]+20.0))
                                                                {
              //m_forceOnVisitor.set(0.0,0.0,0.0);
                                                    //m_forceOnVisitor.set(-0.05*exp(-
Dist_left/70.0),-0.5*exp(-Dist_left/70.0),0.0);

               m_forceOnVisitor.set(0.0,-0.5*exp(-Dist_left/70.0),0.0);
                                                      //printf("%3.6f
%3.6f\n",m_effectorPosition[1],m_PreX[1]);
                                                                } else
                                                                {

//m_forceOnVisitor.set(0.0,0.0, 0.0);
                                        //m_forceOnVisitor.set(-0.1*exp(-Dist_left/70.0),-
2.0*exp(-Dist_left/70.0), 0.0);
                m_forceOnVisitor.set(0.0,-2.0*exp(-Dist_left/70.0), 0.0);
                //printf("%3.6f\n",Previous_Xcoordinate[0]);
                                                                }
```

```
            m_forceOnVisitor.set(0.0,0.0,0.0);
                                                    }
                                    }
                                    else
                                    {
                    if(m_effectorPosition[1]>(Fitts_Start[1]+175.0))
                                                    {
                                                                    //printf("%3.6f
%3.6f\n",m_effectorPosition[0],m_PreX[0]);

            if(m_effectorPosition[1]>(Fitts_End[1]-90.0))
                                                                    {
              //printf("%3.6f %3.6f\n",m_effectorPosition[0],m_PreX[0]);
              if(m_effectorPosition[1]>(Fitts_End[1]-20.0))
                                                                      {
                //m_forceOnVisitor.set(0.0,0.0,0.0);
                                                                  //m_forceOnVisitor.set(0.05*exp(-
Dist_right/70.0),-0.5*exp(-Dist_right/70.0),0.0);

m_forceOnVisitor.set(0.0,-0.5*exp(-Dist_right/70.0),0.0);
                                                                  //printf("%3.6f
%3.6f\n",Previous_Xcoordinate[0]);
                                                                      }
                                    else
                                                                      {
                 //m_forceOnVisitor.set(0.0,0.0,0.0);
                                                  //m_forceOnVisitor.set(0.1*exp(-Dist_right/70.0),-
2.0*exp(-Dist_right/70.0), 0.0);
                m_forceOnVisitor.set(0.0,-2.0*exp(-Dist_right/70.0), 0.0);
                //printf("%3.6f\n",Previous_Xcoordinate[0]);
                                                                      }

                                                                    }

                                                    }
                                                    else
                                                    {
            m_forceOnVisitor.set(0.0,0.0,0.0);
                                                    }
```

```
                {
                                if(m_effectorPosition[1]> m_PreX[1])
                                {
                                  if(m_effectorPosition[1]> Fitts_Start[1])
                                  {
        //printf("%3.6f %3.6f\n",m_PreX[0],m_effectorPosition[0]);
          if(m_effectorPosition[1]<(Fitts_Start[1]+175.0))
                                                { //printf("%3.6f
%3.6f\n",m_PreX[0],m_effectorPosition[0]);
            //printf("%3.6f %3.6f\n",m_effectorPosition[1],Fitts_Start[1]+100.0);
           if(m_effectorPosition[1]<(Fitts_Start[1]+155.0))
                                                                {
                                                                  //printf("%3.6f
%3.6f\n",m_PreX[0],m_effectorPosition[0]);
              if(m_effectorPosition[1]<(Fitts_Start[1]+20.0))
                                                                  {
                //m_forceOnVisitor.set(0.0,0.0,0.0);
                //printf("%3.6f %3.6f\n",m_PreX[0],m_effectorPosition[0]);
                                        //m_forceOnVisitor.set(0.05*exp(-
Dist_left/70.0),0.9*exp(-Dist_left/70.0),0.0);

m_forceOnVisitor.set(0.0,0.9*exp(-Dist_left/70.0),0.0);
                //Previous_Xcoordinate[0]= m_effectorPosition[0];

//m_forceOnVisitor.set(-1.0*exp(-Dist_left/70.0), 2.0*exp(-Dist_left/70.0), 0.0);
                                                                  }
                                  else                                                  {
                //m_forceOnVisitor.set(0.0,0.0,0.0);
                //printf("%3.6f %3.6f\n",m_PreX[0],m_effectorPosition[0]);
                                        //m_forceOnVisitor.set(0.1*exp(-Dist_left/70.0),
2.0*exp(-Dist_left/70.0), 0.0);
              m_forceOnVisitor.set(0.0, 2.0*exp(-Dist_left/70.0), 0.0);
              //Previous_Xcoordinate[0]= m_effectorPosition[0];
              //printf("%3.6f\n",m_PreX);

                                                                  }

                                                                }
                                                        else
                                                                {
```

Appendix C (Continued)

```
                                else
                                {
            if(m_effectorPosition[1]>(Fitts_Start[1]+175.0))
                                                                {     //printf("%3.6f
%3.6f\n",m_PreX[0],m_effectorPosition[0]);
                  //if(m_effectorPosition[0]<(Fitts_Start[0]+135.0))
                  if(m_effectorPosition[1]>(Fitts_End[1]-90.0))
                                                                        {
                    //printf("%3.6f %3.6f\n",m_PreX[0],m_effectorPosition[0]);
                    if(m_effectorPosition[1]>(Fitts_End[1]-20.0))

                    {
                      //m_forceOnVisitor.set(0.0,0.0,0.0);
                      //printf("%3.6f %3.6f\n",m_PreX[0],m_effectorPosition[0]);


                                                //m_forceOnVisitor.set(0.05*exp(-
Dist_right/70.0),0.9*exp(-Dist_right/70.0),0.0);

m_forceOnVisitor.set(0.0,0.9*exp(-Dist_right/70.0),0.0);
                      //Previous_Xcoordinate[0]= m_effectorPosition[0];

                    }
                                else

                    {
                      //m_forceOnVisitor.set(0.0,0.0,0.0);
                     //printf("%3.6f %3.6f\n",m_PreX[0],m_effectorPosition[0]);
                                                //m_forceOnVisitor.set(0.1*exp(-
Dist_right/70.0), 2.0*exp(-Dist_right/70.0), 0.0);
                       m_forceOnVisitor.set(0.0, 2.0*exp(-Dist_right/70.0), 0.0);
                     //Previous_Xcoordinate[0]= m_effectorPosition[0];
                     //printf("%3.6f\n",m_PreX);

                    }
                                                                        }
                                                                        else
                                                                        {
                  m_forceOnVisitor.set(0.0,0.0,0.0);
                                                                        }
```

Appendix C (Continued)

```
   }
   else
   {
     m_forceOnVisitor.set(0.0,0.0,0.0);
   }
  }
 else
 {
   m_forceOnVisitor.set(0.0,0.0,0.0);
 }
}
//#endif
/*****************************************************************************
**************************
Test 11:FITTS TASK/The execution of FITTS TASK implemented in the Z direction
with assistance
*****************************************************************************
**************************/
//#ifdef EXP11
if (m_gIDMenu == 2)

{
  double scale=0.1;
  double Proj_Scale=0.08;


  double y_limit=10.0;
  bool OneTrip = false;
  //HDdouble m_PreX = 0.0;

  Traj_left[0] = ((pointend.x+10.0)-(pointend.x+40.0));

  Traj_left[1] = ((pointend.y+10.0)-(pointend.y+40.0));

  Traj_left[2] = ((pointend.z+10.0)-(pointend.z+40.0));


  Traj_right[0] = ((pointstart.x-10.0)-(pointstart.x-40.0));
```

```
Traj_right[2] = ((pointstart.z-10.0)-(pointstart.z-40.0));

Fitts_Start[0] =pointstart.x;
Fitts_Start[1]= pointstart.y;
Fitts_Start[2]= pointstart.z;

Fitts_End[0]=pointend.x;
Fitts_End[1]= pointend.y;
Fitts_End[2]= pointend.z;

//Trajectory vector between the two goal points//
Fitts_Delta[0]=Fitts_End[0]-Fitts_Start[0];

Fitts_Delta[1]=Fitts_End[1]-Fitts_Start[1];

Fitts_Delta[2]=Fitts_End[2]-Fitts_Start[2];

GoalDist= Fitts_Delta.magnitude();
//Normal vector along the trajectory path
//g_vector[0]=Fitts_Delta[0]/GoalDist;
//g_vector[1]=Fitts_Delta[1]/GoalDist;
//g_vector[2]=Fitts_Delta[2]/GoalDist;

//Distance between effector position and left goal point

Dist_Goalleft[0]= Fitts_Start[0]-m_effectorPosition[0];
Dist_Goalleft[1]= Fitts_Start[1]-m_effectorPosition[1];
Dist_Goalleft[2]= Fitts_Start[2]- m_effectorPosition[2];
g_vector[0]=Dist_Goalleft[0]/Dist_left;
g_vector[1]=Dist_Goalleft[1]/Dist_left;
g_vector[2]=Dist_Goalleft[2]/Dist_left;



Dist_left=  Dist_Goalleft.magnitude();


//Distance between effector position right goal point
Dist_Goalright[0]= Fitts_End[0]-m_effectorPosition[0];
```

Appendix C (Continued)

```
  Mid_Point[0]= Fitts_Start[0]+((Fitts_End[0]-Fitts_Start[0])/2.0);
  Mid_Point[1]= Fitts_Start[1]+((Fitts_End[1]-Fitts_Start[1])/2.0);




  if((m_effectorPosition[0]>(Fitts_Start[0]-
50.0))&&(m_effectorPosition[0]<(Fitts_End[0]+50.0)))
                  {
      //printf("%3.6f %3.6f\n",m_effectorPosition[3],Fitts_End[0]);
                    if((m_effectorPosition[1]>(Fitts_Start[1]-
50.0))&&(m_effectorPosition[1]<(Fitts_End[1]+50.0)))
                    {
                                        //printf("%3.6f
%3.6f\n",m_effectorPosition[2],m_effectorPosition[2]);

                                            if(m_effectorPosition[2]> Mid_Point[2])
                                            //if((m_effectorPosition[0]<((Fitts_End[0]-
Fitts_Start[0])/(1.4)))&&(m_effectorPosition[1]<((Fitts_End[1]-Fitts_Start[1])/(1.4))))
                                                                {

                  //printf("%3.6f %3.6f\n",m_effectorPosition[0],m_effectorPosition[2]);

                     //printf("%3.6f %3.6f\n",m_effectorPosition[2],m_PreX[2]);

if(m_effectorPosition[2]>m_PreX[2])

{
                     m_forceOnVisitor.set(0.0,0.0,1.5*exp(-Dist_right/100.0));
                     //m_forceOnVisitor.set(0.0,0.0,0.0);
                     //printf("%3.6f %3.6f\n",m_effectorPosition[2],m_effectorPosition[2]);

}
                                            else if(m_effectorPosition[2]<m_PreX[2])

{
                      //m_forceOnVisitor.set(0.0,0.0,0.0);
                      m_forceOnVisitor.set(0.0,0.0,-1.3*exp(-Dist_right/70.0));
                      //printf("%3.6f %3.6f\n",m_effectorPosition[0],m_effectorPosition[1]);
```

```
                                                                {

if(m_effectorPosition[2]<(Mid_Point[2]-3.0))
                                                                        {
```

Appendix C (Continued)

```
                if(m_effectorPosition[2]<m_PreX[2])

        {
            //m_forceOnVisitor.set(0.0,0.0,0.0);

                        m_forceOnVisitor.set(0.0,0.0,-0.5*exp(-Dist_left/70.0));

            //m_forceOnVisitor.set(0.1*exp(-Dist_left/100.0),-0.38*exp(-
Dist_left/100.0),-0.5*exp(-Dist_left/70.0));
            //printf("%3.6f
%3.6f\n",m_effectorPosition[0],m_effectorPosition[1]);


        }
                                    else if(m_effectorPosition[2]>m_PreX[2])


        {

                        //m_forceOnVisitor.set(0.0,0.0,0.0);

             m_forceOnVisitor.set(0.0,0.0,0.7*exp(-Dist_left/70.0));
            //m_forceOnVisitor.set(-0.1*exp(-Dist_left/100.0),-0.4*exp(-
Dist_left/100.0),0.9*exp(-Dist_left/70.0));
            //printf("%3.6f
%3.6f\n",m_effectorPosition[0],m_effectorPosition[1]);

        }
                                                                        }
                                                                }

        }
        else
```

Appendix C (Continued)

```
    m_forceOnVisitor.set(0.0,0.0,0.0);
  }
 }

//#endif
}

/*******************************************************************************
 Gets the force on the visitor particle, given the current displacement.
*******************************************************************************/
hduVector3Dd ContactModel::GetCurrentForceOnVisitor()
{
      return m_forceOnVisitor;
}

/*******************************************************************************
 Retrieve the current contact point, i.e. the center of the visitor

 sphere.
*******************************************************************************/
hduVector3Dd ContactModel::GetCurrentContactPoint()
{

      return m_visitorPosition;
}

/*******************************************************************************
 Retrieve the current velocity of the end effector
*******************************************************************************/
hduVector3Dd ContactModel::GetCurrentEndEffectorVelocity()
{

      return m_velocityVec;
```

Appendix C (Continued)

```
/*****************************************************************************
********
 Retrieve the current velocity of the end effector
*****************************************************************************
*******/
hduVector3Dd ContactModel::GetLastVelocity()
{

    return m_lastvelocityVec;
}

/*****************************************************************************
********
 Retrieve the scheduler ticks
*****************************************************************************
*******/
HDdouble ContactModel::GetSchedulerTime()
{

    return schedulerTime;
}

/*****************************************************************************
********
 Retrieve the maximum velocity
*****************************************************************************
*******/
HDdouble ContactModel::GetMaxVel()
{

    return m_VelocityLimit;

}

/*****************************************************************************
*******
Retrieve the UDP socket option
*****************************************************************************
**********/
```

Appendix C (Continued)

```
}

/*******************************************************************************
Retrieve the Current Button State
*******************************************************************************/
HDint ContactModel::GetCurrentButtons()
{
        return m_CurrentButtonState;
}
/*******************************************************************************
Retrieve the Last Button State
*******************************************************************************/
HDint ContactModel::GetLastButtons()
{
        return m_LastButtonState;
}
/*******************************************************************************
Retrieve the MenuID
*******************************************************************************/
HDint ContactModel::GetIDMenu()
{
        return      m_gIDMenu;
}

/*******************************************************************************
 Save force components to a file
*******************************************************************************/

/*char *ContactModel::recordCallback(void *pUserData){
hduVector3Dd mforce;
hdGetDoublev(HD_CURRENT_FORCE, mforce);
```

Appendix C (Continued)

```
                return c;
}*/

//FILE *pFile =
//fopen("c:\\temp\\recordServoLoopData.txt","w");
//hdStartRecord(pFile,recordCallback,NULL,5000);


//MODIFIED//
/******************************************************************************
********
 Retrieve the current contact point, i.e. the center of the visitor
 sphere.
******************************************************************************
*******/

/*hduVector3Dd ContactModel::GetCurrentContactPoint()
{

      hlBeginFrame();
            hlCheckEvents();
      HLboolean buttDown = false;
            hlGetBooleanv(HL_BUTTON2_STATE,&buttDown);
               if (buttDown)
                        {
                                 fprintf(stdout,"button is down\n");
       return m_visitorPosition;
                        }

            hlEndFrame();
                        return m_zeroPosition;

}*/

//MODIFIED//
```

Appendix C (Continued)

```
*************************************************************************
*******/
hduVector3Dd ContactModel::Getlastpos()
{
      return m_lastpos;
}
/*************************************************************************
**********/
HDdouble ContactModel::GetinstRate()
{
      return  m_UpdateRate;
}
/*************************************************************************
*********
Retrieve the current (previous last x position)
*************************************************************************
*********/
/*HDdouble ContactModel::GetPrevious_Xcoordinate()
{
      return  m_PreX;
      printf("%3.6f\n",m_PreX[0]);
}*/

/*************************************************************************
**********
Udpate end effector velocity/velocity scaling
*************************************************************************
**********/

//////////////start here//////////////
void ContactModel::UpdateEndEffectorVelocity(const hduVector3Dd vel)
{
   m_velocityVec = vel;

   hduVector3Dd TrajVec;


                double SCALE = 2.0;
                double DotProduct;
```

Appendix C (Continued)

```
                velocityMag=m_velocityVec.magnitude();
   //fprintf(stdout,"V=%3.4f\n",velocityMag);


                TrajVec[0]=pointend.x-pointstart.x;
   TrajVec[1]=pointend.y-pointstart.y;
   TrajVec[2]=pointend.z-pointstart.z;

                //Projection of velocity vector in the direction of trajectory//
   DotProduct =
(m_velocityVec[0]*TrajVec[0])+(m_velocityVec[1]*TrajVec[1])+(m_velocityVec[2]*Tr
ajVec[2]);
   SquareTraj=(TrajVec.magnitude())*(TrajVec.magnitude());
                VProjected= (DotProduct/ SquareTraj)*TrajVec;
   //VScaled = VProjected;
   //fprintf(stdout,"Vx = %3.4f Vy = %3.4f Vz =
%3.4f",m_velocityVec[0],m_velocityVec[1],m_velocityVec[2], VScaled[0], VScaled[1],
VScaled[2]);


}

/**************************************************************************
**********
Udpate Last end effector velocity
**************************************************************************
**********/

//////////////start here//////////////
void ContactModel::UpdateLastEndEffectorVelocity(const hduVector3Dd lastvel)
{
   m_lastvelocityVec = lastvel;

}
/**************************************************************************
**************
Update the scheduler ticks
**************************************************************************
**************/
```

Appendix C (Continued)

```
}

/*************************************************************************
**************
Update the scheduler ticks
*************************************************************************
**************/
void ContactModel::UpdateRate(const double instRate){

  m_UpdateRate=instRate;

}

/*************************************************************************
********
 Constructor. Force model depends on relative position of two objects,
 a fixed and a visitor. Constructor initializes center positions, and it
 assumes that initially there is no contact. It also initializes radii
 for both spheres.
*************************************************************************
*******/

ContactModel::ContactModel(double fixedRadius,
                    const hduVector3Dd fixed,
                    double visitorRadius,
                    const hduVector3Dd visitor)
{
                                //allocate memory for bezier points
//                mpoints = (POINT_3D*)malloc(sizeof(POINT_3D)*(NP+1));

   m_fixedCenter = fixed;

   /* Intersection of two spheres is equivalent to intersection
      of a point and a sphere of effective radius (arms length)
      equal to the sum of the two radii. */
   m_armsLength = fixedRadius + visitorRadius;


                UpdateEffectorPosition(visitor);
```

Appendix C (Continued)

```
/*****************************************************************************
**********
Retrieve the UDP OPTION that is being clicked
*****************************************************************************
**********/
void ContactModel::UpdateUDP(const HDint option)
{
		m_option=option;
}
/*****************************************************************************
**********
Retrieve the UDP OPTION that is being clicked
*****************************************************************************
**********/
void ContactModel::UpdateVel_Limit(const double Vmax)
{
		m_VelocityLimit=Vmax;
}
/*****************************************************************************
**********
Retrieve the UDP OPTION that is being clicked
*****************************************************************************
**********/
void ContactModel::UpdatePrevious_Xcoordinate(const hduVector3Dd lastpos)
{
		m_PreX=lastpos;
   //printf("%3.6f\n",m_PreX[0]);

}
/*****************************************************************************
**********
Retrieve the current button state
*****************************************************************************
**********/
void ContactModel::UpdateButtonState(const HDint nCurrentButtons)
{
		m_CurrentButtonState=nCurrentButtons;
```

Appendix C (Continued)

```
/*******************************************************************************
**********
Retrieve the last button state that is being clicked
*******************************************************************************
**********/

void ContactModel::UpdateLastButtonState(const HDint nLastButtons)
{
		m_LastButtonState = nLastButtons;
}
/*******************************************************************************
**********
Retrieve the menu option that chooses the required experiment
*******************************************************************************
**********/

void ContactModel::UpdateIDMenu(const HDint gIDMenu)
{
		m_gIDMenu = gIDMenu;
}
```