Fall 11-11-2010

Virtual Dynamic Tunnel: A Target-Agnostic Assistive User Interface Algorithm for Head-Operated Input Devices

Ferrol R. Blackmon
Georgia State University

VIRTUAL DYNAMIC TUNNEL: A TARGET-AGNOSTIC ASSISTIVE USER INTERFACE ALGORITHM FOR HEAD-OPERATED INPUT DEVICES

by

FERROL RAWSON BLACKMON

Under the Direction of Dr. Michael Weeks

ABSTRACT

Today the effective use of computers (e.g. those with Internet browsers and graphical interfaces) involves the use of some sort of cursor control like what a mouse provides. However, a standard mouse is not always the best option for all users. There are currently many devices available to provide alternative computer access. These devices may be divided into categories: brain-computer interfaces (BCI), mouth-based controls, camera-based controls, and head-tilt controls. There is no single solution as each device and application has to be tailored to each user's unique preferences and abilities. Furthermore, each device category has certain strengths and weaknesses that need to be considered when making an effective match between a user and a device. One problem that remains is that these alternative input devices do not perform as well when compared to standard mouse devices. To help with this, assistive user interface techniques can be employed. While research shows that these techniques help, most require that modifications be made to the user interfaces or that a user's intended target be known beforehand by the host computer. In this research, a novel target-agnostic assistive user interface algorithm intended to improve usage performance for both head-operated and standard mouse devices is designed, implemented (as a mouse device driver and in host computer software) and experimentally evaluated. In addition, a new wireless head-operated input device requiring no special host computer hardware, is designed, built and evaluated. It was found that the Virtual Dynamic Tunnel algorithm improved performance for a standard mouse in straight tunnel trials and that nearly 60% of users would be willing to use the head-tilt mouse as a hands-free option for cursor control.

INDEX WORDS: Assistive interfaces, Hands-free computer access, Human-computer interaction

VIRTUAL DYNAMIC TUNNEL: A TARGET-AGNOSTIC ASSISTIVE USER INTERFACE ALGORITHM FOR HEAD-OPERATED INPUT DEVICES

by

FERROL RAWSON BLACKMON

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2010

VIRTUAL DYNAMIC TUNNEL: A TARGET-AGNOSTIC ASSISTIVE USER INTERFACE ALGORITHM FOR HEAD-OPERATED INPUT DEVICES

by

FERROL RAWSON BLACKMON

DEDICATION

This dissertation is dedicated to my mother, the late Madeline Hurst.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Michael Weeks, for all of his guidance and support.
I would also like to thank Dr. Saeid Belkasim, Dr. Xiaolin Hu, and Dr. Xiaochun He for serving on my committee and for their guidance.
Finally, I would like to thank my family and friends for all of their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AT	Assistive Technology
ATD-PA	Assistive Technology Device Predisposition Assessment
BCI	Brain Computer Interface
BASIC	Beginner's All-purpose Symbolic Instruction Code
E	East
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPROM	Erasable Programmable Read-Only Memory
HAAT	Human Activity Assistive Technology
HID	Human Interface Device
ITF	Individual-Technology Fit
N	North
NE	Northeast
NW	Northwest
MPT	Matching Person and Technology
PBASIC	Parallax BASIC
PC	Personal Computer
QUEST	Quebec User Evaluation of Satisfaction with Assistive Technology
RFCOMM	Radio Frequency Communication
S	South
SCD	Spinal Cord Dysfunction
SCI	Spinal Cord Injury
SE	Southeast
SW	Southwest
USB	Universal Serial Bus
W	West

Chapter 1

INTRODUCTION

In Marcia Scherer's book *Assistive Technology: Matching Device and Consumer for Successful Rehabilitation*, she writes,"There are more than 26,000 AT [Assistive Technology] devices available [for disabled persons], making the process of choosing the most appropriate device for a particular user more complex and time consuming" [12] than it was in the past. ABLEDATA is an online database of assistive devices maintained for the National Institute on Disability and Rehabilitation Research of the U.S. Department of Education. It is organized by functional activities like recreation, sensory disabilities, communication, and personal care [13]. To meet the challenge of matching the increasing number of available devices to users, it is arguable that the matching process has gotten better in recent years, first, by shifting the focus more on the user as an active participant [14] in the process and second, by considering "softer" factors in the process like psychosocial effect [12] [14] and third, by getting user feedback and follow-up on device usage [14]. Realizing that "AT devices are tools for enhancing the independent functioning of people who have physical limitations or disabilities" [12], a good match is important to enable this independent functioning while reducing the chance for device abandonment.

This research is focused in part on understanding the process of matching computer mouse type input devices (i.e. those that control the mouse pointer) to users. Specifically, it is related to the matching of hands-free mouse input devices to users wishing to have an alternative to traditional mouse control devices and to disabled persons who have no control of traditional devices, such as users with spinal cord injuries (SCI) or spinal cord dysfunction (SCD) who have limited use of their hands, arms or legs. While the experiments carried out in this work were with able-bodied participants only, the device and algorithm created would be advantageous to these users as well. The matching problem is characterized and described as it uniquely applies to this domain. Existing AT device matching methods will be described against the domain as well. For this research, a new head-operated mouse input device, along with supporting software, has been designed and evaluated against more traditional input devices. Also, a new target-agnostic assistive interface algorithm, called the Virtual Dynamic Tunnel, targeted for both head-operated and standard mouse devices, has been designed, implemented and evaluated.

Users of AT devices can become frustrated with a device, causing them to abandon it. A good match of a device to a user is important to avoid abandonment. Among the factors that can cause abandonment are the failure of providers of devices to take consumers' opinions into account, the device being hard to install and setup, poor performance of the device, and lastly, changing needs or priorities of the customer [12]. This research is mainly focused on looking at the first three factors by deriving improved methods of matching and providing a user with a head-operated mouse input in a cost effective way that has reasonable performance and is something that a user would not choose to abandon. Mouse input to a computer is very important nowadays as it is the primary input and navigation method for the Internet and for most other modern computer applications.

Techniques to improve performance of user interfaces and input devices exist. However, most of these are not target-agnostic, which means that the computer must know beforehand which target is to be selected; nor are they specifically targeted at hands-free mouse cursor input. If current device matching techniques are to be applied to users who want or need a hands-free option for a mouse input device, they must take into account things like 1) the mobility of the user, 2) the user's ability to voluntarily provide device input (e.g. sip/puff, eye movements/blinks, head movement, tongue movement, audio/voice control, jaw movement, etc), 3) the user's preferences in device usage, and 4) the psychosocial aspects of device usage. In this domain a single mouse input device does not fit all of the users. Nor is there a mouse input solution for this domain which is loosely bound to a single host computer; with some solutions requiring special hardware to be installed; thus making movement to a different computer system difficult. Current AT device matching techniques are mostly focused on the user's abilities in a broad sense with little consideration for mouse input for hands-free users specifically. Further, mouse input devices for disabled users, for example, have been provided in an ad hoc fashion on a per-computer basis, with the assumption that the user will access only one computer.

The aim of this research is to analyze the problem of matching AT mouse input devices to users needing or wanting a hands-free mouse input device. First, it proposes to research current methods of AT device-to-user matching to determine if these methods are applicable to mouse cursor control by hands-free users. Second, it offers a new hands-free mouse input device that does not require specialized hardware on the host computer, is small and inconspicuous, and is relatively inexpensive. Third, it proposes to implement improvements in performance, using a novel target-agnostic assistive interface algorithm (implemented as a mouse device driver and in host PC software), for head-operated and standard mouse input devices. Lastly, the effectiveness of the new device and the new algorithm will be evaluated through carefully designed experiments.

Chapter 2

BACKGROUND AND LITERATURE REVIEW

This chapter introduces the problem of matching AT devices to disabled users, then focuses specifically on the domain of this research: effectively matching hands-free computer mouse devices to users in general. First, current AT device matching tools and techniques are presented. Second, the SCI/SCD user is defined. Third, mouse devices available for disabled users are presented and categorized by function. Techniques and tools used to evaluate the effectiveness of AT device/user matching are presented. Finally, results of a preliminary experiment with a newly-designed head-operated mouse cursor device are presented.

2.1 Matching devices to users

The American Occupational Therapy Association's document, *Uniform Terminology for Occupational Therapy* [15], sets guidelines on the language and scope of the field of occupational therapy. This document contains a description of activities that individuals engage in. The activities are grouped into three performance areas. The use of a computer, and subsequently, mouse cursor control can be considered to span all three of the *Uniform Terminology for Occupational Therapy's* performance areas: daily living, work and productivity, and leisure. This fact underlines the importance of providing mouse cursor control to individuals with severe disabilities. However, the general problem of providing the right assistive devices to individuals in order to maximize performance is a problem that remains even when our focus is on the specific problem of providing mouse cursor control to severely disabled persons. To tackle the general problem of matching devices with individuals, several models have been developed. Besides matching, these models provide a means to measure the effectiveness of a match.

Prior to the introduction of the Matching Person and Technology (MPT) model [12] [16] by Scherer, assistive technology practitioners were mainly focused on trying to remove the disability. MPT was the first attempt to consider that the disabled user, their environment, and the assistive technology were important parameters to take into account [12]. The MPT process is carried out by having the assistive technology user and the assistive technology provider fill out forms, known as instruments, together to gather information that will help in choosing an appropriate assistive device. The instruments are also used later, after a device has been selected, to measure satisfaction and the effectiveness of the match. In fact, MPT provided the theory for the Quebec User Evaluation of Satisfaction with assistive Technology (QUEST) satisfaction assessment tool [12]. The main MPT instrument is the Assistive Technology Device Predisposition Assessment (ATD-PA) form [16]. This instrument has "scores in four areas" on the "use and nonuse" of the device; the device itself, personality traits, disability elements, and the environment [12]. When filling out the MPT instruments, the customer sets weights on preferences and expected outcomes [14]. MPT leads the way for other models, such as the Human Activity Assistive Technology (HAAT) model [12].

The HAAT model, developed by Cook and Hussey, "emphasizes the need to assess human ability within the context of the environment where the task will be performed or else there may be a misrepresentation" [2]. The model defines an assistive technology system as consisting of a device, a user and the environment in which the human uses the device. The device is thought of as meeting the needs of the user while also matching the user's skills and functioning in a context (i.e. environment). In HAAT, it is noted that while both behavior and performance (with an assistive device) can be observed, only performance can be accurately measured. And for the performance measure to be meaningful, the context must be taken into account [14]. HAAT is not only derived from MPT, but also from the Human Performance Model [2], which contains three main areas: the human subject, the activity and the context. The HAAT modification to this base is to add assistive technology as a main area, and to make context cover all of the areas of interest. In HAAT, an activity can be considered to be synonymous to a goal the user has. It is defined based on life roles, performance areas and tasks to be performed. Context can be thought of as the setting; such as social, cultural and physical. The human area is defined as the skills the user has as an operator of a system. The assistive technology, of course, describes the characteristics of the assistive device or system.

Randolph looked at matching users with an appropriate biometric interface (i.e. brain-computer interface) to achieve optimal performance [2]. To achieve this she created a new matching framework, based off of *task-technology fit*, which was called *individual-technology fit* (ITF). In ITF, the context is defined as biometric interfaces. Characteristics of the users are captured by profiling individual users through the use of questionnaires. A fit was defined as a measure of performance with a chosen biometric interface. During the research, Randolph found that certain user characteristics and user states did, in fact, determine the best biometric interface device for these users [2]. Next, we look at how performance of input devices has been evaluated by using Fitts' Law and the Steering Law.

Fitts' Law has been used to compare input devices by measuring the time a user takes to acquire target on the computer display [17] [18]. According to Fitts' Law [19], the time it takes for a user to acquire (i.e. select) a desired target is a function of the user's distance from the target and the size of the target as given by

$$MT = a + b \log_2(2A/W) \tag{2.1}$$

where MT is the time to acquire the target, a and b are constants, determined experimentally, A is the distance of the target from the user's starting position, and W is width of the target. MacKenzie [20] derived a now commonly used version of Fitts' Law that has "better fit with observations" [21], more closely models the underlying information theorem, and gives positive values for the "index of task difficulty" [21]. The MacKenzie formulation is given by

$$MT = a + b \log_2(A/W + 1)$$

where

$$\log_2(A/W + 1)$$

is the index of task difficulty. While Fitts' Law has been used successfully in evaluation of input devices, Accot et al. [22] note that Fitts' Law may not hold if the scale of the system changes (i.e. increasing the distance from, and size of the target by a constant may affect performance). In [22], Accot et al. used the steering law to show that input device performance is affected by scaling and, in fact, a scale function produces an inverted U-shaped performance curve, which contradicts literature that implies linear performance using Fitts' Law. The steering law is used to model movement (i.e. steering) through a tunnel. In the context of computer input devices, this means moving a pointer along a trajectory constrained by a 2-dimensional tunnel that has variable length and width. It is analogous to steering a car along the curves of a road. The time T_C required to steer through a tunnel C is given by

$$T_C = a + b \times ID_C \tag{2.2}$$

where a and b are constants as mentioned in Formula 2.1 and ID_C is the index of difficulty given by

$$ID_C = \int_C \frac{ds}{W(s)} \tag{2.3}$$

Here s is the "curvilinear abscissa along the path" [22] and $W(s)$ is the width of the tunnel at s. Like in the formula for Fitts' Law, the index of performance IP is

$$IP = 1/b$$

The authors tested the performance of motion at various scales using a tablet computer [22]. Two different types of tunnels were used: a straight one and a circular one. It was pointed out that while this does not cover all likely tunnel types, it does represent the types of challenges users would face in steering and also that other tunnels can be represented with combinations of these two types of tunnels. The results showed that the movement scale had significant influence on steering performance; that the relationship between test phase and movement scale had no significance, so practice did not help; that there was a significant relationship between movement scale and tunnel type; and that the index of performance resulted in an inverted U-shaped function of movement scale. Scale affected the index of performance, but not as much on the index of difficulty. This was explained by the use of different parts of the body for different types of motion (i.e. the arm for big, large-scale movements; the hand for medium-scale movements; and the finger tips for fine-scale movements) [22]. In [23], Accot et al. apply the steering law to compare different device types like what was done in [17] with Fitts' Law. Besides finding which device types offer the best performance for steering tasks, their paper also shows the steering law can be used successfully for these type of comparisons and even across studies, just as the Fitts' Law is used.

One of the limitations attributed to Fitts' Law is that its focus on target acquisition is limited and that the trajectories taken to targets are important as well when considering devices. For example, an intended target may be a sub-menu in a hierarchy of menus that can be accessed only through a curved path through the sub-menus. The steering law addresses this by modeling movement through a tunnel. However, the shape of the tunnel is not modeled and therefore performance with the steering law can only be compared within device categories [23]. Also, as noted above, both models' performance can be affected by scales that are very large or very small. In the following section, we discuss current alternatives for mouse cursor control including brain-computer interfaces (BCI), mouth-based controls, head-tilt based controls, camera-based controls, and sound-based controls.

2.2 Mouse cursor control

There are many device options for users requiring hands-free cursor control of a computer. Among these, the main categories, arguably listed in the order of most intrusive to least intrusive to the user, are brain-computer interface (BCI) controls, mouth-based controls, head-tilt based controls, camera-based controls, and sound-based controls. The broad categories capture the fact that there is no panacea solution to hands-free cursor control and that a candidate device needs to match to an individual's abilities and preferences. Figure 2.1 shows the users targeted in this research along a continuum of disabilities and control interface options. The users of interest are indicated by light gray and dark gray along the continuum.

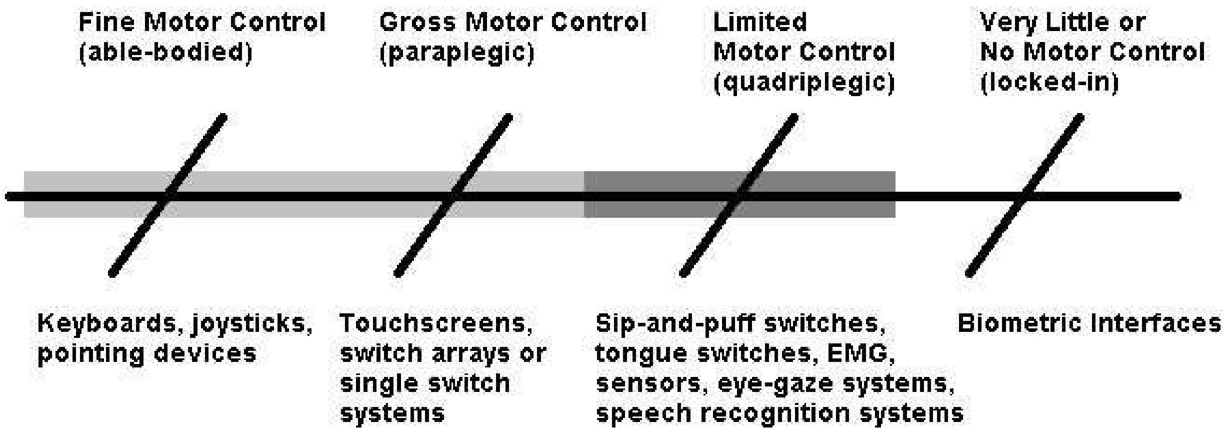


Figure 2.1. Targeted users – continuum of disabilities and control interfaces. Adapted from [2].

Users who have no or little motor control are limited to using biometric intefaces like BCI as shown in Figure 2.2. The BCI controls suffer from a slow data rate (e.g. 5 to 68 bits per minute) [2, 24] and can, with the exception of the P300 [1] type, require a long training phase [25]. In [27], noise (i.e. in the signal) is also mentioned as a drawback. Furthermore, these systems are also invasive and usually expensive. While research on these systems has been promising, they are not yet ready for general use as a low-cost hands-free cursor control option.

Mouth-based controls are also helpful to many users. A system using sips and puffs is shown in Figure 2.3; and one operated by a user's tongue is shown in Figure 2.4. Surdilovic improved upon a mouth-based cursor control interface for persons with spinal cord injuries, in which the user blows or sucks air through a straw device [26]. In Surdilovic's system, the blows are referred to as "puffs" and the sucks are referred to as "sips." Users' "sips" or "puffs" control a "scanning line" that rotates clockwise or counterclockwise, zooms into a target, and then moves a cursor along the scan line to the exact object of interest. The user can, with sips and puffs, then indicate which kind of mouse click they would like to perform on the selected object (e.g. Single Click, Double Click, Right Click, Drag Start, or Drag End) through a form of Morse Code [26]. However, this system requires a specialized

[1]A P300 is a type of Event Related Potential (ERP) variation in an electroencephalogram (EEG) signal produced in response to a significant but low probability event experienced by a subject [25, 26].

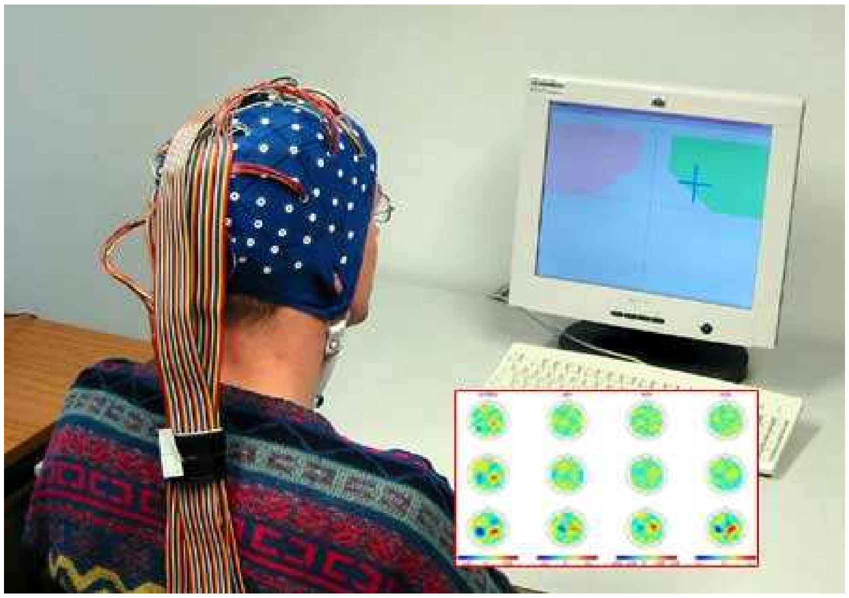


Figure 2.2. Brain computer interface (BCI) [3].

hardware card installed on the computer that reads and digitizes the air flow through the user's straw. Furthermore, mouth-based systems have hygienic concerns and may not be comfortable for all users. Also, as pointed out in [28], the system may not function while the user is talking or eating.

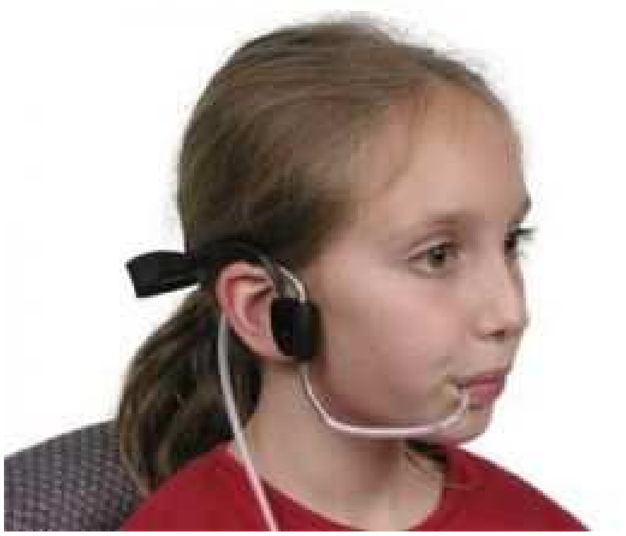


Figure 2.3. Sip/puff interface [4].

Figure 2.4. Tongue-based interface [5].

Emotiv Systems have created a consumer-based headset, called the EPOC (shown in Figure 2.5), which detects and uses electric brain signals to interface with a computer. The headset also includes a gyroscope that can be used for cursor control. The headset connects wirelessly through a dongle that attaches to the host computer. The company asserts that no additional software is needed. However, Emotiv Systems also provides developers with a set of SDKs, which can enable the developers to detect a user's emotional states, conscious thoughts and facial expressions, via the EPOC, to interface with the host computer. The premise is that interaction with the computer can be enhanced through a more natural, multi-modal approach. The electric brain signals are captured through 14 electrodes on the headset. The EPOC uses a lithium battery that gives 12 hours of use. The cost of the EPOC headset is currently $299US [6].

There are camera-based control systems which are more practical, usable and inexpensive such as the Headmouse [7] (shown in Figure 2.6). This is software which runs on the host computer, which has a webcam attached. The software uses the webcam to detect head movement and uses the head movement to manipulate the mouse cursor. Clicks can be generated by blinking, opening and closing the mouth, or automatically after a timeout period. The user chooses which method to use during a calibration phase. Also during this phase, the system is calibrated for a specific user's head movement [7]. It uses a USB

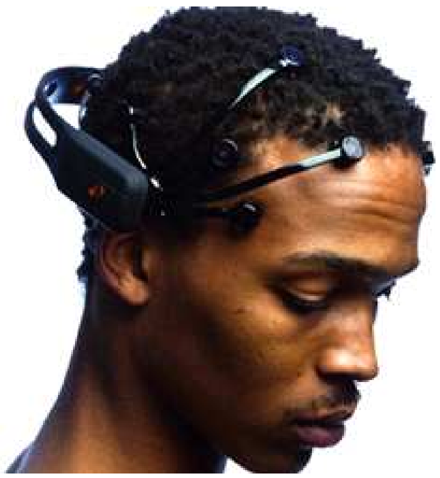


Figure 2.5. Emotiv Systems EPOC device [6].

connection and standard mouse drivers. However, it requires a line of sight to the receiving camera to manipulate the mouse cursor, which may not be desirable for all users in all environments [29].

Figure 2.6. Headmouse2: camera-based interface [7].

In [30], a computer mouse is controlled by the user with vowel sounds spoken into a microphone. The different sounds move the mouse cursor in a different direction. Curved movement is achieved by blending different vowel sounds together. Mouse clicks are done with a soft clicking sound. This system uses a microphone and a standard sound card. This system has been tested with users having spinal cord injuries. In trials of this system, each participant was trained for 12 hours. The training was intended to teach the users how to make the proper sounds, to “memorize directional patterns,” and how to “manipulate the cursor speed." While the work seems promising due in equal parts to its simplicity and light hardware requirements, the 12-hour training phase is long and using the system was often frustrating and tiring [30].

Igarashi and Hughes [31] use a combination of voice and non-verbal sounds to reduce interaction turnaround time for systems that are normally voice only. They accomplish this by using a combination of tonguing, continuation of sound, and pitch to control aspects of a computer interface like scrolling and mouse control. For example, tonguing can control discrete values such as the channels on a television by the user saying, 'Channel up, ta, ta, ta,' which will increase the current channel by three [31]. Continuation of a sound controls continuous aspects of the interface such as scrolling. For example, the user might say, "Scroll down, ahhhhhhh," which would cause a browser to scroll down for the entire time that the "ahhhhhh" is sounded out. The rate of continuous control is increased or decreased by changing the pitch of the continuous sound in their system. Some advantages of their system over traditional voice recognition are that it is simple, is somewhat language independent, and provides "immediate, continuous control" [31]. Some limitations are that use of the system can be tiring; it requires a somewhat unnatural manner of using the voice [31] and since it is sound-based it can be affected by a noisy environment. Another system which uses sound is the system created by Sporka et al. [32], which can control the mouse pointer by having a user whistle, hum or sing. Their system operates in two different modes: orthogonal and melodic. In the orthogonal mode (shown in Figure 2.7), a pitch threshold is defined such that if the user sounds a tone that is above this threshold and subsequently increases pitch, the cursor moves up on the screen. If the user sounds a tone that is above this threshold and decreases pitch, the cursor will move down. If the user wishes to move the cursor left or right, she would sound a tone below the threshold and would increase the pitch to move right and decrease the pitch to move left. Rate of movement is determined by the magnitude of the difference between starting and ending pitches. Mouse clicks and double clicks are indicated by short sounds of any frequency. In the melodic mode (shown in Figure 2.8), 12 notes in an octave were mapped onto a compass to indicate direction. In this mode, the cursor moves in the direction of the sounded note at a constant speed. In experiments with the system, the orthogonal mode proved to be easier to use. Whistling to control the cursor was found to be more accurate but more tiring that humming. Hissing did not work as a method to control the cursor because the targets could not be homed in on. Advantages of the system are that it requires only a microphone and a standard sound card and that it can be controlled in a hands-free manner. Some disadvantages are that it requires some training; it can be tiring; and because it uses sound, it may be affected by a noisy environment [29, 32].

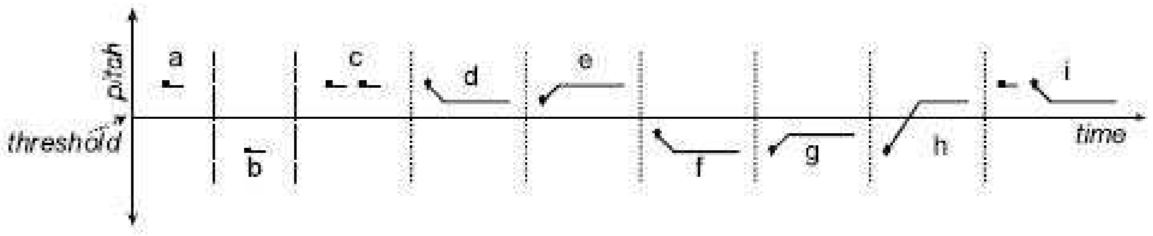


Figure 2.7. Voice control – Orthogonal mode [8].

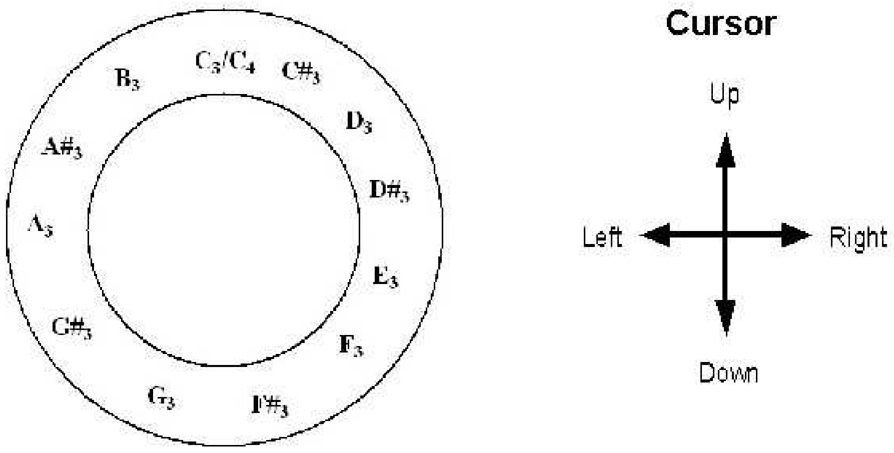


Figure 2.8. Voice control – Melodic mode [8].

Menon [33] describes a system, called Supple, created by the University of Washington, that optimizes the graphical interface for users' abilities (e.g. vision and motor skills). The system evaluates a user's performance of "mouse pointing, dragging, and clicking skills" [33] and with that, generates an interface tailored to the user's ability. The evaluation step took an average of 20 minutes for able-bodied users and an average of 90 minutes for users with motor disabilities. The system was able to narrow the performance gap between able-bodied and disabled users by 62% [33].

MacKenzie et al. [17] compare a mouse, a tablet, and a trackball by applying Fitts' Law, which states that the time to acquire a chosen target is related to the target's size and how far it is away. MacKenzie et al. found that the mouse had the best performance compared with the trackball and comparable to the tablet for pointing. The trackball had poor performance in pointing and dragging. Overall, the tablet was best for pointing and the mouse was found to be the best for dragging. Similarly, a gyroscopic mouse, a trackball and a Twiddler2[2] mouse were compared in [34]. Here, an application was used for testing where circular targets would appear randomly on the screen and the time to move and click on the target was recorded and compared among the different device types. Zucco et al. found that the gyroscopic mouse had the best performance in target selection [34], while the trackball had the lowest error rate. The gyroscopic mouse and Twiddler had about the same error rate. The user surveys given by Zucco et al. rated the gyroscopic mouse first, the trackball second and the Twiddler last [34].

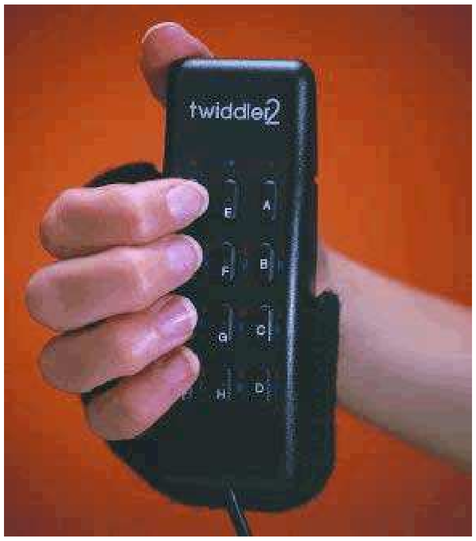


Figure 2.9. Twiddler2 device [9].

[2]A one-handed mouse/keyboard combination device [34]. See Figure 2.9.

In [35] and [36], Blackmon and Weeks compare a new head-tilt mouse's performance to a standard wireless Bluetooth mouse and a laptop touchpad mouse using data captured from a test application similar to [34]. Their results are captured later in this chapter in sections 2.4 and 2.5.

The preceding discussion is summarized in Table 2.1, which shows a representation of hands-free device in the left column for each device type or family (indicated in the middle column). Drawbacks for each device type are listed in the right-hand column. As the table suggests, there is no "one-size-fits-all" solution to device type choices. Users must be matched to devices based on such criteria and models, as was covered at the beginning of this chapter. The models point out that it is important to consider the users' ablilities/preferences, the intended environment and the actual device technology. In the following section, we describe a new, low-cost head-operated mouse cursor device which requires no special host computer hardware as a hands-free alternative to mouse cursor control.

2.3 Apparatus

2.3.1 Head-tilt mouse

This research is concerned with improving performance with hands-free mouse devices. One of the possible benefits is improving the lives of many people with severe physical disabilities, since with hands-free devices they could be given computer access comparable to that which able-bodied people currently enjoy. To help achieve this goal, a wireless mouse which operates by using the tilt of the user's head has been built. This mouse is intended for users who do not have the ability or choose not to use their hands to operate a traditional mouse. The head-tilt mouse described below has been covered by the provisional patent U.S. Patent Application No. 61/221,772 [37]. The cursor movement is detected by the use of an accelerometer embedded inside the device. Head movement information is transmitted wirelessly using the Bluetooth wireless protocol to the computer. Bluetooth is an open protocol specification for wirelessly transmitting voice or data over a short range (i.e. either up to 10 meters or up to 100 meters, depending on the application). It has low-

Device	**Technology**	**Drawbacks**
Emotiv EPOC	BCI & Gyroscope	Conspicuous In beta Technical details not currently available May be overkill Performance as mouse unknown
BCI	BCI	Slow data rates Long training phase Conspicuous & invasive Expensive Frequently requires bulky hardware
Headmouse	Camera-based	Requires line of sight Actual performance unknown
Tougue switches	Mouth-based	Hygienic concerns Conspicuous & invasive Frequently requires bulky hardware
PC Microphone	Audio-based	Susceptible to interference Long training phase Using the system can be tiring
Nintendo Wii	Accelerometer	Closed architecture Unknown how to achieve hands-free click function
Proposed Head-tilt mouse	Accelerometer	Missing hands-free mouse click Host computer software needed

Table 2.1. Hands-free Cursor Control Devices

power requirements which suit portable battery-operated devices, unlike 802.11b wireless LAN protocol, although it operates at the same 2.4GHz band. The Bluetooth protocol was originally designed to eliminate data cables between devices (e.g. keyboards, mice, cell phones, head phones, etc.) [38] [39]. Software on the computer translates the head movements into mouse movements. The head-tilt mouse embedded in a baseball cap is shown in Figure 2.10, with the first version of the head-tilt mouse show in Figure 2.11.

The system block diagram can be seen in Figure 2.12. The accelerometer, shown in the upper left corner of the figure, senses the user's head movement. These data are read by the microprocessor, in the top middle of the figure, where they are processed and transmitted through the Bluetooth wireless radio embedded in the hat, shown in the top right of the figure, to the host computer's Bluetooth radio, shown in the lower left corner of the figure. Currently, the host machine uses a USB Bluetooth radio. The host computer runs translation software, shown in the lower middle of the figure, that then translates these data to actual mouse cursor movements. Other than at the physical layer, the communication is essentially one way. The host computer does not need to communicate back to the tilt mouse. This effectively reduces the complexity of the communication protocol.

As can be seen in Figure 2.10, the device is roughly the size of a mint box or of a deck of playing cards. The two main components making up the size are the Bluetooth radio and the application breadboard. The breadboard circuit can be further reduced in size by implementing it on a printed circuit board. Power is provided by a standard nine volt battery.

2.3.2 Advantages

The advantages of the proposed head-tilt mouse system is that no special hardware is needed, except the Bluetooth radio that commonly comes with a wireless mouse; it is inexpensive to build (<$200US); it does not require a direct line of sight to the computer screen; it is wireless; it is small (currently embedded on the bib of a baseball cap). Probably one of the most important advantages is that its small size makes it inconspicuous; especially if it is embedded in a regular cap or hat. Also, the combination of wireless without the need of

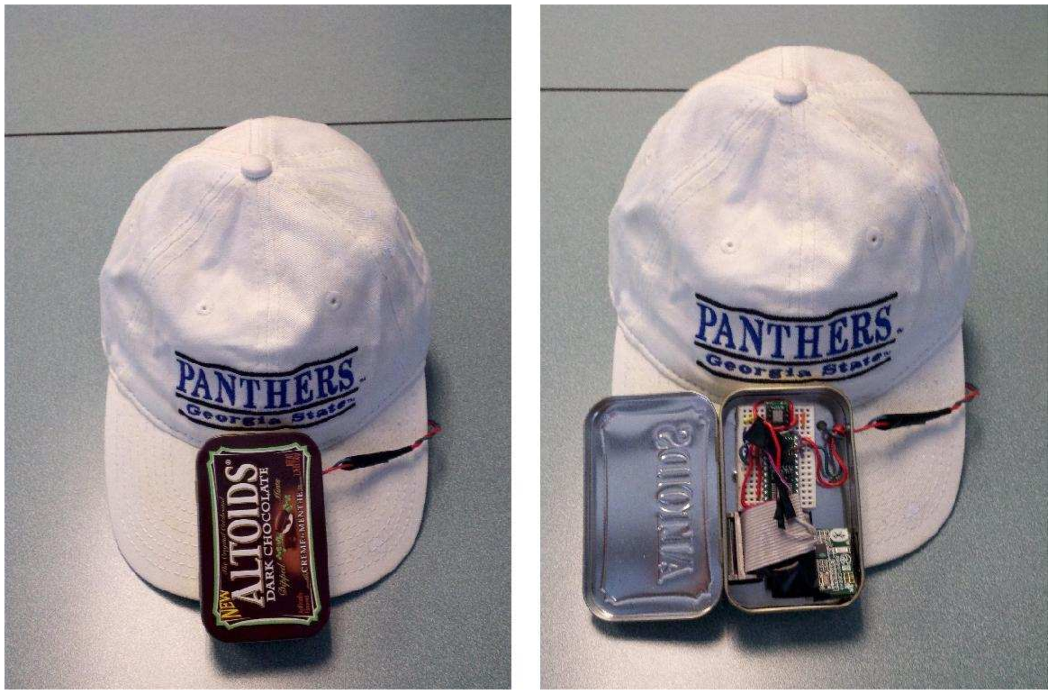


Figure 2.10. Head-tilt mouse embedded in baseball cap

direct line of sight should allow more user mobility than the camera-based systems mentioned previously. In the following three sections we cover a design overview and details about the hardware and software design of the the head-tilt mouse system.

2.3.3 Design Details

To operate the head-tilt mouse, the user tilts his head left or right, as shown in Figure 2.13, to move the mouse cursor left or right and moves his head up or down, in a nodding motion, as in Figure 2.14, to move the cursor up or down.

The tilt is sensed by an embedded accelerometer. In this device, the Memsic 2125 was used. The 2125 is a low cost accelerometer that works well with and connects directly to the BASIC Stamp 2 microprocessor. Among other things, it senses tilt and tilt angle. Each axis (i.e. X and Y) can sense 90 degrees of tilt. This is more than adequate for the head-tilt mouse since it is unlikely someone would want or be able to move their head that much. Also, the chip is sensitive enough for real-time use and has been used in applications like

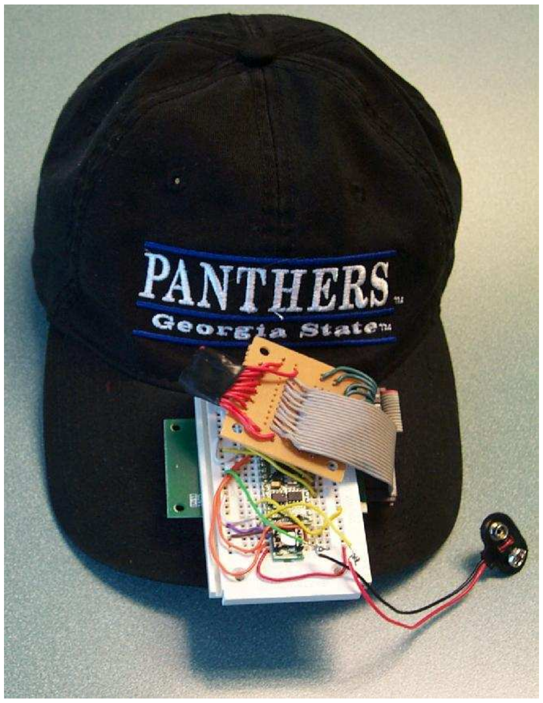


Figure 2.11. First version of head-tilt mouse embedded in baseball cap

real-time robot balancing. These are the values used in the head-tilt mouse to control the cursor [40].

2.3.4 Hardware Design

The information from the Memsic 2125 is read by the BASIC Stamp 2 microprocessor embedded in the tilt mouse. The processor reads information from the Memsic 2125 (shown in Figure 2.16), processes it, and sends it to the Bluetooth transmitter [Figure 2.12]. The BASIC Stamp 2 (shown in detail in Figure 2.18) is an 8-bit 20MHz microprocessor that is programmed with a version of the BASIC language, PBASIC. The PBASIC program is stored in an EEPROM and starts executing whenever power is provided to the chip. There are also debugging capabilities provided through various demonstration boards and serial connections to the host computer [41]. The circuit diagram for the head-tilt mouse is shown in Figure 2.15. The BASIC Stamp 2 microprocessor is the large rectangle on the right of the diagram. The Memsic 2125 accelerometer is in the top left, while the EasyBluetooth Bluetooth radio is on the lower left (shown in detail in Figure 2.17). The BASIC Stamp 2

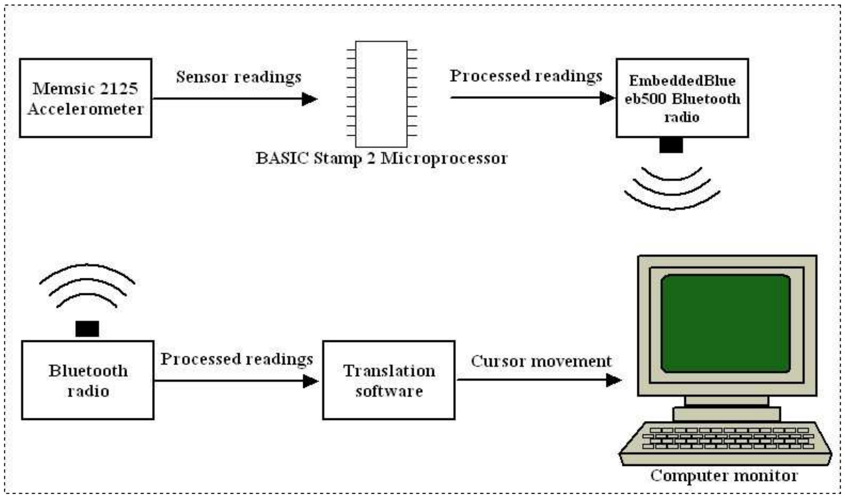


Figure 2.12. System block diagram

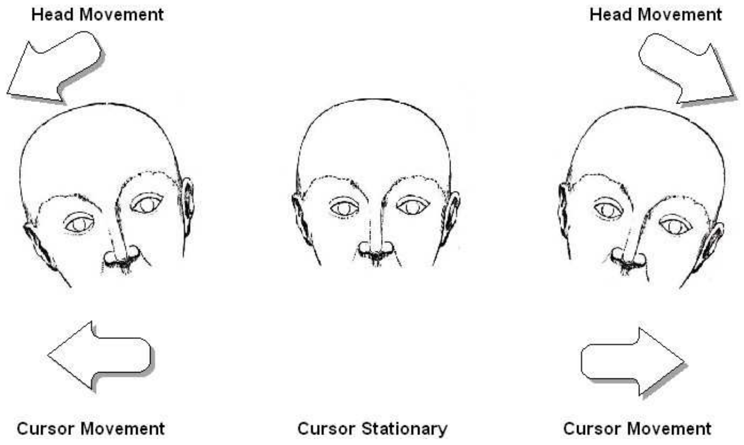


Figure 2.13. Head-Tilt Movements. Figure adapted from [10].

takes nine volts into *Vin* (pin 24) and provides five volts through its *Vdd* (pin 21) to power the Bluetooth radio (pin 20) and the accelerometer (pin 6).

Head movement data are sent to the computer using the Bluetooth protocol. In the head-tilt mouse the A7 Engineering's EasyBluetooth is used. This is a low cost radio that works well with the BASIC Stamp 2 microprocessor. The Bluetooth RFCOMM profile is used to set up a virtual COMM port. The RFCOMM profile provides a programming interface that emulates the behavior of a COMM port over a wireless connection and is used mainly because this is the only Bluetooth profile that is supported by the EasyBluetooth radio. It makes programming the radio simple. However, the EasyBluetooth radio lacks support for the Human Interface Device (HID) profile. The HID profile provides the specifications

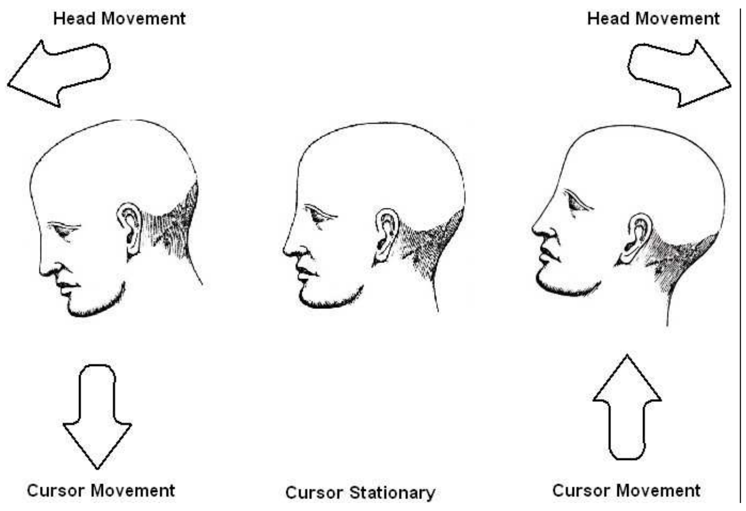


Figure 2.14. Head Nod Movements. Figure adapted from [10].

for passing data between keyboards, mice and other "human interface devices" [38]. In the current head-tilt mouse implementation the head movement data are transmitted to the host machine through this virtual COMM port [42]. The host computer for this system is running Microsoft Windows XP. Since the HID profile is not supported by the EasyBluetooth, a custom program had to be written for the host computer that opens a virtual COMM port on the host machine's Bluetooth radio to read data from the head-tilt mouse. This custom program is written in C++ and uses the Win32 Application Programming Interface (API) to access the Bluetooth RFCOMM library to use the virtual COMM port. The host machine in our setup has a D-Link DBT-120 USB Bluetooth radio [43]. As data is read from the head-tilt mouse, it is translated into mouse cursor movement on the host computer screen as if it were a standard mouse. All of the device hardware components are powered by a single, standard nine volt battery.

Mouse click functionality is provided by a push button connected directly to the microprocessor. In the future, we want to provide a hands-free method of clicking the mouse. We have looked at a possible audio-based solution, which would be advantageous because it would require only additional software (assuming the host computer has a microphone). A possible drawback of an audio solution is that interference may be encountered in a noisy environment. Another possible option might be to use a tongue switch. However, many

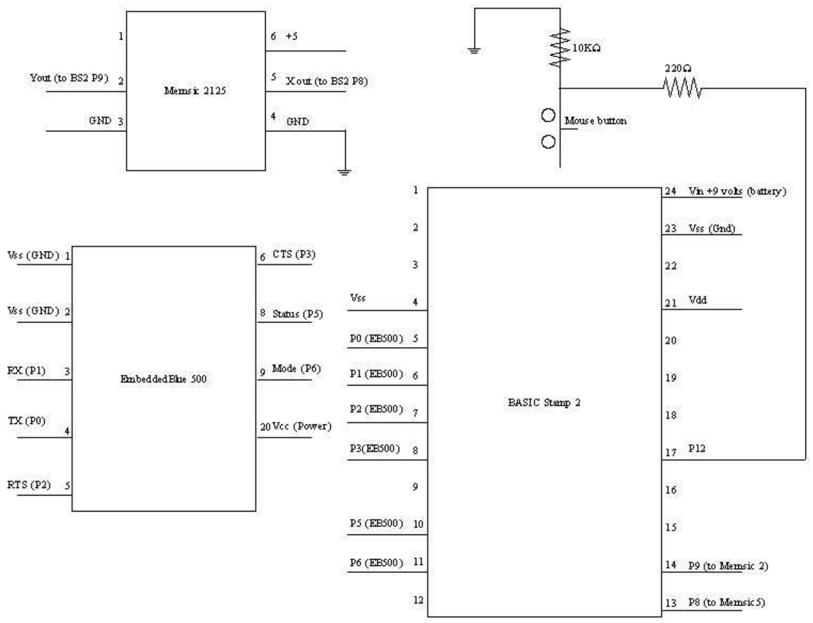


Figure 2.15. Tilt Mouse Circuit Diagram

Figure 2.16. Memsic 2125 accelerometer [11].

special application tongue switches are expensive and would drive the cost of the system up. There is also the hygienic considerations of using a tongue switch, which might not be desirable to some users. Another possible solution that would only require software would be to use the cursor movement itself (e.g. lingering over a target) to indicate clicks.

2.3.5 Software Design

An initial one-time trust relationship must be established between the Bluetooth radio on the head-tilt mouse and the Bluetooth radio on the companion computer. Once this relationship is formed, two COMM ports are created; one for incoming data to the companion computer and one for outgoing data to the head-tilt mouse.

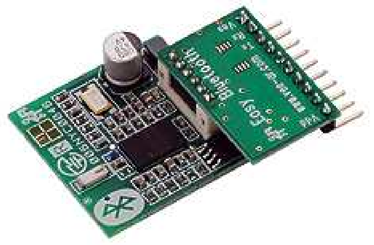


Figure 2.17. EasyBluetooth Bluetooth radio [11].

Figure 2.18. BASIC Stamp 2 microcontroller [11].

Currently, to calibrate the head-tilt mouse, an application is run where the user holds a comfortable neutral position for 10 seconds. A sensitivity level can also be entered, otherwise a default sensitivity is used. As the calibration application exits, the values for the neutral position and sensitivity are written into a configuration file that is read by the mouse control application. Both the calibration application and the mouse control application were written in C++ and run as command-line applications on Microsoft Windows XP.

The mouse control application on the host PC opens the COMM port coming from the head-tilt mouse and processes these data, moving the mouse cursor in response to head tilts proportional to the sensitivity given to the calibration application. The microprocessor on the head-tilt mouse is programmed in PBASIC to read data from the accelerometer and send it over a Bluetooth radio through a Bluetooth COMM port. This program is stored in EEPROM and is executed as soon as the device is powered up. The accelerometer readings are sent constantly over the virtual COMM port. The host PC mouse control application reads these values and processes them; translating them into mouse cursor movements and button presses. The processing algorithm is shown in Figure 2.19. The algorithm consists of an infinite loop in which the COMM port buffer is read and for each x and y value in the buffer (along with a possible mouse button press indicator), the current mouse cursor position is determined. If the values read from the buffer are significantly different from previously read values, and significantly different from the center position (determined during calibration), then a new mouse cursor position is set using the preferred sensitivity value, which was set during calibration. The head-tilt mouse was initially evaluated in an experiment described in the following section.

```
do
   Buffer <- ReadBuffer()  // Buffer format: [B:01] Y:FF X:FF..
   // Process Buffer
   for each data_point in Buffer
      pt <- GetCursorPos()
      if ( data_point is an X value )
         x_value <- GetXValueFromBuffer()
         if( |prev_x - x_value| > x_threshold )
            if ( |X_CENTER - x_value| > THRESHOLD)
               pt.x <- pt.x + ( x_value - X_CENTER ) / sensitivity
         prev_x <- x_value
      if ( data_point is Mouse Button && not clicked recently )
         GenerateMouseClick()
      //**************************************
      //
      // Code for "if ( data_point is a Y value )," same as "if ( data_point is an X value )" above
      //
      //**************************************
      if ( prev_pt != pt )
         SetCursorPos(pt)
         prev_pt <- pt
while (1==1)
```

Figure 2.19. Mouse Movement Processing Algorithm

2.4 Initial experiment

2.4.1 Description

In order to evaluate the performance of the first version of the head-tilt mouse, an experiment was designed similar to [34]. In our experiment, we compare the head-tilt mouse to a standard Bluetooth optical mouse and a laptop touchpad mouse. Price and availability have prohibited us from comparing the head-tilt mouse to other hands-free pointing devices thus far. The experiments were done on a ACER Aspire 1690 laptop running Microsoft's Windows XP.

2.4.2 Participants

There were four participants in the initial experiments. All were able-bodied in that they were able to successfully operate the touchpad mouse and the optical Bluetooth mouse. There were three females and one male participants.

2.4.3 Test Application

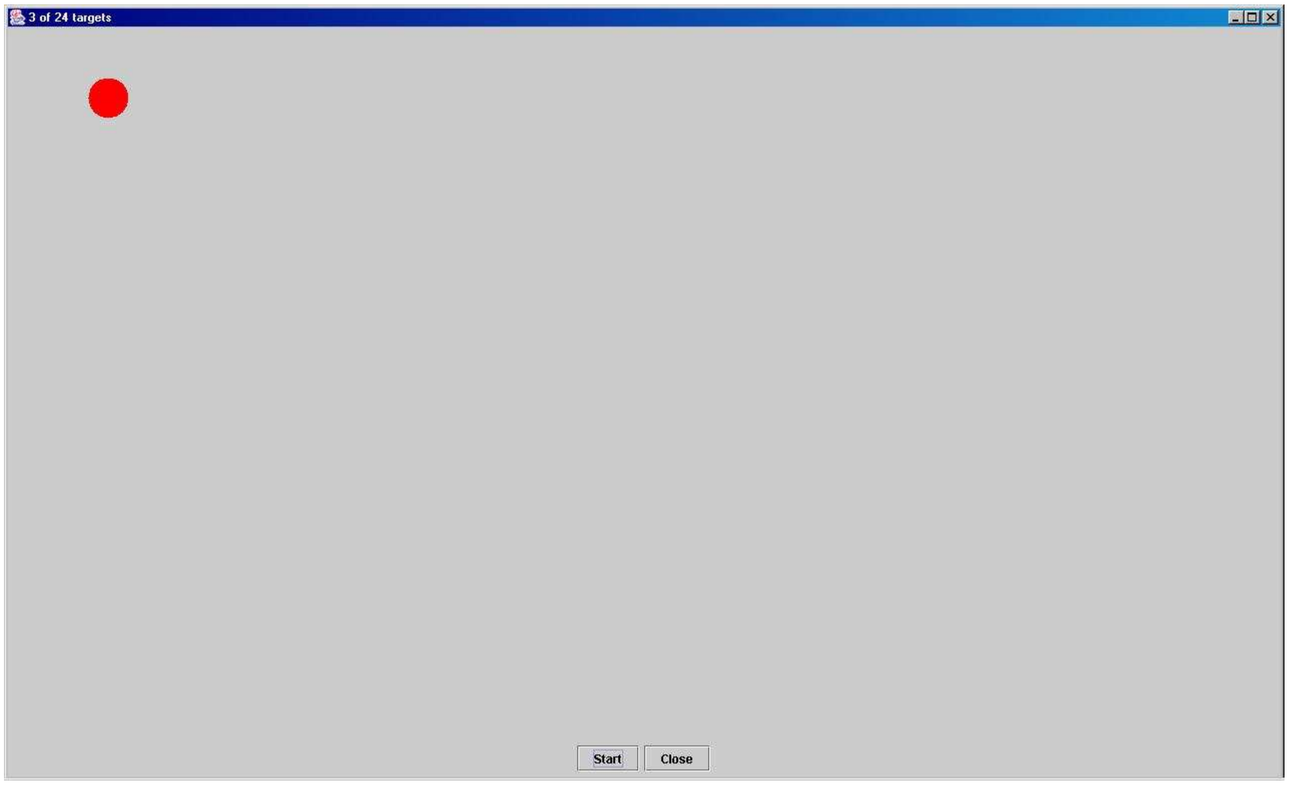


Figure 2.20. Custom test application

To carry out the experiments, a test application was written in Java similar to [34]. The application starts by drawing a 40-pixel diameter circular target in the center of its screen. Once the user selects this target, a timer is started and the target is erased and randomly redrawn in one of eight possible locations; corresponding to map locations N, S, E, W, SE, SW, NE, NW. Each of these locations are randomly presented three times. Therefore, for each task, the target is moved 24 times. The timer is stopped once the final target is selected. Data about target hits and misses are stored as well as time to complete the task (24 trials).

2.4.4 Training Phase

Each user initially took part in a training phase to familiarize themselves with the test application and the different device types. During the training, the user performed a task consisting of 24 movements of the target for each device type. For the head-tilt mouse, the device was first calibrated for each user.

2.4.5 Experimental Phase

The experimental phase is carried out using a $3 \times 6 \times 24$ repeated-measures. This means that each participant performed a task (24 target movements) for all six orderings (3!) of the three devices. Testing all orderings allows us to determine whether or not the order of device use has an effect on the results.

For each task (24 target movements), the target was randomly moved on the screen to each of the 8 map positions 3 times. We used the same random seed for each task, so that for each device type the targets are moved to the same locations. Each ordering of device types is called a block and the seed is changed for each block. Each block is assigned to a participant in a random order.

2.5 Initial experimental results

The experiments showed that users were able to acquire a target with the head-tilt mouse in 10.9 seconds on average, with 48 target misses for 24 targets. Misses are defined as a mouse click event which happens when the mouse cursor is positioned outside of the target. The minimum average time was 8.8 seconds and the maximum was 13.6 seconds. This compares to 1.6 seconds for the optical mouse and 1.9 for the touchpad, each with essentially zero misses. Results are summarized in Figure 2.21. In the figure legend, Bluetooth refers to the

Bluetooth wireless mouse, Touchpad refers to the Touchpad mouse on the laptop computer and Head-tilt refers to the head-tilt mouse. These results show that the head-tilt mouse is feasible as a mouse interface device to provide mouse control to users with disabilities or as an alternative input device for able-bodied users. The results also establish a baseline of performance to be improved upon. We found that people were able to navigate to the

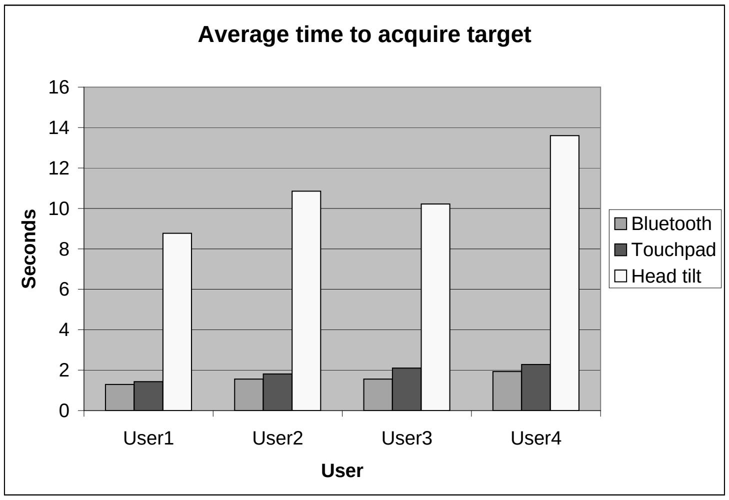


Figure 2.21. Average time to acquire targets for each user per device type

targets in an average of 10.9 seconds with an accuracy of 40-50%. We also estimated that by increasing the target size from a radius of 20 to 30 pixels, this would possibly increase the accuracy from 40 to 60%. We believe that these results show that the device is a feasible option to users with disabilities or as simply a hands-free alternative to the traditional mouse.

Using some sample data from the head-tilt mouse only, we show in Figure 2.22 how our experimental results compare to the Fitts' model prediction that time for target acquisition is related to distance from the target and target size. Since our target size is set to a constant, the only thing that changes is distance to the target; and it can be clearly seen in this figure that as the distance to target increases, so does the time required to acquire that target.

2.6 Research Goals/Methodology

The head-tilt mouse, from section 2.3, has been shown to be a viable hands-free option for mouse cursor control. Over the course of the research the device has been refined and

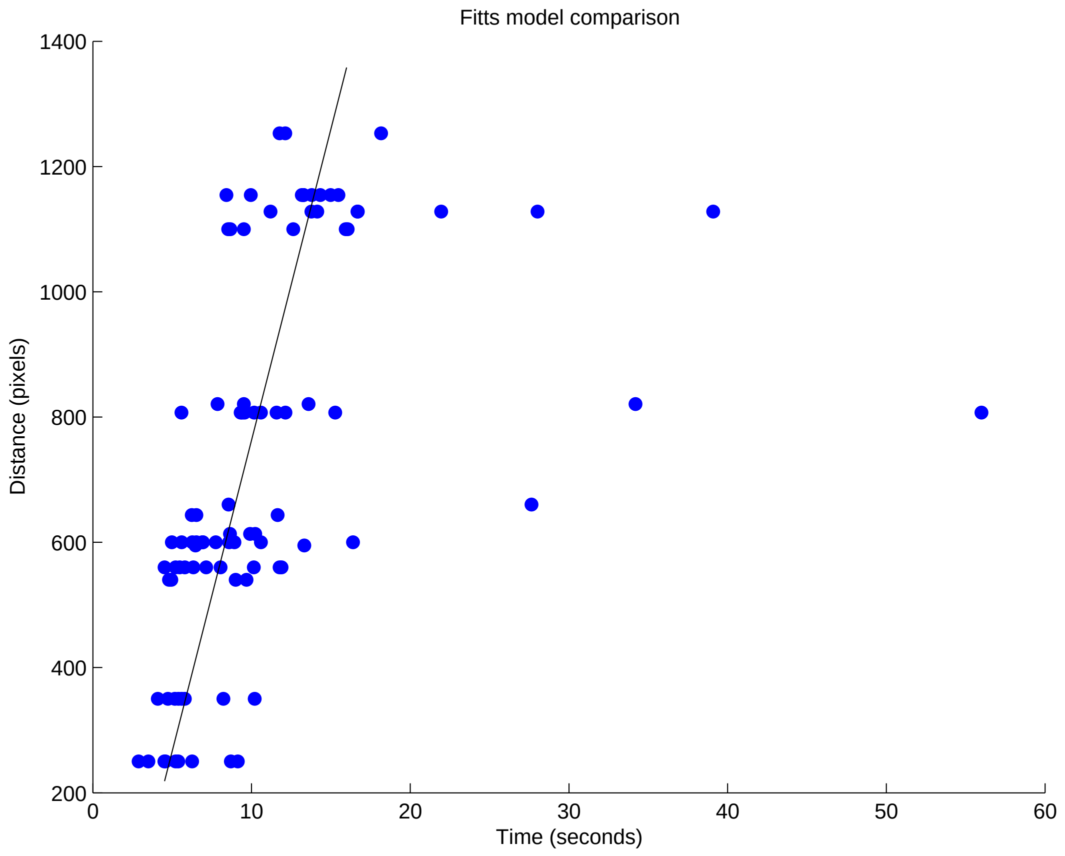


Figure 2.22. Comparison to Fitts' model

improved. Firstly, the host computer and head-tilt mouse software have been refactored and made more robust. Secondly, the calibration and initialization steps have been made more stable while decreasing the duration of these steps. However, there are still areas for future improvement. For example, mouse click functionality could be provided in a hands-free manner. Initially, the head-tilt mouse used a push button, which would not have been usable by some SCI/SCD users, for instance. Also, the head-tilt mouse system currently works only on Microsoft's Windows operating systems with custom software that translates data sent from the mouse to the host computer. If the mouse operated using the Bluetooth protocol's Human Interface Device (HID) profile, which provides the rules and specifications for communications between human interface devices such as mice and keyboards [38], then the host computer software could be removed and the mouse could be used on any operating systems that supports the HID profile. All of these must be done with the goal of increasing performance of the head-tilt mouse. With the first version, from section 2.3, we established a baseline to be improved upon. Our goal was to see how close we can come to the performance of the more traditional forms of cursor control.

The way the head-tilt mouse currently works is that it detects and measures the forward and backward tilt of a person's head (i.e. a nod, or "yes" motion) to move the cursor up and down. For the left and right cursor motion, the mouse uses a tilt motion (not a turn or "no" motion). This tilt motion seemed to be awkward for some of the participants in our initial experiment. Some said that they would have preferred more of a turn of the head to move the cursor left and right. In order to investigate this possibility for improvement, we examined using the Memsic 2125's ability to measure rotation. The Memsic 2125 is able to measure rotation but only in a plane perpendicular to the ground (because of gravity). Further investigation led us to the idea of replacing the Memsic 2125 2-axis accelerometer [40] with a Hitachi H48C 3-axis accelerometer [44]. We also considered offering the ability to switch between alternative control methods to accommodate user preferences and abilities (i.e. some user may prefer the head-tilt and some may prefer the head rotation to control the mouse's X and Y movement). In Figure 2.13 we saw how the user would move the mouse cursor along the x-axis using a tilt motion. In Figure 2.23 we can see how the user would move the mouse cursor along the x-axis using a head shake motion.

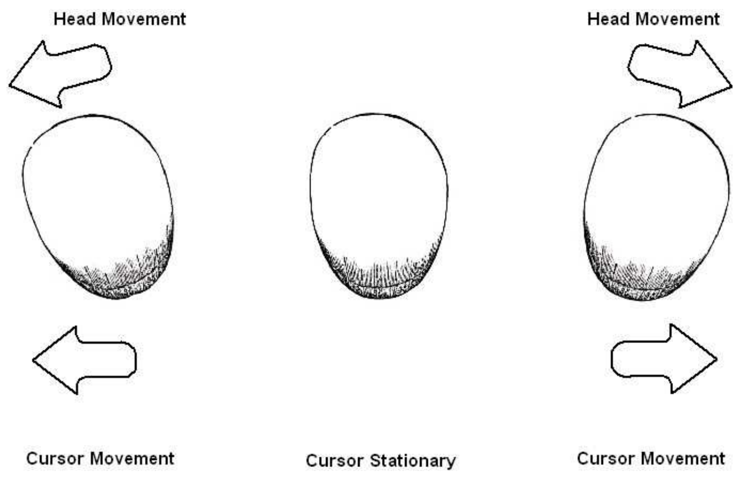


Figure 2.23. Head Shake Movements. Figure adapted from [10].

Since we know that the 2125 can use the X and Y axes to move the cursor up, down, left and right, we considered that the H48C, with its added Z axis that we would be able to use this Z axis for head rotation left and right. Furthermore, this would have freed the Y axis to implement a new mouse click strategy. However, the 3 axis-accelerometer did not work as expected in detecting yaw (i.e. rotation around a vertical axis). The values it returned were not consistent and therefore this approach was abandoned.

The initialization and the operating system generality problems could both be solved if the system supported the Bluetooth HID profile because to the host machine the tilt mouse would appear to be just a regular wireless mouse. To add HID support, we looked at replacing the current EmbeddedBlue [42] Bluetooth radio with one that has on board HID support. We found that BluePacket Communications' BP20422 module [45] might be good candidate as it is inexpensive, supports HID and is rumored [46] to be the module used in the Nintendo Wii controller. One of the problems we encountered while looking into ordering this module was that the company does not take single orders. Therefore, we found an individual reseller, Russ Nelson in August, 2008, who had used the module to implement a shorthand wireless keyboard. However, after discussing the module with him, he informed us that the HID profile is configured at the factory using the module's EPROM and that the ones he was reselling are configured with keyboard HID support only, not mouse HID support [Personal Correspondence]. Furthermore, the leads out of the module are so small that they would require some sort of housing to enable access to them and integration into the head-tilt mouse circuit. As an alternative to finding a Bluetooth radio with HID support, we could implement the HID profile with a generic Bluetooth radio on the microcontroller. While it is doubtful that the memory available to the BASIC Stamp 2 processor (i.e. 32 bytes RAM, 2Kbytes of EEPROM, which is about 500 PBASIC instructions [41]) would support this, it would perhaps be worth the effort to search for one that might. The challenge is to keep the cost low and the size small. This has been added to Chapter 8 as a future work item.

In this chapter we presented a literature review and background by introducing the problem of matching users to devices and specifically to alternative mouse input devices.

Then we discussed the different classes of devices available for hands-free mouse cursor control and their drawbacks. Finally, we presented a head-tilt mouse system and described its implementation in detail. We also showed experimental results comparing the performance on target acquisition tasks of the head-tilt mouse with other traditional hand-operated mouse devices. In the next chapter, we discuss the theory behind the Virtual Dynamic Tunnel algorithm and give background on its inception.

Chapter 3

THEORY

In this this chapter we describe assistive user interface techniques that have been shown to increase performance for cursor control, giving several examples. We also introduce the Virtual Dynamic Tunnel (VDT) algorithm.

Assistive interface techniques have been shown to improve target acquisition performance for motor impaired users using traditional mouse-type controllers [30, 47–56]. The question this research would like to answer is, can such techniques be applied to head-only type controllers and be implemented in a "target-agnostic" fashion while still providing a target acquisition performance gain?

The evaluation of the mouse-type input devices has traditionally been carried out by measuring the time for a user to acquire a target on screen (i.e. difference between the time a user starts to move the cursor and the time the user "clicks" on the target) applying Fitts' Law. Fitts' Law states that the time to acquire a target is related to the size of the target and how far the target is from the starting point. Fitts' Law has been shown to be very useful for this type of evaluation. However, there are limitations to its applicability. For instance, in tasks that require turns or a curvy path (e.g. drawing or nested menu selection) Fitts' Law is not adequate to capture accurate performance. For these curved tasks, the steering law has been applied to input devices with some success. The steering law states that the time to complete a task of moving along a curved path (e.g. a tunnel) is related to the inverse of the width of the path along the trajectory. One of the obvious, but useful observations of Fitts' Law, is that increasing the target size improves overall target acquisition performance. This was found to be true in early experiments in this research as well. It was noticed that most of the targets that were missed, were missed just within a small radius (i.e. 10 to 30 pixels outside radius of a 20 pixel radius target) from the target center but outside the target area. For this reason, a graph (shown in Figure 3.1) was created from the experimental data to show the effect on accuracy of incrementally increasing the target size.

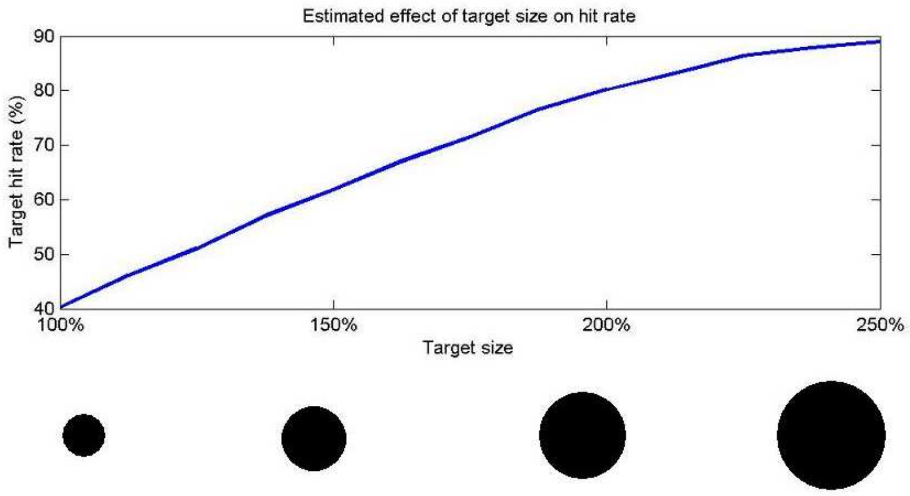


Figure 3.1. Effect of target size on target hit rate [Blackmon and Weeks, 2009]

There has been research in assistive interface techniques to increase input device performance for motion-impaired users and able-bodied users alike. The techniques, derived from this research, either increase target size (virtually or physically) or "straighten" the path to a target (i.e. essentially making the target "close") or a combination of the two. Examples of these techniques are given below.

3.1 Zooming interfaces

One of the assistive interface techniques used to increase input device performance by utilizing Fitts' Law involves simply making the targets larger. This can be done by having everything under the mouse cursor pointer's current position larger, like a sort of magnifying glass. Another way is for the targets to enlarge themselves whenever the mouse pointer is in their vicinity, like the application bar on an Apple Macintosh.

3.2 Haptic gravity wells

Another technique is the use of haptic gravity wells. These wells are centered on a target and when the cursor moves inside the well, a "spring force" pulls the cursor to the center of the target. The result is that the user only has to move the cursor near the target and then wait to click. This technique reduces the number of near target misses by virtually increasing the target size. This technique requires changes in the user interface software so that potential targets have the gravity well functionality. Also, the presence of multiple gravity well targets along a path present a challenge.

3.3 Gain adjustment

Gain can be defined as the ratio of input to output (i.e. amplification). For example, in a standard computer mouse it is the distance the mouse is moved on the desktop compared to how far the cursor moves on the screen. This type of gain is called the Control-Display gain. If the gain is high, then the cursor on the screen moves a long distance compared to the mouse on the desktop. This is good for users who have good motor control and can help increase target acquisition because travel time to targets is reduced. However, for users with motor disabilities, a high gain can reduce target acquisition performance. Systems have been designed that evaluate a users performance and set the gain accordingly. Recently, Wobbrock et al. [53] presented a dynamic gain adjustment technique that uses real-time cursor movement information to set the gain. The technique has the advantage of being "target agnostic" in that the target does not need to be known beforehand and other targets in the path to the actual target do not present a problem. Since the technique sets the gain dynamically, the system can be used by both able-bodied and motor-disabled users alike. However, experiments show that motor-disabled users benefit the most from this technique.

3.4 Haptic tunnels

Langdon et al. created haptic tunnels as an extension of the haptic gravity well to "assist users in moving the cursor in a straight line to the target" [47]. The haptic tunnel is implemented on a special mouse that provides resistance to certain movements. This is accomplished by letting the cursor move freely inside the tunnel. However, when the cursor moves outside a tunnel wall, a spring force ‘pushes’ it back inside as shown in Figure 3.2, where circles indicate a starting position and a target. From experiments, the best tunnels were found to have 0 width and infinite thickness. This would seem to imply that a user performs best when their cursor is always (i.e. no free movement inside a ‘tunnel’) nudged toward the center of the intended path. This has positive implications for the new Virtual Dynamic Tunnel algorithm, mentioned in the next section, because this is the way it works as well. The most obvious drawback of the haptic tunnel technique is that the intended target must be known to the computer in advance.

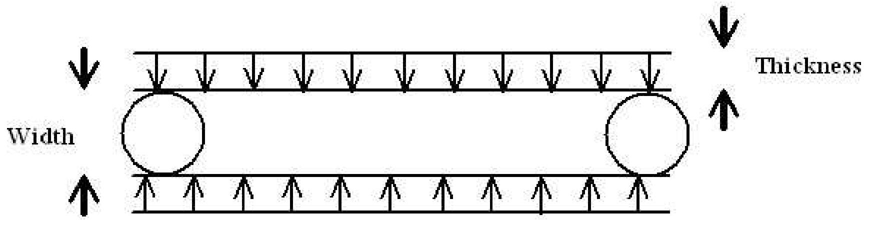


Figure 3.2. Haptic tunnel

3.5 Virtual dynamic tunnels

Assistive interface techniques, such as zooming interfaces, haptic gravity wells, gain adjustment and haptic tunnels, all of which have been shown to improve target acquisition performance for users with motor impairments, could also benefit users of hands-free input devices such as head-operated computer mice. With that in mind, we designed a new assistive technique, a Virtual Dynamic Tunnel (VDT), (Figure 3.3) and evaluate its performance using a head-operated computer mouse and a standard mouse. This technique is not haptic, in that the user does not ‘feel’ resistance to movement. However the mouse cursor itself is ‘nudged’ along a projected target path. An advantage of this is that the technique is applicable on any mouse pointing device and not just haptic ones. Furthermore, unlike most of the previously mentioned assistive interface techniques, the Virtual Dynamic Tunnel algorithm is “target-agnostic.” This means that the intended target does not have to be known beforehand and no special user interface components (e.g. haptic well targets) have to be created.

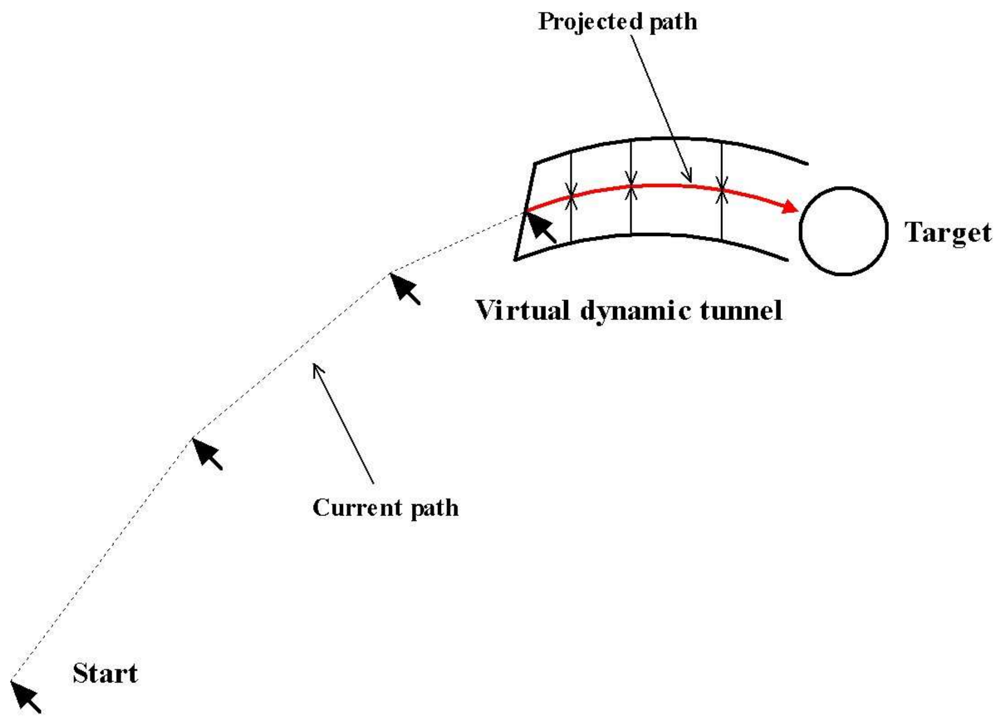


Figure 3.3. Virtual Dynamic Tunnel

Figure 3.3 shows a mouse cursor being moved along a path and the resulting virtual dynamic tunnel created due to this path. As the mouse cursor is moved, movement information is used to construct a virtual dynamic tunnel along a projected path, guiding (or nudging) the cursor along the projected path and subsequently to the intended target. This is done in a similar fashion to the haptic tunnels discussed in the previous section. Details of the implementation of the Virtual Dynamic Tunnel algorithm are covered in the next chapter.

Chapter 4

IMPLEMENTATION OF VIRTUAL DYNAMIC TUNNEL

In this chapter we discuss the details of the implementation of the Virtual Dynamic Tunnel (VDT) algorithm for both a standard USB (Universal Serial Bus) mouse and the head-tilt mouse. The standard (USB) mouse is so called because it is included by manufacturers with a typical, modern PC. The USB designation is because the driver is tied to the USB port and works on any mouse plugged into it, but does not affect, say, the touchpad. We also give an example of execution of the algorithm on a sample data set.

In order to validate and test the VDT algorithm, the algorithm was implemented to work with a standard (USB) mouse and the head-tilt mouse. The standard mouse version works as a Microsoft Windows filter device driver (described in detail below). The head-tilt version of the algorithm was applied inside the mouse movement software that was developed previously for the head-tilt mouse. This new addition to the mouse movement software is also described in detail below.

4.1 Virtual Dynamic Tunnel algorithm details

As the user moves the mouse, the coordinates are passed into the algorithm. The algorithm detects ‘significant’ movement (i.e. the time interval between current position and previous position is not greater than a threshold value, determined through trials and currently set at 100 milliseconds for the filter driver), and if it occurs, the mouse positions are processed by the algorithm.

As mouse position points come in, they are placed into queue b in the following way.

$$b_{x,0} = prevb_{x,N-1} + X$$

$$b_{y,0} = prevb_{y,N-1} + Y$$

$$b_{x,i} = b_{x,i-1} + X$$

$$b_{y,i} = b_{y,i-1} + Y$$

The variables X and Y are the relative motion values reported by the mouse device. The variable *prevb* is the previous b queue values. As can be seen, the first element of b depends on the last value of *prevb*. N is the size of the queues b and *prevb*. Once b is full (i.e. it contains N elements), its contents are copied to *prevb* which is used to construct the weights for our extrapolation function (described in detail below).

$$prevb_x \Leftarrow b_x$$

$$prevb_y \Leftarrow b_y$$

To compute a mouse movement trajectory and subsequently build our virtual dynamic tunnel, we process *prevb* with the barycentric rational interpolation (explained below) and using that, extrapolate the next N points (i.e. those in b). Other interpolation methods were examined such as polynomial, cubic splines and rational functions [57, 58]. These different techniques were compared using data that were representative of the type that mouse movement would produce and the barycentric interpolation produced the best results. The barycentric method gave smooth curves and as is pointed out in [58], and does not produce poles along it's curve, unlike the rational function methods. Cubic splines also produce smooth curves, but they did not give results as good as the barycentric method in our tests. The barycentric method generates a function, given N points. This function, $R(x)$, is given by Equation 4.1.

$$R(x) = \frac{\sum_{i=0}^{N-1} \frac{w_i}{x-x_i} y_i}{\sum_{i=0}^{N-1} \frac{w_i}{x-x_i}} \qquad [58] \tag{4.1}$$

The w_i terms represent weights which are calculated according to Equation 4.2.

$$w_k = \sum_{\substack{i=k-d \\ 0 \leq i < N-d}}^{k} (-1)^k \prod_{\substack{j=i \\ j \neq k}}^{i+d} \frac{1}{x_k - x_j} \qquad [58] \tag{4.2}$$

The barycentric weights were given by equation 4.2. While it may seem while looking at the equations for the barycentric method that the complexity would be a limiting factor in using these equations–keep in mind that in this implementation, N is set to 4 points [1] and the actual computer time to compute a virtual dynamic tunnel was found to be barely one millisecond. To implement the barycentric method, source code from [58] was modified and used. It had to be modified because the algorithm was written in object-oriented C++ and the mouse filter driver was required to use traditional C. This meant that the class constructors had to be replaced with functions that carried out all the same initialization of variables. Then when the structures (previously objects) needed to created, the new functions had to be called explicitly. Also, [58] uses several redefined types in place of 'int,' 'double,' etc. The original types had to be used in place of the redefined types in order for the code to be used in the filter driver.

As mentioned previously, once b is full, it is copied to $prevb$ and $prevb$ is used to generate the barycentric function and the weights for that function on those points. This provides a projection of movement for the next N points and gives us our virtual dynamic tunnel. As the next N points come in the generated function (i.e. the virtual dynamic tunnel) is used and they are compared to the points in the projected tunnel and are moved closer to our projected points by a gravity function. This process is illustrated in Figure 4.1.

Figure 4.1 shows how the Virtual Dynamic Tunnel algorithm is applied to points as they come into the driver function. The point (Cx, Cy) in the figure represents a point on the center of the virtual dynamic tunnel at the current moment in time (i.e. those projected points that are calculated by the barycentric interpolation function). The point (x, y) is the position the mouse device is reporting it would like to place the cursor next. The final point, (new_x, new_y), is the position the virtual dynamic tunnel will actually place the mouse

[1]Four points were found to be a good number through trial and error. It was enough points to produce an approximation of future cursor movement, even in a curve using the barycentric method. Yet, not so many that the extrapolation would start to degrade.

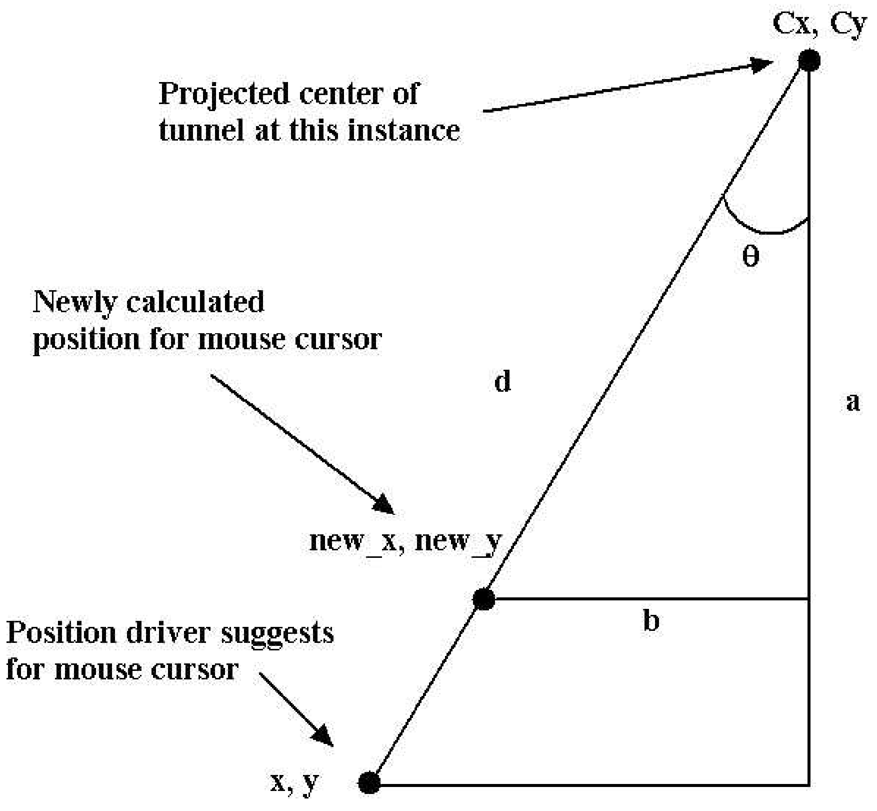

Cx, Cy
Projected center of tunnel at this instance
θ
Newly calculated position for mouse cursor
d
a
new_x, new_y
b
Position driver suggests for mouse cursor
x, y

Figure 4.1. How mouse coordinates are nudged to tunnel center.

cursor. The variable d represents the distance between the center of the virtual tunnel at the current time (Cx, Cy) and the next desired position of the mouse cursor (x, y) and is given by

$$d = \sqrt{(x - Cx)^2 + (y - Cy)^2}.$$

The variable a is the distance from Cy to new_y and is defined by

$$a = \left|(d - d/m)\cos\theta\right|. \tag{4.3}$$

And the variable b is the distance from Cx to new_x and is defined by

$$b = \left|(d - d/m)\sin\theta\right|. \tag{4.4}$$

The angle θ between (Cx, Cy) and (x, y) is

$$\theta = arctan2(y - Cy, x - Cx).$$

The *arctan*2 function gives a sign along with a value so that the quadrant of the angle is taken into account. Finally, the new mouse position is given by

$$new\_x = Cx \pm a.$$

$$new\_y = Cy \pm b.$$

The Virtual Dynamic Tunnel algorithm pseudocode is shown in Figures 4.2 and 4.3. Figure 4.2 is main algorithm and Figure 4.3 is the `apply_virtual_tunnel()` function in detail. In Figure 4.2, on lines 2-14, the variables are defined. Line 2 shows `X` and `Y`, which are passed into the algorithm from the driver code. On line 14, `new_x` and `new_y` are shown, which, at the end of the algorithm, hold the new values for `X` and `Y`. The first thing the algorithm does is check the current position in the queues. If this has reached the maximum size of the queues, we copy the current queues to the previous queues and build the barycentric functions. Otherwise, we process `X` and `Y` normally. We check on line 16 how long it's been since we received movement. If this value is above our threshold (currently set to 100 milliseconds), we clear all our data and start anew. Otherwise, we process the movement normally. Our calls to `apply_virtual_tunnel()` send in `X` and `Y` and send out `new_x` and `new_y`. Notice also that we create two separate barycentric functions in lines 30 and 31. This is in order to handle parametric curves, which are not actually functions, that mouse movement data generates. We do this by creating a constant array (`tb`) that is strictly increasing to act as our *x* coordinate in the calls to create the barycentric functions. This technique was mentioned by Burden and Faires in [59]. However, they used it with LaGrange interpolation.

Figure 4.3 shows the `apply_virtual_tunnel()` function which takes `X` and `Y` as relative mouse movement (in the case of the filter driver) or screen coordinates (in the case of the head-tilt mouse). The function passes back `new_x` and `new_y`, which are updated mouse

```
 1  VARIABLES
 2    X, Y                 : [IN] passed in as relative movement from the device
 3    current_position     : where in queues xb, yb, prev_xb, prev_yb, tb we
 4                           are currently (it's the same for all)
 5    max_q_size           : size of all queues (currently set to 4)
 6    time_since_last_call : elapsed time in milliseconds since previous X and Y were passed in
 7    func1, func2         : our barycentric functions created for extrapolation
 8    tb                     a set of strictly increasing points used to create our barycentric functions
 9                           (one for x and one for y) so that we can handle parametric curves,
10                           which are not functions
11    xb, yb               : the queues where we put newly encountered X and Y points until full
12    prev_xb, prev_yb     : Once xb and yb are full, we copy them here and use them to create the
13                           barycentric functions.
14    new_x, new_y         : [OUT] these hold our new values for X and Y and are passed back to the driver
15
16     if (time_since_last_call >= 100) // time since last point is below threshold.
17            current_pos = 1
18            clear_arrays()
19
20     if (current_position <= max_q_size)
21
22         apply_virtual_tunnel( X [IN], Y [IN], new_x [OUT], new_y [OUT] )
23         current_position = current_position + 1
24
25     else
26
27         prev_xb = xb
28         prev_xy = yb
29
30         construct_barycentric_function( func1 [OUT], tb [IN], prev_xb [IN] )
31         construct_barycentric_function( func2 [OUT], tb [IN], prev_yb [IN] )
32
33         current_pos = 1
34
35         // Apply for last value
36         apply_virtual_tunnel( X [IN], Y [IN], new_x [OUT], new_y [OUT] )
```

Figure 4.2. Virtual Dynamic Tunnel algorithm pseudocode

movement or screen coordinate values after the algorithm has been applied. Lines 6-7 and 11-12 fill the `b` queue while calculating actual screen positions. On lines 19 and 20 the center of the Virtual Dynamic Tunnel is extrapolated from the previously calculated barycentric functions. Line 22 gives the distance between the center of the tunnel and the new point. Line 25 gives the angle, `theta`, so that the distances `a` and `b` from `(Cx, Cy)` to `(new_x, new_y)` can be computed in lines 27 and 28. The rest of the function is used to calculate the new relative cursor movement (filter driver for USB mouse) or screen coordinates (head-tilt mouse).

```
apply_virtual_tunnel( X [IN], Y [IN], new_x [OUT], new_y [OUT] )
{
   // store data points
   if (current_position == 1)

      xb[0] = prev_xb[max_q_size-1] + X
      yb[0] = prev_yb[max_q_size-1] + Y

   else

      xb[current_position-1] = xb[current_position-2] + X
      yb[current_position-1] = yb[current_position-2] + Y


   // Apply previous values
   if ( prev_yb[max_q_size-1] <> 0.0 AND
       prev_xb[max_q_size-1] <> 0.0 )

      Cx = func1( current_position + max_q_size )
      Cy = func2( current_position + max_q_size )

      d = sqrt( (xb[current_position-1]-Cx)^2 +
              (yb[current_position-1]-Cy)^2 )

      theta = arctan2( yb[current_position-1] - Cy, xb[current_position-1] - Cx )

      a = abs((d - d/m) * cos(theta))
      b = abs((d - d/m) * sin(theta))

      if ( xb[current_position-1] >= Cx )
         temp_new_x = Cx + a
      else
         temp_new_x = Cx - a

      if ( yb[current_position-1] >= Cy )
         temp_new_y = Cy - b
      else
         temp_new_y = Cy + b

      new_x = X - xb[current_position-1] - temp_new_x
      new_y = Y - yb[current_position-1] - temp_new_y
}
```

Figure 4.3. The `apply_virtual_tunnel()` function

4.2 Virtual Dynamic Tunnel example

Figure 4.4 shows the operation of the Virtual Dynamic Tunnel on a set of example mouse movement data points. In this figure, it is assumed that the mouse cursor is moving from the top of the figure towards the bottom in a semi-circular manner. The cursor points are denoted by black triangles. During the initial movement, the point queue is filled. Once it's full, the next set of points are predicted by barycentric extrapolation. These extrapolated points are shown as red squares in the figure. When the next point—after the initial four—comes

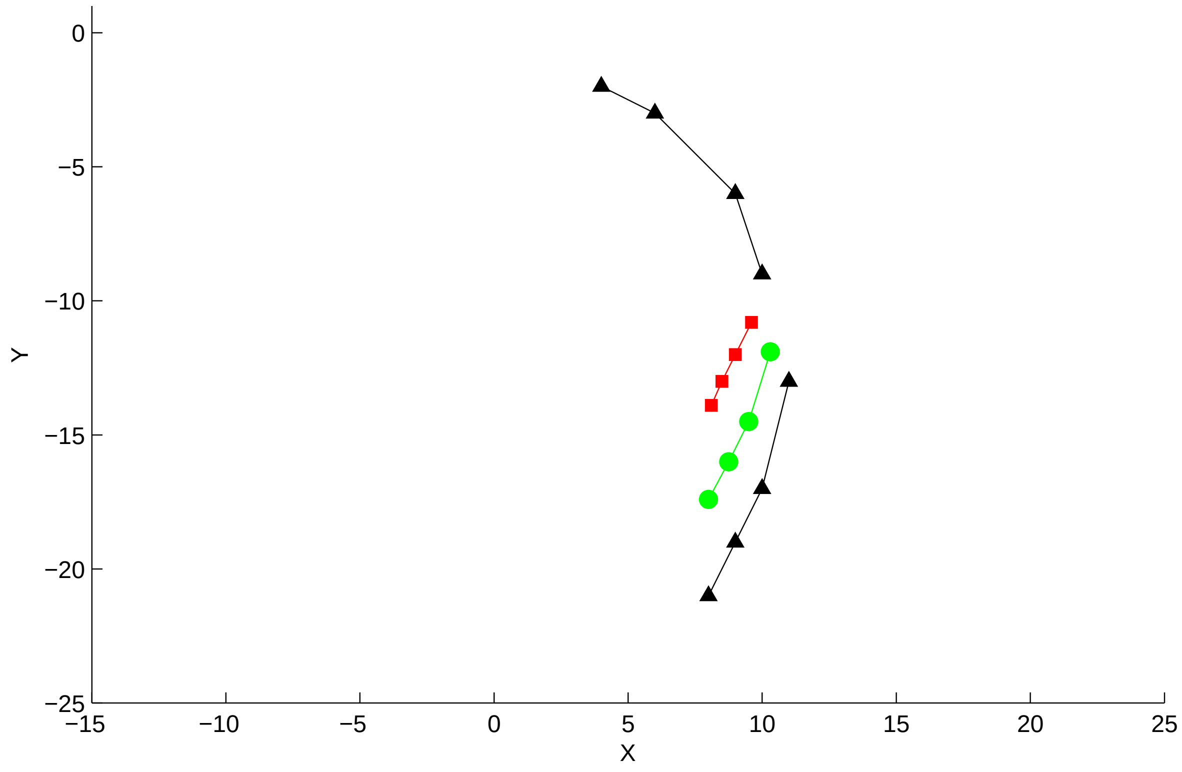


Figure 4.4. Virtual Dynamic Tunnel algorithm walk-through.

in, it is compared to the projected point and ‘nudged’ towards the projected point, which represents the tunnel center. The new points are shown as green circles. The subsequent three cursor points that come in are processed likewise, while the next queue is being filled.

The first four movement points are (4, -2), (6, -3), (9, -6) and (10, -9). For the x coordinate the weights from Equation 4.2 are $w_0 = -1$, $w_1 = 2$, $w_2 = -2$ and $w_3 = 1$. Remember, to handle parametric curves, the x and y coordinates had to be split into two barycentric function calls. Furthermore, since our artificial x coordinate, t, is the same for both x and y, our weights end up being the same for all calls to the barycentric functions. For the initial set of data and the given weights, Equation 4.1 yields the points (9.6, -10.8), (9, -12), (8.5, -13) and (8.1, -13.9)—shown in red in Figure 4.4. As new cursor points come in, we apply the Virtual Dynamic Tunnel algorithm to them. The new points that come in, in order, are (11, -13), (10, -17), (9, -19) and (8, -21)—shown in black. Applying the algorithm, yields points (10.3, -11.9), (9.5, -14.5), (8.75, -16) and (8.0, -17.4)—shown in green.

4.3 Implementation for standard (USB) mice

Here, the mouse driver that implements the Virtual Dynamic Tunnel is described. In Microsoft Windows, a device may have several device drivers. The main device driver is called the *function driver*, because it's responsible for the functioning of a device. A *filter device driver* can reside above or below the function driver (shown as *mouclass* in Figure 4.5) in the device object stack. The filter drivers work by *filtering* I/O request packets[2] that flow through them [61, 62].

Initially, for the standard mouse, the dynamic virtual tunnel algorithm was implemented as a user application, taking the cursor position and updating it according to the algorithm. The reason for writing a device driver rather than implementing the dynamic virtual tunnel algorithm in application software is because even though a user application can get and set the mouse cursor position, a user application has to get access to mouse data from the system software, which has already updated the cursor position, caused by physical movement of the mouse device by the time an application retrieves the cursor position. Therefore, any changes in mouse cursor position applied by application-level software will then occur after 'normal' position changes are done by the system software. Essentially, both the application software and the system software are updating the mouse cursor at the same time with the result being a 'jumpy' mouse cursor. However, when updates to the mouse cursor position are done further down the software stack, the system software sees the assistive-algorithm-updated mouse cursor data as coming *directly* from the mouse device (via the mouse driver stack) and the jumpiness does not occur. This situation and solution are illustrated in Figure 4.5.

In this figure, the column of boxes is the mouse driver stack, with an optional filter driver shown installed. To the right of the filter driver is a box representing our assistive interface algorithm. The filter driver reads mouse movement data from the port driver below it; can update or discard that data; then sends the updated data to the *mouclass* function driver above it. The *mouclass* driver is where the system software (shown in the box in the top right of the figure) gets its mouse movement information. The application software (i.e.

[2]I/O request packets are data structures used to hold information about I/O requests, which are sent between devices, the operating system and device drivers [60].

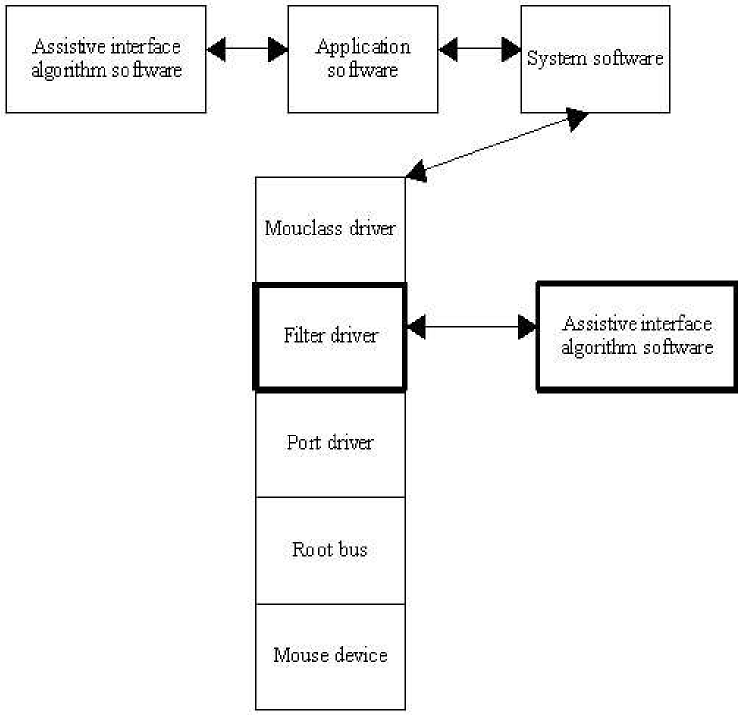


Figure 4.5. Microsoft Windows mouse driver stack with filter driver.

the box in the top center) can represent any application running on the computer. In the case of testing our assistive interface algorithms, it would be our testing application. The box in the top left of Figure 4.5 was the originally proposed location of our assistive interface algorithm. A Microsoft Windows application can get and set the cursor position by using the `GetCursorPos()` and `SetCursorPos()` function calls in the Microsoft Windows API. Calls to these functions are used by the system software to get and update the mouse cursor position. At the same time, the system software is setting the mouse cursor position by responding to mouse device movement via the *mouclass* function driver. The new mouse filter driver reads mouse device movement data from the lower level driver and, using our assistive interface algorithm, changes the data before sending it up to the system software, which moves the mouse accordingly. Our test application no longer needs to call `SetCursorPos()`. The mouse cursor position will have been updated using our assistive algorithm inside the driver filter code. If no movement occurs, the callback is not called.

4.4 Development of the filter driver

To the develop the filter driver, the Microsoft Windows Driver Development Kit (DDK) was used. The *moufiltr* sample filter driver was modified to support the Virtual Dynamic Tunnel algorithm. The kernel-level mode of drivers does not support linking of the floating-point portions of the C runtime library for Microsoft Windows, even though the x87 math coprocessor supports them. Therefore, several floating-point functions needed from the C runtime had to be written in inline assembly. Among these are the `atan2` (arctangent), `sin`, `cos`, `sqrt` (square root) and `ftol` (a floating point to long conversion function). Development of kernel-mode drivers requires debugging the code on a separate machine from the one where development is taking place. One way this can be done by using a software-based virtual machine. Initially, this approach was taken, but for the mouse filter driver, the cursor movements on the host versus the virtual machine were interfering with each other. So, to facilitate the debugging, a separate physical computer, connected with a firewire cable, was used. This setup worked very well. Also to support debugging the actual algorithm, a test program was written that would parse and process mouse movement and debugging data that the filter driver would write out. Besides not supporting the math C runtime library, the driver environment that was used, while supporting some C++ constructs, also did not support object-orientation. This required modifications to the barycentric libraries so that they would work with the driver code. Finally, the setup information file (.inf) had to be modified for the filter driver.

4.5 Installation and configuration of filter driver

The filter driver is currently targeted to be installed and run on the Microsoft Windows XP operating system. Other Windows versions are supported by the Driver Development Kit, and adding support for those operating system versions would be a simple matter of recompiling the code for those platforms and applying modifications to the driver's setup information file (.inf).

The installation of the filter is illustrated by the following figures. Assuming that the driver component files are in a folder on the target computer, right-click on the 'My Computer' and select 'Manage' from the popup menu items shown. This will bring up the 'Computer Management' dialog shown in Figure 4.6.

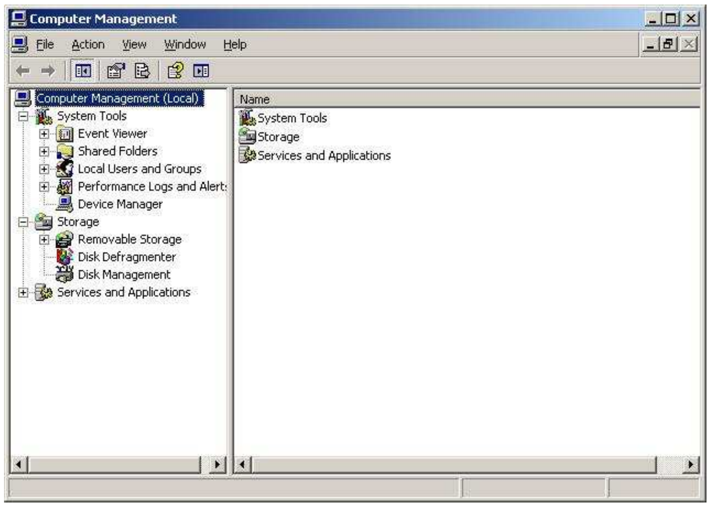


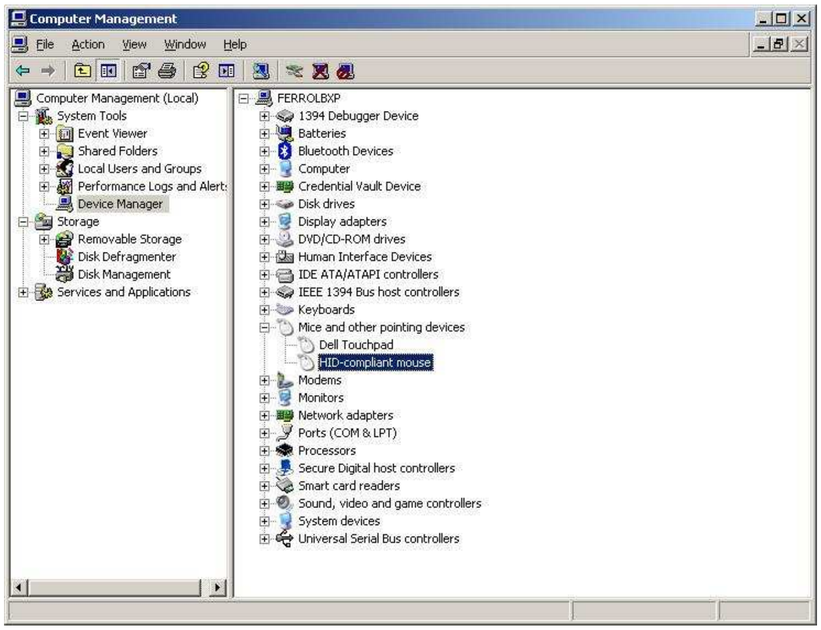
Figure 4.6. Installation – Computer Management.

From the left panel, select 'Device Manager' under 'System Tools'. Then, on the right panel, find and expand 'Mice and other pointing devices' and find and select 'HID-compliant mouse.' This is shown in Figure 4.7. Right-click on 'HID-compliant mouse' and select 'Update driver...'

Then, the dialog in Figure 4.8 will be shown. The question 'Can Windows connect to Windows Update to search for software?' will be presented. Choose 'No, not at this time' to this question and click 'Next >.'

On the next screen of the 'Hardware Update Wizard,' shown in Figure 4.9, select 'Install from a list or specific location (Advanced)' to the question 'What do you want the wizard to do?'

The 'Hardware Update Wizard' will now ask if it should search for the driver as shown in Figure 4.10. Choose 'Don't search. I will choose the driver to install.'



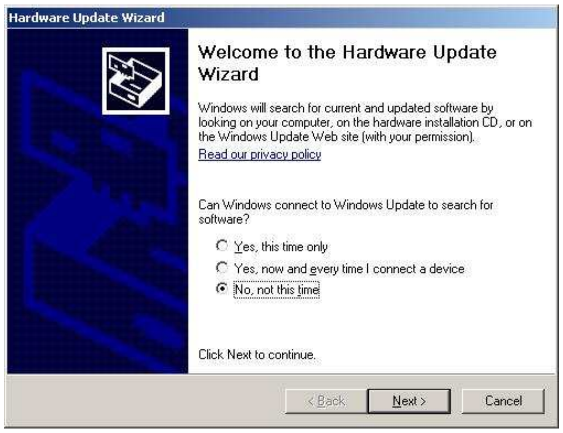
```
Computer Management (Local)
  System Tools
    Event Viewer
    Shared Folders
    Local Users and Groups
    Performance Logs and Alert:
    Device Manager
  Storage
    Removable Storage
    Disk Defragmenter
    Disk Management
  Services and Applications
```

```
FERROLBXP
  1394 Debugger Device
  Batteries
  Bluetooth Devices
  Computer
  Credential Vault Device
  Disk drives
  Display adapters
  DVD/CD-ROM drives
  Human Interface Devices
  IDE ATA/ATAPI controllers
  IEEE 1394 Bus host controllers
  Keyboards
  Mice and other pointing devices
    Dell Touchpad
    HID-compliant mouse
  Modems
  Monitors
  Network adapters
  Ports (COM & LPT)
  Processors
  Secure Digital host controllers
  Smart card readers
  Sound, video and game controllers
  System devices
  Universal Serial Bus controllers
```

Figure 4.7. Installation – HID-compliant mouse.

On the following screen will ask to specify the driver to install as shown in Figure 4.11. Click the ‘Have Disk...’ button. This will bring up the ‘Install from disk’ dialog shown in Figure 4.12. From this dialog, click the ‘Browse...’ button.

Now, navigate to the folder containing the Virtual Dynamic Tunnel filter driver and select the setup information file (.inf) as shown in Figure 4.13. The file is named *moufiltr.inf*. Click the ‘Open’ button.



Welcome to the Hardware Update Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

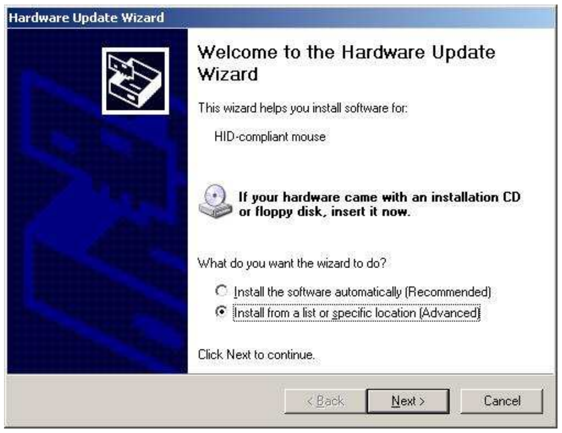
Read our privacy policy

Can Windows connect to Windows Update to search for software?

- Yes, this time only
- Yes, now and every time I connect a device
- [✓ No, not this time]

Click Next to continue.

Figure 4.8. Installation – Hardware Update Wizard.


Hardware Update Wizard

Welcome to the Hardware Update Wizard

This wizard helps you install software for:

HID-compliant mouse

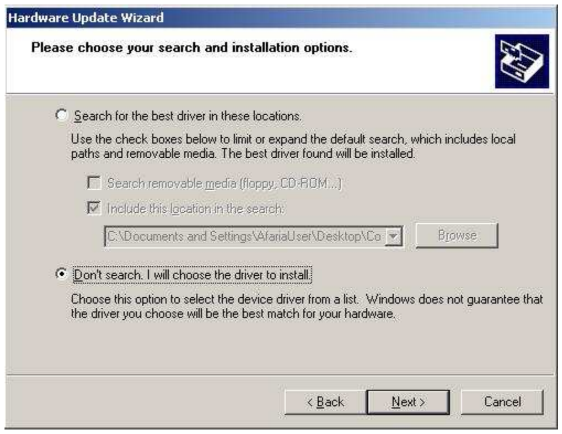
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

Install the software automatically (Recommended)
[✓ Install from a list or specific location (Advanced)]

Click Next to continue.

Figure 4.9. Installation – Install from specific location.


Hardware Update Wizard

Please choose your search and installation options.

Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

Search removable media (floppy, CD-ROM...)
[✓ Include this location in the search:]
C:\Documents and Settings\AfariaUser\Desktop\Co

[✓ Don't search. I will choose the driver to install.]

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Figure 4.10. Installation – Don't search.

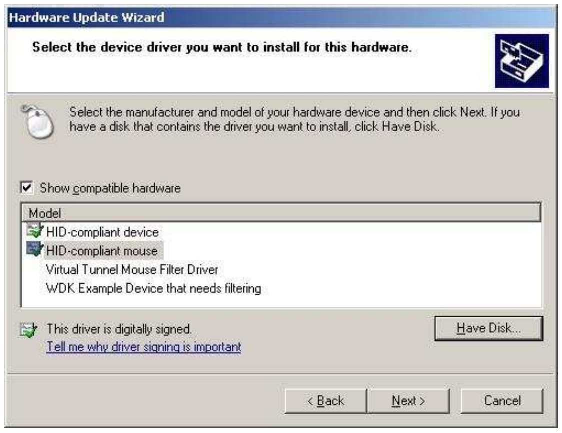
In the dialog shown in Figure 4.14, select 'Virtual Tunnel Mouse Filter Driver' and click the 'Next >' button.

Then, a dialog warning about Windows Logo testing will be shown as indicated in Figure 4.15. Click the 'Continue Anyway' button.

Now the driver components will be copied and installed as shown Figure 4.16

Upon completion of the installation, the final screen of the 'Hardware Update Wizard' will be shown as indicated in Figure 4.17. Click the 'Finish' button to close the wizard.

Now 'Virtual Tunnel Mouse Filter Driver' should be seen in the 'Device Manager' window as shown in Figure 4.18. Close the 'Computer Management' window. The driver is now installed.


Hardware Update Wizard

Select the device driver you want to install for this hardware.

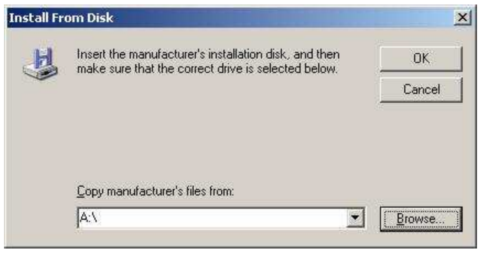
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.

☑ Show compatible hardware

Model
HID-compliant device
HID-compliant mouse
Virtual Tunnel Mouse Filter Driver
WDK Example Device that needs filtering

This driver is digitally signed.
Tell me why driver signing is important

Figure 4.11. Installation – Select driver.


Install From Disk

Insert the manufacturer's installation disk, and then make sure that the correct drive is selected below.

Copy manufacturer's files from:
A:\

Figure 4.12. Installation – Install from disk.

The filter driver reads the contents of a configuration file to configure itself. This file must be named *config.txt* and it must be placed in the root directory of the primary drive of the computer. Example contents of the configuration file are shown in Figure 4.19. To enable the Virtual Dynamic Tunnel algorithm, enter a '1' after 'alg:' in the *config.txt* file. To disable the algorithm, without having to uninstall the filter driver, enter a '0' after 'alg:'. The 'dist:' refers to the variable m in equations 4.3 and 4.4 and affect the distance incoming points are nudged to the center of the virtual dynamic tunnel. The final setting in *config.txt*, 'queue:' is not currently used and can be ignored.

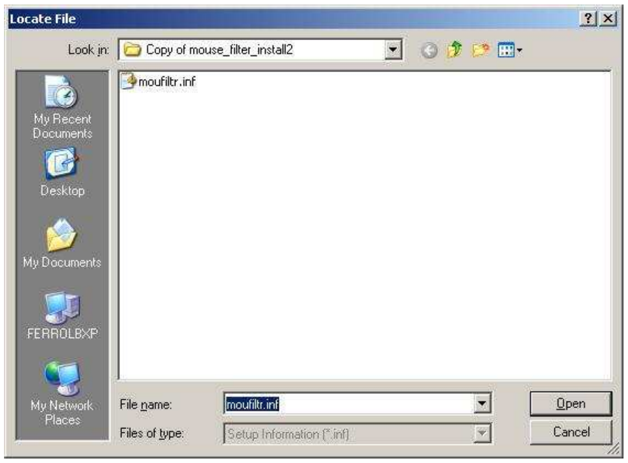
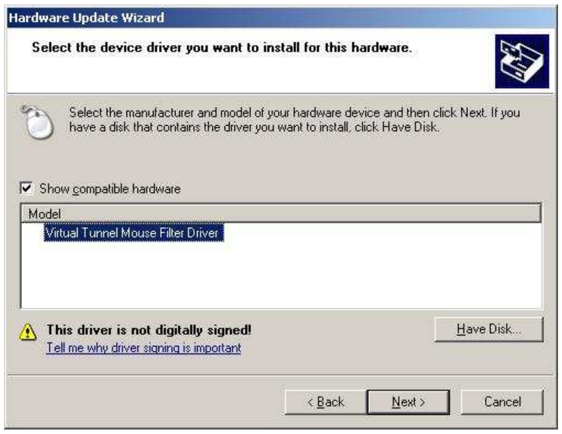


Figure 4.13. Installation – Locate file.


Hardware Update Wizard

Select the device driver you want to install for this hardware.

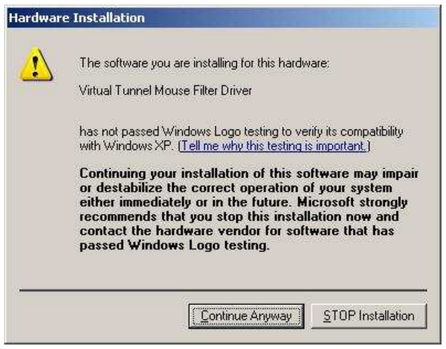
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.

☑ Show compatible hardware

Model
Virtual Tunnel Mouse Filter Driver

This driver is not digitally signed!
Tell me why driver signing is important

Figure 4.14. Installation – Select the driver.


Hardware Installation

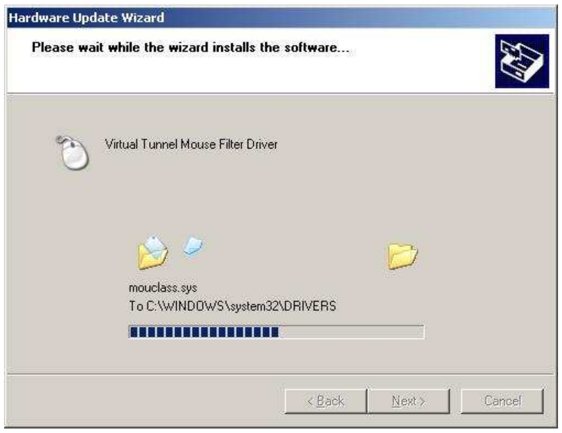
The software you are installing for this hardware:

Virtual Tunnel Mouse Filter Driver

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

Figure 4.15. Installation – Continue.

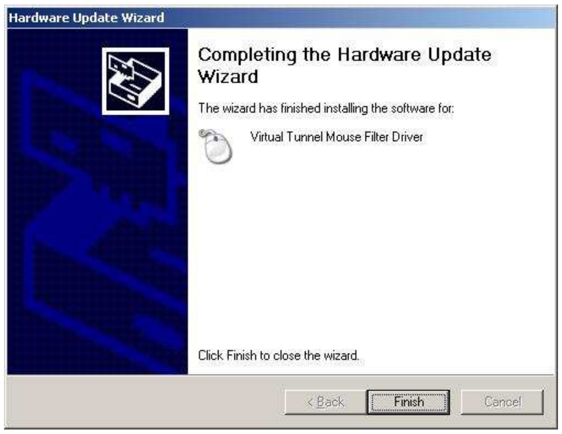

Hardware Update Wizard

Please wait while the wizard installs the software...

Virtual Tunnel Mouse Filter Driver

mouclass.sys
To C:\WINDOWS\system32\DRIVERS

Figure 4.16. Installation – Copying driver files.


Hardware Update Wizard

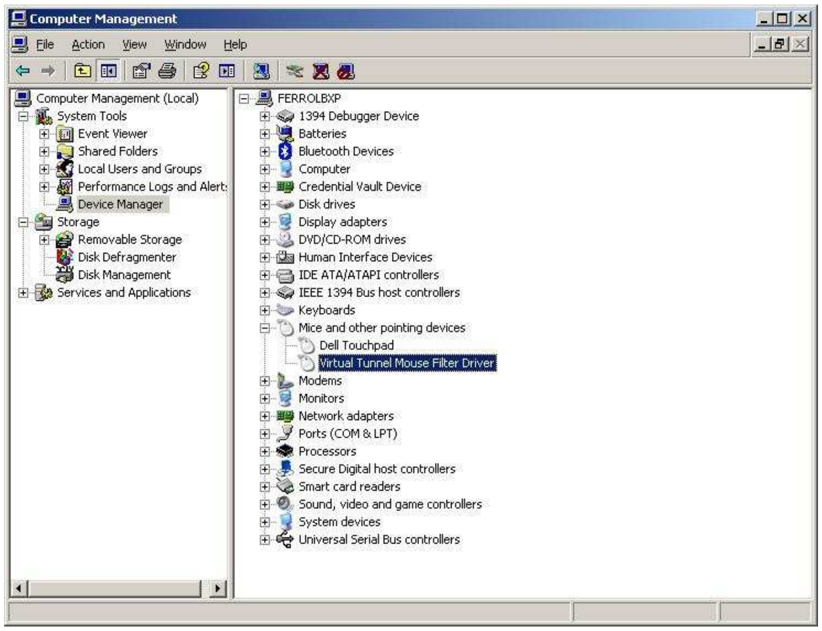
Completing the Hardware Update Wizard

The wizard has finished installing the software for:

Virtual Tunnel Mouse Filter Driver

Click Finish to close the wizard.

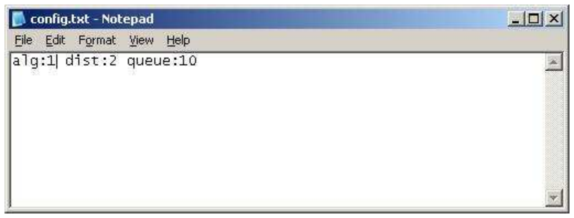
Figure 4.17. Installation – Finish installation.


Computer Management

Computer Management (Local)
- System Tools
  - Event Viewer
  - Shared Folders
  - Local Users and Groups
  - Performance Logs and Alert:
  - Device Manager
- Storage
  - Removable Storage
  - Disk Defragmenter
  - Disk Management
- Services and Applications

FERROLBXP
- 1394 Debugger Device
- Batteries
- Bluetooth Devices
- Computer
- Credential Vault Device
- Disk drives
- Display adapters
- DVD/CD-ROM drives
- Human Interface Devices
- IDE ATA/ATAPI controllers
- IEEE 1394 Bus host controllers
- Keyboards
- Mice and other pointing devices
  - Dell Touchpad
  - Virtual Tunnel Mouse Filter Driver
- Modems
- Monitors
- Network adapters
- Ports (COM & LPT)
- Processors
- Secure Digital host controllers
- Smart card readers
- Sound, video and game controllers
- System devices
- Universal Serial Bus controllers

Figure 4.18. Installation – Virtual Tunnel Mouse Filter Driver.



```
alg:1 dist:2 queue:10
```

Figure 4.19. Configuration of filter driver.

4.6 Implementation for head-tilt mouse

Since the movement of mouse cursor position for the head-tilt mouse is completely controlled by how the head-tilt mouse application interprets signals from the head-tilt mouse device (thereby acting as its own driver), the assistive interface techniques are implemented directly in the head-tilt mouse application and the filter driver is not used. Also, since we have control over cursor position (i.e. we get and set these values as opposed to being sent the values by the operating system) the x and y coordinates are absolute. So the only difference between the Virtual Dynamic Tunnel algorithm for the head-tilt mouse and mouse filter driver is that in lines 6, 7, 11 and 12 from Figure 4.3 the head-tilt mouse uses the `X` and `Y` values directly rather than accumulating relative device motion as in the filter driver. The block diagram for the Virtual Dynamic Tunnel algorithm is shown in Figure 4.20.

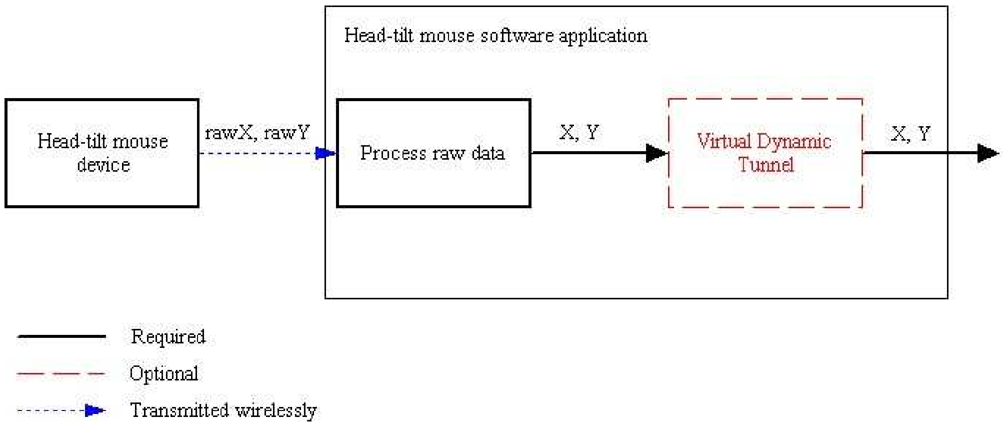


Figure 4.20. Virtual Dynamic Tunnel for head-tilt mouse block diagram.

Furthermore, since the runtime environment of the head-tilt mouse software is at the application level, rather than the kernel mode like the filter driver, C++ object-orientation and constructs were able to be used. This means, also, that the barycentric function library was used directly without modification. It also meant that the C runtime math functions like `atan2` (arctangent), `cos` and `sqrt` were used directly and did not need new implementations.

To use the head-tilt mouse with the Virtual Dynamic Tunnel algorithm, open a command window by clicking 'Start' on the lower left-hand side of the Windows XP screen and select 'Run'. At the 'Open:' prompt, type 'cmd'. This will open a command window. Use 'cd' to navigate to the directory where the `commport.exe` application is stored. At the command prompt type 'commport 1 COMNN' where COMNN is the name of the COMM port associated with the Bluetooth radio. The Bluetooth COMM port must be set up prior to running the head-tilt mouse. The usage of `commport.exe` is:

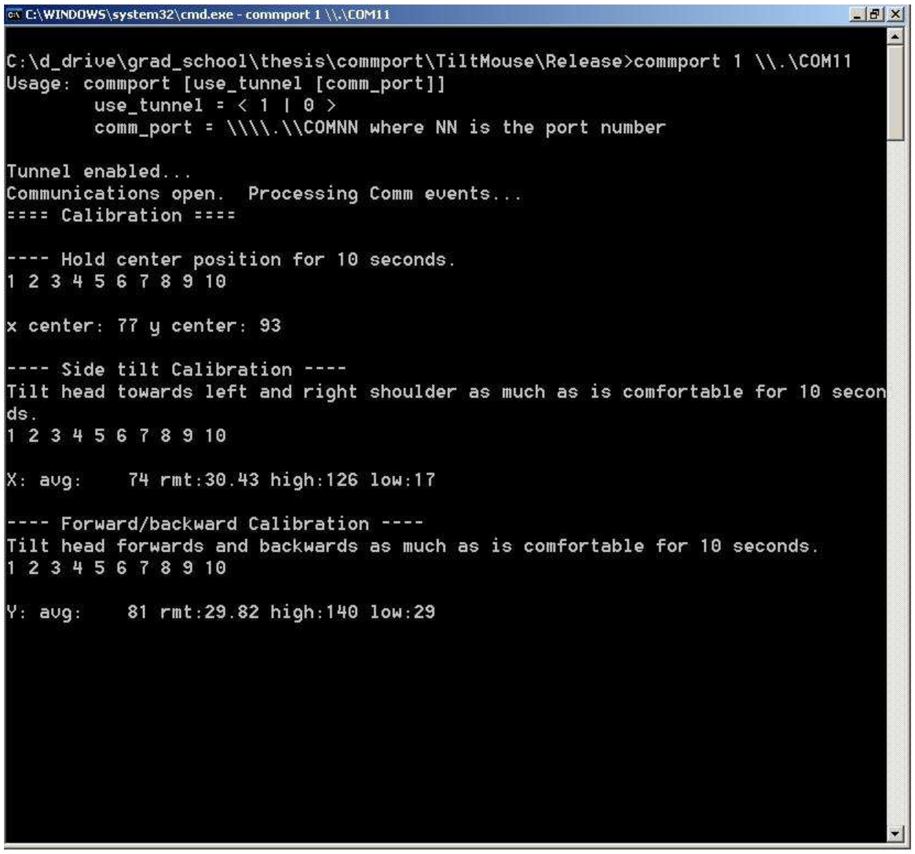
```
commport [use_tunnel [comm_port]]
    use_tunnel = < 0 | 1 >
    comm_port  = \\\\.\\COMNN where NN is the port number.
```

The first parameter determines whether or not the Virtual Dynamic Tunnel algorithm is used. The second parameter is the previously set up COMM port.

4.6.1 Calibration/configuration of head-tilt mouse

Once the head-tilt application starts, the head-tilt mouse is put through a 30 second calibration. This calibration configures the head-tilt mouse to movements of each individual so that it will move the cursor in a way that is comfortable for that user. The first 10 seconds, the head-tilt mouse wearer must hold a comfortable, center position while the software sets this as the point where no mouse movement is to take place. The next 10 seconds, the wearer must move their head to and fro, to their left and right shoulders alternately. This movement will subsequently be used to move the mouse cursor left and right on the screen (i.e. x coordinate motion). During the final 10 seconds the wearer will nod their head in a 'yes'-type motion and this will be used to move the mouse cursor up and down on the screen (i.e. y coordinate motion). After the final calibration step, the wearer has control of the mouse cursor with the head-tilt mouse using the Virtual Dynamic Tunnel algorithm. Figure 4.21 shows an example run of `commport.exe` and the calibration of the head-tilt mouse.

This chapter detailed the implementation of the VDT algorithm for a standard (USB) mouse and the head-tilt mouse. An example run of the algorithm was presented and the installation and configuration of the algorithm software was described. In the next chapter



```
C:\WINDOWS\system32\cmd.exe - commport 1 \\.\COM11

C:\d_drive\grad_school\thesis\commport\TiltMouse\Release>commport 1 \\.\COM11
Usage: commport [use_tunnel [comm_port]]
        use_tunnel = < 1 | 0 >
        comm_port = \\\\.\\COMNN where NN is the port number

Tunnel enabled...
Communications open.  Processing Comm events...
==== Calibration ====

---- Hold center position for 10 seconds.
1 2 3 4 5 6 7 8 9 10

x center: 77 y center: 93

---- Side tilt Calibration ----
Tilt head towards left and right shoulder as much as is comfortable for 10 secon
ds.
1 2 3 4 5 6 7 8 9 10

X: avg:    74 rmt:30.43 high:126 low:17

---- Forward/backward Calibration ----
Tilt head forwards and backwards as much as is comfortable for 10 seconds.
1 2 3 4 5 6 7 8 9 10

Y: avg:    81 rmt:29.82 high:140 low:29
```

Figure 4.21. Head-tilt mouse application – commport.exe.

we introduce an experimental designed which is used to evaluate the effectiveness of the VDT algorithm for two different mouse steering tasks.

Chapter 5

EXPERIMENTAL DESIGN

In this chapter we detail the experimental design that is used to evaluate the effectiveness of the VDT algorithm using two different steering tasks and two different device types. We discuss the testing software application, the experiment procedure and the data analysis also.

5.1 Description

To test our hypothesis that "target-agnostic" (i.e. the target does not need to be indicated beforehand [53]) defined assistive interface techniques would improve the target acquisition performance of users of head-only mouse controls and possibly users of traditional mouse input devices, we design the following experiments. A new test application has been written to use the assistive interface techniques and performance evaluation of steering.

In designing experiments with input devices, it is important to avoid testing bias (i.e. bias introduced via users gaining skill with the devices as the experiments are carried out) [1]. To control the testing bias, the Solomon four-group[1] experiment design is being used. The experiments will be carried out with both able-bodied and disabled users (when appropriate).

5.2 Trials with traditional mouse

The first set of experimental trials will be performed with the traditional mouse. To control for testing bias, the Solomon four-group experimental design will be used as shown in Table 5.1. In this design, experiment participants will be randomly placed into one of four groups (indicated by the two left columns of the table). The 'R' in the table indicates random assignment of the participants to the four groups. The 'O' indicates an observation

[1]The Solomon four-group design is an experimental design approach meant to reduce the learning effect [1] that would likely be present in our experiments involving computer input devices.

Table 5.1. Solomon four-group design. Adapted from [1].

	Time 1		Time 2	
	Assign	Train	Intervene	Test
Group 1	R	O_1	X	O_2
Group 2	R	O_3		O_4
Group 3	R		X	O_5
Group 4	R			O_6

will be performed. The subscripts on 'O' indicate different sets of data that will be collected for each observation. For our experiments, this will involve having the users run our test application; using a certain mouse input device (e.g. the traditional mouse in these first experimental trials); with or without assistive interface techniques employed and recording the users' performance data. The 'Train' column is where we will provide a training phase to some groups of users. Notice that in this new experiment design, we will not provide a training phase to all user groups. This is how the Solomon design controls for testing bias. By not providing a pre-test or training phase to all groups and using random group assignment, we can calculate the effect training has on our final results. The 'Intervene' column of Table 5.1 indicates whether or not we apply the assistive interface technique we are testing. In these experiments, we run the trials with the traditional mouse or the head-tilt mouse, where the intervention is the dynamic tunnel algorithm, mentioned in section 3.5 and discussed in Chapter 4, and analyze the results. The final column of Table 5.1 indicates that observational data is collected during trials whether or not an intervention takes place (i.e. with or without training and with or without intervention). Notice that in all cases, that the fourth group is the control group.

Table 5.2 shows what the raw results gathered from the experiment will look like. Each group in each trial will have the mean acquisition time and the mean number of errors reported. The first two groups of each trial will also have these values for the training phase. Since groups three and four of a trial do not have a training phase, no data will be collected for training. This is shown for groups TS_3, TS_4, HG_3 and HG_4 in the table. The meanings of the two-letter with subscript group names are explained below.

Table 5.2. Example: raw results table

	Train		Test	
	Mean target acquisition	Mean # errors	Mean target acquisition	Mean # errors
TS_1	XX.XX	XX.XX	XX.XX	XX.XX
TS_2	XX.XX	XX.XX	XX.XX	XX.XX
TS_3			XX.XX	XX.XX
TS_4			XX.XX	XX.XX
...	...	...	...	...
...	...	...	...	...
HG_1	XX.XX	XX.XX	XX.XX	XX.XX
HG_2	XX.XX	XX.XX	XX.XX	XX.XX
HG_3			XX.XX	XX.XX
HG_4			XX.XX	XX.XX

Table 5.3. Traditional mouse with straight path.

	Time 1		Time 2	
	Assign	Train	Intervene	Test
Group TS_1	R	O_1	X	O_2
Group TS_2	R	O_3		O_4
Group TS_3	R		X	O_5
Group TS_4	R			O_6

The experiments planned using the traditional mouse are detailed in Tables 5.3 and 5.4. The two-letter user group names are derived by using ‘T’ or ’H’ (indicating the device type used; ‘T’ for traditional mouse and ‘H’ for head mouse) for the first letter and ‘S’ or ‘C’ (indicating straight or curved target acquisition) for the second letter.

Table 5.3 shows the trial using the traditional mouse tested against straight path target acquisition using dynamic tunnel algorithm as the assistive technique, with four randomly assigned groups where the group number is indicated by ‘TS_n’ where n is a number corresponding to the group. The letter ‘R’ indicates that group membership will be assigned randomly. The letter ‘O_n’ represents an observation (i.e. data is collected at this point)

Table 5.4. Traditional mouse with curved path.

	Time 1		Time 2	
	Assign	Train	Intervene	Test
Group TC_1	R	O_1	X	O_2
Group TC_2	R	O_3		O_4
Group TC_3	R		X	O_5
Group TC_4	R			O_6

Table 5.5. Head mouse with straight path.

	Time 1		Time 2	
	Assign	Train	Intervene	Test
Group HS_1	R	O_1	X	O_2
Group HS_2	R	O_3		O_4
Group HS_3	R		X	O_5
Group HS_4	R			O_6

where n is the n^{th} set of observed data. Finally, 'X' indicates where an intervention took place. In this trial, that would be the application of the dynamic tunnel algorithm.

Table 5.4 represents the trial using the traditional mouse tested against a curved path using the dynamic tunnel algorithm. The groups are assigned randomly as indicated by 'R' and are denoted by 'TC_n' where n is a number corresponding to the group. 'O_n' represents an observation where n is the n^{th} set of observed data. The intervention 'X' indicates the application of the dynamic gain adjustment.

5.3 Trials with head-tilt mouse

The experimental trials planned with the head-only mouse are detailed in Tables 5.5 and 5.6. The only difference between these set of trials is the device used. Likewise, the two-letter user group names are derived in a similar fashion, except using 'H' for the first letter.

Table 5.5 shows the trial using the head-only mouse tested against straight path target acquisition using dynamic tunnel algorithm as the assistive technique, with four randomly

Table 5.6. Head mouse with curved path.

	Time 1		Time 2	
	Assign	Train	Intervene	Test
Group HC_1	R	O_1	X	O_2
Group HC_2	R	O_3		O_4
Group HC_3	R		X	O_5
Group HC_4	R			O_6

assigned groups where the group number is indicated by 'HS_n' where n is a number corresponding to the group. As with the traditional mouse trials, the letter 'R' indicates that group membership will be assigned randomly. Also, the letter 'O_n' represents an observation (i.e. data is collected at this point) where n is the n^{th} set of observed data. Finally, 'X' indicates where an intervention took place. In this trial, that would be the application of the dynamic gain adjustment.

Table 5.6 represents the trial using the head-only mouse tested against a curved path using the dynamic gain adjustment. The groups are assigned randomly as indicated by 'R' and are denoted by 'HC_n' where n is a number corresponding to the group. 'O_n' represents an observation where n is the n^{th} set of observed data. The intervention 'X' indicates the application of the dynamic gain adjustment.

5.4 The testing software

To carry out the experiments, a new testing application was written in the Java programming language. The application works by recording mouse cursor movement information as the cursor is moved around inside the application window. During a trial, the application presents one of two different types of mouse cursor movement tasks to the participant in form of "tunnels." A participant's objective is to move the mouse cursor through a tunnel as quickly and accurately as possible, keeping the cursor as close to the center of a tunnel as possible. Once the cursor reaches the end of a tunnel, the trial is over.

The first task is a straight tunnel as shown in Figure 5.1. When participants are presented with a straight tunnel, they move the mouse cursor into the tunnel, past the starting line on the left-hand side, and the trial starts. The color of the tunnel changes to indicate the start of a trial and changes again to indicate the end of a trial. Green is used to indicate the tunnel (in circular trials, yellow is used) is in a pre-trial state, magenta indicates that the trial has begun and cyan indicates that the trial is over. Mouse movement and timing data are recorded during the trial. Also different assistive techniques are applied or not depending on which group a participant has been assigned to. The tunnel width is configurable as a variable in the steering law so that it is a contributing factor in calculating performance. Currently, the straight tunnel is fixed as a horizontal tunnel.

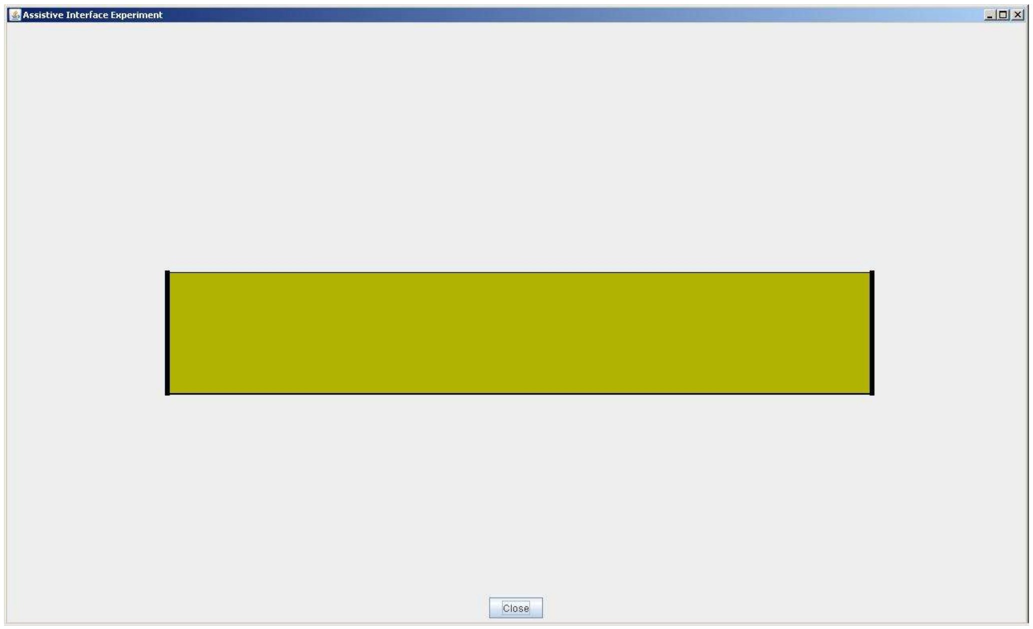


Figure 5.1. Screen shot of test application showing straight tunnel trial type

The second task is a circular movement task wherein the participant moves the mouse cursor inside a circular tunnel as shown in Figure 5.2.

Once inside the tunnel, the cursor is moved to the top of the circular tunnel (see the vertical line). Once the cursor crosses the line at the top of the tunnel (in a clockwise direction), the trial begins. The trial is to see and record how fast the user can move the mouse cursor around the circular tunnel one full revolution, keeping as close to the middle

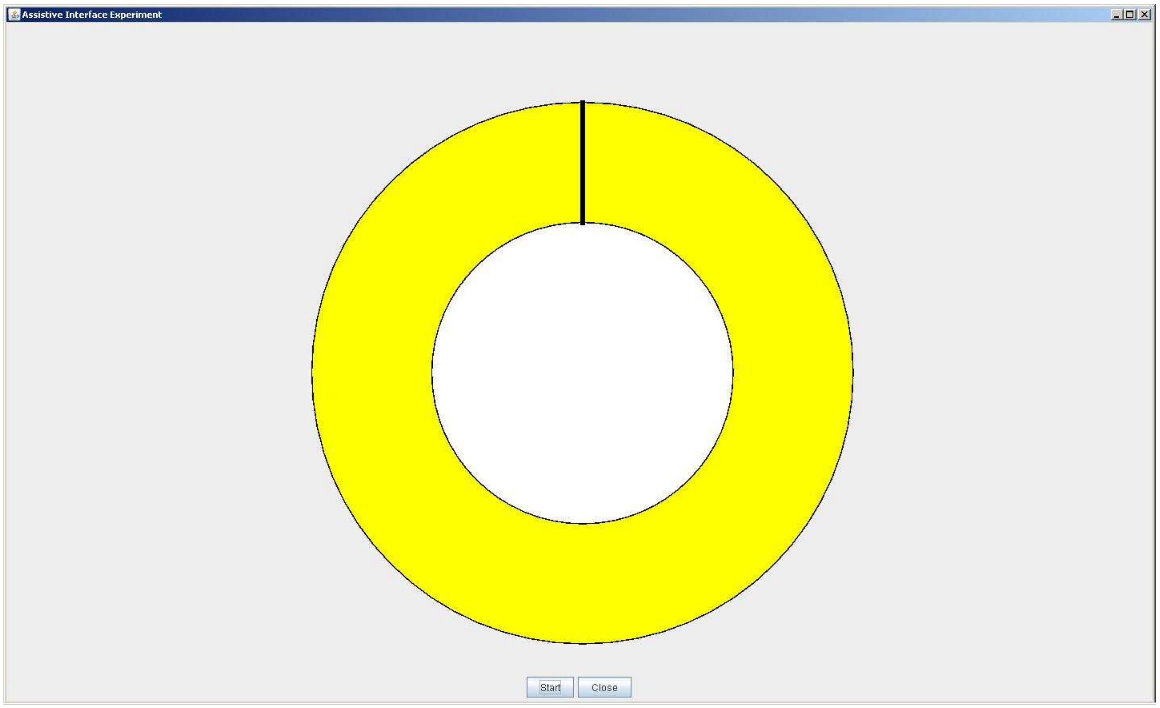


Figure 5.2. Screen shot of test application showing circular tunnel trial type

of the tunnel as possible. The mouse cursor position is recorded throughout the trial, as are tunnel boundary crossings. Mouse movement and timing data are recorded for each trial. The trial ends when the user reaches back to the top of the circular tunnel – the same place the trial started. Movement in the circular tunnel is currently fixed at clockwise and this is explained to the participants. The tunnel changes color to indicate that a trial has started and when a trial is finished. Yellow is used to indicate the tunnel is in a pre-trial state, magenta indicates that the trial has begun and cyan indicates that the trial is over. Assistive techniques are applied as mentioned in the previous section, depending on which group a participant is randomly assigned to.

Why only two different types of tunnels? As mentioned in [22] by Accot and Zhai and first suggested in [23], most pointing tasks can be captured with a combination of only circle and straight movements.

5.5 The procedure

This section describes, step by step and in detail, the procedure that was followed to carry out the experiments. The first step was to select date(s), time(s) and location(s) to run the experiments.

5.5.1 Set the date(s), time(s) and location(s)

A date, time and location was selected that was convenient for the participants to take part in the experiments. The main location chosen was the research section of the Georgia State University (GSU) library (i.e. the location of the initial experiments). The only real requirement was access to a power outlet for our test laptop and a desk or table. Once the date(s), time(s) and location(s) were selected, we next had to acquire the participants. Each participant took part in the experiment one at a time. An initial pilot was run (not unlike a dress rehearsal) to determine the amount of time needed for each trial type so that an adequate amount of time was given to conduct the trials without interfering with the next scheduled participant.

5.5.2 Acquire participants

For the initial experiment, volunteer participants were acquired from the university population by posting flyers around the university describing the experiment and offering some money for the participants' time. For this round of experiments, we got the participants one at a time by displaying a sign and asking library patrons. Once we had volunteers, and at the point where they came in for the experiment, they were be assigned to the experimental groups, described in Sections 5.2 and 5.3. All participants are anonymous in that no personal data including their actual name was collected with the exception of age, sex and whether they were able to use a traditional mouse and a head-operated mouse.

5.5.3 Interview participants

The interview consists of a few basic questions to determine the user's ability to operate a standard mouse and a head mouse. Also during the interview, the participant is told what was going to happen in the experiment. Further, it establishes whether a participant is comfortable with going through with the experiment. The primary purpose of the interview is to inform potential participants about the experiment and also to gauge whether or not they would be willing to participate. Furthermore, any questions or concerns they have were addressed here as well. If a potential participant seemed willing and able, they were given a disclosure statement (see Figure 5.3) to read, date and sign. The disclosure statement puts into writing what was discussed during the interview and clearly states that participation is voluntary; that a participant may stop at any time; and describes the main steps of the experiment. The interview culminated with the administration of the pre-experiment questionnaire. In the actual experiments, no participant declined to participate.

5.5.4 Administer pre-experiment questionnaire

The purpose of the pre-experiment questionnaire is to record the information gathered from the interview, and to fully determine if a participant is able and willing to participate in the experiment. The form is also used to record information about the participant, like age, sex and group assignment. The form asks participants if they are comfortable using a traditional mouse and/or a head mouse (i.e., do they suffer any pain or discomfort in the hands, wrists, head or neck etc.). It asks the participants to rate their perceived ability with a traditional mouse as well as how well they think they could operate a head mouse. It also asks how many years they have used a standard computer mouse. The form is shown in Figure 5.4. In addition to Yes/No questions to determine if the participant will be comfortable using the two mouse devices, there are two "closed-ended with ordered choice," [63] Likert-type [63, 64] questions as well. One asks the participant to gauge their skill with a traditional mouse and the other ask them to rate their physical coordination in general. As a participant finishes filling out the pre-experiment questionnaire, they will be assigned to a group.

5.5.5 Assign participants to groups

The experiments require four trials–one for each device type (traditional mouse and head mouse), and one for each path type (straight and curved). Each trial has four groups. Participants are assigned to two *different* groups – one for the traditional mouse and one for the head mouse. They are in the same group for both trials (i.e. straight and circular paths) of a device type. At the experiment site, two containers initially contained four cards. The containers were labeled corresponding to the device type. The cards correspond to each group, and were also be labeled for its container's trials. As a participant came in, he/she randomly selected a card from each container, thus being assigned to a group for each device type. This assignment information was captured on the pre-experiment questionnaire and the collected data files were named accordingly. When the fourth participant selected the only remaining card from the containers, all the cards were reshuffled and put back into their corresponding containers for the next set of participants. After a participant had been assigned to a group, the experiment was run.

5.5.6 Training

The participants who are assigned to a group that receives training for a device type will be trained in the following manner. In addition to the verbal instruction and description provided to all participants, those who are to be trained are told that they will receive a training phase and that it does not count as the actual test, but is used to get them acquainted with the device/environment. This phase consists of an extra trial for each tunnel type (i.e. straight and circular). This extra trial/training phase (indicated as 'Time 2' in Figure 5.1) is only given once to a participant without repeat. Therefore, the user may not be fully proficient with the device/environment before the actual test phase begins (indicated as 'Time 2' in Figure 5.1). Also the intervention is applied or not dependent on the group assignment so that the intervention is the same in the training phase as it is in the regular phase. Data, identical to what is collected in a regular trial is also collected in the training phase. However, the data are labeled and saved as training data so that it may be analyzed later as such. It is very probable that a participant may receive training on one device type and not the other or even training on both or neither depending on which groups they are assigned to for the two device types.

5.5.7 Run experiment

Depending on which groups the participant had been assigned, the trials were run as described in Sections 5.2 and 5.3. For each task (i.e. a line in Table 5.2 for a group), data files were named and stored for the participant, anonymously identifying the participant as a member of their assigned group. After the participant was finished with the experiment, they were asked to fill out a post-experiment questionnaire.

5.5.8 Administer post-experiment questionnaire

The purpose of the post-experiment questionnaire was to capture the opinions of the participants about the experiment and perhaps ways they would suggest to improve it and any and all thoughts they may have. The form consists of five "close-ended ordered" [63] Likert-type [63, 64] questions which are set to gauge the participants' opinion of using the devices for the experiment. The form also contains two "open-ended" [63] question which are meant to capture participants' overall attitude about the experiment and to give them a space to offer suggestions for improvement. At this point, the participant was thanked and paid for their participation. The post-experiment form is shown in Figure 5.5

5.6 Data analysis

The null hypothesis in these experiments is that there will be no significant difference between means of target acquisition in groups which apply the "target-agnostic" techniques and those which do not. To test our alternate hypothesis: that "target-agnostic" assistive interface techniques be applied to head-only type controllers and result in a target acquisition performance gain, we need to analyze the data from the experiments performed and see if there is a "significant" difference in target acquisition performance from where we applied the interventions and where we did not. To compare differences between mean target acquisition times, paired repeated-measures[2] *t-test* will be used [1, 65, 66]. The *t-test* is used to test whether a null hypothesis is true by determining if a difference exists between means (e.g. in these experiments, it would be the difference between groups who had an intervention and groups that did not). To calculate the t statistic the following formula is used.

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

In this formula, $\bar{x}$ is the mean target acquisition time measured for an intervention. μ_0 is the mean target acquisition time measure without an intervention. The difference between these values is divided by $s/\sqrt{n}$, where s is the standard deviation of the means and n is the number of subjects in the trial [65]. To determine if the difference between the means is significant (i.e. in order to reject the null hypothesis) we select our α to be 0.05 where α is the probability that t is greater than some value – in this case, 1.645, as show here.

$$\alpha = P(t > 1.645) = 0.05$$

Setting α to 0.05 is standard in statistical analysis [1, 65, 66]. Also we will simply state the observed values of t and α and have the readers draw their own conclusions. In order to construct confidence intervals, Bonferroni will be used. Bonferroni states that, assuming the null hypothesis is true, if the probability that all tests pass (i.e. the difference of the means between the groups with an intervention and those without) is less than or equal to α/n, then the probability that some tests pass is less than or equal to α [67]. This can be shown by the following. If

$$P(T_i \text{ passes } |H_0) \leq \frac{\alpha}{n}$$

for $1 \leq i \leq n$, then

$$P(\text{some } T_i \text{ passes } |H_0) \leq \alpha.$$

[2]In paired repeated-measures, subjects are tested before and after an intervention [65].

The experiments will require four trials (i.e. two device types; and two assessed task types), which are indicated by a single table in the previous discussion. The human subjects must be divided into four groups for each trial. Therefore, data will be collected for $4 \times 4 = 16$ groups. It is possible that the human subjects could take part in multiple trials, assuming they are able to (i.e. their abilities do not prevent using a device type). The human subjects cannot, however, take part in a different group in the same trial. Another way of describing α is that it is the probability of getting a Type-I error[3]. On the other hand, β is the probability of getting a Type-II error[4]. In order to determine an acceptable minimum group size for the experiments, power analysis was used. Power is defined as follows.

$$power = (1 - \beta)$$

Therefore, it is the probability that we will not get a Type-II error (i.e. the probability of finding a difference between the means if one exists). Power goes up with group size. What we need to know is what group size will produce an acceptable power. To determine this, the power analysis software, G*Power, was used [68]. The graph in Figure 5.6 was produced by G*power. The x-axis indicates sample sizes and the resultant power is indicated on the y-axis. As can be seen, 10 participants in a group should produce a power level of 0.75.

Assuming a participant is able to use both devices (i.e. the head-tilt mouse and the traditional one) they should be able to be reused for each of the four trials. It would be desirable to get disabled participants. However, the hypothesis is general enough to test without this being a requirement. When a participant shows up for an experiment, he/she will be randomly assigned to a group in each of the four trials. Therefore, to get between 5-10 participants for each group in a trial, we will need 20-40 total participants in our experiments.

The graph in Figure 5.7 shows the power for different sample sizes for independent *t-tests*, which is what would be used to compare performance across trials, which will be

[3]Type-I errors occur when the null hypothesis is rejected even when it is actually true [65].

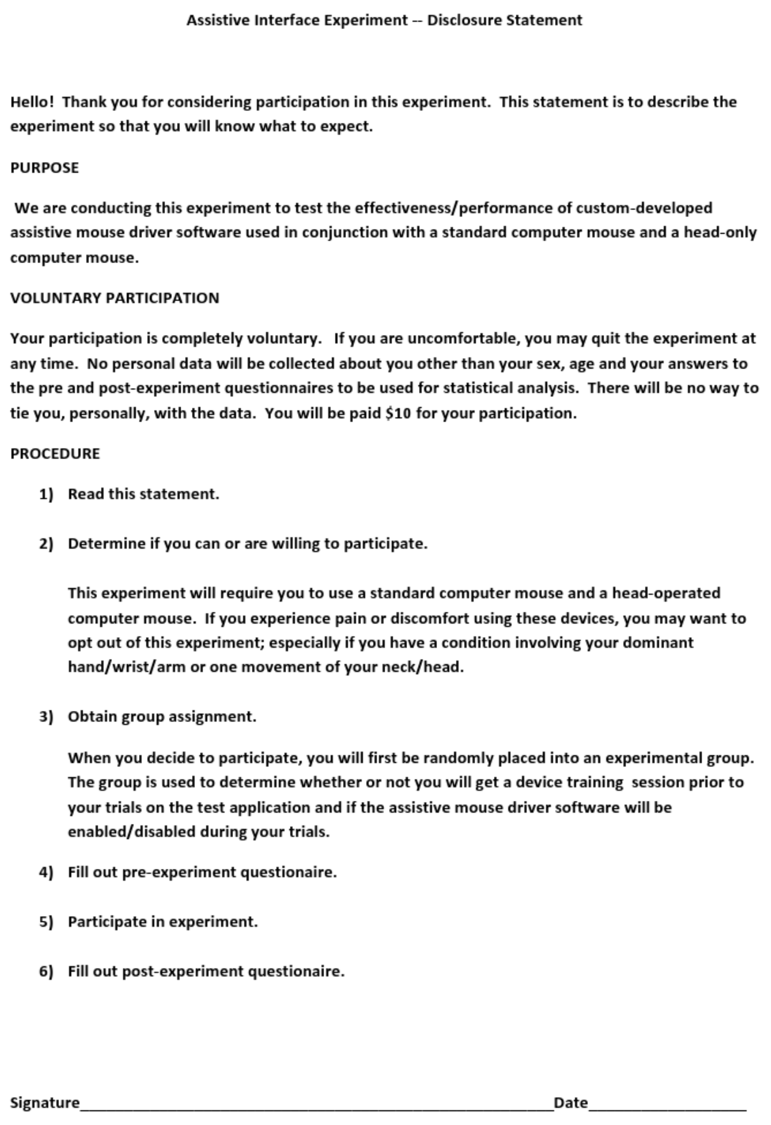
[4]Type-II errors occur when the null hypothesis is false and it's accepted [65].

done to compare the effectiveness of the difference assistive techniques. If we have 20-40 participants, the requirement for power should be acceptable.

5.7 Steering law and Fitts' law

In these experiments, we are comparing performance with and without the VDT algorithm for a device type. Since our tunnels (both circular and straight) do not change width or length and we do not change device type, all the variables in the steering law equations 2.2 and 2.3 are constant. We assume a and b are constants for each device. The same is true when applying Fitts' law to the circular and straight trial types (i.e. all the variables are constant) in these experiments. Therefore, the results are consistent with Fitts' and the Steering laws.

In this chapter we detailed the experimental design that we use to evaluate the effectiveness of the VDT algorithm. In the following chapter, we provide details on the results from this experiment.



Assistive Interface Experiment -- Disclosure Statement

Hello! Thank you for considering participation in this experiment. This statement is to describe the experiment so that you will know what to expect.

PURPOSE

We are conducting this experiment to test the effectiveness/performance of custom-developed assistive mouse driver software used in conjunction with a standard computer mouse and a head-only computer mouse.

VOLUNTARY PARTICIPATION

Your participation is completely voluntary. If you are uncomfortable, you may quit the experiment at any time. No personal data will be collected about you other than your sex, age and your answers to the pre and post-experiment questionnaires to be used for statistical analysis. There will be no way to tie you, personally, with the data. You will be paid $10 for your participation.

PROCEDURE

1) **Read this statement.**

2) **Determine if you can or are willing to participate.**

   This experiment will require you to use a standard computer mouse and a head-operated computer mouse. If you experience pain or discomfort using these devices, you may want to opt out of this experiment; especially if you have a condition involving your dominant hand/wrist/arm or one movement of your neck/head.

3) **Obtain group assignment.**

   When you decide to participate, you will first be randomly placed into an experimental group. The group is used to determine whether or not you will get a device training session prior to your trials on the test application and if the assistive mouse driver software will be enabled/disabled during your trials.

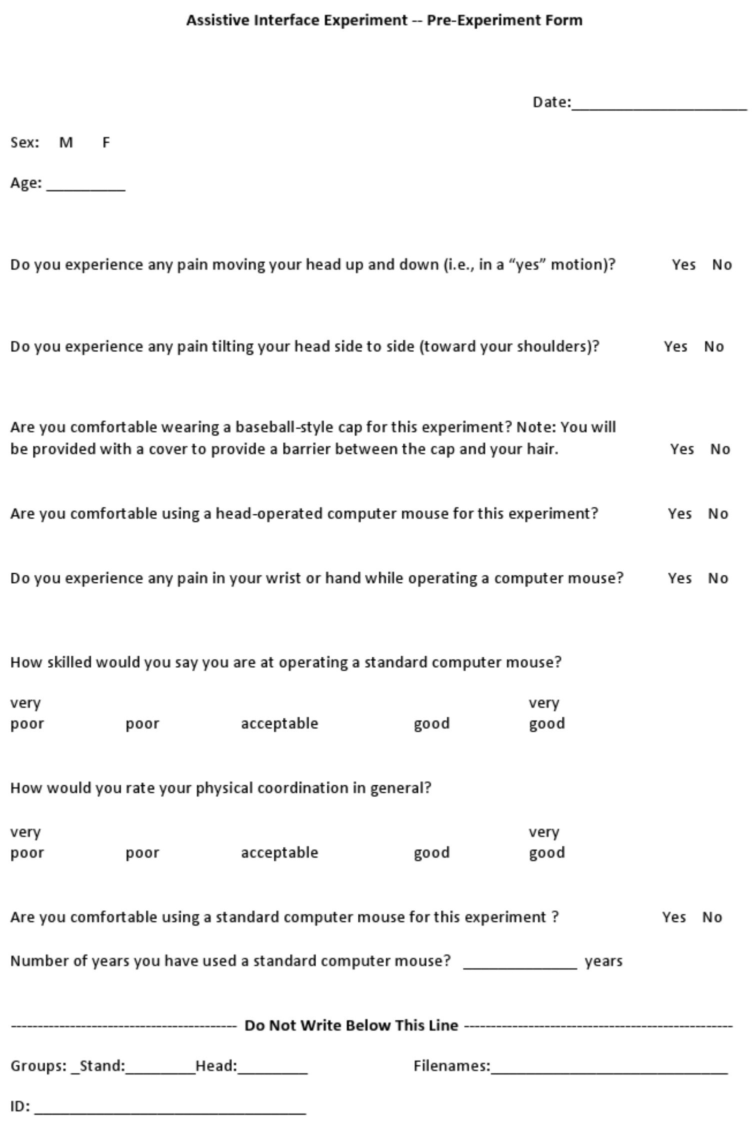
4) **Fill out pre-experiment questionaire.**

5) **Participate in experiment.**

6) **Fill out post-experiment questionaire.**

Signature____Date____________________

Figure 5.3. Experiment disclosure statement



Assistive Interface Experiment -- Pre-Experiment Form

Date:_____________________

Sex: M F

Age: _________

Do you experience any pain moving your head up and down (i.e., in a "yes" motion)? Yes No

Do you experience any pain tilting your head side to side (toward your shoulders)? Yes No

Are you comfortable wearing a baseball-style cap for this experiment? Note: You will be provided with a cover to provide a barrier between the cap and your hair. Yes No

Are you comfortable using a head-operated computer mouse for this experiment? Yes No

Do you experience any pain in your wrist or hand while operating a computer mouse? Yes No

How skilled would you say you are at operating a standard computer mouse?

very poor	poor	acceptable	good	very good

How would you rate your physical coordination in general?

very poor	poor	acceptable	good	very good

Are you comfortable using a standard computer mouse for this experiment ? Yes No

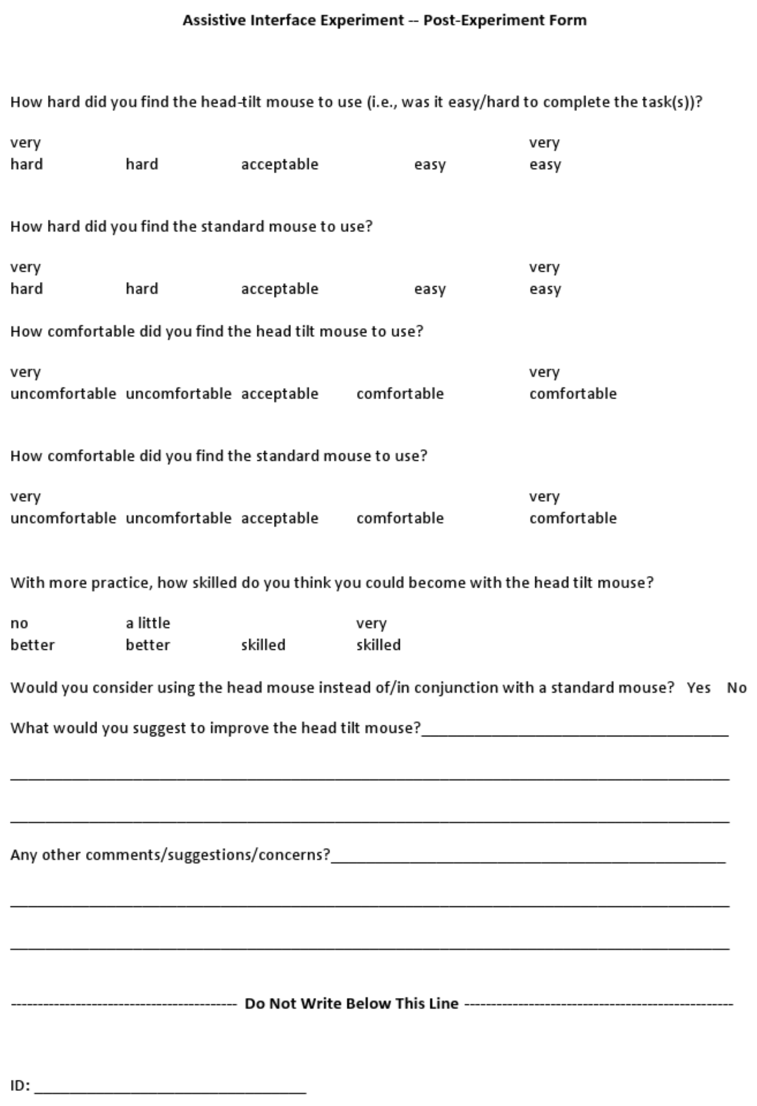
Number of years you have used a standard computer mouse? _____________ years

-- **Do Not Write Below This Line** --

Groups: _Stand:________Head:________ Filenames:____________________________

ID: _________________________________

Figure 5.4. Pre-experiment questionnaire



Assistive Interface Experiment -- Post-Experiment Form

How hard did you find the head-tilt mouse to use (i.e., was it easy/hard to complete the task(s))?

very hard	hard	acceptable	easy	very easy

How hard did you find the standard mouse to use?

very hard	hard	acceptable	easy	very easy

How comfortable did you find the head tilt mouse to use?

very uncomfortable	uncomfortable	acceptable	comfortable	very comfortable

How comfortable did you find the standard mouse to use?

very uncomfortable	uncomfortable	acceptable	comfortable	very comfortable

With more practice, how skilled do you think you could become with the head tilt mouse?

no better	a little better	skilled	very skilled

Would you consider using the head mouse instead of/in conjunction with a standard mouse? Yes No

What would you suggest to improve the head tilt mouse?______________________________________

____

____

Any other comments/suggestions/concerns?____

____

____

-- **Do Not Write Below This Line** --

ID: ________________________________

Figure 5.5. Post-experiment questionnaire

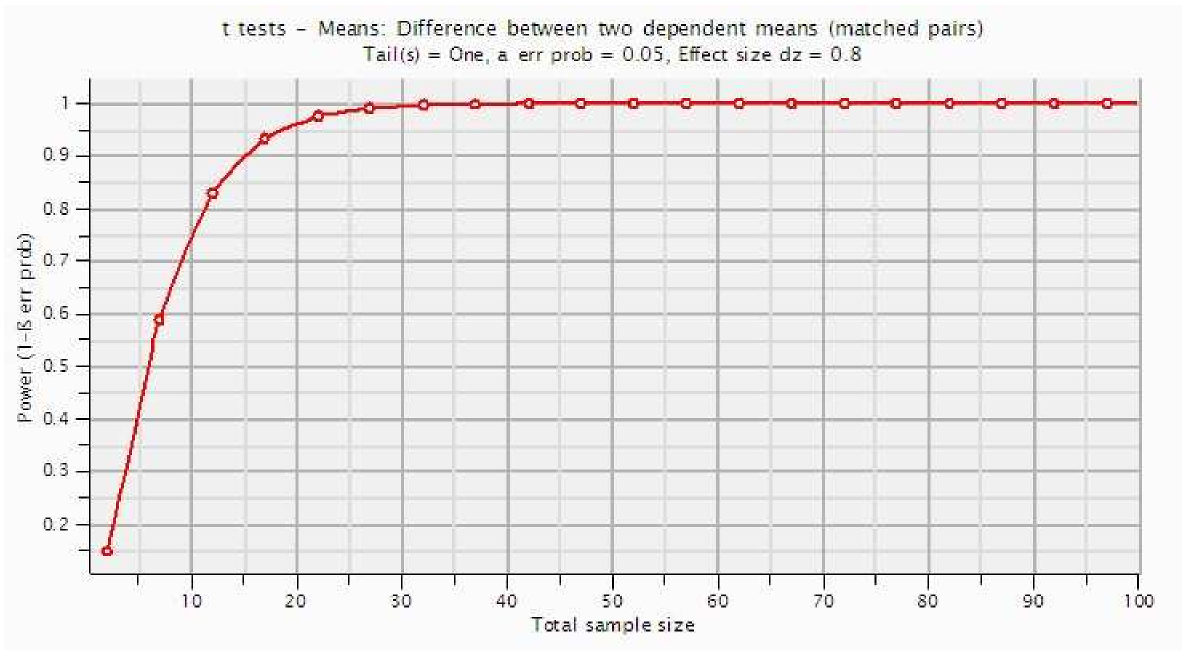


Figure 5.6. Power for sample sizes for matched pairs (i.e. in the same trial)

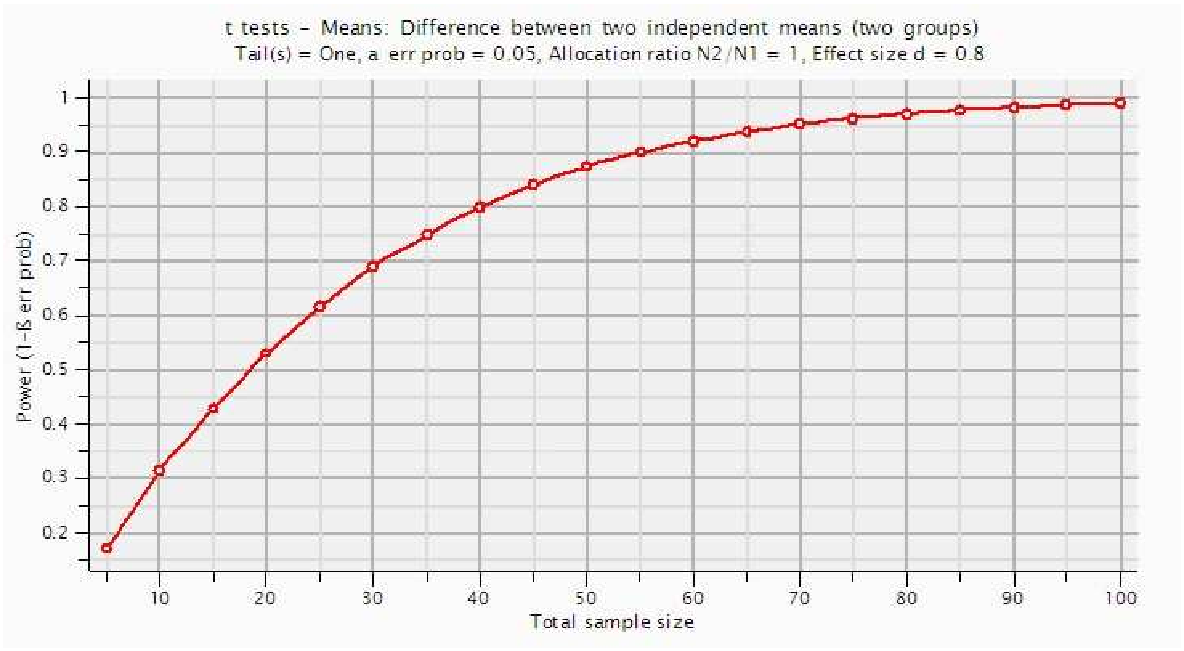


Figure 5.7. Power for sample sizes for independent $t - tests$ (i.e. across trials)

Chapter 6

RESULTS

This chapter presents the results from the second round of experiments, described in detail in Chapter 5. The experiments took an average of 30 minutes for each participant. This time includes the time to go through the verbal description and interview; the time for the participant to fill out the pre- and post-questionnaires; the training phase(s)(if any); the regular trials for both device types; and the 30 second calibration for the head-tilt mouse. The time for the trials themselves ran about 20 minutes, with the other parts taking around 10 minutes, including the training phase(s). The shortest time to complete an experiment was 16 minutes, while the longest was 47 minutes. On average, the standard mouse trials took 1–2 minutes, while the head-tilt mouse trials lasted 4–6 minutes.

Recall from Chapter 5 that the subjects of the experiments were randomly divided into four groups which determined whether or not they received a training phase and an intervention (i.e. the Virtual Dynamic Tunnel algorithm was applied). Each subject was placed into a group for the standard (USB) mouse and a group for the head-tilt mouse. The group numbering scheme is shown in Table 6.1. Note that the control group for the experiments is Group 4 (i.e. this group has no algorithm applied and the subjects receive no training). As an example, suppose a subject participates in the experiment. She may be randomly placed in Group 3 for the standard (USB) mouse and Group 1 for the head-tilt mouse. This would mean that for the standard (USB) mouse she would not get training, but the dynamic tunnel algorithm would be applied (i.e. there is an intervention) for the standard (USB) mouse trials. Since she is assigned to Group 1 for the head-tilt mouse, she will get a training phase and the dynamic tunnel algorithm would be applied (i.e. there is an intervention).

Table 6.1. Group number and type.

Group no.	Group description
Group 1	Has training and intervention.
Group 2	Has training, but not intervention.
Group 3	No training, but intervention.
Group 4	No training and no intervention (i.e. control group).

6.1 Performance summaries

In Figures 6.1, 6.2, 6.3 and 6.4, a summary of the experimental results is shown graphically. In these figures, the mean time to complete a trial and the mean number of errors incurred during the trial are shown as bars on the graph for each of the groups mentioned in Table 6.1. The mean time is shown as the left-hand bar for each group and the mean number of errors is represented by the right-hand bar. The mean time is in seconds. Each figure shows the means for a device type and a trial type (e.g. standard (USB) mouse and a straight tunnel trial in Figure 6.1). After the summary graphs, tables are given comparing different groups using *t-test* results and Bonferroni *p-values* to establish confidence levels. From these *t-test* and *p-value* results, conclusions are made about the significance of the differences between groups and the effectiveness of the Virtual Dynamic Tunnel algorithm is determined.

In Figure 6.1, it can be seen that the difference between mean trial time between Group 1 and Group 2 is close to zero. This implies that for the standard (USB) mouse with a straight tunnel and training, the dynamic tunnel algorithm had little or no impact over training. Comparing Groups 1 and 3, however, show that for standard (USB)/straight, training with the algorithm was better than the algorithm alone; the difference between Groups 2 and 4 also support the effect of training on time. Lastly, comparing Groups 3 and 4 indicates that the algorithm actually impedes performance for standard (USB)/straight when used without training. The best performance in time and number of errors is when the algorithm is used with training for the standard (USB)/straight trials.

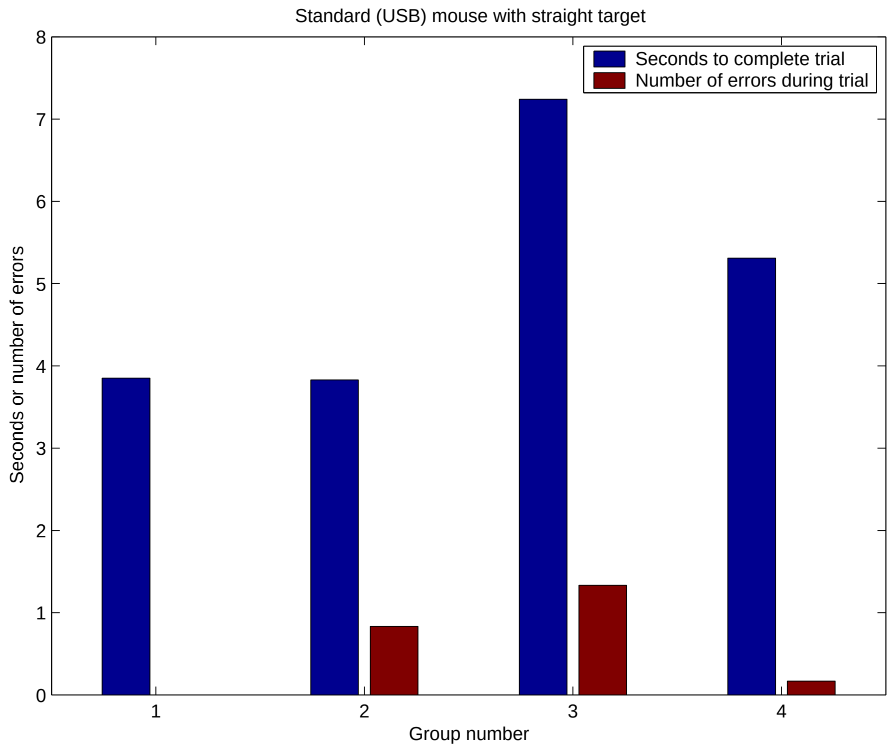


Figure 6.1. Standard (USB) mouse and straight trial.

Figure 6.2 shows the mean results for the standard (USB) mouse with a circular trial. In this case, the difference between Groups 1 and 2 indicate that in groups given training, the algorithm makes performance worse except with respect to the number of errors, which are lower in the intervention group. This can be explained by noting that since a circular tunnel requires more fine motor skills, a trade-off between the number of errors and the completion time becomes more apparent (i.e. as a user tries to minimize errors, she/he may have to slow down and thereby increasing completion time) than in straight tunnel trials. However, in the non-trained groups (i.e. 3 and 4), both mean trial time and number of errors are worse in Group 3, which has the algorithm applied. As far as training is concerned, for standard (USB)/circular trials, there is no significant advantage to training. This can be explained by noting that typical user movement is not a complete circle, but in a semi-circular fashion and that the training phase, consisting of only one extra trial may not be sufficient for the user to get accustomed to this new way of moving the mouse cursor.

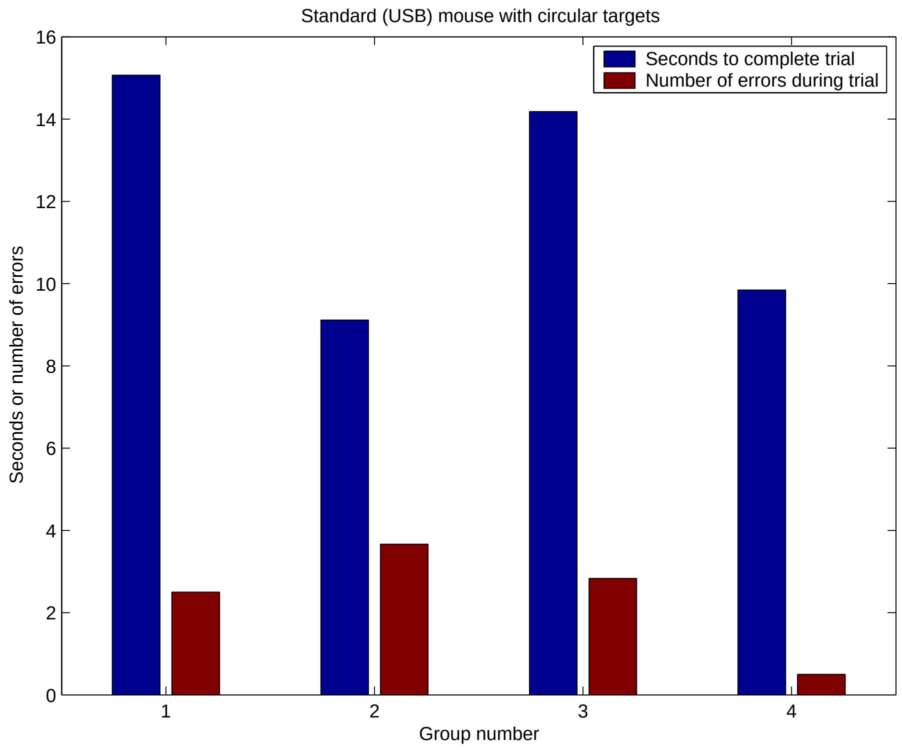


Figure 6.2. Standard (USB) mouse and circular trial.

The mean results for the head-tilt mouse using a straight trial are show in Figure 6.3. In this figure, it can be seen that the mean trial time for Group 1 is less than Group 2 and also less than Group 4. This indicates that training with the algorithm for the head-tilt mouse and straight trials improves completion time performance over the control group. The error counts across all groups remained flat in this case.

The last summary bar graph is shown in Figure 6.4 and represents the results for the head-tilt mouse with a circular trial. In this figure, Group 1 and Group 4 have similar performance for both mean trial time and mean number of errors. This suggests that for circular trials with the head-tilt mouse, the algorithm with training performs no better than the control group. Also, Group 3's low performance indicates that the algorithm impedes performance if training is not used in this case.

The preceding graphs show the results summary for the experiments with some conjecture about the meaning of those results. In the following section we further analyze

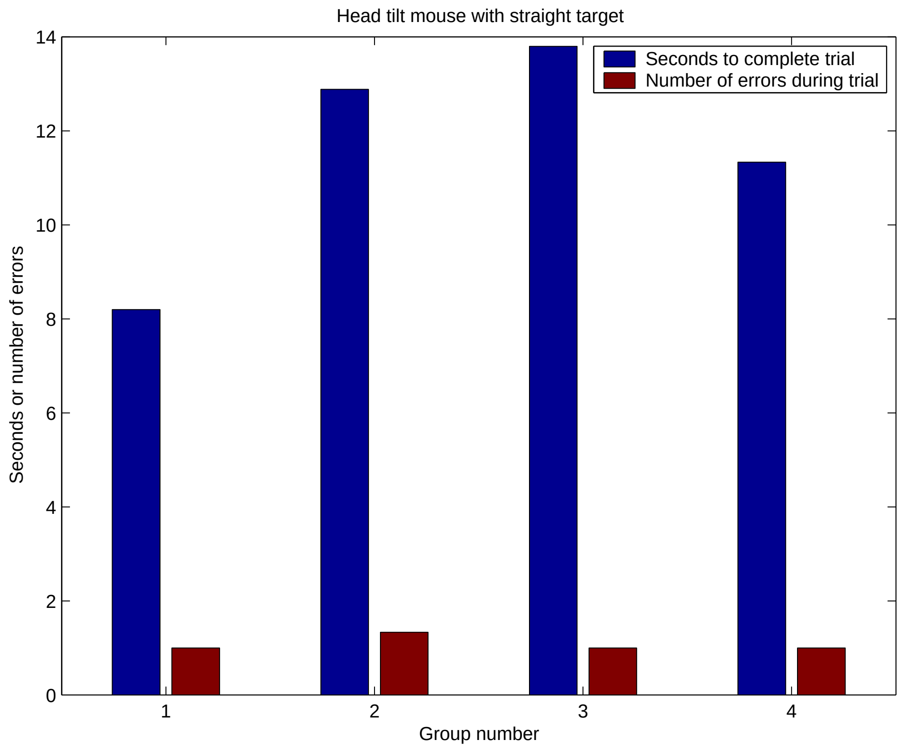


Figure 6.3. Head-tilt mouse and straight trial.

the results using *t-tests* betweens groups and establish confidence levels using Bonferroni adjustment for our experimental sample sizes.

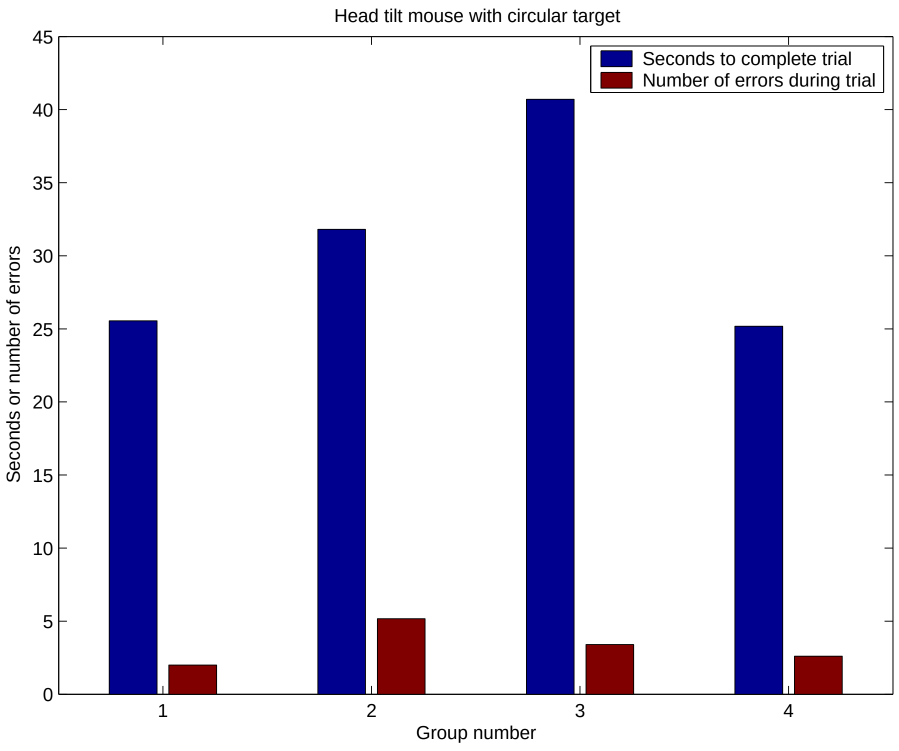


Figure 6.4. Head-tilt mouse and circular trial.

Table 6.2. T-tests of trial time comparing Group 3 to Group 4.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	1.18	0.292	Figure 6.1
standard mouse	circle	1.98	0.104	Figure 6.2
head mouse	straight	0.48	0.653	Figure 6.3
head mouse	circle	2.31	0.069	Figure 6.4

6.2 T-tests

In Tables 6.2-6.7, the *t-tests* comparing the mean trial times for all group pairs are given. In Tables 6.8-6.13, the *t-tests* comparing the mean number of errors for all group pairs are given. In both of these sets of tables, a *t-test* value less than -1.65 indicates that the mean of the first group is significantly less than the mean of the second group, as explained in Chapter 5. Likewise, a *t-test* value of 1.65 or greater indicates that the mean of the first group is significantly greater than the mean of the second group. Further, in each table a Bonferroni *p-value* is given to establish the likelihood that a result occurred by chance given the sample size for our groups. This also gives a confidence level in the corresponding *t-test* result.

Table 6.2 shows the *t-tests* for the mean trial times comparing Groups 3 and 4. Recall that Group 3 has no training but has the intervention and that Group 4 is the control group (i.e. has no training and no intervention). From the *t-tests* in this table, it can be seen that Group 3's trial time performance is significantly lower (indicated by the positive *t-test*) than Group 4's in the standard (USB)/circle case as well as the head-tilt/circle case. This implies that the algorithm without training applied has poorer performance in reference to trial time on circular trials for both device types. The *p-values* indicate that given the sample size, there is an approximately 10% and 7% chance, respectively, of this occurring by chance.

In Table 6.3, the *t-tests* comparing Group 1 (i.e. trained while using the algorithm) and Group 2 (i.e. trained without using the algorithm) are shown. There is no significant difference between the two groups. This implies that the algorithm does not improve or impede trial time performance over training for any trial or device type.

Table 6.3. T-tests of trial time comparing Group 1 to Group 2.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	-0.51	0.63	Figure 6.1
standard mouse	circle	1.55	0.18	Figure 6.2
head mouse	straight	-1.24	0.27	Figure 6.3
head mouse	circle	-0.68	0.53	Figure 6.4

Table 6.4. T-tests of trial time comparing Group 1 to Group 3.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	-2.03	0.10	Figure 6.1
standard mouse	circle	-0.31	0.77	Figure 6.2
head mouse	straight	-2.81	0.04	Figure 6.3
head mouse	circle	-2.23	0.08	Figure 6.4

T-tests comparing Group 1 to Group 3 are shown in Table 6.4. There is significant difference between these two groups for the standard (USB)/straight, head-tilt/straight and head-tilt/circular cases. This implies that training significantly improves mean trial time performance for the algorithm in all cases but the standard (USB)/circle case. Even for our sample size, Bonferroni *p-values* support this with a 96% confidence level.

In Table 6.5, comparing Groups 2 and 4, no significant differences were found. This, surprisingly, implies that training, without the use of the algorithm, does not improve (nor impede) mean trial time performance significantly in any combination of device type or trial type.

Table 6.5. T-tests of trial time comparing Group 2 to Group 4.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	-1.36	0.23	Figure 6.1
standard mouse	circle	-0.46	0.66	Figure 6.2
head mouse	straight	0.59	0.58	Figure 6.3
head mouse	circle	0.69	0.52	Figure 6.4

Table 6.6. T-tests of trial time comparing Group 1 to Group 4.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	-1.76	0.14	Figure 6.1
standard mouse	circle	1.33	0.24	Figure 6.2
head mouse	straight	-1.39	0.22	Figure 6.3
head mouse	circle	0.28	0.79	Figure 6.4

Table 6.7. T-tests of trial time comparing Group 2 to Group 3.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	-1.98	0.11	Figure 6.1
standard mouse	circle	-2.51	0.05	Figure 6.2
head mouse	straight	-0.12	0.91	Figure 6.3
head mouse	circle	-0.50	0.64	Figure 6.4

Groups 1 and 4 are compared in Table 6.6. The only significant difference found was with the standard (USB)/straight case. This implies that the algorithm used with training only improves performance in trial time for the standard (USB) mouse when used in a straight tunnel. For this result, the table indicates an 86% confidence level.

In Table 6.7, Groups 2 and 3 are compared with significant differences found in the standard (USB)/straight and standard (USB)/circular cases. This implies that training with the algorithm is much better in reference to trial time performance than use of the algorithm alone when using the standard (USB) mouse on either trial type.

Table 6.8. T-tests of number of errors comparing Group 3 to Group 4.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	1.47	0.20	Figure 6.1
standard mouse	circle	1.90	0.11	Figure 6.2
head mouse	straight	0.00	1.00	Figure 6.3
head mouse	circle	0.67	0.53	Figure 6.4

Table 6.9. T-tests of number of errors comparing Group 1 to Group 2.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	-1.27	0.26	Figure 6.1
standard mouse	circle	-0.27	0.80	Figure 6.2
head mouse	straight	-1.00	0.36	Figure 6.3
head mouse	circle	-2.06	0.09	Figure 6.4

Tables 6.8-6.13 show the *t-tests* comparing all groups for mean number of errors in the experimental trials. In Table 6.8, Groups 3 and 4 are compared with the only significant difference indicated in the standard (USB)/circle case. This implies that the algorithm without training is significantly worse in reference to number of errors in the standard (USB)/circular case.

Groups 1 and 2 are compared in Table 6.9. A significant difference was found in the case of the head-tilt/circular case. This implies that training with the algorithm significantly out-performs training alone in the head-tilt/circular case. A 91% confidence level is given for this result.

No additional significant differences where found in reference to mean number of errors for the rest of the group comparisons as shown in Tables 6.10-6.13. This implies that neither the presence/absence of the algorithm nor the presence/absence of training had any effect on the performance in the mean number of errors for these combinations of device type and trial type.

Table 6.10. T-tests of number of errors comparing Group 1 to Group 3.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	-1.40	0.22	Figure 6.1
standard mouse	circle	0.00	1.00	Figure 6.2
head mouse	straight	no errors	N/A	Figure 6.3
head mouse	circle	-1.46	0.20	Figure 6.4

Table 6.11. T-tests of number of errors comparing Group 2 to Group 4.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	1.34	0.24	Figure 6.1
standard mouse	circle	1.25	0.26	Figure 6.2
head mouse	straight	0.79	0.47	Figure 6.3
head mouse	circle	1.25	0.27	Figure 6.4

Table 6.12. T-tests of number of errors comparing Group 1 to Group 4.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	-1.00	0.36	Figure 6.1
standard mouse	circle	1.38	0.23	Figure 6.2
head mouse	straight	0.00	1.00	Figure 6.3
head mouse	circle	-0.29	0.78	Figure 6.4

Table 6.13. T-tests of number of errors comparing Group 2 to Group 3.

Device	Trial type	t-test	Bonferroni p-value	Reference figure
standard mouse	straight	-1.46	0.20	Figure 6.1
standard mouse	circle	0.28	0.79	Figure 6.2
head mouse	straight	1.00	0.36	Figure 6.3
head mouse	circle	0.99	0.37	Figure 6.4

Table 6.14. T-tests for training using Group 1.

Device	Trial type	Trial time t-test	Bonf. p-value	Errors t-test	Bonf. p-value
standard mouse	straight	1.94	0.11	1.55	0.19
standard mouse	circle	-0.25	0.81	0.12	0.91
head mouse	straight	0.96	0.38	no errors	N/A
head mouse	circle	3.10	0.03	1.96	0.12

Table 6.15. T-tests for training using Group 2.

Device	Trial type	Trial time t-test	Bonf. p-value	Errors t-test	Bonf. p-value
standard mouse	straight	0.23	0.83	0.89	0.41
standard mouse	circle	0.05	0.96	-0.65	0.54
head mouse	straight	-0.18	0.86	1.00	0.36
head mouse	circle	-0.41	0.70	-0.46	0.67

6.3 Training effect

In Tables 6.14 and 6.15, the effect of training on performance (both trial time and number of errors) is examined. In Table 6.14 the training phase is compared to the experimental phase for mean trial time and mean number of errors. Significant difference was found in the standard (USB)/straight and head-tilt/circle cases for trial time. Significant difference was also found for mean number of errors for the head-tilt/circular case. This implies that users got much better using the algorithm once they were trained in the standard (USB)/straight case and for both metrics in the head-tilt/circular case.

Table 6.15 shows a comparison between training phase and experimental phase without using the algorithm. In all cases, no significant improvement was found. Since the training phase was limited to only a single extra trial for each trial type (i.e. circular and straight), it is likely that this is the reason why training was found to have little effect during these experiments.

6.4 Age versus errors

In this section results comparing the average number of errors made by participants of given ages during their initial trial (training or non-training, whichever was their initial trial) for each device and trial-type combination are given. The results are shown in Figures 6.5-6.8. Looking at the figures, it can be seen that there were more younger participants than older ones. Looking at Figure 6.5, there appears to be a spike in the number of errors for

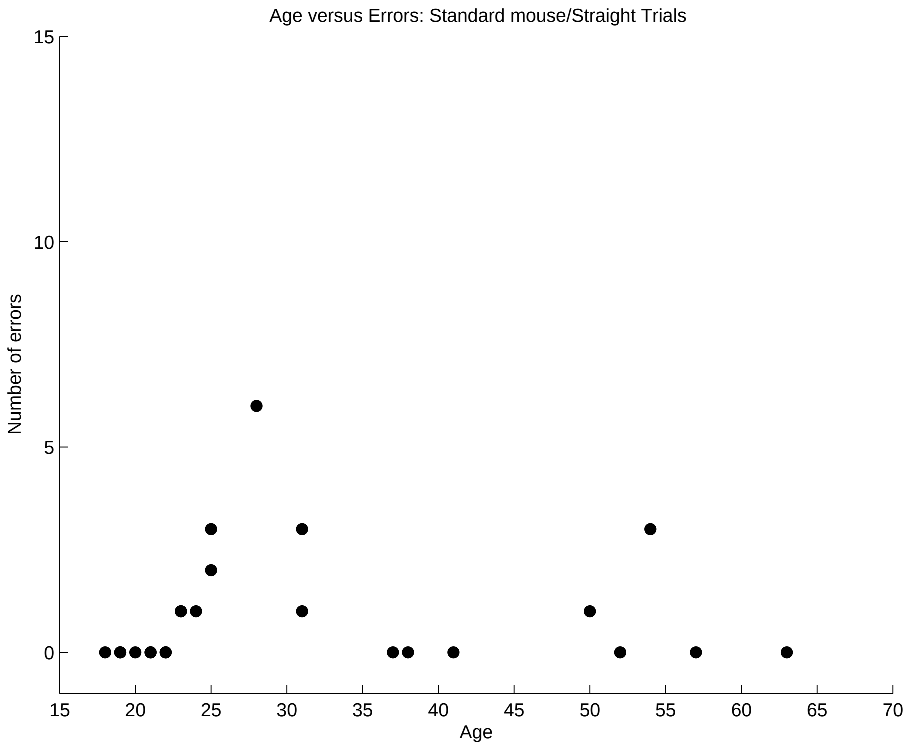


Figure 6.5. Age vs. errors – standard (USB) mouse and straight trial.

standard (USB) mouse/straight trials in the age range of 23–38, compared to the previous range of 18–22. The the spike then drops down again for ages 31–37. In Figure 6.6, there seems to be a rise in the number of errors for age of participant. This could be worth further investigation and testing for significance.

However, if we assumed an overall correlation between age and an increase in the average number of errors across all device types and trials, that relationship did not manifest itself in these data for these experiments.

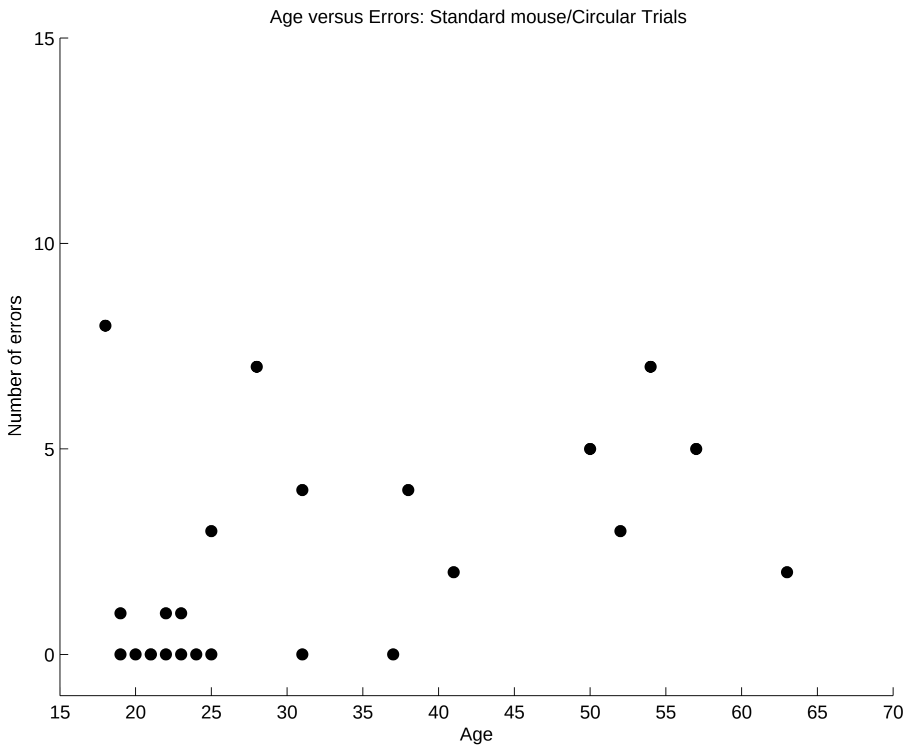


Figure 6.6. Age vs. errors – standard (USB) mouse and circular trial.

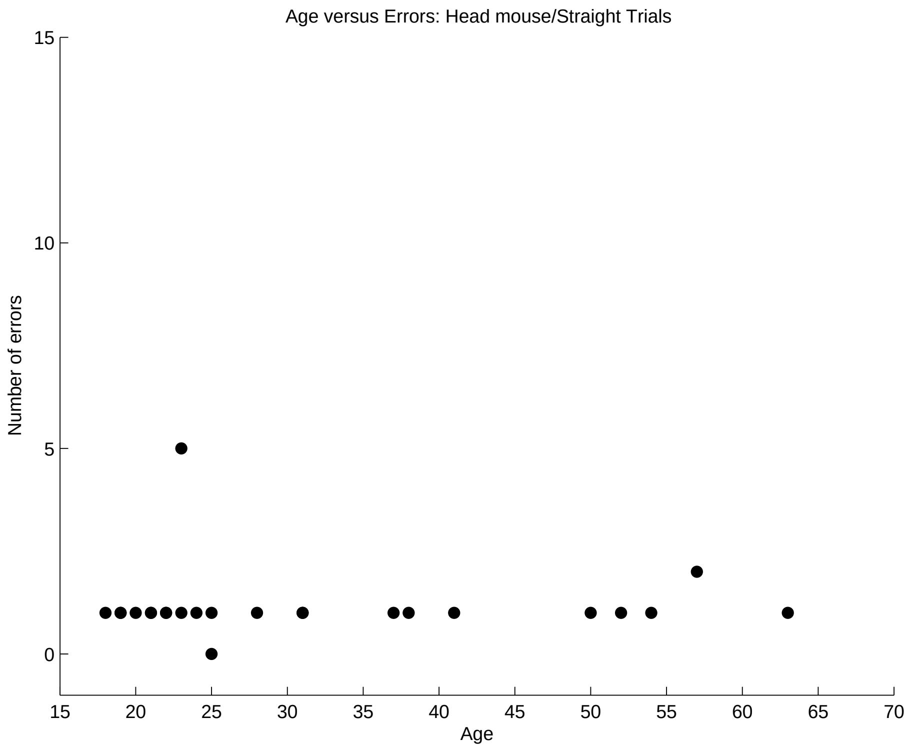


Figure 6.7. Age vs. errors – head-tilt mouse and straight trial.

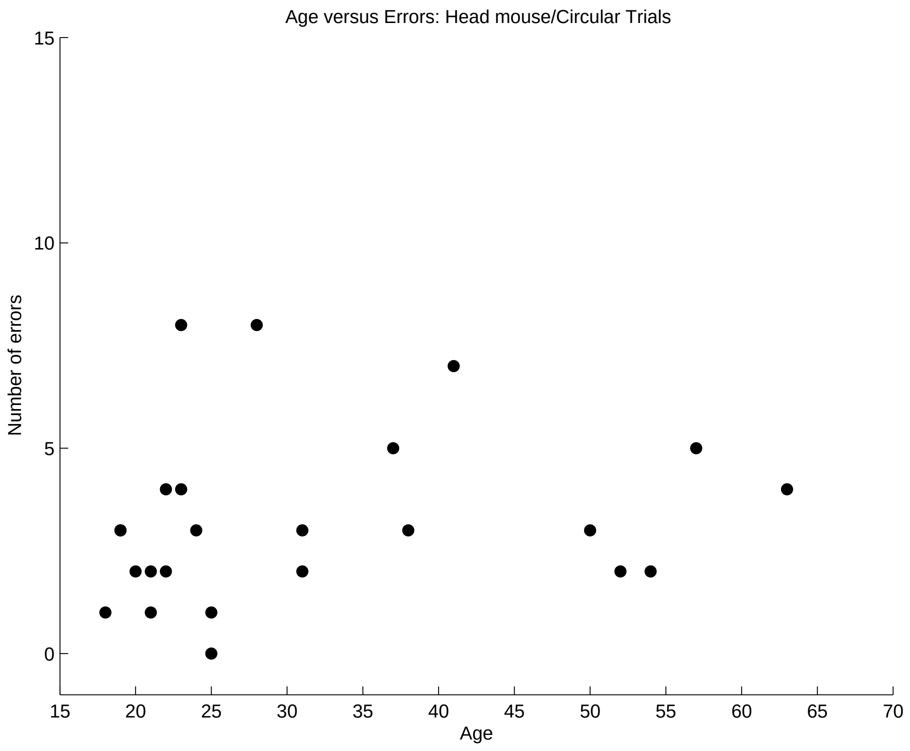


Figure 6.8. Age vs. errors – head-tilt mouse and circular trial.

Table 6.16. Questionnaire forms summary data.

standard (USB) mouse groups						
Group	Avg. age	Min. age	Max. age	Male	Female	Would use head mouse
1	31	18	52	2	4	5
2	35	19	54	3	3	3
3	33	22	57	2	4	2
4	28	19	63	3	3	4
Head mouse groups						
1	30	21	54	2	4	4
2	32	19	52	2	4	4
3	37	21	63	3	3	4
4	30	18	57	3	3	2
	32	18	63	10	14	14

6.5 Questionnaire data

In Tables 6.16 and 6.17 data collected from the participants on the questionnaire forms are summarized. Table 6.16 shows break-downs of age, sex, and responses to the question,“Would you use the head-tilt mouse?” for each group assignment. The last row of the table contains the overall statistics which indicate that the average age of the subjects was 32 with a minimum age of 18 and a maximum age of 63. Ten subjects were male, 14 female and 14 out of 24 said they would use the head-tilt mouse.

Table 6.17 shows the means and standard deviations of responses to some of the questions on the form (i.e. those not related to a user’s ability to participate in the experiment but used for collecting data about the user and their experience of participation in the experiment). Questions 1-5, and 8 (missing from the table) are used to determine whether or not a potential participant is capable of taking part in the experiment. Therefore, these questions were omitted from the table. The range of values for questions 6, 7, 10, 11, 12, 13 and 14 was 1–5.

Table 6.17. Questionnaire responses statistics.

No.	Question	Mean	Std. dev.
6	How skilled w/stand. mouse?	4.6	0.7
7	Phys. coord.?	4.5	0.7
9	Num. yrs. using mouse?	12.9	5.5
10	How hard head-tilt?	3.3	0.8
11	How hard std. mouse?	4.5	0.5
12	How comfort. head-tilt?	3.6	0.9
13	How comfort. stand. mouse?	4.4	0.7
14	How skilled could you become with head-tilt?	3.7	0.6

6.6 Results summary

From the previous results, the following assertions from the data can be made:

- The Virtual Dynamic Tunnel algorithm essentially needs training to be used effectively in all cases except using the standard (USB) mouse during straight tunnel trials.
- Surprisingly, training, without the use of Virtual Dynamic Tunnel algorithm, seems to have no effect on performance in any of the test cases. This result likely indicates that the training phase, which consisted of only one extra trial for each tunnel type per device, should be more extensive.
- The Virtual Dynamic Tunnel algorithm was most effective in improving performance when the subject was trained and using the standard (USB) mouse with a straight tunnel trial. The first point seems to contradict this. However, it is supported by the second point. The *t-test* result was 1.76 with a Bonferroni *p-value* of 0.14 or an 86% confidence level.
- For the standard (USB) mouse circular trials the VDT algorithm made performance worse in the respect to both errors and time. This could be due the constantly changing course of the mouse pointer. It is worth noting that typical mouse movement would be in a semi-circular fashion and that a trial involving, say, four quarter-circles may be more appropriate than one full circle. At any rate, it shows that the algorithm would need to be modified for full circular tasks on the standard mouse.
- Users significantly improved using the Virtual Dynamic Tunnel algorithm during the training for the standard (USB)/straight (Table 6.14, row 1) and head-mouse/circle (Table 6.14, row 4) trials.
- Nearly 60% of users in the experiment said they would consider using the head-tilt mouse as an alternative input device.

In this chapter we presented details of the results from our VDT experiment. In the next chapter, we discuss the conclusions of this work.

Chapter 7

CONCLUSIONS

In this chapter we present the conclusions of this research starting with a summary and followed by a list of contributions.

7.1 Summary

This work introduces a novel target-agnostic assistive interface algorithm for mouse cursor control that is designed to improve usage performance for head-operated mouse devices and standard mouse devices alike. The algorithm was designed, implemented and evaluated for both device classes. For the standard mouse, the algorithm was implemented in the form of a filter driver installed at the operating system kernel level. For the head-operated mouse, the algorithm was integrated into the signal processing software already running on the host machine. The algorithm creates a target-agnostic virtual dynamic tunnel by examining recent mouse cursor movement and, based on that, predicts future movement. If the user tries to move the cursor away from the predicted path (i.e. tunnel), the cursor movement is nudged back to the tunnel center by a gravity-like force. In addition to the Virtual Dynamic Tunnel (VDT) algorithm a new wireless head-operated mouse device was designed, implemented and tested for this work. The wireless head-tilt mouse provides users with a hands-free alternative to computer mouse control at a low cost (<\$200US) without the need for specialized hardware. The system operates by having the user wear a cap embedded with the head-tilt mouse device. Head motion is sensed and head movement data are transmitted over the Bluetooth wireless protocol to the user's computer. There, the signals are mapped to mouse cursor movement and depending on configuration, the VDT algorithm is applied. The effectiveness of the VDT algorithm and the head-tilt mouse was evaluated in two rounds of experiments. In the first round, performance of the head-tilt mouse (without the algorithm) was compared to a regular wireless mouse and a touchpad-type mouse. From these experiments the feasibility of using the head-tilt mouse was established. The second round of experiments tested the performance of the virtual dynamic tunnel algorithm implemented on a standard mouse and the head-tilt mouse using a Solomon four-group experiment technique with six users for each of the four groups.

7.2 Contributions

This work helped contribute to the field of human-computer interaction through two research publications described here as well as a provisional patent on the original head-tilt mouse device, described below.

7.2.1 Publications

Workshop paper: "Wireless Tilt Mouse: Providing Mouse-type Access for Computer Users with Spinal Cord Injuries or Disabilities," *PETRA08, Athens, Greece, ACM Int. Conf. Proceed.*, July 2008.

This paper introduced the wireless head-tilt mouse as a plausible alternative to traditional input devices for users with disabilities. It also provided a description of a test application developed to capture target acquisition time for the device or other mouse control devices. Finally, the results of an experiment comparing the head-tilt mouse to a touchpad and a standard Bluetooth mouse were presented. This paper showed that the wireless head-tilt mouse could be a plausible option for users with disabilities or an alternative input method for able-bodied users.

Conference paper: "Target Acquisition by a Hands-free Wireless Tilt Mouse," *IEEE Conf. on Sys., Man and Cyber.*, Oct. 2009.

This paper provided more detailed analysis of the target acquisition performance of the head-tilt mouse compared to the touchpad and Bluetooth mouse devices. It also set the stage for the development of the VDT algorithm by showing the effect increasing the target size would have had on the hit rate in the initial experiment.

7.2.2 Patent

Provisional patent: *U.S. Patent Application No. 61/221,772*, July 30, 2009

This was a provisional patent on the original head-tilt mouse device.

7.2.3 Head-tilt mouse

A new head-operated mouse cursor device was designed, built and evaluated. This device is wireless, small, inexpensive and requires no special host computer hardware. In experiments, able-bodied users say that they are willing to a use head-operated mouse input device at a rate of nearly 60% as an alternative to the standard mouse and users find it only moderately hard to use.

7.2.4 Virtual Dynamic Tunnel (VDT) algorithm

This work advances knowledge in the field of assistive user interfaces by showing, through experiments, that training alone (without assistive techniques) has nearly no effect on performance in straight tunnel tasks for either head-operated devices or standard mouse devices. The work also provides an opportunity for future investigation for the possibility that the VDT algorithm improves performance on straight tunnel tasks for the standard mouse. Also, with the filter driver design, this research provides a framework where cursor movement history can be analyzed and subsequent cursor movement updated in a nearly-pluggable fashion (controlled, for example, by driver configuration settings).

In this chapter we presented the conclusions of this work and listed the contributions. In the next chapter, we discuss possible future work.

Chapter 8

FUTURE WORK

In this chapter we list possible future work. Some of the limitations of this work that could be explored in future work are:

- Implementing mouse click functionality for the head-tilt mouse.
- Carrying out additional experiments to determine the effect of changing parameters to the Virtual Dynamic Tunnel algorithm—like queue size, and coefficient values.
- Carrying out additional experiments to determine the effect of tunnel widths on the performance of the Virtual Dynamic Tunnel algorithm and on training results performance. We know from the literature that tunnel widths affect performance, but how does it affect training performance and especially considering the Virtual Dynamic Tunnel algorithm?
- Determining the long-term effect of training with the mouse devices. Would longer training sessions improve performance over the short ones carried out in these experiments? How long would any training effects last over time away from an input device?
- Modification of the algorithm to capitalize on its strength (i.e. straight tunnels) by disabling it when a curve-type movement is detected and enhancing the effect when straight movement is detected. How would this compare to the dynamic gain algorithm mentioned in the literature?
- Determining the effect of multiple targets (e.g. like a cluttered desktop) on the Virtual Dynamic Tunnel algorithm.
- Because all of the experiments involved able-bodied participants, it would be good to expand this work to include disabled users as the head-tilt mouse (in conjunction with

the VDT algorithm) could be an advantage to disabled users taking into account the drawbacks of other devices shown in Table 2.1 and referenced in [2, 24–29, 32].

- Exploring the effect of changing the VDT algorithm where we feed the newly projected points for use as the extrapolation parameters. For example, exploring the effect of using the green line in Figure 4.4 versus using the black line in that figure as we currently do.

REFERENCES

[1] H. Russell Bernard. *Social Research Methods: Qualitative and Quantitative Approaches*. Sage Publications, Inc., 2000.

[2] Adriane B. Randolph. *Individual-Technology Fit: Matching Individual Characteristics and Features of Biometric Interface Technologies with Performance*. Doctoral dissertation, Georgia State University, 2007.

[3] Berlin Brain Computer Interface. Brain computer interface image. *http://www.eurescom.eu/message/messageDec2004/The_Berlin_Brain_Computer% _Interface.asp*, 2010.

[4] Technology for Education. Sip/puff switch. *http://www.tfeinc.com/shop/index.php? _a=viewProd &productId=1751*, 2010.

[5] eSchool News. Tongue switch. *http://www.eschoolnews.com/2008/07/11/emerging-tech-makes-learning-more-accessible/*, 2010.

[6] Emotiv Systems. Emotiv EPOC Neuroheadset. *http://emotiv.com/corporate/2_0/2_1.htm*, January 2009.

[7] Grupo de Robotica. HeadMouse 2. *http://robotica.udl.es/ headmouse/headmouse2 /headmouse2e.html*, accessed February, 2009.

[8] Adam J. Sporka. *Non-speech Sounds for User Interface Control*. Doctoral dissertation, Electrical Engineering, Czech Technical University in Prague, 2008.

[9] Washington Apple Pi. Twiddler2. *http://www.wap.org/journal/twiddler/default.html*, 2010.

[10] Jupiterimages Corporation. Clipart.com. *http://www.clipart.com*, 2008.

[11] Inc. Parallax. Corporate website. *http://www.parallax.com*, 2010.

[12] Marcia J. Scherer. *Assistive Technology: Matching Device and Consumer for Successful Rehabilitation*, pages 33–35,77,81. American Psychological Association, 2002.

[13] National Institute on Disability and Rehabilitation Research of the U.S. Dept. of Education. ABLEDATA Classification System website. *http://www.abledata.com*, 2006.

[14] Albert M. Cook and Susan M. Hussey. *Assistive Technologies: Principles and Practice.* Mosby, Inc., 2002.

[15] American Occupational Therapy Association. *Uniform Terminology for Occupational Therapy : Uniform Terminology, third edition: Application to Practice.* American Occupational Therapy Association, 1994.

[16] Marcia J. Scherer and Gerald Craddock. Matching Person & Technology (MPT) Assessment Process. *Technology and Disability*, 14:125–131, 2002.

[17] I. S. MacKenzie, A. Sellen, and W. Buxton. A Comparison of Input Devices in Elemental Pointing and Dragging Tasks. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 161–166, 1991.

[18] I. S. MacKenzie. Fitts' Law as a Research and Design Tool in Human-Computer Interaction. *Human-Computer Interaction*, 7:91–139, 1992.

[19] P.M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47:381–391, 1954.

[20] I. S. MacKenzie. A Note on the Information-Theoretic Basis for Fitts' Law. *Journal of Motor Behavior*, 21:323–330, 1989.

[21] I. S. MacKenzie and William Buxton. Extending Fitts' Law to Two-Dimensional Tasks. *Proceedings of the CHI'92 Conference on Human Factors in Computing Systems*, pages 219–226, 1992.

[22] Johnny Accot and Shumin Zhai. Scale effects in steering law tasks: Do device size and different motor joints matter? *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, 2001.

[23] Johnny Accot and Shumin Zhai. Performance evaluation of input devices in trajectory-based tasks: an application of the steering law. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 466–472, New York, NY, USA, 1999. ACM.

[24] Julien Kronegg, Svyatoslav Voloshynovskiy, and Thierry Pun. Brain-computer interface model: Upper-capacity bound, signal-to-noise ratio estimation, and optimal number of symbols. *Tech. Rep. 04.03*, 2004.

[25] L. Citi, R. Poli, C. Cinel, and F. Sepulveda. P300-based BCI Mouse With Genetically-Optimized Analogue Control. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 16(1):51–61, February 2008.

[26] T. Surdilovic. *Fuzzy Mouse Cursor Control System For Computer Users with Spinal Cord Injuries.* Masters thesis, Georgia State University, 2005.

[27] Anton Nijholt, Boris Reuderink, and Danny Oude Bos. Turning shortcomings into challenges: Brain-computer interfaces for games. *ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 153–168, 2009.

[28] Lotte N. S. Andreasen Struijk. An inductive tongue computer interface for control of computers and assistive devices. *IEEE Transactions on Biomedical Engineering*, 53(12), December 2006.

[29] Xueliang Huo, Jia Wang, and Maysam Ghovanloo. Introduction and preliminary evaluation of the tongue drive system: Wireless tongue-operated assistive technology for people with little or no upper-limb function. *Journal of Rehabilitation Research and Development*, 45(6):921–930, 2008.

[30] Richard Seven. UW Creates a Computer Mouse Driven by Sound. *http://seattletimes.nwsource.com*, October 2008.

[31] Takeo Igarashi and John F. Hughes. Voice as Sound: Using Non-verbal Voice Input for Interactive Control. *Proceedings of 14th annual ACM symposium on User Interface Software and Technology*, pages 155–156, 2001.

[32] Adam J. Sporka, Sri H. Kurniawan, and Pavel Slavik. Acoustic Control of Mouse Pointer. *Information Society*, 4(3):125–131, 2006.

[33] Anuradha Menon. Special GUI for Your Eyes Only. *http://thefutureofthings.com/news/5829/ special-gui-for-your-eyes-only.html*, November 2008.

[34] J. E. Zucco, B. H. Thomas, and K. Grimmer. Evaluation of Three Wearable Computer Pointing Devices for Selection Tasks. *Proceedings of the 2005 ninth IEEE International Symposium on Wearable Computers (ISWC-05)*, pages 178–185, October 2005.

[35] Ferrol R. Blackmon and Michael Weeks. Wireless tilt mouse: Providing Mouse-type Access for Computer Users with Spinal Cord Injuries or Disabilities. In *1st International Conference on Pervasive Technologies Related to Assistive Environments (PETRA08)*, International Workshop on Ambient Assistive Technologies for Intelligent Healthcare Services (AASTIH08), Athens, Greece. ACM International Conference Proceedings Series, July 2008.

[36] Ferrol R. Blackmon and Michael Weeks. Target Acquisition by a Hands-free Wireless Tilt Mouse. In *IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC 2009)*. Proceedings 2009 IEEE International Conference on Systems, Man and Cybernetics, October 2009.

[37] Ferrol R. Blackmon and Michael Weeks. Wireless Cursor Control. *U.S. Patent Application No. 61/221,772*, 2009.

[38] Bruce Hopkins and Ranjith Antony. *Bluetooth for Java*. Apress, 2003.

[39] C Bala Kumar, Paul J. Kline, and Timothy J. Thompson. *Bluetooth Application Programming with Java APIs.* Morgan Kaufmann Publishers, 2004.

[40] Parallax Inc. *Memsic 2125 Accelerometer Demo Kit (#28017)*, 2004.

[41] Inc. Parallax. *BASIC Stamp Syntax and Reference Manual*, version 2.2 edition, 2005.

[42] A7 Engineering. *EmbeddedBlue 500 User Manual*, revision E edition, April 2005.

[43] D-Link Corporation. DBT-120 wireless Bluetooth 2.0 USB adapter. *http://www.dlink.com/products/?pid=34*, accessed February, 2008.

[44] Inc. Parallax. *Hitachi H48C 3-Axis Accelerometer Module (#28026)*, version 1.2 edition, 2007.

[45] BluePacket Communications. *BP20422 Product Brief*, revision E edition, August 2008.

[46] WiiLi.org. WiiLi, a GNU/Linux port for the Nintendo Wii. *http://www.wiili.org/index.php/Main_Page*, January 2009.

[47] Patrick Langdon, Faustina Hwang, Simeon Keates, P John Clarkson, and Peter Robinson. Investigating haptic assistive interfaces for motion-impaired users: Force-channels and competitive attractive-basins. In *Proceedings of Eurohaptics 2002 (Edinburgh UK)*, pages 122–127, 2002.

[48] P.M. Langdon, F. Hwang, S. Keates, P.J. Clarkson, and P. Robinson. Developing assistive interfaces for motion-impaired users using cursor movement analysis in conjunction with haptic feedback. In *The 4th International Conference on Disability, Virtual Reality and Associated Technologies (ICDVRAT 2002)*, pages 18–20, 2002.

[49] Faustina Hwang, Simeon Keates, Patrick Langdon, P John Clarkson, and Peter Robinson. Perception and haptics: Towards more accessible computers for motion-impaired users. In *In Proceedings of the 2001 workshop on Percetive user interfaces*, pages 1–9. ACM Press, 2001.

[50] Simeon Keates, Faustina Hwang, Patrick Langdon, P. John Clarkson, and Peter Robinson. Cursor measures for motion-impaired computer users. In *Assets '02: Proceedings of the fifth international ACM conference on Assistive technologies*, pages 135–142, New York, NY, USA, 2002. ACM.

[51] Faustina Hwang, Simeon Keates, Patrick Langdon, and P John Clarkson. Multiple haptic targets for motion-impaired users. In *In Proceedings of the conference on Human factors in computing systems*, pages 41–48. ACM Press, 2003.

[52] F. Hwang, S. Keates, P. Langdon, and J. Clarkson. Gravity Well Visibility in Hatpic Interfaces for Motion-Imparied Users. In *Proceedings of the 25th Annual International Conference of the IEEE EMBS*, pages 1670–1673, 2003.

[53] Jacob O. Wobbrock, Jame Fogarty, Shih-Yen (Sean) Liu, Shunichi Kimuro, and Susumu Harada. The angle mouse: target-agnostic dynamic gain adjustment based on angular deviation. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 1401–1410, New York, NY, USA, 2009. ACM.

[54] TFOT. Special GUI for Your Eyes Only. *http://thefutureofthings.com/ news/5829/ special-gui-for-your-eyes-only.html*, 2007.

[55] Maruthappan Shanmugasundaram and Pourang Irani. The effect of animated transitions in zooming interfaces. In *AVI '08: Proceedings of the working conference on Advanced visual interfaces*, pages 396–399, New York, NY, USA, 2008. ACM.

[56] Richard Bates and Howell Istance. Zooming interfaces!: enhancing the performance of eye controlled pointing devices. In *Assets '02: Proceedings of the fifth international ACM conference on Assistive technologies*, pages 119–126, New York, NY, USA, 2002. ACM.

[57] Joe D. Hoffman. *Numerical Methods for Engineers and Scientists.* McGraw-Hill, Inc., 1992.

[58] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing, Third Edition.* Cambridge University Press, 2007.

[59] Richard L. Burden and J. Douglas Faires. *Numerical Analysis, Sixth Edition.* Brooks/-Cole Publishing Company, 1997.

[60] Microsoft Corporation. I/O Flow and dispatching in WDF drivers. White paper, Microsoft Corporation, 2007. Available online (63 pages).

[61] Microsoft Corporation. Handling IRPs: What every driver writer needs to know. White paper, Microsoft Corporation, 2006. Available online (21 pages).

[62] Walter Oney. *Programming the Microsoft Windows Driver Model.* Microsoft Press, 2002.

[63] Jeffery A. Gliner and George A. Morgan. *Research Methods in Applied Settings: An Integrated Approach to Design and Analysis.* Lawrence Erlbaum Associates, Inc., 2000.

[64] Thomas R. Black. *Doing Quantitative Research in the Social Sciences: An Integrated Approach to Research Design, Measurement and Statistics.* Sage Publications, Inc., 1999.

[65] James T. McClave and Terry Sincich. *Statistics, ninth edition.* Printice Hall, Inc., 2003.

[66] Julie Pallant. *SPSS Survival Manual, Second Edition.* Open University Press, 2005.

[67] Jerry Banks and II John S. Carson. *Discrete-Event System Simulation.* Prentice-Hall , Inc., 1984.

[68] Axel Buchner, Edgar Erdfelder, and Franz Faul. G*power software. *http://www.psycho.uni-duesseldorf.de/aap/projects/gpower/*, February 2010.

APPENDIX A: DRIVER SOURCE CODE

This section of the Appendix contains the source code for the Virtual Dynamic Tunnel filter device driver. The code was originally from a floating-point unit test sample by the Microsoft Corporation and Walter Oney. It provided the skeleton for the Virtual Dynamic Tunnel filter driver, shown here.

A.1 control.cpp

```
// Control.cpp -- IOCTL handlers for fputest driver
// Copyright (C) 1999 by Walter Oney
// All rights reserved
#include "control.h"
#include "interp.h"
// This sample is only appropriate for x86 targets

//#ifndef _X86_
//#error The FPUTEST sample is only for the x86 platform

// Whenever a C program uses floating point, the compiler generates a reference
// to the dummy symbol __fltused in order to drag in the runtime support. We
// don't want the standard support in a kernel-mode driver, however, so we
// satisfy the reference this way:

extern "C" ULONG _fltused = 0;
// Many floating point functions, including SIN and SQRT, are implemented by
// runtime library routines even though the x87 coprocessor has instructions
// for performing them. Since we can't link these routines into a driver, we
// need to supply intrinsic implementations for them, such as the following. Since
// these routines just use the native coprocessor instructions, they imply default
// error handling.
#pragma warning( push )
#pragma warning( disable : 4725 )
inline double myatan2(double y, double x)
{
double result;

_asm
{
        fld y
fld x
fpatan
fstp result
}

return result;
}
#pragma warning( pop )

inline long my_ftol(double f)
    {
    long i;

    _asm
        {
        fld f
        fistp i
        }
    return i;
    }


inline double sin(double x)
{
double result;

_asm
{
fld x
fsin
fstp result
}

return result;
}


inline double cos(double x)
{
double result;

_asm
{
fld x
fcos
fstp result
}

return result;
}

inline double sqrt(double x)
{
double result;

_asm
{
fld x
fsqrt
fstp result
}

return result;
}


bool is_above (double c_ratio, long Sumx, long Sumy)
{
    bool retval = true;
    double Ly, dSumx, dSumy;

    dSumx = (double) Sumx;
    dSumy = (double) Sumy;

    Ly = c_ratio * dSumx;

    if (Ly > dSumy)
        retval = false;
    return retval;
}

inline bool is_same_sign (long x, long y)
{
    return ( ((x >= 0) && (y  >= 0)) || ((x < 0)  && (y < 0)) ) ? true : false;
}

int current_pos = 1;
```

```
double Cx = 0.0;
double Cy = 0.0;

long prev_Cy = 0;
long prev_Cx = 0;

double prev_theta = 0.0;
double d = 0.0;
double d_new_x, d_new_y;

double tb[]         = { 1.0, 2.0, 3.0, 4.0 };
double xb[]         = { 0.0, 0.0, 0.0, 0.0 };
double yb[]         = { 0.0, 0.0, 0.0, 0.0 };

double prev_xb[]    = { 0.0, 0.0, 0.0, 0.0 };
double prev_yb[]    = { 0.0, 0.0, 0.0, 0.0 };

MyBaryRat_interp mybaryrat_func1;
MyBaryRat_interp mybaryrat_func2;

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

void copyarray(double a[], double b[], int size)
{
    double temp;
    for (int i = 0; i < size; ++i)
    {
        temp = a[i];
        a[i] = b[i];
        b[i] = temp;
    }
}

void cleararray(double a[], int size)
{
    for (int i=0; i < size; ++i)
        a[i] = 0.0;
}

void apply_virtual_tunnel(LONG X, LONG Y, LONG * new_x, LONG * new_y, int max_q_size, int m)
{
    // store our data points
    if (current_pos == 1)
    {
        xb[0] = prev_xb[max_q_size-1] + (double) X;
        yb[0] = prev_yb[max_q_size-1] + (double) Y;
    }
    else
    {
        xb[current_pos-1] = xb[current_pos-2] + (double) X;
        yb[current_pos-1] = yb[current_pos-2] + (double) Y;
    }

    // Apply previous values
    if (!( (prev_yb[max_q_size-1] == 0.0) && (prev_xb[max_q_size-1] == 0.0) ))
    {
        Cx = mybaryrat_func1.interp( (double) current_pos+max_q_size );
        Cy = mybaryrat_func2.interp( (double) current_pos+max_q_size );

        d = sqrt( (xb[current_pos-1]-Cx)*(xb[current_pos-1]-Cx) +
                  (yb[current_pos-1]-Cy)*(yb[current_pos-1]-Cy) );

        prev_theta = myatan2( (yb[current_pos-1]-Cy), (xb[current_pos-1]-Cx) );
        double a = abs((d - d/(double)m) * cos(prev_theta));
        double b = abs((d - d/(double)m) * sin(prev_theta));

        if (xb[current_pos-1] >= Cx)
            d_new_x = Cx + a;
        else
            d_new_x = Cx - a;

        if (yb[current_pos-1] >= Cy)
            d_new_y = Cy - b;
        else
            d_new_y = Cy + b;

//        cout <<  "\t" << xb[current_pos-1] << "\t" << yb[current_pos-1] << "\t"
//                       << Cx << "\t" << Cy << "\t" << d_new_x << "\t" << d_new_y;
        *new_x = my_ftol( X - (xb[current_pos-1] - d_new_x) );
        *new_y = my_ftol( Y - (yb[current_pos-1] - d_new_y) );

        KdPrint(("###4- new_x: %d, new_y: %d\n", *new_x, *new_y));

        if ( abs(*new_x) > abs(X) ) *new_x = X;
        if ( abs(*new_y) > abs(Y) ) *new_y = Y;

        KdPrint(("###4- new_x: %d, new_y: %d\n", *new_x, *new_y));
    }
}


#pragma LOCKEDCODE

NTSTATUS DoTunnel(/*char* irql,*/ LONG X, LONG Y, LONG * new_x,
                                  LONG * new_y, int max_q_size, int m )
{ // DoTest

    static LARGE_INTEGER prev_time;
    LARGE_INTEGER now;
// Before performing any floating point operations in a thread (other than
// a kernel mode thread we created by calling PsCreateSystemThread), save
// the non-volatile state of the coprocessor. This operation should also
// reinitialize the coprocessor. In Win98, however, the code is optimized
// in such a way that it may not actually do an FSAVE, so the initialization
// may not actually occur. It's better practice to perform our own FINIT and
// then load a control word containing whatever values we'd like, as shown
// here.

KFLOATING_SAVE savearea;
NTSTATUS status = KeSaveFloatingPointState(&savearea);
if (!NT_SUCCESS(status))
{
KdPrint(("Moufiltr - KeSaveFloatingPointState failed - %X\n", status));
return status;
        }

// Although we can use structured exceptions to catch floating point errors, the
// compiler may insert WAIT instructions at random places outside the scope of our
// __try blocks, leading to bug checks that we can't control. Rather than trying to
// work around this, it's just simpler to mask all floating point exceptions. That
// causes the FPU to perform its default fixup action for any error.

USHORT cw = 0x03FF; // round to nearest, 64-bit precision, all exceptions masked

_asm
{ // initialize FPU
fninit
fldcw cw
} // initialize FPU

    KeQuerySystemTime(&now);  // returns System time as a count of 100-nanosecond
                              //         (10-7) intervals since January 1, 1601.
    long diff = (long) (now.QuadPart - prev_time.QuadPart);
    diff = diff/10000; // divide by 10,000 (10^4) to change to milliseconds
    prev_time = now;

    *new_x = X;
    *new_y = Y;
    KdPrint(("###1- X: %d, Y: %d, diff: %d, current_pos: %d\n", X, Y, diff, current_pos ));

    if (diff >= 100)
    {
        current_pos = 1;
        cleararray( xb,      max_q_size );
        cleararray( yb,      max_q_size );
        cleararray( prev_xb, max_q_size );
        cleararray( prev_yb, max_q_size );
    }

    if (current_pos <= max_q_size)
    {
        apply_virtual_tunnel(X, Y, new_x, new_y, max_q_size, m);
        current_pos++;
    }
    else
    {
        copyarray(prev_xb, xb, max_q_size);
        copyarray(prev_yb, yb, max_q_size);

        MyBaryRat_interp func1;
        func1.construct(tb, prev_xb, 1);
        MyBaryRat_interp func2;
        func2.construct(tb, prev_yb, 1);

        mybaryrat_func1 = func1;
        mybaryrat_func2 = func2;

        current_pos = 1;

        // Apply for last value
        apply_virtual_tunnel(X, Y, new_x, new_y, max_q_size, m);
}


//    KdPrint(("###4- new_x: %d, new_y: %d\n", *new_x, *new_y));


// If MMX instructions are available on this processor, exercise a few of them.
if (ExIsProcessorFeaturePresent(PF_MMX_INSTRUCTIONS_AVAILABLE))
{ // perform MMX test
static UCHAR opnd1[8] = {0, 1, 2, 3, 4, 5, 6, 7};
static UCHAR opnd2[8] = {8, 7, 6, 5, 4, 3, 2, 1};
static UCHAR expect[8] = {8, 8, 8, 8, 8, 8, 8, 8};
UCHAR result[8];

_asm
{
movq mm0, opnd1 ; load 8 8-bit values
paddsb mm0, opnd2 ; do 8 additions at once
movq result, mm0 ; save result
emms ; end MMX stream
}
} // perform MMX test
else
KdPrint(("Moufiltr - skipping MMX test because feature not available\n"));

KeRestoreFloatingPointState(&savearea);
return status;
} // DoTest
// ####################################################################################

//#endif // _X86_
```

A.2 moufiltr.c

```
/*--
Copyright (c) 2008  Microsoft Corporation

Module Name:

    moufiltr.c

Abstract:

Environment:

    Kernel mode only- Framework Version

Notes:


--*/

#include "moufiltr.h"
#include "control.h"

#ifdef ALLOC_PRAGMA
#pragma alloc_text (INIT, DriverEntry)
#pragma alloc_text (PAGE, MouFilter_EvtDeviceAdd)
#pragma alloc_text (PAGE, MouFilter_EvtIoInternalDeviceControl)
#endif

#pragma warning(push)
#pragma warning(disable:4055) // type case from PVOID to PSERVICE_CALLBACK_ROUTINE
#pragma warning(disable:4152) // function/data pointer conversion in expression

// Globals
HANDLE hfile;              // the output from this process
int algorithm    = -1;
int distance     = -1;
int queue_length = -1;

// ******************
// HELPER FUNCTIONS
// ******************
int my_atoi(const char *string)
{ // non-negative only
    int i;
    i=-9999;
    if ( (0 != string) && (0<strlen(string)) )
    {
        i = 0;
        while(*string)
        {
            if (!(*string>='0' && *string<='9' ))
                return -9999;
            i=(i<<3) + (i<<1) + (*string - '0');
            string++;
        }
    }
    return i;
}

int Search( char *x,
            char *y )
{
   int i, j;
   int m, n;
   int retval;

   retval = -1;
   if ((0 != x && 0 != y)
        && (0<strlen(x)) && (0<strlen(y))
      )
   {
       m = strlen(x);
       n = strlen(y);


       for (j = 0; j <= (n - m); ++j)
       {
          for (i = 0; i < m && x[i] == y[i + j]; ++i);
            if (i >= m)
            {
                retval = j;
            }
       }
   }
   return retval;
}

int extractNumberFromString(char * source_buff, int pos,
                             char * ret_buff,    int size)
{   // right now we only handle positive integers
    int cnt = 0;

    char * ptr = &source_buff[pos];

    // find start of number
    while (*ptr && !(*ptr>='0' && *ptr<='9' ))
        ptr++;

    // while we see digits, store them
    while (*ptr && (cnt < size) && (*ptr>='0' && *ptr<='9' ) )
    {
        ret_buff[cnt++] = *ptr++;
    }

    // we ran off the end of the buffer
    if (cnt == size)
    {
        ret_buff[0] = '\0';
        return 0;
    }
    else
        ret_buff[cnt] = '\0';

    return 1;
}
// ********************

/*
***************************************************************************
DRIVER ENTRY
***************************************************************************
*/
NTSTATUS
DriverEntry (
    IN  PDRIVER_OBJECT  DriverObject,
    IN  PUNICODE_STRING RegistryPath
    )
/*++
Routine Description:

     Installable driver initialization entry point.
    This entry point is called directly by the I/O system.

--*/
{
    WDF_DRIVER_CONFIG               config;
    NTSTATUS                        status;
```

```
OBJECT_ATTRIBUTES oa;
IO_STATUS_BLOCK iostatus;
UNICODE_STRING pathname;

#define BUFFER_SIZE 128
CHAR buffer[BUFFER_SIZE];
#define NUM_BUFF_SIZE 10
CHAR num_buff[NUM_BUFF_SIZE];
int a, d, q;

//size_t  cb;
LARGE_INTEGER      byteOffset;

    DebugPrint(("Mouse Filter Driver Sample - Driver Framework Edition.\n"));
    DebugPrint(("Built %s %s\n", __DATE__, __TIME__));

    // Initiialize driver config to control the attributes that
    // are global to the driver. Note that framework by default
    // provides a driver unload routine. If you create any resources
    // in the DriverEntry and want to be cleaned in driver unload,
    // you can override that by manually setting the EvtDriverUnload in the
    // config structure. In general xxx_CONFIG_INIT macros are provided to
    // initialize most commonly used members.
    config.EvtDriverUnload = MyDriverUnload;

    WDF_DRIVER_CONFIG_INIT(
        &config,
        MouFilter_EvtDeviceAdd
    );

    //
    // Create a framework driver object to represent our driver.
    //
    status = WdfDriverCreate(DriverObject,
                            RegistryPath,
                            WDF_NO_OBJECT_ATTRIBUTES,
                            &config,
                            WDF_NO_HANDLE); // hDriver optional
    if (!NT_SUCCESS(status)) {
        DebugPrint( ("WdfDriverCreate failed with status 0x%x\n", status));
    }


RtlInitUnicodeString(&pathname, L"\\DosDevices\\C:\\config.txt");
InitializeObjectAttributes(&oa, &pathname, OBJ_CASE_INSENSITIVE
                                    | OBJ_KERNEL_HANDLE, NULL, NULL);
RtlZeroMemory(buffer, BUFFER_SIZE);
status = ZwCreateFile(&hfile,
                        GENERIC_READ,
                        &oa,
                        &iostatus,
                        NULL,
                        FILE_ATTRIBUTE_NORMAL,
                        0,
                        FILE_OPEN,
                        FILE_SYNCHRONOUS_IO_NONALERT,
                        NULL, 0);
if (NT_SUCCESS(status)) {
    byteOffset.LowPart = byteOffset.HighPart = 0;
    status = ZwReadFile(hfile, NULL, NULL, NULL, &iostatus,
                          buffer, BUFFER_SIZE, &byteOffset, NULL);
    if(NT_SUCCESS(status)) {
        buffer[BUFFER_SIZE-1] = '\0';

        DebugPrint(("###### File Buffer: %s\n", buffer));

        // "alg:0 dist:1 queue:10"
        a = Search("alg:", buffer);
        extractNumberFromString(buffer, a, num_buff, NUM_BUFF_SIZE);
        algorithm = my_atoi(num_buff);
        DebugPrint(("###### algorithm: %d\n", algorithm));

        d = Search ("dist:", buffer);
        extractNumberFromString(buffer, d, num_buff, NUM_BUFF_SIZE);
        distance = my_atoi(num_buff);
        DebugPrint(("###### distance: %d\n", distance));

        q = Search ("queue:", buffer);
        extractNumberFromString(buffer, q, num_buff, NUM_BUFF_SIZE);
        queue_length = my_atoi(num_buff);
        DebugPrint(("###### queue_length: %d\n", queue_length));

    }
    ZwClose(hfile);
}

    return status;
}

NTSTATUS
MouFilter_EvtDeviceAdd(
    IN WDFDRIVER        Driver,
    IN PWDFDEVICE_INIT  DeviceInit
    )
/*++
Routine Description:

    EvtDeviceAdd is called by the framework in response to AddDevice
    call from the PnP manager. Here you can query the device properties
    using WdfFdoInitWdmGetPhysicalDevice/IoGetDeviceProperty and based
    on that, decide to create a filter device object and attach to the
    function stack.

    If you are not interested in filtering this particular instance of the
    device, you can just return STATUS_SUCCESS without creating a framework
    device.

Arguments:

    Driver - Handle to a framework driver object created in DriverEntry

    DeviceInit - Pointer to a framework-allocated WDFDEVICE_INIT structure.

Return Value:

    NTSTATUS

--*/
{
    WDF_OBJECT_ATTRIBUTES   deviceAttributes;
    NTSTATUS                            status;
    WDFDEVICE                          hDevice;
    WDF_IO_QUEUE_CONFIG        ioQueueConfig;

    UNREFERENCED_PARAMETER(Driver);

    PAGED_CODE();

    DebugPrint(("Enter FilterEvtDeviceAdd \n"));

    //
    // Tell the framework that you are filter driver. Framework
    // takes care of inherting all the device flags & characterstics
    // from the lower device you are attaching to.
    //
    WdfFdoInitSetFilter(DeviceInit);

    WdfDeviceInitSetDeviceType(DeviceInit, FILE_DEVICE_MOUSE);

    WDF_OBJECT_ATTRIBUTES_INIT_CONTEXT_TYPE(&deviceAttributes,
        DEVICE_EXTENSION);


    //
    // Create a framework device object.  This call will in turn create
    // a WDM deviceobject, attach to the lower stack and set the
    // appropriate flags and attributes.
    //
    status = WdfDeviceCreate(&DeviceInit, &deviceAttributes, &hDevice);
    if (!NT_SUCCESS(status)) {
        DebugPrint(("WdfDeviceCreate failed with status code 0x%x\n", status));
        return status;
    }


    //
    // Configure the default queue to be Parallel. Do not use sequential queue
    // if this driver is going to be filtering PS2 ports because it can lead to
    // deadlock. The PS2 port driver sends a request to the top of the stack when it
    // receives an ioctl request and waits for it to be completed. If you use a
    // a sequential queue, this request will be stuck in the queue because of the
    // outstanding ioctl request sent earlier to the port driver.
    //
    WDF_IO_QUEUE_CONFIG_INIT_DEFAULT_QUEUE(&ioQueueConfig,
                             WdfIoQueueDispatchParallel);

    //
    // Framework by default creates non-power managed queues for
    // filter drivers.
    //
    ioQueueConfig.EvtIoInternalDeviceControl = MouFilter_EvtIoInternalDeviceControl;

    status = WdfIoQueueCreate(hDevice,
                            &ioQueueConfig,
                            WDF_NO_OBJECT_ATTRIBUTES,
                            WDF_NO_HANDLE // pointer to default queue
                            );
    if (!NT_SUCCESS(status)) {
        DebugPrint( ("WdfIoQueueCreate failed 0x%x\n", status));
        return status;
    }

    return status;
}



VOID
MouFilter_DispatchPassThrough(
    __in WDFREQUEST Request,
    __in WDFIOTARGET Target
    )
/*++
Routine Description:

    Passes a request on to the lower driver.


--*/
{
    //
    // Pass the IRP to the target
    //

    WDF_REQUEST_SEND_OPTIONS options;
    BOOLEAN ret;
    NTSTATUS status = STATUS_SUCCESS;

    //
    // We are not interested in post processing the IRP so
    // fire and forget.
    //
    WDF_REQUEST_SEND_OPTIONS_INIT(&options,
                                  WDF_REQUEST_SEND_OPTION_SEND_AND_FORGET);

    ret = WdfRequestSend(Request, Target, &options);

    if (ret == FALSE) {
        status = WdfRequestGetStatus (Request);
        DebugPrint( ("WdfRequestSend failed: 0x%x\n", status));
        WdfRequestComplete(Request, status);
    }

    return;
}

VOID
MouFilter_EvtIoInternalDeviceControl(
    IN WDFQUEUE      Queue,
    IN WDFREQUEST    Request,
    IN size_t        OutputBufferLength,
    IN size_t        InputBufferLength,
    IN ULONG         IoControlCode
    )
/*++

Routine Description:

    This routine is the dispatch routine for internal device control requests.
    There are two specific control codes that are of interest:

    IOCTL_INTERNAL_MOUSE_CONNECT:
        Store the old context and function pointer and replace it with our own.
        This makes life much simpler than intercepting IRPs sent by the RIT and
        modifying them on the way back up.

    IOCTL_INTERNAL_I8042_HOOK_MOUSE:
        Add in the necessary function pointers and context values so that we can
        alter how the ps/2 mouse is initialized.

    NOTE:  Handling IOCTL_INTERNAL_I8042_HOOK_MOUSE is *NOT* necessary if
           all you want to do is filter MOUSE_INPUT_DATAs.  You can remove
           the handling code and all related device extension fields and
           functions to conserve space.


--*/
{

    PDEVICE_EXTENSION           devExt;
    PCONNECT_DATA               connectData;
    NTSTATUS                   status = STATUS_SUCCESS;
    WDFDEVICE                 hDevice;
    size_t                           length;

    UNREFERENCED_PARAMETER(OutputBufferLength);
    UNREFERENCED_PARAMETER(InputBufferLength);

    PAGED_CODE();

    hDevice = WdfIoQueueGetDevice(Queue);
    devExt = FilterGetData(hDevice);

    switch (IoControlCode) {

    //
    // Connect a mouse class device driver to the port driver.
    //
    case IOCTL_INTERNAL_MOUSE_CONNECT:
        //
        // Only allow one connection.
        //
        if (devExt->UpperConnectData.ClassService != NULL) {
            status = STATUS_SHARING_VIOLATION;
            break;
        }

        //
        // Copy the connection parameters to the device extension.
        //
         status = WdfRequestRetrieveInputBuffer(Request,
                            sizeof(CONNECT_DATA),
                            &connectData,
                            &length);
        if(!NT_SUCCESS(status)){
            DebugPrint(("WdfRequestRetrieveInputBuffer failed %x\n", status));
            break;
        }


        devExt->UpperConnectData = *connectData;

        //
        // Hook into the report chain.  Everytime a mouse packet is reported to
        // the system, MouFilter_ServiceCallback will be called
        //
        connectData->ClassDeviceObject = WdfDeviceWdmGetDeviceObject(hDevice);
        connectData->ClassService = MouFilter_ServiceCallback;

        break;

    //
    // Disconnect a mouse class device driver from the port driver.
    //
    case IOCTL_INTERNAL_MOUSE_DISCONNECT:

        //
        // Clear the connection parameters in the device extension.
        //
        // devExt->UpperConnectData.ClassDeviceObject = NULL;
        // devExt->UpperConnectData.ClassService = NULL;

        status = STATUS_NOT_IMPLEMENTED;
        break;


    //
    // Might want to capture this in the future.  For now, then pass it down
    // the stack.  These queries must be successful for the RIT to communicate
    // with the mouse.
    //
    case IOCTL_MOUSE_QUERY_ATTRIBUTES:
    default:
        break;
    }

    if (!NT_SUCCESS(status)) {
        WdfRequestComplete(Request, status);
        return ;
    }

    MouFilter_DispatchPassThrough(Request,WdfDeviceGetIoTarget(hDevice));
}



VOID
MouFilter_ServiceCallback(
    IN PDEVICE_OBJECT DeviceObject,
    IN PMOUSE_INPUT_DATA InputDataStart,
    IN PMOUSE_INPUT_DATA InputDataEnd,
    IN OUT PULONG InputDataConsumed
    )
/*++

Routine Description:

    Called when there are mouse packets to report to the RIT.  You can do
    anything you like to the packets.  For instance:

    o Drop a packet altogether
    o Mutate the contents of a packet
    o Insert packets into the stream

Arguments:

    DeviceObject - Context passed during the connect IOCTL

    InputDataStart - First packet to be reported

    InputDataEnd - One past the last packet to be reported.  Total number of
                   packets is equal to InputDataEnd - InputDataStart

    InputDataConsumed - Set to the total number of packets consumed by the RIT
                        (via the function pointer we replaced in the connect
                        IOCTL)

Return Value:

    Status is returned.

--*/
{
    PDEVICE_EXTENSION   devExt;
PMOUSE_INPUT_DATA pCursor; // cursor for looping
    LONG new_x, new_y;
    NTSTATUS status;
    int relative = 0;

    WDFDEVICE   hDevice;

//DbgPrint("FERROLDBG - I'm in the callback\n");

    hDevice = WdfWdmDeviceGetWdfDeviceHandle(DeviceObject);

    devExt = FilterGetData(hDevice);

// this is where we can mangle/delete/add packets
for (pCursor = InputDataStart; pCursor < InputDataEnd; pCursor++)
    {
// do something with the current MOUSE_INPUT_DATA
        relative = (MOUSE_MOVE_RELATIVE | pCursor->Flags) ? 1 : 0;

        if (1 == algorithm)
        {
            status = DoTunnel( pCursor->LastX, pCursor->LastY,
                               &new_x, &new_y, 4, distance );
            pCursor->LastX = new_x;
            pCursor->LastY = new_y;
        }
}

    //
    // UpperConnectData must be called at DISPATCH
    //
    (*(PSERVICE_CALLBACK_ROUTINE) devExt->UpperConnectData.ClassService)(
        devExt->UpperConnectData.ClassDeviceObject,
        InputDataStart,
        InputDataEnd,
        InputDataConsumed
        );
}
#pragma warning( push )
#pragma warning( disable : 4100 )
VOID
MyDriverUnload (
IN WDFDRIVER Driver
    )
{
    DbgPrint("FERROLDBG - Unloading driver");
}

#pragma warning(pop)

#pragma warning(pop)
```

APPENDIX B: HEAD-TILT MOUSE CODE

This Appendix contains the source code related to the head-tilt mouse device.

B.1 head_test2.bs2

This is the head-tilt device source code written in the BASIC Stamp language.

```
' {$STAMP BS2}
' {$PBASIC 2.5}

  Xin           PIN 6
  Yin           PIN 7
  HiPulse       CON 1 ' measure high-going pulse
  TX            CON 0
  Lowlimit      CON 4000

' Set scale factor for PULSIN
#SELECT $STAMP
#CASE BS2, BS2E
Scale CON $200 ' 2.0 us per unit
#CASE BS2SX
Scale CON $0CC ' 0.8 us per unit
#CASE BS2P
Scale CON $0C0 ' 0.75 us per unit
#CASE BS2PE
Scale CON $1E1 ' 1.88 us per unit
#ENDSELECT

  pulse         VAR Word ' pulse input
  x_tilt        VAR Word '
  y_tilt        VAR Word '

Main:
    PAUSE 50 ' Wait 0.1 seconds

DO
    GOSUB Read_x_tilt
    GOSUB Read_y_tilt

'    DEBUG "N:", HEX4 MyData, "X:", HEX4 y_tilt, "Y:", HEX4 x_tilt, CLREOL, CR
    SEROUT TX, 84, ["X:", HEX4 y_tilt, "Y:", HEX4 x_tilt]

'    PAUSE 100
LOOP
END

Read_x_tilt:
  PULSIN Xin, HiPulse, pulse ' read pulse output
  x_tilt = pulse */ Scale ' convert to uSecs
  x_tilt = (x_tilt-Lowlimit)/16
RETURN

Read_y_tilt:
  PULSIN Yin, HiPulse, pulse ' read pulse output
  y_tilt = pulse */ Scale ' convert to uSecs
  y_tilt = (y_tilt-Lowlimit)/16
RETURN
```

B.2 commport.cpp

This is the host computer source that calibrates the head-tilt mouse and moves the mouse cursor according to signals received from the head-tilt mouse device. It will also apply the Virtual Dynamic Tunnel depending on command line settings.

```
// commport.cpp : Defines the entry point for the application.
//
#include "stdafx.h"
#include "commport.h"
#include "windows.h" // needed for Sleep()
#include <iostream>
#include <time.h>
#include <math.h>
#include <list>

#include "nr3.h"
#include "interp_1d.h"

const int DATA_POINTS = 16;//18;        // should be a parameter
const unsigned int SIZE_DATA_POINT = 6;                  // set by data
const int SIZE_DATA_POINTS = SIZE_DATA_POINT*2;  // set by data
const int buffersize = DATA_POINTS * SIZE_DATA_POINTS;

int errorcount = 0;
HANDLE comReadHandle = 0;

char str[] =
"N:FD54X:003DY:003DN:FD58X:003CY:003DN:FD5CX:003BY:003D\
N:FD52X:003FY:003BN:FD62X:003EY:003EN:FD58X:003DY:003E\
N:FD54X:003FY:003FN:FD64X:003FY:003EN:FD48X:0041Y:003C\
N:FD48X:0042Y:003BN:FD5AX:003FY:003BN:FD58X:003DY:0039\
N:FD58X:0040Y:003AN:FD54X:0041Y:003EN:FD56X:003EY:003C\
N:FD54X:003DY:003BN:FD5AX:003EY:003DN:FD5AX:003FY:003D\
N:FD58X:003DY:003DN:FD54X:003DY:003CN:FD58X:003EY:003D\
N:FD58X:0040Y:003CN:FD58X:003EY:003DN:FD5EX:003EY:003C\
N:FD58X:003DY:003CN:FD54X:0040Y:003DN:FD5AX:003FY:003E\
N:FD5EX:003EY:003EN:FD58X:003FY:003EN:FD54X:0040Y:003E\
N:FD54X:003FY:003EN:FD54X:003FY:003FN:FD52X:003EY:003D\
N:FD5EX:0040Y:003FN:FD52X:0040Y:003FN:FD58X:003EY:0040\
N:FD56X:003FY:003FN:FD58X:003FY:003DN:FD50X:0043";

// Calibration stats
int X_CENTER = 0;
int Y_CENTER = 0;
int X_LOW    = 0;
int X_HIGH   = 0;
int Y_LOW    = 0;
int Y_HIGH   = 0;

struct valstats
{
    double total;
    long   count;
    double avg;
    double rmt;
    double rmt_avg;
    int    highest;
    int    lowest;
};

struct valstats x_stats;
struct valstats y_stats;
//char str[buffersize];
```

```
void ErrorHandler( char * message, DWORD error )
{
errorcount++;
std::cout << message << std::endl;
std::cout << "Error number = " << error << std::endl;
if (errorcount > 3) ExitProcess(1);
}

int parseValue ( char * chr, char ** data )
{
char * idx=0;
int val = 0;
idx = (char *) *data;
char hex_str[5];

// process the buffer
if (idx && (strlen(idx) >= SIZE_DATA_POINT) )
    {
idx = strstr(idx, chr); // find the occurrance of "A:"
if (idx)
{
    memset(hex_str, 0, 5);
    strncpy_s(hex_str, idx+2, 4);  // get the string rep. hex number
    sscanf_s(hex_str, "%x", &val);  // convert to hex value
    *data = idx+6;
}

//printf("idx:%d str:%d str+50-idx:%d\n", idx, str, str+50-idx);
}
    return val;
}

void print_values_and_stats(
                            const struct valstats & x_stats,
                            const struct valstats & y_stats)
{
    printf("X: total:%6.0f cnt:%d avg:%6.0f rmt:%4.2f rmt_avg:%4.2f high:%d low:%d\n",
            x_stats.total, x_stats.count, x_stats.avg, x_stats.rmt,
            x_stats.rmt_avg, x_stats.highest, x_stats.lowest );
    printf("Y: total:%6.0f cnt:%d avg:%6.0f rmt:%4.2f rmt_avg:%4.2f high:%d low:%d\n",
            y_stats.total, y_stats.count, y_stats.avg, y_stats.rmt,
            y_stats.rmt_avg, y_stats.highest, y_stats.lowest );
}

void init_stats(struct valstats & stats)
{
    memset(&stats, 0, sizeof(stats));
    stats.lowest = 65535;
    stats.highest = -65535;
}

void update_stats( const int & v, struct valstats & stats)
{
    stats.total += (double) v;
    stats.count++;
    stats.avg = stats.total / (double)stats.count;
    stats.rmt_avg += pow((stats.avg - (double)v), 2.0);
    stats.rmt = sqrt( stats.rmt_avg / stats.count );
    if (v > stats.highest) stats.highest = v;
    if (v < stats.lowest)  stats.lowest = v;
}

void cal_and_store_stats(
                            const int & x,
                            const int & y,
                            struct valstats & x_stats,
                            struct valstats & y_stats)
{
    update_stats( x, x_stats );
    update_stats( y, y_stats );
```

```
}

void fill_buffer( char ** idx)
{
    // ***FOR SIMULATION***  Uncomment the line below and comment out
    //                       the rest of the function
    // NOTE: also set the while loop timer, in go(), up so that we terminate.
//    *idx = str;

    BOOL success;
    //char str[buffersize];
    DWORD numRead;

DWORD mask=0;
WaitCommEvent( comReadHandle, &mask, 0 );
// fill the buffer
memset(str, 0, buffersize);
success = ReadFile( comReadHandle, str, buffersize, &numRead, 0 );
if (!success)
    ErrorHandler("In ReadFile", GetLastError() );
// std::cout << "Raw:>>" << str << "<<" << std::endl;
*idx = (char *) str;
}

void process_buffer( char ** idx, int & x, int & y )
{
    if (*idx &&
           (((str + buffersize) - *idx) >= SIZE_DATA_POINTS) )
    {
        int value = parseValue( "X:", idx );

        x = value;
        value = parseValue( "Y:", idx );

        y = value;

    }
    else
        if ( ((str + buffersize) - *idx) < SIZE_DATA_POINTS )
        {
            // refill the buffer
            fill_buffer( idx );
        }

}

void calibrate_center(char ** idx)
{
    time_t start_time = time(NULL);  // get start time
double xSum, ySum;
int numDataPts = 0;
xSum = ySum = 0.0;
double avg = 0.0;
int secs = 1;
    int x,y;

    do
    {
        // process buffer
        process_buffer(idx, x, y);

        xSum += (double) x;
        ySum += (double) y;
        numDataPts++;

        // inside buffer loop
        if ( (time(NULL)-start_time) > secs )
        {
        std::cout << secs++ << " ";
        std::flush(std::cout);
        }
    }
    while ( (time(NULL)-start_time) < 11);

    X_CENTER = (int) (xSum/numDataPts);
    Y_CENTER = (int) (ySum/numDataPts);

std::cout << "\n\nx center: " << X_CENTER << " y center: "
                                  << Y_CENTER << std::endl;
}


void calibrate_variable(char ** idx, struct valstats & stats,
                        const int & variable )
{
    time_t start_time = time(NULL);  // get start time
    int secs = 1;
    int x,y;

    init_stats( stats );

    do
    {
        // process buffer
        process_buffer(idx, x, y);
//        printf("%x %x\n", x, y);

        switch (variable)
        {
            case 2:
                update_stats( x, stats );
                break;
            case 3:
                update_stats( y, stats );
                break;
            default: // something bad happened
                break;
        };

        // inside buffer loop
        if ( (time(NULL)-start_time) > secs )
        {
        std::cout << secs++ << " ";
        std::flush(std::cout);
        }

    }
    while ((time(NULL)-start_time)<11);

    switch (variable)
    {
        case 2:
            X_HIGH = (int) (stats.avg + 3.0*stats.rmt);
            X_LOW  = (int) (stats.avg - 3.0*stats.rmt);
            printf("\n\nX: avg:%6.0f rmt:%4.2f high:%d low:%d\n",
                stats.avg, stats.rmt,
                stats.highest, stats.lowest );
            break;
        case 3:
            Y_HIGH = (int) (stats.avg + 3.0*stats.rmt);
            Y_LOW  = (int) (stats.avg - 3.0*stats.rmt);
            printf("\n\nY: avg:%6.0f rmt:%4.2f high:%d low:%d\n",
                stats.avg, stats.rmt,
                stats.highest, stats.lowest );
            break;
        default: // something bad happened
            break;
    };
}
```

```
void calibrate_shoulder_tilt( char ** idx, struct valstats & x_stats )
{
    calibrate_variable(idx, x_stats, 2 );
}
void calibrate_forward_tilt( char ** idx, struct valstats & y_stats )
{
    calibrate_variable(idx, y_stats, 3 );
}

void calibrate (char ** idx)
{
    printf("==== Calibration ====\n\n");
    printf("---- Hold center position for 10 seconds.\n");
    calibrate_center( idx );
    printf("\n---- Side tilt Calibration ----\nTilt head towards left and right \\
            shoulder as much as is comfortable for 10 seconds.\n");
    calibrate_shoulder_tilt( idx, x_stats );
    printf("\n---- Forward/backward Calibration ----\nTilt head forwards and \\
            backwards as much as is comfortable for 10 seconds.\n");
    calibrate_forward_tilt( idx, y_stats );
}
void Usage( const char * argv[] )
{
    printf("Usage: %s [use_tunnel [comm_port]]\n", argv[0]);
    printf("\tuse_tunnel = < 1 | 0 >\n");
    printf("\tcomm_port = \\\\\\\\.\\\\COMNN where NN is the port number\n\n");
}
void connect_to_device( int argc, const char* argv[], int & use_tunnel )
{
BOOL success;
DCB dcb;
use_tunnel = 0;
do
{

    if ((1 == argc) || (2 == argc))
        // any COM port over 1 digit, must use the \\\\.\\ poop
    comReadHandle = CreateFile("\\\\.\\COM11", GENERIC_READ, 0, 0,
                                    OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0 );

if (2 == argc)
{
    use_tunnel = atoi(argv[argc-1]);
}
else if (3 == argc)
{
    comReadHandle = CreateFile( argv[argc-1], GENERIC_READ, 0, 0,
                                   OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0 );
    use_tunnel = atoi(argv[argc-2]);
}

if ( INVALID_HANDLE_VALUE == comReadHandle )
{
    char errstr[100];
    sprintf_s( errstr, "Could not connect to device. Try #%d", errorcount+2 );
ErrorHandler( errstr, GetLastError() );
}
}
while (INVALID_HANDLE_VALUE == comReadHandle);

// Get port settings
success = GetCommState( comReadHandle, &dcb );
if (!success) ErrorHandler("In GetCommState", GetLastError() );

dcb.BaudRate = 9600;
dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;

// Set new port settings
success = SetCommState( comReadHandle, &dcb );
if (!success) ErrorHandler("In SetCommState", GetLastError() );

// Register for Comm events
SetCommMask( comReadHandle, EV_RXCHAR );

std::cout << "Tunnel " << (use_tunnel ? "enabled" : "disabled")
                           << "..." << std::endl;

std::cout << "Communications open.  Processing Comm events..." << std::endl;
}

float fx, fX_CENTER, fX_LOW, fX_HIGH;
float fy, fY_CENTER, fY_LOW, fY_HIGH;
void moveCursor( POINT & pt, int x, int y )
{
    const float sensitivity = 0.1875;
    fx          = (float) x;
    fX_CENTER   = (float) X_CENTER;
    fX_LOW      = (float) X_LOW;
    fX_HIGH     = (float) X_HIGH;

    fy          = (float) y;
    fY_CENTER   = (float) Y_CENTER;
    fY_LOW      = (float) Y_LOW;
    fY_HIGH     = (float) Y_HIGH;

    // Handle x coordinate
    if ( x < X_CENTER )
    {
        if ( fx < (fX_CENTER - (fX_CENTER - fX_LOW) * 0.1) )
            pt.x = pt.x + (long) (abs(fx-fX_CENTER)*sensitivity);
    }
    else // x >= X_CENTER
    {
        if ( fx > ((fX_HIGH - fX_CENTER) * 0.1 + fX_CENTER) )
            pt.x = pt.x - (long) (abs(fx-fX_CENTER)*sensitivity);
    }

    // Handle y coordinate
    if ( y < Y_CENTER )
    {
        if ( fy < (fY_CENTER - (fY_CENTER - fY_LOW) * 0.1) )
            pt.y = pt.y - (long) (abs(fy-fY_CENTER)*sensitivity);
    }
    else // y >= Y_CENTER
    {
        if ( fy > ((fY_HIGH - fY_CENTER) * 0.1 + fY_CENTER) )
            pt.y = pt.y + (long) (abs(fy-fY_CENTER)*sensitivity);
    }

}


// ============================================================================
//  Virtual Tunnel
// ============================================================================
void myQuerySystemTime(ULARGE_INTEGER * now)
{
    //* Convert the SYSTEMTIME structure to a FILETIME structure.
    //* Copy the resulting FILETIME structure to a ULARGE_INTEGER structure.
    //* Use normal 64-bit arithmetic on the ULARGE_INTEGER value.
    SYSTEMTIME st;
    FILETIME   ft;
    GetSystemTime( &st );
    SystemTimeToFileTime( &st, &ft );
    now->HighPart = ft.dwHighDateTime;
    now->LowPart  = ft.dwLowDateTime;
}
```

```
// globals needed by virtualTunnel
const int MAX_Q_SIZE = 50;
int current_pos      = 1;
long T               = 0;
double Cx            = 0.0;
double Cy            = 0.0;
long prev_Cy         = 0;
long prev_Cx         = 0;
double theta         = 0.0;
double prev_theta    = 0.0;
double d             = 0.0;
double d_new_x, d_new_y;
long   *new_x,  *new_y;
int use_tunnel       = 1;

inline long my_ftol(double f)
    {
    long i;

    _asm
        {
        fld f
        fistp i
        }

    return i;
    }

bool is_above (double c_ratio, long Sumx, long Sumy)
{
    bool retval = true;
    double Ly, dSumx, dSumy;

    dSumx = (double) Sumx;
    dSumy = (double) Sumy;

    Ly = c_ratio * dSumx;

    if (Ly > dSumy)
        retval = false;
    return retval;
}

inline bool is_same_sign (long x, long y)
{
    return ( ((x >= 0) && (y  >= 0)) || ((x < 0)  && (y < 0)) ) ? true : false;
}


double tbd[]          = { 1.0, 2.0, 3.0, 4.0 };
double xbd[]          = { 0.0, 0.0, 0.0, 0.0 };
double ybd[]          = { 0.0, 0.0, 0.0, 0.0 };
double prev_xbd[]     = { 0.0, 0.0, 0.0, 0.0 };
double prev_ybd[]     = { 0.0, 0.0, 0.0, 0.0 };
VecDoub tb(4), xb(4), yb(4), prev_xb(4), prev_yb(4);

BaryRat_interp mybaryrat_func1;
BaryRat_interp mybaryrat_func2;

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

void copyarray(VecDoub & a, VecDoub & b, int size)
{
    double temp;
    for (int i = 0; i < size; ++i)
    {
        temp = a[i];
        a[i] = b[i];
        b[i] = temp;
    }
}

void cleararray(VecDoub & a, int size)
{
    for (int i=0; i < size; ++i)
        a[i] = 0.0;
}

void apply_virtual_tunnel(LONG X, LONG Y, LONG * new_x, LONG * new_y, int max_q_size, int m)
{
    // store our data points

    // Because our points are absolute and not relative to the prev. position,
    //          we don't need to add them.
    xb[current_pos-1] = (double) X;
    yb[current_pos-1] = (double) Y;


    // Apply previous values
    if (!( (prev_yb[max_q_size-1] == 0.0) && (prev_xb[max_q_size-1] == 0.0) ))
    {
        Cx = mybaryrat_func1.interp( (double) current_pos+max_q_size );
        Cy = mybaryrat_func2.interp( (double) current_pos+max_q_size );

        d = sqrt( (xb[current_pos-1]-Cx)*(xb[current_pos-1]-Cx) +
                  (yb[current_pos-1]-Cy)*(yb[current_pos-1]-Cy) );

        prev_theta = atan2( (yb[current_pos-1]-Cy), (xb[current_pos-1]-Cx) );
        double a = abs((d - d/(double)m) * cos(prev_theta));
        double b = abs((d - d/(double)m) * sin(prev_theta));

        if (xb[current_pos-1] >= Cx)
            d_new_x = Cx + a;
        else
            d_new_x = Cx - a;

        if (yb[current_pos-1] >= Cy)
            d_new_y = Cy - b;
        else
            d_new_y = Cy + b;


        // We do things differently here because we are dealing with absolute points
        if ( is_same_sign( (LONG) d_new_x, X ) )
        {
            if( abs(d_new_x) > abs(X) )
                d_new_x = X;
        }
        else if (d_new_x > X)
            d_new_x = X;

        if ( is_same_sign( (LONG) d_new_y, Y ) )
        {
            if( abs(d_new_y) > abs(Y) )
                d_new_y = Y;
        }
        else if (d_new_y > Y)
            d_new_y = Y;

        *new_x = my_ftol( d_new_x );
        *new_y = my_ftol( d_new_y );

    }
}
void virtualTunnel( POINT & pt, int max_q_size, int m )
{
    LONG X, Y, x, y;
    x = 0;
    y = 0;
    new_x = &x;
    new_y = &y;

    X = pt.x;
    Y = pt.y;

    static ULARGE_INTEGER prev_time;
    ULARGE_INTEGER now;

    myQuerySystemTime(&now);  // returns System time as a count of 100-nanosecond
                              // (10-7) intervals since January 1, 1601.
    long diff = (long) (now.QuadPart - prev_time.QuadPart);
    diff = diff/10000; // divide by 10,000 (10^4) to change to milliseconds
    prev_time = now;

    *new_x = X;
    *new_y = Y;

    //printf("###1- X: %d, Y: %d, diff: %d, current_pos: %d\n", X, Y, diff, current_pos );

    if (current_pos <= max_q_size)
    {
        if (diff < 100) // time since last point is below threshold.
        {
            apply_virtual_tunnel(X, Y, new_x, new_y, max_q_size, m);
            current_pos++;
        }
        else
        {
            current_pos = 1;
            cleararray( xb,      max_q_size );
            cleararray( yb,      max_q_size );
            cleararray( prev_xb, max_q_size );
            cleararray( prev_yb, max_q_size );
            apply_virtual_tunnel(X, Y, new_x, new_y, max_q_size, m);
        }
    }
    else
    {
        copyarray(prev_xb, xb, max_q_size);
        copyarray(prev_yb, yb, max_q_size);

        BaryRat_interp func1(tb, prev_xb, 1);
        BaryRat_interp func2(tb, prev_yb, 1);

        mybaryrat_func1 = func1;
        mybaryrat_func2 = func2;

        current_pos = 1;

        // Apply for last value
        apply_virtual_tunnel(X, Y, new_x, new_y, max_q_size, m);
}
    pt.x = *new_x;
    pt.y = *new_y;
    //printf("###4- new_x: %d, new_y: %d\n", pt.x, pt.y);
return;
}
// ================================================================================
//  End - Virtual Tunnel
// ================================================================================

void go (char ** idx)
{
    time_t start_time = time(NULL);  // get start time
    int x,y;

double xMax = (double) GetSystemMetrics( SM_CXSCREEN );
```

```
int secs = 0;

    do
    {
        // process buffer
        process_buffer(idx, x, y);

        POINT pt;
        GetCursorPos(&pt);

        moveCursor( pt, x, y );

        if ( use_tunnel )
            virtualTunnel( pt, 4, 2 );

        SetCursorPos(pt.x, pt.y);
    }
    while ( 1==1 ); //((time(NULL)-start_time) < 31);
}

void init_vectors()
{
    for (int i = 0; i < 4; ++i)
    {
        tb[i] = tbd[i];
        prev_xb[i] = prev_xbd[i];
        prev_yb[i] = prev_ybd[i];
    }
}

int main (int argc, const char* argv[])
{

    char * idx = 0;

    Usage( argv );

    init_vectors();

    connect_to_device( argc, argv, use_tunnel );

    fill_buffer( &idx );

    calibrate( &idx );

    go( &idx );

    return 0;
}
```

APPENDIX C: EXPERIMENT SOURCE CODE

This Appendix contains source code for the first and second experiment test applications.

C.1 Target acquisition experiment source

C.1.1 MouseTest.java

```
import javax.swing.*;

public class MouseTest {
  public MouseTest() {}
  public static void main(String[] args) {
    int seed = 1234;
    if (1==args.length)
    {
      seed = Integer.parseInt(args[0]);
    }
    MouseTest mouseTest1 = new MouseTest();
    JFrame frame = new MouseTestFrame(seed);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

C.1.2 MouseTestFrame.java

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import java.awt.HeadlessException;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.*;
import java.awt.event.*;

public class MouseTestFrame extends JFrame {
  public MouseTestFrame(int seed) throws HeadlessException {
    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
    setSize(dim.width, dim.height-25);
    setTitle("Mouse Test");
    panel = new TargetPanel(seed, ROUND_SIZE, TRIAL_SIZE, this);
    getContentPane().add(panel, BorderLayout.CENTER);

    JPanel buttonPanel = new JPanel();

    addButton(buttonPanel, "Start",
              new ActionListener() {
      public void actionPerformed(ActionEvent event) {
        addTarget();
      }
    });

    addButton(buttonPanel, "Close",
              new ActionListener()
    {
      public void actionPerformed(ActionEvent event)
      {
        System.exit(0);
      }
    });
    getContentPane().add(buttonPanel, BorderLayout.SOUTH);
  }

  public void addButton(Container c, String title, ActionListener listener)
  {
    JButton button = new JButton(title);
    c.add(button);
    button.addActionListener(listener);
  }
  public void addTarget()
  {
    Target target = new Target();
    panel.add(target);
    panel.paint(panel.getGraphics());
  }
  private TargetPanel panel;
  private int ROUND_SIZE = 8;
  private int TRIAL_SIZE = 3;
}
```

C.1.3 Target.java

```
import java.awt.geom.Ellipse2D;

public class Target {
  public Target() {
  }
  public Ellipse2D getShape()
  {
    return new Ellipse2D.Double(x, y, XSIZE, YSIZE);
  }
  public void setX(int inx) {x=inx;};
  public void setY(int iny) {y=iny;};
  private static final int XSIZE = 100;//40;
  private static final int YSIZE = 100;//40;
  private double x = 640;
  private double y = 400;
}
```

C.1.4 TargetPanel.java

```
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JFrame;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Color;
import java.awt.BorderLayout;
import java.util.Date;
import java.io.FileWriter;
import java.io.IOException;

public class TargetPanel extends JPanel {
  public class MyMouseListener extends MouseAdapter
  {
    public MyMouseListener(JPanel panel)
    {
      m_panel = panel;
      round_count=0;
      trial_count=0;
      start_time=0;
      m_output_file=null;
      m_tp = new TargetProcessor(m_seed);
      m_tp.generatePositions(0);
    };

    public void mouseClicked(MouseEvent e)
    {
      Date datetime = new Date();
      long curr_time = datetime.getTime();
      int x = e.getX();
      int y = e.getY();
      boolean hit = false;
      try
      {
        hit = m_t.getShape().getBounds().contains(x, y);
      }
      catch (Exception ex)
      {
      }

      try
      {
        double targ_x_f = m_t.getShape().getX();
        double targ_y_f = m_t.getShape().getY();
        int targ_x = (int) (targ_x_f + 20.0);
        int targ_y = (int) (targ_y_f + 20.0);

        if (hit) // we got a hit
        {
          if (start_time == 0)
          {
            start_time = curr_time;
            m_output_file = new FileWriter("MouseTestData.txt");
          }

          m_frame.setTitle((trial_count * m_round_size + round_count + 1)+" of "
                           +m_round_size*m_trial_size+" targets");
          m_output_file.write(curr_time + "," +
                             (trial_count * m_round_size + round_count + 1) +
                             ",1," + x + "," + y + "," + targ_x + "," + targ_y + "\n");
//          System.out.println(curr_time + " Trial " +
//                              (trial_count * m_round_size + round_count + 1) +
//                              " - Hit:" + x + "," + y);
          x = m_tp.target_positions[m_tp.test_positions[round_count]].x;
          y = m_tp.target_positions[m_tp.test_positions[round_count++]].y;
          m_t.setX(x);
          m_t.setY(y);

          if (round_count == m_round_size)
          {
            if (trial_count++ == (m_trial_size - 1))
            {
              m_output_file.close();
              System.out.println(
                  "Time difference between 1st hit and last hit: " +
                  (double) (curr_time - start_time) / 1000.0 + " secs.");
              System.exit(0);
            }
            // restart the trial
            m_tp.generatePositions(trial_count);
            round_count = 0;
          }
        }
        else // We got a miss
        {
          m_output_file.write(curr_time + "," +
                              (trial_count * m_round_size + round_count + 1) +
                              ",0," + x + "," + y +  "," + targ_x + "," +
                               targ_y + "\n");
//          System.out.println(curr_time + " Trial " +
//                             (trial_count * m_round_size + round_count + 1) +
//                             " - Miss:" + x + "," + y);
        }
      }

      catch (IOException ioe)
      {
        System.out.println(ioe.toString());
      }

      m_panel.paint(m_panel.getGraphics());
    }
    int round_count;
    int trial_count;
    JPanel m_panel;
    TargetProcessor m_tp;
  }
  private TargetPanel() {}
  public TargetPanel(int seed, int rs, int ts, JFrame frame)
  {
    m_trial_size = ts;
    m_round_size = rs;
    m_seed = seed;
    m_frame = frame;
  }
  public void add(Target t)
  {
    m_t = t;
    this.addMouseListener(new MyMouseListener(this));
  }
  public void paintComponent(Graphics g)
  {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    g2.setColor(Color.RED);
    if (null != m_t)
      g2.fill(m_t.getShape());
  }
  private Target m_t;
  private int m_trial_size;
  private int m_round_size;
  private long start_time;
  private FileWriter m_output_file;
  private JFrame m_frame;
  private int m_seed;
}
```

C.1.5 TargetProcessor.java

```
import java.util.Random;
import java.awt.Point;
import java.awt.Dimension;
import java.awt.Toolkit;

public class TargetProcessor {
  public TargetProcessor(int seed) {
    m_seed = seed;
  }
  void generatePositions(int trial)
  {
    test_positions = new int[TEST_SIZE];
    target_positions = new Point[TEST_SIZE];
    Random ran = new Random();
    ran.setSeed(m_seed+trial);
    for (int i=0; i<TEST_SIZE; ++i)
    {
      boolean found = false;
      int val=0;
      do
      {
        val = ran.nextInt(TEST_SIZE);
        found = false;
        for (int j = 0; j < i; ++j)
        {
          if (test_positions[j] == val)
          {
            found = true;
            break;
          }
        }

      } while(found);
      test_positions[i] = val;
    }
    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
    int y_max = dim.height;
    int x_max = dim.width;
//    System.out.println("Max x:"+x_max+"Max y:"+y_max);
    target_positions[0] = new Point(x_max/2, y_max/16);
    target_positions[1] = new Point(x_max*15/16, y_max/16);
    target_positions[2] = new Point(x_max*15/16, y_max/2);
    target_positions[3] = new Point(x_max*15/16, y_max*15/16-100);
    target_positions[4] = new Point(x_max/2, y_max*15/16-100);
    target_positions[5] = new Point(x_max/16, y_max*15/16-100);
    target_positions[6] = new Point(x_max/16, y_max/2);
    target_positions[7] = new Point(x_max/16, y_max/16);

    for(int i=0; i<TEST_SIZE;++i)
    {
//      System.out.println("target_postions["+i+"]="+"(x="+target_positions[i].x
//                          +",y="+target_positions[i].y+")");
      System.out.println("test_positions["+i+"]="+test_positions[i]);
    }
  }
  private int TEST_SIZE = 8;
  private int m_seed;
  public int test_positions[];
  public Point target_positions[];
}
```

C.2 Tunnel experiment source

C.2.1 Experiment.java

```
package com.ferrol;

import javax.swing.*;

public class Experiment
{
    private static float tunnelwidth = 150.0f;
    private static int tunnel_type = 2;

    public Experiment()
    {
    }

    private static void Usage()
    {
     System.out.println("Usage: java com.ferrol.Experiment \\
                         [type_of_tunnel [width_of_tunnel]]\n");
     System.out.println("\ttype_of_tunnel : <1=circular | 2=straight>\n");
     System.out.println("\twidth_of_tunnel : <20.0..150.0>");
    }

    public static void main(String[] args)
    {
     Usage();

     try
     {
     if (args.length==1 || args.length==2)
     // assume we are just setting the tunnel type
     {
     tunnel_type = Integer.parseInt( args[0] );
     }
     if (args.length==2)
     // assume we are setting the tunnel width as well
     {
     tunnelwidth = Float.parseFloat( args[1] );
     }
     }
     catch (NumberFormatException e)
     {
     System.out.println("Exiting.."+ e.toString());
         return;
     }

        try
        {
            UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
        } catch (UnsupportedLookAndFeelException ex)
        {
            ex.printStackTrace();
        } catch (IllegalAccessException ex)
        {
            ex.printStackTrace();
        } catch (InstantiationException ex)
        {
            ex.printStackTrace();
        } catch (ClassNotFoundException ex)
        {
            ex.printStackTrace();
        }
        UIManager.put("swing.boldMetal", Boolean.FALSE);
        javax.swing.SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                showGUI();
            }
        });
    }

    private static void showGUI()
    {
        // Make a frame
        JFrame frame = new ExperimentFrame("Assistive Interface Experiment",
                                        tunnel_type, tunnelwidth);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Show the frame
        frame.pack();
        frame.setVisible(true);
    }
}
```

C.2.2 ExperimentFrame.java

```
package com.ferrol;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import java.awt.*;
import java.awt.event.*;

public class ExperimentFrame extends JFrame
{
  static final long serialVersionUID = 12347;
  private Dimension dim;
  private int tunnel_type;
  private float tunnelwidth;

  public ExperimentFrame( String title, int t_type, float t_width )
  {
  tunnel_type = t_type;
  tunnelwidth = t_width;

      setTitle(title);

      dim = Toolkit.getDefaultToolkit().getScreenSize();
      dim.height = dim.height - 25;
      this.setPreferredSize(dim);

      panel = new ExperimentPanel( dim.width, dim.height, tunnel_type, tunnelwidth );
      getContentPane().add(panel, BorderLayout.CENTER);
      JPanel buttonPanel = new JPanel();

      addButton(buttonPanel, "Close", new ActionListener()
      {
        public void actionPerformed(ActionEvent event)
        {
          System.exit(0);
        }
      });
      getContentPane().add(buttonPanel, BorderLayout.SOUTH);

  }

  public void addButton(Container c, String title, ActionListener listener)
  {
    JButton button = new JButton(title);
    c.add(button);
    button.addActionListener(listener);
  }
  private JPanel panel;

}
```

C.2.3 ExperimentPanel.java

```
package com.ferrol;

import javax.swing.JPanel;
import java.awt.BasicStroke;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Shape;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.awt.Color;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.Math;

public class ExperimentPanel extends JPanel implements MouseMotionListener
{
    private Shape circle;
    private Shape inner_circle;
    private final float outsideborder = 100.0f;

    private float tunnelwidth;
    private int tunnel_type;

    private float x;
    private float y;
    private float circleheight;
    private float circlewidth;

    // rect settings
    private float rect_x;
    private float rect_y;
    private float rect_height;
    private float rect_width;
    private Shape rect;

    private FileWriter m_output_file;

static final long serialVersionUID = 12345;
private enum eState {START, IN_TUNNEL_BEFORE, IN_TUNNEL_DURING,
                         OUT_TUNNEL_DURING, OUT_TUNNEL_BEFORE, DONE};

private eState m_state;

private int prev_x = 0;
private int prev_y = 0;
private double total_distance = 0.0;

    public ExperimentPanel(int width, int height, int t_type, float t_width)
    {
     tunnel_type = t_type;
     tunnelwidth = t_width;

     circleheight = (float)height - (2.0f*outsideborder);
     circlewidth = circleheight;
     x = ((float)width - circlewidth)/2.0f;
     y = outsideborder;

     circle = new Ellipse2D.Float(x, y, circlewidth, circleheight);
     inner_circle = new Ellipse2D.Float( x+tunnelwidth,
                 y+tunnelwidth,
                 circleheight-(tunnelwidth*2.0f),
                 circleheight-(tunnelwidth*2.0f));

     rect_x = outsideborder*2.0f;
     rect_y = (float)height/2.0f-(tunnelwidth/2.0f);
     rect_width = width-(4.0f*outsideborder);
     rect_height = tunnelwidth;

     rect = new Rectangle2D.Float(rect_x, rect_y, rect_width, rect_height);

     addMouseMotionListener(this);

     m_state = eState.START;
    }

    void eventOutput(String eventDescription, MouseEvent e)
    {
    }
    public void mousePressed(MouseEvent e)
    {
        eventOutput("Mouse pressed (# of clicks: "
                + e.getClickCount() + ")", e);
    }

    public void mouseReleased(MouseEvent e)
    {
        eventOutput("Mouse released (# of clicks: "
                + e.getClickCount() + ")", e);
    }

    public void mouseEntered(MouseEvent e)
    {
        eventOutput("Mouse entered", e);
    }

    public void mouseExited(MouseEvent e)
    {
        eventOutput("Mouse exited", e);
    }

    public void mouseClicked(MouseEvent e)
    {
        eventOutput("Mouse clicked (# of clicks: "
                + e.getClickCount() + ")", e);
    }

@Override
public void mouseDragged(MouseEvent e) {

}

private boolean inCircularTunnel(int ix, int iy)
{
double center_x = x+(circlewidth/2.0f);
    double center_y = y+(circleheight/2.0f);
    double radius = circleheight/2.0f;
//     System.out.println("ix: " + ix + " iy: " + iy + " center_x: " + center_x
/                            + " center_y: " + center_y + " radius: "+ radius);
    double term1 = ( Math.pow((center_x - ix), 2.0));
    double term2 = ( Math.pow((center_y - iy), 2.0));
    double outter_radius2 = Math.pow(radius, 2.0);
    double inner_radius2 = Math.pow(radius-tunnelwidth, 2.0);
//     System.out.println("\tterm1: " + term1 + " term2: " + term2 + " r^2: " + r2);
if ( ((term1+term2) < outter_radius2) &&
!((term1+term2) < inner_radius2)
   )
return true;
else
return false;
}

private boolean cursorCrossesLine(int ix, int iy)
{
if ( (ix > (int)(x+(circlewidth/2.0f))) && (ix <=
             (int)(x+(circlewidth/2.0f))+ 20) &&
 (iy <= (int)(y+tunnelwidth)) &&
 (iy >= (int)(y))
   )
return true;
else
return false;
}

private boolean goneThroughTunnel(int ix, int iy )
{
if (total_distance >= (6.28 * (circleheight-(tunnelwidth*2.0f))/2.0f ))
return true;
```

```
else
return false;
}
@Override
public void mouseMoved(MouseEvent e)
{
if (1==tunnel_type)
{
mouseMovedCircular(e);
}
else
{
mouseMovedStraight(e);
}
}

    public void paint(Graphics g)
    {
     if (2==tunnel_type)
     {
     paintStraightTunnel(g);
     }
     else
     {
     paintCircularTunnel(g);
     }
    }

    private void openFile()
    {
     try
     {
     int cnt = 0;
     File f = new File("Data"+cnt+".txt");
     while ( f.exists() )
     {
     cnt++;
     f = new File("Data"+cnt+".txt");
     }
     m_output_file = new FileWriter("Data"+cnt+".txt");
     }
     catch (IOException ex)
     {
     System.out.println(ex);
     }
    }

    private void myWrite(String str)
    {
     try
     {
     m_output_file.write(str);
     }
     catch(IOException ex)
     {
     System.out.println(ex);
     }
    }

    // =================================================================
    // Circular Tunnel specific code
    //==================================================================
    private void mouseMovedCircular(MouseEvent e)
    {
int ix = e.getX();
int iy = e.getY();
if ( inCircularTunnel(ix, iy) )
{
switch(m_state)
{
```

```
case START:
m_state = eState.IN_TUNNEL_BEFORE;
     total_distance = 0.0;
     break;
case OUT_TUNNEL_DURING:
m_state = eState.IN_TUNNEL_DURING;
     myWrite(System.currentTimeMillis() + ", " +
                       "OUT_TUNNEL_DURING->IN_TUNNEL_DURING, " + ix + ", "+ iy + "\r\n");
break;
case OUT_TUNNEL_BEFORE:
m_state = eState.IN_TUNNEL_BEFORE;
break;
case IN_TUNNEL_BEFORE:
     total_distance = 0.0;
     if (cursorCrossesLine(ix, iy))
     {
     openFile();
     myWrite(System.currentTimeMillis() + ", " +
                           "IN_TUNNEL_BEFORE->IN_TUNNEL_DURING, " + ix + ", "+ iy + "\r\n");

m_state = eState.IN_TUNNEL_DURING;
     }
break;
case IN_TUNNEL_DURING:
if (cursorCrossesLine(ix, iy) && goneThroughTunnel(ix, iy))
{
     myWrite(System.currentTimeMillis() + ", " +
                           "IN_TUNNEL_DURING->DONE, " + ix + ", "+ iy + "\r\n");
     try { m_output_file.flush(); m_output_file.close(); }
                    catch(IOException ex) {System.out.println(ex);}
m_state = eState.DONE;
}
break;
case DONE:
    default:
     total_distance = 0.0;
     break;

}
}
else
{
switch(m_state)
{
case IN_TUNNEL_BEFORE:
m_state = eState.OUT_TUNNEL_BEFORE;
break;
case IN_TUNNEL_DURING:
// record tunnel crossing
     myWrite(System.currentTimeMillis() + ", " +
                       "IN_TUNNEL_DURING->OUT_TUNNEL_DURING, " + ix + ", "+ iy + "\r\n");
m_state = eState.OUT_TUNNEL_DURING;
break;
case OUT_TUNNEL_DURING:
break;
    default:
     break;
}
}

        eventOutput("Mouse moved", e);

        repaint();

double distance = Math.sqrt(Math.pow((ix-prev_x), 2.0) +
                          Math.pow((iy - prev_y), 2.0) );
total_distance += distance;
        prev_x = ix;
        prev_y = iy;
    }
    private void paintCircularTunnel(Graphics g)
    {
        Graphics2D ga = (Graphics2D)g;
        ga.setStroke(new BasicStroke(3));
        ga.setColor(Color.black);

        ga.draw(circle);

        switch (m_state)
        {
        case START:
         ga.setPaint(Color.yellow.darker());
         break;
        case IN_TUNNEL_BEFORE:
         ga.setPaint(Color.yellow);
         break;
        case IN_TUNNEL_DURING:
        case OUT_TUNNEL_DURING:
         ga.setPaint(Color.magenta);
         break;
        case OUT_TUNNEL_BEFORE:
         ga.setPaint(Color.blue);
         break;
        case DONE:
         ga.setPaint(Color.CYAN);
         break;
        default:
         ga.setPaint(Color.yellow.darker());
         break;
        }
        ga.fill(circle);

        ga.setColor(Color.black);
        ga.draw(inner_circle);
        ga.setPaint(Color.lightGray.brighter());
        ga.fill(inner_circle);

        ga.setColor(Color.black);
        ga.setStroke(new BasicStroke(6));
        ga.drawLine((int)(x+(circlewidth/2.0f)), (int)y, (int)(x+(circlewidth/2.0f)),
                    (int)(y+tunnelwidth));

    }
    //=================================================================
    // Straight Tunnel specific code
    //=================================================================
private boolean goneThroughStraightTunnel(int ix, int iy)
{
if ( (total_distance >= rect_width ) && (ix >= (rect_x + rect_width)))
return true;
else
return false;
}
private boolean cursorCrossedStartLine(int ix, int iy)
{
if ( (ix >= rect_x) && (ix<=rect_x+20) && (iy>=rect_y && iy<=(rect_y+rect_height)))
return true;
else
return false;
}

    private boolean inStraightTunnel(int ix, int iy)
    {
    if (ix>=rect_x && ix<=(rect_x+rect_width) &&
        iy>=rect_y && iy<=(rect_y+rect_height))
return true;
else
return false;
}
```

```
    private void mouseMovedStraight(MouseEvent e)
    {
int ix = e.getX();
int iy = e.getY();
//System.out.println(" ix: "+ ix + " iy: " + iy + " rect_x: " + rect_x + " rect_y: "
//                     + rect_y + " rect_width: " + rect_width + " rect_height: " + rect_height );
if ( inStraightTunnel(ix, iy) )
{
switch(m_state)
{
case START:
m_state = eState.IN_TUNNEL_BEFORE;
     total_distance = 0.0;
     break;
case OUT_TUNNEL_DURING:
m_state = eState.IN_TUNNEL_DURING;
     myWrite(System.currentTimeMillis() + ", " +
                       "OUT_TUNNEL_DURING->IN_TUNNEL_DURING, " + ix + ", "+ iy + "\r\n");
break;
case OUT_TUNNEL_BEFORE:
     m_state = eState.IN_TUNNEL_BEFORE;
break;
case IN_TUNNEL_BEFORE:
     total_distance = 0.0;
     if (cursorCrossedStartLine(ix, iy))
     {
     openFile();
     myWrite(System.currentTimeMillis() + ", " +
                           "IN_TUNNEL_BEFORE->IN_TUNNEL_DURING, " + ix + ", "+ iy + "\r\n");
m_state = eState.IN_TUNNEL_DURING;
     }
break;

case IN_TUNNEL_DURING:
// write data to file
if ( goneThroughStraightTunnel(ix, iy) )
{
// close data file
     myWrite(System.currentTimeMillis() + ", " +
                           "IN_TUNNEL_BEFORE->DONE, " + ix + ", "+ iy + "\r\n");
     try { m_output_file.flush(); m_output_file.close(); }
                    catch(IOException ex) {System.out.println(ex);}
m_state = eState.DONE;
}
break;
case DONE:
    default:
     total_distance = 0.0;
     break;
}
}
else
{
switch(m_state)
{
case IN_TUNNEL_BEFORE:
m_state = eState.OUT_TUNNEL_BEFORE;
break;
case IN_TUNNEL_DURING:
// record tunnel crossing
if (goneThroughStraightTunnel(ix, iy))
{
     myWrite(System.currentTimeMillis() + ", " +
                           "IN_TUNNEL_DURING->DONE, " + ix + ", "+ iy + "\r\n");
     try { m_output_file.flush(); m_output_file.close(); }
                    catch(IOException ex) {System.out.println(ex);}
m_state = eState.DONE;
}
else
{
     myWrite(System.currentTimeMillis() + ", " +
                           "IN_TUNNEL_DURING->OUT_TUNNEL_DURING, " + ix + ", "+ iy + "\r\n");
m_state = eState.OUT_TUNNEL_DURING;
}
break;
case OUT_TUNNEL_DURING:
if (goneThroughStraightTunnel(ix, iy) && iy>=rect_y && iy<=(rect_y+rect_height))
{
     myWrite(System.currentTimeMillis() + ", " +
                           "IN_TUNNEL_DURING->DONE, " + ix + ", "+ iy + "\r\n");
     try { m_output_file.flush(); m_output_file.close(); }
                   catch(IOException ex) {System.out.println(ex);}
m_state = eState.DONE;
}
break;
case DONE:
    default:
     total_distance = 0.0;
     break;
}
}

        eventOutput("Mouse moved", e);

        repaint();

double distance = Math.sqrt(Math.pow((ix-prev_x), 2.0) + Math.pow((iy - prev_y), 2.0) );
total_distance += distance;
        prev_x = ix;
        prev_y = iy;

    }
    private void paintStraightTunnel(Graphics g)
    {
        Graphics2D ga = (Graphics2D)g;
        ga.setStroke(new BasicStroke(3));
        ga.setColor(Color.black);

     ga.draw(rect);
        switch (m_state)
        {
        case START:
         ga.setPaint(Color.yellow.darker());
         break;
        case IN_TUNNEL_BEFORE:
         ga.setPaint(Color.yellow);
         break;
        case IN_TUNNEL_DURING:
        case OUT_TUNNEL_DURING:
         ga.setPaint(Color.magenta);
         break;
        case OUT_TUNNEL_BEFORE:
         ga.setPaint(Color.blue);
         break;
        case DONE:
         ga.setPaint(Color.CYAN);
         break;
        default:
         ga.setPaint(Color.yellow.darker());
         break;
        }

     ga.fill(rect);

        ga.setColor(Color.black);
        ga.setStroke(new BasicStroke(6));
        ga.drawLine((int)rect_x, (int)rect_y, (int)(rect_x), (int)(rect_y+rect_height));
        ga.drawLine((int)(rect_x+rect_width), (int)rect_y,
        (int)(rect_x+rect_width), (int)(rect_y+rect_height));
    }
```

```
}
```

APPENDIX D: SUPPORT CODE

Helper code, like parsing and testing code, is contained in this section of the Appendix.

D.1 MATLAB data analysis

D.1.1 GetTrialTime.m

```
% GetTrialTime
%     char str1[] =  "IN_TUNNEL_BEFORE->IN_TUNNEL_DURING";
%     This was mislabeled.  Should have been IN_TUNNEL_DURING->DONE.
%     char str2[] =  "IN_TUNNEL_BEFORE->DONE";
%     char str3[] =  "IN_TUNNEL_DURING->DONE";
%     char str4[] =  "OUT_TUNNEL_DURING->IN_TUNNEL_DURING";
%     char str5[] =  "IN_TUNNEL_DURING->OUT_TUNNEL_DURING";
%     char str6[] =  "MOVE";
function [results] = GetTrialTime(str_device_type, str_trial_type)

trial_results   = zeros(5,4);
files = dir('.\preprocessed_data\*.out');
for i = 1:size(files,1)
    filename = files(i).name;
s = filename;
    M = load( strcat('.\preprocessed_data\',filename) );
    endstate = M(size(M,1), 2);
    if ( (endstate ~= 3) && (endstate ~= 2) )
        disp( sprintf('Error: in file: %s', filename) );
    end
idxs = strfind(s, '_');
if (size(idxs,2)==4)
        id          = s(1:idxs(1)-1);
        device_type = s(idxs(1)+1:idxs(2)-1);
        group       = s(idxs(2)+1:idxs(3)-1);
        trial_type  = s(idxs(3)+1:idxs(4)-1);
%         disp(sprintf('TRAINING: %s, %s, %s, %s', id, device_type, group, trial_type ));
        if ( ~strcmp( s(idxs(4)+1:idxs(4)+5), 'train') )
            disp(sprintf('ERROR - wrong number of underscores in filename.'));
        end
elseif (size(idxs,2)==3)
        periods = strfind(s, '.');
        id          = s(1:idxs(1)-1);
        device_type = s(idxs(1)+1:idxs(2)-1);
        group       = s(idxs(2)+1:idxs(3)-1);
        trial_type  = s(idxs(3)+1:periods(1)-1);

        if ( strcmp(device_type, str_device_type) && strcmp(trial_type, str_trial_type) )
            % calculate number of errors for this trial
            count_errors = 0.0;
            for row = 1:size(M,1)
                if (M(row,2) == 5)
                    count_errors = count_errors + 1.0;
                end
            end
            % calculate time for this trial
            secs = ( M(size(M,1), 1) - M(1, 1) ) / 1000;

            % split into groups
            if (strcmp(group,'1'))
```

```
                trial_results(3,1) = trial_results(3,1) + count_errors;   % accumulate errors
                trial_results(4,1) = trial_results(4,1) + secs;           % accumulate time
                trial_results(5,1) = trial_results(5,1) + 1;              % count
            elseif (strcmp(group,'2'))
                trial_results(3,2) = trial_results(3,2) + count_errors;   % accumulate errors
                trial_results(4,2) = trial_results(4,2) + secs;           % accumulate time
                trial_results(5,2) = trial_results(5,2) + 1;              % count
            elseif (strcmp(group,'3'))
                trial_results(3,3) = trial_results(3,3) + count_errors;   % accumulate errors
                trial_results(4,3) = trial_results(4,3) + secs;           % accumulate time
                trial_results(5,3) = trial_results(5,3) + 1;              % count
            elseif (strcmp(group,'4'))
                trial_results(3,4) = trial_results(3,4) + count_errors;   % accumulate errors
                trial_results(4,4) = trial_results(4,4) + secs;           % accumulate time
                trial_results(5,4) = trial_results(5,4) + 1;              % count
            else
                disp(sprintf('ERROR - invalid group ID.'));
            end
%             disp(sprintf('No training: %s, %s, %s, %s', id, device_type, group, trial_type ));
            disp(sprintf('%40s : %5.0f seconds, %3d errors', filename, secs, count_errors));
        end
else
        disp(sprintf('ERROR - wrong number of underscores in filename.'));
end
end
for j=1:4
    trial_results(1,j) = trial_results(4,j)/trial_results(5,j);  % store avg secs.
    trial_results(2,j) = trial_results(3,j)/trial_results(5,j);  % store avg errors
end

trial_results_plot = trial_results(1:2, :);
trial_results_plot = trial_results_plot';
results = trial_results_plot;
```

D.1.2 PlotAllTrials.m

```
[results_plot] = GetTrialTime('standard', 'straight');
figure;
bar(results_plot);
title('Standard (USB) mouse with straight target');
xlabel('Group number');
ylabel('Seconds or number of errors');
legend('Seconds to complete trial','Number of errors during trial');

[results_plot] = GetTrialTime('standard', 'circle');
figure;
bar(results_plot);
title('Standard (USB) mouse with circular target');
xlabel('Group number');
ylabel('Seconds or number of errors');
legend('Seconds to complete trial','Number of errors during trial');

[results_plot] = GetTrialTime('head', 'straight');
figure;
bar(results_plot);
title('Head tilt mouse with straight target');
xlabel('Group number');
ylabel('Seconds or number of errors');
legend('Seconds to complete trial','Number of errors during trial');

[results_plot] = GetTrialTime('head', 'circle');
figure;
bar(results_plot);
title('Head tilt mouse with circular target');
xlabel('Group number');
ylabel('Seconds or number of errors');
legend('Seconds to complete trial','Number of errors during trial');
```

D.1.3 my_ttest.m

```
function [result] = my_ttest(x, mu)

temp = mean( x(2,:) - mu(2,:) );
s = std( x(2,:) - mu(2,:) );

result = temp/(s/sqrt(size(mu,2)));
```

D.1.4 my_ttest_for_errors.m

```
function [result] = my_ttest_for_errors(x, mu)

temp = mean( x(1,:) - mu(1,:) );
s = std( x(1,:) - mu(1,:) );

result = temp/(s/sqrt(size(mu,2)));
```

D.1.5 ttestsForErrors.m

```
% Groups
% 1 - train w/alg.
% 2 - train NO alg.
% 3 - w/alg.
% 4 - NO alg.

% diff. btw. training data and trial data for group 1
disp(sprintf('diff. in train. standard/straight 1:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '1');
t = my_ttest_for_errors(x,mu)

disp(sprintf('diff. in train. standard/circle 1:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '1');
t = my_ttest_for_errors(x,mu)

disp(sprintf('diff. in train. head/straight 1:\n'));
x = GetTrialTimePerGroupForTraining('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '1');
t = my_ttest_for_errors(x,mu)

disp(sprintf('diff. in train. head/circle 1:\n'));
x = GetTrialTimePerGroupForTraining('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '1');
t = my_ttest_for_errors(x,mu)

% diff. btw. training data and trial data for group 2
disp(sprintf('diff. in train. standard/straight 2:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'straight', '2');
mu = GetTrialTimePerGroup('standard', 'straight', '2');
t = my_ttest_for_errors(x,mu)

disp(sprintf('diff. in train. standard/circle 2:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '2');
t = my_ttest_for_errors(x,mu)

disp(sprintf('diff. in train. head/straight 2:\n'));
x = GetTrialTimePerGroupForTraining('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '2');
t = my_ttest_for_errors(x,mu)

disp(sprintf('diff. in train. head/circle 2:\n'));
```

```
x = GetTrialTimePerGroupForTraining('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '2');
t = my_ttest_for_errors(x,mu)

% diff. btw. w/alg. and NO alg. (3 and 4)
disp(sprintf('standard/straight 3/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '3');
mu = GetTrialTimePerGroup('standard', 'straight', '4');
t = my_ttest_for_errors(x,mu)

disp(sprintf('standard/circle 3/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '3');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/straight 3/4:\n'));
x = GetTrialTimePerGroup('head', 'straight', '3');
mu = GetTrialTimePerGroup('head', 'straight', '4');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/circle 3/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '3');
mu = GetTrialTimePerGroup('head', 'circle', '4');
t = my_ttest_for_errors(x,mu)

% diff. btw. w/alg. and NO alg. with training (1 and 2)
disp(sprintf('standard/straight 1/2:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '2');
t = my_ttest_for_errors(x,mu)

disp(sprintf('standard/circle 1/2:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '2');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/straight 1/2:\n'));
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '2');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/circle 1/2:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '2');
t = my_ttest_for_errors(x,mu)

% diff. btw. trained group w/alg. and untrained w/alg. (1 and 3)
disp(sprintf('standard/straight 1/3:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '3');
t = my_ttest_for_errors(x,mu)

disp(sprintf('standard/circle 1/3:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '3');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/straight 1/3:\n'));
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '3');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/circle 1/3:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '3');
t = my_ttest_for_errors(x,mu)

% diff. btw. trained group NO alg. and untrained NO alg. (2 and 4)
disp(sprintf('standard/straight 2/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '2');
```

```
mu = GetTrialTimePerGroup('standard', 'straight', '4');
t = my_ttest_for_errors(x,mu)

disp(sprintf('standard/circle 2/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/straight 2/4:\n'));
x = GetTrialTimePerGroup('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '4');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/circle 2/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '4');
t = my_ttest_for_errors(x,mu)

% diff. btw. trained group w/alg. and untrained NO alg. (1 and 4)
disp(sprintf('standard/straight 1/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '4');
t = my_ttest_for_errors(x,mu)

disp(sprintf('standard/circle 1/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/straight 1/4:\n'));
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '4');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/circle 1/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '4');
t = my_ttest_for_errors(x,mu)

% diff. btw. trained group NO alg. and untrained w/alg. (2 and 3)
disp(sprintf('standard/straight 2/3:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '2');
mu = GetTrialTimePerGroup('standard', 'straight', '3');
t = my_ttest_for_errors(x,mu)

disp(sprintf('standard/circle 2/3:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '3');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/straight 2/3:\n'));
x = GetTrialTimePerGroup('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '3');
t = my_ttest_for_errors(x,mu)

disp(sprintf('head/circle 2/3:\n'));
x = GetTrialTimePerGroup('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '3');
t = my_ttest_for_errors(x,mu)
```

D.1.6 ttestsForTime.m

```
% Groups
% 1 - train w/alg.
% 2 - train NO alg.
% 3 - w/alg.
% 4 - NO alg.

% diff. btw. training data and trial data for group 1
disp(sprintf('diff. in train. standard/straight 1:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '1');
t = my_ttest(x,mu)

disp(sprintf('diff. in train. standard/circle 1:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '1');
t = my_ttest(x,mu)

disp(sprintf('diff. in train. head/straight 1:\n'));
x = GetTrialTimePerGroupForTraining('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '1');
t = my_ttest(x,mu)

disp(sprintf('diff. in train. head/circle 1:\n'));
x = GetTrialTimePerGroupForTraining('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '1');
t = my_ttest(x,mu)

% diff. btw. training data and trial data for group 2
disp(sprintf('diff. in train. standard/straight 2:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'straight', '2');
mu = GetTrialTimePerGroup('standard', 'straight', '2');
t = my_ttest(x,mu)

disp(sprintf('diff. in train. standard/circle 2:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '2');
t = my_ttest(x,mu)

disp(sprintf('diff. in train. head/straight 2:\n'));
x = GetTrialTimePerGroupForTraining('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '2');
t = my_ttest(x,mu)

disp(sprintf('diff. in train. head/circle 2:\n'));
x = GetTrialTimePerGroupForTraining('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '2');
t = my_ttest(x,mu)

% diff. btw. w/alg. and NO alg. (3 and 4)
disp(sprintf('standard/straight 3/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '3');
mu = GetTrialTimePerGroup('standard', 'straight', '4');
t = my_ttest(x,mu)

disp(sprintf('standard/circle 3/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '3');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
t = my_ttest(x,mu)

disp(sprintf('head/straight 3/4:\n'));
x = GetTrialTimePerGroup('head', 'straight', '3');
mu = GetTrialTimePerGroup('head', 'straight', '4');
t = my_ttest(x,mu)

disp(sprintf('head/circle 3/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '3');
mu = GetTrialTimePerGroup('head', 'circle', '4');
t = my_ttest(x,mu)

% diff. btw. w/alg. and NO alg. with training (1 and 2)
disp(sprintf('standard/straight 1/2:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '2');
t = my_ttest(x,mu)
```

```
disp(sprintf('standard/circle 1/2:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '2');
t = my_ttest(x,mu)

disp(sprintf('head/straight 1/2:\n'));
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '2');
t = my_ttest(x,mu)

disp(sprintf('head/circle 1/2:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '2');
t = my_ttest(x,mu)

% diff. btw. trained group w/alg. and untrained w/alg. (1 and 3)
disp(sprintf('standard/straight 1/3:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '3');
t = my_ttest(x,mu)

disp(sprintf('standard/circle 1/3:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '3');
t = my_ttest(x,mu)

disp(sprintf('head/straight 1/3:\n'));
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '3');
t = my_ttest(x,mu)

disp(sprintf('head/circle 1/3:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '3');
t = my_ttest(x,mu)

% diff. btw. trained group NO alg. and untrained NO alg. (2 and 4)
disp(sprintf('standard/straight 2/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '2');
mu = GetTrialTimePerGroup('standard', 'straight', '4');
t = my_ttest(x,mu)

disp(sprintf('standard/circle 2/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
t = my_ttest(x,mu)

disp(sprintf('head/straight 2/4:\n'));
x = GetTrialTimePerGroup('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '4');
t = my_ttest(x,mu)

disp(sprintf('head/circle 2/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '4');
t = my_ttest(x,mu)

% diff. btw. trained group w/alg. and untrained NO alg. (1 and 4)
disp(sprintf('standard/straight 1/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '4');
t = my_ttest(x,mu)

disp(sprintf('standard/circle 1/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
t = my_ttest(x,mu)

disp(sprintf('head/straight 1/4:\n'));
```

```
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '4');
t = my_ttest(x,mu)

disp(sprintf('head/circle 1/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '4');
t = my_ttest(x,mu)

% diff. btw. trained group NO alg. and untrained w/alg. (2 and 3)
disp(sprintf('standard/straight 2/3:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '2');
mu = GetTrialTimePerGroup('standard', 'straight', '3');
t = my_ttest(x,mu)

disp(sprintf('standard/circle 2/3:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '3');
t = my_ttest(x,mu)

disp(sprintf('head/straight 2/3:\n'));
x = GetTrialTimePerGroup('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '3');
t = my_ttest(x,mu)

disp(sprintf('head/circle 2/3:\n'));
x = GetTrialTimePerGroup('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '3');
t = my_ttest(x,mu)
```

D.1.7 GetTrialTimePerGroup.m

```
% GetTrialTimePerGroup
%     char str1[] =  "IN_TUNNEL_BEFORE->IN_TUNNEL_DURING";
%     This was mislabeled.  Should have been IN_TUNNEL_DURING->DONE.
%     char str2[] =  "IN_TUNNEL_BEFORE->DONE";
%     char str3[] =  "IN_TUNNEL_DURING->DONE";
%     char str4[] =  "OUT_TUNNEL_DURING->IN_TUNNEL_DURING";
%     char str5[] =  "IN_TUNNEL_DURING->OUT_TUNNEL_DURING";
%     char str6[] =  "MOVE";
function [results] = GetTrialTimePerGroup(str_device_type, str_trial_type, str_group)

group_cnt = 0;
files = dir('.\preprocessed_data\*.out');
for i = 1:size(files,1)
    filename = files(i).name;
s = filename;
    M = load( strcat('.\preprocessed_data\',filename) );
    endstate = M(size(M,1), 2);
    if ( (endstate ~= 3) && (endstate ~= 2) )
        disp( sprintf('Error: in file: %s', filename) );
    end
idxs = strfind(s, '_');
if (size(idxs,2)==4)
        id          = s(1:idxs(1)-1);
        device_type = s(idxs(1)+1:idxs(2)-1);
        group       = s(idxs(2)+1:idxs(3)-1);
        trial_type  = s(idxs(3)+1:idxs(4)-1);
%         disp(sprintf('TRAINING: %s, %s, %s, %s', id, device_type, group, trial_type ));
        if ( ~strcmp( s(idxs(4)+1:idxs(4)+5), 'train') )
            disp(sprintf('ERROR - wrong number of underscores in filename.'));
        end
elseif (size(idxs,2)==3)
        periods = strfind(s, '.');
        id          = s(1:idxs(1)-1);
        device_type = s(idxs(1)+1:idxs(2)-1);
        group       = s(idxs(2)+1:idxs(3)-1);
        trial_type  = s(idxs(3)+1:periods(1)-1);

        if ( strcmp(device_type, str_device_type) &&
             strcmp(trial_type, str_trial_type) && strcmp(group, str_group))
            group_cnt = group_cnt + 1;
            % calculate number of errors for this trial
            count_errors = 0.0;
            for row = 1:size(M,1)
                if (M(row,2) == 5)
                    count_errors = count_errors + 1.0;
                end
            end
            % calculate time for this trial
            secs = ( M(size(M,1), 1) - M(1, 1) ) / 1000;

            T(1,group_cnt) = count_errors;
            T(2,group_cnt) = secs;

        end
else
        disp(sprintf('ERROR - wrong number of underscores in filename.'));
end
end
results = T;
```

D.1.8 GetTrialTimePerGroupForTraining.m

```
% GetTrialTimePerGroupForTraining
%     char str1[] =  "IN_TUNNEL_BEFORE->IN_TUNNEL_DURING";
%     This was mislabeled.  Should have been IN_TUNNEL_DURING->DONE.
%     char str2[] =  "IN_TUNNEL_BEFORE->DONE";
%     char str3[] =  "IN_TUNNEL_DURING->DONE";
%     char str4[] =  "OUT_TUNNEL_DURING->IN_TUNNEL_DURING";
%     char str5[] =  "IN_TUNNEL_DURING->OUT_TUNNEL_DURING";
%     char str6[] =  "MOVE";
function [results] = GetTrialTimePerGroupForTraining(str_device_type,
                         str_trial_type, str_group)

group_cnt = 0;
files = dir('.\preprocessed_data\*.out');
for i = 1:size(files,1)
    filename = files(i).name;
s = filename;
    M = load( strcat('.\preprocessed_data\',filename) );
    endstate = M(size(M,1), 2);
    if ( (endstate ~= 3) && (endstate ~= 2) )
        disp( sprintf('Error: in file: %s', filename) );
    end
idxs = strfind(s, '_');
if (size(idxs,2)==4)
        id          = s(1:idxs(1)-1);
        device_type = s(idxs(1)+1:idxs(2)-1);
        group       = s(idxs(2)+1:idxs(3)-1);
        trial_type  = s(idxs(3)+1:idxs(4)-1);
        if ( ~strcmp( s(idxs(4)+1:idxs(4)+5), 'train') )
            disp(sprintf('ERROR - training file reported as non-training file.'));
        end
        if ( strcmp(device_type, str_device_type) &&
             strcmp(trial_type, str_trial_type) && strcmp(group, str_group))
            group_cnt = group_cnt + 1;
            % calculate number of errors for this trial
            count_errors = 0.0;
            for row = 1:size(M,1)
                if (M(row,2) == 5)
                count_errors = count_errors + 1.0;
            end
        end
        % calculate time for this trial
        secs = ( M(size(M,1), 1) - M(1, 1) ) / 1000;

        T(1,group_cnt) = count_errors;
        T(2,group_cnt) = secs;
    end

elseif (size(idxs,2)==3)


else
        disp(sprintf('ERROR - wrong number of underscores in filename.'));
end
end
results = T;
```

D.1.9 GetErrorsPerGroup.m

```
% GetErrorsPerGroup
%     char str1[] =  "IN_TUNNEL_BEFORE->IN_TUNNEL_DURING";
%     This was mislabeled.  Should have been IN_TUNNEL_DURING->DONE.
%     char str2[] =  "IN_TUNNEL_BEFORE->DONE";
%     char str3[] =  "IN_TUNNEL_DURING->DONE";
%     char str4[] =  "OUT_TUNNEL_DURING->IN_TUNNEL_DURING";
%     char str5[] =  "IN_TUNNEL_DURING->OUT_TUNNEL_DURING";
%     char str6[] =  "MOVE";
function [results] = GetErrorsPerGroup(str_device_type, str_trial_type, str_group)

group_cnt = 0;
files = dir('.\preprocessed_data\*.out');
for i = 1:size(files,1)
    filename = files(i).name;
s = filename;
    M = load( strcat('.\preprocessed_data\',filename) );
    endstate = M(size(M,1), 2);
    if ( (endstate ~= 3) && (endstate ~= 2) )
        disp( sprintf('Error: in file: %s', filename) );
    end
idxs = strfind(s, '_');
if (size(idxs,2)==4)
        id          = s(1:idxs(1)-1);
        device_type = s(idxs(1)+1:idxs(2)-1);
        group       = s(idxs(2)+1:idxs(3)-1);
        trial_type  = s(idxs(3)+1:idxs(4)-1);
%         disp(sprintf('TRAINING: %s, %s, %s, %s', id, device_type, group, trial_type ));
        if ( ~strcmp( s(idxs(4)+1:idxs(4)+5), 'train') )
            disp(sprintf('ERROR - wrong number of underscores in filename.'));
        end
elseif (size(idxs,2)==3)
        periods = strfind(s, '.');
        id          = s(1:idxs(1)-1);
        device_type = s(idxs(1)+1:idxs(2)-1);
        group       = s(idxs(2)+1:idxs(3)-1);
        trial_type  = s(idxs(3)+1:periods(1)-1);

        if ( strcmp(device_type, str_device_type) &&
             strcmp(trial_type, str_trial_type) && strcmp(group, str_group))
            group_cnt = group_cnt + 1;
            % calculate number of errors for this trial
            count_errors = 0.0;
            for row = 1:size(M,1)
                if (M(row,2) == 5)
                    count_errors = count_errors + 1.0;
                end
            end
            % calculate time for this trial
            secs = ( M(size(M,1), 1) - M(1, 1) ) / 1000;

            T(1,group_cnt) = count_errors;
            T(2,group_cnt) = secs;
        end
else
        disp(sprintf('ERROR - wrong number of underscores in filename.'));
end
end
results = T;
```

D.1.10 bonf_time.m

```
pairs = [1 2];

% diff. btw. training data and trial data for group 1
disp(sprintf('diff. in train. standard/straight 1:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '1');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('diff. in train. standard/circle 1:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '1');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('diff. in train. head/straight 1:\n'));
x = GetTrialTimePerGroupForTraining('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '1');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('diff. in train. head/circle 1:\n'));
x = GetTrialTimePerGroupForTraining('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '1');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. training data and trial data for group 2
disp(sprintf('diff. in train. standard/straight 2:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'straight', '2');
mu = GetTrialTimePerGroup('standard', 'straight', '2');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('diff. in train. standard/circle 2:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '2');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));
disp(sprintf('diff. in train. head/straight 2:\n'));
x = GetTrialTimePerGroupForTraining('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '2');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('diff. in train. head/circle 2:\n'));
x = GetTrialTimePerGroupForTraining('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '2');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. w/alg. and NO alg. (3 and 4)
disp(sprintf('standard/straight 3/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '3');
mu = GetTrialTimePerGroup('standard', 'straight', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 3/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '3');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 3/4:\n'));
x = GetTrialTimePerGroup('head', 'straight', '3');
mu = GetTrialTimePerGroup('head', 'straight', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/circle 3/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '3');
mu = GetTrialTimePerGroup('head', 'circle', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. w/alg. and NO alg. with training (1 and 2)
disp(sprintf('standard/straight 1/2:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '2');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 1/2:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '2');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 1/2:\n'));
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '2');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/circle 1/2:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '2');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. trained group w/alg. and untrained w/alg. (1 and 3)
disp(sprintf('standard/straight 1/3:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '3');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 1/3:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '3');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 1/3:\n'));
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '3');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/circle 1/3:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '3');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. trained group NO alg. and untrained NO alg. (2 and 4)
disp(sprintf('standard/straight 2/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '2');
mu = GetTrialTimePerGroup('standard', 'straight', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 2/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 2/4:\n'));
x = GetTrialTimePerGroup('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));
disp(sprintf('head/circle 2/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. trained group w/alg. and untrained NO alg. (1 and 4)
disp(sprintf('standard/straight 1/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 1/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 1/4:\n'));
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/circle 1/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '4');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. trained group NO alg. and untrained w/alg. (2 and 3)
disp(sprintf('standard/straight 2/3:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '2');
mu = GetTrialTimePerGroup('standard', 'straight', '3');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 2/3:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '3');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 2/3:\n'));
x = GetTrialTimePerGroup('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '3');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/circle 2/3:\n'));
x = GetTrialTimePerGroup('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '3');
xx(:,1) = x(2,:)';
xx(:,2) = mu(2,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));
```

D.1.11 bonf_errors.m

```
pairs = [1 2];

% diff. btw. training data and trial data for group 1
disp(sprintf('diff. in train. standard/straight 1:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '1');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('diff. in train. standard/circle 1:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '1');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('diff. in train. head/straight 1:\n'));
x = GetTrialTimePerGroupForTraining('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '1');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('diff. in train. head/circle 1:\n'));
x = GetTrialTimePerGroupForTraining('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '1');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. training data and trial data for group 2
disp(sprintf('diff. in train. standard/straight 2:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'straight', '2');
mu = GetTrialTimePerGroup('standard', 'straight', '2');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('diff. in train. standard/circle 2:\n'));
x = GetTrialTimePerGroupForTraining('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '2');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('diff. in train. head/straight 2:\n'));
x = GetTrialTimePerGroupForTraining('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '2');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));
disp(sprintf('diff. in train. head/circle 2:\n'));
x = GetTrialTimePerGroupForTraining('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '2');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. w/alg. and NO alg. (3 and 4)
disp(sprintf('standard/straight 3/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '3');
mu = GetTrialTimePerGroup('standard', 'straight', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 3/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '3');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 3/4:\n'));
x = GetTrialTimePerGroup('head', 'straight', '3');
mu = GetTrialTimePerGroup('head', 'straight', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/circle 3/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '3');
mu = GetTrialTimePerGroup('head', 'circle', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. w/alg. and NO alg. with training (1 and 2)
disp(sprintf('standard/straight 1/2:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '2');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 1/2:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '2');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 1/2:\n'));
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '2');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/circle 1/2:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '2');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. trained group w/alg. and untrained w/alg. (1 and 3)
disp(sprintf('standard/straight 1/3:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '3');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 1/3:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '3');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 1/3:\n'));
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '3');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/circle 1/3:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '3');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. trained group NO alg. and untrained NO alg. (2 and 4)
disp(sprintf('standard/straight 2/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '2');
mu = GetTrialTimePerGroup('standard', 'straight', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 2/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 2/4:\n'));
x = GetTrialTimePerGroup('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/circle 2/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));
```

```
% diff. btw. trained group w/alg. and untrained NO alg. (1 and 4)
disp(sprintf('standard/straight 1/4:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '1');
mu = GetTrialTimePerGroup('standard', 'straight', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 1/4:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '1');
mu = GetTrialTimePerGroup('standard', 'circle', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 1/4:\n'));
x = GetTrialTimePerGroup('head', 'straight', '1');
mu = GetTrialTimePerGroup('head', 'straight', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/circle 1/4:\n'));
x = GetTrialTimePerGroup('head', 'circle', '1');
mu = GetTrialTimePerGroup('head', 'circle', '4');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

% diff. btw. trained group NO alg. and untrained w/alg. (2 and 3)
disp(sprintf('standard/straight 2/3:\n'));
x = GetTrialTimePerGroup('standard', 'straight', '2');
mu = GetTrialTimePerGroup('standard', 'straight', '3');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('standard/circle 2/3:\n'));
x = GetTrialTimePerGroup('standard', 'circle', '2');
mu = GetTrialTimePerGroup('standard', 'circle', '3');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/straight 2/3:\n'));
x = GetTrialTimePerGroup('head', 'straight', '2');
mu = GetTrialTimePerGroup('head', 'straight', '3');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));

disp(sprintf('head/circle 2/3:\n'));
x = GetTrialTimePerGroup('head', 'circle', '2');
mu = GetTrialTimePerGroup('head', 'circle', '3');
xx(:,1) = x(1,:)';
xx(:,2) = mu(1,:)';
[h,p,sigPairs] = ttest_bonf(xx, pairs, 0.05, 0);
disp(sprintf('H:%d P-value: %g\n\n',h,p));
```

D.1.12 process_forms.m

```
M = load( strcat('.\questionnaire_data.txt') );

% qustionnaire data format
%
% Participant ID
% Standard mouse group number
% Head tilt mouse group number
% Sex
% Age
% 9 pre-test question answers
% 6 post-test question answers
% 002,1,1,2,21,2,2,1,1,2,5,5,1,10,3,5,3,5,4,2

% maleAllCnt = 0;
% femaleAllCnt = 0;

% age stats: rows 1 - 4 are for each standard mouse group,
%            rows 5 - 8 are for each head tilt mouse group,
%            row 9 is for overall ages
%            col 2: min. age
%            col 3: max. age
age = [ 0 9999 -1 0;
0 9999 -1 0;
0 9999 -1 0;
0 9999 -1 0;
0 9999 -1 0;
0 9999 -1 0;
0 9999 -1 0;
0 9999 -1 0;
0 9999 -1 0;
        ]

% C H E C K  F O R M A T
for row = 1:size(M,1)
%     disp( sprintf('%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d\n'...
%                   , M(row,1), M(row,2), M(row,3), M(row,4), M(row,5), M(row,6)...
%                   , M(row,7), M(row,8), M(row,9), M(row,10), M(row,11), M(row,12)...
%                   , M(row,13), M(row,14), M(row,15), M(row,16), M(row,17), M(row,18)...
%                   , M(row,19), M(row,20) ) );
%

    % Overall age stats
    age(9,1) = age(9,1) + M(row,5);
    age(9,4) = age(9,4) + 1;
    % Store min.
    if (M(row,5)<age(9,2))
        age(9,2) = M(row,5);
    end
    % Store max.
    if (M(row,5)>age(9,3))
        age(9,3) = M(row,5);
    end

    % Age stats for standard mouse groups
    if ( 1==M(row,2) )
        age(1,1) = age(1,1) + M(row,5);
        age(1,4) = age(1,4) + 1;
        % Store min.
        if (M(row,5)<age(1,2))
            age(1,2) = M(row,5);
        end
        % Store max.
        if (M(row,5)>age(1,3))
            age(1,3) = M(row,5);
        end
    elseif ( 2==M(row,2) )
        age(2,1) = age(2,1) + M(row,5);
        age(2,4) = age(2,4) + 1;
        % Store min.
        if (M(row,5)<age(2,2))
            age(2,2) = M(row,5);
        end
        % Store max.
        if (M(row,5)>age(2,3))
            age(2,3) = M(row,5);
        end
    elseif ( 3==M(row,2) )
        age(3,1) = age(3,1) + M(row,5);
        age(3,4) = age(3,4) + 1;
        % Store min.
        if (M(row,5)<age(3,2))
            age(3,2) = M(row,5);
        end
        % Store max.
        if (M(row,5)>age(3,3))
            age(3,3) = M(row,5);
        end
    elseif ( 4==M(row,2) )
        age(4,1) = age(4,1) + M(row,5);
        age(4,4) = age(4,4) + 1;
        % Store min.
        if (M(row,5)<age(4,2))
            age(4,2) = M(row,5);
        end
        % Store max.
        if (M(row,5)>age(4,3))
            age(4,3) = M(row,5);
        end
    else
        disp( sprintf('Error unknown standard mouse group type on line %d', row) );
    end

    % Age stats for head-tilt mouse groups
    if ( 1==M(row,3) )
        age(5,1) = age(5,1) + M(row,5);
        age(5,4) = age(5,4) + 1;
        % Store min.
        if (M(row,5)<age(5,2))
            age(5,2) = M(row,5);
        end
        % Store max.
        if (M(row,5)>age(5,3))
            age(5,3) = M(row,5);
        end
    elseif ( 2==M(row,3) )
        age(6,1) = age(6,1) + M(row,5);
        age(6,4) = age(6,4) + 1;
        % Store min.
        if (M(row,5)<age(6,2))
            age(6,2) = M(row,5);
        end
        % Store max.
        if (M(row,5)>age(6,3))
            age(6,3) = M(row,5);
        end
    elseif ( 3==M(row,3) )
        age(7,1) = age(7,1) + M(row,5);
        age(7,4) = age(7,4) + 1;
        % Store min.
        if (M(row,5)<age(7,2))
            age(7,2) = M(row,5);
        end
        % Store max.
        if (M(row,5)>age(7,3))
            age(7,3) = M(row,5);
        end
    elseif ( 4==M(row,3) )
```

```
            age(8,1) = age(8,1) + M(row,5);
            age(8,4) = age(8,4) + 1;
            % Store min.
            if (M(row,5)<age(8,2))
                age(8,2) = M(row,5);
            end
            % Store max.
            if (M(row,5)>age(8,3))
                age(8,3) = M(row,5);
            end
        else
            disp( sprintf('Error unknown head-tilt mouse group type on line %d', row) );
        end

    % Participant ID
    if (0>=M(row,1) || 24<M(row,1))
            disp( sprintf('Error in 1st field on line %d', row) );
    end
    % Standard mouse group number
    if (1>M(row,2) || 4<M(row,2))
            disp( sprintf('Error in 2nd field on line %d', row) );
    end
    % Head tilt mouse group number
    if (1>M(row,3) || 4<M(row,3))
            disp( sprintf('Error in 3rd field on line %d', row) );
    end
    % Sex
    if (1>M(row,4) || 2<M(row,4))
            disp( sprintf('Error in 4th field on line %d', row) );
    end
    % Age
    if (4>M(row,5) || 70<M(row,5))
            disp( sprintf('Error in 5th field on line %d', row) );
    end
    % PRE-TEST QUESTIONS
    % Question 1
    if (1>M(row,6) || 2<M(row,6))
            disp( sprintf('Error in 6th field on line %d', row) );
    end
    % Question 2
    if (1>M(row,7) || 2<M(row,7))
            disp( sprintf('Error in 7th field on line %d', row) );
    end
    % Question 3
    if (1>M(row,8) || 2<M(row,8))
            disp( sprintf('Error in 8th field on line %d', row) );
    end
    % Question 4
    if (1>M(row,9) || 2<M(row,9))
            disp( sprintf('Error in 9th field on line %d', row) );
    end
    % Question 5
    if (1>M(row,10) || 2<M(row,10))
            disp( sprintf('Error in 10th field on line %d', row) );
    end
    % Question 6
    if (1>M(row,11) || 5<M(row,11))
            disp( sprintf('Error in 11th field on line %d', row) );
    end
    % Question 7
    if (1>M(row,12) || 5<M(row,12))
            disp( sprintf('Error in 12th field on line %d', row) );
    end
    % Question 8
    if (1>M(row,13) || 2<M(row,13))
            disp( sprintf('Error in 13th field on line %d', row) );
    end
    % Question 9
    if (0>M(row,14) || 25<M(row,14))
            disp( sprintf('Error in 14th field on line %d', row) );
```

```
end
% POST-TEST QUESTIONS
% Question 10
if (1>M(row,15) || 5<M(row,15))
        disp( sprintf('Error in 15th field on line %d', row) );
end
% Question 11
if (1>M(row,16) || 5<M(row,16))
        disp( sprintf('Error in 16th field on line %d', row) );
end
% Question 12
if (1>M(row,17) || 5<M(row,17))
        disp( sprintf('Error in 17th field on line %d', row) );
end
% Question 13
if (1>M(row,18) || 5<M(row,18))
        disp( sprintf('Error in 18th field on line %d', row) );
end
% Question 14
if (1>M(row,19) || 4<M(row,19))
        disp( sprintf('Error in 19th field on line %d', row) );
end
% Question 15
if (1>M(row,20) || 2<M(row,20))
        disp( sprintf('Error in 20th field on line %d', row) );
end
end  % C H E C K  F O R M A T

for i =1:size(age,1)
    average_ages(i) = age(i,1)/age(i,4);
end
```

D.2 Virtual tunnel data analysis

D.2.1 parsedatafiles.cpp

```
// parsedatafiles.cpp :
//

#include "stdafx.h"
#include "windows.h"

#include <string>
#include <iostream>
#include <fstream>
#include <list>
#include <algorithm>
#include <cctype>
using namespace std;

static const basic_string <char>::size_type npos = -1;

void replaceMovement(string & str, string & outstr)
{
    char str1[] =  "IN_TUNNEL_BEFORE->IN_TUNNEL_DURING";
    char str2[] =  "IN_TUNNEL_BEFORE->DONE";
    char str3[] =  "IN_TUNNEL_DURING->DONE";
    char str4[] =  "OUT_TUNNEL_DURING->IN_TUNNEL_DURING";
    char str5[] =  "IN_TUNNEL_DURING->OUT_TUNNEL_DURING";
    char str6[] =  "MOVE";

    outstr = "0";
    if (0==str.compare(str1))
    {
        outstr = "1";
    }
    else if (0==str.compare(str2))
    {
        outstr = "2";
    }
    else if (0==str.compare(str3))
    {
        outstr = "3";
    }
    else if (0==str.compare(str4))
    {
        outstr = "4";
    }
    else if (0==str.compare(str5))
    {
        outstr = "5";
    }
    else if (0==str.compare(str6))
    {
        outstr = "6";
    }
    else
    {
        cout << "CRAZY ERROR-EXITING" << endl;
        exit(1);
    }
}

void processLine(ofstream & out, string & str)
{


 //   cout << str << endl;

    str.erase(remove_if(str.begin(), str.end(), isspace), str.end());

    size_t idx1 = str.find(',', 0);
    if (npos != idx1)
    {   // first string
        string sub1 = str.substr(0,idx1);

        size_t idx2 = str.find(',', idx1+1);
        if (npos != idx2)
        {   // second string
            string sub2 = str.substr(idx1+1, idx2-idx1-1);
            string outstring2 = "0";
            replaceMovement( sub2, outstring2);
//            cout << sub2 << " | " << outstring2 << endl;

            idx1 = str.find(',', idx2+1);
            if (npos != idx1)
            {   // third string
                string sub3 = str.substr(idx2+1, idx1-idx2-1);

                // fourth string
                string sub4 = str.substr(idx1+1, str.length());

                out << sub1 << "," << outstring2 << "," << sub3 << ","
                            << sub4 << endl;
                //cout << sub1 << " | " << sub2 << " | " << sub3
                       << " | " << sub4 << endl << endl;
            }
        }
    }
}


typedef basic_string<TCHAR> tstring;
void processAFile( _TCHAR filename[] )
```

```
{
    char pmbbuf[MAX_PATH];

    size_t i = wcstombs( pmbbuf, filename, MAX_PATH );

    string str;
    ifstream in;    // Create an input file stream.
    in.open(pmbbuf);  // Use it to read from a file
    ofstream out;
    string outputfile = pmbbuf;
    outputfile += ".out";
    out.open(outputfile.c_str());
    str = pmbbuf;
//    cout << str << endl;
    if ( ! in )
    {
        cout << "Error: Can't open the file named " << pmbbuf << endl;;
        exit(1);
    }

    getline(in,str);  // Get the frist line from the file, if any.
    while ( in )
    {  // Continue if the line was sucessfully read.
        processLine(out, str);  // Process the line.
        getline(in,str);   // Try to get another line.
    }

    out.flush();
    out.close();
}

int _tmain(int argc, _TCHAR* argv[])
{

    WIN32_FIND_DATA FindFileData;
    HANDLE hFind;

    hFind = FindFirstFile(_T("*.txt"), &FindFileData);
    if (hFind != INVALID_HANDLE_VALUE)
    {
        processAFile( FindFileData.cFileName );
        while (FindNextFile(hFind, &FindFileData))
        {
            processAFile(FindFileData.cFileName);
        }
    }

    FindClose( hFind );
    return 0;
}
```

D.2.2 test_data.cpp

```
// test_data.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <string>
#include <iostream>
#include <fstream>
#include <math.h>
#include <assert.h>

#include "windows.h"

#include "nr3.h"
#include "interp_1d.h"
```

```
//#include "C:\WinDDK\7600.16385.0\src\input\my_moufiltr\interp.h"

int X, Y;
int Cx, Cy, sumX, sumY;
double prev_theta;


using namespace std;

const double radians_const = 57.2957795131;
bool is_above (double c_ratio, long Sumx, long Sumy)
{
    bool retval = true;
    double Ly, dSumx, dSumy;

    dSumx = (double) Sumx;
    dSumy = (double) Sumy;

    Ly = c_ratio * dSumx;

    if (Ly > dSumy)
        retval = false;
    return retval;
}

inline bool is_same_sign (long x, long y)
{
    return ( ((x >= 0) && (y  >= 0)) || ((x < 0)  && (y < 0)) ) ? true : false;
}
int current_pos = 0;

void apply_virtual_tunnel(double Cx, double Cy, double X, double Y,
                          double * new_x, double * new_y, int max_q_size, int m)
{

    // Apply previous values
    {
        //Cx = mybaryrat_func1.interp( (double) current_pos+max_q_size );
        //Cy = mybaryrat_func2.interp( (double) current_pos+max_q_size );

        double d = sqrt( (X-Cx)*(X-Cx) + (Y-Cy)*(Y-Cy) );

        prev_theta = atan2( (Y-Cy), (X-Cx) );
        double a = abs((d - d/(double)m) * cos(prev_theta));
        double b = abs((d - d/(double)m) * sin(prev_theta));

        double d_new_x, d_new_y;

        if (X >= Cx)
            d_new_x = Cx + a;
        else
            d_new_x = Cx - a;

        if (Y >= Cy)
            d_new_y = Cy + b;
        else
            d_new_y = Cy - b;

        //*new_x = my_ftol( X - (xb[current_pos-1] - d_new_x) );
        //*new_y = my_ftol( Y - (yb[current_pos-1] - d_new_y) );

        //if ( abs(*new_x) > abs(X) ) *new_x = X;
        //if ( abs(*new_y) > abs(Y) ) *new_y = Y;

        *new_x = d_new_x;
        *new_y = d_new_y;
    }
}

void processLine(std::string & str)
```

```
{
    union
    {
        double d;
        unsigned long long u;
    } x1, x2, x3;

   unsigned long u1, u2, u3, u4, u5, u6;
   int i1, i2, i3, i4;

   string pre = str.substr(0, 6);

   if (pre.compare("###1- ")==0)
   {
        sscanf_s(str.c_str(), "###1- X: %d, Y: %d, diff: %d,
                 current_pos: %d", &i1, &i2, &i3, &i4);

        X = i1;
        Y = i2;
        current_pos = i4;

        sumX += X;
        sumY += Y;

        cout << ">>>>1 - X:" << i1 << " Y:" << i2 << " diff:" << i3
                             << " current_pos:" << i4 << endl;
   }
   else if (pre.compare("###8- ")==0)
   {
        sscanf_s(str.c_str(), "###8- X: %d, Y: %d, diff: %d,
                 theta %08lX %08lX\n", &i1, &i2, &i3, &u1, &u2 );
           //    KdPrint(("###8- X: %d, Y: %d, diff: %d,
           //                 theta %8.8lX %8.8lX\n", X, Y, diff, newtheta));
        x1.u = u2;
        x1.u <<= 32;
        x1.u |= u1;

        cout << i1 << "," << i2 << "," << i3 << "," << x1.d << endl;
   }
   else if (pre.compare("###7- ")==0)
   {
        sscanf_s(str.c_str(), "###7- old_dist: %08lX %08lX, new_dist: %08lX %08lX\n",
                              &u1, &u2, &u3, &u4);
           //KdPrint(("###7- old_dist: %8.8lX %8.8lX, new_dist: %8.8lX %8.8lX\n",
                       old_dist, new_dist ));
        x1.u = u2;
        x1.u <<= 32;
        x1.u |= u1;

        x2.u = u4;
        x2.u <<= 32;
        x2.u |= u3;
        cout << "\t>>>>7 - old_dist:" << x1.d << " new_dist:" << x2.d << endl;
   }
   else if (pre.compare("\t###2-")==0)
   {
        sscanf_s(str.c_str(), "\t###2- d: %08lX %08lX, d_new_x: %08lX %08lX, d_new_y: %08lX %08lX",
                 &u1, &u2, &u3, &u4, &u5, &u6 );

        x1.u = u2;
        x1.u <<= 32;
        x1.u |= u1;

        x2.u = u4;
        x2.u <<= 32;
        x2.u |= u3;

        x3.u = u6;
        x3.u <<= 32;
        x3.u |= u5;
```

```
  //      cout << "\t>>>>2 - d:" << x1.d << " d_new_x:" << x2.d
                               << " d_new_y:" << x3.d << endl;
   }
   else if (pre.compare("\t\t###3")==0)
   {
        sscanf_s(str.c_str(), "\t\t###3- prev_theta: %08lX %08lX,
                 prev_Cx: %d, prev_Cy: %d",  &u1, &u2, &i1, &i2 );

        x1.u = u2;
        x1.u <<= 32;
        x1.u |= u1;

        Cx = i1;
        Cy = i2;
        prev_theta = x1.d;

        sumX = 0;
        sumY = 0;
   }
   else if (pre.compare("\t###5-")==0)
   {
        sscanf_s(str.c_str(), "\t###5- above: %d, same: %d, a: %08lX %08lX,
                 b: %08lX %08lX", &i1, &i2, &u1, &u2, &u3, &u4);

        x1.u = u2;
        x1.u <<= 32;
        x1.u |= u1;

        x2.u = u4;
        x2.u <<= 32;
        x2.u |= u3;

//        cout << "\t\t>>>>5 - above:" << i1 << " same:" << i2
//               << " a:" << x1.d << " b:" << x2.d << endl;
   }
   else if (pre.compare("###4- ")==0)
   {
        sscanf_s(str.c_str(), "###4- new_x: %d, new_y: %d", &i1, &i2 );
        cout << ">>>>4 - new_x:" << i1 << " new_y:" << i2 << endl;
        if (Cx != 0 && Cy != 0 && prev_theta != 0.0)
        {
                double C_ratio = (double)Cy / (double)Cx;

                double d = abs(C_ratio * sumX - sumY) / ( sqrt(C_ratio * C_ratio+1) );

                double d_new_x;// = X - ((d/2.0) * cos( prev_theta )) ;
                double d_new_y;// = Y + ((d/2.0) * sin( prev_theta )) ;

                bool above = is_above( C_ratio, sumX, sumY );
                bool same_sign = is_same_sign( sumX, sumY );

                double a = abs((d/(double)2.0) * cos(prev_theta));
                double b = abs((d/(double)2.0) * sin(prev_theta));

                if ( !above && same_sign )
                {
                    d_new_x = X - (a);
                    d_new_y = Y + (b);
                }
                else if ( above && same_sign )
                {
                    d_new_x = X + (a);
                    d_new_y = Y - (b);
                }
                else if ( !above && !same_sign )
                {
                    d_new_x = X + (a);
                    d_new_y = Y + (b);
                }
                else if (above && !same_sign )
                {
                    d_new_x = X - (a);
                    d_new_y = Y - (b);
                }

                long new_x = d_new_x;
                long new_y = d_new_y;

                double calc_new_d = abs(C_ratio * (sumX - X + new_x) - (sumY - Y + new_y ))
                                    / ( sqrt(C_ratio * C_ratio+1) );
        }
//          cout << endl;
   }
   else
        cout << "The impossible has happened" << endl;
}

/*
double TT[] = {  1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0  };
double XX[] = {   4.0,  6.0,  9.0, 10.0,  11.0,  10.0,  9.0,    8.0,   6.0,   3.0 };
double YY[] = {  -2.0, -3.0, -6.0, -9.0, -13.0, -17.0, -19.0, -21.0, -23.0, -26.0};
*/
/*
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
const int MAX_Q_SIZE = 50;
int current_pos = 1;
double Cx = 0.0;
double Cy = 0.0;
long prev_Cy = 0;
long prev_Cx = 0;
double theta = 0.0;
double prev_theta = 0.0;
double d = 0.0;
double d_new_x, d_new_y;

double tb[] = { 1.0, 2.0, 3.0, 4.0 };
double xb[] = { 0.0, 0.0, 0.0, 0.0 };
double yb[] = { 0.0, 0.0, 0.0, 0.0 };
double prev_xb[] = { 0.0, 0.0, 0.0, 0.0 };
double prev_yb[] = { 0.0, 0.0, 0.0, 0.0 };

MyBaryRat_interp mybaryrat_func1;
MyBaryRat_interp mybaryrat_func2;

void copyarray(double a[], double b[], int size)
{
    double temp;
    for (int i = 0; i < size; ++i)
    {
        temp = a[i];
        a[i] = b[i];
        b[i] = temp;
    }
}

void cleararray(double a[], int size)
{
    for (int i=0; i < size; ++i)
        a[i] = 0.0;
}



void apply_virtual_tunnel(LONG X, LONG Y, LONG * new_x, LONG * new_y, int max_q_size, int m)
{
    // store our data points
    if (current_pos == 1)
    {
        xb[0] = prev_xb[max_q_size-1] + (double) X;
        yb[0] = prev_yb[max_q_size-1] + (double) Y;
    }
    else
    {
        xb[current_pos-1] = xb[current_pos-2] + (double) X;
        yb[current_pos-1] = yb[current_pos-2] + (double) Y;
    }

    // Apply previous values
    if (!( (prev_yb[max_q_size-1] == 0.0) && (prev_xb[max_q_size-1] == 0.0) ))
    {
        Cx = mybaryrat_func1.interp( (double) current_pos+max_q_size );
        Cy = mybaryrat_func2.interp( (double) current_pos+max_q_size );

        d = sqrt( (xb[current_pos-1]-Cx)*(xb[current_pos-1]-Cx) +
                  (yb[current_pos-1]-Cy)*(yb[current_pos-1]-Cy) );

        prev_theta = atan2( (yb[current_pos-1]-Cy), (xb[current_pos-1]-Cx) );
        double a = abs((d - d/(double)m) * cos(prev_theta));
        double b = abs((d - d/(double)m) * sin(prev_theta));

        if (xb[current_pos-1] >= Cx)
            d_new_x = Cx + a;
        else
            d_new_x = Cx - a;

        if (yb[current_pos-1] >= Cy)
            d_new_y = Cy - b;
        else
            d_new_y = Cy + b;

        cout <<  "\t" << xb[current_pos-1] << "\t" << yb[current_pos-1]
             << "\t" << Cx << "\t" << Cy << "\t" << d_new_x << "\t" << d_new_y;
        *new_x = (LONG)  X - (xb[current_pos-1] - d_new_x);
        *new_y = (LONG)  Y - (yb[current_pos-1] - d_new_y);
        double old_dist = sqrt ( (Cx-xb[current_pos-1])*(Cx-xb[current_pos-1])
                          + (Cy-yb[current_pos-1])*(Cy-yb[current_pos-1]) );
        double new_dist = sqrt ( (Cx-(xb[current_pos-1]-X+*new_x))*
                                 (Cx-(xb[current_pos-1]-X+*new_x))
                          + (Cy-(yb[current_pos-1]-Y+*new_y))*
                            (Cy-(yb[current_pos-1]-Y+*new_y)) );
        assert( old_dist >= new_dist );
    }
}


int DoTunnel(LONG X, LONG Y, LONG * new_x, LONG * new_y, int max_q_size, int m )
{

    static LARGE_INTEGER prev_time;
    LARGE_INTEGER now;
    FILETIME ft;
    GetSystemTimeAsFileTime(&ft);  // returns System time as a count of 100-nanosecond
                                   // (10-7) intervals since January 1, 1601.
    now.HighPart = ft.dwHighDateTime;
    now.LowPart  = ft.dwLowDateTime;

    long diff = (long) (now.QuadPart - prev_time.QuadPart);
    diff = diff/10000; // divide by 10,000 (10^4) to change to milliseconds
    prev_time = now;

    *new_x = X;
    *new_y = Y;

    if (current_pos <= max_q_size)
    {
        if (diff < 100) // time since last point is below threshold.
        {
            apply_virtual_tunnel(X, Y, new_x, new_y, max_q_size, m);
            current_pos++;
        }
        else
        {
            current_pos = 1;
            cleararray( xb,      max_q_size );
            cleararray( yb,      max_q_size );
            cleararray( prev_xb, max_q_size );
            cleararray( prev_yb, max_q_size );
            apply_virtual_tunnel(X, Y, new_x, new_y, max_q_size, m);
        }
    }
    else
    {
        copyarray(prev_xb, xb, max_q_size);
        copyarray(prev_yb, yb, max_q_size);

        MyBaryRat_interp func1;
        func1.construct(tb, prev_xb, 1);
        MyBaryRat_interp func2;
        func2.construct(tb, prev_yb, 1);

        mybaryrat_func1 = func1;
        mybaryrat_func2 = func2;

        current_pos = 1;

        // Apply for last value
        apply_virtual_tunnel(X, Y, new_x, new_y, max_q_size, m);
}
    return 0;
}
*/
//double XX[] = {   4.0,  6.0,  9.0, 10.0,  11.0,  10.0,  9.0,    8.0,   6.0,   3.0 };
//double YY[] = {  -2.0, -3.0, -6.0, -9.0, -13.0, -17.0, -19.0, -21.0, -23.0, -26.0};
//LONG XX[] = { 4,  2,  3, 1,  1,  -1,  -1,    -1,   -2,   -3};
//LONG YY[] = { -2, -1, -3, -3, -5, -4, -2, -2, -2, -3};

void write_w(char str[], VecDoub w)
{
    cout << str << endl;
    for (int i = 0; i < 4; ++i)
    {
        cout << "\tw[" << i << "] = " << w[i] << endl;
    }
}
int _tmain(int argc, _TCHAR* argv[])
{
/*
    LONG x, y;
    cout << "X\tY\tXb\tYb\tCx\tCy\td_new_x\td_new_y\tnewx\tnewy" << endl;
    for (int i = 0; i < 10; ++i)
    {
        cout << XX[i] << "\t" << YY[i];
        DoTunnel(XX[i], YY[i], &x, &y, 4, 2 );
        cout << "\t" << x << "\t" << y << endl;
    }
*/
/*
    X = Y = Cx = Cy = sumX = sumY = 0;
    prev_theta = 0.0;

    std::string str;
    ifstream in;    // Create an input file stream.
    in.open("c:\\data.txt");  // Use it to read from a file named data.txt.
    if ( ! in )
    {
        cout << "Error: Can’t open the file named data.txt.\n";
        exit(1);
    }
    getline(in,str);  // Get the frist line from the file, if any.
    while ( in )
    {  // Continue if the line was sucessfully read.
        processLine(str);  // Process the line.
        getline(in,str);   // Try to get another line.
    }
*/

    VecDoub xx(4), yy(4), tt(4);
    Doub dy;

    double x[] = {4.0,   6.0,  9.0, 10.0,  11.0,  10.0,   9.0,   8.0,   6.0,   3.0};
    double y[] = {-2.0, -3.0, -6.0, -9.0, -13.0, -17.0, -19.0, -21.0, -23.0, -26.0};
    double t[] = {1.0, 2.0, 3.0, 4.0};


// ====================================================================================
    cout << "DATA--";
    for (int i = 0; i < 4; ++i)
    {
        xx[i] = x[i];
        yy[i] = y[i];
        tt[i] = t[i];
        cout << "[" << x[i] << ", " << y[i] << "] ";
    }
    cout << endl;

    {
        BaryRat_interp myfunc1(tt, xx, 1);
        write_w("myfunc1 set 1", myfunc1.w);
        BaryRat_interp myfunc2(tt, yy, 1);
        write_w("myfunc2 set 1", myfunc2.w);
        Doub dd = 5.0;
        Doub dx;
        cout << "RESULTS--";
        for (int j = 0; j<4; ++j)
        {
            dx = myfunc1.interp( dd+j );
            dy = myfunc2.interp( dd+j );
            cout << "[" << dx << ", " << dy << "] ";
        }
        cout << endl << endl;
    }
// ====================================================================================
    cout << "DATA--";
    for (int i = 0; i < 4; ++i)
    {
        xx[i] = x[i+4];
        yy[i] = y[i+4];
        tt[i] = t[i];
        cout << "[" << x[i+4] << ", " << y[i+4] << "] ";
    }
    cout << endl;

    {
        BaryRat_interp myfunc1(tt, xx, 1);
        write_w("myfunc1 set 2", myfunc1.w);
        BaryRat_interp myfunc2(tt, yy, 1);
        write_w("myfunc2 set 2", myfunc2.w);
        Doub dd = 5.0;
        Doub dx;
        cout << "RESULTS--";
        for (int j = 0; j<4; ++j)
        {
            dx = myfunc1.interp( dd+j );
            dy = myfunc2.interp( dd+j );
            cout << "[" << dx << ", " << dy << "] ";
        }
        cout << endl << endl;
    };
/*
    double new_x, new_y;

    double results_x[] = {  9.6,   9,  8.5,    8.11765,  7,    6 };
    double results_y[] = {-10.8, -12,  -13,   -13.9412,  -24.2,  -28 };

    cout << "X: ";
    for (int i = 0; i < 6; ++i)
    {
        apply_virtual_tunnel(results_x[i], results_y[i], x[4+i], y[4+i], &new_x, &new_y, 4, 2);
        cout << new_x << " ";
    }

    cout << endl << "Y: ";
    for (int i = 0; i < 6; ++i)
    {
        apply_virtual_tunnel(results_x[i], results_y[i], x[4+i], y[4+i], &new_x, &new_y, 4, 2);
        cout << new_y << " ";
    }
*/
    return 0;
}
```