

Haptically Enhanced Mouse

E90 Final Report

Heather Jones and Lauren Stadler

Advisor: Erik Cheever

May 4, 2006

Abstract

We designed and built a haptically enhanced mouse for blind and severely visually impaired computer users. A haptic mouse uses tactile feedback to convey information about the cursor position on the screen. A basic trackball mouse was modified to include a brake to indicate when the pointer reaches the edge of the screen and a vibrating motor that alerts the user when the pointer passes over an icon or the edge of a window. In addition, an audio feedback feature was developed that provides information about what the pointer is over. The mouse connects through the USB port to a computer running Linux. A GUI in Java was developed to test the functionality of the mouse. Although there was a large variation in the performance of our test subjects, the results indicated that users who are unable to the screen can locate and identify objects faster with haptic and speech feedback than with only speech feedback. They also demonstrated that users can determine basic spatial relationships between objects on a screen through exploration using the haptic mouse.

Table of Contents

Abstract.....	1
Table of Contents.....	2
List of Figures	3
List of Tables	3
1. Introduction.....	4
2. Overview and Design Choices.....	5
3. Hardware.....	7
3.1. Box Base.....	8
3.2. Brake Mechanism.....	8
3.2.1. Initial Design.....	9
3.2.2. Final Design.....	10
3.3. Vibrating Motor.....	12
3.4. Circuit Schematic.....	12
3.5. Controller Chip.....	13
4. Software.....	14
4.1. PIC Software.....	14
4.2. Host to Device Communication.....	15
4.3. Text-to-Speech Synthesis System.....	17
4.4. GUI in Java.....	18
4.5. C-Java Interface.....	20
4.6. X Window Interaction.....	20
5. Testing.....	21
5.1. Design of Experiments.....	21
5.1.1. Test 1.....	21
5.1.2. Test 2.....	23
5.2. Results.....	25
5.2.1. Test 1.....	25
5.2.2. Test 2.....	26
6. Conclusions and Future Work.....	27
7. Acknowledgements.....	29
8. References.....	29
9. Appendices.....	30
9.1. Appendix A: PIC Microcontroller Code.....	31
9.2. Appendix B: Host to Device Communication Code.....	34
9.3. Appendix C: GUI Code.....	38
9.4. Appendix D: Other Code.....	51
9.5. Appendix E: Test 1 Data.....	55
9.6. Appendix F: Test 2 Drawings.....	60

List of Figures

Figure 1. Block diagram.....	5
Figure 2. Top and side views of mouse box.....	8
Figure 3. Underside of mouse box lid.....	9
Figure 4. Initial brake design.....	9
Figure 5. Final brake design.....	10
Figure 6. View of mouse with trackball removed and brake emerging.....	11
Figure 7. Vibrating motor.....	12
Figure 8. Vibrating motor mounted inside trackball mouse shell.....	12
Figure 9. Circuit schematic for brake and vibrating motor.....	13
Figure 10. Some objects available in the javax.swing package.....	19
Figure 11. Screenshot of our GUI.....	20
Figure 12. Blindfolded test subject using mouse.....	21
Figure 13. Tactile screenshot for blind test subject.....	22
Figure 14. GUI used for test 1.....	23
Figure 15. Tactile sketching for blind test subject.....	24
Figure 16. GUI used in test 2.....	24

List of Tables

Table 1. PIC input signal and corresponding action.....	15
Table 2. Average test 1 times for all subjects	25
Table 3. Blind test subject's results.....	26
Table 4. Trackball vs. non-trackball users (with speech and haptic feedback)	26

1 Introduction

Our motivation for this project comes from the changing nature of computer interfaces. In the early days of computing, the output on a screen was much more text-based and sequential than it is today; what a blind user with a screen reader could perceive was pretty much the same as what a sighted user could perceive. As the dependency on graphical user interfaces increases, however, computers are becoming less accessible to the blind. It is very hard for a screen reader to accurately convey the arrangement of icons and windows on a screen. Haptic technology, which is technology that uses the sense of touch to convey information, provides a better sense of spatial relationships than the audio feedback from a screen reader.

The objective of this project was to build a mouse with supporting software, comprised of simple, inexpensive, off-the-shelf components, which uses haptic and audio cues to improve the experience of a blind or visually impaired user navigating a computer screen. The functions the mouse can perform include braking when the cursor reaches the edge of the screen and vibrating when the cursor passes over an icon or edge of a window; an audio feedback option that informs the user what the cursor is currently over has been added in software. The goal in designing this mouse was to allow a blind or visually impaired user form a mental image of the screen and to enable him or her to perform simple tasks such as finding icons on the screen. It should, at a minimum, help blind users better understand how sighted people experience a computer and facilitate their performance of day-to-day computing activities.

This paper is a detailed description of the design, implementation, and testing involved in building a haptically enhanced mouse. Section 2 is a general overview of how the mouse works and the major design choices made. The construction of the physical mouse and the hardware components are discussed in section 3 and the software developed for the microcontroller and the host computer is described in section 4. Section 5 discusses how the mouse was tested for effectiveness as well as the results of the tests. Conclusions drawn from those tests and suggestions for future work are presented in Section 6.

2 Overview and Design Choices

Figure 1 is a block diagram that illustrates the major components in the mouse and how they interact. The design of the mouse can be broken into two main categories: the physical mouse and the software on the computer. The physical mouse is comprised of a trackball mouse, plus a brake mechanism, a vibrating motor, and a controller, which we installed to perform additional functions. The software portion of the mouse consists of a write function that controls the communication between the host and the device, a graphical user interface (GUI) which was developed for testing the mouse's effectiveness, a C-Java interface program that allows the GUI, written in java, to call the write function, written in C, and a speech-to-text synthesis program. There is also code for the microcontroller that receives data from the computer through the USB port and activates the vibrating motor and the brake.

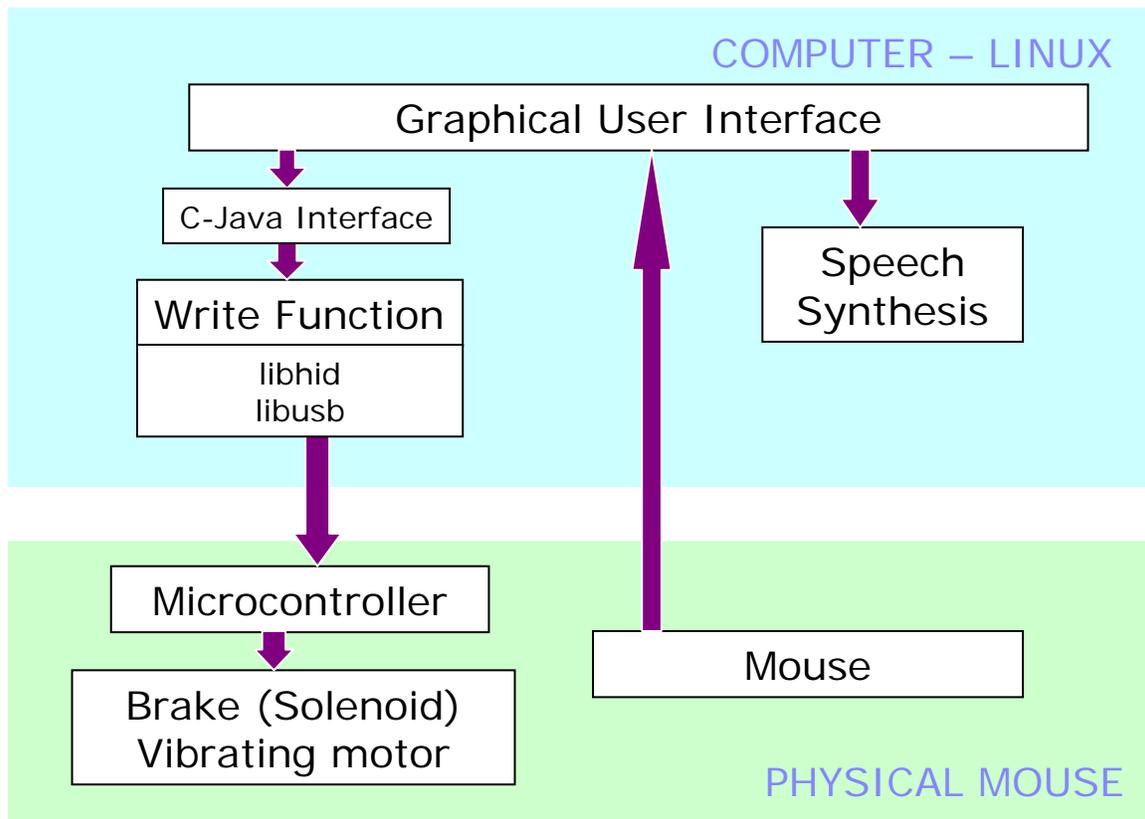


Figure 1. Block Diagram

The three major design choices that will be described in this section include the use of a trackball mouse, the choice of USB to interface with the computer, and the decision to work with a Linux operating system.

We decided to use a trackball as opposed to a traditional mouse. This choice was based on the fact that a trackball mouse is easier to brake because only the motion of the ball needs to be prevented, and not the movement of the entire mouse. The direct contact between the user's thumb and the trackball will also make it easier to detect when the brake is engaged. In addition, since trackball mice are stationary, the added size and weight of the brake and vibrating motor will not be a disadvantage.

There are many interfaces available for computers to communicate with peripheral devices. In most systems on the market today, however, the Universal Serial Bus (USB) has become the dominant interface, with lower-cost systems phasing out the older "legacy" ports completely. Since we want our mouse to be a prototype of a device that would be useful to the mainstream blind user without requiring the purchase of additional expensive hardware, we chose to use USB. Additionally, the USB specification is controlled by the USB Implementer's Forum, which includes over 900 companies in the technology industry. This means USB is not a proprietary technology: although a \$1500 fee is required to obtain a vendor ID before selling any product including a USB interface, there is no licensing fee required to develop USB software.

There are four transfer modes in the USB protocol: control, interrupt, bulk and isochronous. Control transfers are generally used for enumeration, configuration setting, and other similar tasks. Enumeration occurs when the device is first plugged in to the computer. It allows the host to determine what device has been attached, what driver should be used to handle the new device, and what transfer speed, (low, full or high), and transfer type, should be used with the device. Interrupt transfers are usually used to send data to and from a keyboard, mouse or other human interface device. Most of the communication for our device is done through interrupt transfers. Bulk transfers are used to send large blocks of data. Isochronous transfers are used for time-critical data, such as streaming audio. In each transfer type, data is sent as part of a packet. The number of packets needed to complete a given data transfer, as well as the amount of handshaking that occurs, depends on the transfer type.

Two versions of the USB specification are currently in use. USB 1.1 supports a low-speed data transfer rate of 1.5 Megabits per second and a full-speed data transfer rate of 12 Megabits per second. The newest version of the specification, USB 2.0 also supports a high speed data transfer rate of 480 Megabits per second. For this project, because we take input from and provide feedback to a human user, who necessarily operates at a much lower speed than the computer, low-speed data transfer is sufficient.

To create a USB device, we need

- A controller chip with a USB interface and the ability to control all other device functions
- Code on this chip to handle the USB interface and other device functions
- A program on the host to communicate with the device
- A program on the host to test the device
- A program that allows the device to perform its desired functions, which in our case was a graphical user interface.

In order to make our mouse usable for the greatest number of visually impaired users, we would ideally like it to work with the Windows™ operating system. Neither of us, however, have done any software development for Windows, so we chose to work within the Linux operating system. Unlike Windows, Linux is open source, meaning all the source code for the operating system is openly available. Open source has both its advantages and disadvantages. Since we have access to all of the source code, we can look at it and discover what might be causing problems and alter the code if need be. However, because open source software is often developed separately by a number of different people, it may lack cohesion. Should our device actually go into production, it would not be difficult for someone more familiar with Windows driver development to write a new driver for our device.

3 Hardware

We purchased a Logitech Trackman® Wheel trackball mouse and added a base for the mouse to rest on and to hold the additional hardware components, a brake mechanism, a vibrating motor, and a controller chip.

3.1. Box Base

We built a wooden box on which the trackball mouse rests. to which the brake mechanism and controller are mounted. The box is 8" wide, 10" long, and 4" deep. For our prototype, we chose to make it unnecessarily large because we did not want to be limited by any space constraints in the development process. Figure 2a is a top view of the mouse and Figure 2b is a side view.

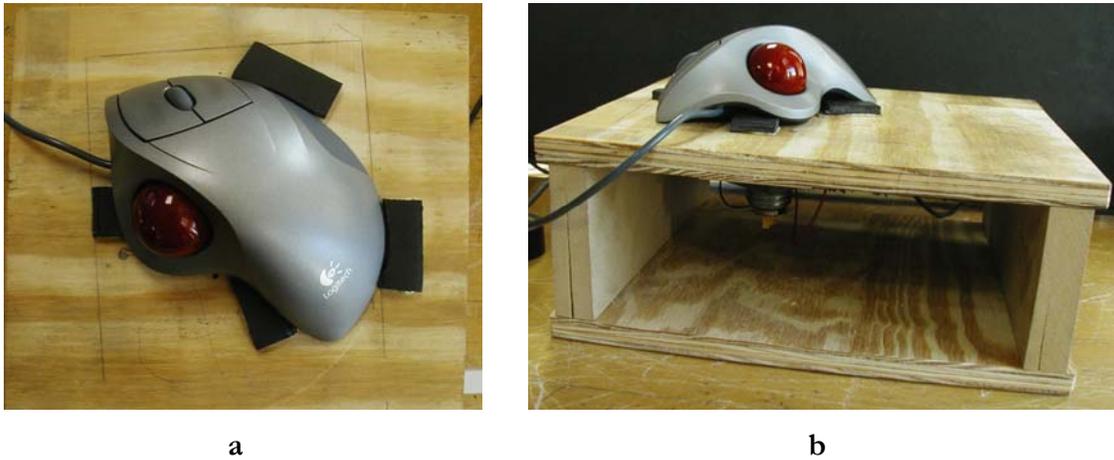


Figure 2. Top and side views of the mouse box

The lid of the box is attached using dowels such that it is removable. In addition, the box is open on the sides and so that we can easily access and adjust the controller and brake mechanism inside the box without removing the lid. Mounted on the underside of the lid is the brake mechanism, the circuitry for the vibrating motor and the brake, and the PIC development board (See Figure 3). Three cables emerge from the mouse body: a power cable that plugs into the wall and supplies 12V to the solenoid (the brake), a USB cable from the PIC, and a USB cable from the trackball mouse. Currently our mouse requires two USB ports, but they could be consolidated to one port using a hub.

3.2. Brake Mechanism

To brake the trackball, we chose to use a small 12V push/pull solenoid. A solenoid consists of a coil of wire with a metal shaft in its core. When current runs through the coil it creates a magnetic field that pushes the shaft out of the coil. The solenoid rests inside a threaded tube, which screws into the lid of the mouse box. The brake goes through a hole in the lid of the box and a hole in the underside of the mouse and pushes against the underside of the trackball when engaged. The threaded tube allows for easy height adjustment of the

solenoid. This feature was extremely useful because it permitted us to easily change the position of the solenoid as the brake design evolved. We wanted to power the solenoid from the USB port, however it requires 12V and the USB only provides 5V. Therefore, it is powered using a 12V AC/DC wall-adaptor. The circuit schematic for the brake is presented in section 3.4.

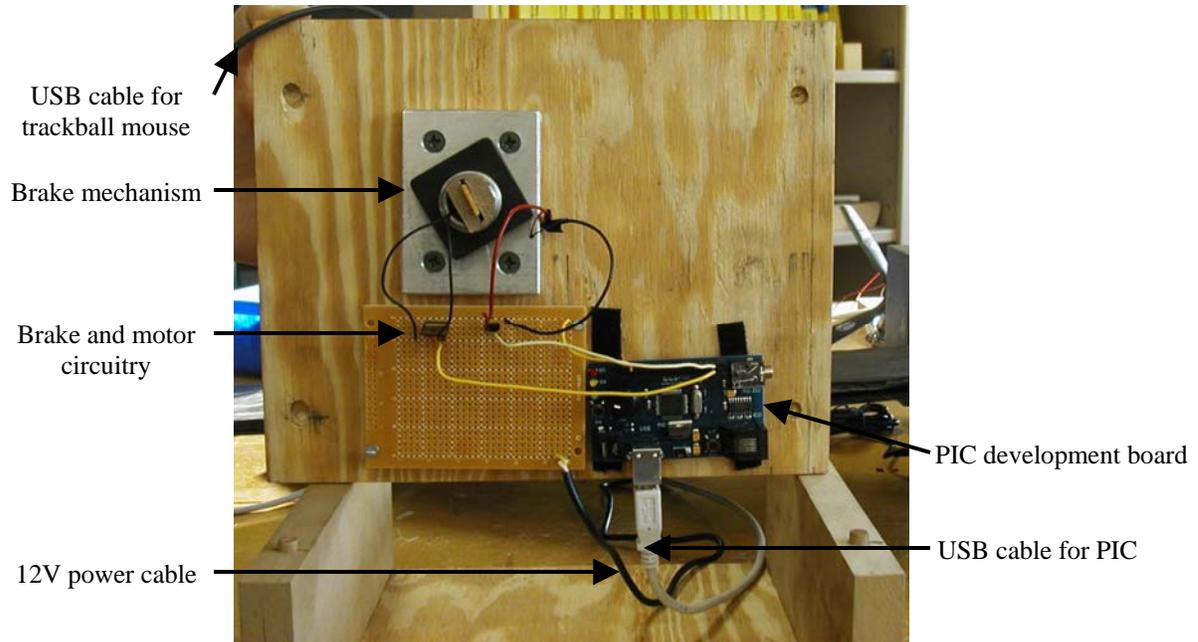


Figure 3. Underside of mouse box lid

3.2.1. Initial Design

In the initial brake design, an aluminum cylinder was simply attached to the end of the solenoid shaft and a small circular piece of textured rubber was super-glued to the end of the cylinder. Figure 4 is a sketch and a photo of the original brake configuration.

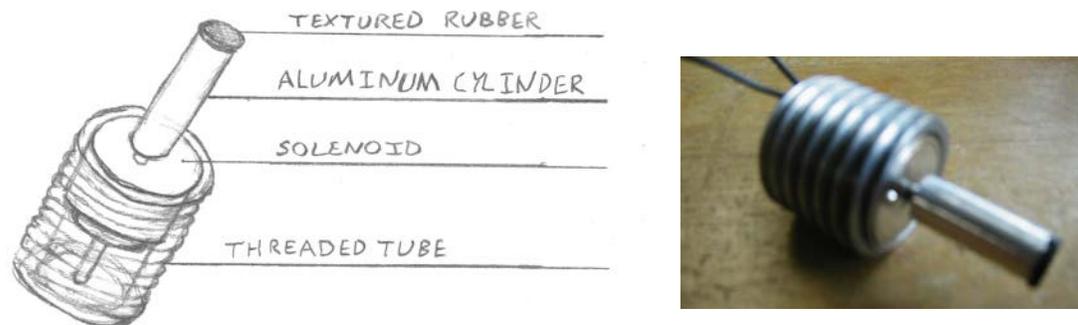


Figure 4. Initial brake design

This design had only one point of contact with the trackball and proved to be insufficient for resisting movement of the ball in all directions. Since the rubber pad was flat, the area of the pad in contact with the curved surface of the trackball was very small. In addition, the textured rubber failed to produce enough friction to create a significant amount of resistance. When we tested the mouse we found that this design prevented the trackball from spinning left-to-right and front-to-back, in which case the ball would have to slide over the pad. However, it did not prevent the ball from rotating about the brake. This was a problem because the spinning motion of the trackball corresponds to the cursor moving up and down on the screen; a new design was imperative.

3.2.2. Final Design

To improve the brake mechanism, we decided to move from one point of contact to two points of contact. This was accomplished by removing the rubber pad at the end of the solenoid and attaching an aluminum cup-section to the end of the cylinder. On top of the cup-section are two pieces of foam with textured rubber circles glued on top of them. The foam allows for the brake to form to the surface of the trackball. In addition, a different kind of textured rubber was used with finer bumps and a higher coefficient of friction. Figure 5 is a sketch and a picture of the final brake design.

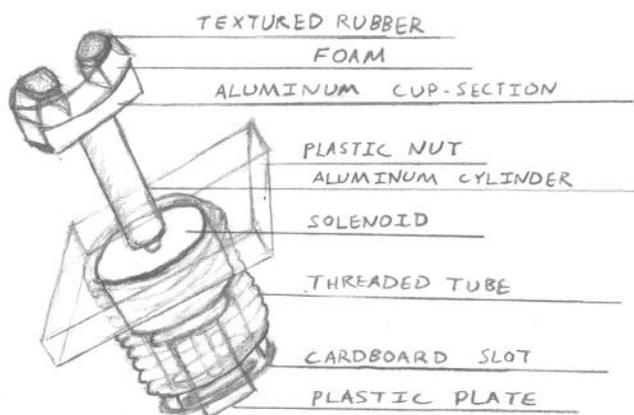


Figure 5. Final brake design

Since the head of the new brake is much larger than the first, it required us to increase the size of the hole in the lid and in the underside of the mouse so that it could fit through. In Figure 6, the trackball is removed and the brake can be seen emerging through

the rectangular hole in the bottom of the mouse. Ideally we would have created a brake with three points of contact. However this was not possible due to space constraints; the size of the hole that we could carve out of the underside of the mouse was limited by the electrical components and optical sensor, so the hole is just barely large enough for the brake to fit through.



Figure 6. View of mouse with trackball removed and brake emerging

In testing the new design, it came to our attention that if the brake was slightly misaligned with the hole, it would get caught and could no longer engage or disengage. The misalignment was most often due to the solenoid twisting, since there was nothing keeping it from turning in the mount or to keep the threaded tube from unscrewing. The plastic nut and the plastic plate that fits into the cardboard slot were added to prevent the solenoid body and shaft from rotating (See Figure 5). The plastic nut screws tightly around the threaded tube and prevents it from unscrewing. The plastic plate is glued to the shaft of the solenoid and fits into the cardboard slot to prevent the shaft from turning, but still allows the it to move up and down as the solenoid engages and disengages.

While the final brake mechanism does not stop the movement of the ball completely, it does produce a noticeable amount of resistance which is sufficient to alert the user when the cursor reaches the edge of the screen.

To disengage the brake we considered installing an additional button on the mouse that would serve as a brake-release. For the final design, however, we decided that instead of adding more hardware that would be difficult to put in, we could simply use the buttons already available and write the software such that one of them could function as a brake release. Currently, clicking on the right mouse button releases the brake.

3.3. Vibrating Motor

To vibrate the mouse, we decided to use a small 3V asymmetrically weighted motor such as those found in cellular phones (See Figure 7).



Figure 7. Vibrating motor

The motor is encased by plastic tubing to ensure that it is free to spin. It is super-glued to the inside of the upper shell of the mouse, underneath where the palm rests (See Figure 8). A small hole was drilled in the bottom of the mouse and the lid of the box for the wires to pass through. The motor is powered by the USB port with a voltage regulator that steps down 5V to 3V. The circuit schematic for the motor is discussed in section 3.4.

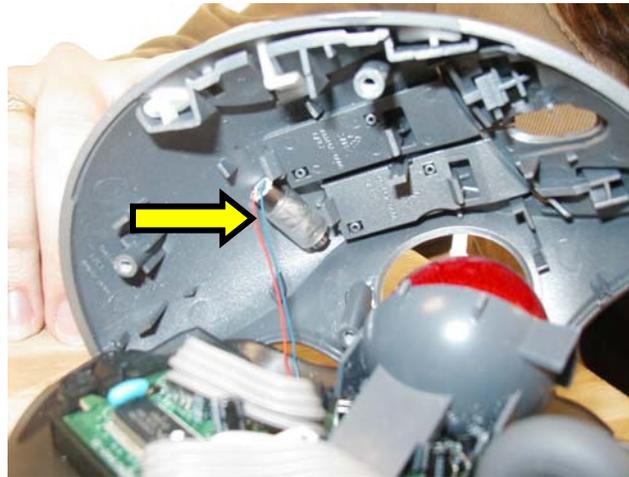


Figure 8. Vibrating motor mounted inside trackball mouse shell

3.4. Circuit Schematic

Figure 9 shows the wiring schematic for the solenoid and the vibrating motor. The PIC is powered by the USB port, and therefore can output a 5V signal. As mentioned earlier, the solenoid requires 12V and it gets this from an AC/DC converter plugged into the wall. A 5V signal from the PIC activates a transistor in series with the solenoid and turns the 12V source on and off. The vibrating motor requires 3V. This is achieved using a voltage regulator to step down the 5V from the PIC to 3V.

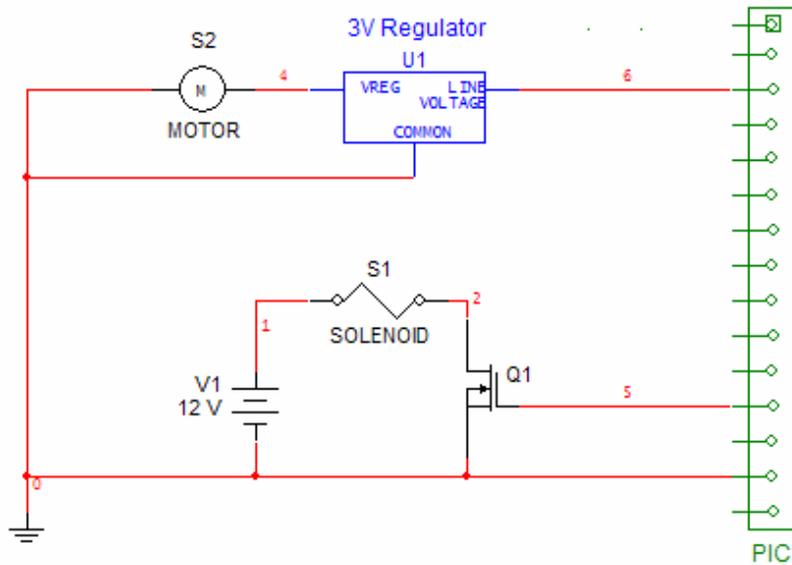


Figure 9. Circuit schematic for brake and vibrating motor

3.4. Controller Chip

There are several controllers available that include USB compatibility, meaning they come with a number of USB functions built-in. Considerations in choosing one of these chips include cost, availability, speed, reprogrammability, and difficulty of development. We chose to look at PIC microcontrollers because of their low cost and high availability. Since we only need low-speed data transfer, we first examined the PIC16C745/765, but these chips only come in one-time-programmable and UV-erasable PROM versions, neither of which would facilitate easy reprogramming. Since the 18F2455/2550/4455/4550 series chips that support full-speed data transfer also support low-speed data transfer, and they have Flash memory, which is easily reprogrammed, we decided to work with these chips instead. A USB development kit, including a development board, an In-Circuit Debugger (ICD), and an exercise booklet, was available from CCS Inc. for the PIC18F4550, and the Engineering Department already had the software necessary to program the chip using this board. Programming is done in the C language, which we also used in other parts of this project; this was better than having to learn an additional programming language to work with the microcontroller.

The software components of this project include the program for our PIC microcontroller, the code on the computer that allows the computer to communicate with the device, the graphical user interface (GUI) used to test our device, and an interface that allows the GUI to call the code that communicates with the PIC. We also make use of a speech synthesis package that enables our GUI to provide audio feedback (See Figure 1). In addition, we began work on code that interacts with the windowing system.

4.1. PIC Software

In addition to sending information to the computer about mouse motion and mouse clicks, as an ordinary mouse does, we want our mouse to activate a brake and vibrating motor according to commands sent over USB from the computer. We decided that the existing functionality of the mouse would continue to be handled by the trackball mouse's built-in controller; the added functions would be performed by a second microcontroller. Thus, this second microcontroller must be able to receive data from the computer using USB interrupt transfers, turn the brake on and off, and turn the motor on for a short period of time.

The microcontroller chosen for this project, a PIC 18F4550, has built-in functions that handle the complicated low-level details of the USB protocol. To receive commands from the host computer, our program must initialize the PIC's USB functionality, ensure that the device is enumerated, and get data sent by the host. Initialization can be accomplished by calling the built-in function `usb_init`. The built-in function `usb_enumerated` returns true or false, depending on whether enumeration (the process that allows the computer to establish what device has been attached and how to communicate with it) has been completed successfully. The function `usb_kbhit` can be used to check if a new packet of data has arrived from the host, and if one has, `usb_get_packet` can be used to retrieve it. The code written for our microcontroller is provided in Appendix A. For more background on the USB protocol, please refer to section 2.

The brake is quite simple – when a message is received from the host that the brake should be turned on, it should go on, and when a message is received that it should be turned off, it should go off. The motor is slightly more complicated. When a message is

received from the computer to turn the motor on, it should go on for a specified amount of time, T , and then go off. In fact, in order to differentiate between icons and window edges, we need both short and long vibration times (T_S and T_L). After experimentation, we chose $T_S = 200$ ms and $T_L = 500$ ms.

The program that we wrote executes an infinite loop, constantly checking if the device has been enumerated and if an interrupt packet has arrived. This packet includes two 8-bit integers that are treated as Boolean variables. Table 1 indicates which input signals corresponded to which actions. We also allow the brake to be turned off by pressing a pushbutton on the PIC development board. (See Appendix A for our microcontroller code).

Table 1. PIC input signal and corresponding action

Input Signal	Action
00	Turn off brake and motor
01	Short motor vibration (200ms)
10	Long motor vibration (500ms)
11	Brake

4.2. Host to Device Communication

We initially assumed that communication with our device would require writing a device driver (a driver is a program that handles the interaction between the computer and a given device). However, in the process of researching the existing Linux-USB driver framework, we found a driver, `hiddev.c`, that we thought might work. We successfully created a test program that used the `hiddev_read` function to get data sent from the PIC, but when we went to extend our program and allow it to send data to the device, we discovered that the symmetric `hiddev_write` function had not been implemented.

At about this time, however, we discovered the open source library projects `libusb` and `libhid`. Both are written in C and are currently in development. `libhid` extends `libusb` and provides among others, functions that detach a device from the kernel driver (in other words, allow the host-device interaction be handled by our program instead of the operating system's default driver), and perform interrupt reads and writes. We had some problems installing these libraries due to a synchronization issue between the engineering server, where we were storing our programs, and the local machine to which our USB device was connected. This issue was resolved by moving the source files for each library to the local

machine before building. We also had to run the configuration script for each with a flag disabling locks. The fact that these libraries are currently in development and have somewhat limited documentation available made working with them more difficult. In the case of libhid, we had to take code from a newer version of the source file hid_exchange.c (from the repository used by the developers) and combine it with the code from the most recent release of the library in order to get it to compile with the functions we needed to use.

To access our mouse device using the libhid functions without having root privileges, we needed to change the operating system's permissions settings for the device. Since the operating system creates a "file" interface for a device whenever it is plugged in, we need to change the permissions at this time. This can be done by creating a hotplug script that runs when a new USB device is connected to the system (hotplug is what allows a device to be used as soon as it is attached to the computer; for computers that do not support hotplugging, a device will only be recognized if it is attached when the computer boots). For devices with a certain vendor and product ID, (vendor and product ID are part of the device descriptor in the USB protocol – for the PIC18F4550, a default vendor and product ID are provided), this script changes the permissions to allow the owner of the device (root) and the owner's group to access it; the script also changes root's effective group in this case to "smouse." (See Appendix D for a copy of this hotplug script). Any user added to the group "smouse" in the file /etc/group can then use the device.

After successfully installing libusb and libhid and handling our problem with permissions, we worked from an example program written by Arnaud Quette, one of the developers of libhid, to write a simple test program to send and receive data from our mouse. We found that there was a one cycle lag in the data read in this manner; when requesting information about whether a button on the PIC development board is pressed, the first request for information after pressing the button returns that it is not pressed, the second request correctly indicates that it is. However, since we decided to use an existing mouse button for our brake release, and information about the motion of the mouse is being relayed by the mouse's built-in controller, we do not need to read information from our PIC. The portion of this test program that sent data from the computer to the PIC worked well, so we transformed it into a function which could then be called from other programs. This function takes two characters (8 bit integers) as arguments and writes them to the PIC, which then performs the appropriate action according to the code in Table 1.

4.3. Text-to-Speech Synthesis System

To implement the audio feedback component of the mouse, we needed a speech synthesis system to convert text to speech. We first tried to use the Festival Speech Synthesis System, but ran into a plethora of problems. Festival was developed by the Centre for Speech Technology Research at the University of Edinburgh. We chose Festival for several reasons: it is free, open source, and relatively well known. It is written in C++ and is controlled by a Scheme (SIOD) based command interpreter. There are a number of different voices available in Festival, including English, American, and Spanish speakers. Festival was also attractive because it offers text to speech conversion through several application program interfaces (APIs), including C++. This allowed us to perform text-to-speech conversion as a function call in our higher level program.

The newest version of Festival available for download is a beta version of 1.95. In our first attempt to use it, we downloaded and installed this beta-version. It worked successfully using the Scheme controller, but in order to integrate it with the rest of the mouse we needed to embed it as a function in a higher level program. Originally we wanted to write a program in Java that could call Festival because our graphical user interface (GUI) was developed in Java. However, we found that the implementation of Festival's Java API was incomplete and the documentation provided was confusing. Next we tried writing a simple program in C++ that called Festival as a function, but we could not get it to compile. As Festival was written over 5 years ago, it was developed using an older version of C++. Due to changes in the C++ language and compiler, there was a long list of warnings related to deprecated features. We thought that if we tried downloading a newer version of gcc (our C/C++ compiler), our program would compile. However, the attempts in updating the gcc resulted in having to re-image the machine we were working on. After the computer was re-imaged, we discovered that version 1.4.3 was already installed on the machine. By linking in additional libraries, the C++ test program that used Festival functions compiled.

After finally getting the program to compile, we discovered that the voices were extremely high-pitched and fast. We were able to slow the voices down by changing a setting in the speech synthesizer, but we were not able to change the pitch. A small amount of online research convinced us that the pitch problem might have to do with an incompatibility between Festival 1.4.3 and the sound card on our system.

Setting aside that issue, we converted our test program into a pair of functions that could be called by our GUI through our C-Java interface (See section 4.5). However, when we ran a test program for these, it had a segmentation fault. By calling the Festival initialization function at the very beginning of our program, before any other functions were called, we were able to get the program to work. Calling these functions through the C-Java interface still produced a segmentation fault though, and we concluded that this was due to the fact that the Festival initialization function had to be the first function called in any program using Festival features, and with the C-Java conversion, we could not call this function first. We therefore abandoned Festival as the speech synthesizer for this project.

Instead of Festival, we chose FreeTTS, a speech synthesis system written in Java. With FreeTTS, as with Festival, speech can be generated by passing in a string of text; a sound file containing the system's best guess of how to pronounce this text is created and played. Obviously, this speech will only make sense if the text contains easy-to-pronounce words.

4.4. GUI in Java

In order to test our system in a controlled environment, we needed to write a graphical user interface (GUI) test program. By creating a GUI, we could control what objects appear on the screen, track the user's movement about the screen, and label the objects on the screen with understandable names. In the desktop environment, objects such as icons and windows are not guaranteed to have meaningful names. Clear and specific names are necessary because the user must be able to identify them using the speech feedback feature.

We decided to write this program in Java in order to take advantage of the tools available in the `javax.swing` package. This package includes objects such as frames, panels, icons, labels and buttons (See Figure 10). Swing provides several layout managers; we chose the `GridBagLayout`, the most flexible of these, which creates a grid with variable width rows and columns, within which the designer can specify, among other parameters, the grid location of an object's top corner and the number of rows or columns it should span.

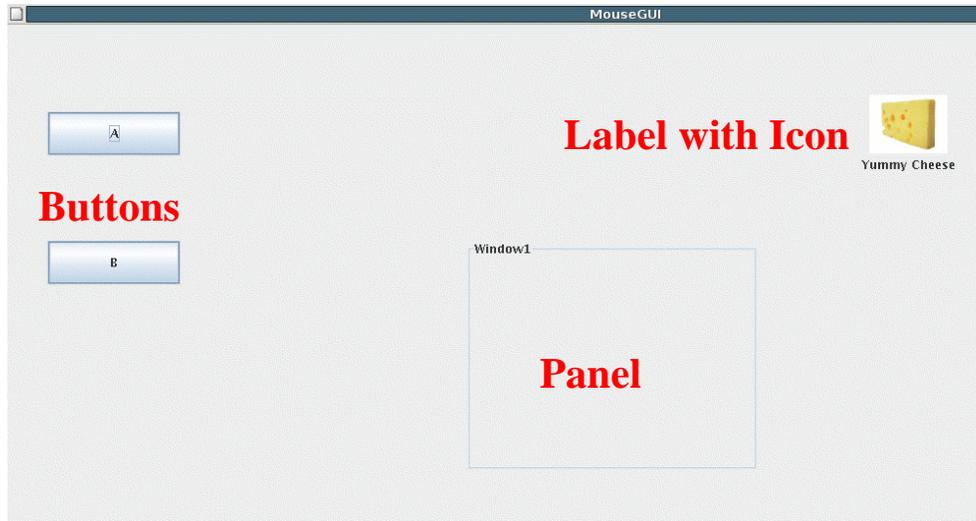


Figure 10: Some of the objects available in the javax.swing package

In Java, objects have fields and methods; fields are variables that can be set to specific values and methods are functions that can be called. The GUIs that we created consist of several windows (panels), buttons, and icons with labels. Each of the objects uses a “MouseListener” to detect when the cursor enters and exits the object and when the mouse is clicked on it. MouseListener is an interface. An interface is a template for an object that specifies what methods the object must have in order to implement the interface. By adding a MouseListener interface to each of the objects on the screen, we can monitor the events (entrances, exits, and mouse clicks) that occur on the object and specify what should be done when those events occur. We chose to have a short vibration occur when the mouse enters or exits a panel, a long vibration occur when the mouse passes over an icon or a button, and the brake activate when the mouse comes near to the edge of the GUI window. This last was accomplished by placing all components on a borderless panel, which was then placed, with some amount of padding, on an outer panel. We could then detect when the mouse passed from the main panel to the outer panel. Also, in order to keep track of how a user moves the mouse around the screen and how often they use the speech feedback feature, every event is recorded in a text file with a time stamp. A screenshot of our test GUI is shown in Figure 11. Appendix C contains the code for two GUIs that we created.

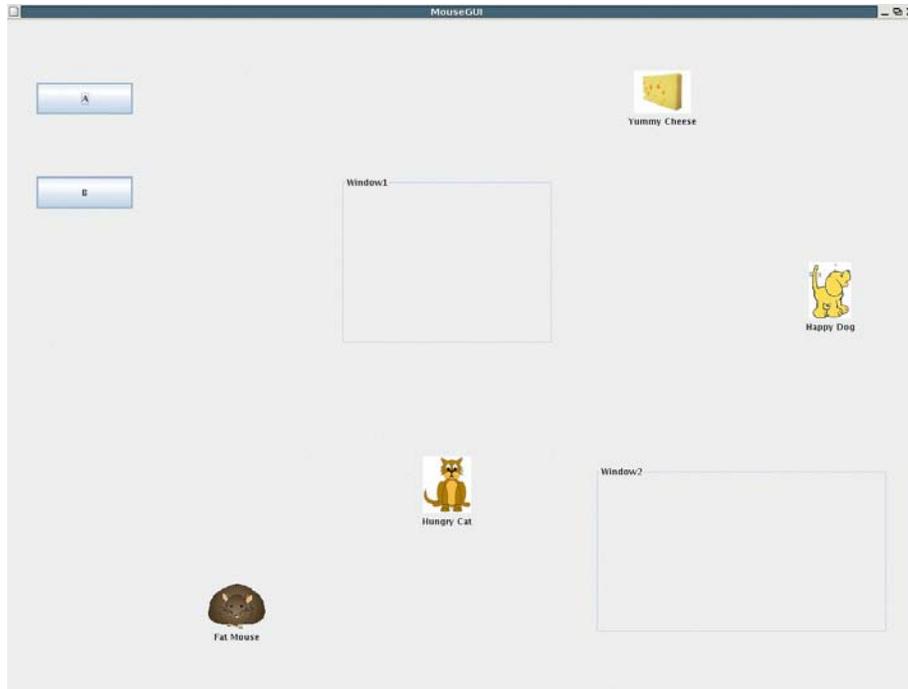


Figure 11: Screenshot of our GUI

4.5. C-Java Interface

Because our communication with our PIC board is written in C, and our GUI is written in Java, we need an interface between the two. For this, we used the Java Native Interface (JNI). We followed the procedure outlined in the paper “Calling C Library Routines from Java,” written by the UK-based non-profit Numerical Algorithms Group (NAG). This involves creating a shared library in C, several interface files in both Java and C, and a Java class that calls the desired functions. We were then able to call these functions from our GUI by creating an instance of this class. (See Appendix D for C-Java interface code).

4.6. X Window Interaction

We began work on software to allow the mouse to interact with the X Window system, which is a necessary step if we want our mouse to work in the general desktop environment. X Window is the windowing system used by Linux. It handles drawing windows on the screen and interacting with the mouse and keyboard. Ultimately, we want the mouse to be a tool that can identify visual elements outside of our GUI program, such as icons on the desktop or the edge of a window. This is accomplished by monitoring the

position of the pointer and the events that occur when the pointer passes over icons and windows on the desktop. X library functions can be used to get the mouse position and detect mouse events. We found a program online (Xquerypointer.c) that monitors the position of the pointer using a function called XQueryPointer, which returns the coordinates of the pointer in the root window. Several sources warned that if the function is used constantly, it would crash. We modified the program such that it is constantly querying the location of the pointer and printing its coordinates, to test if it would work or crash. The program worked successfully. Currently, the program exits if the pointer reaches the edge of the screen. We could alter this program so that instead of exiting when the cursor reaches the edge of the screen, it activates the brake. Other event handling functions could be used to signal when to activate the vibrating motor and speech feedback.

5 Testing

5.1. Design of Experiments

Two experiments were designed to test the functionality and effectiveness of the mouse. These were performed using the GUIs we created. In order to draw in participants for the tests, we recruited classmates, friends, and professors and offered a candy bar in exchange for 15 minutes to an hour of their time. In total, we tested 23 subjects, one of whom was blind, and 6 of whom were trackball users.



Figure 12. Blindfolded test subject using mouse

5.1.1. Test 1

The first experiment involved blindfolding the user and asking him or her to complete three tasks of increasing difficulty, twice. For one set, the user was allowed only

speech feedback, and for the other the user was allowed both speech and haptic (vibrating and braking) feedback. We alternated the order of the sets to avoid the bias of a learning curve. Each task was timed, and the speech only and speech and haptic feedback times were compared. Before the user began the test, we asked him or her to practice moving the cursor up and down and right-to-left on the screen so that he or she could get a feel for how trackball movement translated to cursor movement. In addition, the user was allowed to see the screen and familiarize him or herself with the positions of the objects on the screen before the start of the test; this experiment was not intended to test the user's memory. If at any point during the test the user could not remember where an object was located, he or she was permitted to look at a printed screen shot, but not at the screen. For our blind test subject we created a tactile screen shot using layered poster board, as shown in Figure 13. The blind user familiarized herself with objects on the screen by feeling around the tactile screen as we identified the names of each object for her. She also commented afterwards that tactile screenshots (like this) for common applications like Microsoft Word and Mozilla Firefox, would be very helpful in learning to use their interfaces since she has no concept of how the screen is setup or where to look for commonly used buttons and menus.

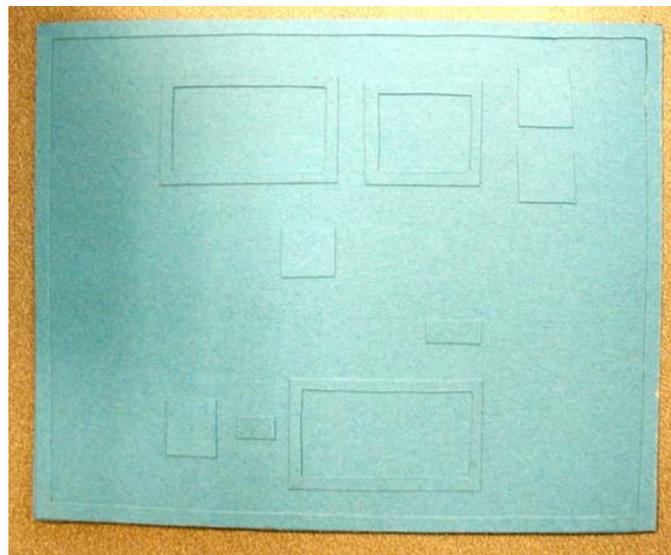


Figure 13. Tactile screen shot for blind test subject

In each task the cursor started in the top left hand corner of the screen. For the first task, the user had to click inside window 1 and then inside window 2. The second task involved clicking inside window 1, clicking on the internet icon, and then clicking on window 3. The final and most difficult task required the user to click on button Y, the

garbage icon, back on button Y, the word processor icon, and button Y for a third time. Figure 14 is a screen shot of the GUI used for this test and an illustration of each of the tasks.

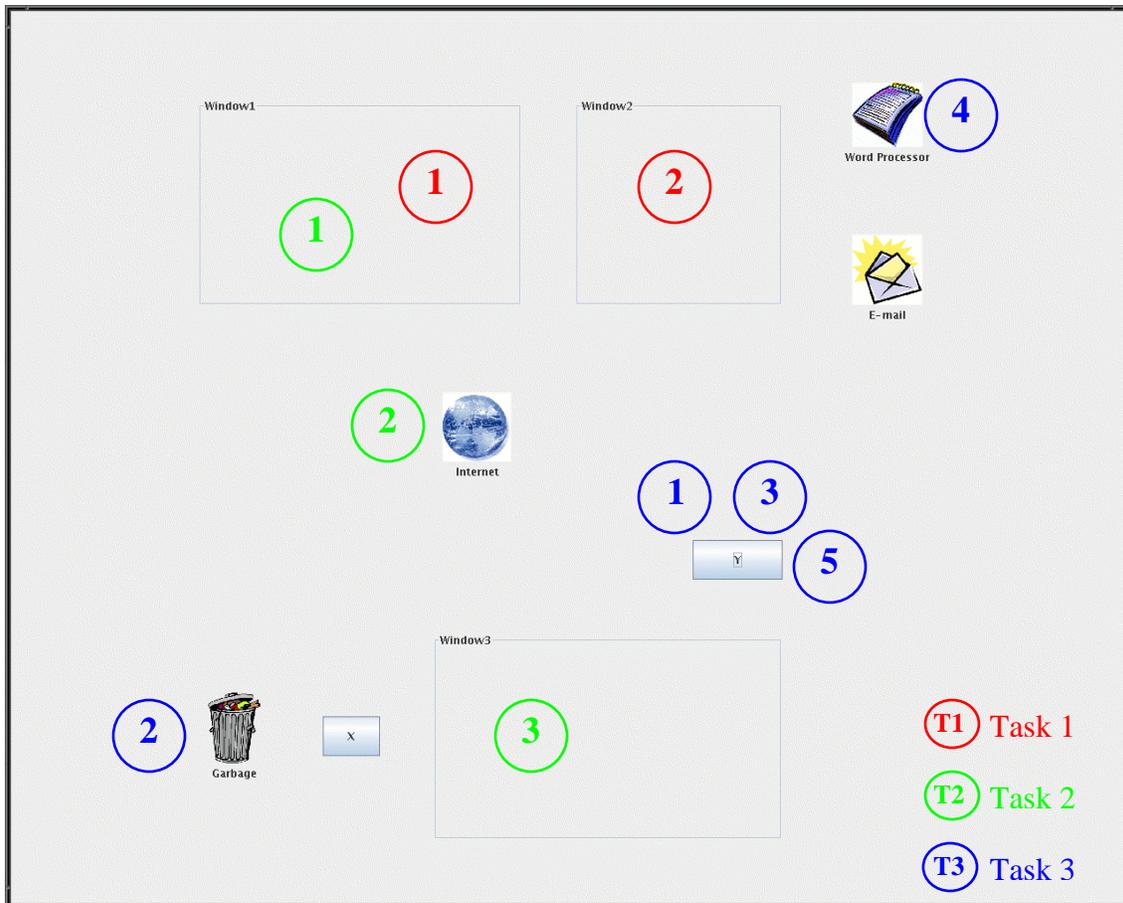


Figure 14. GUI used for test 1

5.1.2. Test 2

The second experiment involves blindfolding the user and asking him or her to explore the GUI for a set amount of time using the haptically enhanced mouse. Then the user is asked to sketch a drawing of what he or she thinks the screen looks like, including as many of the names of the icons and windows as he or she can remember. For our blind test subject we cut out buttons, icons, and windows from poster board that she could arrange on a blank screen. The “tactile sketching” materials are shown in Figure 15.

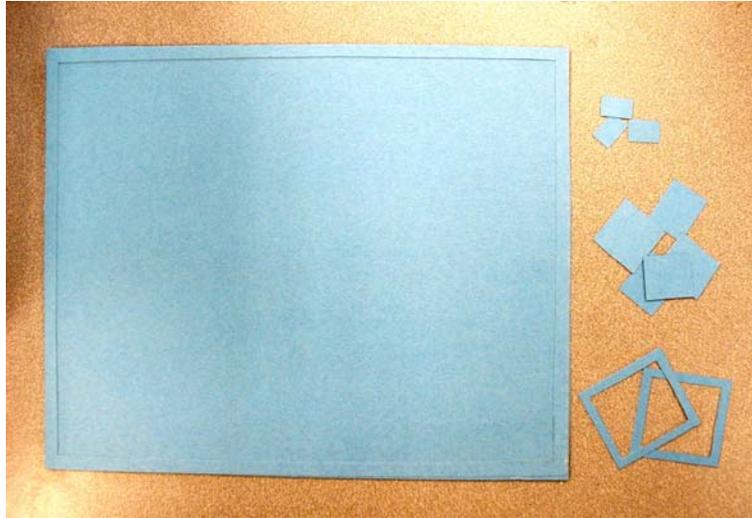


Figure 15. Tactile sketching for blind test subject

The results of this experiment were purely qualitative, and they were useful in understanding if the user could perceive accurate spatial relationships between objects with the aid of the mouse. The GUI used in this experiment is shown in Figure 16.

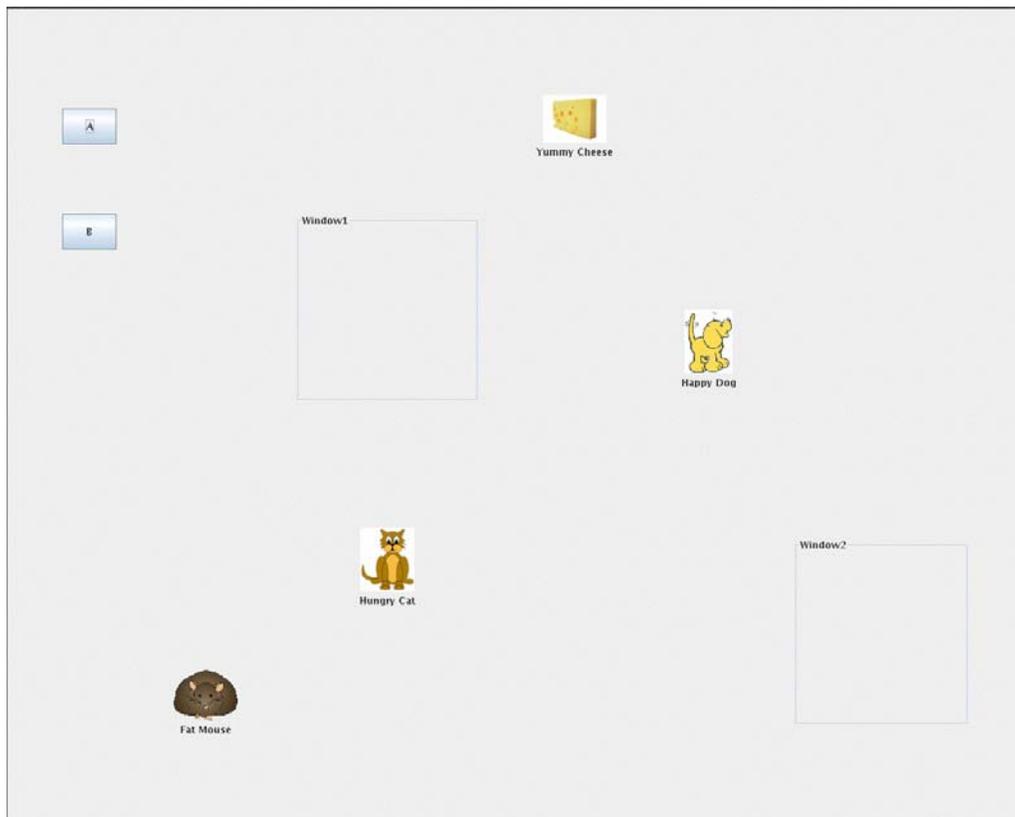


Figure 16. GUI used in test 2

5.2. Results

5.2.1. Test 1

Table 2 shows the average time and standard deviation for each task with speech feedback and with speech and haptic feedback. Also listed is the percent improvement of the averages for each task. As the numbers indicate, the average time for each task improved significantly when haptic features were available. It is important to note, however, not everyone improved, but most did. As the standard deviations suggest, there was a huge range of ability due to influencing factors such as the user's ability to adapt to using the trackball, the strategy the user chose to search for an object, the user's ability to combat frustration, and the effects of blind luck. An extension of this experiment that might diminish the effects of luck would involve having each subject perform the tasks several times. Due to time constraints, we were not able to attempt this extension.

Table 2. Average test 1 times for all subjects (N* = 23)

Task	Time (seconds)				Improvement
	Speech only		Speech and haptic		
	Average time	Standard deviation	Average time	Standard deviation	
Task 1	46.4	37.9	35.3	22.3	23.9%
Task 2	130.2	108.0	61.8	40.2	52.5%
Task 3	465	302	238	175.7	48.8%

*N = sample size

We tested one blind subject and her results are presented in Table 3. The subject does not use a mouse on a regular basis and had never used a trackball mouse. The percent improvement for the blind subject is greater than the average in every task. While this is clearly not conclusive because it is based on one person's data, it is interesting and it does suggest that blind users who are more accustomed to focusing on their sense of touch can use the mouse more effectively than blind-folded, sighted users.

Table 4 compares trackball users to non-trackball users in terms of their average time on each task and the number of speech requests they made. The average time is lower for trackball users than non-trackball users in every task, as well as the number of speech requests. This was expected, as we often found that non-trackball users had an extremely difficult time adjusting to using the trackball. One possible reason is that orthogonal axes on

the computer screen do not correspond to orthogonal axes on the trackball. Thus, for many users the intuitive movement of the thumb on the trackball to move the cursor up and down and left to right on the screen instead caused the cursor to move diagonally across the screen. In addition, non-trackball users had a hard time judging the sensitivity of the trackball and distances on the screen. Some would barely move the cursor at all before they requested speech feedback, while others would zoom from one edge to the other. The averages presented in Table 4 imply that experience using a trackball mouse significantly affects the speed at which the user can locate objects on the screen. The results also suggest that as a users gain experience using this mouse, their speed at finding objects increases and their accuracy in navigating the screen improves. The raw data for this test can be found in Appendix E.

Table 3. Blind test subject's results

Task	Time (seconds)		Improvement
	Speech only	Speech and haptic	
Task 1	151	11	92.7%
Task 2	114	36	68.40%
Task 3	547	200	63.40%

Table 4. Trackball vs. non-trackball users (with speech and haptic feedback)

Task	Time (seconds)				Number of speech requests			
	Trackball users (N = 6)		Non-trackball users (N = 17)		Trackball users (N = 6)		Non-trackball users (N = 17)	
	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation
Task 1	23.0	20.3	38.9	21.5	3.20	1.166	5.94	3.88
Task 2	46.0	30.7	67.4	42.5	7.67	5.24	12.88	10.22
Task 3	194.2	78.4	253	198.9	29.0	12.51	53.4	59.5

5.2.2. Test 2

The goal of Test 2 was to determine if the user could create a mental picture of a screen, using only the speech and haptic feedback provided by our mouse. Twelve of our test subjects completed this task: seven sighted non-trackball users, four sighted trackball

users and one blind user (Appendix F shows each user's drawing). Two of these users were allowed to explore for 15 minutes, all others had 5 minutes.

The users' success in correctly identifying spatial relationships for the objects on the screen in this task varied widely, but only one user failed to capture any of the spatial relationships present (Non-Trackball User 7). Users allowed to explore for 15 minutes found all the objects, those who were only allowed 5 minutes did not. Non-Trackball User 1 did in fact find the cheese icon, but forgot about it before drawing the screen. This highlights how memory affects the results of this task – some of the inaccuracies in the drawings may be due to memory problems and not perception problems. Most users ascertained that Window 1 was to the left of Window 2 or above it, or both. Only one user, (Non-Trackball User 3), missed a window, but many missed some buttons or icons, which is understandable since the windows were much larger than buttons or icons. The mouse icon, located in the lower left corner, was missed most often, whereas the cat, a centrally located icon, (which was actually larger than it appears in the screenshot because the object is padded with blank space), was never missed. Each button was missed as often as the mouse icon; in most cases users who found one button found the other, and the relative position of the two buttons was the most accurately drawn of any spatial relationship. The high number of users who failed to find the buttons and the mouse icon shows that objects in the corners were most likely to be missed. Also, all three of these objects are on the left edge of the screen, which may indicate that it is harder for the user to move to the left (moving the thumb away from the hand), than to the right (moving the thumb toward the hand). This problem could be eliminated if a symmetrical trackball controlled by the fingers were to be used.

6 Conclusions and Future Work

In its current state, our haptic mouse successfully functions within the GUI that we created to test it. The results of the tests confirm that haptic features can improve a blind user's experience of a graphical environment. Test 1 demonstrates this with faster location of objects on the screen using haptic features and test 2 illustrates that blind users can determine basic spatial arrangements of objects using tactile feedback. In addition, these tests also demonstrate that translation of mouse movement to cursor movement has a significant effect on user perceptions. When we decided to use a trackball mouse as

opposed to a normal mouse when we began this project in the fall, we did not take into consideration the ease of use or the learning curve for a trackball mouse compared to a standard mouse. We did not realize that the axes on this trackball are not orthogonal and consequently difficult and confusing for new users to adapt to. We also failed to consider the sensitivity of the trackball and the acceleration of the cursor, two parameters that can be set on the computer.

While the mouse is functional and works within our GUI, there are many improvements that can be made that would increase the capability of our mouse tremendously. In terms of hardware, the brake design can be improved so that it will provide more resistance. One way this can be accomplished is by introducing a coil spring so that the shaft is not completely rigid and the brake pads can form to fit the surface of the trackball even more closely. Using three points of contact instead of two could also improve the performance of the brake.

The physical appearance and structure of the mouse can be improved by consolidating the mouse body. The mouse box is clearly too large for the size of the components it needs to house. A smaller, sleeker, and more stylish design is necessary if this mouse is to be marketable.

The most obvious software improvement for this mouse is to write the code that would allow the mouse to function in the desktop environment (outside of our GUI). We began work on allowing the mouse to interact with the X Window system, but did not have time to complete the task this semester. Ultimately, it would be great if the mouse were Microsoft Windows compatible, since Windows is an extremely popular operating system.

Many of our test subjects also made suggestions for improvements for our mouse. One suggested that a different vibration be used for movement in versus out of a window. Another commented that it would be very useful if the left, right, top, and bottom edges of the screen could be differentiated using the speech feedback. Anticipation of vibrations might also improve performance. Often if a user moved the cursor quickly across the screen and passed over an icon, the cursor would be far past the icon by the time the vibration indicating its presence finished. It would also be interesting to experiment with the sensitivity of the trackball and the acceleration of the cursor to see what effects those parameters have on the performance of the mouse.

While the performance of blindfolded test subjects provides some indication of how a device might work for blind users, the real measure of its success lies in what blind users think. Our blind test subject explained to us her frustration with the difference between the way in which the computer interface is presented to her when she uses a screen reader and the way in which it is presented to her sighted friends. She was excited about the ability of devices like our mouse to finally show her what a friend meant when he talked about the icon in the top left corner of the screen. We hope that future engineering students will extend our project so that it works in a desktop environment and gives users like her better access to the graphical world of modern computing.

7 Acknowledgements

We would like to thank the Swarthmore College Department of Engineering, especially our advisor Professor Cheever, Professor Maxwell, Professor Orthlieb, Professor Siddiqui, Professor Everbach, Grant Smith, Edmond Jaoudi, and Holly Castleman. The Swarthmore College Computer Science Department, especially Professor Newhall, Professor Kuperman and Professor Wicentowski, also provided a great deal of help. We would also like to acknowledge everyone who participated in our tests, especially Laura Wolk, and the Festival, Linux-USB, and libhid developers who provided very helpful replies to our emails.

8 References

- Axelson, Jan. 2001. *USB Complete*. 2nd Ed. Lakeview Research. Madison, WI.
- “Creating a GUI with JFC/Swing.” *The Java Tutorial*. 2000. Sun Microsystems, Inc. <http://java.sun.com/docs/books/tutorial/uiswing/> 4 May 2006.
- Custom Computer Services, Inc.* <http://www.ccsinfo.com>. 11 Nov. 2005.
- Jameco Electronics.* <http://www.jameco.com>. 11 Nov. 2005.
- Krafft, Martin F. “External: libphidgets/hotplug/phidgets.hotplug.” *Libphidgets*. 2003 – 2005. <http://cvs.ailab.ch/cgi-bin/viewcvs.cgi/external/libphidgets/hotplug/phidgets.hotplug?rev=1.4&view=auto> 4 May 2006.
- Microchip Technology Inc.* <http://www.microchip.com>. 11 Nov. 2005.
- Mouser Electronics.* <http://www.mouser.com>. 11 Nov. 2005.

Numerical Algorithms Group. “Calling C Library Routines from Java.”
<http://www.nag.co.uk/IndustryArticles/CallingCLibraryRoutinesfromJava.pdf> 4
May 2006.

Rinne, Karl. Lecture Slides. *ET4508: Computer Systems Architecture*. University of Limerick.
Feb.-May 2005.

Verling, Eoin. “Development of a User-Space Application for an HID Device, Using
libhid.” *Linux Journal*. 26 August 2005. <http://www.linuxjournal.com/article/8145> 4
May 2006.

“Xquerypointer.c.” *XS4ALL*. [http://www.xs4all.nl/~hanb/software/
xquerypointer.c](http://www.xs4all.nl/~hanb/software/xquerypointer.c). 4 May 2006.

9 Appendices

Appendix A: PIC microcontroller code

Appendix B: Host to device communication code

Appendix C: GUI code

Appendix D: Other code

Appendix E: Test 1 Data

Appendix F: Test 2 Drawings

Appendix A: PIC Microcontroller Code

- Header file
- Main program

Header file:

```
////////////////////////////////////
//CCSUSB.H
//This file is provided in the
//example booklet that comes with
//the PIC18F4550 development kit
//available from Custom Computer
//Services (CCS).
//
//Modifications were added by
//Lauren Stadler and Heather Jones
//for an engineering senior design
//project.
////////////////////////////////////

//Include the header file for this device
#include <18F4550.h>
//Settings
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
#use delay(clock=4800000)

//Define pins and functions
#define LED1 PIN_A5
#define LED2 PIN_B4
#define LED3 PIN_B5
#define BUTTON PIN_A4
#define LED_ON output_low
#define LED_OFF output_high
//Special pin definitions for haptic mouse project
#define MOTOR PIN_E0
#define BRAKE PIN_E1
#define GROUND PIN_B6
```

Main program:

```
////////////////////////////////////
//mouseCon.c
//Written by Lauren Stadler and Heather Jones
//Engineering Senior Design Project, Spring 2006
//-----
//This program runs on a PIC18F4550 to control
//the brake and motor for a haptically enhanced
//trackball mouse, according to signals sent
//from the computer. A two byte packet is
//read, and this program performs the following
//actions depending on the value of each byte:
//
//receive 00: turn off brake and motor
//receive 01: turn on motor for 200 ms
//receive 10: turn on motor for 500 ms
//receive 11: turn on brake
//
//A button on the PIC development board can
//also be used to switch off the brake.
////////////////////////////////////

#include "CCSUSB.H"
#define USB_EP1_TX_ENABLE USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE 8

#define USB_EP1_RX_ENABLE USB_ENABLE_INTERRUPT
#define USB_EP1_RX_SIZE 8

#include <pic18_usb.h>
```

```

#include <usb_desc_hid.h>
#include <usb.c>

void main(void) {
    //Counter for timing motors
    int16 Mdelay=0;
    //Input buffer
    int8 in[2];
    //Motor and brake flags
    int8 M1=0;
    int8 M2=0;
    int8 B=0;

    //Set a pin low to serve as a reference for
    //external circuitry
    output_low(GROUND);

    //Initialize USB
    usb_init();

    //loop continuously
    while (TRUE) {
        //Light the green LED to indicate that enumeration
        //has been performed correctly
        if (usb_enumerated()) {
            LED_ON(LED1);
            //If there is new data to get, get it and
            //place it in "in"
            if (usb_kbhit(1)) {
                //get a packet from endpoint 1, put it in "in," get 2 bytes
                usb_get_packet(1,in,2);

                //Signal sent is 10 - long motor vibration (500 ms)
                //set flag M1
                if (in[0] && !in[1]) {
                    M1 = 1;
                }
                else M1 = 0;
                //Signal sent is 01 - short motor vibration (200 ms)
                //set flag M2
                if (!in[0] && in[1]) {
                    M2 = 1;
                }
                else M2 = 0;
                //Signal sent is 11 - brake on
                //set flag B
                if (in[0] && in[1]) {
                    B = 1;
                }
                else B = 0;
            }

            //If button on development board is pressed (low),
            //clear flag B
            if (!input(BUTTON)) {B = 0;}

            //If flag B is set, turn on brake and red LED, otherwise turn them off
            if (B) {
                LED_ON(LED3);
                output_high(BRAKE);
            }
            else {
                LED_OFF(LED3);
                output_low(BRAKE);
            }

            //If flag M1 is set, turn on motor and yellow LED while Mdelay counts
            //to 500, then clear flag M1 and reset Mdelay
            if (M1) {
                Mdelay++;
            }
        }
    }
}

```

```

        LED_ON(LED2);
        output_high(MOTOR);
        if (Mdelay >= 500) {
            M1 = 0;
            Mdelay = 0;
        }
    }
    //If flag M2 is set, turn on motor and yellow LED while Mdelay counts
    //to 200, then clear flag M2 and reset Mdelay
    else if(M2){
        Mdelay++;
        LED_ON(LED2);
        output_high(MOTOR);
        if(Mdelay >= 200) {
            M2 = 0;
            Mdelay = 0;
        }
    }
    //If neither M1 nor M2 is set, turn off motor and yellow LED
    else {
        LED_OFF(LED2);
        output_low(MOTOR);
    }

    //Each run through loop should take 1 ms
    delay_ms(1);
}
//If this device has not been successfully enumerated yet, turn off
//the green LED
else
    LED_OFF(LED1);
}
}

```

Appendix B: Host to Device Communication Code

- Header file
- Function definitions
- Test program

Header file

```
/*
 * mouselib.h
 * Written by Heather Jones and Lauren Stadler
 * Engineering Senior Design Project, Spring 2006
 * -----
 * This header file provides function declarations
 * and some constant definitions for mouselib.c
 */

#ifndef __MOUSELIB_H__
#define __MOUSELIB_H__

#include <usb.h>
#include <hidtypes.h>
#include <stdio.h>
#include <string>
#include <hid.h>

//The std namespace allows us to use C++ strings
using namespace std;

//Vendor and product ID for the PIC development board
#define MOUSE_VENDOR 0x0461
#define MOUSE_PRODUCT 0x0020

//Path length, determined using hid_dump_tree()
#define PATH_LEN 2

//We write two bytes of data to the mouse
#define WRITE_PACKET_LEN 2

//Host to device communication
int mouseWrite(char arg1, char arg2);

//Speech, using Festival
int mouseInitSpeech(void);
int mouseSpeak(string str);

#endif /* __MOUSELIB_H__ */
```

Function definitions

```
/*
 * mouselib.c
 * Written by Heather Jones and Lauren Stadler
 * Engineering Senior Design Project, Spring 2006
 * -----
 * This file includes a write function used to send
 * data to the microcontroller used in our haptic
 * mouse project. It also includes two speech-related
 * functions that were not used in the final project
 * due to incompatibilities with other project code.
 */

#include "mouselib.h"
#include <iostream>
#include "/usr/include/festival/festival.h"
#include <fstream>

using namespace std;
```

```

/*
Function: mouseWrite
//-----
This function takes two character arguments and
sends these over USB, using an interrupt transfer,
to the device with the vendor and product ID
specified in mousselib.h (the PIC18F4550 development
board). If any of the libhid functions called from
this function fail, the name of the failing function
and the error number will be printed to stderr.

arg1: the first byte to send
arg2: the second byte to send

returns:0 on success, 1 on error

*/
int mouseWrite(char arg1, char arg2){
    HIDInterface* hid;
    hid_return ret;
    HIDInterfaceMatcher matcher = {MOUSE_VENDOR, MOUSE_PRODUCT, NULL, NULL, 0};
    char write_packet[WRITE_PACKET_LEN];
    int PATH_IN[PATH_LEN];

    //Path determined using hid_dump_tree()
    PATH_IN[0] = 0xff000001;
    PATH_IN[1] = 0x00000000;

    write_packet[0] = arg1;
    write_packet[1] = arg2;

    /*We don't want to print out all the debugging information*/
    hid_set_debug(HID_DEBUG_NONE);
    hid_set_debug_stream(stderr);
    hid_set_usb_debug(0);

    /*Inialization for libhid functions*/
    ret = hid_init();
    if(ret != HID_RET_SUCCESS){
        fprintf(stderr, "hid_init failed with return code %d\n", ret);
        return 1;
    }

    /*Create a new HIDInterface*/
    hid = hid_new_HIDInterface();
    if(hid == 0){
        fprintf(stderr, "hid_new_HIDInterface() failed, out of memory?\n");
        return 1;
    }

    /*Open the device with the vendor and product ID specified in matcher*/
    ret = hid_force_open(hid, 0, &matcher, 3);
    if(ret != HID_RET_SUCCESS){
        fprintf(stderr, "hid_force_open failed with return code %d\n", ret);
        return 1;
    }

    /*Perform the write*/
    ret = hid_interrupt_write(hid, 1, write_packet, WRITE_PACKET_LEN, 1000);
    if(ret != HID_RET_SUCCESS){
        fprintf(stderr, "hid_interrupt_write failed with return code %d\n", ret);
        return 1;
    }

    /*Close the interface and clean up*/
    ret = hid_close(hid);
    if (ret != HID_RET_SUCCESS) {
        fprintf(stderr, "hid_close failed with return code %d\n", ret);
        return 1;
    }
}

```

```

hid_delete_HIDInterface(&hid);
ret = hid_cleanup();
if (ret != HID_RET_SUCCESS) {
    fprintf(stderr, "hid_cleanup failed with return code %d\n", ret);
    return 1;
}

return 0;
}

/*
These two functions use festival for speech synthesis. They were written
as functions in order to allow them to be put into a shared library and
called through a C-Java interface from a Java GUI. However, because
festival_initialize must be called before any other function in a given
program, these functions did not work with the C-Java interface. They
do work when mouseInitSpeech is the first function called in a C++
program.
*/

/*
Function: mouseInitSpeech
//-----
This function initializes the festival speech
synthesis system, sets the voice, and changes the
Duration_Stretch - this last was needed because the
voice was initially very high pitched and fast -
changing the Duration_Stretch does not change the
pitch but it does slow down the voice.
*/
int mouseInitSpeech(void){
    int heap_size = 210000;
    int load_init_files = 1;

    festival_initialize(load_init_files, heap_size);
    festival_eval_command("(voice_reset)");
    festival_eval_command("(voice_kal_diphone)");
    festival_eval_command("(Parameter.set 'Duration_Stretch 3)");
    return 0;
}

/*
Function: mouseSpeak
//-----
This function takes a C++ string, converts it to an EST_String,
and calls festival_say_text to have the string spoken by
the festival speech synthesis system.

str: string of text to be spoken
*/
int mouseSpeak(string str){
    cout<<str<<endl;
    const EST_String estr = EST_String((const char *)
str.c_str(),str.size(),0,str.length());
    cout<<estr<<endl;
    festival_say_text(estr);
    return 0;
}

```

Test file

```

/*
* testmouselib.c
* Written by Heather Jones and Lauren Stadler
* Engineering Senior Design Project, Spring 2006
* -----
* This is a test program for the mouselib library it
* takes 3 arguments - the first two are characters
* which will be sent to the haptic mouse device
* using mouseWrite, the third is a string which will

```

```
* be spoken twice.
*/

#include <string>
#include <iostream>
#include "mouselib.h"

using namespace std;
int main(int argc, char* argv[]){
    string text;
    mouseInitSpeech();
    if(argc == 4){
        mouseWrite((char) atoi(argv[1]), (char) atoi(argv[2]));
        text = string(argv[3]);
        mouseSpeak(text);
        mouseSpeak(text);
        return 0;
    }
    else{
        printf("testmouselib requires 3 arguments: sendchar1 sendchar1 string\n");
        return 1;
    }
}
}
```

Appendix C: GUI Code

- MouseGUI.java – used for test 2
- MouseGUI2.java – used for test 1

MouseGUI.java

```
/*
 * MouseGUI.java
 * Written by Heather Jones and Lauren Stadler
 * Engineering Senior Design Project, Spring 2006
 * -----
 * This file contains the program MouseGUI, used to
 * test the haptically enhanced mouse project. The
 * program runs a GUI with icons, buttons and panels
 * (called windows). When mouse events are detected
 * over these objects, the GUI reacts by calling the
 * write function from Mouse.java, which allows it
 * to control the haptic mouse device, or by calling
 * a FreeTTS speech synthesizer function which allows
 * it to speak. The mouseCJavaInterface library
 * must be in the path specified by java.library.path.
 *
 */

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.ImageIcon;
import java.io.*;
import java.util.*;
import java.text.DateFormat;
import com.sun.speech.freetts.jsapi.FreeTTSEngineCentral;
import java.util.Locale;
import javax.speech.EngineList;
import javax.speech.EngineCreate;
import javax.speech.synthesis.Synthesizer;
import javax.speech.synthesis.SynthesizerModeDesc;

/* This GUI implements the interface MouseListener, which allows
it to easily obtain mouse events*/
public class MouseGUI implements MouseListener{
    /* fields that allow us to write to a file */
    static FileOutputStream fout;
    static PrintStream ps;

    /* characters to be sent to the mouse device */
    char moutput1 = 0;
    char moutput2 = 0;

    int flag = 0;
    /* used to include the timestamp for data collection */
    static String TimeStamp;

    /* used for speech synthesis */
    protected Synthesizer synthesizer;

    /* Function: createSynthesizer
    / -----
    This functions handles the setup of the
    speech synthesizer that allows the
    program to talk.
    */
    public void createSynthesizer(){
        try {
            SynthesizerModeDesc desc =
                new SynthesizerModeDesc(null,
                                        "general",
```

```

        null,
        Boolean.FALSE,
        null);

FreeTTSEngineCentral central = new FreeTTSEngineCentral();
EngineList list = central.createEngineList(desc);

if(list == null){
    System.err.println("null engine list");
    System.exit(1);
}
else{
    System.out.println("engine list size = " + list.size());
}

if (list.size() > 0) {
    EngineCreate creator = (EngineCreate) list.get(0);
    synthesizer = (Synthesizer) creator.createEngine();
}
if (synthesizer == null) {
    System.err.println("Cannot create synthesizer");
    System.exit(1);
}
synthesizer.allocate();
synthesizer.resume();

} catch (Exception e) {
    e.printStackTrace();
}
}

/*
Function: speak
-----
This function checks to see if there is
a synthesizer intialized, and if there
is, calls the synthesizer's
speakPlainText method to speak the
text passed in as an argument

text: text to be spoken
*/
public void speak(String text){
    if(synthesizer == null){
        System.err.println("null synthesizer");
        System.exit(1);
    }
    synthesizer.speakPlainText(text,null);
}

/*
Function: createComponents
-----
This function creates, places and adds
the graphical components of the program
*/
public Component createComponents(){
    JPanel pane = new JPanel(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();
    c.fill = GridBagConstraints.HORIZONTAL;

    /* Create an object buttonA at (1, 1) */
    JButton buttonA = new JButton("A");
    /* Add a MouseListener so we can keep track of events on this object*/
    buttonA.addMouseListener(this);
    /* Set the name of the object*/
    buttonA.setName("button A");
    /* Specify how the object fits into the layout of the main panel - place it at
(1,1) */
    c.weightx = 0.5;

```

```

c.gridx = 1;
c.gridy = 1;
/* Make the button larger than its minimum size */
c.ipadx = 15;
c.ipady = 20;
c.insets = new Insets(30,30,30,30);
/* Add the object to the main panel */
pane.add(buttonA, c);

/* Create an object button B at (1, 2) */
JButton buttonB = new JButton("B");
buttonB.addMouseListener(this);
buttonB.setName("button B");
c.gridx = 1;
c.gridy = 2;
pane.add(buttonB, c);

/* Create icons */
ImageIcon fatmouse = new ImageIcon("small-mouse-fat.gif","fat mouse icon");
ImageIcon cheese = new ImageIcon("small-cheese.gif","cheese icon");
ImageIcon cat = new ImageIcon("small-cat.gif","cat icon");
ImageIcon dog = new ImageIcon("small-dog.gif","dog icon");

/* Create label with mouse icon at (2, 8) */
JLabel label1 = new JLabel("Fat Mouse",fatmouse,JLabel.CENTER);
label1.addMouseListener(this);
label1.setName("mouse icon");
label1.setVerticalTextPosition(JLabel.BOTTOM);
label1.setHorizontalTextPosition(JLabel.CENTER);
c.gridx = 2;
c.gridy = 8;
pane.add(label1, c);

/* Create label with cheese icon at (7, 1) */
JLabel label2 = new JLabel("Yummy Cheese",cheese,JLabel.CENTER);
label2.addMouseListener(this);
label2.setName("cheese icon");
label2.setVerticalTextPosition(JLabel.BOTTOM);
label2.setHorizontalTextPosition(JLabel.CENTER);
c.gridx = 7;
c.gridy = 1;
pane.add(label2, c);

/* Create label with cat icon at (3, 7) */
JLabel label3 = new JLabel("Hungry Cat",cat,JLabel.CENTER);
label3.addMouseListener(this);
label3.setName("cat icon");
label3.setVerticalTextPosition(JLabel.BOTTOM);
label3.setHorizontalTextPosition(JLabel.CENTER);
c.gridx = 3;
c.gridy = 7;
pane.add(label3, c);

/* Create label with dog icon at (8,4) */
JLabel label4 = new JLabel("Happy Dog",dog,JLabel.CENTER);
label4.addMouseListener(this);
label4.setName("dog icon");
label4.setVerticalTextPosition(JLabel.BOTTOM);
label4.setHorizontalTextPosition(JLabel.CENTER);
c.gridx = 8;
c.gridy = 4;
pane.add(label4, c);

/* Create blank label for spacing purposes */
JLabel label5 = new JLabel();
label5.setName("blanklabel");
c.gridx = 7;
c.gridy = 5;
c.ipadx = 30;
c.ipady = 30;
pane.add(label5, c);

```

```

/* Create window 1 panel at (3, 2) */
JPanel innerPane = new JPanel(new FlowLayout());
innerPane.setBorder(BorderFactory.createTitledBorder("Window1"));
innerPane.setName("window 1");
innerPane.addMouseListener(this);
c.ipady = 200;
c.ipadx = 200;
c.weightx = 0.0;
c.gridwidth = 4;
c.gridheight = 3;
c.gridx = 3;
c.gridy = 2;
pane.add(innerPane, c);

/* Create window 2 panel at (9, 6) */
JPanel innerPane2 = new JPanel(new FlowLayout());
innerPane2.setBorder(BorderFactory.createTitledBorder("Window2"));
innerPane2.addMouseListener(this);
innerPane2.setName("window 2");
c.ipady = 200;
c.ipadx = 200;
c.weightx = 0.0;
c.gridwidth = 4;
c.gridheight = 3;
c.gridx = 9;
c.gridy = 6;
pane.add(innerPane2, c);

/* Create a border for the main panel, add a mouse listener and name it */
pane.setBorder(BorderFactory.createEmptyBorder(1,1,1,1));
pane.addMouseListener(this);
pane.setName("main panel");

/* Create the outer panel (brake zone) */
JPanel outerPane = new JPanel(new GridBagLayout());
GridBagConstraints c2 = new GridBagConstraints();
c2.ipadx = 50;
c2.gridx = 1;
c2.gridy = 1;
c2.insets = new Insets(5,1,5,1);
outerPane.add(pane,c2);

outerPane.setName("outer panel");
outerPane.addMouseListener(this);

return outerPane;
}

/* Function: mousePressed
-----
If the right mouse button is pressed, send a
signal to the mouse device to release the
brake. If the middle button is pressed, speak
the name of the object where the event occurred.
If any mouse button is pressed, record the
event with an associated timestamp

e: event that triggered this function call
*/
public void mousePressed(MouseEvent e){
    TimeStamp = new java.util.Date().toString();
    char test = 1;
    if(SwingUtilities.isRightMouseButton(e) &&
        moutput1 == test && moutput2 == test){
        writeOutput( TimeStamp + " Brake release pressed", e);
        moutput1 = 0;
        moutput2 = 0;
        new Mouse().mouseWrite(moutput1, moutput2);
    }else if(SwingUtilities.isMiddleMouseButton(e)){
        writeOutput( TimeStamp + " Speech Request pressed", e);
    }
}

```

```

        speak(e.getComponent().getName());
    }else {
        writeOutput( TimeStamp + " Mouse pressed", e);
    }
}

/* Function: mouseReleased
/-----
This function is required in order to
implement MouseListener, but we don't actually
want to do anything when this event occurs
*/
public void mouseReleased(MouseEvent e){
}

/* Function: mouseEntered
/-----
Check if the component where the event occurred
is the not main panel, if it isn't, check if it
is the outer panel - if it is the outer panel
turn on the brake - if it is another panel
do the window-edge vibration, otherwise do
the icon/button vibration. If this is the
main panel, then if a flag is set to indicate
that we're coming from the main panel, turn
the brake off and reset this flag. In any
case, record the event with a timestamp
*/
public void mouseEntered(MouseEvent e){
    TimeStamp = new java.util.Date().toString();
    writeOutput( TimeStamp + " Mouse entered", e);
    if(e.getComponent().getName().compareTo(new String("main panel")) != 0){
        if(e.getComponent().getName().compareTo(new String("outer panel")) == 0){
            moutput1 = 1;
            moutput2 = 1;
            flag = 1;
        }else if(e.getComponent().getClass().getName().compareTo(new
String("javax.swing.JPanel")) == 0){
            moutput1 = 0;
            moutput2 = 1;
        }
        else{
            moutput1 = 1;
            moutput2 = 0;
        }
        new Mouse().mouseWrite(moutput1,moutput2);
    }
    else{
        if(flag == 1){
            moutput1 = 0;
            moutput2 = 0;
            new Mouse().mouseWrite(moutput1,moutput2);
            flag = 0;
        }
    }
}

/* Function: mouseExited
/-----
If the component where the event occurred is a
panel that is not the main or outer panel,
do the window-edge vibration*/
public void mouseExited(MouseEvent e){
    TimeStamp = new java.util.Date().toString();
    writeOutput( TimeStamp + " Mouse exited", e);
    String test1 = new String("javax.swing.JPanel");
    if(e.getComponent().getClass().getName().compareTo(test1)==0
    && e.getComponent().getName().compareTo(new String("outer panel")) != 0
    && e.getComponent().getName().compareTo(new String("main panel")) != 0){
        moutput1 = 0;
        moutput2 = 1;
    }
}

```

```

        new Mouse().mouseWrite(moutput1,moutput2);
    }
}

/* Function: mouseClicked
/-----
This function is required in order to
implement MouseListener, but we don't actually
want to do anything when this event occurs
*/
public void mouseClicked(MouseEvent e){
}

/* Function: writeOutput
/-----
This function writes information about an event
and to a file.*/
public void writeOutput(String eventDescription,
    MouseEvent e){
    System.out.println(eventDescription
        + " event detected on "
        + e.getComponent().getName());

    ps.println(eventDescription + " event detected on "
        + e.getComponent().getName());
}

/* Function: createAndShowGUI
/-----
Set the exit behavior of our GUI, create an
instance of this class, get the current time
and use this to create a file name that will be
different for each run. Add the graphical
components and create the synthesizer used to
generate speech
*/
private static void createAndShowGUI(){
    JFrame frame = new JFrame("MouseGUI");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    MouseGUI app = new MouseGUI();
    try{
        long TS = new java.util.Date().getTime();
        app.fout = new FileOutputStream("mouseTestData" + "." + TS + ".txt");
        app.ps = new PrintStream(fout);
        ps.println("Hello");
    }
    catch (IOException ex){
        System.err.println("Unable to open file");
        System.exit(-1);
    }

    Component contents = app.createComponents();
    frame.getContentPane().add(contents, BorderLayout.CENTER);

    frame.pack();
    frame.setVisible(true);
    app.createSynthesizer();
}

/* Run the GUI */
public static void main(String[] args){
    javax.swing.SwingUtilities.invokeLater(new Runnable(){
        public void run(){
            createAndShowGUI();
        }
    });
}
}

```

MouseGUI2.java

```
/*
 * MouseGUI2.java
 * Written by Heather Jones and Lauren Stadler
 * Engineering Senior Design Project, Spring 2006
 * -----
 * This file contains the program MouseGUI2, used to
 * test the haptically enhanced mouse project. The
 * program runs a GUI with icons, buttons and panels
 * (called windows). When mouse events are detected
 * over these objects, the GUI reacts by calling the
 * write function from Mouse.java, which allows it
 * to control the haptic mouse device, or by calling
 * a FreeTTS speech synthesizer function which allows
 * it to speak. The mouseCJavaInterface library
 * must be in the path specified by java.library.path.
 *
 */

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.ImageIcon;
import java.io.*;
import java.util.*;
import com.sun.speech.freetts.jsapi.FreeTTSEngineCentral;
import java.util.Locale;
import javax.speech.EngineList;
import javax.speech.EngineCreate;
import javax.speech.synthesis.Synthesizer;
import javax.speech.synthesis.SynthesizerModeDesc;

/* This GUI implements the interface MouseListener, which allows
it to easily obtain mouse events*/
public class MouseGUI2 implements MouseListener{
    JTextArea tarea;
    /* fields that allow us to write to a file */ static FileOutputStream fout; static
    PrintStream ps;
    /* characters to be sent to the mouse device */
    char moutput1 = 0;
    char moutput2 = 0;

    int flag = 0;
    /* used to include the timestamp for data collection */
    static String TimeStamp;

    /* used for speech synthesis */
    protected Synthesizer synthesizer;

    /*This method adapted from JSAPIClock.java*/
    /* Function: createSynthesizer
    /-----
    This functions handles the setup of the
    speech synthesizer that allows the
    program to talk.
    */
    public void createSynthesizer(){
        try {
            SynthesizerModeDesc desc =
                new SynthesizerModeDesc(null,
                    "general",
                    null,
                    Boolean.FALSE,
                    null);

            FreeTTSEngineCentral central = new FreeTTSEngineCentral();
            EngineList list = central.createEngineList(desc);

            if(list == null){
```

```

        System.err.println("null engine list");
        System.exit(1);
    }
    else{
        System.out.println("engine list size = " + list.size());
    }

    if (list.size() > 0) {
        EngineCreate creator = (EngineCreate) list.get(0);
        synthesizer = (Synthesizer) creator.createEngine();
    }
    if (synthesizer == null) {
        System.err.println("Cannot create synthesizer");
        System.exit(1);
    }
    synthesizer.allocate();
    synthesizer.resume();

} catch (Exception e) {
    e.printStackTrace();
}
}

/*
Function: speak
-----
This function checks to see if there is
a synthesizer intialized, and if there
is, calls the synthesizer's
speakPlainText method to speak the
text passed in as an argument

text: text to be spoken
*/
public void speak(String text){
    if(synthesizer == null){
        System.err.println("null synthesizer");
        System.exit(1);
    }
    synthesizer.speakPlainText(text,null);
}

/*
Function: createComponents
-----
This function creates, places and adds
the graphical components of the program
*/
public Component createComponents(){
    JPanel pane = new JPanel(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();
    c.fill = GridBagConstraints.HORIZONTAL;

    /* Create an object buttonX at (2, 9) */
    JButton buttonX = new JButton("X");
    /* Add a MouseListener so we can keep track of events on this object*/
    buttonX.addMouseListener(this);
    /* Set the name of the object*/
    buttonX.setName("button X");
    c.weightx = 0.5;
    /* Specify how the object fits into the layout of the main panel - place it at
(2,9) */
    c.gridx = 2;
    c.gridy = 9;
    c.ipadx = 15;
    c.ipady = 20;
    c.insets = new Insets(30,30,30,30);
    /* Add the object to the main panel */
    pane.add(buttonX, c);

```

```

/* Create an object button Y at 8, 6 */
JButton buttonY = new JButton("Y");
buttonY.addMouseListener(this);
buttonY.setName("button Y");
c.gridx = 8;
c.gridy = 6;
pane.add(buttonY, c);

/* Create icons */
ImageIcon trash = new ImageIcon("trash.gif","trash");
ImageIcon internet = new ImageIcon("globe.gif","internet");
ImageIcon email = new ImageIcon("envelope.gif","email");
ImageIcon wordprocessor = new ImageIcon("notepad.gif","wordprocessor");

/* Create label with garbage icon at (1, 9) */
JLabel label1 = new JLabel("Garbage",trash,JLabel.CENTER);
label1.addMouseListener(this);
label1.setName("garbage");
label1.setVerticalTextPosition(JLabel.BOTTOM);
label1.setHorizontalTextPosition(JLabel.CENTER);
c.gridx = 1;
c.gridy = 9;
pane.add(label1, c);

/* Create label with internet icon at (5, 5) */
JLabel label2 = new JLabel("Internet",internet,JLabel.CENTER);
label2.addMouseListener(this);
label2.setName("internet");
label2.setVerticalTextPosition(JLabel.BOTTOM);
label2.setHorizontalTextPosition(JLabel.CENTER);
c.gridx = 5;
c.gridy = 5;
pane.add(label2, c);

/* Create label with email icon at (12, 4) */
JLabel label3 = new JLabel("E-mail",email,JLabel.CENTER);
label3.addMouseListener(this);
label3.setName("E mail");
label3.setVerticalTextPosition(JLabel.BOTTOM);
label3.setHorizontalTextPosition(JLabel.CENTER);
c.gridx = 12;
c.gridy = 4;
pane.add(label3, c);

/* Create label with word processor icon at (12, 2) */
JLabel label4 = new JLabel("Word Processor",wordprocessor,JLabel.CENTER);
label4.addMouseListener(this);
label4.setName("word processor");
label4.setVerticalTextPosition(JLabel.BOTTOM);
label4.setHorizontalTextPosition(JLabel.CENTER);
c.gridx = 12;
c.gridy = 2;
pane.add(label4, c);

/* Create blank label for spacing purposes */
JLabel label5 = new JLabel();
label5.setName("blanklabel");
c.gridx = 7;
c.gridy = 5;
c.ipadx = 30;
c.ipady = 30;
pane.add(label5, c);

/* Create window 1 panel at (1, 1) */
JPanel innerPane = new JPanel(new FlowLayout());
innerPane.setBorder(BorderFactory.createTitledBorder("Window1"));
innerPane.setName("window 1");
innerPane.addMouseListener(this);
c.ipady = 200;
c.ipadx = 200;
c.weightx = 0.0;

```

```

        c.gridwidth = 5;
        c.gridheight = 4;
        c.gridx = 1;
        c.gridy = 1;
        pane.add(innerPane, c);

/* Create window 2 panel at (6, 1) */
JPanel innerPane2 = new JPanel(new FlowLayout());
    innerPane2.setBorder(BorderFactory.createTitledBorder("Window2"));
    innerPane2.addMouseListener(this);
    innerPane2.setName("window 2");
    c.ipady = 200;
    c.ipadx = 200;
    c.weightx = 0.0;
    c.gridwidth = 5;
    c.gridheight = 4;
    c.gridx = 6;
    c.gridy = 1;
    pane.add(innerPane2, c);

/* Create window 3 panel at (7, 3) */
JPanel innerPane3 = new JPanel(new FlowLayout());
    innerPane3.setBorder(BorderFactory.createTitledBorder("Window3"));
    innerPane3.addMouseListener(this);
    innerPane3.setName("window 3");
    c.ipady = 200;
    c.ipadx = 200;
    c.weightx = 0.0;
    c.gridwidth = 7;
    c.gridheight = 3;
    c.gridx = 5;
    c.gridy = 7;
    pane.add(innerPane3, c);

pane.setBorder(BorderFactory.createEmptyBorder(1,1,1,1));
pane.addMouseListener(this);
pane.setName("main panel");

/* Create the outer panel (brake zone) */
JPanel outerPane = new JPanel(new GridBagLayout());
GridBagConstraints c2 = new GridBagConstraints();
c2.ipadx = 50;
c2.gridx = 1;
c2.gridy = 1;
c2.insets = new Insets(5,1,5,1);
outerPane.add(pane,c2);

outerPane.setName("outer panel");
outerPane.addMouseListener(this);

return outerPane;
}

/* Function: mousePressed
/-----
If the right mouse button is pressed, send a
signal to the mouse device to release the
brake. If the middle button is pressed, speak
the name of the object where the event occurred.
If any mouse button is pressed, record the
event with an associated timestamp

e: event that triggered this function call
*/
public void mousePressed(MouseEvent e){
    TimeStamp = new java.util.Date().toString();
    char test = 1;
    if(SwingUtilities.isRightMouseButton(e) &&
        moutput1 == test && moutput2 == test){
        writeOutput( TimeStamp + " Brake release pressed", e);
        moutput1 = 0;

```

```

        moutput2 = 0;
        new Mouse().mouseWrite(moutput1, moutput2);
    }else if(SwingUtilities.isMiddleMouseButton(e)){
        writeOutput( TimeStamp + " Speech Request pressed", e);
        speak(e.getComponent().getName());
    }else {
        writeOutput( TimeStamp + " Mouse pressed", e);
    }
}

/* Function: mouseReleased
-----
This function is required in order to
implement MouseListener, but we don't actually
want to do anything when this event occurs
*/
public void mouseReleased(MouseEvent e){

}

/* Function: mouseEntered
-----
Check if the component where the event occurred
is the not main panel, if it isn't, check if it
is the outer panel - if it is the outer panel
turn on the brake - if it is another panel
do the window-edge vibration, otherwise do
the icon/button vibration. If this is the
main panel, then if a flag is set to indicate
that we're coming from the main panel, turn
the brake off and reset this flag. In any
case, record the event with a timestamp
*/
public void mouseEntered(MouseEvent e){
    TimeStamp = new java.util.Date().toString();
    writeOutput(TimeStamp + " Mouse entered", e);
    if(e.getComponent().getName().compareTo(new String("main panel")) != 0){
        if(e.getComponent().getName().compareTo(new String("outer panel")) == 0){
            moutput1 = 1;
            moutput2 = 1;
            flag = 1;
        }else if(e.getComponent().getClass().getName().compareTo(new
String("javax.swing.JPanel")) == 0){
            moutput1 = 0;
            moutput2 = 1;
        }
        else{
            moutput1 = 1;
            moutput2 = 0;
        }
        new Mouse().mouseWrite(moutput1,moutput2);
    }
    else{
        if(flag == 1){
            moutput1 = 0;
            moutput2 = 0;
            new Mouse().mouseWrite(moutput1,moutput2);
            flag = 0;
        }
    }
}

/* Function: mouseExited
-----
If the component where the event occurred is a
panel that is not the main or outer panel,
do the window-edge vibration*/
public void mouseExited(MouseEvent e){
    TimeStamp = new java.util.Date().toString();
    writeOutput(TimeStamp + " Mouse exited", e);
    String test1 = new String("javax.swing.JPanel");
    if(e.getComponent().getClass().getName().compareTo(test1)==0

```

```

        && e.getComponent().getName().compareTo(new String("outer panel")) != 0
        && e.getComponent().getName().compareTo(new String("main panel")) != 0){
            moutput1 = 0;
            moutput2 = 1;
            new Mouse().mouseWrite(moutput1,moutput2);
        }
    }

    /* Function: mouseClicked
    /-----
    This function is required in order to
    implement MouseListener, but we don't actually
    want to do anything when this event occurs
    */
    public void mouseClicked(MouseEvent e){
    }

    /* Function: writeOutput
    /-----
    This function writes information about an event
    and to a file.*/
    public void writeOutput(String eventDescription,
                            MouseEvent e){
        System.out.println(eventDescription
                            + " event detected on "
                            + e.getComponent().getName());

        ps.println(eventDescription + " event detected on "
                   + e.getComponent().getName());
    }

    /* Function: createAndShowGUI
    /-----
    Set the exit behavior of our GUI, create an
    instance of this class, get the current time
    and use this to create a file name that will be
    different for each run. Add the graphical
    components and create the synthesizer used to
    generate speech
    */
    private static void createAndShowGUI(){
        JFrame frame = new JFrame("MouseGUI");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        MouseGUI2 app = new MouseGUI2();
        try{
            long TS = new java.util.Date().getTime();
            app.fout = new FileOutputStream("mouseTestData2" + "." + TS + ".txt");
            app.ps = new PrintStream(fout);
            ps.println("Hello");
        }
        catch (IOException ex){
            System.err.println("Unable to open file");
            System.exit(-1);
        }

        Component contents = app.createComponents();
        frame.getContentPane().add(contents, BorderLayout.CENTER);

        frame.pack();
        frame.setVisible(true);
        app.createSynthesizer();
    }

    public static void exitFunction(){
        try{
            fout.close();
        }
    }

```

```
        catch (IOException ex){
            System.err.println("Unable to close file");
            System.exit(-1);
        }
    }
    /* Run the GUI */
    public static void main(String[] args){
        javax.swing.SwingUtilities.invokeLater(new Runnable(){
            public void run(){
                createAndShowGUI();
            }
        });
    }
}
```

Appendix D: Other Code

- Hotplug Script
- C-Java Interface
 - Header file
 - C++ file
 - Java file
- X Window Interaction – Xquerypointer.c

Hotplug Script

```
#!/bin/sh -e
#
#smartmouse.hotplug
#Written by Heather Jones and Lauren Stadler
#Engineering Senior Design Project, 2006
#
#adapted from phidgets.hotplug, written by Martin F. Krafft
#
# To activate:
#  install -m755 -oroot -groot smartmouse.hotplug /etc/hotplug.d/usb
#
# to allow users to use the smartmouse device, add the group
#  smouse to the file /etc/group and add the desired users
#  to the group

#only need to do this for new devices
[[${ACTION} != add]] && exit 0

# wait a while for the device node to become available
COUNT=3
while [[ ! -f $DEVICE ]] && ((COUNT-- > 0)); do sleep 1; done

/usr/bin/logger -t $0 "new Smartmouse $PRODUCT appeared at $DEVICE"
# configure device
chgrp smouse $DEVICE
chmod 0660 $DEVICE

exit 0
```

C-Java Interface:

Header file

```
/* Mouse.h */
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class Mouse */

#ifdef _Included_Mouse
#define _Included_Mouse
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:      Mouse
 * Method:     mouseWrite
 * Signature:  (CC)I
 */
JNIEXPORT jint JNICALL Java_Mouse_mouseWrite
    (JNIEnv *, jobject, jchar, jchar);
/*
 * Class:      Mouse
```

```

* Method:    mouseInitSpeech
* Signature: ()I
*/
JNIEXPORT jint JNICALL Java_Mouse_mouseInitSpeech
    (JNIEnv *, jclass);

/*
* Class:     Mouse
* Method:    mouseSpeak
* Signature: (Ljava/lang/String;)I
*/
JNIEXPORT jint JNICALL Java_Mouse_mouseSpeak
    (JNIEnv *, jobject, jstring);
#ifdef __cplusplus
}
#endif
#endif

```

C++ file

```

/*
* MouseImp.c
* Written by Heather Jones and Lauren Stadler
* Engineering Senior Design Project, Spring 2006
* -----
* This file provides the function definitions needed
* in order to call the functions in the mousselib
* library from Java using the Java Native Interface
* (JNI). The latter two functions, mouseInitSpeech
* and mouseSpeak, do not actually work with this
* interface because mouseInitSpeech must be the first
* function called in a given program. The C-Java
* interfacing in this project was developed following
* the procedure outlined at:
* http://www.nag.co.uk/doc/TechRep/html/Tr3\_00/TRExample1.html
*/

#include <jni.h>
#include "mousselib.h"
#include "Mouse.h"
#include <string>

using namespace std;
JNIEXPORT jint JNICALL Java_Mouse_mouseWrite(JNIEnv *env, jobject object,
                                             jchar arg1, jchar arg2){

    int ret = 7;
    ret = mouseWrite(arg1, arg2);
    return ret;
}

JNIEXPORT jint JNICALL Java_Mouse_mouseInitSpeech(JNIEnv *env, jclass c){
    int ret;
    ret = mouseInitSpeech();
    return ret;
}

JNIEXPORT jint JNICALL Java_Mouse_mouseSpeak(JNIEnv *env, jobject object,
                                             jstring str){

    int ret = 7;
    jboolean iscopy;
    const char *tmpstr = env->GetStringUTFChars(str, &iscopy);

    ret = mouseSpeak(string(tmpstr));
    env->ReleaseStringUTFChars(str, tmpstr);
    return ret;
}

```

Java file

```
/*
 * Mouse.java
 * Written by Heather Jones and Lauren Stadler
 * Engineering Senior Design Project, Spring 2006
 * -----
 * This is the Java portion of the C-Java interface.
 * If the program is run, it calls the mouseWrite
 * function on the two command line arguments it is
 * given (these should be 0 or 1). This class can
 * also be used from other programs to call the
 * mouseWrite function.
 */

import java.lang.Character;
public class Mouse
{
    /*
    Function: mouseWrite
    //-----
    This function takes two character arguments and
    sends these over USB, using an interrupt transfer,
    to the device with the vendor and product ID
    specified in mouselib.h (the PIC18F4550 development
    board). If any of the libhid functions called from
    this function fail, the name of the failing function
    and the error number will be printed to stderr.

    arg1: the first byte to send
    arg2: the second byte to send

    returns:0 on success, 1 on error

    */
    public native int mouseWrite(char arg1, char arg2);
    static
    {
        /* Load the C-Java interface library - this library must be in the
        the java.library.path in order to be loaded */
        System.loadLibrary("mouseCJavaInterface");
    }

    public static void main(String[] args)
    {
        char arg1, arg2;
        int ret;

        /* Check that we have the right number of arguments */
        if (args.length != 2)
        {
            System.out.println("Usage: java Mouse arg1 arg2");
            System.out.println("Sends arguments arg1 and arg2 to mouse device");
            System.exit(1);
        }

        Mouse mouse = new Mouse();

        /*
        If the first or second command line argument is zero, set
        the appropriate character to be sent to the mouse to zero,
        otherwise set this character to 1
        */
        arg1 = args[0].charAt(0);
        if(arg1 == '0') arg1 = 0;
        else arg1 = 1;

        arg2 = args[1].charAt(0);
        if(arg2 == '0') arg2 = 0;
```

```

else arg2 = 1;

/* Print out the values we're going to send*/
System.out.println(arg1 + ", " + arg2);

/*
   Send two bytes to the mouse - if this write fails, print
   an error
*/
ret = mouse.mouseWrite(arg1,arg2);
if(ret != 0)
{
    System.out.println("Error writing to device");
}
}
}

```

X Window Interaction – Xquerypointer.c

```

/*
 * Xquerypointer.c
 * Written by Heather Jones and Lauren Stadler
 * Engineering Senior Design Project, Spring 2006
 * -----
 */
/*
 * This program constantly outputs the root coordinates of the pointer and exits when the
 * pointer reaches the edge of the screen.
 *
 * Modified from http://www.gusnet.cx/proj/miscunix/code/xquerypointer.c
 *
 * Compile with:
 * gcc -Wall -I/usr/X11R6/include -L/usr/X11R6/lib -lXmu -o Xquerypointer
Xquerypointer.c
 */
#include <stdio.h>
#include <stdlib.h>
#include <X11/Xlib.h>

int main(int argc, char **argv)
{
    Display *dpy;
    Window root;
    Window ret_root;
    Window ret_child;
    Window root_return, parent_return;
    Window **children_return;
    unsigned int numchildren;
    int root_x;
    int root_y;
    int win_x;
    int win_y;
    int test;
    int brake;
    unsigned int mask;

    dpy = XOpenDisplay(NULL);
    root = XDefaultRootWindow(dpy);
    test = 1;
    brake = 0;

    while (test){
        /* Uses the XQueryPointer function to find the coordinates of the pointer */
        if(XQueryPointer(dpy, root, &ret_root, &ret_child, &root_x, &root_y,
            &win_x, &win_y, &mask))
        {
            /* Prints the root coordinates of the pointer */
            printf("+%d+%d\n", root_x, root_y);
            /* Sets test = 0, which exits the while loop, if the pointer touches the
            edge of the screen */

```

```
        if(root_x == 0 || root_y == 0 || root_x == 1279 || root_y == 1023){
            test = 0;
        }
    }
else
    {
        printf("hmmmm, where is that sneaky pointer?\n");
        test = 0;
    }
}
return 0;
}
```

Appendix E: Test 1 Data

Task 1

User	Order	Speech						Haptic						% Improve
		Time to W1	S R to W1	Time to W2	S R to W2	Total Time	Total S R	Time to W1	S R to W1	Time to W2	S R to W2	Total Time	Total S R	
Non-Trackball 1	Hap only							31	6	5	1	36	7	
Non-Trackball 2	Hap only							19	4	41	6	60	10	
Non-Trackball 3	Speech	28	4	9	2	37	6	9	1	9	2	18	3	0.5135135
Non-Trackball 4	Speech	18	3	2	1	20	4	9	1	4	1	13	2	0.35
Non-Trackball 5	Hap	31	6	76	11	107	17	27	10	9	3	36	13	0.6635514
Blind	Speech	14	1	137	18	151	19	5	1	6	1	11	2	0.9271523
Non-Trackball 6	Hap	54	18	10	4	64	22	8	1	40	3	48	4	0.25
Non-Trackball 7	Hap	16	1	7	2	23	3	11	2	14	2	25	4	-0.086957
Non-Trackball 8	Speech	6	2	19	7	25	9	37	9	5	1	42	10	-0.68
Non-Trackball 9	Hap	9	2	44	12	53	14	5	2	20	2	25	4	0.5283019
Non-Trackball 10	Speech	45	7	11	1	56	8	26	1	50	6	76	7	-0.357143
Non-Trackball 11	Hap	12	2	24	5	36	7	7	1	10	1	17	2	0.5277778
Non-Trackball 12	Speech	35	5	34	13	69	18	26	4	36	10	62	14	0.1014493
Non-Trackball 13	Hap	4	1	6	2	10	3	33	5	6	1	39	6	-2.9
Non-Trackball 14	Speech							13	0	18	6	31	6	
Non-Trackball 15	Speech	8	2	12	3	20	5	23	0	14	1	37	1	-0.85
Non-Trackball 16	Speech	3	1	3	1	6	2	67	1	18	5	85	6	-13.16667
Trackball 1	Speech	5	1	20	3	25	4	7	2	9	2	16	4	0.36
Trackball 2	Hap	108	30	4	1	112	31	17	1	46	4	63	5	0.4375
Trackball 3	Speech	15	3	6	1	21	4	4	1	4	1	8	2	0.6190476
Trackball 4	Hap	5	0	24	6	29	6							
Trackball 5	Hap	2	1	28	6	30	7	4	1	7	2	11	3	0.6333333
Trackball 6	Speech	28	3	5	1	33	4	4	0	13	2	17	2	0.4848485
Average						46.35	9.65					35.2727	5.31818	0.2389919
Standard Deviation						37.8922	7.87585					22.2928	3.63038	

Task 2 – Speech only

User	Order	Speech							
		Time to W1	S R to W1	Time to Int	S R to Int	Time to W3	SR to W3	Total Time	Total S R
Non-Trackball 1	Hap only								
Non-Trackball 2	Hap only								
Non-Trackball 3	Speech	39	11	327	39	20	6	386	56
Non-Trackball 4	Speech	4	1	34	15	4	1	42	17
Non-Trackball 5	Hap	11	2	43	8	9	1	63	11
Blind	Speech	53	12	49	11	12	3	114	26
Non-Trackball 6	Hap	51	18	51	15	3	1	105	34
Non-Trackball 7	Hap	9	1	62	24	9	2	80	27
Non-Trackball 8	Speech	5	2	68	28	22	10	95	40
Non-Trackball 9	Hap	7	3	12	5	12	5	31	13
Non-Trackball 10	Speech	38	8	126	37	75	17	239	62
Non-Trackball 11	Hap	11	2	363	74	22	3	396	79
Non-Trackball 12	Speech	65	11	39	10	51	23	155	44
Non-Trackball 13	Hap	5	2	58	27	4	1	67	30
Non-Trackball 14	Speech	16	4	108	59	5	2	129	65
Non-Trackball 15	Speech	12	2	21	6	5	1	38	9
Non-Trackball 16	Speech	4	1	222	76	7	2	233	79
Trackball 1	Speech	3	1	194	80	7	2	204	83
Trackball 2	Hap	75	30	4	1	3	0	82	31
Trackball 3	Speech	8	2	137	47	6	0	151	49
Trackball 4	Hap	5	2	70	18	10	1	85	21
Trackball 5	Hap	4	1	10	5	3	1	17	7
Trackball 6	Speech	4	1	12	3	6	1	22	5
Average								130.1904762	37.52380952
Standard Deviation								107.9720422	25.04120414

Task 2 – Speech and haptic

User	Order	Haptic								% Improve
		Time to W1	S R to W1	Time to Int	S R to Int	Time to W3	SR to W3	Total Time	Total S R	
Non-Trackball 1	Hap only	3	1	48	11	4	1	55	13	
Non-Trackball 2	Hap only	4	1	102	13	5	1	111	15	
Non-Trackball 3	Speech	53	5	15	1	23	2	91	8	0.764248705
Non-Trackball 4	Speech	4	1	45	13	4	1	53	15	-0.26190476
Non-Trackball 5	Hap	9	3	41	22	21	7	71	32	-0.12698413
Blind	Speech	11	2	7	1	18	2	36	5	0.684210526
Non-Trackball 6	Hap	8	1	156	18	7	1	171	20	-0.62857143
Non-Trackball 7	Hap	13	3	74	13	16	3	103	19	-0.2875
Non-Trackball 8	Speech	20	5	99	33	4	1	123	39	-0.29473684
Non-Trackball 9	Hap	5	1	17	1	6	1	28	3	0.096774194
Non-Trackball 10	Speech	6	1	23	4	7	1	36	6	0.849372385
Non-Trackball 11	Hap	10	2	8	1	28	3	46	6	0.883838384
Non-Trackball 12	Speech	24	7	7	2	8	2	39	11	0.748387097
Non-Trackball 13	Hap	6	1	13	2	7	1	26	4	0.611940299
Non-Trackball 14	Speech	4	1	10	3	3	1	17	5	0.868217054
Non-Trackball 15	Speech	11	1	78	12	11	2	100	15	-1.63157895
Non-Trackball 16	Speech	24	1	10	1	5	1	39	3	0.832618026
Trackball 1	Speech	6	2	7	1	4	0	17	3	0.916666667
Trackball 2	Hap	10	2	65	9	7	1	82	12	0
Trackball 3	Speech	3	1	5	1	4	1	12	3	0.920529801
Trackball 4	Hap	7	1	55	4	7	1	69	6	0.188235294
Trackball 5	Hap	15	4	6	1	6	1	27	6	-0.58823529
Trackball 6	Speech	4	1	61	14	4	1	69	16	-2.13636364
Average								61.7826087	11.52173913	0.525444483
Standard Deviation								40.20962856	9.360796223	

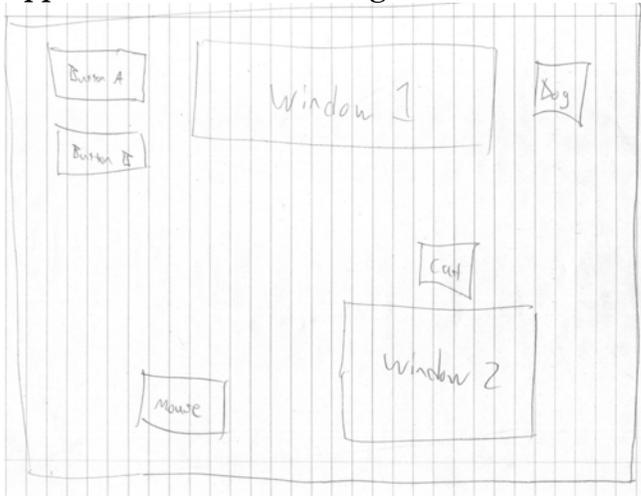
Task 3 – Speech only

User	Order	Speech											
		Time to Y	SR to Y	Time to Gar	S R to Gar	Time to Y 2	S R to Y 2	Time to WP	S R to WP	Time to Y 3	SR to Y 3	Total Time	Total S R
Non-Trackball 1	Hap only												
Non-Trackball 2	Hap only												
Non-Trackball 3	Speech	128	31	32	8	244	71	138	29	150	42	692	181
Non-Trackball 4	Speech	45	21	36	23	70	50	22	8	24	12	197	114
Non-Trackball 5	Hap	48	7	18	3	155	16	54	9	46	11	321	46
Blind	Speech	79	31	71	26	161	46	39	12	197	48	547	163
Non-Trackball 6	Hap	59	22	147	40	481	114	167	65	55	28	909	269
Non-Trackball 7	Hap	193	62	38	9	48	15	111	19	36	11	426	116
Non-Trackball 8	Speech	453	232	52	25	30	7	50	19	877	480	1462	763
Non-Trackball 9	Hap	190	67	13	1	101	36	22	6	124	29	450	139
Non-Trackball 10	Speech	70	15	24	6	31	10	27	7	73	17	225	55
Non-Trackball 11	Hap	129	30	36	7	24	5	44	8	54	11	287	61
Non-Trackball 12	Speech	12	2	18	8	25	11	6	1	107	47	168	69
Non-Trackball 13	Hap	90	53	12	4	127	63	47	7	10	3	286	130
Non-Trackball 14	Speech	15	8	115	30	242	99	63	24	36	12	471	173
Non-Trackball 15	Speech	142	40	7	3	294	75	30	9	224	71	697	198
Non-Trackball 16	Speech	327	107	129	40	25	8	17	6	92	40	590	201
Trackball 1	Speech	60	31	151	59	24	10	9	2	34	15	278	117
Trackball 2	Hap	140	50	232	86	62	22	49	16	148	49	631	223
Trackball 3	Speech	74	37	123	53	35	27	35	20	22	13	289	150
Trackball 4	Hap	117	44	19	6	80	41	7	2	27	13	250	106
Trackball 5	Hap	48	25	146	76	62	30	23	7	52	29	331	167
Trackball 6	Speech	26	10	19	9	66	28	40	11	98	48	249	106
Average												464.5714	168.9048
Standard Deviation												302.2464	147.7944

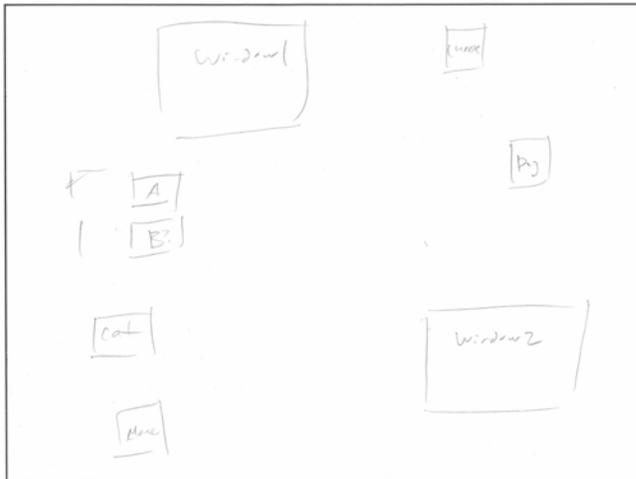
Task 3 – Speech and haptic

User	Order	Haptic													% Improve
		Time to Y	SR to Y	Time to Gar	S R to Gar	Time to Y 2	S R to Y 2	Time to WP	S R to WP	Time to Y 3	SR to Y 3	Total Time	Total S R		
Non-Trackball 1	Hap only	35	7	11	2	12	1	35	7	13	3	106	20		
Non-Trackball 2	Hap only	56	12	41	4	36	6	12	3	36	7	181	32		
Non-Trackball 3	Speech	121	23	67	10	66	9	19	3	23	13	296	58	0.572254335	
Non-Trackball 4	Speech	36	15	19	4	19	7	12	4	8	2	94	32	0.52284264	
Non-Trackball 5	Hap	155	74	24	7	121	59	80	26	19	8	399	174	-0.242990654	
Blind	Speech	66	7	30	1	34	5	16	1	54	4	200	18	0.634369287	
Non-Trackball 6	Hap	101	6	341	26	32	6	19	4	23	3	516	45	0.432343234	
Non-Trackball 7	Hap	423	73	186	30	67	11	86	19	20	5	782	138	-0.835680751	
Non-Trackball 8	Speech	106	46	148	55	162	75	7	1	98	28	521	205	0.643638851	
Non-Trackball 9	Hap	36	7	15	1	15	1	10	1	27	2	103	12	0.771111111	
Non-Trackball 10	Speech	52	11	15	4	14	2	21	4	39	11	141	32	0.373333333	
Non-Trackball 11	Hap	121	11	20	2	21	1	14	1	29	2	205	17	0.285714286	
Non-Trackball 12	Speech	10	3	59	18	56	13	54	10	31	7	210	51	-0.25	
Non-Trackball 13	Hap	14	1	9	2	19	3	7	1	14	1	63	8	0.77972028	
Non-Trackball 14	Speech	17	6	14	5	13	2	25	8	18	4	87	25	0.815286624	
Non-Trackball 15	Speech	81	8	191	13	17	1	12	0	14	3	315	25	0.548063128	
Non-Trackball 16	Speech	16	4	6	1	19	3	22	5	25	3	88	16	0.850847458	
Trackball 1	Speech	33	5	46	7	41	9	11	2	21	5	152	28	0.45323741	
Trackball 2	Hap	116	14	58	11	86	13	44	6	15	2	319	46	0.494453249	
Trackball 3	Speech	63	8	63	10	34	7	6	1	100	14	266	40	0.079584775	
Trackball 4	Hap	42	5	15	2	48	18	16	2	12	1	133	28	0.468	
Trackball 5	Hap	60	6	33	6	13	2	13	1	21	5	140	20	0.577039275	
Trackball 6	Speech	11	1	9	1	85	6	17	1	33	3	155	12	0.37751004	
Average												237.91304	47.04348	0.487887053	
Standard Deviation												175.68927	52.24982		

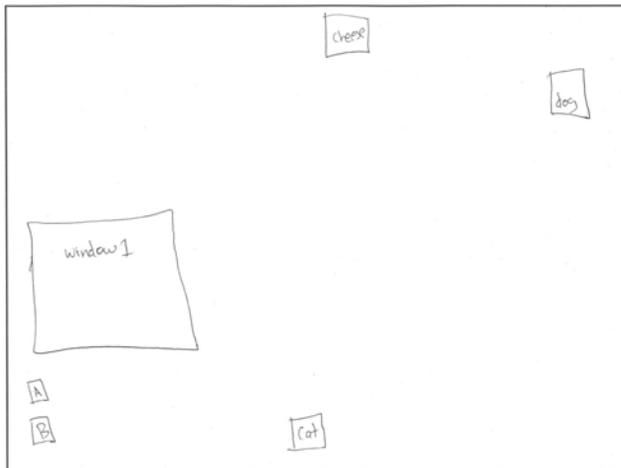
Appendix F: Test 2 Drawings



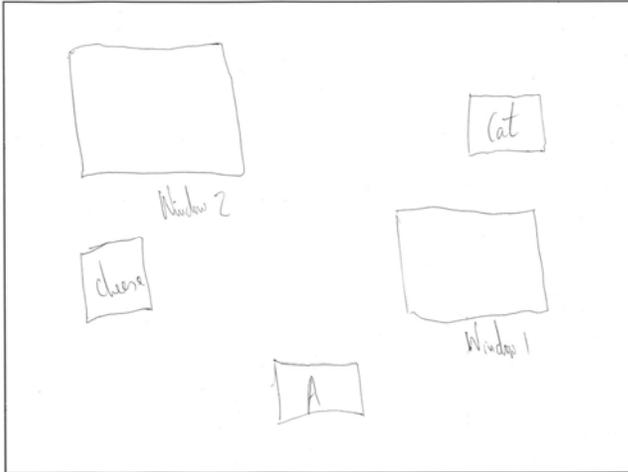
Non-Trackball User 1: 15 minutes



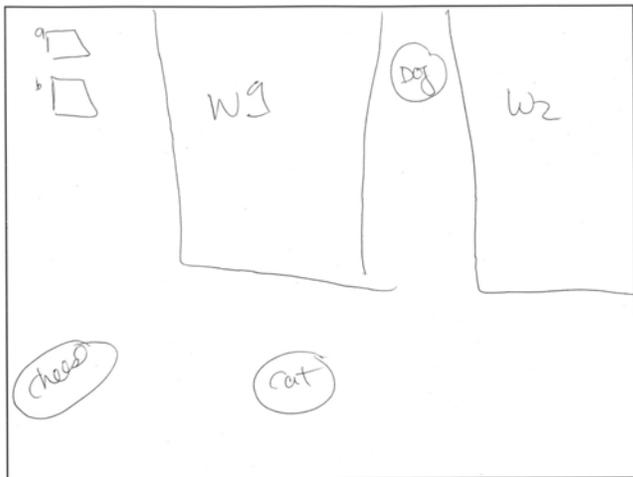
Non-Trackball User 2: 15 minutes



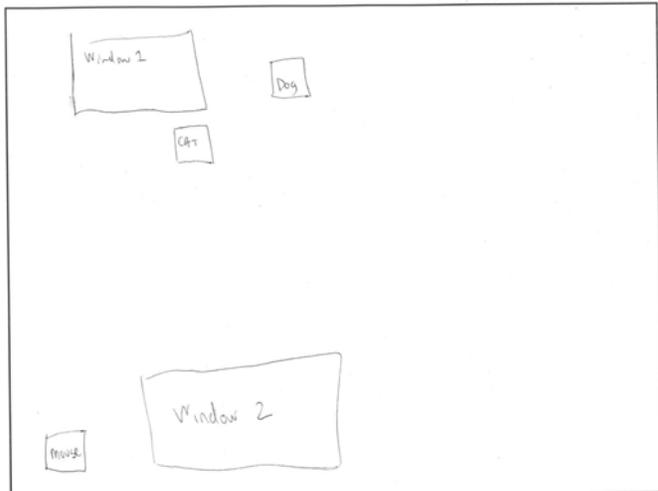
Non-Trackball User 3: 5 minutes



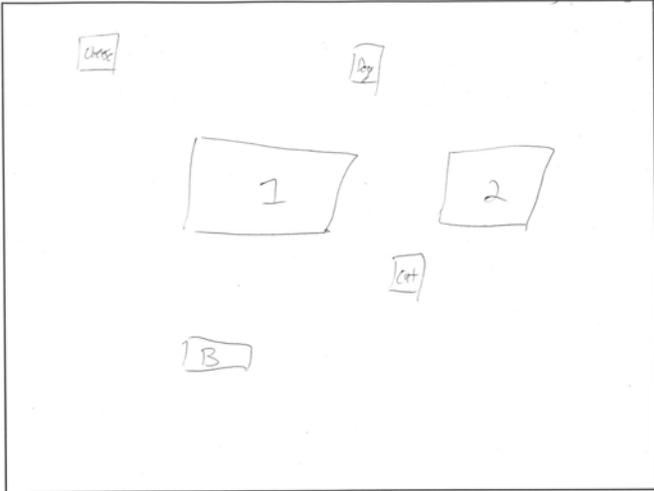
Non-Trackball User 7: 5 minutes



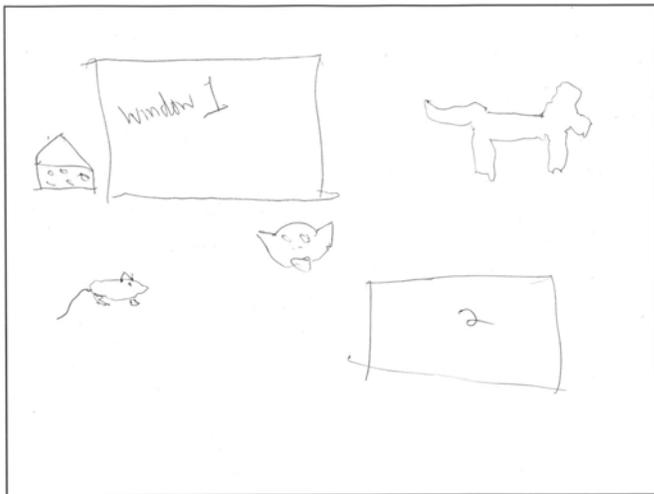
Non-Trackball User 8: 5 minutes



Non-Trackball User 10: 5 minutes



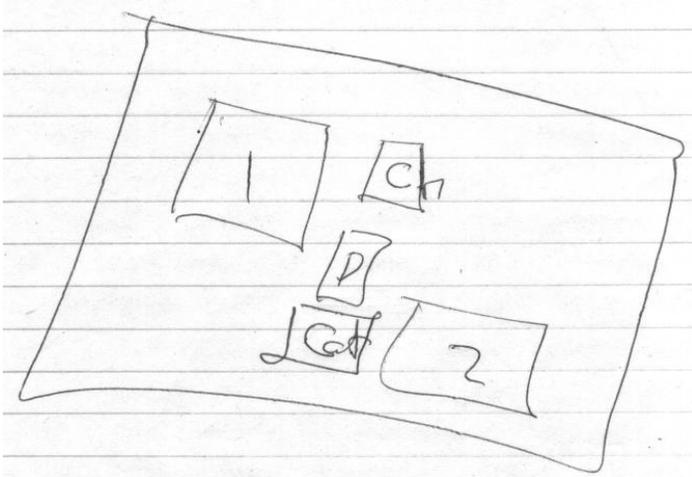
Non-Trackball User 14: 5 minutes



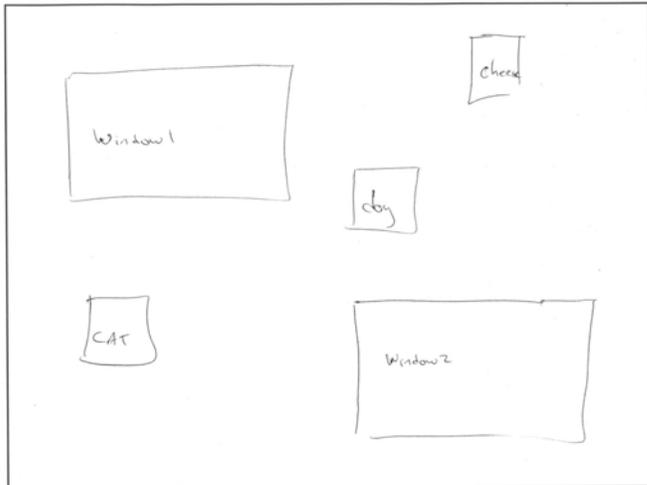
Trackball User 2: 5 minutes



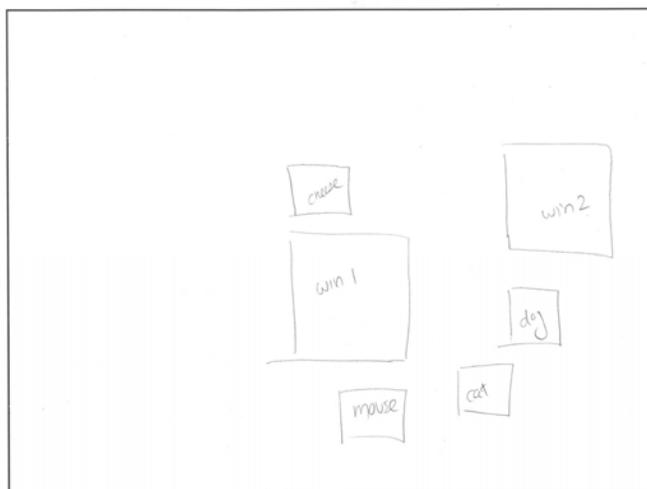
Trackball User 3: 5 minutes



Trackball User 4: 5 minutes



Trackball User 5: 5 minutes



Blind User: 5 minutes