FINGER MOUSE AND TEXT-TO-SPEECH APPLICATION AS ADDITIONS TO THE SMART WHEELCHAIR

NSF Summer Undergraduate Fellowship in Sensor Technologies
(Supported in part by Microsoft Corporation)
Karla Conn (Electrical Engineering) - University of Kentucky
Advisor: Dr. Jim Ostrowski

ABSTRACT

The smart wheelchair project is a unique investigation into the possibilities of helping the impaired navigate in a mobile chair. Many disabled people who need the help of a wheelchair to move about also need help communicating orally. My project allows the "walking" wheelchair to do some "talking." I developed the addition of a communication program for the wheelchair as well as a finger mouse to be implemented into all the programs. The finger mouse is a switch button small enough to wear on one's finger. When the button is pressed, a signal is sent out from the transmitter and picked up by the receiver, which sends a signal through the parallel port of the computer to execute the desired application. The application I created is a speech program that speaks text messages. The finger mouse and speech application are connected through a communication display interface. The mouse, display, and speech software work together by speaking phrases when the mouse is clicked over the icon linked to that phrase. Thus the communication program gives the freedom of speech to anyone using the wheelchair for free range of motion.

1. BACKGROUND

During the summer of 2001 I had the opportunity to work at the University of Pennsylvania as part of an REU (research experience for undergraduates) program. I worked with Dr. Jim Ostrowski, associate professor in the Department of Mechanical Engineering and Applied Mechanics, in the GRASP (General Robotics, Automation, Sensing, and Perception) lab. Dr. Ostrowski, overseer of many projects in the GRASP lab, along with a group of Penn graduate students, developed the very interesting "smart" wheelchair project during the eight months prior to my work. During my ten weeks at Penn, I designed and implemented a text-to-speech application as a supplementary application for the mobile wheelchair. I also worked on a finger mouse that signals to the computer which function the user wants to execute. My speech program uses the computer's default voice to speak certain phrases when the finger mouse selects the appropriate icon on my communication display. The finger mouse can also be implemented into the navigation program for the wheelchair.

The smart wheelchair project is the development of an autonomous wheelchair, which interweaves human and computer control for navigation and communication. The wheelchair in the GRASP lab is equipped with a desk surface across the chair, and computer programs are projected onto the surface from a camera mounted above the chair. See figures 1 and 2 [1].

Figure 1. Smart Wheelchair at the University of Pennsylvania in the GRASP lab.

Because the camera is sensitive to light and dark images, the mobile chair can be operated by having a person cover white squares on a display board with his or her black-gloved hands. See figures 2 and 3 [1]. When a white square is covered by a black glove, the camera and computer can interpret what square is covered. The program code is set up to execute an application when more than half of an "application square" is covered along with more than half of the "click" square (comparable to clicking a computer mouse when the cursor is covering the program to be executed). The operator first covers the square signifying the function he or she wants to execute and then covers the click square to execute that function. Covering two squares is not the most efficient way to select an operation. Therefore, the first part of my project involved developing a "mouse", small enough to wear on the user's finger, to eliminate the click square. The finger mouse is a switch button that signals the computer when it is pressed, and is currently only being used with my speech program.

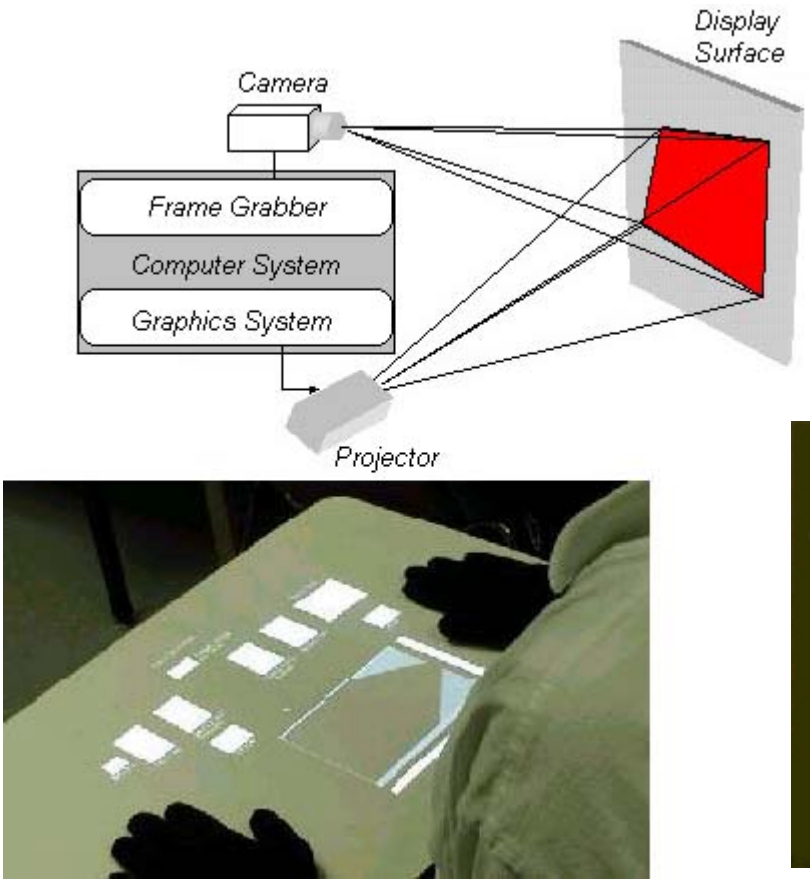

Figure 2. The camera and projector work together to produce the image displayed below to a person sitting in the wheelchair.

Figure 3. Navigation display for mobile wheelchair. Click square is in the middle of the display, to the left.

Dr. Ostrowski and the students have designed a program to control the chair's motors to move about the GRASP lab. For instance, simultaneously covering a square labeled "Turn right" and the click square causes the chair to make a right turn. Their goal

is to create an autonomous wheelchair that has discerning capabilities as well as leaving some control to the user. For example, such commands as forward, back, turn right, and turn left are controlled by the user, but the computer contributes its own assistance during navigation. An interesting innovation is the retrace command, which comes in handy when a person gets stuck in a corner or tight spot where there is no room to turn around. In such a situation, the user need only select the retrace command to undo his or her wayward movements. For a more detailed demonstration of how the chair operates, a short movie clip filmed in the GRASP lab can be viewed by visiting the smart-chair web page [1].

To expand the wheelchair's movement capabilities, the project's next goal is to use an omni-directional camera and Sick laser to maneuver to any point in the wheelchair's surroundings. The laser is mounted at the foot of the chair as a guide, especially for avoiding obstacles in the path of movement. The camera takes a 360-degree picture of the room, which is displayed onto the desk surface to the user. The graduate students are working on the challenge of being able to point to a place in the picture and have the chair navigate itself there. Difficulties include not only the navigation itself but also the ability to recognize where the user is pointing. The graduate students, Rahul Rao in particular, are trying to determine how to program the camera to tell where the user's hand is when he or she selects a doorway in the room to move to or covers an application square. To aid all the wheelchair's programs, they are trying to determine where the user's hand is if it covers two squares at once. For now, a person must be careful to cover only one application square at a time [1]. Therefore, I arranged the icons on my communication display to make it easy for a user to cover only the desired application.

2. OVERVIEW

Using both hands to execute an application is not ideal. Therefore, the wheelchair group originally looked into creating a small mouse for the user to click when the correct square is covered. This mouse interacts with the wheelchair application as the "click" square does now. However, the first mouse was very susceptible to noise from the machinery in the lab. Therefore, that unfinished part of the project was put on hold while the group focused on navigating the chair. I took a second look at the finger mouse idea over the summer, giving the students the freedom to continue with their work. I began refining the design of the wireless mouse to make the signal more resistant to interference from the equipment in the GRASP lab.

I also developed a new program for the wheelchair that uses functions in the Microsoft Speech software design kit (SDK) 5.0 to "talk." Many disabled people who need the help of a wheelchair to move about also need help communicating orally. My project adds a second feature to the chair: a new display for interaction, similar to the navigation display. See figure 3 [1]. I made icons that were then linked to phrases. My code and the SDK allow these phrases to be spoken when the mouse is clicked over the associated icons. My project will open the gateways of communication at the touch of a button.

3. FINGER MOUSE

For the finger mouse portion of the project, I worked with Terry Kientz, a lab technician in the MEAM (Mechanical Engineering and Applied Mechanics) Department. He designed a preliminary finger mouse and guided my work on the next generation. The new mouse has a form-fitting design, with a switch button under the user's index finger. The small circuit board is mounted on Plexiglas and then fashioned to a band of Velcro, making the finger mouse adjustable to any finger. Also, the switch is connected to a flexible piece of brass that curves around the tip of the user's finger to adjust to the most comfortable feel. See figures 4 and 5. When the switch button is pressed, a signal is sent out from the transmitter and picked up by the receiver. Terry and I programmed a microprocessor PIC chip to transmit a unique signal from the finger mouse. This signal is recognized by the receiver with the help of another programmed PIC chip. When the receiver detects the correct transmission, a bit of data is sent through the parallel port of the computer to execute the desired application. This bit is read in the program code, and the application executes. The mouse can be used in all the wheelchair applications, including the communication program I designed.

Figure 4. First version of the finger mouse.

Figure 5. Current finger mouse.

3.1 Need for Wireless Mouse

It would be optimal to have a wireless mouse, small enough to fit on one's finger, to replace the click square used now with the black gloves. There are several advantages to the size and independence of the wireless finger mouse. First, a person sitting in the wheelchair needs to use only one hand when selecting a phrase or direction of motion. This frees up the other hand for making gestures, such as waving, while the computer says "hello" to a friend. In addition, because the finger mouse fits on the end of one hand's index finger, it uses minimal space on the desk surface, for which a regular computer mouse is much too large. The finger mouse is a compact switch button on the

pad of the finger and a small circuit board resting atop the finger. Therefore, it is conducive to *true* "point-and-click" actions. When the person wants to move about or speak, he or she needs only to *point* with a finger and *click* the button.

3.2 Circuit Design and Transmitter

When the switch button is pressed, a signal is sent out from the transmitter within the mouse circuit board [2]. See figure 6 for circuit schematic.

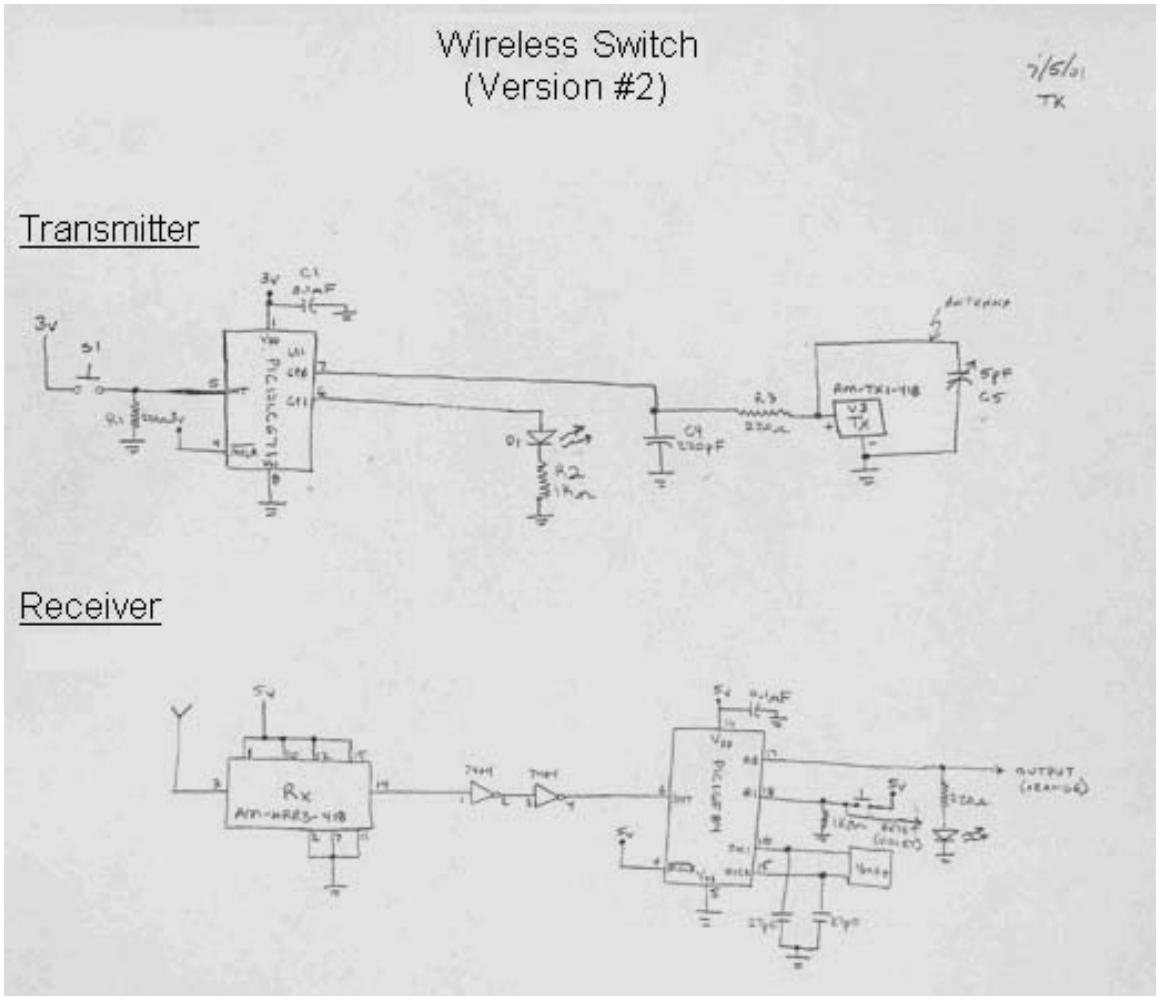

Figure 6. Finger Mouse Circuit Schematic.

This transmitter is connected to a microprocessor PIC chip, which creates a unique signal from the transmission. A microprocessor is in effect a very simple computer. The PIC chip is programmable and therefore adaptable to numerous engineering applications, including computers. It has memory ports for storage, which can be written to or read from. Also, the 1/3" chip has input and output pins that feed data in and extract data out after the bits have been manipulated by program code. Every microprocessor has its own programming language, called an assembly language, though all are similar [3]. Becoming fluent in the chip's particular language is key to cracking the code for a given project. Fully understanding the parameters needed to execute a particular program is the

highest hurdle to leap. Once over that hurdle the programmer can adapt the capabilities of the chip to fit his or her needs.

Our needs for the mouse involved creating a unique signal that could be recognized by the receiver. We decided on a signal of four square pulses one fourth of a millisecond each. A larger number of pulses could have been used, as well as a longer or shorter pulse length. We chose the size and number for trouble-shooting and preliminary design purposes. The design can be expanded later with a change to the PIC chip code. In between the four pulses are sections of off time of equal length, adding up to half a millisecond time period. See figure 7. In order to prevent a missed click, this signal is continuously repeated while the button on the mouse is held down. If the string of four pulses were sent out only once when the mouse was clicked, the receiver might not recognize the single transmission correctly, thus missing the click and not executing the desired application. Therefore, the code has been modified to account for error or noise in the signal. The transmitter and PIC chip work extremely well together. The signal is neat, clean, and stable.

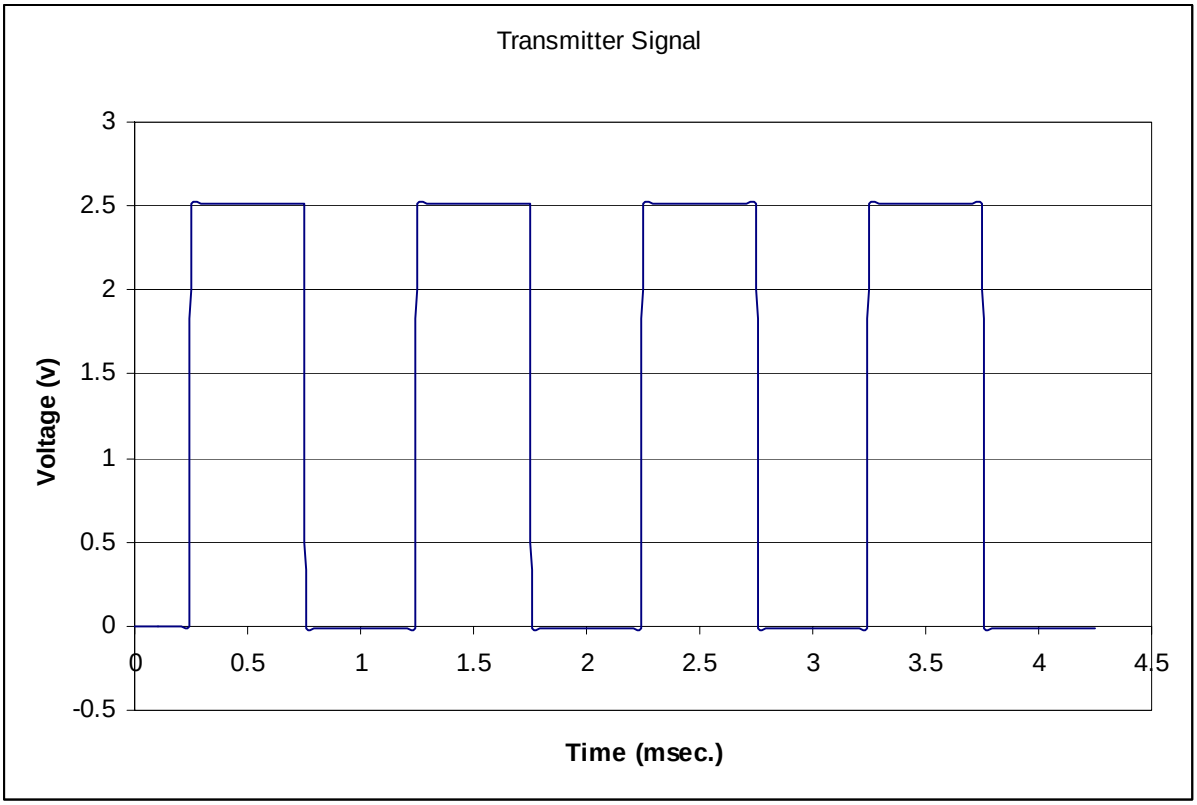

Figure 7. Graph of Transmitter Signal.

3.3 Receiver and Parallel Port

Once the transmitter sends its unique signal through the PIC chip, the receiver picks it up. Another PIC chip in the receiver circuit recognizes this signal and disregards any noise. The chip on the receiver side counts the width and number of pulses sent out.

The measured width of the pulses is compared to a high and low limiting bound. Therefore, any pulse of smaller or larger width is disregarded as noise. After a pulse clears the width tests, the PIC chip begins to count the number of pulses of correct width. When the counter hits four, the receiver has interpreted the correct signal from the transmitter, and the counter is reset to zero to await the next set of pulses. At the same time, the receiver sets a bit of data high and sends it out to the parallel port on the computer [4]. This bit of data is represented by a variable called "Mouse" in the computer program code. Mouse is originally set to zero, and it is written within the code to look for changes from zero to one. The mouse click signal is half of the two actions that must be performed before the computer speaks. The second half involves covering the squares on the communication display.

4. COMMUNICATION DISPLAY

The second phase of the project entailed developing the communicator board to be displayed on the desk surface. My board has a page of square bitmap images with labels such as "Hello," "How are you?," and "What is your name?" for the finger mouse to select. See figure 8 [6]. This page is created by Microsoft Virtual C++ code with commands from OpenGL to import the images. Currently I can display up to twelve images without cramping the space, allowing for several phrases and a few program functions such as "Exit Program." The next step goal for the communication board is to complete the code to "flip" to additional pages, but the current page suffices for the mouse and speech applications.

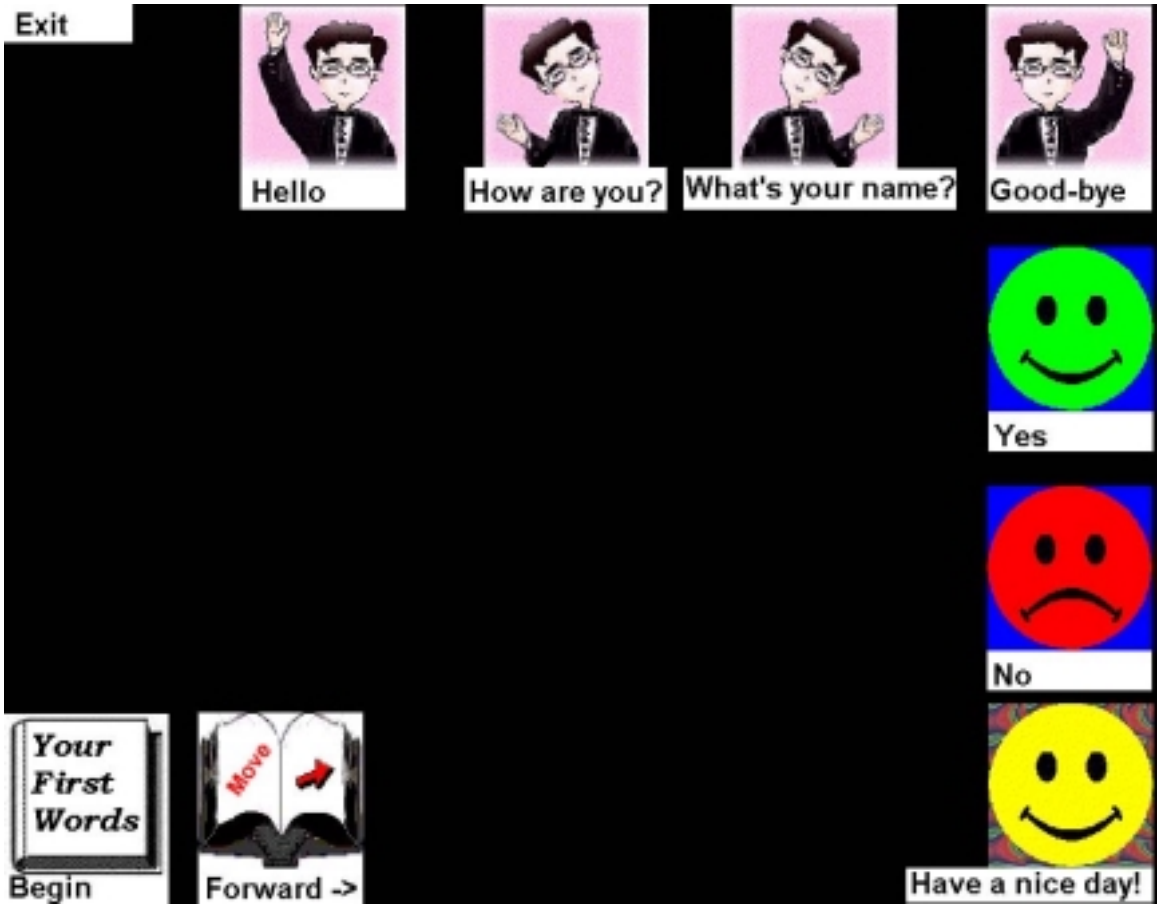

Figure 8. Introduction page of my Communication Display.

4.1 C++ and OpenGL Code

My communication display is similar to Rahul Rao's navigation display, but it has a few extra features. Like our other programs, these displays are created in Microsoft Virtual C++ code. This programmable computer language is similar to the assembly language used to program the PIC chips in the mouse design, but much more sophisticated. It can be used to program even more applications, from controlling the motors of a wheelchair, to teaching robotic dogs how to play soccer, to speaking any phrase that is written in text. A full explanation of C++ programming is not possible here. However, I will explain the sections that are essential to my project.

The C++ language understands a vocabulary of many particular words. A program is defined by these words and written in this language. We used a graphics programming guide called OpenGL to create the squares on the display. OpenGL uses variables that can be directly implemented into C code, with the library files provided by the company. This guide steps the programmer through lessons on how to create computer graphics using C++ code. The user is expected to have basic programming abilities in C++, but OpenGL explains how to expand that ability [5].

4.2 Texture Mapping

Rahul's display is made up of simple rectangles in different colors. I took the next step of researching how to import bitmap images into the rectangles created by OpenGL commands. This was accomplished by creating a slightly different kind of rectangle than previously used. OpenGL has a process called texture mapping that fit my need to paste images onto the rectangles. Texture mapping allows a user to map a picture to any type of object created in OpenGL. As long as the object is well defined, the picture can be wrapped around it without having to define the picture any further. For example, a 360-degree picture of an entire room could be taken with a rounded mirror and camera, and then be pasted onto a goblet created in OpenGL. Because the goblet is defined by the OpenGL code, the picture is pasted so that the goblet perfectly reflects the room as if it were placed on a table in the middle of it. Our needs with the wheelchair do not call for such elaborate displays. I only needed to master how to paste a flat, square bitmap onto a flat, square texture rectangle [5].

After reading the chapter on texture mapping in the OpenGL guide and searching online for any help on the subject, I successfully imported my first bitmap. The code is as follows:

```
AUX_RGBImageRec *LoadBMP(char *Filename)          // Loads A Bitmap Image
{
  FILE *File=NULL;                                // File Handle
  if (!Filename)                                  // Make Sure A Filename Was Given
  {
    return NULL;                                  // If Not Return NULL
  }
  File=fopen(Filename,"r");                       // Check To See If The File Exists
  if (File)                                       // Does The File Exist?
  {
    fclose(File);                                 // Close The Handle
    return auxDIBImageLoad(Filename);             // Load The Bitmap
                                                  // And Return A Pointer
  }
  return NULL;                                    // If Load Failed Return NULL
}

int LoadGLTextures(char *filename, int texNum)    // Convert To Textures
{
  int Status=FALSE;                               // Status Indicator
  AUX_RGBImageRec *TextureImage[1];               // Create Storage Space For Textures
  memset(TextureImage,0,sizeof(void *)*1);        // Set The Pointer To NULL
  if (TextureImage[0]=LoadBMP(filename))          // Load The Bitmap
  {
    Status=TRUE;                                  // Set The Status To TRUE
    glGenTextures(10, &texture[texNum]);          // Create The Texture
      // Typical Texture Generation Using Data From The Bitmap
    glBindTexture(GL_TEXTURE_2D, texture[texNum]);
                                          8
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, TextureImage[0]->sizeX,
                 TextureImage[0]->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,
                 TextureImage[0]->data);
  }
  if (TextureImage[0])                            // If Texture Exists
  {
    if (TextureImage[0]->data)                    // If Texture Image Exists
    {
      free(TextureImage[0]->data);                // Free The Texture Image Memory
    }
    free(TextureImage[0]);                        // Free The Image Structure
  }
  return Status;                                  // Return The Status
}
// End... loading textures
```

Every time I call the function LoadGLTexures, I simply send the filename of a bitmap and the image is pasted on the rectangle texture I made with other OpenGL

commands [5]. The first page of the display has seven bitmaps that correspond to words or phrases and three bitmaps for functions. See figure 8.

Accompanying the bitmap images are the written text of what is spoken or executed. These text messages are written on top of OpenGL rectangles. As the rectangles in Rahul's display interact with the gloves and camera to execute a movement in the chair, my rectangles with the text interact with the gloves, camera, and finger mouse to execute speech from the chair's computer. The bitmap images are not interactive because of the limits of the camera. As previously mentioned, the camera is best at discerning between light and dark images and therefore can interpret when a black glove is covering a white rectangle. The multi-colored bitmap images would make it difficult or impossible for the camera to determine whether or not a black glove covered such an image. To the grayscale eye of the camera, the bitmaps look dark or even black. Therefore, it is best to link a white rectangle to the program being executed, and displaying the text messages on white rectangles achieves this nicely.

4.3 Extended Vocabulary

The seven text phrases on the display are linked to code that speaks the words when the gloves and mouse select them. If the "Exit" rectangle is covered when the finger mouse is clicked, the program closes. The two icons in the bottom right corner are for an extended vocabulary. They are comparable to the "Back" and "Forward" buttons on a web browser. Selecting them will flip through the next pages of bitmaps and phrases. The code for flipping pages is only in the very first stages of development and is not currently implemented in the speech code and display. However, I have considered how the extra pages will work with the first page and finished a second page of dialogue. See figure 9 [6] [7]. This second page consists of phrases a user would say when navigating the wheelchair. Determining how to run the speech and navigation programs simultaneously is a future goal of the wheelchair project, but first we would like to link the first and second pages of the speech display.

On the first display, the "Begin" rectangle is not linked to any function; it is just a placeholder to indicate that the user is on the first page of phrases. See figure 4. The "Begin" rectangle is therefore like the grayed-out "Back" button on a browser window. The "Forward ->" rectangle is linked to the next page of phrases, and its icon displays the topic of those phrases. The second page has an active "<- Back" rectangle with the appropriate icon displaying the topic of the previous page. The last page of phrases has an active "<-Back" button and an inactive "End" rectangle. This icon indicates the end of the pages, just as a grayed-out "Forward" button on a web browser tells the user that there are no following pages to view.

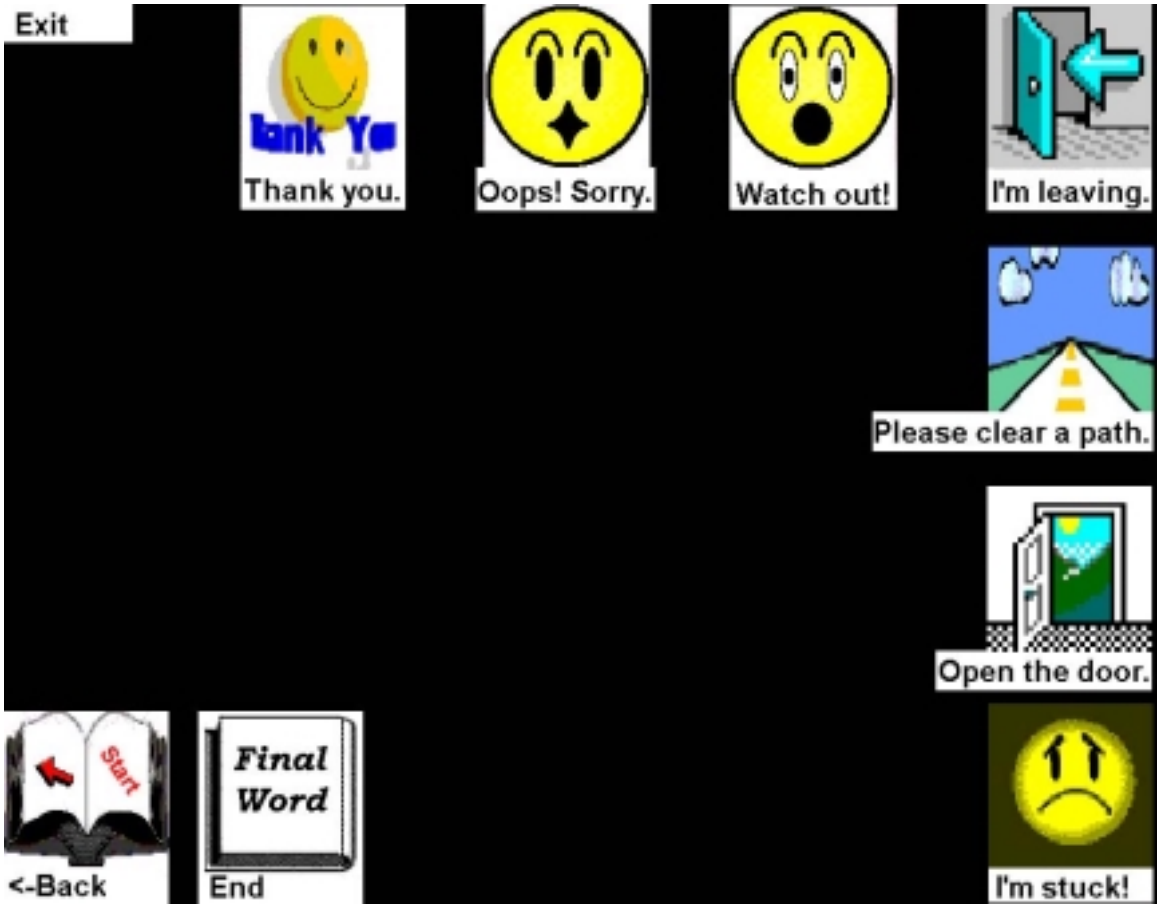

Figure 8. Navigation Page of my Communication Display.

5. TEXT-TO-SPEECH APPLICATION

The final word on the project comes from a voice synthesizer in the computer. Using the tools in Microsoft Speech SDK 5.0, I programmed the computer to speak the phrases on my communication board. A sample text-to-speech application (TTSApp) that speaks what is written in text files is provided with the SDK. Using the TTSApp as an example, I linked the labels from the display with text files to develop the "talking" part of the wheelchair. The black gloves and the finger mouse work together to select the desired icon. Then the phrase associated with that icon is "spoken."

5.1 Computer Voice Speaks Written Text

Microsoft's software design kit (SDK) works in the same way OpenGL works for images. The kit provides a library of functions that can be implemented unto C++ programs for speech applications. There are several possibilities to choose from in this kit. One could develop a speech recognition program that interprets what a user is speaking into a computer's microphone. But this software is limited by the different dialects of users in every city, country, and continent. Another, more reliable application of the kit is the text-to-speech application. These commands allow a user to create a program that speaks any text message. The specifics of how the speech is executed are

within the kit files; for this project I needed only to know how to properly use the kit's utilities.

The core code for executing a text message is as follows:

```
if((HelloPercent > 0.50) && ((Mouse !=0) ))              // Rectangle covered and Mouse Clicked
{
  if( FAILED(CoInitialize(NULL)) )
  {
    cout << "UH OH!!!" << endl;                          // If failed, Terminate the application
  }                                                      // Else full speed ahead!

  ISpVoice *g_cpTTSVoice;                                // Global pointer to TTS object
  g_cpTTSVoice = CreateTTS();                            // Create TTS object

  if( g_cpTTSVoice != NULL )
  {
    HRESULT hr;
    hr = g_cpTTSVoice->Speak(L"Hello.", 0 | SPF_IS_XML,NULL );        //Speak Hello.
  }
}
```

The first line shows that the white rectangle must be covered up along with the mouse click to execute the speech application. Therefore, we are still using one black glove, but the mouse has taken over the click square function. When the black glove covers more than fifty percent of the white rectangle and the mouse button is pressed - and only when these two events occur at the same time - the computer speaks the word or phrase linked to that text rectangle.

5.2 XML Tags

The "SPF_IS_XML" parameter in the command that executes the text commands the program to read certain tags within the text and manage them appropriately. These XML tags can modify the way the text is spoken. For example, the <SPELL> XML tag tells the program to spell out the word within the tag. The following code spells out my name:

```
hr = g_cpTTSVoice->Speak(L"<SPELL> Karla </SPELL>", 0 | SPF_IS_XML,NULL );
```

Other tags include features such as <RATE> for speeding up or slowing down a phrase, <EMPH> for emphasizing words in a phrase (as a cheerleader emphasizes words in a cheer), and <SILENCE> for setting a certain number of milliseconds to pause in a phrase. A programmer can experiment with these tags to create the most natural flow for a phrase or sentence. For a complete list and description of the XML tags provided by the Microsoft Speech SDK, see the help document [8].

6. CONCLUSIONS AND RECOMMENDATIONS

The finger mouse, communication display, and speech program have all been tested together and are fully functioning. The new mouse design is not affected by any of the first generation's noise interference problems. It is a beautiful, compact design with a clear signal and fits our needs perfectly. The communication display is completely interactive and adjustable. Expansions could be made for flipping between pages of phrases, but the two individual pages is an excellent start. The speech software package is comprehensive, and the entire text-to-speech program with its incorporation of the finger mouse is a wonderful addition to the wheelchair. Thus the communication program gives the freedom of speech to anyone using the wheelchair for free range of motion.

6.1 Frequent Errors and Their Solutions

Even though the program is working properly, resolvable errors do occur. This section describes a few of the most frequent errors and relays the solutions that worked for us.

6.1.1 Error 1: The display executes and the mouse receives the signal but the phrase is not spoken.

Solution: Make sure the external speaker volume is turned up loud enough and that the volume control on the computer is not set to mute.

6.1.2 Error 2: GLUT: Fatal error in …\Debug\glut0.exe. Failed to create OpenGL rendering context.

Solution: Check path under ControlPanel\System\Environment\User. Make certain that there is a Variable named PATH and that its Value is set to D:\pkg\glut\. Click the Set, Apply, and Okay buttons in that order. Close all programs and restart the computer. We were unable to find a quicker way to remedy this error.

6.1.3 Error 3: The memory could not be "read". Debugger will point to the line "End->Next = new CPoint(x,y,NULL);"in the file Arrays.cpp.

Solution: This error occurs when the program no longer reads one of the variable names properly. No matter how may times the program executed perfectly before, this inexplicable error may still arise. Troubleshoot the problem with feedback lines in the C++ code to determine where the program is stopping, and change the name of the variable that the program is stuck on. This is as simple as adding an extra letter to the error variable.

7. ACKNOWLEDGMENTS

I would like to thank Dr. Jim Ostrowski for being a very supportive and encouraging advisor. I appreciate his motivation and kudos as each part of this project was complete. Terry Kientz was the essential key for finishing the second generation of the finger mouse. I learned while he designed and am grateful for the experience. Many thanks to all the great students and faculty in the GRASP lab for their fellowship, direction, and expertise – Rahul Rao, Bill Sacks, Sarangi Patel, Ray McKendall, Dan Walker, and C.J. Taylor. They each helped me as I took many little and large steps along the way.

Financial support of the National Science Foundation and Microsoft Corporation is greatly appreciated.

8. REFERENCES

1. University of Pennsylvania, GRASP Laboratory, The Smart Wheelchair - An Overview. Information available electronically at http://www.cis.upenn.edu/smartchair/, 2001.

2. ABACOM Technologies, *AM Transmitter/Receiver Module Manual,* Etobicoke, ON, Canada.

3. Microchip Technology, Inc., *PIC16F8X Data Sheet and Instruction Set Summary,* 1998.

4. D. Walker, *Remote Manipulation of Mobile Robots Using Stock Radio Components,* March 2001, p. 7.

5. M. Woo, J. Neider, and T. Davis, *The Official Guide to Learning OpenGL, Version 1.1,* Addison-Wesley, Reading, Massachusetts, 2nd ed., 1997, pp. 317-350.

6. http://www.barrysclipart.com/

7. http://www.busprod.com/nlaird/smileycol.htm

8. Microsoft Speech SDK 5.0 Help document, sapi.xsd.