Summer 2009

Grocery Shopping Assistant for the Blind/Visually Impaired (GroZi)

Prepared by:

Grace Sze-en Foo (TIES Intern)

Client:

National Federation of the Blind

Summer 2009

Executive Summary

There are currently 10 million blind and visually impaired people living in the United States. While many grocery shops now offer online shopping, they are not always available. Blind and visually impaired people that require assistance in the grocery shop are still treated as “high cost” customers. Developing assistive technologies and handheld devices allow the possibility of the increase in independence for the blind and visually impaired.

The GroZi project was created in the hopes of empowering visually impaired individuals to shop at their own convenience and privacy using computer vision technology like object recognition, sign reading and text to speech notification.

In summer 2009, the team worked on several sub-segments of GroZi towards the goal of implementing a Remote Sighted Guide (RSG) interface. We explored the feasibility of using spatialized audio, packaged wiimote functions for easy integration in the RSG interface, adjusted the audio properties to enable the wiimote to play the sound file, finalized the choice of a camera and implemented an RSG interface.

Contents

Introduction

According to the American Foundation for the Blind[1], there are currently 10 million blind and visually impaired people live in the United States. At least 1.5 million Americans with vision loss use computers. Recently, a student team in the Virginia Tech College of Engineering managed to provide the blind with the opportunity to drive[2]. Henceforth, it can be seen that there is an increase in rate of the number of people in the blind community demanding to be more independent from physical human assistants. Developing assistive technologies and handheld devices allow the possibility of the increase in independence for the blind and visually impaired.

Blind and visually impaired people in the United States often face daily obstacles with routine tasks, including grocery shopping. While many grocery stores offer online shopping and home delivery, they are not always available. Blind people are then forced to request for an employee to accompany them around the store or depend on friends and family to go grocery shopping. [3] The GroZi project was created in the hopes of empowering visually impaired individuals to shop at their own convenience and privacy using computer vision technology like object recognition, sign reading and text to speech notification.

In conjunction with CalIT2, UCSD's Computer Vision Lab and TIES, the GroZi project is working to develop a portable handheld device that can "see", allowing the blind and visually impaired to navigate more efficiently within difficult environments as well as better locate objects and locations of interest. GroZi's primary research is focused on the development of a navigational feedback device that combines a mobile visual object recognition system with haptic feedback. Although still in its early stages of development, when complete, the GroZi system will allow a shopper to navigate the supermarket, find a specific aisle, read aisle labels, scan the aisle for products on the shopping list and direct the shopper to acquire the product.

The summer goal of GroZi was to implement a Remote Sighted Guide (RSG) that incorporates the use of audio and haptic feedback. In order to achieve that goal, four sub-segments of GroZi were explored: spatialized audio, wiimote hacking and software, cameras and the RSG interface.

[1] http://www.afb.org/Section.asp?SectionID=15&DocumentID=4398
[2] http://www.nfb.org/nfb/NewsBot.asp?MODE=VIEW&ID=453
[3] http://www.nfb.org/nfb/Answers_about_Blindness.asp?SnID=1052246829

Team Organization Chart

Description of Project Tasks

Formalized Stages

It is important for both users and the designing team to fully understand each and every stage involved in the seemingly rudimentary task of grocery shopping. It is also crucial for both groups to know how directions are given to the user, and in some degrees, the technology or method used in each stage. The table below illustrates the formalized stages of the grocery shopping experience. It also demonstrates what technology each stage uses, and how directions are conveyed to the blind user.

No.	Stage	Method	Command Conveyance
0	User walks into store		
1	Locate aisle	Read aisle signs	Audio/ Speech Synthesis
2	Locate product shelf	Object detection	Audio/Tactile Alert
3	Approximate product location	Object detection	Spatialized Audio
4	Detailed product location	Object tracking	Haptics/Tone Progression
5	Product selection	Object detection	Audio Alert
6	Product confirmation	Bar code scanning/detailed object recognition	Audio/Speech Synthesis

In stage 0, the blind person walks into the grocery shop, with the shopping list already preloaded into the GroZi system. Next, he/she has to locate the correct product aisle. This is done in stage 1, where the computer vision system will read the aisle signs using audio or speech synthesis.

After entering the correct aisle, the blind user then proceeds to locate the product shelf (Stage 2). The object detection system is constantly panning the product shelves within that particular aisle for the item in the shopping list. Once the correct product shelf is located, an audio or tactile alert is conveyed to the user.

In stage 3, the blind user is now standing in front of the correct product shelf where the product is located. The approximate product location, for example above, below or top left, is given using spatialized audio. Determination of the approximate product location is done using object detection. This is followed by guiding the user's hand to the product's precise location (stage 4) using object tracking. The blind user is guided by some form of haptic feedback or tone progression.

Once the blind user has his/her hand on the correct product, an audio alert is played to inform the user to select that particular product (stage 5). Object detection is used to achieve this task. Finally, it must be followed by product confirmation (stage 6), where either bar code scanning or detailed object recognition is used to check if the item chosen is correct or wrong. The accuracy is conveyed to the blind user using audio or speech synthesis.

Spatialized Audio

Tommy Mueller and Grace Foo were the main students that worked on this sub-section.

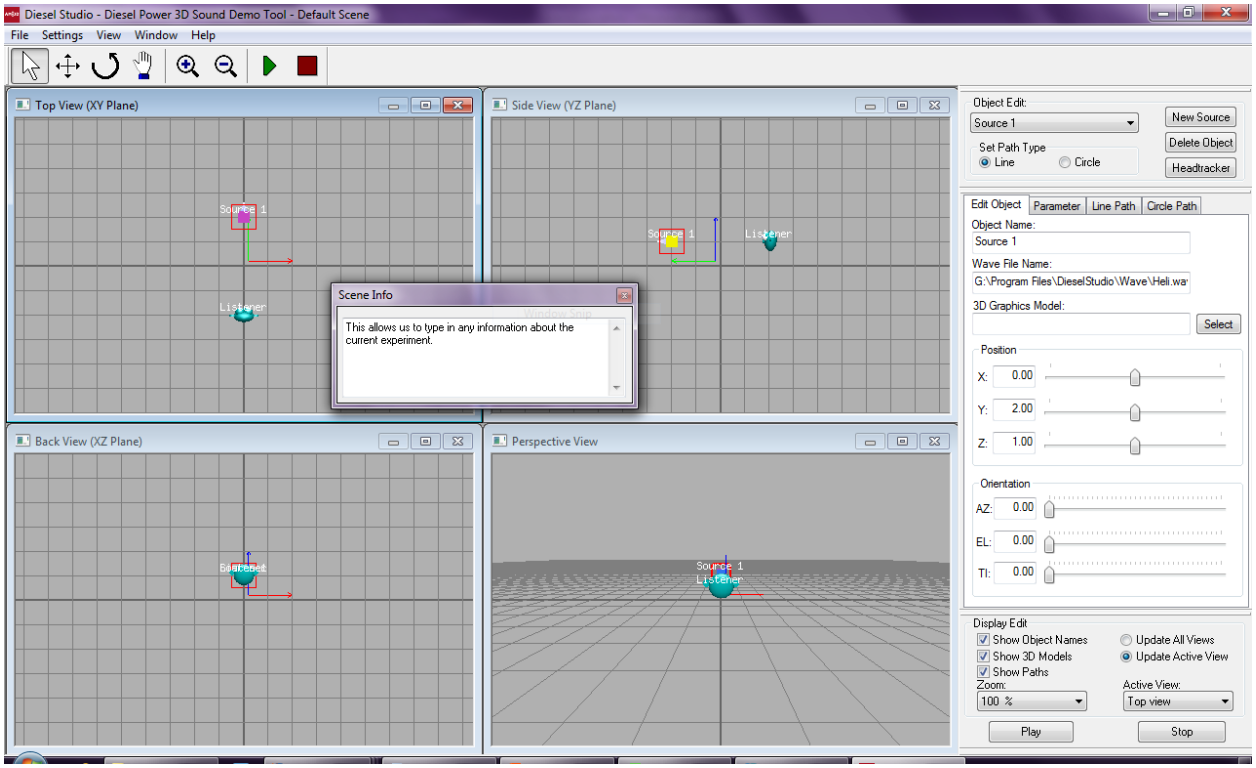
The human ear has the natural ability to perceive different sounds coming from a source at a particular position. For example, if a sound was coming towards us from the right side, the human ear is able to speculate that a sound source is coming towards us from the right. This is under the assumption that the said person has good or perfect hearing in both ears.

Spatialized audio is modeled after human's biology. Sound is processed to give the listener the impression of a sound source within a three-dimensional environment. [4] True binaural spatial audio, when presented from headphones or earphones, appears to come from a particular point in the space outside of the listener's head. [5] Some examples of spatialized audio include a virtual tour of London city[6] and a visit to the neighbourhood barber shop[7].

Instead of using conventional two-dimensional audio that would've been played via the wiimote built-in speakers in usual circumstances, we decided to experiment with this technology that is relatively new. The use of conventional two-dimensional audio would include words like in front, above, below, left, right, top left, top right, bottom left and bottom right. Unfortunately, although the wiimote has basic audio functionality, so far the sound quality from wiimote has been low and muffled. For audio files that contain words, the words are practically inaudible, unless you know the words beforehand and have been listening to the audio file play multiple times on the wiimote. Thus, with the current sound quality of the wiimote built-in speakers, it is not feasible to use the wiimote built-in speakers to inform the blind user of the approximate location of the product relative to where they are standing (Stage 3).

The software that we used for testing is a demo program, Diesel Studio. The demo version of Diesel Studio can be downloaded for free from: http://www.am3d.com/526fabbc-7eb1-45b7-8db5-344342d9a894.W5Doc. Diesel Studio uses Head-Related Transfer Functions (HRTF), making it possible to perceive a sound source in any direction. Although a 3D sensation can be experienced using a headset, 2 or 4 speaker system, to ensure full immersion, we utilized earphones in all our tests.

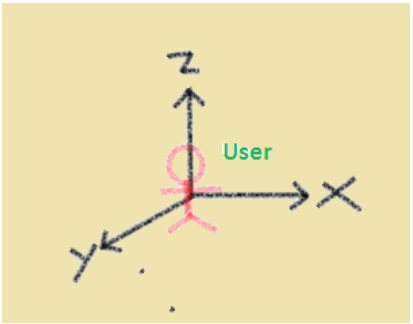
[4] http://www.noisebetweenstations.com/personal/essays/audio_on_the_internet/Spatialization.html
[5] http://www.cc.gatech.edu/gvu/multimedia/spatsound/spatsound.html
[6] http://www.qsound.com/demos/london-tour.htm
[7] http://listenwithyourownears.com/3d-audio-demo-showdown/



Diesel Studio

Experimental Design

The coordinate system used in Diesel Studio is the x, y and z axis. The x-coordinate signifies the horizontal movement in front of the user. The y-coordinate signifies the movement of sound moving towards or away from the listener. The z-coordinate signifies the vertical movement in front of the user. The figure below illustrates the coordinate system. The y-coordinate is fixed in this experiment as it is assumed that the blind user is static and standing in front of the product shelf.



The goal of the experiment was to determine the accuracy of determining the sound source using spatialized audio.

Tommy and Grace as Test Subjects

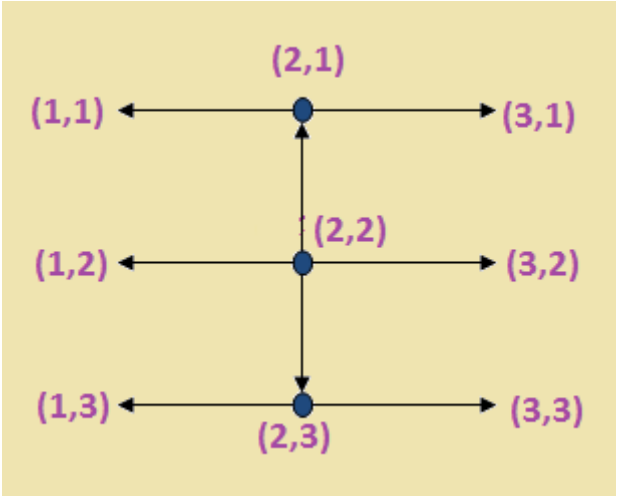
Before testing spatialized audio with John, Tommy and Grace tested the software together. Tommy and Grace alternated between being the controller on Diesel Studio who determines the location of the sound and being the listener who identifies the location of the sound. A simple audio splitter is used so that two earphones can be used at the same time to listen to the same sound.

2-way Audio Splitter

A few simple steps are followed:

1. Controller sets x-, y- and z-coordinates of the sound source with earphones put on.
2. The listener stands in front of the whiteboard at arm's-length with earphones put on. Make sure that the earphones are put on correctly (Left earphone in left ear, right earphone in right ear).
3. When both parties are ready, the controller plays the sound and the listener marks it on whiteboard.
4. The results are recorded.

Two modes were tested, the former without calibration (the sound plays without first centering it) and the latter with calibration (the sound plays from the center, then migrates towards the final location). Without calibration, the audio file played should be a "ding" while the calibrated test uses the sound of a helicopter moving. The sound source that is calibrated before migrating towards its final location must follow the system below:



The sound source always starts at (2,2) for calibration. For sound sources that are going to (2,1), (1,2), (3,2) and (2,3), the sound source will take 4 seconds to migrate from (2,2) to the respective coordinates. For sound sources that are migrating from (2,2) to (1,1), (3,1), (1,3) or (3,3), the sound source will take 2 seconds to migrate from (2,2) to (2,1), (1,2), (3,2) or (2,3), and another 2 seconds to migrate from those coordinates onwards to the corner coordinates. The paths are set using the "Line Path" option in Diesel Studio.

Below are the results for Tommy/Grace without calibration:

Tommy: 4/7 accuracy. Out of the 3 that were wrong, 2 were up/down mistakes.
Grace: 3/6 accuracy. Out of the 3 that were wrong, all 3 were up/down mistakes.

With calibration, Tommy was able to get 100% accuracy.

Dr. John Miller as test subject

The way the experiment was conducted closely resembled the way the experiment was conducted with Tommy and Grace. The steps are as follows:

1. Controller sets x-, y- and z-coordinates of the sound source with earphones put on.
2. The listener stands in front of the whiteboard at arm's-length with earphones put on. Make sure that the earphones are put on correctly (Left earphone in left ear, right earphone in right ear).
3. When both parties are ready, the controller plays the sound and the listener marks it on whiteboard.
4. The results are recorded.

However, in the experiment conducted with John, it was determined that testing would only incorporate calibration as the accuracy without calibration was too low (from results for Tommy and Grace).

A 5-way audio splitter was used during the experiment so that everyone that attended the GroZi meeting could listen together. Unfortunately, it appeared that a 5-way audio splitter also affected the quality of the sound, resulting in John not being able to hear the sound source clearly. Hence, the 5-way audio splitter was removed and only the listener (John) had earphones on.

Below are the results for John with calibration:

Try #	Location	Result	Accuracy
1	(1,1)	??	Wrong
2	(1,3)	(1,1)	Wrong
3	(3,2)	(3,2)	Right
4	(3,1)	(3,1)	Right
5	(1,1)	(1,1)	Right
6	(1,3)	(1,3)	Right
7	(3,3)	(3,3)	Right
8	(3,1)	(3,3)	Wrong
9	(2,3)	(2,3)	Right
10	(2,1)	(2,1)	Right

John achieved 70% accuracy in this test. From the results, several conclusions can be drawn:

1. With training, accuracy can be improved. Therefore, listeners should be trained before using this technology.
2. It is important to concentrate while the sound source is playing. In try #8, John lost concentration by talking to us, and hence was unable to tell if the sound source was up or down.
3. It is harder to tell if a sound source is coming from above or below the listener, as compared to the horizontal direction.

Since spatialized audio is heavily reliant on the assumption that the user has good or perfect hearing in both ears, the current technology level will prove inadequate in the case of a user that has only partial hearing in one ear, or has hearing problems.

Cameras

Grace Foo was the main person that worked on this sub-section.

The role of the camera is to provide images of the blind user's view. With regards to camera selection, the process started off from modeling it after the MAE156B Spring '07 team's camera choice – the Swann SW-P-MWC Mini Wireless Camera. This was because they were the only other GroZi team that had successfully implemented a Remote Sighted Guide (RSG) was the MAE156B team. The Swann SW-PMWC is a security camera, about the size of a thumb. According to the MAE156B team, the small camera has a large enough resolution to identify items clearly from 20 feet away.

Swann SW-P-MWC Mini Wireless Camera

One of the main discoveries from the research on cameras was that the Swann SW-P-MWC operates in the 2.4GHz bandwidth. While the 2.4GHz bandwidth might have been relatively unpopulated territory in 2007, that bandwidth is now cluttered with 802.11 wireless LAN (WLAN) systems, Bluetooth devices, home-RF solutions, cordless phones and a flurry of other unlicensed wireless systems.[8] With the increased use of the 2.4GHz band, a valid concern has arisen regarding mutual interference between users of the bandwidth. Theoretically, as long as there are no other 2.4GHz wireless systems installed in the same environment, the device should be able to operate properly without a fear of interference. However, as the wiimote communicates with the computer using Bluetooth, which operates in the same band, with such close proximity to each another; it is highly likely for the camera system and wiimote system to interfere with each other.

This led to the search for a camera system that operates in a different bandwidth. It was discovered that 5.8GHz is a bandwidth above the 2.4GHz band that is relatively uncluttered. It has been described as being ideal for the wireless transmission of both audio and video signals. With a 5.8GHz wireless cameras and a 5.8GHz receiver connected to a monitor, we can expect a signal range from a 300 feet range onwards.

However, it was revealed later that most systems, at least for cordless phones currently on the market, use a hybrid transmission. For example, if it claims to operate in the 2.4GHz band, they use 2.4 GHz to send a signal from the base to the handset, but use 900MHz to return the signal from the handset to the

[8] http://www.commsdesign.com/csdmag/columns/programming/showArticle.jhtml?articleID=16502631

base. [9] The same concept is applied to the 5.8GHz frequency. It has not been determined yet if the security camera companies use the half-half strategy, but if we were to spend more money by getting a 5.8GHz wireless camera, it is crucial to ensure that the system sends and receives at the same frequency.

While using higher frequencies do give us extra clarity of signal, keeping in mind also that the 2.4GHz is too crowded, this is done at the expense of losing a little range. This however, might not be a huge problem for GroZi, as the Remote Sighted Guide (RSG) will probably only be in the next room or at least in close proximity. Besides, RSG is a short-term solution that is merely substituting the automated computer vision (CV) system until the CV system is fully functional.

Taking into consideration price and size, it was then decided that the Mini Wireless Camera (Item # SV-C1182) from spyville.com [10]was the best choice if we were to go along with 5.8GHz. It looks similar to the Swann SW-P-MWC Security Camera, with a camera dimension of 1" x 2" x 0.75". The price of $150 includes a camera, receiver with antenna, 2AC adapters and a RCA cable.

5.8 GHz Mini Wireless Cameras

Both the Swann SW-P-MWC and Spyville 5.8GHz Mini Wireless Camera transmits via RCA, hence we also need a USB Frame Grabber to capture the live feed on the computer. The DigiVue EDV-ADSTK has been proven to work for the MAE156B RSG interface and is reasonably priced, so we decided to use the same product.

DigiVue EDV-ADSTK

[9] http://www.epinions.com/content_1277730948
[10] http://www.spyville.com/mini-wireless-camera-5-8ghz.html

Although we were able to identify the two suitable wireless cameras (2.4GHz and 5.8GHz) and the USB frame grabber to be used, we had to keep in mind what this system should be transferrable to computer vision, namely OpenCV compatible. OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision. From current research showcased on the internet, so far no one has tested and proven that the DigiVue EDV-ADSTK frame grabber software will work with the OpenCV library. In fact, based on the OpenCV community forum[11], most developers stick to using USB cameras.

According to customer support at DigiVue, the EDV-ADSTK uses a generic Windows driver and most softwares work with it. Extrapolating from that, OpenCV will most probably work with the EDV-ADSTK software. Even if OpenCV didn't have a particular function to acquire frames from the camera, there is literature on the internet on developers who write their own functions or tweak with the OpenCV functions. Given the timeframe that we had to implement the RSG interface, we decided to go with tested and proven USB cameras on the OpenCV community forum. Wireless cameras were put on the backburner and but should not be eliminated as a possibility for future prototypes.

The final camera that we bought is the Microsoft LifeCam VX-6000. It is OpenCV compatible with a stable 6-foot long USB connection and has a high resolution. It has 3x digital zoom and has pan and tilt functionality. It has a reasonably high frame rate (30 frames/second), ensuring that video images are smooth and seamless. Additionally, it also has a 71 degrees wide angle lens which allows for a larger view of the product shelf.

Microsoft LifeCam VX-6000

[11] http://opencv.willowgarage.com/wiki/FullOpenCVWiki

Wiimote

Grace Foo was the main person that worked on this sub-section.

The reason why the Nintendo wiimote was chosen for GroZi has been nicely summed up in the GroZi Spring '09 final report:

> ***The device may seem extremely complex and technologically advanced however there is technology readily available today which can perform the tasks needed for this project, such as Nintendo's Wiimote. The Wiimote is actually a very powerful device and is perfect for the GroZi Parrot application. When connected to the Nintendo Wii, the Wiimote uses its infrared sensor and accelerometers to translate position and motion to the television screen. In the parrot device, an infrared LED placed on the user's hand is tracked using the IR sensor and the accelerometers to recognize position and motion. It is also equipped with a vibrating motor and speakers. These are some of the most important functions of the Wiimote as the vibrations are used as the actual haptic feedback provided to the user, and the speakers are used to indicate successful location of the desired product. Furthermore, the Wiimote connects to the Nintendo Wii via Bluetooth communication. The GroZi team uses the Wiimote's Bluetooth capabilities to connect to a computer. With an established connection between Wiimote and computer, software can then program the Wiimote's functions to fit the parrot projects specific needs.***

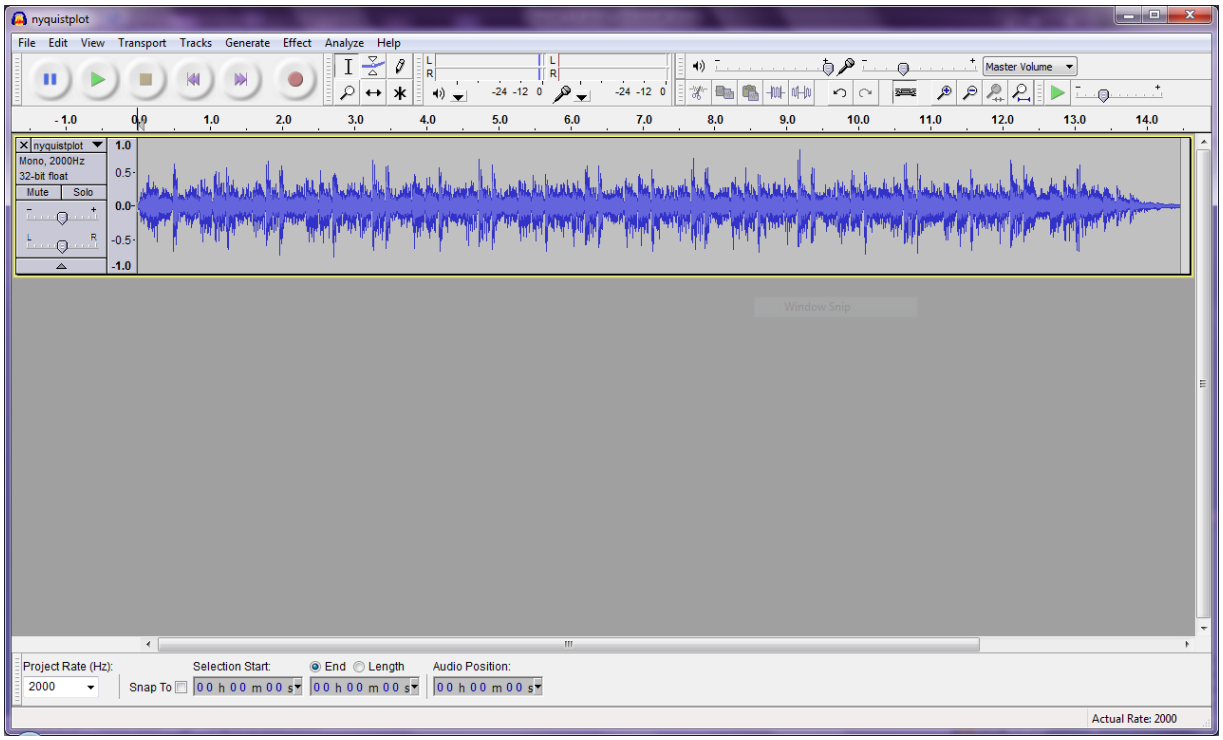
The primary functions of the wiimote harnessed by GroZi are vibration, the infrared (IR) sensor, and basic audio functions. Last quarter (spring '09), the wiimote team worked on connecting wiimotes to a laptop computer using Bluetooth and develop software which exploits various functions of the wiimote. The software had several capabilities:

1. When the wiimote senses infrared (IR), it will output the coordinates of the IR source, vibrate and play the specified audio file.
2. When there are at least 2 wiimotes connected, wiimote 2 is used to control wiimote 1. Commands include vibrating for a specified length and playing audio files.
3. Display the accelerograph of the wiimotes.
4. Output the x, y, z-coordinates, roll and yaw of the wiimote.
5. Disconnect all wiimotes when one wiimote disconnects.

This summer, the program was modified and improved by removing unnecessary codes and then packaged into an executable file that can be implemented by the RSG interface. We also worked on correcting the audio sampling rate using a program, Audacity. Last quarter, only a crackling noise was being played by the wiimote speakers because of incorrect property settings. Finally, we also identified 7 audio cues for the 7 formalized stages (shown above).

Audio Sampling Rate

The program, Audacity, was used to edit audio files. Audacity is a free, open source software for recording and editing sounds that can be used on multiple operating systems. Audacity can be downloaded for free from: http://audacity.sourceforge.net/.



Diesel Studio

Sample rate is the number of times per second that a snapshot of audio information is captured during recording. Higher sample rates result in greater audio detail, the same way smoother motion in video is achieved with a higher frame rate.

The wiimote can use multiple sound formats at multiple sampling rates, namely signed 8-bit PCM and 4-bit Yamaha ADPCM. The 4-bit ADPCM is Yamaha ADPCM, and in 8-bit mode, the sampling frequency must be made so low that the audio quality is rather bad. [12]

In the case of GroZi, we decided to use the same settings as the sample audio file, Audio.au, which was part of the sample file provided by the wiimote library WiiRemoteJ. The settings used are as follows:

1. Project Rate : 2000Hz or lower
2. Mono
3. Sound format: signed 8-bit PCM
4. File format: .au

[12] http://wiibrew.org/wiki/Wiimote

The sound files were either downloaded from the internet or taken from our personal music library. They were then edited using Audacity.

Audio Cues

Out of the seven formalized stages identified, three stages use the wiimote to convey directions to the blind user. At stage 2, when the product shelf where the shopping item is spotted, an audio alert needs to be sent to the blind user to stop walking and turn to face the product shelf. At stage 5, an audio alert need to be sent to the blind user to stop moving their hand and select the product below their hand. Finally, at stage 6, the system needs to notify the blind user whether the product chosen is the correct item or the wrong item.

Below are the different sound files that are played:

- Stage 2 and stage 5: stop_mov_high.au
- Stage 6 (correct) : nyquistplot.au
- Stage 6 (wrong) : stop.au

Packaging of files and Code Clean Up

Since the software created last quarter was mostly a modification of the sample code written by the developer of the WiiRemoteJ library, Michael Diamond, there were also much redundant code that was originally written for debugging purposes. This summer, redundant code was deleted and comments were edited to keep the java file neat and clean.

Last quarter's work also involved two wiimotes, where Wiimote 2 controlled Wiimote 1. For summer, the RSG interface's role was to replace the function of Wiimote 2. Instead of using Wiimote 2 controlling Wiimote 1, the remote sighted guide will be controlling one wiimote using the interface developed. To do this, arguments were used to emulate the RSG interface calling the function.

When the argument is –vibrate 100, the wiimote will vibrate for 100ms. When the argument is –vibrate500, the wiimote will vibrate for 500ms.

```
if (args.length > 0) {
    System.out.println("\n Arg0 was " + args[0] +"\n");

    //vibrate for 100ms
    if (args[0].equals("-vibrate100")) {
        System.out.println("\n Vibrating for 100ms\n");
        remote1.vibrateFor(100);
    }

    //vibrate for 500ms
    if (args[0].equals("-vibrate500")) {
        System.out.println("\n Vibrating for 500ms\n");
        remote1.vibrateFor(500);
    }
```

When the argument is –play1, the wiimote will play a song (Montezuma). When the argument is –play2, the wiimote will play the audio file that signifies stop. When the argument is –play3, the wiimote will play the audio file that signifies that the correct product has been selected. When the argument is –play4, the wiimote will play the audio file that signifies that the wrong product has been selected.

```
// montezuma.au
if (args[0].equals("-play1")) {
    System.out.println("\n playing Montezuma \n");
    //put code here for playing first sound
    remote1.setSpeakerVolume(0.55);
    remote1.playPrebufferedSound(prebuf, WiiRemote.SF_PCM8S);
    while (remote1.isPlayingSound()) {
        Thread.sleep(100);
    }
}

// stop moving stop_mov_high.au
if (args[0].equals("-play2")) {
    System.out.println("\n playing sound file for stop moving\n");
    remote1.setSpeakerVolume(0.55);
    remote1.playPrebufferedSound(prebuf2, WiiRemote.SF_PCM8S);
    while (remote1.isPlayingSound( )) {
        Thread.sleep(100);
    }
}

// Confirmation of product - correct nyquistplot.au
if (args[0].equals("-play3")) {
    System.out.println("playing sound file for right product");
    remote1.setSpeakerVolume(0.55);
    remote1.playPrebufferedSound(prebuf4, WiiRemote.SF_PCM8S);
    while (remote1.isPlayingSound( )) {
        Thread.sleep(100);
    }
}

// Confirmation of product - wrong stop.au
if (args[0].equals("-play4")) {
    System.out.println("playing sound file for wrong product");
    remote1.setSpeakerVolume(0.65);
    remote1.playPrebufferedSound(prebuf5, WiiRemote.SF_PCM8S);
    while (remote1.isPlayingSound( )) {
        Thread.sleep(100);
    }
}
```

Once the sound file or vibration has ended, the program must first check that the Bluetooth connection between the wiimote and the computer has been severed before shutting down the program. If the connection is not severed completely before terminating the program, it is highly likely that the computer will have a problem reconnecting with the wiimote without first restarting the computer.

```
//close everything down and quit. VERY IMPORTANT.
if (remote1 != null) {
    remote1.disconnect();
    //System.out.println(" \n Remote 1 disconnected!!!");
    remote1 = null;
}

System.exit(0);
```

If no arguments were called, then the default action is to search for a second wiimote. If there is a second active wiimote, then the program connects to Wiimote 2. Wiimote 2 will resume the function of controlling Wiimote 1 in that situation.

```
} else {
    System.out.println("Running in default mode.....");
    //no argument was passed, put the default action for your program here, or do nothing and quit.
    remote2 = newRemote();
}
```

We managed to package the file in two formats that can be easily implemented by the RSG interface, which is written in C++. One, in the Java executable JAR format; second, in the cross platform EXE format.

The wiimote software was developed in the Java Runtime Environment (JRE) using the Java Development Program, Eclipse. Eclipse has a user-friendly interface to easily create a JAR file. The Eclipse help file to create a JAR file can be found here: http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.jdt.doc.user/tasks/tasks-35.htm. However, to make the process easier, some minor changes were made. In the "Export" window, instead of creating a "JAR file", a "Runnable JAR file" is created. Subsequently, depending on which option is chosen, Eclipse will either automatically package the required libraries into the generated JAR or extract the required libraries into the generated JAR. As a safeguard, all files or libraries that are used by the new JAR file should be placed in the same folder as the new JAR file.

To call the JAR file in cmd prompt, the following text is used:

- Java –jar wiimoteG.jar

To call the specific function in the JAR file in cmd prompt, the following text is used:

- Java –jar wiimoteG.jar –vibrate100
- Java –jar wiimoteG.jar –vibrate500
- Java –jar wiimoteG.jar –play1
- Java –jar wiimoteG.jar –play2
- Java –jar wiimoteG.jar –play3
- Java –jar wiimoteG.jar –play4

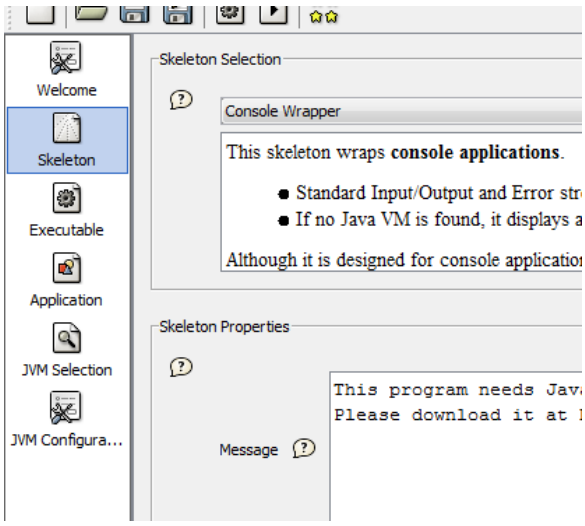
Creating an EXE file required a slightly more complicated process compared to creating a JAR file. Another application, JSmooth, was used to convert the JAR file into a EXE file. Jsmooth can be downloaded for free at: http://sourceforge.net/projects/jsmooth/. JSmooth creates standard Windows

executable files (EXE) that smartly launch Java applications. It makes Java deployments smoother and user-friendly, as it is able to find and run Java Virtual Machines (JVM) or help the user get one if none are available on the user's operating system.

JSmooth

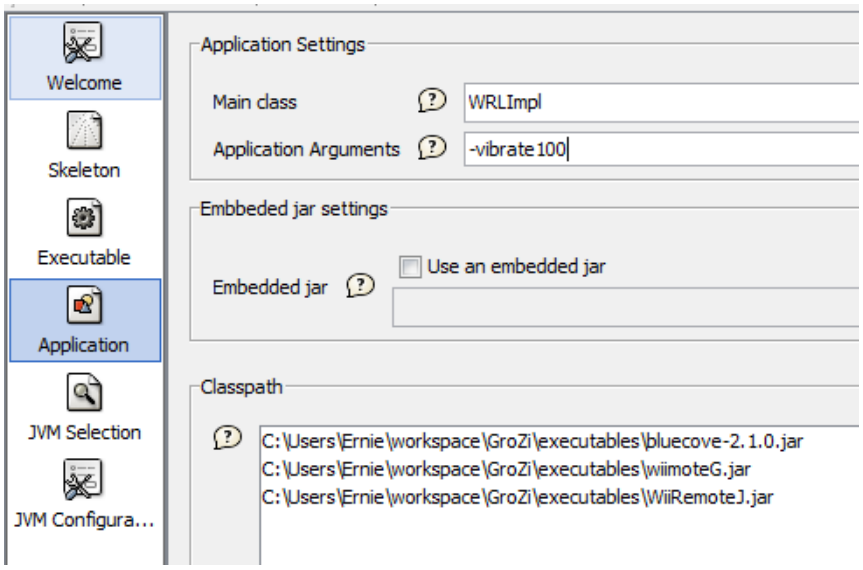
JSmooth has excellent documentation that has a step-by-step guide to creating the EXE file from a JAR file. Several notes about the creation of the EXE file from the JAR file created using the Eclipse JDK are listed below:

- The skeleton wrapper selected should be "Console Wrapper".



Skeleton Selection

- On the Application tab, the Classpaths selected must include the BlueCove (~\bluecove-2.1.0.jar) and WiiRemoteJ (~\WiiRemoteJ.jar) libraries. It also includes the JAR file (~\wiimoteG.jar) we created. The main class is the source file that contains the Main() function. Depending on what function we want the wiimote to do, the different arguments are typed into the "Application Arguments" cell. In the example below, this EXE will cause the wiimote to vibrate for 100ms when called.



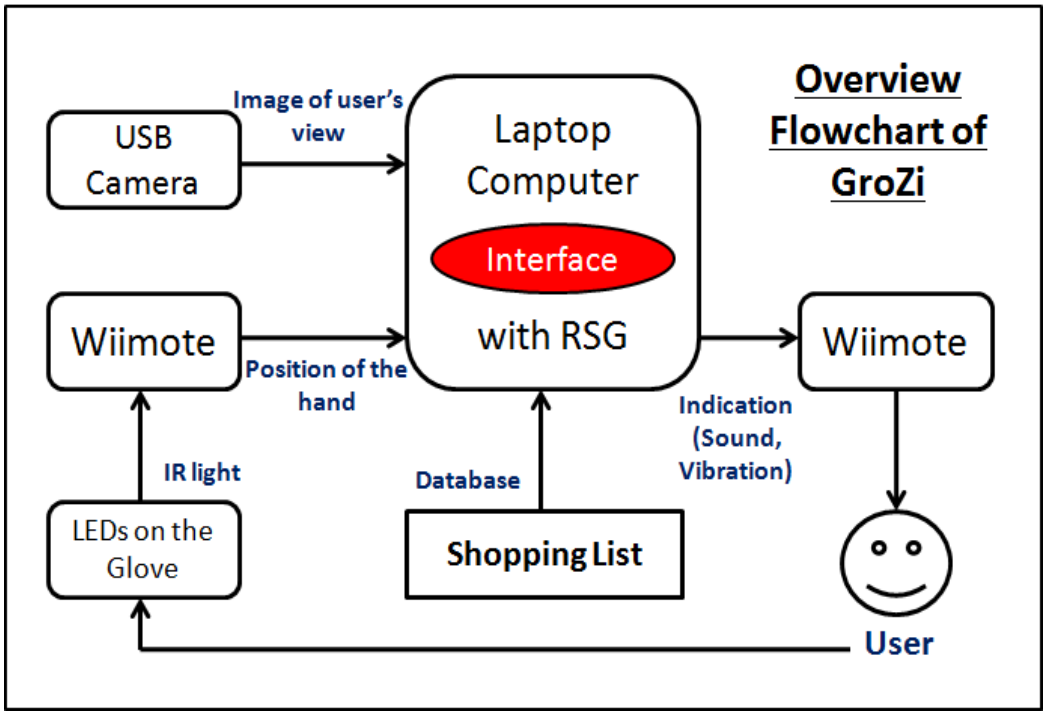
Application Settings and Class Paths

Once again, as a safeguard, all files or libraries that are used by the new EXE file should be placed in the same folder as the new EXE file.

Remote Sighted Guide

Masaaki Kokawa was the main person that worked on this sub-section.

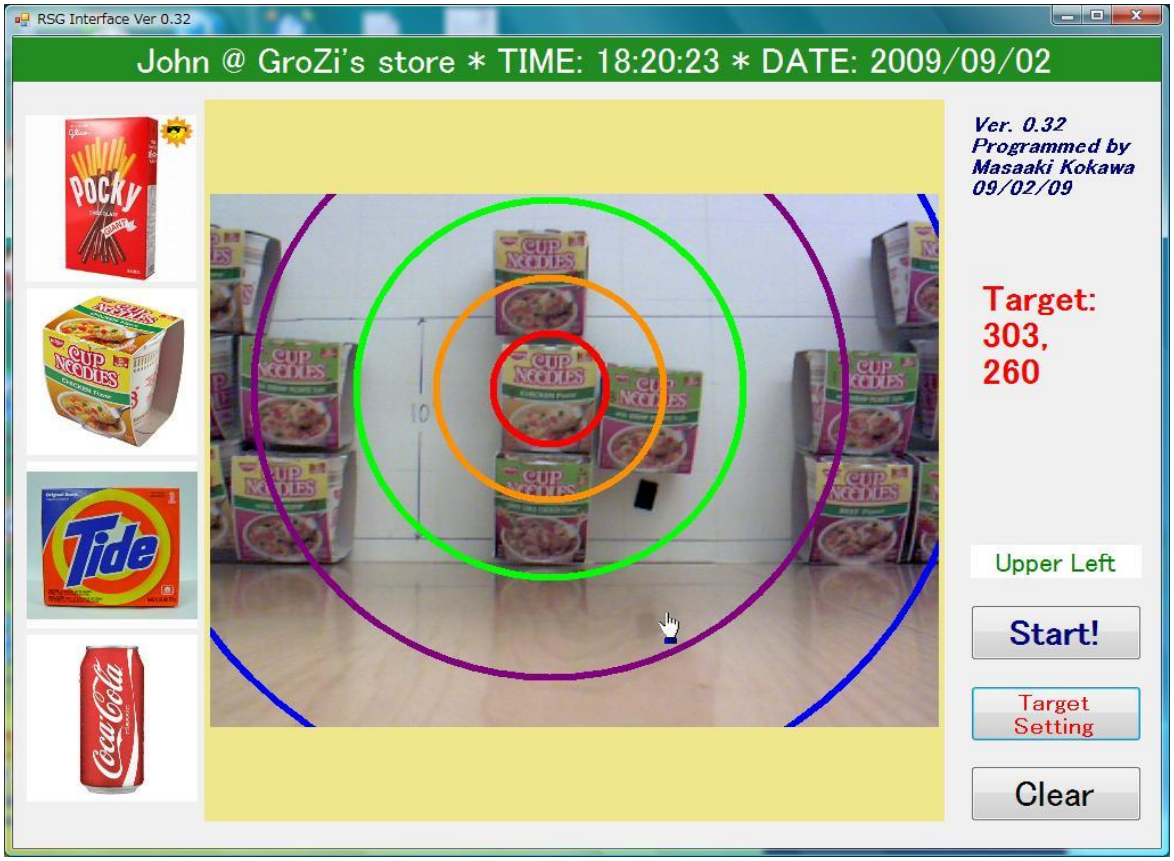
The flowchart below clearly demonstrates the relationship between the different components of our work during summer and the role of the remote sighted guide (RSG) interface.



The USB camera provides the image of the blind user's "view". Images are fed into the RSG interface. At the same time, the wiimote that is mounted on the blind user's shoulder feeds in information about the position of the hand (using the IR sensor) into the RSG interface. Another input into the RSG interface is the preloaded shopping list from the database. The RSG interface takes in those three inputs and then conveys commands to blind the user using either audio or tactile feedback accordingly.

The picture below illustrates GroZi's conceived impression of the system set up:

The figure below is a screenshot of the current RSG interface :



At the top of the GUI, the blind person's name, grocery shop, time and date are listed. On the left of the window is the preloaded grocery shopping list. In the center is the live feed from the USB camera. On the right panel of the window is the option to set a target, which will then be the center of five concentric circles. Each "zone" denotes a set radius from the target and different tones are played depending on how close the user's hand (currently simulated by the mouse pointer) is from the target. On the right panel of window is also the location of the target relative to the user's hand (mouse pointer). In the screenshot above, the target is located on the upper left relative to where the mouse pointer is.

The RSG interface was programmed using C++ in Microsoft Visual Studio 2008 and also the OpenCV library. The complete code and comments are attached in the appendix.

Future Work

This summer, the GroZi team has managed to accomplish their goal of implementing a RSG interface. We have also managed to work on spatialized audio, the wiimote and cameras. However, there is much room for improvement.

Over summer, the team has done some preliminary tests on spatialized audio using the demo program Diesel Studio. For future purposes, extensive experimental design should be implemented to fully test out the potential of using spatialized audio. Furthermore, the future team should research the possibility of writing software for spatialized audio, just like how the wiimote team created software to control the wiimote, as well as seeking ways to overcome the problem of listeners with impaired hearing not being able to use this technology.

Although the team has managed to get audio files to play correctly on the wiimote speakers, the quality is still sub-par because signed 8-bit PCM is used. As described in the Wiimote section, using signed 8-bit PCM places a constraint on audio sampling rate. Perhaps, the 4-bit ADPCM format allows for high sampling rate, resulting in clearer, crisper sounds. The future team could examine and compare the quality of audio files in 4-bit ADPCM format to signed 8-bit PCM.

Currently, the wiimote is connecting and disconnecting each time the executables (both JAR and EXE) are called by the RSG interface. This is unfeasible in the long run if the Geiger Counter concept is implemented with haptic feedback. It introduces unnecessary delay and is more prone to errors if the executable is called not only repeatedly and rapidly by the RSG interface. The future team should brainstorm ways to conquer this problem.

In addition, there is certainly a need for a mechanical design team to be formed to design a more sophisticated glove (with IR LEDs) and shoulder mount, now that all components are relatively integrated.

Once all components have been integrated, experimental design is also an essential part. The focus of each design and experiment should not only accuracy, but must also place equal importance on the time taken to select the product.

Appendix

RSG Interface Code

```
#pragma once

namespace Interface_001 {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	using namespace System::Media; // For playing wav files

(snip)

	private:
	CvCapture *capture;   // Structure for capturing a image from a camera
	IplImage *cvImage;    // Structure for storing captured images

(snip)

#pragma endregion

// Declaration of static variables in the program
	static int Start_Clear_Flag=0;      // Flag for Start / Clear
	static int Target_Set = 0;          // Flag for a status of Target Setting
	static int Target_X, Target_Y;      // Location of the Target

// For storing wav files
	static SoundPlayer^ player = gcnew SoundPlayer("data\\C#.wav");

// Initializing a Camera when a Windows Form is Loaded
	private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {

				//     Initializing a Camera
			if ( ( capture = cvCreateCameraCapture( 0 ) ) == NULL ) {
				//     No Camera Found
				label2->Text = "Error ! \nNo \nCamera \nFound";
				Target_Set = -1;
			}

		}

// Timer for getting location of a mouse cursor
	private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e) {

			Point pnt;
```

```
// Getting a position of the mouse curso in the screen
// and converting the position into the one in the window client
	pnt = this->PointToClient(Control::MousePosition);

// Variable for storing distance from the Target (for playing sound)
			double distance_snd;

// Playing different sounds according to the Target
			if (Start_Clear_Flag == 1 && Target_Set == 3 &&
rectangle_snd.Contains(pnt.X-pictureBox5->Location.X, pnt.Y-pictureBox5->Location.Y)) {
				distance_snd = sqrt(pow(pnt.X-pictureBox5->Location.X-
Target_X, 2.0)+pow(pnt.Y-pictureBox5->Location.Y-Target_Y, 2.0));
				if (distance_snd >= 260 && distance_snd < 370) {
					PlaySound("data\\C#.wav");
				} else if (distance_snd >= 170 && distance_snd < 260) {
					PlaySound("data\\D.wav");
				} else if (distance_snd >= 100 && distance_snd < 170) {
					PlaySound("data\\E.wav");
				} else if (distance_snd >= 50 && distance_snd < 100) {
					PlaySound("data\\F#.wav");
				} else if (distance_snd < 50){
					PlaySound("data\\G#.wav");
				}

// Detection of directions toward the Target
				Target_Direction (pnt.X-pictureBox5->Location.X, pnt.Y-
pictureBox5->Location.Y);

			} else {
				label4->Text = L"-----";
			}

		}

// Another Timer for displaying a Clock and Captured Images from the camera
	private: System::Void timer2_Tick(System::Object^  sender, System::EventArgs^  e) {

// Displaying the Clock
			label1->Text = "John @ GroZi's store * TIME: " +
DateTime::Now.ToString("T") + " * DATE: " + DateTime::Now.ToString("d");

			// Call the function to capture images from the camera and display
			if (Start_Clear_Flag == 1) {
				captureImage();
			}
		}

// Events when "Start!" button is clicked
	private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {

			Start_Clear_Flag=1; // the Flag is set "1"
			label2->Text = "Push \n" + "Target \n" + "Setting \n" +"Button";
```

```
                    Target_Set = 0; // The target is cleared
                    this->button3->Text = L"Target Setting";

                    // Painting pictureBox5 with color Khaki
                    Graphics^ g=pictureBox5->CreateGraphics();
                    g->FillRectangle(Brushes::Khaki,System::Drawing::Rectangle(0,0,650,650));

             }

// Events when "Clear" button is clicked
      private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e) {

// the Flag is cleared as "0"
                    Start_Clear_Flag=0;

                    Target_Set = 0; // The target is cleared
                    this->button3->Text = L"Target Setting";

                    label2->Text = L"Push \n"+"Start! \n" + "Button";

// Painting pictureBox5 with color White
                    Graphics^ g=pictureBox5->CreateGraphics();
                    g->FillRectangle(Brushes::White,System::Drawing::Rectangle(0,0,650,650));

//Writing down Instructions
                    System::Drawing::Font^ font1=gcnew System::Drawing::Font("MS UI Gothic",
24);
                    g->DrawString("Instructions", font1,Brushes::Blue,250,150);
                    g->DrawString("1. Push Start! button to start capture",
font1,Brushes::Blue,30,200);
                    g->DrawString("2. Push Target Setting button",
font1,Brushes::Blue,30,250);
                    g->DrawString("3. Click in the camera image",
font1,Brushes::Blue,30,300);
                    g->DrawString("    for tentative Target", font1,Brushes::Blue,30,335);
                    g->DrawString("4. Push OK? button to set the Target",
font1,Brushes::Blue,30,385);
                    g->DrawString("5. Click the Target in center square",
font1,Brushes::Blue,30,435);
                    g->DrawString("6. If you make it, you can see a nice guy!",
font1,Brushes::Blue,30,485);

             }

// Events when "Target Setting" button is clicked
      private: System::Void button3_Click(System::Object^  sender, System::EventArgs^  e) {

// Setting Flags
             if (Start_Clear_Flag == 1) {
                          switch (Target_Set) {
                                 case 3:
                                        Target_Set = 1;
                                        break;
```

```
                                    case 2:
                                           label2->Text = "Target: \n" + Target_X.ToString()
+ ", \n" + Target_Y.ToString();
                                           this->button3->Text = L"Target Setting";
// If a target is set, play a sound.
                                           PlaySound("data\\se_sod05.wav");
                                           Target_Set++;
                                           break;
                                    case 1:
                                           label2->Text = "Please \nSet \nTarget \nPoint";
                                           break;
                                    case 0:
                                           label2->Text = "Please \nSet \nTarget \nPoint";
                                           Target_Set++;
                                           break;
                             }
                      }
               }


// Checking mouse clicks
       private: System::Void pictureBox5_MouseClick(System::Object^  sender,
System::Windows::Forms::MouseEventArgs^  e) {

// Position of the mouse click
int mouse_x=e->X;
int mouse_y=e->Y;

// Distance from the mouse click point to the Target
double distance=sqrt(pow(mouse_x-Target_X, 2.0)+pow(mouse_y-Target_Y, 2.0));

// Seting a region near the center of the Target
System::Drawing::Rectangle rectangle_target=System::Drawing::Rectangle(Target_X-15,Target_Y-
15,30,30);

// Setting a region of the captured image
System::Drawing::Rectangle rectangle_set=System::Drawing::Rectangle(5, 85, 640, 480);

       if(e->Button==System::Windows::Forms::MouseButtons::Left){

// If the Target is caught successfully
if(Start_Clear_Flag==1 && Target_Set == 3 && rectangle_target.Contains(mouse_x,mouse_y)){

               PlaySound("data\\se_sod05.wav");

// Painting pictureBox5 with color Gold
               Graphics^ g=pictureBox5->CreateGraphics();
               g->FillRectangle(Brushes::Gold,System::Drawing::Rectangle(0,0,650,650));

// Text on label2
               label2->Text = L"Push \n"+"Start! \n" + "Button";

//Displaying "Good Job!!", the mouse click position (X, Y) and distance
```

```
        System::Drawing::Font^ font1=gcnew System::Drawing::Font("MS UI Gothic", 36);
        g->DrawString("Good Job!!", font1,Brushes::Blue,200,450);
        char buf[80];
        String^ string1;
        sprintf_s(buf,80,"X = %d, Y = %d, D = %d",mouse_x,mouse_y,(int)distance);
        string1=gcnew String(buf);
        g->DrawString(string1, font1,Brushes::Blue,50,500);

// Serge's Smile
        Bitmap^ bmap=gcnew Bitmap("data\\serge.jpg");
        g->DrawImage(bmap,165,150);

// Reset of Start_Clear_Flag to "0"
        Start_Clear_Flag=0;

// Reset of Target_Set to "0"
        Target_Set=0;

// Confirmation of the Target set
        } else if (Start_Clear_Flag==1 && Target_Set == 1 &&
rectangle_set.Contains(mouse_x,mouse_y)) {

        Target_X = mouse_x;
        Target_Y = mouse_y;
        Target_Set = 2;
        label2->Text = "Please \nPush \nOK \nButton.";
        this->button3->Text = L"OK?";

// Confirmation of the Target set
        } else if (Start_Clear_Flag==1 && Target_Set == 2 &&
rectangle_set.Contains(mouse_x,mouse_y)) {

        Target_X = mouse_x;
        Target_Y = mouse_y;

        label2->Text = "Please \nPush \nOK \nButton.";
        this->button3->Text = L"OK?";
                                        }
                                }
                        }

// Function for playing wave files
private: void PlaySound(String^ waveFile){

// Stop the sound playing now
        player->Stop();

// Read a wave file
        player = gcnew SoundPlayer(waveFile);

// Play Assynchronously
        player->Play();
                }
```

```
// Function for getting directions toward the target and display it
private: void Target_Direction(int m_pos_x, int m_pos_y) {

// Seting a region near the center of the Target
System::Drawing::Rectangle rectangle_target=System::Drawing::Rectangle(Target_X-15,Target_Y-
15,30,30);

       if( rectangle_target.Contains(m_pos_x, m_pos_y) ){
              label4->Text = L"Here!!";
       } else if ( m_pos_y >= -tan(3*M_PI/8)*(m_pos_x-Target_X)+Target_Y && m_pos_y >
tan(3*M_PI/8)*(m_pos_x-Target_X)+Target_Y ){
              label4->Text = L"Up";
       } else if ( m_pos_y > tan(M_PI/8)*(m_pos_x-Target_X)+Target_Y && m_pos_y <=
tan(3*M_PI/8)*(m_pos_x-Target_X)+Target_Y ){
              label4->Text = L"Upper Left";
       } else if ( m_pos_y > -tan(M_PI/8)*(m_pos_x-Target_X)+Target_Y && m_pos_y <=
tan(M_PI/8)*(m_pos_x-Target_X)+Target_Y ){
              label4->Text = L"Left";
       } else if ( m_pos_y > -tan(3*M_PI/8)*(m_pos_x-Target_X)+Target_Y && m_pos_y <= -
tan(M_PI/8)*(m_pos_x-Target_X)+Target_Y ){
              label4->Text = L"Lower Left";
       } else if ( m_pos_y < -tan(3*M_PI/8)*(m_pos_x-Target_X)+Target_Y && m_pos_y <=
tan(3*M_PI/8)*(m_pos_x-Target_X)+Target_Y ){
              label4->Text = L"Down";
       } else if ( m_pos_y > tan(3*M_PI/8)*(m_pos_x-Target_X)+Target_Y && m_pos_y <=
tan(M_PI/8)*(m_pos_x-Target_X)+Target_Y ){
              label4->Text = L"Lower Right";
       } else if ( m_pos_y > tan(M_PI/8)*(m_pos_x-Target_X)+Target_Y && m_pos_y <= -
tan(M_PI/8)*(m_pos_x-Target_X)+Target_Y ){
              label4->Text = L"Right";
       } else {
              label4->Text = L"Upper Right";
       }
              }

       private: System::Void captureImage(void) {

// Storing 1 frame from camera image to cvImage into cvImage
// Width:640, Height:480
       cvImage = cvQueryFrame( capture );

// If the Target is set, drawing circles which center is located in the Target
if (Target_Set >= 2) {
       cvCircle( cvImage, cvPoint(Target_X-5, Target_Y-85), 50, CV_RGB( 255, 0, 0 ), 5, 8, 0);
       cvCircle( cvImage, cvPoint(Target_X-5, Target_Y-85), 100, CV_RGB( 255, 144, 0 ), 5, 8,
0);
      cvCircle( cvImage, cvPoint(Target_X-5, Target_Y-85), 170, CV_RGB( 0, 255, 0 ), 5, 8, 0);
       cvCircle( cvImage, cvPoint(Target_X-5, Target_Y-85), 260, CV_RGB( 128, 0, 128 ), 5, 8,
0);
      cvCircle( cvImage, cvPoint(Target_X-5, Target_Y-85), 370, CV_RGB( 0, 0, 255 ), 5, 8, 0);
                    }
```

```
////////////////////////////////
// Converting the image in OpenCV format into the one for Visual C++
//
Bitmap^ bmp = gcnew Bitmap(cvImage->width, cvImage->height, cvImage->widthStep,
System::Drawing::Imaging::PixelFormat::Format24bppRgb, IntPtr(cvImage->imageData));


////////////////////////////////
// Drawing camera image
//
Graphics^ g = pictureBox5->CreateGraphics();
g->DrawImage(bmp, 5, 85, cvImage->width, cvImage->height);

delete g;
}

// Release the structure for capture when the Form is closed
private: System::Void Form1_Closed(System::Object^  sender,
System::Windows::Forms::FormClosedEventArgs^  e) {

         CvCapture *release;
         release = capture;
         if(capture) cvReleaseCapture( &release );
               }
       };
}
```