Grocery Shopping Assistant for the Blind (GroZi)

UCSD TIES—Winter 2010

Faculty Advisor:
Serge Belongie

Tutor:
Victor Correa

Community Client:
National Federation of the Blind (NFB)

Client Representative:
John Miller—NFB Representative

Visiting Scholar:
Masaaki Kokawa

Team Members:
Tess Winlock, Jeffrey Wurzbach, Jaina Chueh, Cankut Guven,
Saitejaswi Kondapalli, Edward Liu, Jeffrey Su, Kevin Tran, Andrea Wong

Table of Contents

INTRODUCTION

There are currently 1.3 million legally blind people living in the United States who face daily obstacles with routine tasks. These individuals cannot shop independently for grocery store items without sighted assistance.

Developing assistive technologies and handheld devices allows for the possibility of increasing independence for the blind and visually impaired. Currently, many grocery stores treat those that are blind as "high cost" customers, and dramatically undersell to this market, neglecting to take their needs into consideration. The use of computer vision can be advantageous in helping these blind customers, as restrictions such as the limited ability of guide dogs of white canes, frequently changing store layouts, and existing resources do not allow for a completely independent shopping experience. Using technologies such as object recognition, sign reading, and text-to-speech notification can allow for a greater autonomous solution to the relevant problem.

In conjunction with Calit2, UCSD's Computer Vision Lab and TIES, the GroZi project is working to develop a portable handheld device that can help the blind to collect information and 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, and use the handheld grocery assistant device to then scan the aisle for objects that look like products on the shopper's list (compiled online and downloaded onto the handheld device prior to going into the store).

This quarter, under the supervision of our advisor, Serge Belongie, we pursue the computer vision aspects of the project that allows for autonomous detection and localization in the near future. In the past quarter, our team successfully customized the User Interface (UI) for new labeling tasks as well as improved the computer program that allows for inserting and storing data into the database as effortlessly as possible. However, there is still room for improvement. While this improvement awaits refinement, the focus this quarter has been shifted to the integrating CVSG to develop the software that processes images of a grocery shelf model. Additionally, a special request by John Miller has the team involved in challenging themselves to build a recording program that will facilitate communication for the blind in industrial work involving visual graphics. The following document will serve as a description of what has been accomplished thus far, what has been learned and overcome, and the processes involved in designing and implementing a usable grocery assistant device for the blind to assist future members of TIES GroZi team.

Sub Teams, this quarter…

This quarter the GroZi team was divided into Sub teams:

WiiMote Team: Andrea and Jaina
Demo Board Design Team: Jeffrey S. and Edward
Protocol Team: Kevin

Gameboard Team: Saitejaswi Kondapall

Computer Vision Sighted Guide (CVSG) team: Cankut, , Teja
Hardware Design Team: Jeffrey W

Webmaster: Jaina Chueh

Official TIES GroZi website: http://ties.ucsd.edu/projects/gsa/index.shtml
Official GroZi website: http://grozi.calit2.net/

Introduction

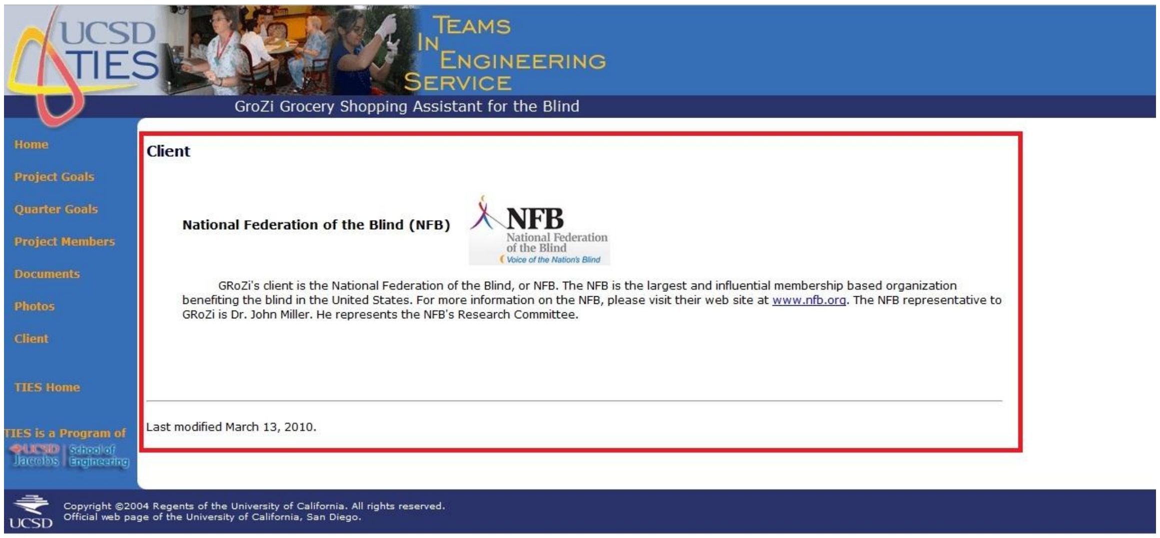
This section is written for future webmasters to gain an understanding of the setup of existing files. The task of a webmaster is to update the TIES GroZi website with the current progress of the project.

Most of the content on this website are done in SHTML. The only difference between regular HTML and SHTML is the extra letter in the extension (.shtml) and this code.

```
<!--#include file="addedfile.txt" -->
```

The S stands for 'Server Side Include' or SSI. When an SHTML webpage is sent to the web browser window, it gets assembled on the server and then sent to be viewed. The normal HTML tags all still work the same, the SHTML simply lets you INCLUDE other pieces into the HTML page.

Below is an example:


UCSD TIES
Teams In Engineering Service
GroZi Grocery Shopping Assistant for the Blind
Home
Project Goals
Quarter Goals
Project Members
Documents
Photos
Client
TIES Home
TIES is a Program of
UCSD Jacobs School of Engineering
NFB National Federation of the Blind
Voice of the Nation's Blind

Client

National Federation of the Blind (NFB)

GRoZi's client is the National Federation of the Blind, or NFB. The NFB is the largest and influential membership based organization benefiting the blind in the United States. For more information on the NFB, please visit their web site at www.nfb.org. The NFB representative to GRoZi is Dr. John Miller. He represents the NFB's Research Committee.

Last modified March 13, 2010.

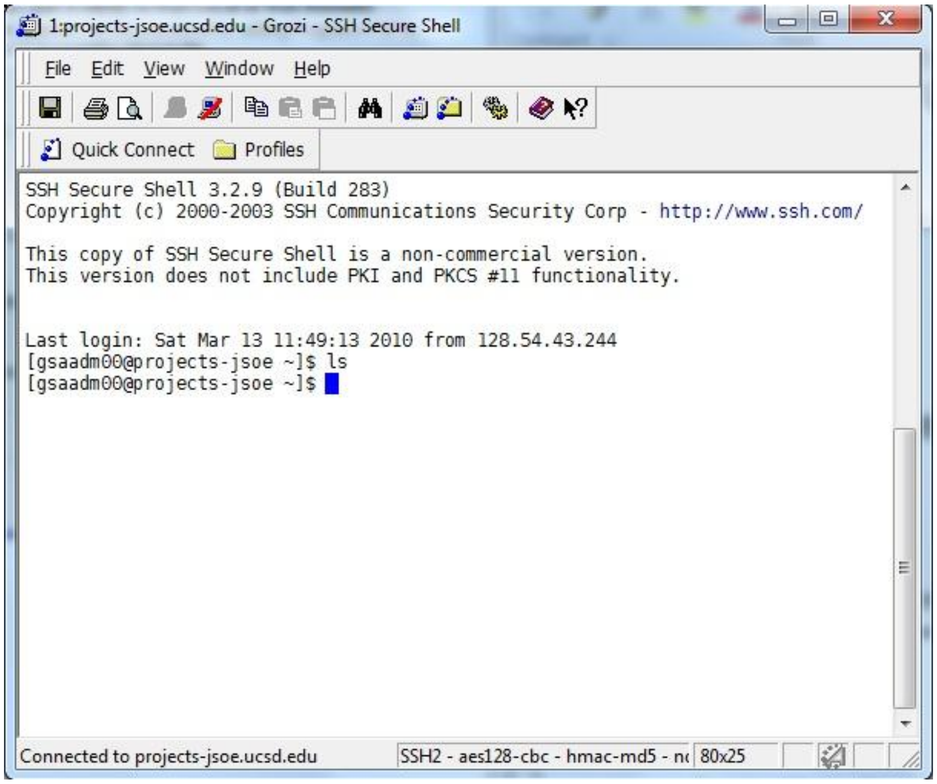
Copyright ©2004 Regents of the University of California. All rights reserved.
Official web page of the University of California, San Diego.

This is a screenshot of the Client page. The red box contains all the content the webmaster can edit in the “client.shtml” file.

Log-in

Username and password will be provided to the webmaster from TIES when the webmaster is designated.
Once you have received the username and password:
Host name: <server name>
Port: 22 (default)
Login: <username>
Password: <password>

To upload files to the server in SSH, first log in:


1:projects-jsoe.ucsd.edu - Grozi - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

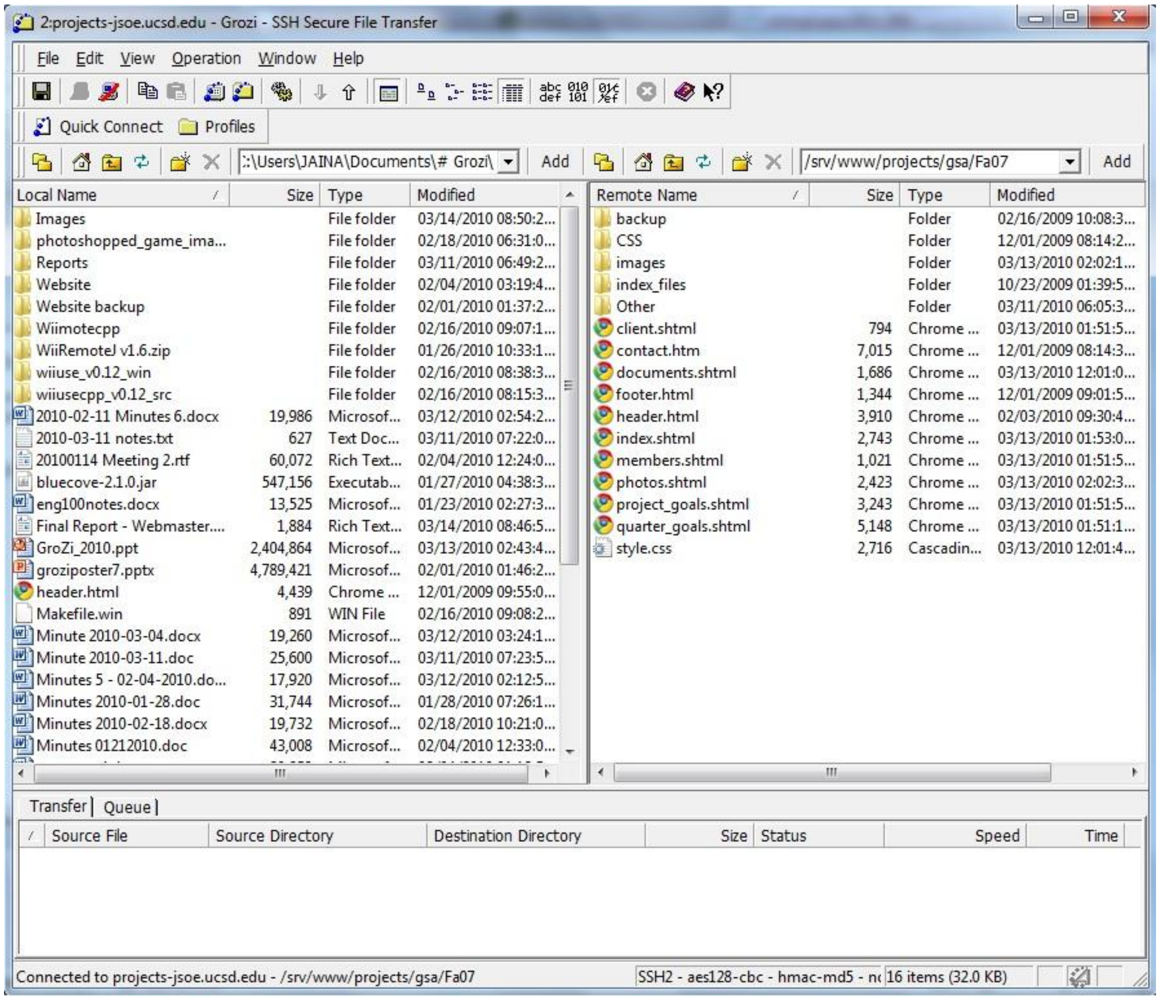
```
SSH Secure Shell 3.2.9 (Build 283)
Copyright (c) 2000-2003 SSH Communications Security Corp - http://www.ssh.com/

This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.


Last login: Sat Mar 13 11:49:13 2010 from 128.54.43.244
[gsaadm00@projects-jsoe ~]$ ls
[gsaadm00@projects-jsoe ~]$
```

Connected to projects-jsoe.ucsd.edu | SSH2 - aes128-cbc - hmac-md5 - n | 80x25

Open File Transfer Client:


2:projects-jsoe.ucsd.edu - Grozi - SSH Secure File Transfer
File Edit View Operation Window Help
Quick Connect Profiles
:\Users\JAINA\Documents\# Grozi\
Add
/srv/www/projects/gsa/Fa07
Add
Transfer Queue

Local Name	Size	Type	Modified
Images		File folder	03/14/2010 08:50:2...
photoshopped_game_ima...		File folder	02/18/2010 06:31:0...
Reports		File folder	03/11/2010 06:49:2...
Website		File folder	02/04/2010 03:19:4...
Website backup		File folder	02/01/2010 01:37:2...
Wiimotecpp		File folder	02/16/2010 09:07:1...
WiiRemoteJ v1.6.zip		File folder	01/26/2010 10:33:1...
wiiuse_v0.12_win		File folder	02/16/2010 08:38:3...
wiiusecpp_v0.12_src		File folder	02/16/2010 08:15:3...
2010-02-11 Minutes 6.docx	19,986	Microsof...	03/12/2010 02:54:2...
2010-03-11 notes.txt	627	Text Doc...	03/11/2010 07:22:0...
20100114 Meeting 2.rtf	60,072	Rich Text...	02/04/2010 12:24:0...
bluecove-2.1.0.jar	547,156	Executab...	01/27/2010 04:38:3...
eng100notes.docx	13,525	Microsof...	01/23/2010 02:27:3...
Final Report - Webmaster....	1,884	Rich Text...	03/14/2010 08:46:5...
GroZi_2010.ppt	2,404,864	Microsof...	03/13/2010 02:43:4...
groziposter7.pptx	4,789,421	Microsof...	02/01/2010 01:46:2...
header.html	4,439	Chrome ...	12/01/2009 09:55:0...
Makefile.win	891	WIN File	02/16/2010 09:08:2...
Minute 2010-03-04.docx	19,260	Microsof...	03/12/2010 03:24:1...
Minute 2010-03-11.doc	25,600	Microsof...	03/11/2010 07:23:5...
Minutes 5 - 02-04-2010.do...	17,920	Microsof...	03/12/2010 02:12:5...
Minutes 2010-01-28.doc	31,744	Microsof...	01/28/2010 07:26:1...
Minutes 2010-02-18.docx	19,732	Microsof...	02/18/2010 10:21:0...
Minutes 01212010.doc	43,008	Microsof...	02/04/2010 12:33:0...

Remote Name	Size	Type	Modified
backup		Folder	02/16/2009 10:08:3...
CSS		Folder	12/01/2009 08:14:2...
images		Folder	03/13/2010 02:02:1...
index_files		Folder	10/23/2009 01:39:5...
Other		Folder	03/11/2010 06:05:3...
client.shtml	794	Chrome ...	03/13/2010 01:51:5...
contact.htm	7,015	Chrome ...	12/01/2009 08:14:3...
documents.shtml	1,686	Chrome ...	03/13/2010 12:01:0...
footer.html	1,344	Chrome ...	12/01/2009 09:01:5...
header.html	3,910	Chrome ...	02/03/2010 09:30:4...
index.shtml	2,743	Chrome ...	03/13/2010 01:53:0...
members.shtml	1,021	Chrome ...	03/13/2010 01:51:5...
photos.shtml	2,423	Chrome ...	03/13/2010 02:02:3...
project_goals.shtml	3,243	Chrome ...	03/13/2010 01:51:5...
quarter_goals.shtml	5,148	Chrome ...	03/13/2010 01:51:1...
style.css	2,716	Cascadin...	03/13/2010 12:01:4...

Source File	Source Directory	Destination Directory	Size	Status	Speed	Time

```
Connected to projects-jsoe.ucsd.edu - /srv/www/projects/gsa/Fa07    SSH2 - aes128-cbc - hmac-md5 - nc 16 items (32.0 KB)
```

Drag files from your machine on the left to the directories on the server on the right to upload.

Files currently in use on the website

- index.shtml
- project_goals.shtml
- quarter_goals.shtml
- members.shtml
- documents.shtml
- photos.shtml
- client.shtml
- header.html
- footer.html
- contact.htm
- style.css
- /images – images used in “index.shtml” and “photos.shtml”
  - grozi.jpg
  - grozi_board.jpg

    - grozi_grocery.jpg
    - grozi_redrobot.jpg
    - grozi_wiimote.jpg
    - NFBlogo.jpg
- /Other – files listed in “documents.shtml”
    - grozi_slam.pdf
    - grozi_techbrief.pdf
    - TIESGroZiFa08.pdf
    - TIESGroZiFa09.pdf
    - TIESGroZiSp07.pdf
    - TIESGroZiSp09.pdf
    - TIESGroZiSu09.pdf
    - TIESGroZiWi07.pdf
    - TIESGroZiWi08.pdf
    - TIESGroZiWi09.pdf

Wiimote Team: Jaina Chueh, Andrea Wong

INTRODUCTION

The main purpose of the Wiimote team is to direct a blind user's hand to the desired product through the strategic use of haptic feedback. This is done by replacing the red push pin currently used by the Demo Board with the user's hand and directing the user to the desired grocery product by communicating with the CVSG module. Nintendo's Wiimote is able to perform the needed tasks and is perfect for GroZi applications. The Wiimote is able to translate its position and motion using infrared sensor and accelerometers. The Wiimote's vibrating motors as the actual haptic feedback provided to the user, and its speakers as a way of communicating to the user the successful location of the desired product. The Wiimote can also be connected to a computer or laptop via Bluetooth communication. In this way, software can then program the Wiimote to suit GroZi's specific needs.

GENERAL OVERVIEW AND USAGE

The objective of the Wiimote subteam was to connect a Wiimote to a laptop computer and develop and implement software which uses the various functions of the Wiimote to specifically fit the GroZi prototype requirements. The general procedures taken by the Wiimote team can be broken into two basic parts: establishing a Bluetooth connection and implementing software.

ESTABLISHING BLUETOOTH CONNECTION

The first step was to establish the Bluetooth connection. This task proved to be rather difficult with hours of troubleshooting. The problems encountered stem from compatibility issues between the computer and the Wiimote. Although both devices are Bluetooth capable, the Wiimote was not made for use with a normal computer but with the Wii. The obstacles we faced this quarter includes incompatibilities between Wiimote and the Bluebooth stack and Java libraries that handles the communications between a Bluetooth device and the computer in different version of Windows (Windows XP, Windows Vista, Windows 7, and 64-bit Windows). We will explain these problems in detail in the "This Quarter" section.

To establish a Bluetooth connection, a computer must have a Bluetooth device installed and become Bluetooth capable. Some computers have an internal Bluetooth device, while others need an external adapter. The choice is usually given to the buyer when purchasing a computer. One of the main objectives of the Wiimote team last quarter was to find a Bluetooth device on a Windows machine that works with the Wiimote. The team was able to compile a list of compatible and incompatible devices, which also includes working Bluetooth driver stacks to be used with the specific device. The particular combination of device and driver stack used was the Rocketfish 2.0 Bluetooth Dongle and the Widcomm Bluetooth Software v. 5.1.0.1100.
One of the biggest issues the team last quarter faced was the Microsoft Bluetooth Stack interfering with the communication between the additional driver and the Wiimote. The team resolved this issue by ripping, or removing and disabling the Microsoft stack from the system. Ripping the Microsoft Bluetooth stacks was a long process in itself. Instructions to perform this task are available from http://www.dev-toast.com/2007/01/05/uncrippling-bluetooth-in-vista-rtm/.

OBJECTIVES THIS QUARTER

RUN AND TEST AND DEBUG JAVA PROGRAM

The first thing the team needed to do was to establish a Bluetooth connection with their laptops and the Wiimote. Due to the fact that both members of the team did not have Bluetooth in their laptops, the team had to buy a Bluetooth dongle, an external Bluetooth device. After reading the previous quarter's report and availability at Best Buy, the team decided to buy a Rocketfish Bluetooth 2.1 dongle at a cost of $38.05 (including taxes). The device comes with its own installation CD. Bypassing last quarter's problem with Windows Vista OS, this quarter's two laptops used Windows XP and Windows 7. Once installed on both laptops, the device was capable of identifying other Bluetooth devices, including the Wiimote, with no problem for the next weeks.

At nearly at the end of the quarter (around week 9), the team encountered a problem with the Bluetooth device: the dongle had stopped working. It was showing itself as a hidden device USB and not as a Bluetooth device. The solution was to uninstall it and reinstall it again with the installation CD that came with the dongle. So far, there has not been another problem with the Bluetooth dongle.

Once the Bluetooth connection was established, the team worked on setting up the Java program from previous quarter's team wrote for tracking the hand position using the Wiimote's IR sensor. Directions to set up the Java program The Java program only works on the Windows XP laptop and not on the Windows 7. The program (working with WinXP) was tested by moving the Wiimote and with a light source. The Wiimote vibrates when it finds a light source such as sunlight, a kitchen lighter (fire) or a flashlight.

Another requisite for the team was to set up the Java program in the new GroZi's laptop. The reason for this was so that the GroZi team gets to have all working programs under one laptop in order to be able to present GroZi to future clients and expos.
Since the GroZi's laptop has its own Bluetooth, the Rocketfish dongle was not installed. Once the Java library and program was setup, the program did not work. It kept trying to locate the Wiimote but it was unsuccessful. Due to this, the team thought that there was a possible Bluetooth stacking incompatibility in the GroZi laptop. After ripping the current stacks, it turns out it was not a Bluetooth stacking problem. Our reasoning is because the Bluetooth device in the laptop is able to identify and find the Wiimote. With additional help, the team has educationally guessed that it may be a Java library to Windows 7 or Win bit 64 incompatibility problem or both.

Currently, the Java Program does not work in the GroZi's laptop. The team's temporary solution was to record two videos simultaneously to show that the Java Program does work. The videos show a moving Wiimote with exaggerated motions and a real time graph of its acceleration in the x, y, z space. The colored lines: Red, Blue, and Green are coordinates of the Wiimote going left/right (x-axis), up/down (y-axis) and forward/backwards (z-axis). The white line is the time axis. When the Wiimote finds light, it vibrates. One can see this when the red, blue, and/or green lines have a higher frequency. Additionally, pictures of the running program were taken.

C++ WIIMOTE LIBRARY

In order for integration of software between CVSG and Wiimote Team to be possible in the future, the

Wiimote team must provide code in C++. The reason is because the CVSG code utilizes OpenCV, which is written in C++. The Wiimote team this quarter researched and found a few C++ Wiimote libraries written and distributed by various developers on the internet, available in APPENDIX. The C++ Wiimote library the team has decided to research further in depth is "Wiiusecpp" (http://www.missioncognition.net/wiiusecpp).

Wiiusecpp is the C++ wrapper for Wiiuse, which is written in C. Wiiusecpp supports functions that can be utilize to accomplish our project requirements, such as tracking Infrared and generating the acceleration of the Wiimote. These two functions can generate data for the CVSG program to analyze and eventually provide instructions to guide the user's hand to the desired grocery product.

Setup C++ library with Eclipse Galileo for C/C++

To start development of the Wiimote C++ program, the team needs to install Eclipse Galileo for C/C++. The team this quarter has encountered many problems in the process due to unfamiliarity of Eclipse. To prevent future Wiimote team from running the same problems, a complete list of instruction to set up Eclipse for C/C++ is available in APPENDIX. Below shows a few issues that took a long period of time to research and debug.

Problems & Solutions

- "Launch failed. Binary not found." when try to run the program
    - You must "Build" the project first. The shortcut in Eclipse is Ctrl+B. You must build an object file before you can compile it, because otherwise Eclipse cannot link and load that object file, and will not have the required binary numbers to execute.
- Eclipse console shows no activity when running the program
    - Eclipse was unable to produce any output in its console when the team run any program. To resolve this, you must first generate the executable using Eclipse's BUILD function, then use Windows' Command Prompt (cmd.exe) to run the executable
    - To find the executable in Window's Command Prompt
        - From the Desktop, click on **Start**, and then type in "cmd.exe"
        - Navigate to your Eclipse workspace folder
            - To change directory, use command "cd <directory name>"
            - To list all items in the current directory, use command "dir"
        - Navigate to your project directory
        - In your project directory, go in the "Debug" directory
        - Once inside the Debug directory, type in the name of the executable "<source file name>.exe" to run

FUTURE PLANS

An immediate future plan is to keep on coding in C++ since our end goal is to be able to integrate the program with the CVSG code. The Wiimote team is still having problems with integrating the Java library with Windows 7. Future Wiimote teams will need to solve this problem. Although the Java program will not be in use, this problem may come up again in the C++ library. Next quarter's team will need to make sure that

the C++ program is able to connect with the Wiimote and Windows 7. Once the C++ or Java program work, the teams should gather information on the sensitivity of the Wiimote. This means, what is the smallest motion that one can make (i.e. moving 1cm or 1mm) that the C++/Java program will capture?
Another plan is to edit the videos taken this quarter. Currently, the videos are raw but it will be needed to be edited to be more presentable. Although the Java program does not work in the GroZi's laptop, having videos will be a small proof that the Java program does function in Windows XP. To reiterate, this is a temporary solution until the C++ or Java program starts working.
This quarter, the team was not able to work on the reflective tape ring. Yet, future teams should recreate the reflective tape ring that the previous team did and buy and LED array. There is a list of requirements for the reflective tape ring that the teams should follow.

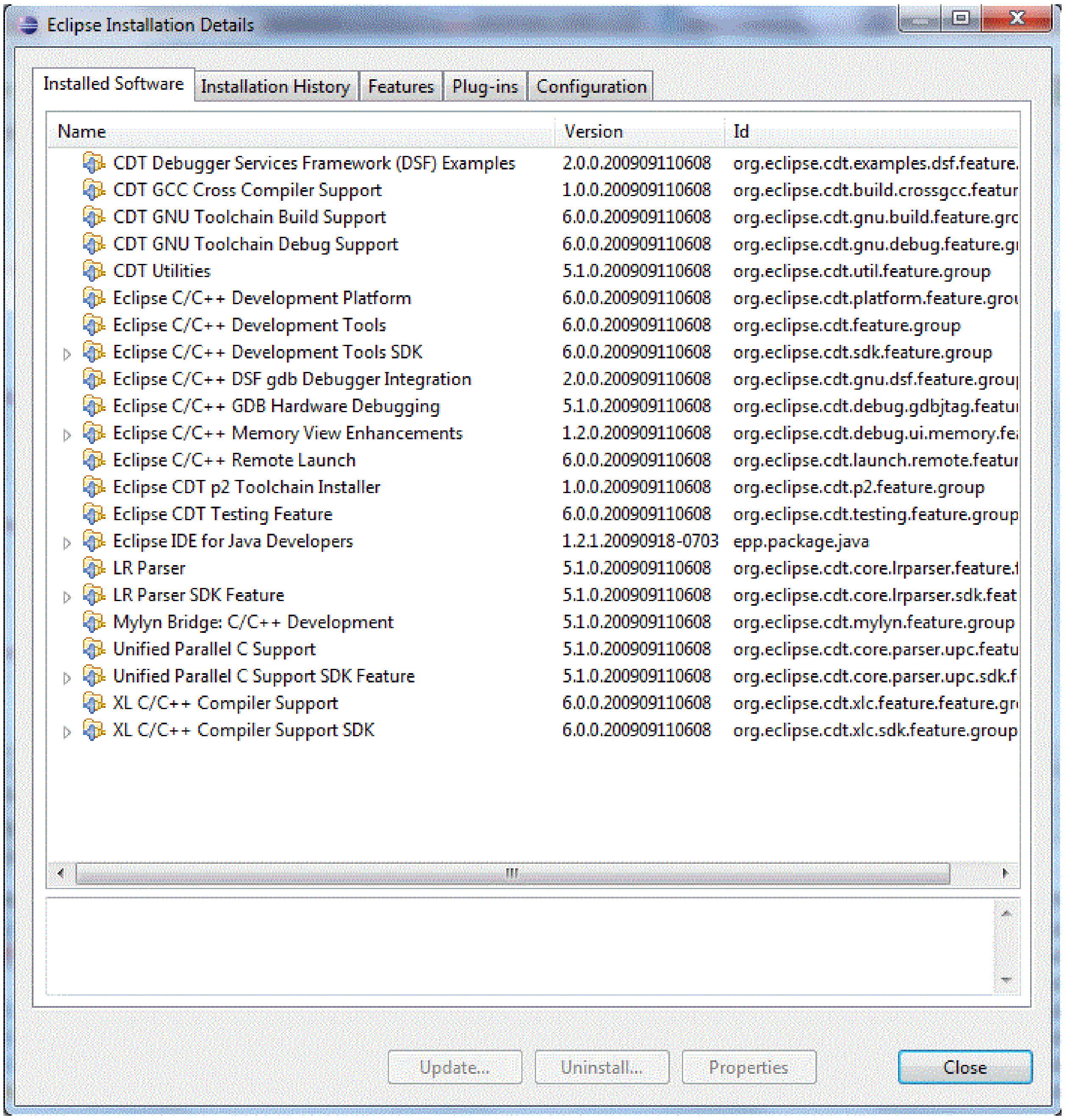
LIST OF REQUIREMENTS

1. The user must wear a ring that reflects IR light, within ±100nm of 590nm (?).
2. The user must use the hand with the ring to select products
3. The user will wear an IR source directed towards the hand with the ring, such that the reflective ring is not in a shadow of the IR source.
4. The Wiimote shall track the hand position within TBD centimeters of the actual hand position.
5. The ring shall be made of velcro type material.

APPENDIX

To run both Java and C++ files on Eclipse
Install Eclipse IDE for Java Developers + Eclipse CDT

1. Download and install Eclipse IDE for Java Developers (http://www.eclipse.org/downloads/)
2. Download Eclipse C/C++ Development Tooling (CDT) (http://www.eclipse.org/cdt/)
3. Extract CDT zip file to where your Eclipse.exe is located
4. After unzipping the file into the Eclipse directory, your Eclipse plugin directory should have **org.eclipse.cdt.*** folders
5. To verify that you have successfully installed CDT, launch Eclipse
6. You should see something similar to the following screenshot in Help > About Eclipse > Installation Details



Name	Version	Id
CDT Debugger Services Framework (DSF) Examples	2.0.0.200909110608	org.eclipse.cdt.examples.dsf.feature.
CDT GCC Cross Compiler Support	1.0.0.200909110608	org.eclipse.cdt.build.crossgcc.featur
CDT GNU Toolchain Build Support	6.0.0.200909110608	org.eclipse.cdt.gnu.build.feature.grc
CDT GNU Toolchain Debug Support	6.0.0.200909110608	org.eclipse.cdt.gnu.debug.feature.gı
CDT Utilities	5.1.0.200909110608	org.eclipse.cdt.util.feature.group
Eclipse C/C++ Development Platform	6.0.0.200909110608	org.eclipse.cdt.platform.feature.grou
Eclipse C/C++ Development Tools	6.0.0.200909110608	org.eclipse.cdt.feature.group
Eclipse C/C++ Development Tools SDK	6.0.0.200909110608	org.eclipse.cdt.sdk.feature.group
Eclipse C/C++ DSF gdb Debugger Integration	2.0.0.200909110608	org.eclipse.cdt.gnu.dsf.feature.grou
Eclipse C/C++ GDB Hardware Debugging	5.1.0.200909110608	org.eclipse.cdt.debug.gdbjtag.featu
Eclipse C/C++ Memory View Enhancements	1.2.0.200909110608	org.eclipse.cdt.debug.ui.memory.fe
Eclipse C/C++ Remote Launch	6.0.0.200909110608	org.eclipse.cdt.launch.remote.featu
Eclipse CDT p2 Toolchain Installer	1.0.0.200909110608	org.eclipse.cdt.p2.feature.group
Eclipse CDT Testing Feature	6.0.0.200909110608	org.eclipse.cdt.testing.feature.group
Eclipse IDE for Java Developers	1.2.1.20090918-0703	epp.package.java
LR Parser	5.1.0.200909110608	org.eclipse.cdt.core.lrparser.feature.
LR Parser SDK Feature	5.1.0.200909110608	org.eclipse.cdt.core.lrparser.sdk.feat
Mylyn Bridge: C/C++ Development	5.1.0.200909110608	org.eclipse.cdt.mylyn.feature.group
Unified Parallel C Support	5.1.0.200909110608	org.eclipse.cdt.core.parser.upc.featu
Unified Parallel C Support SDK Feature	5.1.0.200909110608	org.eclipse.cdt.core.parser.upc.sdk.f
XL C/C++ Compiler Support	6.0.0.200909110608	org.eclipse.cdt.xlc.feature.feature.gr
XL C/C++ Compiler Support SDK	6.0.0.200909110608	org.eclipse.cdt.xlc.sdk.feature.group

Tutorial instructions taken from http://www.cs.umanitoba.ca/~eclipse/7-EclipseCDT.pdf

Directions to set up the Java program

Currently, the Java program only works on Windows XP

I. You will need to download:

1. Eclipse Galileo for Java SDK
   a. Can be found http://www.eclipse.org/downloads/
2. Bluecove-2.1.0 – Executable Jar
   a. Can be found at http://sourceforge.net/projects/bluecove/files/

    b. Main website at http://bluecove.org/
3. WiiRemoteJ – Executable Jar
    a. Can be found at http://wiibrew.org/wiki/Wiimote/Library#WiiRemoteJ.2C_a_Java_library_for_the_Wii_Remote
4. WRLImpl – Java File
    a. Can be found at http://grozi.calit2.net/files/TIESGroZiSp09.pdf
    b. It is an edited version of Michael Diamond's code which is part of the installation package downloaded from WiiRemoteJ

II. Once Eclipse is installed, you will need to:
1. Create a workspace
2. Create a new project
3. Move WRLImpl Java file to the source folder of your project (src).
4. Add in a JRE System Library [Java SE 1.6]
5. In Referenced Libraries add in the bluecove-2.1.0.jar and WiiRemote.jar
6. Build path on all 3 files (WRLImpl, Bluecove and WiiRemote)

III. To run the program:
1. Turn on the Wiimote
2. Sync the Wiimote by pressing the red button (located in the batteries area)
3. Make sure the Bluetooth dongle is able to find and identify the Wiimote
4. Run the program

IV. A black window with red, green, blue and white lines should pop up. This is the real time graph.

Instructions to install Eclipse for C/C++, MinGW, and Wiiusecpp

1. Download and install Eclipse for IDE C/C++ Developers
   Download and install MinGW as C/C++ compiler (the team used v. 5.1.6)
    a. Add to path
        1. From the Desktop, right-click **My Computer** and click **Properties**.
        2. In the System Properties window, click on the **Advanced** tab.
        3. In the Advanced section, click the **Environment Variables** button.
2. Download Wiiuse (the team used v. 0.12)
3. Create a workspace (default)
4. Create a new project
5. To link the library:
    a. Right-click on project name->Properties->C/C++ Build->Settings->MinGW C++ Linker->Libraries
    b. Under Libraries, add "wiiuse"
    c. Under Library search path, add your project via Add...->Workspace->project name
6. Create a source folder and drag & drop these files in:
    - wiiuse.h
    - wiiusecpp.h
    - wiiusecpp.cpp
    - Drag & drop "wiiuse.lib" into the project
7. Start development of code!

The list of C++ Wiimote libraries the team has found:

1. Wiiyourself http://wiiyourself.gl.tter.org/
2. GlovePIE http://glovepie.org/glovepie_download.php

Other References (not C++)

1. http://www.brianpeek.com/blog/pages/net-based-wiimote-applications.aspx
2. http://www.wiimoteproject.com/
3. http://johnnylee.net/projects/wii/
4. http://blogs.msdn.com/coding4fun/archive/2007/03/14/1879033.aspx

Demo Board Design Team: Jeffrey Su & Ed Liu

Introduction

The Demo Board design team's goals are to create demo boards that allow performance evaluation of various algorithms like those used in the Remote Sighted Guide. The demo boards need to mimic grocery store shelves and still be easily transportable. Some of the key points these demos will help us solve is variability problems likes those induced by shading/poor lighting conditions, occlusion of the image, and perspective issues. This quarter's demo boards need to help to replace some of the material that was causing trouble for the CVSG team. The electric tape was causing trouble for the camera because it was reflecting light. Also, we did an experiment where we had people try to navigate the board by having the instructions from a program tell us where to go. We then realized that we needed more pins because the game was too easy for people to guess at the answer. Finally we are prototyping a new demo board that is easier to use. We found that blindfolded subjects found placing the pushpin in the pin-hole difficult and too slow.

The Game Board

The role of the demo game board was to allow the various teams, especially the Computer Vision Sighted Guide (CVSG) and Protocol teams, to test their algorithms within a controlled environment. If the team could not get their Computer Vision and Protocol programs and hardware working with the board, then getting it to work in an actual grocery store would be even less plausible. Thus, our goal was to design a cheap and easy do-it-yourself board to work with certain isolated variables in the grocery store environment.

The board itself is arranged in a 23x25 square grid with wooden dowels splitting certain rows to mimic the shelves of a grocery store. Push pins were used as "game pieces" to denote the target, the corners of the board, and the hand piece. Other game pieces were placed throughout the board as "distracters." With this demo board, one can play a game in which Player 1 chooses a target square on the grid, and, through a series of predetermined commands, guides Player 2 (the "blind" player) to that target square.

Board Specifications

The following are the items and materials used to build the board. A camera tripod, though not specified, was also used to hold up the camera. (See Table 2.1 below)

Board materials	2 foam boards multicolored push-pins 3/4 in. felt cloth super glue
Board (edge to edge including felt border)	51 cm x 51 cm
Grid Squares (ea. w/ a hole in the center for push-pins)	2 cm x 2 cm
Grid (inside felt border)	50 cm x 46 cm
Width of border	3/4 in
# Squares between shelves	5 (for the middle three shelves) 4 (for the top- and bottom-most shelves)
# of shelves (wooden dowels)	4
Distance from edge of black tape to edge of grid	Top: 5 mm Bottom: 9 mm
Distance from table to bottom of camera lens (mounted on tripod)	50 cm
Distance from edge of board to center of tripod	18.5 cm
colors of pieces used (multicolored push-pins)	red (target. Has spherical head on top rather than cylindrical to be located by touch), blue (hand location), green (distractors), yellow (corner demarcations)

Table 2.1: *Board Specifications*

What's next?

Future Considerations and Recommendations

Prototyping a New Demo Board

We are considering on building a new demo board. The board size will be approximately the same as the one as the one as we have now. There are plastic tubes(anchors) and can fit in these holes and can represent the pins that we use now. Issues that we are having are coming from the weight and portability of the board. One possible solution is being able to fold up the board, as well as being able to keep the pieces on.

Protocol Team: Kevin Tran & Saitejaswi Kondapalli

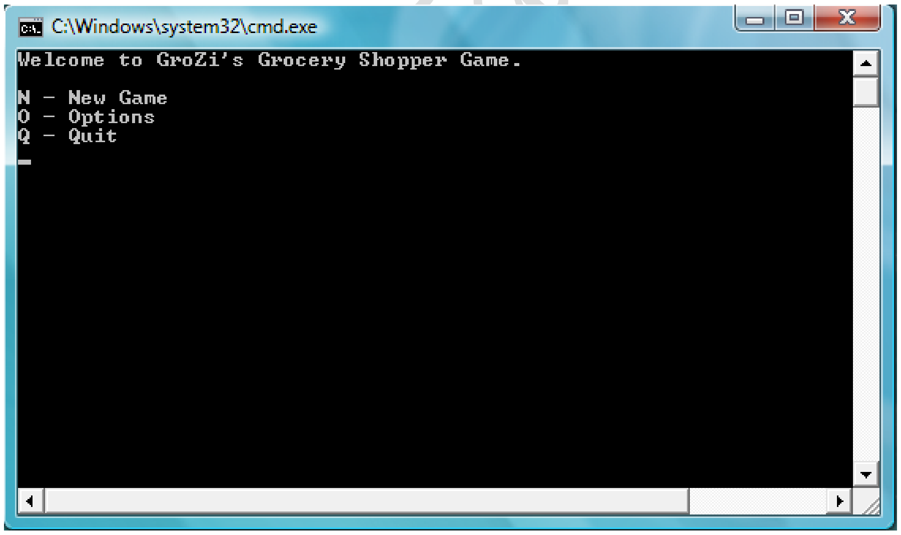
Introduction

What is the GroZi Grocery Shopper Game Program?

The GroZi Grocery Shopper Game application is new software that is under development to work in conjunction with the images the Computer Vision Sighted Guide (CVSG) team is able to process. The purpose of this program is to be able to take the image data (i.e. hand token location, item token location) provided in a file from CVSG and process it in such a way that it will provide directions from the current hand location to the target item location. Ideally, once the implementation of this software is complete, it should help analyze how to efficiently guide a blind person to a grocery product.

General Overview of Usage

The GroZi Grocery Shopper Game program is a Win32 Application written in C programming language. Currently, the code (located in files Game_Code.c and Game_Code.h) can be run using Visual Studio 2008 or in a Linux environment by compiling using the gcc command.
Upon starting the program, the user will be prompted with the main menu (see figure 3.1a) in which they can select to play a "new game", view "options", or "quit".



```
Welcome to GroZi's Grocery Shopper Game.

N - New Game
O - Options
Q - Quit
_
```

Figure 3.1a : *GroZi Grocery Shopper Game main menu options*

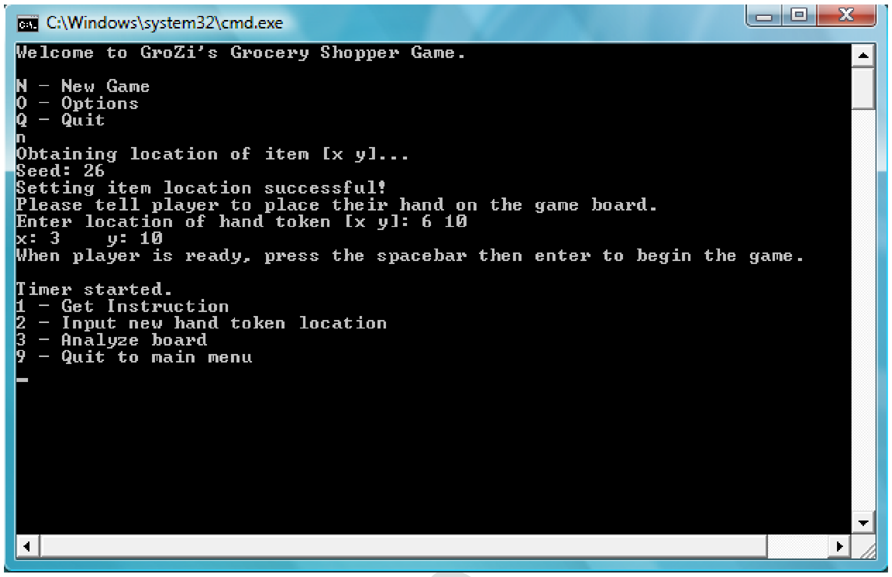
If the user chooses to start a new game, (by entering the letter N), the software will automatically generate a random (x, y) coordinate from a supplied file which will be used for the target location. Next, the program will prompt the user to enter their initial hand location and store this into an array. The program then lets the user know to press spacebar when they would like the timer to begin. As soon as spacebar has been pressed, the game menu is shown (see figure 3.1b) with the following options:

1 – Get Instruction

2 – Input new hand token location

3 – Analyze board

9 – Quit to main menu



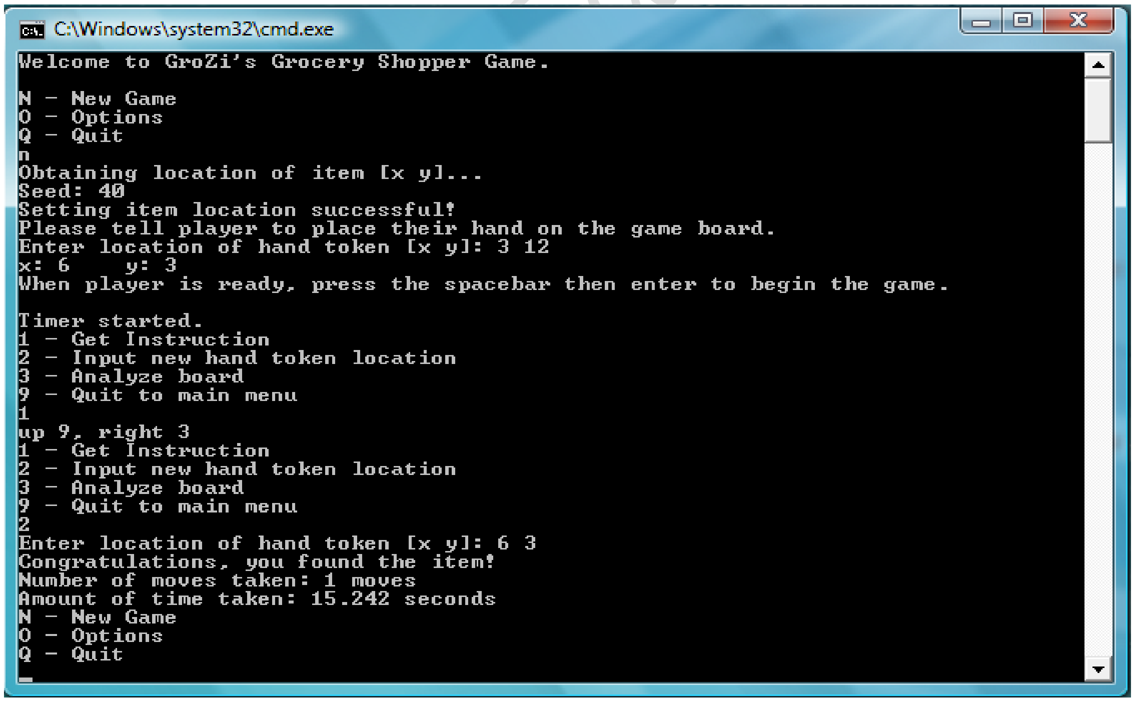
```
C:\Windows\system32\cmd.exe
Welcome to GroZi's Grocery Shopper Game.

N - New Game
O - Options
Q - Quit
n
Obtaining location of item [x y]...
Seed: 26
Setting item location successful!
Please tell player to place their hand on the game board.
Enter location of hand token [x y]: 6 10
x: 3     y: 10
When player is ready, press the spacebar then enter to begin the game.

Timer started.
1 - Get Instruction
2 - Input new hand token location
3 - Analyze board
9 - Quit to main menu
_
```

Figure 3.1b: *In play menu options*

Get Instruction provides the user with the next move to get to the target location by providing an up or down and right or left instruction. The user then is able to input a new hand token location at which point the program will either congratulate you because you found the item, or wait for you to ask for a new instruction. This menu keeps looping until the item is found. At this point the final time and number of moves it took to find the item will be displayed (Figure 3.1c).



```
C:\Windows\system32\cmd.exe
Welcome to GroZi's Grocery Shopper Game.

N - New Game
O - Options
Q - Quit
n
Obtaining location of item [x y]...
Seed: 40
Setting item location successful!
Please tell player to place their hand on the game board.
Enter location of hand token [x y]: 3 12
x: 6     y: 3
When player is ready, press the spacebar then enter to begin the game.

Timer started.
1 - Get Instruction
2 - Input new hand token location
3 - Analyze board
9 - Quit to main menu
1
up 9, right 3
1 - Get Instruction
2 - Input new hand token location
3 - Analyze board
9 - Quit to main menu
2
Enter location of hand token [x y]: 6 3
Congratulations, you found the item!
Number of moves taken: 1 moves
Amount of time taken: 15.242 seconds
N - New Game
O - Options
Q - Quit
_
```

Figure 3.1c: *Output from a full game, beginning to end with user finding target item*

The Code

Data structures used (declared in Game_Code.h) :

```
typedef struct {
int x,y;
int sign;
} GameSquareType;

GameSquareType hand_array[POSS_MAX];   // array of hand token positions
GameSquareType item_array[POSS_MAX];   // array of target item positions
GameSquareType other_item_array[POSS_MAX];
```

Main code (Game_Code.c):

```
/* Looping main menu */
            switch(main_option)
            {
                  case 'N':
                  case 'n':
                        /* new game */
                              break;
                  case 'O':
                  case 'o':
                        /* options */
                        break;
                  case 'Q':
                  case 'q':
                        printf("Thank you for playing.\n");
                        exit(0);
                        break;
            }

/* In game looping menu */
            while(game_win == 0 || game_running == 1)
            {
                  game_menu();
                  scanf("%d", &game_option);
                  getchar();

                  switch(game_option)
                  {
                        case 1:
                              my_get_move(hand_array, item_array, 0);
                              break;
                        case 2:
                              set_hand_location(hand_array, 0);
                              total_moves++;
                              break;
                        case 3:
                              // analyze_virtual_board();
                              break;
                        case 9:
                              game_win = 0;
                              exit (0);
                              break;
            }
```

```
/* Function to automatically set item location from randomly generated x y
coordinates file */
static int set_item_location(GameSquareType itemarray[], int index)
{
      FILE * fp;
      char * mode = 'r';                      // read
      int temp_x, temp_y;
      int rand_num, counter = 0;

      printf("Obtaining location of item [x y]...\n");

// Change file path according to where rand100xy.txt is located on local machine
      fp = fopen("..\\..\\..\\..\\..\\Desktop\\Grozi Game Code\\Grozi
ver.2\\rand100xy.txt", "r");

      if (fp == NULL)
      {
            fprintf(stderr, "Can't open input file!!!\n");
            exit(1);
      }

      srand(time(NULL));
      rand_num = rand() % 100;
      printf("Seed: %d\n", rand_num);

      while ( !feof(fp) )
            if( fscanf(fp, "%d %d", &temp_x, &temp_y) != NULL && rand_num ==
                counter++ )
                  break;

      itemarray[index].x = temp_x;
      itemarray[index].y = temp_y;

      num_item_tokens++;

      return 1;
      }


/* Function to set the initial hand location provided by user */
static int set_hand_location(GameSquareType handarray[], int index)
{
      int temp_x, temp_y;

      printf("Enter location of hand token [x y]: ");

      /* Setting location of item */
      scanf("%d %d", &temp_x, &temp_y);
      getchar();

      /* Do error checking here if necessary */
      handarray[index].x = temp_x;
      handarray[index].y = temp_y;
      return 1;
}
```

```c
/* Function to calculate actual directions based on user input of hand
   location */
static void my_get_move(GameSquareType handarray[], GameSquareType itemarray[], int index)
{
        diff_x = item_array[index].x - hand_array[index].x; // x-cord diff
        diff_y = item_array[index].y - hand_array[index].y; // y-cord diff

        // Moving up __ left __
        if (diff_x < 0 && diff_y < 0)
        {
                printf("%s %d, %s %d\n", up, (diff_y * -1), left, (diff_x * -1));
        }

        // Moving up __ right __
        if (diff_x > 0 && diff_y < 0)
        {
                printf("%s %d, %s %d\n", up, (diff_y * -1), right, diff_x);
        }

        // Moving down __ right __
        if (diff_x > 0 && diff_y > 0)
        {
                printf("%s %d, %s %d\n", down, diff_y , right, diff_x);
        }

        // Moving down __ left __
        if (diff_x < 0 && diff_y > 0)
        {
                printf("%s %d, %s %d\n", down, diff_y, left, (diff_x * -1));
        }

        // Moving just down __
        if (diff_x == 0 && diff_y > 0)
        {
                printf("%s %d\n", down, diff_y);
        }

        // Moving just up __
        if (diff_x == 0 && diff_y < 0)
        {
                printf("%s %d\n", up, (diff_y * -1));
        }

        // Moving just left __
        if (diff_x > 0 && diff_y == 0)
        {
                printf("%s %d\n", left, diff_x);
        }

        // Moving just right __
        if (diff_x < 0 && diff_y == 0)
        {
                printf("%s %d\n", right, (diff_x * -1));
        }

        // Item found
        if (diff_x == 0 && diff_y == 0)
        {
                printf("You got it!\n");
        }
}
```

Experimental Results

Generating the Protocol

At first, we used a Scrabble board to create a list of protocols that we thought were necessary to run the game. It was a 15 x 15 board and the group just did trials of choosing a spot for the item, and a place for the hand token and manually told the player to move from position to position. This generated our base list of words that we needed. This allowed for fast movement between shelves instead of saying the exact number of moves needed. We also decided that a reasonable amount of distance to tell the player the exact coordinates of the item had to be less than five in both vertical and horizontal direction. Other protocols were added either to enhance the interaction with the game or for debugging purposes. The following table (Table 3.3a) lists the protocols we decided to include in the game.

Number of steps	**Direction**	**Shelves**	**Responses**	**Other**	**Debugging**
One	Left	Upper	Yes	Same	X
Two	Right	Middle	No	Row	Y
Three	Up	Lower	Correct	Column	Item at location (x, y)
Four	Down	Shelf	Wrong	Item	
Five		Shelves	Item found	A lot	
				Start	

Table 3.3a*: List of protocols used in the game*

The output of the protocol was designed to respond differently depending on the distance of the user's hand pin. When given the hand location, we process it as such: If the hand is on a different shelf than the item, we will use text-to-speech to say—"Wrong shelf, shelf "shelf_number" row "row-number," where shelf_number and row_number are variables within the program. It then proceeds to TTS horizontal information as such—If the item is directly below or above the hand location, we prompt "same column." Otherwise, if it is between one and five squares different from the hand location, either left or right, we will tell the user the exact distance left or the exact distance to the right. If it is any greater than five, we tell the user "a lot" to the left or the right. And of course, when the hand is directly on the item, we say "Congratulations, you've found the item!" and the program will continue to begin another instance of the game.

Trials testing the Protocol

Having developed a working protocol, our goal this quarter was to generate experimental data on how the commands were performing when using a configuration file that represented the demoboard accurately. Previously we had conducted tests on checkerboard style arrangements, and thusly we felt it necessary to test the protocol on the actual prototype. As before, these are the protocol commands issued by the program:

Number of steps	**Direction**	**Shelves**	**Responses**	**Other**	**Debugging**
One	Left	Upper	Yes	Same	X
Two	Right	Middle	No	Row	Y
Three	Up	Lower	Correct	Column	Item at location (x, y)
Four	Down	Shelf	Wrong	Item	
Five		Shelves	Item found	A lot	
				Start	

***Table 3.3a**: List of protocols used in the game*

The following is a sample output of the game using a 23x25 board. The origin is at (0, 0) which is the bottom left corner. We placed the item at (6, 4) and the player arbitrarily chose a spot which happens to be (10, 12). Player – P Director – D
P: "this one?" (at position 10, 12)
D: "no, down two shelves"
P: (chooses position 10, 3) "this one?"
D: "correct shelf, left four, up one"
P: "this one?" (at position 6, 4)
D: "Item found"

We conducted seven timed trials using the demoboard. Trials were conducted by Kevin and time was taken via stopwatch by Cankut. Our test environment was generated by a random target location on the demoboard and a statically set hand starting position. We did not allow communication between the conductor of the test and the participant as to ensure the times were strictly based on the performance of the protocol. The demoboard setup was loaded from a file named "config_fall09.txt."

Trial Times (Randomly generated item location and hand position (0,0))—see table 3.3b below
Size of board: 23x25 (demoboard)

Player: John Miller	Director: Kevin Tran	Timer: Cankut Guven

Time in seconds	2:06	
Player: John Miller	Director: Kevin Tran	Timer: Cankut Guven
Time in seconds	0:52 (ERROR)	
Player: Jeff Su	Director: Kevin Tran	Timer: Cankut Guven
Time in seconds	0:00 (ERROR)	
Player: Jeff Su	Director: Kevin Tran	Timer: Cankut Guven
Time in seconds	1:48	
Player: Andrea Wong	Director: Kevin Tran	Timer: Cankut Guven
Time in seconds	1:06	
Player: Andrea Wong	Director: Kevin Tran	Timer: Cankut Guven
Time in seconds	1:00	
Player: Jaina Chueh	Director: Kevin Tran	Timer: Cankut Guven
Time in seconds	1:05	

Table 3.3b: *Trial times*

The (ERROR) markings above indicated a latent bug present in our configuration file. This error was categorized as an "off-by-one" where the configuration file listed incorrect shelf locations. However the error only manifested if the target location was on the board between the first and second shelves, otherwise everything else was valid. That is to say, the trials without error can be considered acceptable data.

Program Consolidation

Merging the Protocol and CVSG programs

After demonstrating that the protocol program worked well with the demoboard prototype, we felt it beneficial to merge the protocol and CVSG programs into one executable. For this quarter, the integration was done at the most basic level. The programs themselves remained entirely separated, that is to say there exists no communication between the two. However, being that they are consolidated into one module, we needed to provide a way to run each program separately. We decided to achieve this goal by implementing a mandatory command line argument, either 1 or 2, which would act as a switch between the two programs. Here is an example:

```
%: grozi.exe 1 /* this would launch the CVSG program */
%: grozi.exe 2 /* this would launch the protocol program */
```

Having these two programs integrated will help to facilitate the future goal of having the CVSG communicate with the protocol through the sharing of program memory and storage.

How to run the Protocol

A Quick Tutorial

The protocol program is intended to have few prompts so that the text-to-speech system is not inundated with words to say. However, this may increase the difficulty of operation due to users who are unfamiliar with the program not knowing how to proceed. The beginning of the program contains enough prompted information for one to continue unhindered. But as soon as the program speaks "Press space and enter to continue" an empty line is presented to the user. Whenever the program reaches this state, the user has three options:

1.) Enter in the number 1 and <enter> to check the current hand location against the target location. This will generate the protocol output that will direct the user to the target location.

2.) Enter in the number 2 and <enter> to enter in a new hand location. This will allow the program to begin parsing two numbers, x and y, to "move" the hand to a different location on the board. The input should be "x y" where there exists a space between the first coordinate and the second.

3.) Enter in the number 9 and <enter> to prematurely exit the program.

Requirements

Items that must be met

-The protocol shall not speak extra information such as long menus, a counting timer, etc.

- The software shall support Windows 7 x64.

-The software shall support Windows XP Professional x86.

What's next?

Future Considerations and Recommendation

Having confirmed the correctness of the protocol when using the demoboard specifications, and integrating the protocol and CVSG code into one, we hope in the future to allow communication between the two programs. That is, we desire feed the information gained from image analysis done by the CVSG directly into the protocol so that the protocol can respond to the user based on the real life movements performed by the user. This is in comparison to a trial director who is manually inputting new user locations as they arise.

Once this can be done, we may begin using this consolidated module to run more meaningful tests on the demoboard. In this way, we can also indirectly test CVSG code for efficiency and speed.

Computer Vision Sighted Guide (CVSG) Team: Cankut Guven, Kevin Tran, Saitejaswi Kondapalli

Introduction

The CVSG(Computer Vision Sighted Guide) team's task is to create a program to recognize various colors on a board in a static situation. An idealized situation will be presented to the program in which computer algorithms will be used to evaluate a still image of the idealized environment. The blind user's hand will not be present physically, but instead represented by a red pin. The idealized environment will include a “desired” token (blue pin), on a demo board, and a series of "noise" tokens (green pins). The software will determine the location of the green pin and report it to the Game Square software.

This quarter, the CVSG team was able to accomplish the following tasks: identifying the location of the pins on the board; cycling through four game images. The program was able to successfully identify 159 of 160 pins (a success rate of 99.38%). In an ideal world, this would be 100%, but realistically, a success rate of 99% is desired such that the program does not lead the user to an incorrect item (note that 1% error rate is still pretty high considering that 1 out of 100 times the user of the program might end up making cream of mushroom soup when chicken noodle soup was desired).

Code, documents, and images can be found at http://code.google.com/p/grozi-cvsg

Requirements for the CVSG Software

- Recognize pin locations with 99% accuracy.
- Image of the board must be taken as an overhead view
- The angle between the camera face and the Demo Board shall not be more than ±5 degrees from parallel.
- The center of the image must be within the white (inner) portion of the board.
- The image must be at least 800x600 pixels.
- The input to the protocol software must be within the dimensions of the demo board.
- The images of the Demo Board shall not be taken with a flash
- The software will support Windows 7 x64
- The software will support Windows XP Professional x86
- The software will make use of OpenCV

- The images will be taken in normal indoor lighting conditions
- The gameboard imagery shall be photoshopped such that pixels outside of the black border of the gameboard are a uniform white color
- Gameboard imagery pixels inside the black border region and in the interior of the black border region (the white part of the board where the shelf dividers and pins are) shall not be photo-shopped or modified.
- The protocol software shall work with an off-the-shelf screen reader software program for the blind such as JAWS 10.0 or higher from Freedom Scientific.
- The protocol software shall generate text output that a screen reader can turn in to the guiding instructions. No additional screen output will be present in this mode.

General Overview and Usage

The software can be run in one of two modes: protocol or CVSG. The program must be provided with an argument of 1 or 2 in order to initialize.

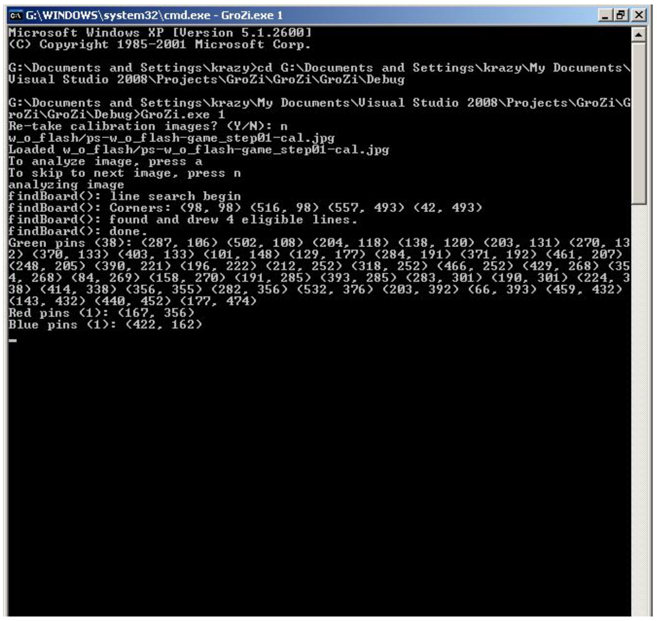
Upon executing with the argument of 1, the software will proceed with the CVSG software. It will prompt the user whether or not calibration images should be retaken. This was implemented in a prior quarter when the webcam was in use, thus is a feature that is currently disregarded and thus a reply of [N]o should be given. The program will then prompt the user to make a selection of whether the software should analyze the image, and thus provide the pixel locations of the pins and the corners of the board, or to move on to the next image. In order to detect the board and the pin locations, some initial assumptions were made. Detection is successful under the following conditions:

- Lighting such that glare from the tape is not too bright
- The middle of the image/photograph is on the game board (i.e. if the width x length of the board is 500x500, the point 250x250 is on the game board)
- The black border is visible in its entirety (no obstructions, such as arms, can be in the way)
- The image is photoshopped to remove any noise outside of the board (such as carpet)
- The images used are calibrated with the camera

With these restrictions in play, the CVSG software was able to detect the location of all the pins and the corners as shown in Figure 1.1a

Figure 1.1a: *On the left, the console output of the program is shown. On the right, the image of the board with the green, red, and blue pins circled with their respective colors is shown, in addition to the detection of the edges of the*

borde

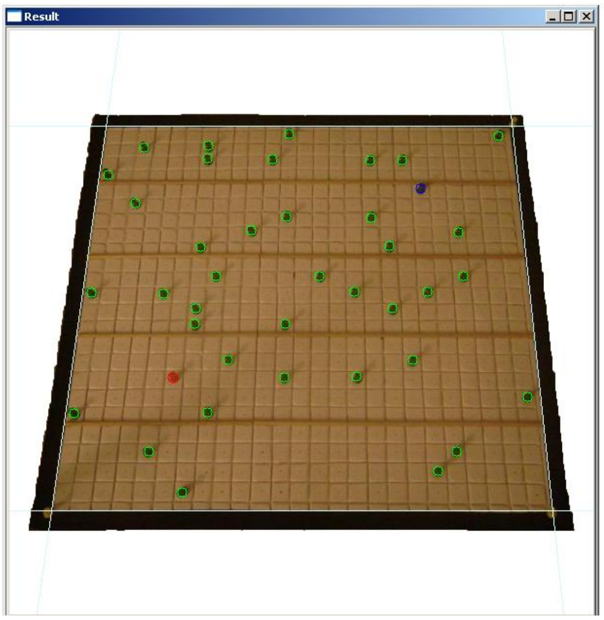


G:\WINDOWS\system32\cmd.exe - GroZi.exe 1

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

G:\Documents and Settings\krazy>cd G:\Documents and Settings\krazy\My Documents\
Visual Studio 2008\Projects\GroZi\GroZi\GroZi\Debug

G:\Documents and Settings\krazy\My Documents\Visual Studio 2008\Projects\GroZi\G
roZi\GroZi\Debug>GroZi.exe 1
Re-take calibration images? (Y/N): n
w_o_flash/ps-w_o_flash-game_step01-cal.jpg
Loaded w_o_flash/ps-w_o_flash-game_step01-cal.jpg
To analyze image, press a
To skip to next image, press n
analyzing image
findBoard(): line search begin
findBoard(): Corners: (98, 98) (516, 98) (557, 493) (42, 493)
findBoard(): found and drew 4 eligible lines.
findBoard(): done.
Green pins (38): (287, 106) (502, 108) (204, 118) (138, 120) (203, 131) (270, 13
2) (370, 133) (403, 133) (101, 148) (129, 177) (284, 191) (371, 192) (461, 207)
(248, 205) (390, 221) (196, 222) (212, 252) (318, 252) (466, 252) (429, 268) (35
4, 268) (84, 269) (158, 270) (191, 285) (393, 285) (283, 301) (190, 301) (224, 3
38) (414, 338) (356, 355) (282, 356) (532, 376) (203, 392) (66, 393) (459, 432)
(143, 432) (440, 452) (177, 474)
Red pins (1): (167, 356)
Blue pins (1): (422, 162)
_
```



Camera Calibration

In order to calibrate the camera, one can follow these steps:

1. Print out the chessboard (chessboard.pdf) and tape it to a flat surface (e.g. cardboard, wooden panel) if not already done so.
2. Place Game Board on a flat surface and set up a digital camera on a tripod such that it is aimed towards the Game Board.
3. Take pictures of the Game Board.
4. In the same setting (without altering the height or location of the tripod), take several pictures of the chessboard (in experiments that were run in Winter 2010, six images were taken; this number does not have any scientific significance).
5. Run Board::calibrate with the proper settings (alter std::stringstream imagename such that it points to the correct files) and use undistortImage(image) on a non-calibrated board image to correct calibration errors. Then, have the program save the images.
6. Images should now be calibrated.

Several tests on calibrated and non-calibrated images were run and a higher success rate was discovered in running the algorithm on calibrated images. One of the algorithms in use is the Hough Transform, which searches for straight lines. In non-calibrated images, what should be straight lines become distorted due to the lens of the camera, thus masking a straight line as a curve.

Next Steps

While detection of the board and the pin locations is a positive step forward, there is a lot that remains to be accomplished. Here are the goals that would be ideal in helping the project move forward:

- Increase detection rate from 99.3% to 99.9%
- Attempt board detection with flash photography
- Attempt board detection without photoshopping the image
- Integrate CVSG software with Game Square software to allow Game Square to identify the row and column of a pin

APPENDIX

Language: C++

IDE: Microsoft Visual Studio 2008

Operating System: Windows XP x86

Digital Camera: Sony Cybershot DSC-S650

GameSquare Team: Saitejaswi Kondapalli, Kevin Tran, Cankut Guven

What is the GameSquare Team?

The GroZi Grocery Shopper Game application is new software that is under development that incorporates the work three teams: CVSG, GameSquare and Protocol. The purpose of this program is to be able to take the image data (i.e. hand token location, item token location) provided in a file from CVSG and convert the pixel coordinates of each pin to respective coordinates on the game board in order for the Protocol team to do its calculations. This step is essential for the CVSG and the Protocol software to communicate

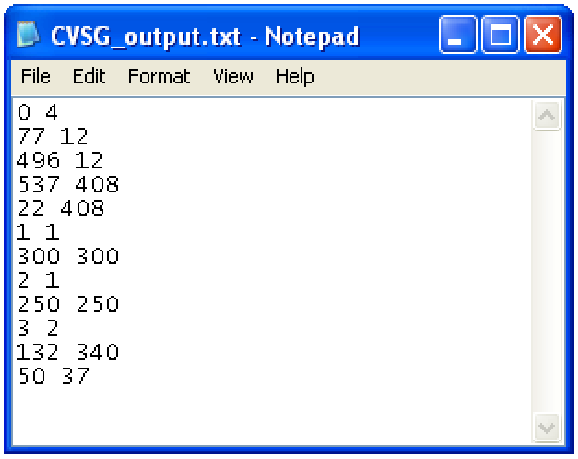
The input file into the GameSquare program will be the output file of the CVSG program. The GameSquare code will process its input and produce an output file that will be the input file to the Protocol program. These files will be plain text files and will follow strict formatting guidelines.

CVSG_output.txt

The file that is the output of CVSG will be called *CVSG_output.txt,* which will be a plain text file that is located in the same directory as the rest of the files. In this file, the first line in each block will have two integers separated by white space, the first is a integer represents a color and the second is a integer represents the (n) number of pins that are of the color. The next n lines will each contain two integers separated by white space. The first integer is the x-coordinate pixel location of the pin and the second integer is the y-coordinate pixel location of the pin. There will be at most 4 such blocks of data, each blocks gives information about the location of a set of pins of each color.

The numbering for each color is as follows:

0	yellow (corner)
1	blue (item)
2	red (hand)
3	green (distractor)



```
0 4
77 12
496 12
537 408
22 408
1 1
300 300
2 1
250 250
3 2
132 340
50 37
```

Figure 1.a: Sample CVSG_output.txt

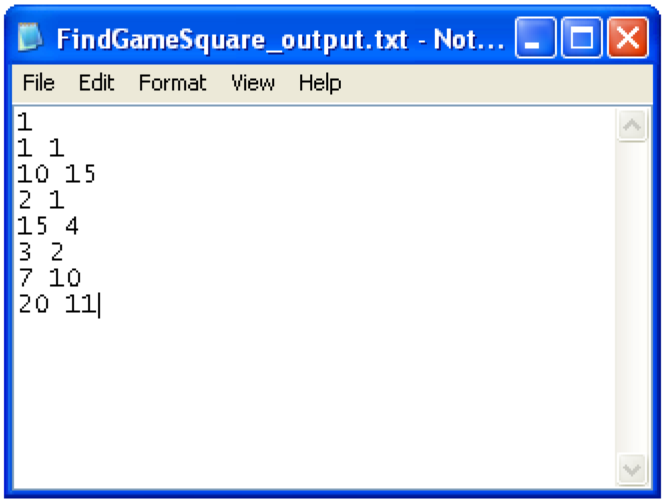
FindGameSquare Code

The GameSquare code must do the following:

1. Parse through the CVSG_output.txt to get pixel locations for all pins
2. Determine the validity of the board, the follow rules must be observed
    a. Pictures shall contain no more than 4 yellow pins
    b. Pictures shall contain exactly one red pin
    c. Pictures shall contain exactly one blue pin
3. If the gameboard is valid, the next step is to convert each pixel location into a game board location
4. The validity of the board and the gameboard locations must be written to an output file called FindGameSquare_output.txt

FindGameSquare_output.txt

The file that is the output of GameSquare will be called *FindGameSquare_output.txt,* which will be a plain text file that is located in the same directory as the rest of the files. In this file the first line will either contain a '0' or '1'. A '1' indicates that the gameboard is valid in the way it is set up. A '0' indicates an invalid gameboard that the protocol will have to discard. This file also contains blocks of information. The first line in each block will have two integers separated by white space, the first is a integer represents a color and the second is a integer represents the (n) number of pins that are of the color. The next n lines will each contain two integers separated by white space. The first integer is the x-coordinate board square location of the pin and the second integer is the y-coordinate board square location of the pin. There will be at most 4 such blocks of data, each blocks gives information about the location of a set of pins of each color.


FindGameSquare_output.txt - Not...
File Edit Format View Help

```
1
1 1
10 15
2 1
15 4
3 2
7 10
20 11
```

Figure 1.b: Sample FindGameSquare_output.txt

The numbering for each color is as follows:

0 - yellow (corner)
1 - blue (item)
2 - red (hand)
3 – green (distractor)

Hardware Design Team: Jeffrey W

Introduction

The Hardware design team's duties are to create physical mock ups of what the finished GroZi device might look like, from an industrial design point of view.   They also are tasked with the fabrication aspects of the project. They will be creating various devices to assist in the tracking of the blind user's hand as well as creating a prototype of the shoulder mounted camera system.

Objectives met this quarter…

This quarter we created several concept drawings and one set of CAD models for what a final revision of the product might also look like.

We began with two concept drawings for the final product. Both of the designs are whimsical in their nature. A third and forth concept, both in a more serious tone were also developed. The first design is a classic boxy robot design (see figure 5.1a) . It has been modeled in CAD and a series of renderings were made. The second design is very similar to the Wall-E character from the Pixar/Disney movie and the Apple iSight webcam (see figure 5.1b). The second design was not modeled in CAD. The third design is based on a Sony camera like the ones used on the 3rd floor of the CSE building. This design was also not modeled in CAD. The fourth design is based on the iPhone and a Microsoft webcam. Both design 3 and 4 are monocular designs and will require some additional software to pull out the position of the hand. Designs 1 and 2 are both bi-ocular and use one camera for product tracking and the second for tracking of the hand. The main benefit for the monocular design is that the coordinate system for the hand tracking and the coordinate system for the object tracking will be the same. The bi-ocular system will require a linear shift to align the coordinate systems, assuming the respective cameras have the same resolution and lens configuration. The monocular solution also has a reduced part count and reduced need for support electronics and USB bandwidth and consumes less power. The monocular solution also means that instead of two cameras of lower resolution, a single higher resolution camera could be used for the same cost due to the reduced need for the support components.

The Wiimote team brought fourth an idea to use a fixed IR light source and a reflective ring or similar fob to allow the hand tracking. This removes the need for an active system on the user's hand and should reduce the overall cost of the final design. Furthermore it is easier to use since the IR light source can be controlled by software. Another benefit to this design is that the IR light can also be used to provide additional illumination for the object tracking system.

The Industrial design team also decided to build the shoulder mounted camera system as a USB device so that it is simpler for the OpenCV team to deal with the cameras. This also solves the problem of getting the camera signals to the host machine and the power issue of powering a camera and its support electromechanical system. The USB cable can provide all needed power for the electronics and the camera. If the mechanical systems are built judiciously and the drive electronics for those systems are equally well designed, the whole thing should be able to operate from the 5V, 500mA supply a USB port provides. However, it is minimally difficult to add external power lines if needed in the form of a custom cable. One motivation for this path is if the motors that position the camera inject too much noise into the power supply rails.

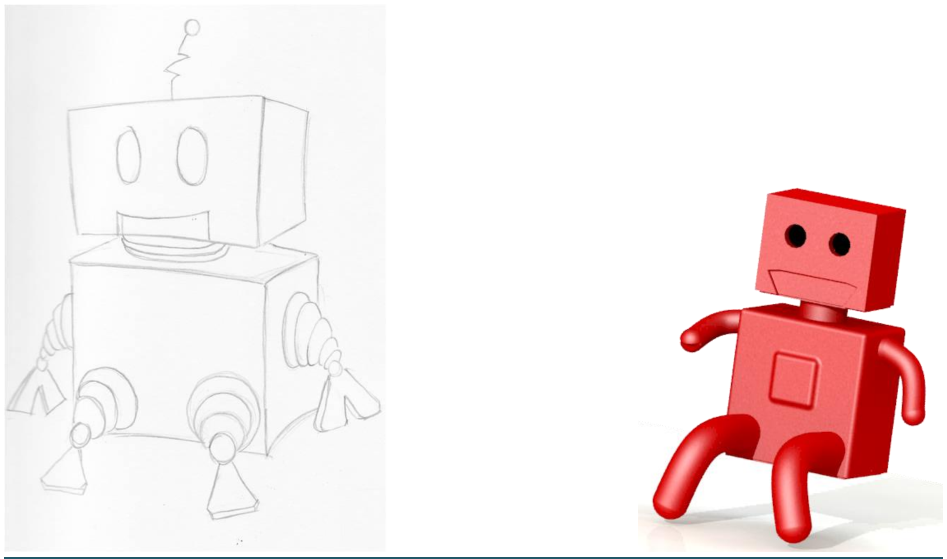


Figure 5.1a(Left to Right): *Classic Boxy Robot initial design; Classic Boxy Robot (CAD version)*

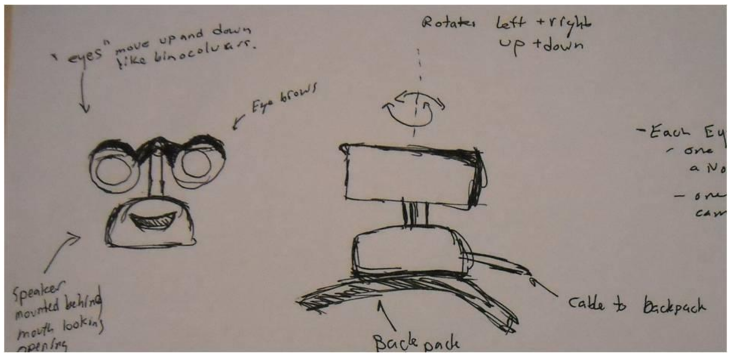

"eyes" move up and down like binoculars.
Eye brows
Rotates left + right up + down
- Each Ey
- one
a No
- one
cam
Speaker mounted behind mouth looking opening
Cable to Backpack
Back pack

Figure 5.1b: *Wall-E prototype*

This quarter the industrial design team created a list of requirements for what the final system must contain.

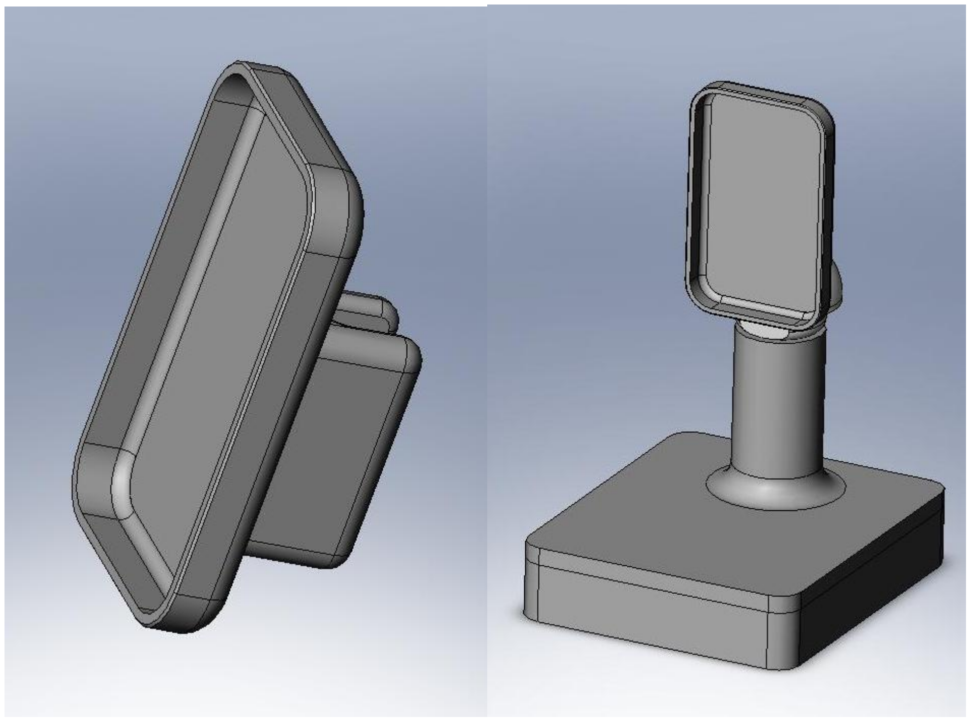
Industrial Design

1. The shoulder mounted camera assembly shall not weigh more than 1 pound.
2. The shoulder mounted camera assembly shall not interfere with the movements of the user's head, within 30 degrees of the plane of the shoulder.
3. The shoulder mounted camera assembly shall not have any sharp edges.
4. The shoulder mounted camera assembly shall be on the user's right hand shoulder.
5. The shoulder mounted camera assembly shall carry the camera for the computer vision system.
6. The shoulder mounted camera assembly shall contain an IR light source.

7. The shoulder mount shall carry a system for tracking the position of the user's hand.

The present iPhone based design should meet all of the requirements listed above. The most critical requirements are the camera carry details and ergonomics requirements. Since the iPhone-based design is specifically designed for production via injection molding, it has rounded edges and low weight due to the use of plastics. The shoulder mount from the Spring 2009 team can be modified to carry the iPhone-based design on the user's shoulder. The iPhone based design has not matured enough to have the front of the camera carrier designed in CAD, however, it will have both the camera used for computer vision and the IR light source on it.

This quarter we continued to model the iPhone based design. The design has advanced to include a base and support structure for panning and tilting the camera. The camera enclosure was modified to interface with the new base and support structure.



The team also helped with board design in terms of materials selection and guidance in the early stage of the design of the new demo board. Lastly, Jeff helped John get Visual Studio and SVN repository access installed and configured on his laptop.

Goals for next quarter

- Complete the iPhone CAD Model

        - Model the front panel of the camera carrier. This sub assembly will contain the camera and IR light source.
- Create block diagram for hardware for preprocessing images and extracting the hand position
    - Begin designing off board hardware for preprocessing the imagery from the camera.
        - Preprocessing will include keystone correction, image stabilization, and hand tracking.
        - USB interface that is compatible with OpenCV

References

1. http://www.wiili.org/index.php/Compatible_Bluetooth_Devices

2. http://wiibrew.org/wiki/List_of_Working_Bluetooth_Devices
(Compatible Bluetooth Devices)

3. http://www.wiimoteproject.com/bluetooth-and-connectivity-knowledge-center/a-summary-of-windows-bluetooth-stacks-and-their-connection/

4. http://www.dev-toast.com/2007/01/05/uncrippling-bluetooth-in-vista-rtm/

5.http://www.rapidsharedownload.net/software/widcomm-bluetooth-software-5.1.0.1100/

6. http://www.wiili.org/forum/bluecove-210-on-bluez-tips-t6355.html

7. http://xii9190.wordpress.com/page/15/ (mainly to find out how to set vibrate)

http://www.wiili.org/forum/wiiremote-disconnection-problem-t5359.html (help on disconnection problem, but I did it slightly differently in my code)

8. http://www.wiili.org/forum/bluetooth-fails-to-initialize-without-wiiremotedisconnect()-t6805.html

http://wiki.multimedia.cx/index.php?title=PCM (Audio file needs to be a signed 8-bit PCM)

9. http://grozi.calit2.net/files/TIESGroZiSp09.pdf