Grocery Shopping Assistant for the Blind (GroZi)

UCSD TIES—Fall 2009

Faculty Advisor:
Serge Belongie

Student Advisors:
Grace Foo

Community Client:
National Federation of the Blind (NFB)

Client Representative:
John Miller—NFB Representative

Visiting Scholar:
Masaaki Kokawa

Team Members:
Tess Winlock, Jeffrey Wurzbach, Tom Mueller, Amir Shirkhani, Cankut Guven, Christine Luu, Nil Kumar, Jeffrey Lu, Tejal Vora, Aritrick Chatterjee, Darren Lou, Joo Byoung Park.

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, especially in regards to their experiences within supermarkets and stores. 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 mechanical aspects of the project including the design of the GroZi parrot device, the mechanics and theory behind its functionality, and an experimental design phase that improves understanding of what changes and modifications are necessary for better efficiency of the device and its usability. 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: Joo and Artrick
Demo Board Design Team: Darren and Christine
Protocol Team: Nil and Jeffrey L
Computer Vision Sighted Guide (CVSG) team: Cankut, Nil, Jeffrey L
Hardware Design Team: Jeffrey W, Tom, Tejal

WiiMote Team: Aritrick Chatterjee, Joo Park

Introduction:

The goal of the Wiimote team is to direct a blind user toward the desired grocery product through the strategic use of haptics. If the device is to communicate with the user through haptics, then it must first be able to visually detect and locate a grocery product as well as the user's hand relative to the grocery product. The machine must then process and use the information to send haptic feedback to the user.

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 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 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 GroZi projects specific needs.

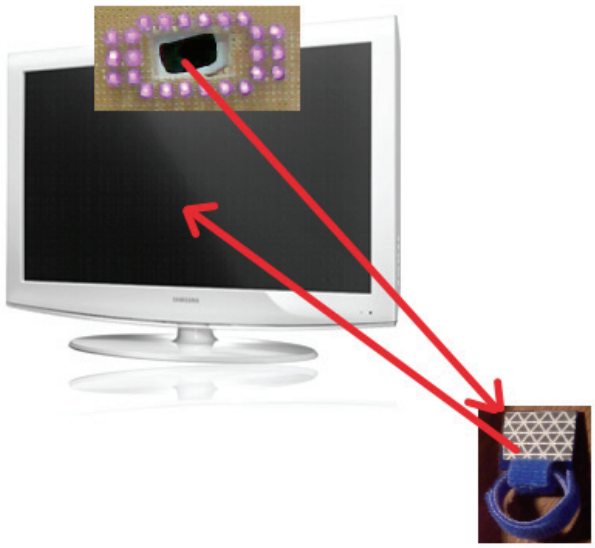

Figure 1.1

General Overview and Usage

The objective of the Wiimote/Bluetooth 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 a Bluetooth Connection

The first step was to establish the Bluetooth connection. This task proved to be rather frustrating and involved tedious hours of troubleshooting. In fact, this proved to be the more difficult task of this quarter. 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. Furthermore, using the Windows Vista OS may have complicated the process even more.

The computer needs a Bluetooth device installed in order to be Bluetooth capable. Some computers come with an internal device, while others need an external adapter. One of the 7 problems faced this quarter was finding a Bluetooth device which worked with the Wiimote. After doing research the team was able to find a list of compatible and incompatible devices. The entire list is included in [9]. The list also includes working Bluetooth driver stacks to be used with the specific device. Specific driver stacks also needed to be used in order to successfully establish connection. This quarter 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.

Even with a listed compatible device and driver stack, the team still encountered connection problems. The Microsoft Bluetooth stacks installed on the Microsoft OS caused problems and interfered with the additional driver stacks which were needed. The solution to this problem was to rip, or remove and disable the Microsoft stacks from the system. After this was done the other Bluetooth driver stack was able to act independently without interference. Ripping the Microsoft Bluetooth stacks was a long process in itself. Instructions to perform this task are provided in the appendix. This guide is almost exclusively copied from http://www.dev-toast.com/2007/01/05/uncrippling-bluetooth-in-vista-rtm/, and was edited according our own requirements.

Objectives met this Quarter:

Finger Tracking using IR light and Wiimote-

The main aim is to create a system that allows us to track the users hand/ finger. Using an Infrared LED and some reflective tape, you can use the infrared camera in the Wii remote to track objects, like your fingers, in 2D space. This lets you interact with your computer simply by waving your hands in the air similar to the interaction seen in the movie "Minority Report"

(http://jhonnylee.net/projects/wii/). It has a video on how to use an LED array and a Wiimote for tracking up to four points. It gave us a direction in which we wanted to direct the Wiimote teams work this quarter.

Finger Mount Design (reflective illumination)

Instead of using reflective tape, a permanent solution for reflection of IR light is necessary. The finger mount was the solution to this issue. The finger mount is mainly a Velcro strap made into a ring with high quality reflective tape mounted on an acrylic piece. The ring is made of Velcro. This allows the user to adjust the ring size and it makes it accessible for use for all individuals. The reflective tape is stuck to the acrylic plate that is mounted facing the shoulder. The wiimote and the IR LED are placed on the shoulder. This allows for reflective illumination for the wiimote and allows us to know the finger position. There was a lot of debate whether to go with active or reflective illumination. Active illumination would be to take an IR LED pointing at the shoulder where the wiimote is placed. We decided to avoid this as this would lead to the eyes being exposed to IR radiation could be harmful.

Another consideration is to which finger or which position the finger mount must be worn on. We looked at various fingers and positions and the index finger was the best option. If finger mount is worn over the index finger closest to the palm it gives the best probability of having the area exposed to IR light and thus allows for successful finger tracking. Measure such as hand rotation and other hand movements were taken to make this decision.

How the system works

IR LED→ Reflection off reflective Tape on finger mount → IR detection (Wiimote)

The system works on the simple mechanism. The reflection of IR radiation supplied by the IR LED sources from the shoulder off the finger mount is detected on the Wiimote. The wiimote which is connected to the computer and thus the program will get acceleration and pitch values. The program has written code to convert this into the position of the hand. Taking the shoulder to be at the origin (0, 0, 0) and knowing/determining the finger position (x, y, z) we can use the distance formula to know the hand's distance.

When this is further works in tandem with Remote Sighted Guide, the distance of object of interest from the shoulder can be tracked. We can further determine the position of the hand from the object that is being tracked which then allows us to give an instruction to the user to reach out to the tracked/ desired object.

All the material was bought from Staples- the reflective tape (~$15) and Velcro (~$4). The acrylic board was fabricated in the EBU 2 MAE workshop. The items are easily accessible and cheap and it takes no time to create a finger mount device. Make sure to put reflective tape on both sides of the acrylic finger mount as a visually impaired person will have decision to make on what side to wear. This quarter we worked with a TV remote which has an inbuilt IR LED but later Grozi project are recommended to buy a LED/ LED array.

Sample Java code

This is a section of the java code that is responsible for tracking the hand position that leads us to track the finger.

```
public void accelerationInputReceived(WRAccelerationEvent evt) {
    if (accelerometerSource) {
        lastX = x;
        lastY = y;
        lastZ = z;
        // green line: coordinates for wiimote coming towards person and leaving
person (z-axis)
        // red line: coordinates for wiimote going left and right (x-axis)
        // blue line: coordinates for wiimote going up and down (y-axis)
        x = (int) (evt.getXAcceleration() / 5 * 300) + 300;
        y = (int) (evt.getYAcceleration() / 5 * 300) + 300;
        z = (int) (evt.getZAcceleration() / 5 * 300) + 300;
        t++;
        graph.repaint();
```

What's next?

Future considerations and recommendations

In the future the main aim is to integrate the Wiimote and IR reflection system with Remote Sighted Guide. This integration will enable us to track objects and allows the user to be guided to the tracked item. There are questions as to when should the finger tracking work. John came up with interesting scenarios whether finger tracking should work every 10 sec or only the 1st time. It would be a good idea to get the finger position and object position to continuously update the protocols needed to be given by the RSG. This is something that needs to be looked into by future TIES/GroZi teams. Demonstration with the demo board can allow us to come up with a good explanation to this.

There can be an error with having two different cameras for object tracking and finger tacking. This gives a wrong object distance from the hand which can be an area of concern. Future teams should consider integration the two cameras or having them very close to each other to remove this distortion. The wiimote teams had a few problems with integration of the wiimote to the Bluetooth connection with the computer using Windows 7. This is something we want to work on to make this stable and reliable connection.

Demo Board Design Team: Darren Lou & Christine Luu

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 demonstrate and prove static image identification of items and the methods used to guide the user to the item.

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. Thumbtacks 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. electrical tape glue/adhesive
Board (edge to edge including electrical tape 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 electrical tape 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), blue (hand location), green (distractors), yellow (corner demarcations)

Table 2.1: *Board Specifications*

What's next?

Future Considerations and Recommendations

Overall, the construction of the demo game board serves its purpose well to the Vision and Protocol teams. However, there were a few minor obstacles that the other teams came across upon implementing their algorithms and protocols. One issue involved the light reflective properties of the board itself and the electrical tape boundary on the outer edge of the demo board. When under well lit conditions, the tape and the foam core board emit a glare that doesn't carry over a color value in the static image taken by the webcam. This poses a problem to the CVSG algorithm because it makes recognition and line detection of the "game area" and border very difficult. This can be easily fixed by constructing a new board with non-reflective materials such as felt or cloth. It was also proposed that we consider making the target game piece -- in this case the red thumbtack-- tactually distinguishable from the rest of the thumbtacks in order for it to be easily recognized when found by the test subject. Another future consideration that is of lesser priority includes making the board more portable.

Protocol Team: Nil Kumar & Jeff Lu

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".

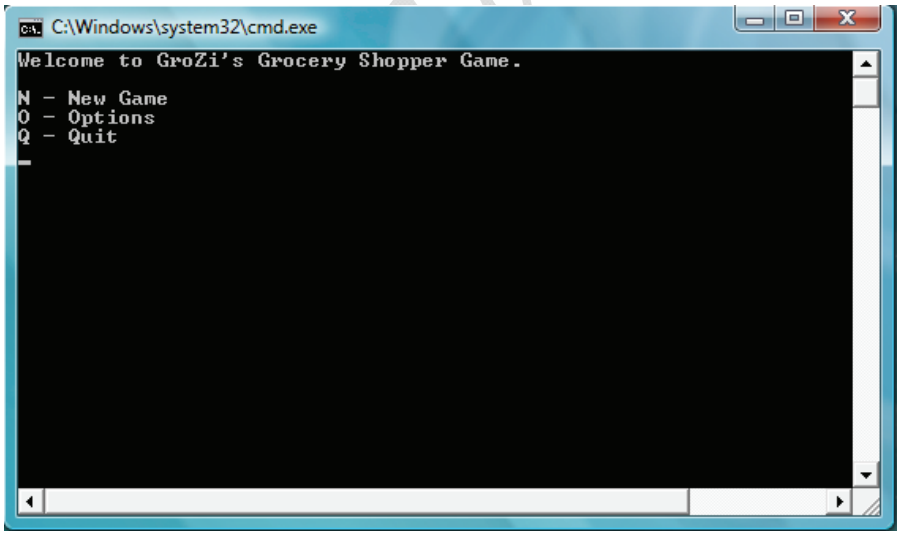

Figure 3.1a : *GroZi Grocery Shopper Game main menu options*

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

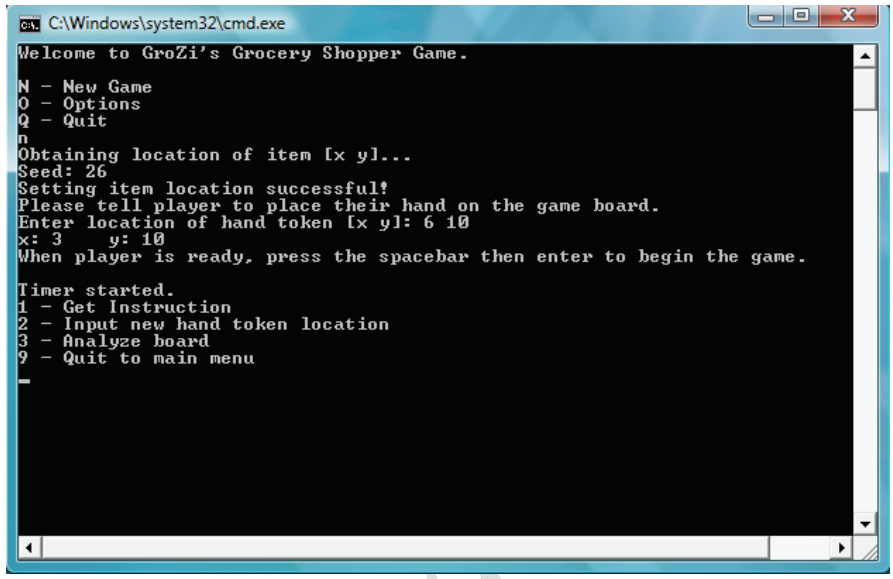

Figure 3.1b: *In play menu options*

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).

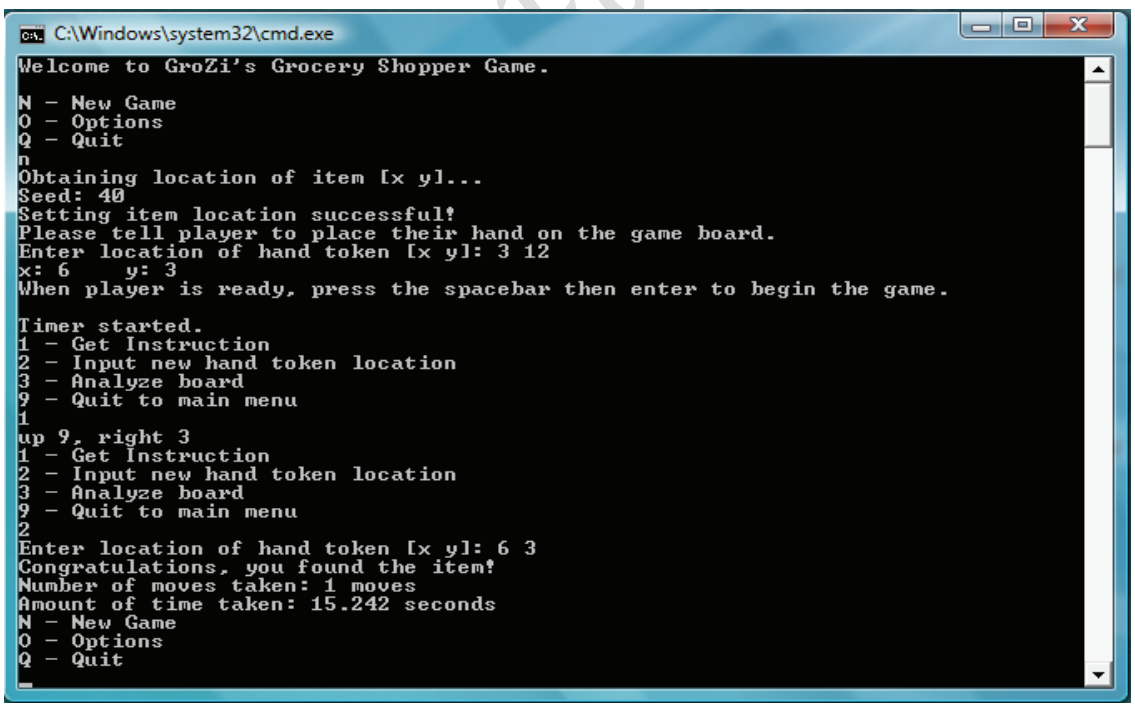

Figure 3.1c: *Output from a full game, beginning to end with user finding target item*

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;
            }
```

```c
/* 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

Trials to generate protocol

Currently, the only trials that have taken place are the ones that were used to design 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. Additionally, since we were trying to simulate a grocery shelf, we decided to split up the board into three quadrants. There was an "upper", "middle", and "lower" shelf. There were equally spaced and were usually referenced when the player was on the wrong shelf. 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 following is a sample output of the game using a 15 x 15 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 thirty timed trials using the 8x8 Chess board. Trials were conducted and timed by three different individuals. Prior to the start of each trial, the timer pointed out a random spot on the board to the director. The trial then started when the player selected an arbitrary spot on the board and asked "is this it". Director then went on to provide directions until the player found correct location.

Trial Times (without shelves, arbitrary starting position)—see table 3.3b below
Size of board: 8 x 8 (Chess board)

Player: Tim		Director: Jeff		Timer: Nil	
Time in seconds	20.83	22.51	16.37	7.52	4.15
Player: Tim		Director: Nil		Timer: Jeff	
Time in seconds	12.80	14.49	7.56	9.40	5.74
Player: Nil		Director: Jeff		Timer: Tim	
Time in seconds	11.27	5.77	5.26	8.15	16.24
Player: Nil		Director: Tim		Timer: Jeff	
Time in seconds	7.45	4.05	4.42	7.16	4.92
Player: Jeff		Director: Tim		Timer: Nil	
Time in seconds	5.38	6.63	4.27	7.77	7.97
Player: Jeff		Director: Nil		Timer: Tim	
Time in seconds	5.79	2.43	2.04	4.97	1.39

Table 3.3b: *Trial times*

What's next?

Future Considerations and Recommendations

In the near future, the protocol team hopes to complete the code for running a virtual simulation of the game. Some code that needs to be refined is the menu for running the program and the calculation of the moves. The refinement of calculating the moves also requires the implementation of a variable dimension of the board. In addition to refining the code, the group needs to implement a movement tracker and counter, a database for storing the trials for the player, a calculation of the average time, and a validation check to see if the game is valid.

After a virtual game can be played without problems, the protocol team wants to integrate with the Wiimote team and experimental design team to make use of the game board that was designed during this quarter. From there the protocol needs to be integrated with OpenCV code. This will allow the group to do game demos on the physical board with item tokens. After this is complete, if a database for the protocol team has been implemented, the group will be able to do plenty of simulations to analyze what could happen in a grocery store.

Computer Vision Sighted Guide (CVSG) Team: Cankut Guven, Nil Kumar, Jeffrey L.

Introduction

The CVSG(Computer Vision Sighted Guide) team's tasks are 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 added by a software overlay. The idealized environment will include a “desired” token, on a demo board, and a series of "noise" tokens. The software will determine the location of the green candy and report it to the user.

This quarter, the CVSG team was able to accomplish two tasks: Calibration of the camera in use (and quick calibration of any other camera that might be used) and detection of a board with detailed specifications under pre-determined conditions.

General Overview and Usage

In order to detect the board, some initial assumptions were made. Detection is successful under the following conditions:

- Lighting such that glare from the tape is not too bright (the setting BINARY_COLOR_THRESHOLD can be modified to accommodate lighting, which is explained further on)
- 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 board is not at a large angle with the camera (a ~15 degree slant is allowed). This angle refers to the perpendicular from the camera's lens to the board (i.e. the degree would be 0 if the camera were directly facing the front of the board)

If these conditions are met, the board's edges will be detected and outlined with a red line. Another task that was met this quarter was the calibration of the camera. It was not until later in the quarter that calibration surfaced as an issue. Figure 4.1a is an image of a non-calibrated screenshot taken of the board. As can be seen, the lines are not entirely straight with respect to the board's borders, especially the line running across the bottom border of the border which exceeds off the board and towards the exterior. Figure 4.1b shows an image taken seconds later with the same camera, but this time with the distortion of the camera taken into account. The lines in this image tend to match the board's borders more accurately and thus provide the algorithm with a better chance of detecting the board.

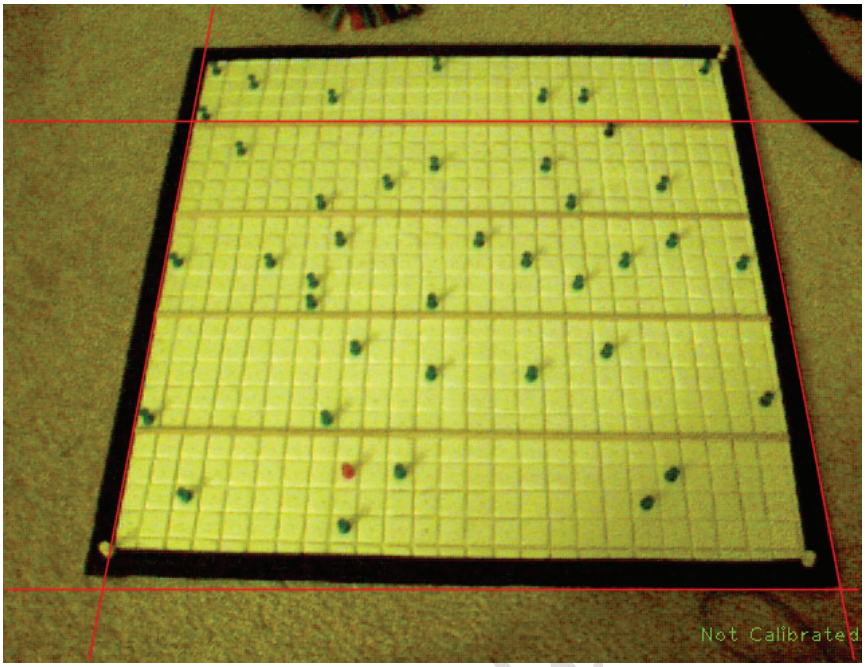

Figure 4.1a: *image of a non-calibrated screenshot taken of the board*

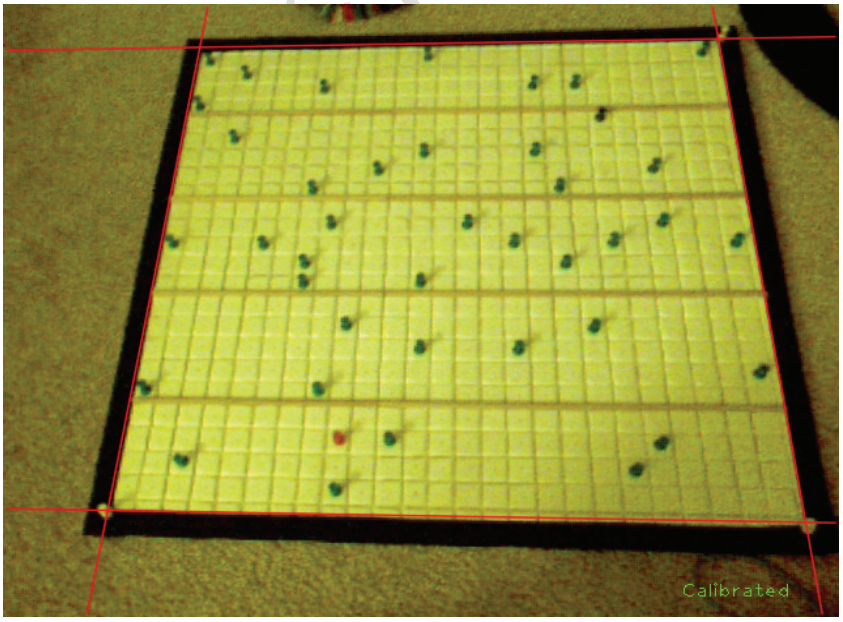

Figure 4.1b: *image taken seconds later with the same camera, but this time with the distortion*

Camera Caliberation

In order to calibrate the camera, one can follow these steps (this assumes the camera is connected to the computer and video is functional):

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. Run the calibration program included in the samples directory of OpenCV (<OpenCV_directory>/sampes/c/): calibrate -n 25 -w 7 -h 10
    I. n = frames
    II. w = width of chessboard (8 blocks, starting at 0)
    III. h = height of chessboard (11 blocks, starting at 00
3. Put chessboard within view of the camera and press 'g' to begin capturing images
4. Once 25 images have been taken, copy the image files view000.jpg – view0024.jpg to the calibration_data folder for the GroZi program.
5. Once the GroZi program is run, the camera should now be calibrated.

Testing has shown that under similar lighting, minor movement of the camera should not require recalibration. However, if a new setting is to be used, the camera should be recalibrated to ensure proper results.

There yet remain many goals to be accomplished. One of the goals for next quarter is to be able to play a full game on the board with the integration of the CVSG team's software and the Protocol Team's software such that the CVSG software can detect the user's hand, report to the protocol software, which will in turn process the information, and ultimately guide the user to the destination with no human intervention. In addition, hopes are to be able to devise a conversion system between computerized x,y and the board's x,y (e.g. 214,100 on the image for the computer could be 3,4 for the board).

APPENDIX

Clarification of #defines in board.h

- DEBUG: Four levels of debug exist:
  - 0: No debugging messages/windows
  - 1: Minimal debugging messages/windows
  - 2: Moderate debugging messages/windows
  - 3: Maximum debugging messages/windows
- NUM_CORNERS_IN_A_SQUARE:
  - Self explanatory, no longer used (perhaps future use?).
  - Can be safely removed for now
- CALIBRATION_IMAGE_COUNT
  - Number of images used to calibrate the camera (as of this writing 25 images are used, though that can be lowered or raised to fit needs)
  - Side note: 25 images can seem excessive, it was chosen with 25 frames per second in mind for videos. There is no concrete data as to whether this is truly of any benefit.
- BINARY_COLOR_THRESHOLD
  - This is the threshold number used to convert the frame (or image) into a black and white image (see getBinaryColor() to see details of its use).
  - A higher number means that the lighting experienced is rather bright.
  - A lower number indicates that the board is in a dark setting.
  - Recommendations:
    - If you're in a softly lit room, a setting from 1-10 is most likely the best option
    - "Regularly" lit room, a setting of 40-60 is recommended.
    - If shining a bright light directly on the board, a higher number is recommended, such as 70-90 would work best, depending on the brightness of the source.
  - It is expected most rooms will be moderately lit, therefore a setting of 50 should work for most situations.

Hardware Used:

- Camera: Microsoft LifeCam VX6000
- Computer: Compaq Laptop F761US
- Operating System: Windows 7 Pro 64-Bit
- SDK: Microsoft Visual Studio 2008
- Tripod: Velbon CX 200

Hardware Design Team: Tom, Jeffrey W, & Tejal Vora

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.

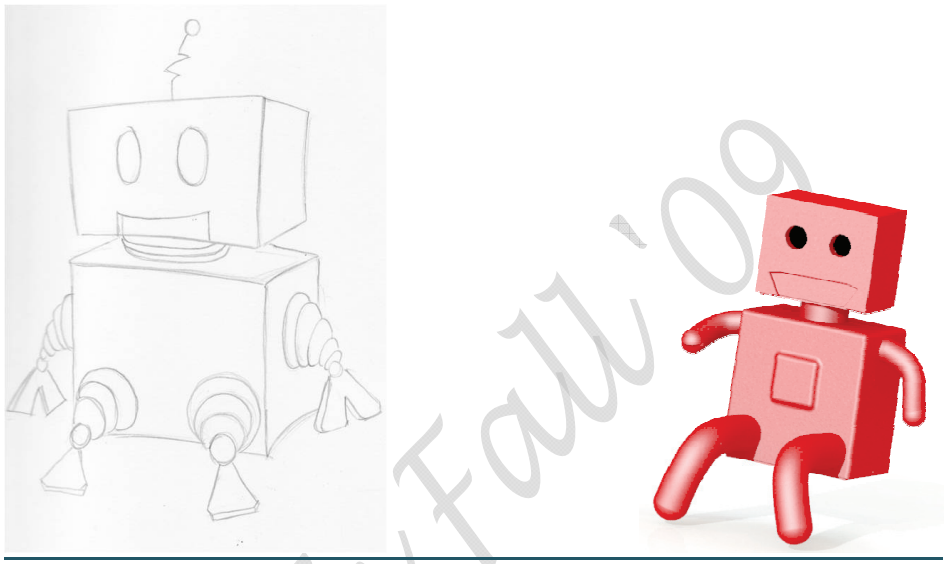

Figure 5.1a(Left to Right): *Classic Boxy Robot initial design; Classic Boxy Robot (CAD version)*

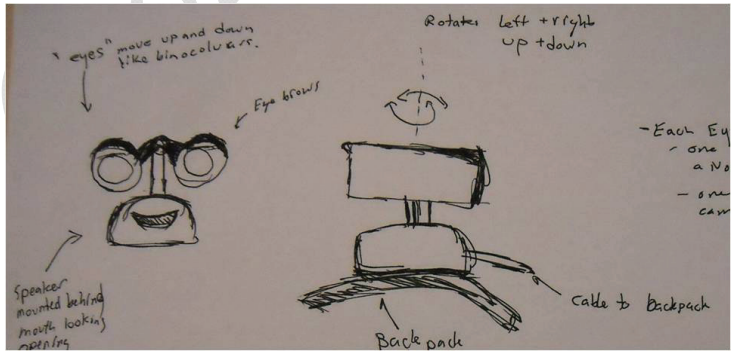

Figure 5.1b: *Wall-E prototype*

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