**Wiiqualizer**

**LAB 7**

**SECTION A**

**SUBMITTED BY:**

**Brian Reber**

**SUBMISSION DATE:**

**11/6/2009**

**Lab Problem**

Our goal is to use the sensors inside the Wiimote to help us figure out the pitch and roll of the Wiimote. We will be creating a live ‘equalizer’ type model on the screen. Our goal in this project is to practice top-down program design, writing functions, working with loops, and output parameters.

**Analysis**

The goal of this lab is to print out an equalizer graph that represents the current roll or pitch of the Wiimote. We should be able to change from roll to pitch using the one and two buttons on the Wiimote. To do this we will need to implement multiple functions and loops.

**Inputs**:

There will be data coming in via the Wiimote and Wiiwrap.

**Outputs**:

The output should look something like this.

0

rrrrrrrrrrrr

rrrrrrrrrrrrrrrrrrrr

rrrrrrrrrrrrrrr

rrrrrrrrr

rrrrr

rr

r

lll

llllllllll

lllllllllllllllllllll

Meaning that the Wiimote starts sitting flat, and then rolls to the right, and rolls back to the left.

**Design**

Necessary Formulas:

* Roll = asin(acceleration\_x)
* Pitch = asin(acceleration\_y)

Algorithm:

1. First off, we will create a function called “read\_acc” that will scan the data from Wiiwrap and then use the output parameters passed in to change the value of the variables in the main function so that we can use the data. Also, this function will return true (1) if either button a or the home button have been pressed. We will catch this return value and stop our do-while loop if this is true.
2. We will then implement the function “roll” that will take the acceleration in the x direction and return the arcsine of this value (as long as it is between -1 and 1). This value will be used to figure out the amount of ‘r’ or ‘l’ to print when we are in roll mode.
3. We will then implement the function “pitch” that will take the acceleration in the y direction and return the arcsine of this value (as long as it is between -1 and 1). This value will be used to figure out the amount of ‘r’ or ‘l’ to print when we are in pitch mode.
4. We then need to implement the function “scaleRadsForScreen.” This function will take the radian value returned by “pitch” or “roll” and return it with a scaled value that we can use for printing on the screen.
5. We then need to implement the function “print\_chars.” This function will contain a for loop that prints out the specified number of the specified character on the screen.
6. We next implement the “graph\_line” function. This function will take one parameter, the scaled number given returned by “scaleRadsForScreen”.
   1. If the number is zero, we will print out 39 spaces, and then a zero.
   2. If the number is greater than zero, we will print out 39 spaces, and then ‘number’ amount of ‘r’s.
   3. If the number is less than zero, we will print out (39+’number’) amount of spaces, then (-‘number’) amount of ‘l’s. This part was a bit tricky because the number was less than zero, so we had to add, and then use the negative amount of ‘l’s.
7. In the main function, we will have a variable created for each important button, time, one that tells us whether we need to stop the loop, accelerations, roll, pitch and a scaled value. We use the do-while loop given, edit it to stop when “read\_acc” returns 1. Inside that loop, we assign the returned value of “read\_acc” to our run variable. We calculate roll and pitch. We check whether the one or two button had been pressed, and switch modes if so. Then we scale the radian value of either pitch or roll depending on our current mode. We then graph it.

**Testing**

We ran into a few small problems when writing this program.

* One problem we experienced was with the pitch/roll functions. Sometimes, the x\_mag or the y\_mag were greater than 1, which is not allowed when taking the arcsine. To solve this problem, we added 2 if-statements in each function. If the magnitude was greater than 1.0, we would assign the magnitude to 1. And if it was less than -1.0, we would assign it to -1.0. This solved our problems with this.
* Another problem we ran into was with the graph line function. When we were printing out the ‘l’ character, we forgot that our number was negative. Apparently the computer doesn’t know how to print out a negative number of something. So to solve this, we just used the negative value of number (essentially taking the negative value of a negative number, therefore making it positive).
* We ran into another problem with the challenge part – dealing with the b button switching. I figured that it was because we read in data so often that by the time we released our finger from the b button, it had switched back to the original mode. This is why it said it was harder to do this than originally thought. I was able to bypass this problem by creating a variable that tells me whether the b button was pressed the previous time through the loop.

**Comments**

Lab Questions

* In order to scale our values we used the formula (radians \* .5 ((SCREENWIDTH-1) / (PI / 2))). This formula will take the constant SCREENWIDTH and subtract 1 because we don’t want to count the center character. It will then divide that by (PI / 2), which is the maximum value that radians will be. We then divide by two because there are two sides we need to account for. So far, this has just given us our scalar value. Now we multiply that value by our variable, radians to get our scaled value. This formula works because our scalar guarantees that we won’t go above 1. When the Wiimote is on its side, radians and PI / 2 will cancel each other out.
* When we put the Wiimote on its side and push it over, this is what is displayed.

lll

llllll

lllllll

llllllll

llllllllll

llllllllll

lllllllllll

lllllllllll

llllllllll

llllllllll

llllllll

llllllll

llllll

llll

0

rr

rrrrr

rrrrrrr

rrrrrrrr

rrrrrrrrr

rrrrrrrrrr

rrrrrrrrrrr

rrrrrrrrrrr

rrrrrrrrrr

rrrrrrrrrr

rrrrrrrrr

rrrrrrr

rrrrrr

rrr

0

ll

llll

lllllll

llllllll

llllllll

llllllllll

llllllllll

llllllllll

llllllllll

llllllll

llllllll

lllllll

lllll

lll

0

rr

rrrrr

rrrrrr

rrrrrrrr

rrrrrrrrr

rrrrrrrrr

rrrrrrrrrr

rrrrrrrrrr

rrrrrrrrr

rrrrrrrr

rrrrrrrr

rrrrrr

rrrr

r

0

The amount of r’s or l’s that get printed start out big, and then the number continues to get smaller until the Wiimote stops moving. This makes sense because the Wiimote starts making bigger rotations, and gravity slows it down to a stop.

* Each letter in our graph represents about 2.25 degrees. To figure this out, I took PI / 2 and divided that by 40 because that is the amount of characters printed when the radian value is PI / 2. I then converted this value into degrees. When the Wiimote was close to being on its side, I noticed that there were some problems; every so often there would be one line that printed out about 10 characters too few. I am not sure why exactly this is. But otherwise, when it was being rolled, it seemed to sort of hover at the side – there amount of letters that were printed was really close, whereas when it was coming back from its side the number of letters would change more rapidly.

Other Comments

In general, I thought that this lab wasn’t too difficult to figure out. The toughest part was making it fit the given parameters. Making it print out the given output wasn’t difficult, but doing it in a certain way was somewhat difficult.

**Implementation**

// wiiqualizer.c

//

// Brian Reber and Nathan Brinkman

//

// This is the outline for your program

// Please implement the functions given by the prototypes below and

// complete the main function to make the program complete.

// You must implement the functions which are prototyped below exactly

// as they are requested.

#include <stdio.h>

#include <math.h>

#define PI 3.141592653589

#define SCREENWIDTH 80

//PRE: Arguments must point to double variables

//This function scans a line of wiimote data, and returns

// True when either HOME or A is pressed

// False Otherwise

//POST: it modifies its arguments to return values read from the input line.

int read\_acc(float\* a\_x, float\* a\_y, float\* a\_z, int \*t, int\* Button\_one, int \* Button\_two, int\* Button\_B, int\* Button\_A);

// PRE: -1.0 <= x\_mag <= 1.0

// This function computes the roll of the wiimote in radians

// POST: -PI/2 <= return value <= PI/2

double roll(double x\_mag);

// PRE: -1.0 <= x\_mag <= 1.0

// This function computes the pitch of the wiimote in radians

// if x\_mag outside of -1 to 1, treat it as if it were 1 or -1

// POST: -PI/2 <= return value <= PI/2

double pitch(double y\_mag);

// PRE: -PI/2 <= rad <= PI/2

// This function scales the roll value to fit on the screen

// POST: -(1/2 screen width - 1) <= return value <= (1/2 screen width -1)

int scaleRadsForScreen(double rad);

// PRE: num >= 0 and use = a valid character

// This function prints the character use to the screen num times

// POST: nothing is returned, but use has been printed num times

void print\_chars(int num, char use);

//PRE: -39 <= 0 <=39

// Uses print\_chars to graph a number from -39 to 39 on the screen.

// You may assume that the screen is 80 characters wide.

void graph\_line(int number);

int main()

{

int t; //time

int run = 0; //will be false unless read\_acc returns true

int rollRad = 2; //2 means we will print out the graph for roll, 1 means pitch

float ax, ay, az; // magnitude values of x, y, and z accelerations

int b\_a, b\_b, b\_home, b\_1, b\_2, b\_junk, b\_previous; // variables to hold the button statuses; b\_junk is just a temp variable

double roll\_rad, pitch\_rad; // value of the roll measured in radians

int scaled\_value; // value of the roll adjusted to fit screen display

//insert any beginning needed code here

do

{

// Get line of input

run = read\_acc(&ax,&ay,&az,&t,&b\_1,&b\_2, &b\_b, &b\_a);

// calculate roll and pitch

roll\_rad = roll(ax);

pitch\_rad = pitch(ay);

// switch between roll and pitch(1 or 2 button)

if (b\_1)

{

rollRad = 1;

}

if (b\_2)

{

rollRad = 2;

}

if (b\_b)//if the b button is pressed

{

//check to see that the b button wasn't pressed the last run through

//if it was pressed last time, it will not change the mode

if (!b\_previous){

if (rollRad == 1) rollRad = 2;

else if (rollRad == 2) rollRad = 1;

}

//set it so that it knows that the previous time through, the b button was pressed

b\_previous = 1;

}

//resets b\_previous if the b button was not pressed (resets it)

if (!b\_b && b\_previous)

b\_previous = 0;

// Scale your output value

if (rollRad == 2)

scaled\_value = scaleRadsForScreen(roll\_rad);

if (rollRad == 1)

scaled\_value = scaleRadsForScreen(pitch\_rad);

// Output your graph line

graph\_line(scaled\_value);

} while (!run); // Modify to stop when A is pressed

return 0;

}

int read\_acc(float\* a\_x, float\* a\_y, float\* a\_z, int \*t, int\* Button\_one, int \* Button\_two, int\* Button\_B, int\* Button\_A)

{

int b\_junk;

int b\_home;

scanf("%d,%f,%f,%f,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d", t,a\_x, a\_y, a\_z,

Button\_A,Button\_B,&b\_junk,&b\_home,&b\_junk,Button\_one,Button\_two,&b\_junk,&b\_junk,&b\_junk,&b\_junk);

return (\*Button\_A || b\_home);

}

double roll(double x\_mag)

{

if (x\_mag > 1.0)

x\_mag = 1.0;

if (x\_mag < -1.0)

x\_mag = -1.0;

return asin(x\_mag);

}

double pitch(double y\_mag)

{

if (y\_mag > 1.0)

y\_mag = 1.0;

if (y\_mag < -1.0)

y\_mag = -1.0;

return asin(y\_mag);

}

int scaleRadsForScreen(double rad)

{

return rad\*(((1.0/2.0)\*SCREENWIDTH-1)/(PI/2));

}

void print\_chars(int num, char use)

{

int count;

for (count = 0; count < num; count++)

{

printf("%c", use);

}

}

void graph\_line(int number)

{

int spaceChars = SCREENWIDTH/2 -1;

if (number == 0)

{

print\_chars(spaceChars, ' ');

print\_chars(1,'0');

}

if (number > 0)

{

print\_chars(spaceChars, ' ');

print\_chars(number, 'r');

}

if (number < 0)

{

print\_chars((spaceChars+number), ' ');

print\_chars((-number), 'l');

}

printf("\n");

fflush(stdout);

}