WHAT’S UP CONDITONALS LAB REPORT

LAB #4

SECTION M

SUBMITTED BY:

SHOUNAK LAHIRI

SUBMISSION DATE:

9/22/2018

Problem

The purpose of this lab is to modify a program to output the orientation of the Dualshock 4 controller while the controller is not moving and only output when there is a change in the orientation of the controller as opposed to continuously outputting. To find the orientation of the controller the program should make use of the gyroscopic values, and to find when the controller is not moving the program should use the acceleration. Additionally, the program must include at least three functions and stop executing when the triangle button on the controller is pressed.

Analysis

The gyroscopic values fall between -1 and 1 with some amount of variance on both ends of the range. To account for the variance there should be a tolerance value that will be added and subtracted form the points -1 or 1 depending on the direction of the gyroscopic value. After looking at a lot of the incoming data for the gyroscopic values, we decided that .25 was a good number to be the tolerance to account for the variation in the numbers. The various variables and their goal values are listed below:

Orientation	Axis	Value
Right	GX	1.0
Left	GX	-1.0
Top	GY	1.0
Bottom	GY	-1.0
Front	GZ	1.0
Back	GZ	-1.0

Design

Our problem was to output the orientation of the controller when it was not moving. This task could be broken into smaller problems/steps.

1. Output the correct orientation (continuously)
2. Output the orientation only when the orientation changes
3. Output the orientation when the controller is not moving
4. Exit the program when the triangle button is pressed on the controller

To start we had to create a function called close_to that would receive the tolerance, point, and value. It would then create a variable named difference and take the absolute value of the difference between point and value. If the difference was less than the tolerance, then the function would return 1 (true) otherwise it would return 0 (false). The function close_to was called multiple times inside of a large if then else statement in the main function. Each if branch was testing to see if the values they sent to the close_to function was inside the tolerance level, if they were then they continued with the code inside of the if brackets which said to print the orientation of the controller based on the axis and levels being sent to the close_to function, if they weren't then the code continued to evaluate the if statements until one was true. This system printed the orientation of the controller correctly.

To make the code print only when the orientation of the controller changed, we decided to use the idea of flags, which would help keep track of which orientation was currently being printed. For the flag, we decided to use a char because we had so many different orientations it was easier than creating multiple Boolean variables. The char would take the first letter of each direction when that orientation was being printed. The if branches in the main function were altered to add the condition that the char was not equal to that branches orientation character. By changing the if conditions and creating the char flag, we were able to only print when the orientation of the controller changed.

To only output when the controller was not moving we decided to use the acceleration values in addition to the mag function that was written in Lab 3. The mag function took in the acceleration values for each axis, then calculated the magnitude of the acceleration and returned it to the function call. We used the magnitude function call inside of a call to the close_to function to see if the acceleration was inside of a tolerance level that we could deem as not moving. We added an if statement around the main if branches which tested to see that the acceleration was zero. This ensured that the program was only outputting the change in orientation when the controller was not moving.

To make it so the program stopped running when the triangle button was pressed on the controller, we had to find which button number in the scanf statement corresponded to the triangle button on the controller. Once we found that button 1 was the triangle button, we added a function called ButtonTest, which tested to see if button 1 was pressed. We added another if statement inside of the while loop that checked whether the ButtonTest function was returning true (1), if it was then we added the code for the program to exit, stop running.

Testing

To test that the program was working correctly, we checked it after completing each smaller step. First, we checked that when the controllers orientation was changed the program was outputting the correct orientation. Then, we checked that the orientation was only being outputted when the controller's orientation was changed. Third, we shook the controller around while changing the orientation to check that the orientation was being outputted only when the controller was not being moved. Lastly, we pressed the triangle button while changing the orientation of the controller to test that the program was exiting promptly when the triangle button was pressed.

Comments

I learned that where the function calls are placed inside of the code can make a huge difference, especially when the code is inside of a loop. We did encounter a problem where the program was only outputting top or bottom for the orientation, even when the controller was turned to the right. The problem was the way that the close_to function was being calculated, by changing it we were able to correct the problem.

Questions/Experiments:

1. I approached the design by the smaller steps that I needed to complete, like getting the correct orientation output, only outputting when the orientation changed, when the controller was not moving, and when the triangle button was pressed exiting the program.
2. For all the functions to work correctly, we needed the acceleration values for each axis, the gyroscopic values for each axis, and at least the btn1 value.
3. I chose to implement the mag function to see when the controller was moving, and outputting when it was not moving. I used the close_to function to test the points that fell near our value with tolerance in mind. Lastly, I added the ButtonTest function to see when the btn1 button (triangle) was pressed.
4. I picked .25 as the tolerance value for the gyroscopic axis because when I was analyzing the incoming gyroscopic values, none of the values seemed to be off by more than .3.

Implementation

```
/*-----------------------------------------------------------------------------
-                                         SE/CprE 185 Lab 04
-             Developed for 185-Rursch by T.Tran and K.Wang
-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------
-                                  Includes
-----------------------------------------------------------------------------*/
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>


/*-----------------------------------------------------------------------------
-                                  Defines
-----------------------------------------------------------------------------*/



/*-----------------------------------------------------------------------------
-                                  Prototypes
-----------------------------------------------------------------------------*/
double mag( double x, double y, double z);
int close_to (double tolerance, double point, double value);
int ButtonTest ( int button);


/*-----------------------------------------------------------------------------
-                                              Implementation
```

```
-----------------------------------------------------------------------------*/
int main(void)
{
    int t, b1, b2, b3, b4;
    double ax, ay, az, gx, gy, gz;
      char c;


    while (1)
      {
         scanf("%d, %lf, %lf, %lf, %lf, %lf, %lf, %d, %d, %d, %d", &t, &ax, &ay,
&az, &gx, &gy, &gz, &b1, &b2, &b3, &b4 );

             if(ButtonTest(b1) == 1)
             {
                    exit(0);
             }



             if ( (close_to(.50, 1.0,mag(ax,ay,az)) == 1))
             {

                    if(close_to(.25, 1.0 , gy) == 1 && c != 't')
                    {
                           printf("TOP\n");
                           c ='t';
                    }
                    else if (close_to(.25, -1.0 , gy) == 1 && c != 'd')
                    {
                           printf("BOTTOM\n");
                           c ='d';
                    }
                    else if (close_to(.25, 1.0, gx) == 1 && c != 'r')
                    {
                           printf("RIGHT\n");
                           c= 'r';
                    }
                    else if (close_to(.25, -1.0 , gx) == 1 && c != 'l')
                    {
                           printf("LEFT\n");
                           c= 'l';
                    }
                    else if (close_to(.25, 1.0, gz) == 1 && c != 'b')
                    {
                           printf("BACK\n");
                           c= 'b';
```

```
                        }
                        else if (close_to(.25, -1.0, gz)== 1 && c!= 'f')
                        {
                                printf("FRONT\n");
                                c= 'f';
                        }


                        fflush(stdout);
                }
        }

        return 0;
}

/* Put your functions here */

double mag(double x, double y, double z)
{
        double value= pow(x,2) + pow(y,2) + pow(z,2);
        return sqrt(value);

}

int close_to(double tolerance , double point, double value)
{
        double difference;
        difference = fabs(point - value);
        if (difference <= tolerance)
        {
                return 1;
        }
        else
        {
                return 0;
        }

}

int ButtonTest( int btn1)
{
        if (! btn1 == 0)
        {
                return 1;
        }
        else
        {
                return 0;
        }

}
```