THE DS4 EQUALIZER LAB REPORT

LAB #6

SECTION M

SUBMITTED BY:

SHOUNAK LAHIRI

SUBMISSION DATE:

10/24/2018

Problem

The Dual Shock controller provides data to the user, but being able to visualize the data is not as simple for the user. To make visualizing the data a simple process we needed to create a program that could have a visual representation of the pitch and roll of the controller, where pitch is defined as moving the controller back and forth, and roll is moving the controller left and right. The program should aid in visualizing the data by creating a graph that shows when the controller is moving in the right direction by printing r's and changing the number of r's depending on the magnitude of the motion. Similarly, the program should print l's when the controller is tilted to the left , depending on the magnitude of the motion; these directions would be for tilting the controller backwards and forwards if the program was in pitch mode. Additionally, the program can only contain one printf and scaf statement.

Analysis

To help us calculate what the values of roll and pitch were, we were provided the equations roll = asin(g_x) and pitch = asin(g_z), where g_x is the magnitude of the gyroscope in the x direction and g_z is the magnitude of the gyroscope in the z direction. We also had to figure out how to scale the incoming values (ranging from -PI/2 to PI/2) to range from -39 to 39. To do this we found that multiplying the incoming values with 78 and diving the result by pi gave us a value inside the range from -39 to 39. The goal of the program is to seamlessly display the magnitude of the tilt and print that many r's or l's. Additionally, the program should be able to switch between roll and pitch modes by pressing buttons, triangle for roll mode and x for pitch mode; the program should stop running is the square button is pressed.

Design

Our problem could be broken into smaller pieces, which would easily fit into the provided functions.

1. Getting user input
2. Getting a value to fit the screen based on the magnitude
3. Outputting with the correct formatting

To get the user input the skeleton code said to use the read_line function. One of the conditions for the program was that it can only have one printf and one scanf statement. The scanf statement for the entire program was placed in the read_line function because the read_line function had parameters which included pointers to variables. The pointers enabled the function to change the variables that are defined in the main function, without having to create new temporary variables and sending them back to the main function. The scanf statement gets the values for time, gyroscopic, acceleration, and button, to get them into the corresponding variables, they need to be placed in the correct order that they are coming in in, and the variables which have pointers do not need to have ampersands before them in the scanf statement because there are already ampersands in the call to read_line (in the main function). Some of the variables like time and the values for acceleration are not used in the program.

The next thing to do was to make the values for the gyroscopic values to map to a scaled value between -39 and 39, the length of the screen (provided in lab manual). After the read_line function we had the gyroscopic values in the correct variables in the main function. We then needed to correct the magnitudes of these gyroscopic values, so we wrote an if statement that made values less than -1 equal to -1, and values greater than 1 equal to 1. We placed the if statement in the roll and pitch functions, sending the g_x value to the roll function and the g_y value to the pitch function. The roll/pitch functions, according to the skeleton code, should calculate the roll/pitch. In the lab manual we were provided the equations for calculating the roll/pitch, roll = asin(g_x) pitch = asin(g_y). To incorporate this equation in to the if statement that was already placed in the roll/pitch functions we added a return statement, which returned the roll/pitch value according to the equations (listed above). Once the values were returned to the main function we needed to scale the values to make them between -39 and 39 instead of between -pi/2 and pi/2. To do this we called the scaleRadsForScreen function, sending the value from the roll/pitch functions. Inside the scaleRadsForScreen function we multiplied the incoming roll/pitch value by 78.0 (to avoid integer division), then divided the result by pi. The result was a number between -39 and 39, which was returned to the main function and stored in the scale_value variable.

Lastly, we needed to output the correct information. We started by writing the print_chars function. Inside the print_chars function we wrote a for loop with a printf inside the loop. By placing the printf statement inside the loop, we satisfied the condition that there can only be on printf in the entire program. The only remaining function was the graph_line function which got the scaled_value from the main function. The graph_line function included one if statement which said that if the number was equal to 0, meaning the controller was flat, we need to print a 0 in the center of the screen. To do this we needed to call the print_chars function and print 39 spaces, then print one 0, then print one new line. The if statement then continued to test if the number was less than zero, the controller was moving to the right, print 39 spaces, then multiply the number by -1, to make it positive, the print that many r's using the print_chars function. The last condition in the if statement, was if the number was positive, the controller was tilted left, then print 39 minus the number spaces, then print l's the number times. After that the program would be working correctly.

Testing

To test the program, we ran the program and moved the controller, tilting it from right to left and left to right. We also pressed the x button to test the pitch mode, moving the controller in the same way, then pressing the triangle button to switch back into roll mode. Finally, ending the testing session by pressing the square button and making sure the program stopped running. There were not examples for the test because the exact angle that the controller is tilted is difficult to recreate by hand. The ultimate test of whether the program was working correctly was to demo it to a TA.

Comments

I think the design that was implemented was pretty good at doing everything the program had to do in a timely manner. I learned that writing down the steps on paper is really helpful especially when there is math involved. Additionally, I learned that when using variables in pointer format in scanf statements there is no need to use ampersands.

Questions/Experiments

1. I scaled my values by multiplying the number that was between -pi/2 and pi/2 by 78.0, to avoid integer division I added the decimal, then I divided the result by pi. This works because it divides a circle into 78 parts, where each part maps to a certain number of degrees. The number of degrees would be helpful in determining the tilt of the controller.
2. Each letter would correspond to about 2.3 degrees. Near the limits of the graph, if the controller is turned too much it counts it as the magnitude of the other direction.

Source Code

```
// 185 lab6.c
//
// 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

//NO GLOBAL VARIABLES ALLOWED


//PRE: Arguments must point to double variables or int variables as
appropriate
//This function scans a line of DS4 data, and returns
//  True when the square button is pressed
//  False Otherwise
//This function is the ONLY place scanf is allowed to be used
//POST: it modifies its arguments to return values read from the input
line.
int read_line(double* g_x, double* g_y, double* g_z, int* time, int*
Button_T, int* Button_X, int* Button_S, int* Button_C);


// PRE: -1.0 <= x_mag <= 1.0
// This function computes the roll of the DS4 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 roll(double x_mag);

// PRE: -1.0 <= y_mag <= 1.0
// This function computes the pitch of the DS4 in radians
// if y_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: -39 <= return value <= 39
int scaleRadsForScreen(double rad);

// PRE: num >= 0
// This function prints the character use to the screen num times
```

```
// This function is the ONLY place printf is allowed to be used
// POST: nothing is returned, but use has been printed num times
void print_chars(int num, char use);

//PRE: -39 <= number <=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);
//uses print_chars

int main()
{
	double x, y, z;				// magnitude values of x,
y, and z
	int time , b_Triangle, b_X, b_Square, b_Circle;	// variables to
hold the button statuses
	double roll_rad, pitch_rad;		// value of the roll
measured in radians
	int scaled_value;
	int mode = 0;

	// value of the roll adjusted to fit screen display

	//insert any beginning needed code here

	do
	{

		// Get line of input
		read_line(&x, &y, &z, &time, &b_Triangle, &b_X, &b_Square,
&b_Circle);
		// calculate roll and pitch.  Use the buttons to set the
condition for roll and pitch

		roll_rad = roll(x);
		pitch_rad = pitch(y);

		// switch between roll and pitch(up vs. down button)
		if(b_Triangle == 1)
		{
			mode = 0;
		}
		else if(b_X == 1)
		{
			mode = 1;
		}

		// Scale your output value
		if(mode == 0)
```

```
            {
                  scaled_value = scaleRadsForScreen(roll_rad);
            }
            else if(mode == 1)
            {
                  scaled_value = scaleRadsForScreen (pitch_rad);
            }
            //scaled_value = scaleRadsForScreen(gen_rad);
            // Output your graph line
            graph_line(scaled_value);

            fflush(stdout);
      } while (read_line(&x, &y, &z, &time, &b_Triangle, &b_X, &b_Square,
&b_Circle)==1);
      return 0;
}//End Main


int read_line(double* g_x, double* g_y, double* g_z, int* time, int*
Button_T, int* Button_X, int* Button_S, int* Button_C)
{
      double ax, ay , az;
      scanf(" %d ,%lf, %lf, %lf, %lf , %lf , %lf , %d , %d , %d , %d",
time, &ax, &ay, &az, g_x, g_y, g_z, Button_T, Button_C, Button_X,
Button_S);

      if(*Button_S == 0)
      {
            return 1;
      }
      else
      {
            return 0;
      }
}//End ReadLine



double roll(double x_mag)
{

      if(x_mag < -1)
      {
            x_mag = -1;
      }
      else if (x_mag > 1)
      {
            x_mag = 1;
      }
```

```
	return asin(x_mag);

}

double pitch(double y_mag)
{
	if(y_mag < -1)
	{
		y_mag = -1;
	}
	else if (y_mag > 1)
	{
		y_mag = 1;
	}
	return asin(y_mag);

}

int scaleRadsForScreen(double rad)
{
	return (rad * 78.0) / PI;
}

void print_chars(int num, char use)
{
	int i;
	for( i =0; i < num; i++)
	{
		printf("%c", use);
	}
}

void graph_line(int number)
{

	int spaces = 0;
	if(number == 0)
	{
		print_chars(39, ' ');
		print_chars(1 ,'0');
		print_chars(1, '\n');
	}
	else if(number < 0)
	{
		number *= -1;
		spaces = 39;
		print_chars(spaces, ' ');
		print_chars(number, 'r');
		print_chars(1,'\n');
```

```
        }
        else
        {
                spaces = 39 - number;
                print_chars(spaces, ' ');
                print_chars(number, '1');
                print_chars(1, '\n');
        }


}


//When the controller is tilted to the left, the number is +; when the
controller is tilted right the number is -
```