The DS4 Equalizer

Lab 6

Section M

Submitted by:

Haadi-Mohammad Majeed

Submission Date

26.10.2018

Date

24.10.2018

Lab Problem

The objective of the lab is to output a visual text base graph that is centre aligned with zero being the middle/rest point, and right and left being outputted respectively from the centre as the controller was tilted around.

Analysis

The output of the program will depend solely on the orientation of the controller, at rest it should be zero, at an angle left or right it should output r or l respectively. The code should exit on the square button being pressed and should switch between roll and pitch upon pressing the x and triangle buttons.

Design

The template provided gave us a rough outline of what each method had to achieve and what needed to occur within each one as there was a precondition of what was going into the method, and what was going out, labeled the post condition. The first method was the only place where scanf was allowed and was used to import the controller data from ds4rd into the programme. It utilized pointers as “output” multiple variables. The second was used to calculate the angle that the controller was at during that instance of the loop returning in radians. The next method was the exact same however dealt with the pitch instead of the roll and returned the y aspect instead of the x, yet still returned an angle in radians. scaleRadsForScreen simply converted the radian input from radians to a value between -39 to positive 39. PrintChars does exactly as it entails, it uses loops to output the correct angle relation so either, 0, r or l this method is the only printf, and is exclusively called from the next method, graph_line; This function calls the printChars with the arguments of a character and how much it should be repeated so it can correctly output the number.

Testing

This lab did not supply test data, so I had decided to create some of my own, thus the day of the lab, I recorded some basic tilting recordings and button inputs as to have the ability to test from anywhere, not just the labs. I then ran the code against it and was receiving an output that appeared correct at first, but upon further inspection, the r’s were offset due to a logistical error.

Comments

One of the more interesting parts of this lab, was figuring out that the order that the buttons were initialized, was not the order that the buttons were read in from DS4RD which lead to a tonne of confusion when trying to get certain buttons to cause different functions. To fix this I ran the command ./ds4rd -d 054c:05c:4 -D DS4_BT -b just to figure out what button correlated to what in the order of input/outputs. One thing that was very frustrating with this lab was the inconsistencies in variable naming, some were snake case, others camel, and yet some were a mix of the two. Working with them in such a fashion was agitating and caused many errors in each compile.

Implementation

```
// 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* Button_T, int*
Button_C, int* Button_X, int* Button_S)
{

    scanf("%lf ,%lf ,%lf ,%d ,%d ,%d, %d", g_x, g_y, g_z, Button_T, Button_C,
Button_X, Button_S);
    if (*Button_S == 1)
        return 1;
    else
        return 0;

}
```

```
// 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)
{

    if(x_mag > 1)
        x_mag = 1;
    else if(x_mag < -1)
        x_mag = -1;

    return asin(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)
{

    if(y_mag > 1)
        y_mag = 1;
    else if(y_mag < -1)
        y_mag = -1;

    return asin(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)
{

    return (int)(rad*78.0/PI);

}

// PRE: num >= 0
```

```c
// 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)
{
    int i;
    if (num <= 0) {
        for (i = 0; i < num + 40; i++)
        {
            printf(" ");
        }
        for (i = 0; i < abs(num); i++)
        {
            printf("%c", use);
        }
    }
    if (num > 0)
    {
        for (i = 0; i < 40; i++)
        {
            printf(" ");
        }
        for (i = 0; i < abs(num); i++)
        {
            printf("%c", use);
        }
    }
    printf("\n");
    fflush(stdout);
}

//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)
{

    if(number > 0)
        print_chars(number, 'r');
    else if(number == 0)
        print_chars(1, '0');
    else if(number < 0)
        print_chars(number, 'l');
}
```

```
int main()
{
    double x, y, z;                             // magnitude values of x, y, and
z
    int 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;                           // value of the roll adjusted to
fit screen display
    b_Square = 0;
    //insert any beginning needed code here
    int out;
    int state = 0;
    do
    {
        // Get line of input
        out = read_line(&x, &y, &z, &b_Triangle, &b_Circle, &b_X, &b_Square);
        //printf("%lf, %lf, %lf, %d, %d, %d, %d\n", x, y, z, b_Triangle, b_X,
b_Square, b_Circle);
        // calculate roll and pitch.  Use the buttons to set the condition for
roll and pitch
        if(b_Triangle == 1)
        {
            state = 1;
        }

        if (b_X == 1)
        {
            state = 0;
        }

        // switch between roll and pitch(up vs. down button)
        if(state == 0)
            roll_rad = roll(x);
        else
            pitch_rad = pitch(y);

        // Scale your output value
        if(state == 0)
            scaled_value = scaleRadsForScreen(roll_rad);
        else
            scaled_value = scaleRadsForScreen(pitch_rad);

        // Output your graph line
```

```
        graph_line(scaled_value);

        //fflush(stdout);
    } while (b_Square != 1 ); // Modify to stop when the square button is pressed
    return 0;
}
```

Pre-Lab

1. How did you scale your values? Write an equation and justify it.

The values were scaled by multiplying 39 by pi/2 because the conversion from degrees to radians calls for such

rad*78.0/PI

2. How many degrees does each letter in your graph represent? This is the

precision of your graph. As your experiment with the roll and pitch, what do you

notice about the graph's behavior near the limits of its values?
0.04027682889 is how many radians each letter represents, or to be answered in a cleaner fashion, pi/(2*39) with pi/2 being the max angle, and 39 being each potential increase.

As you angle it with a steeper input, the programme will output more characters to compensate.