

The DS4 Equalizer

LAB 6

SECTION C

SUBMITTED BY:

WESLEY REYNOLDS

SUBMISSION DATE:

10/28/25

Problem

The problem of this lab is to visualize either the roll or the pitch of a DualShock 4 controller as a function of time. The premise of the lab is that we are working on a game, with the controller as a data source, and need to visualize the readings from the controller. We are in a hurry and cannot produce proper graphics. We will need to graph the variables via a horizontal bar chart made of characters in the terminal. The lab defines roll as when the DualShock 4 is tipped left or right while holding it as one normally would. Pitch is defined as the angle from level that the DualShock 4 is tipped forward or backward.

Both pitch and roll may be positive or negative, so we will need a column to denote when the value is reading close to zero. If the value is positive, we should print the character ‘r’ to the right side of the graph, while if the value is negative, we should print the character ‘l’ to the left side of the graph. The number of characters should be proportional to the rotation of the controller. Additionally, we should be able to switch between roll and pitch during the program to be able to accurately test these for our “game.”

Analysis

We are reading the gravity affected accelerometer, button, and time inputs from a DualShock 4 controller. In this lab, we have no real use for the time input. Once the program is run, it should start off printing the roll of the controller. We should be able to switch between roll and pitch with the press of a single button (I went for the extra credit), and the program should exit once the square button is pressed.

There will need to be a bar graph where ‘0’ is halfway across the screen at column forty. As stated above, if the value is positive, we should print the character ‘r’ to the right side of the graph. If the value is negative, we should print the character ‘l’ to the left side of the graph. If the value is not large to justify printing either ‘l’ or ‘r’, ‘0’ should be printed in column forty. The number of characters should be proportional to the value being graphed such that a rotation of $\pi/2$ corresponds to 39 characters.

For clarity, when the DualShock 4 is reading roll and titled to the right, it should graph the character ‘r’ proportional to the amount of rotation on the right side. This is the same for when the controller is tilted to the left, but with ‘l’. Note that instead of being to the right of column forty, the first ‘r’ prints in column forty, while the first ‘l’ prints in column thirty-nine. When the DualShock 4 is reading pitch and is titled forward, it should graph the character ‘r’ proportional to the amount of rotation. This is the same for when the controller is titled backward, but with ‘l’.

Design

In my mind, there were *four* major problems that needed to be solved. I needed to:

- 1) Scale the controller's accelerometer values to between -39 and 39,
- 2) Graph the roll or pitch of the controller,
- 3) Switch between the roll or pitch with a single button press,
- 4) Only use one (non-debug) *scanf* and *printf* in the entire file.

To implement problem one, there are a few steps to follow. First, we read the controller's accelerometer input and put that into the arcsine function. This limits the value from $-\pi/2$ to $\pi/2$. We then divide this value by $\pi/2$. This will give us a range of -1 to 1. We can then finally multiply by 39 to get our desired range. This is explored further down in the questions section.

To explain how I solved the second problem, I will analyze my *graph_line* function. The function itself takes in one integer as parameter (called *number*), which is the controller's rotational value normalized from -39 to 39, and has a return type of void.

- First, I check if *number* is equal to zero. If it is, we know that the controller isn't being rotated in any which way, so we use my *print_chars* function to print thirty-nine whitespace characters followed by a '0'. The actual implementation of this is to print thirty-nine minus *number* whitespace characters, but when *number* is equal to zero it has the same effect.
- Second, I check if *number* is less than zero. If it is, we know that the controller is either being titled right or forward. Note that this happens because the controller reports a negative value when rotated in these directions. Knowing that the controller is tilting to the right, I print thirty-nine white space characters, followed by an amount of 'r' equal to *number*.
- Lastly, I check if *number* is greater than zero. If it is, we know that the controller is either being titled left or backward. We do not need to print on the right side of the graph in this case. I print a number of whitespace space characters equal to thirty-nine minus *number*, then I print a number of 'l' equal to *number*. In practice, if number were twenty, this would equate to nineteen whitespace characters followed by twenty 'l' characters.

The third problem was simple to solve. Initially, I had a time-based solution, where the button would only work a certain amount of time after a press. This would make it flip-flop back and forth after however many milliseconds I had set, and I was not pleased with that. Basically, I have a variable called *prevPressed* that can either be set to 0 or 1. If it is 0, the button was not pressed in the previous iteration. If it is 1, the button was pressed in the previous iteration. When the circle button is pressed and the button is not pressed on the previous iteration (*prevPressed* is equal to 0), we swap between roll and pitch using a *state* variable, where 0 and 1 equate to roll and pitch respectively. This also sets *prevPressed* equal to 0, meaning that the *state* variable will not switch again. *prevPressed* is only reset back to 0 once the circle button is released.

To implement the fourth problem, I simply utilized the functions provided to me by the skeleton code. My only *printf* is in *print_chars* and my only *scanf* is in *read_line*. I only end up using *read_line* once in my code. This is during the do-while loop. I use *print_chars* multiple times in my *graph_line* function as explained above, but no *printf*.

Testing

To verify that my program was behaving correctly, I fashioned a few tests.

- 1) Run your program and place the controller flat on a table. It should print a ‘0’ in the fortieth column.
- 2) Pick up the controller and tilt it to the right. It should graph the character ‘r’ in an amount proportional to how far you tilted it to the right.
- 3) Tilt it to the left. It should graph the character ‘l’ in an amount proportional to how far you tilted it to the right.
- 4) Now press the circle button (this was the button I designated as my switch) to switch to pitch.
- 5) Tilt it forward. It should graph the character ‘r’ in an amount proportional to how far you tilted it forward.
- 6) Tilt it backward. It should graph the character ‘l’ in an amount proportional to how far you tilted it backward.
- 7) Press and hold the circle button. It should only change once. Nothing else should happen.
- 8) Release and repress the circle button. It should now change again with nothing happening afterwards. This and the previous tests both show the consistency and reliability of my switching implementation.

My program successfully passes all of these tests.

Comments

I really think that my design here was great. It worked on my first attempt. It properly graphs the roll and pitch of the controller with the proper amount of 'r' and 'l'. I successfully implemented the extra credit, and it is not buggy. I managed to go into the lab and crack out all my code in thirty or so minutes, too. This goes to show just how much prior thought can help with code design. There were no real problems I ran into with this lab. The predefined structure took away most of the leg work that comes with designing a program. I know this is our 'introduction' to top-down program design, but I think the students should be given more freedom in how they implement all the requirements. This would make the lab more difficult, but more rewarding.

Questions

Part One:

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

I scale my values in a very simple way. With the way the program is set up, we read from one of the accelerometers on the controller and then use the arcsine function on it.

Arcsine can *only* return a value between $-\pi/2$ and $\pi/2$ due to the definition of arcsine. We can then divide our value by $\pi/2$ to get a value somewhere between -1 and 1. Once we are at this stage, all we need to do is multiply the value by 39. This gives us a scale from -39 to 39 based on the rotation of the controller.

Verbose Equation:

$$x_{scaled} = \frac{x_{original}}{\pi/2} * 39$$

Simplified Equation:

$$x_{scaled} = \frac{78 * x_{original}}{\pi}$$

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?

Each letter in my graph is roughly equal to 2.25 degrees of rotation. I got this value by taking the maximum left tilt (-90 degrees), the maximum right tilt (90 degrees), and

subtracting the two to get a value of 180 degrees. We are working with an 80-character graph, so we can divide 180 by 80 to get a degree per letter value of 2.25. The graph slows down as it reaches the limits of its values. This is explained by graphing and inspecting the arcsine function. Its growth slows down greatly as it approaches $-\pi/2$ or $\pi/2$.

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* time, int* Button_T, 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);

int main()
```

```

// You may assume that the screen is 80 characters wide.
void graph_line(int number);

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

    int time;
    int prevPressed = 0;                  // I am using this to ensure that the button doesn't double press
    int state = 0;                       // reading roll (0) or pitch (1)

    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(z);

        // printf("Roll: %2.4lf\n", roll(x));
        // printf("Pitch: %2.4lf\n", pitch(z));
        // printf("Screen Coord (r - p) %2d - %2d\n", scaleRadsForScreen(roll(x)), scaleRadsForScreen(pitch(z)));

        // switch between roll and pitch(up vs. down button)
        if (b_Circle == 1 && prevPressed == 0) {
            prevPressed = 1;

            switch (state) {
                case 0:
                    state = 1;
                    break;
                case 1:
                    state = 0;
                    break;
                default:
                    printf("We shouldn't be here!!!\n");
                    return -1;
            }
        } else if (b_Circle == 0) {
            prevPressed = 0;
        }

        // Scale your output value

        if (state == 0) {
            scaled_value = scaleRadsForScreen(roll_rad);
            //printf("Scaled Roll: %3d\n", scaled_value);
        }
    }
}

```

```

    if (state == 0) {
        scaled_value = scaleRadsForScreen(roll_rad);
        //printf("Scaled Roll: %3d\n", scaled_value);
        graph_line(scaled_value);
    }

    if (state == 1) {
        scaled_value = scaleRadsForScreen(pitch_rad);
        //printf("Scaled Pitch: %3d\n", scaled_value);
        graph_line(scaled_value);
    }

    // Output your graph line

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

return 0;
}

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)
{
    scanf("%d, %lf, %lf, %lf, %d, %d, %d, %d", time, g_x, g_y, g_z, Button_T, Button_C, Button_X, Button_S);

    if (*Button_S == 1)
        return 1;

    return 0;
}

double roll(double x_mag)
{
    if (x_mag > 1)
        x_mag = 1.0;

    if (x_mag < -1)
        x_mag = -1.0;

    return asin(x_mag);
}

double pitch(double y_mag)
{
    if (y_mag > 1)
        y_mag = 1.0;

    if (y_mag < -1)
        y_mag = -1.0;

    return asin(y_mag);
}

```

```

        x_mag = -1.0;
    }
    return asin(x_mag);
}

double pitch(double y_mag)
{
    if (y_mag > 1)
        y_mag = 1.0;

    if (y_mag < -1)
        y_mag = -1.0;

    return asin(y_mag);
}

int scaleRadsForScreen(double rad)
{
    // Alleviate some of the noise
    if ((rad < 0.03 && rad > 0) || (rad > -0.03 && rad < 0))
        rad = 0;

    rad = rad / (PI / 2);
    rad *= 39;
    return rad;
}

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

void graph_line(int number)
{
    if (number == 0) {
        print_chars(39 - number, ' ');
        print_chars(1, '0');
    }

    if (number < 0) {
        print_chars(39, ' ');
        print_chars(number * -1, 'r'); // Multiply by negative one because number is negative here.
    }

    if (number > 0) {
        print_chars(39 - number, ' ');
        print_chars(number, 'l');
    }

    print_chars(1, '\n');
}

```