CPRE 1850 – The DS4 Equalizer

LAB 6

SECTION CN

SUBMITTED BY:

MAXWELL MILLER

10/31/2025

Problem

The problem this lab tackles is building a user interface for the relatively arbitrary output of ds4rd.exe. We are asked to build an output using only the ASCII characters ‘0’, ‘r’, and ‘|’. These characters are then used to show the current roll or pitch of the controller in a user-friendly manner.

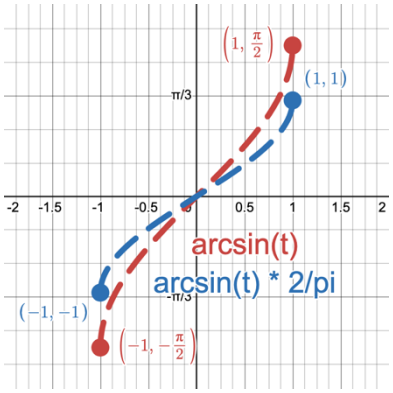
Analysis

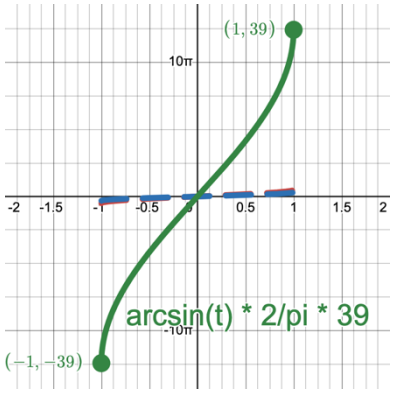
This will require modifying the scale of the gravitational relative acceleration to something that can be displayed in whole numbers, then it must be displayed using only the allowed characters.

Design

The range of arcsine is -$\pi/2$ to $\pi/2$, so in order to get out a range of -1 to 1 we must multiply by the inverse of $\pi/2$ – same thing as dividing by $\pi/2$. Then in order to get a range from -39 to 39, we simply multiply by 39.

$$\sin^{-1}(g_x) \cdot \left(\frac{\pi}{2}\right)^{-1} \cdot 39 = \frac{78 \sin(g_x)}{\pi}$$





In order to output to the terminal in a way that is user-friendly, a combination of ‘0’, ‘r’, and ‘|’. With ‘0’ representing around zero degrees, ‘r’ representing positive degree values, and ‘|’ representing negative degree values. The total length each output line is up to 78 characters long, with the first 39 characters in the negative direction, and 39 characters in the positive direction. If zero, it will add the ‘0’ character on the 40th space.

Testing

When testing as the controller reached its value limit, the output became quite unpredictable, even when holding the controller steady. This is likely due to the nature of the arc-sine function. It has a steep slope to it towards the endpoints, and that results in even slight variants in accelerometer data to have a much larger impact on what is displayed.

Comments

Each character represents: $\frac{360_{\text{deg/line}}}{78_{\text{chars/line}}} \approx 4.6_{\text{deg/char}}$

Code implementation

```c
// 185 lab6.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.141592653589

// Functional Prototypes
int read_line(int* time, double* g_x, double* g_y, double* g_z, int* Button_T, int* Button_C,
int* Button_X, int* Button_S);
double roll(double x_mag);
double pitch(double y_mag);
int scaleRadsForScreen(double rad);
void print_chars(int num, char use);
void graph_line(int number);

// Enum for mode selection with the controller button
enum Mode {
    ROLL_MODE,
    PITCH_MODE
};

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, apitch_rad;                 // value of the roll measured in radians
    int scaled_value;                           // value of the roll adjusted to fit screen
display

    int time;
    enum Mode mode = ROLL_MODE;                 // Default to roll mode
    int val;

    do
    {
        // Get line of input
        read_line(&time, &x, &y, &z, &b_Triangle, &b_Circle, &b_X, &b_Square);

        // calculate roll and pitch.
        roll_rad = roll(x);
        pitch_rad = pitch(y);
```

Continued on the next page.

```
        // switch between roll and pitch (up vs. down button)
        // Written like this for clarity
        if (b_Triangle) {
            if (mode == ROLL_MODE) {
                mode = PITCH_MODE;
            }
            else {
                mode = ROLL_MODE;
            }
        }

        // Scale your output value
        if (mode == ROLL_MODE) {
            val = scaleRadsForScreen(roll_rad);
        }
        else if (mode == PITCH_MODE) {      // Written this way for clarity
            val = scaleRadsForScreen(pitch_rad);
        }

        // Output your graph line
        graph_line(val);

        fflush(stdout);
    } while (!b_Square); // Stops when the square button is pressed

    return 0;
}


// Formatted weird to fit on the pdf page
int read_line(
        int* time, double* g_x, double* g_y, double* g_z,
        int* Button_T, int* Button_C, int* Button_X, int* Button_S
) {

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

    return (*Button_S);
}
```

Continued on the next page

```c
double roll(double x_mag) {
    if (x_mag > 1.0)    {   x_mag = 1.0;    }
    if (x_mag < -1.0)   {   x_mag = -1.0;   }

    // Negative to make the roll make visual sense on the screen
    // i.e. Left is left, right is right
    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) {
    // -PI/2 to PI/2 -> -1 to 1 -> -39 to 39
    return (rad * (2 / PI)) * 39;
}

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

void graph_line(int number) {
    int side = 39 - abs(number);

    if (number < 0) {
        print_chars(side, ' ');
        print_chars(abs(number), '|');
    }

    else if (number == 0) {
        print_chars(39, ' ');
        print_chars(1, '0');
    }

    else {
        print_chars(39, ' ');
        print_chars(number, 'r');
    }

    print_chars(1, '\n');
}
```