CPRE 1850 – Moving Averages

LAB 7
SECTION CN

SUBMITTED BY:

MAXWELL MILLER

11/07/2025

Problem

Data from accelerometers, in this case the DualShock controller, tend to be quite noisy, and so often times it is unclear on what the user's actual intended motion is. The goal of this lab is to implement a so-called "moving average" which uses a moving set of samples to determine a real time average.

Analysis

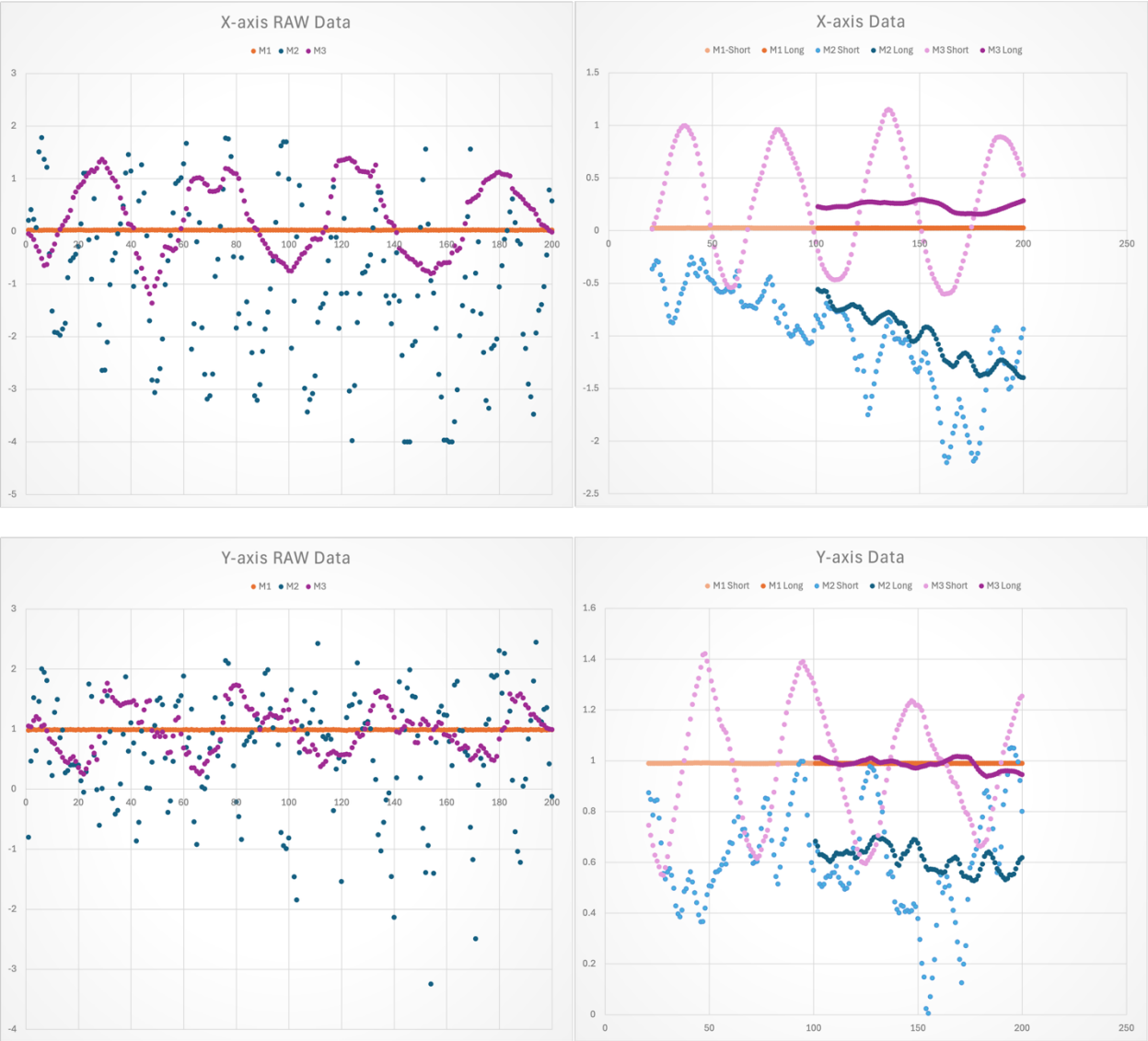
In order to accomplish a moving average, the program must be able to store a buffer of data of recent data points and then find the average of those values.

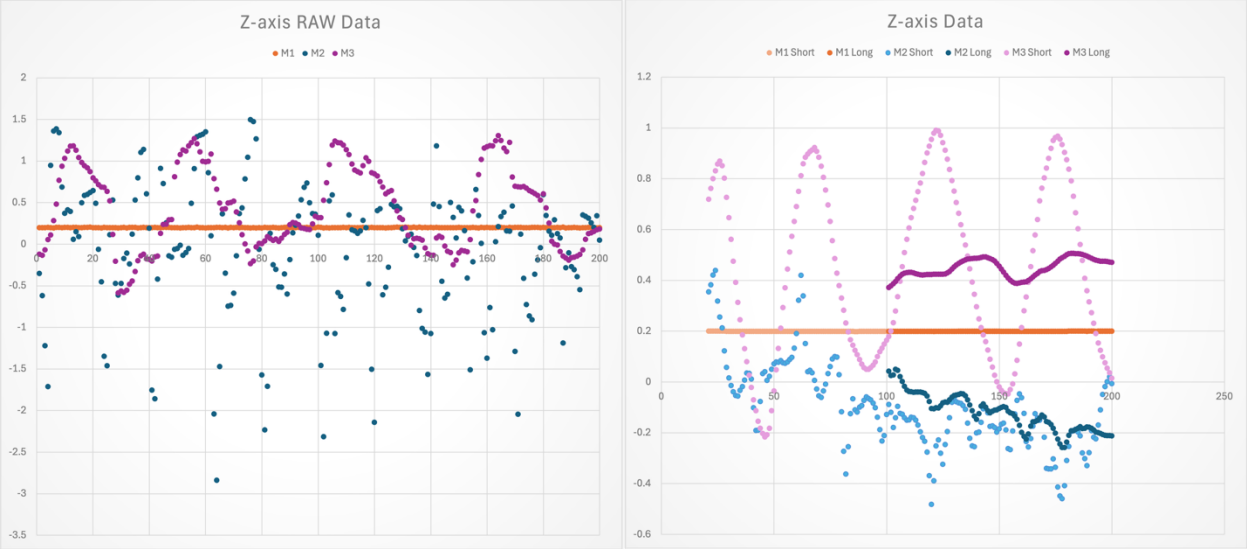
Design

I began with implementing a loop in the main function that takes a scanf input each iteration. Each time the loop executes, it calls updatebuffer, which shifts all values down one in each of the xyz arrays, then adds the most recent value to the lengthofavg – 1 index of the array. Then we check if there are enough values to calculate a moving average by checking if the "i" counter has reached the lengthofaverage value. If it hasn't then we just print zero for each of the moving average values and continue onto the next loop execution. Then we calculate the average of all values currently in the array and print them to the terminal. Finally, the program finds the maximum and minimum of each array and prints them out.

Testing

When testing the program, three motions were tested. The first was just still on the table (orange); the second was rocking the controller back and forth in a parabolic manner (blue); and the third was moving the controller in a cyclical fashion like you might stir a pot (purple).





When looking at the window length in the moving average for each test we see each of the raw graphs become much clearer when a moving average is applied because it eliminates outliers from the graph. For movement one it doesn't seem to matter much since the data is relatively stable from the get-go. However, when looking at movement two and three is becomes a different story. Because movement two is very rapid and random, the longer window gives more useable data with a clear trend. While for movement three the shorter window gives more clear data because the movement was not rapid, this gives the sinusoidal function that we would expect from the motion. With these repetitive motions as the window length increases, the data becomes more compressed and loses definition, but if the window is too small the outliers of the data will have too much weight, making the graph have little meaning.

Comments

While originally raw data was collected independently for both short and long windows, the data was out of phase of each other. As a result, to allow for a direct comparison, this report uses the same raw data for both window sizes for each movement when generating the moving average output.

Code implementation

```c
// 185 Lab 7
#include <stdio.h>


#define MAXPOINTS 10000

// compute the average of the first num_items of buffer
double avg(double buffer[], int num_items);

//update the max and min of the first num_items of array
void maxmin(double array[], int num_items, double* max, double* min);

//shift length-1 elements of the buffer to the left and put the
//new_item on the right.
void updatebuffer(double buffer[], int length, double new_item);


int main(int argc, char* argv[]) {

    /* DO NOT CHANGE THIS PART OF THE CODE */

    double x[MAXPOINTS], y[MAXPOINTS], z[MAXPOINTS];

    // Length of average
    int lengthofavg = 0;
    if (argc>1) {
        sscanf(argv[1], "%d", &lengthofavg );
        printf("You entered a buffer length of %d\n", lengthofavg);
    }
    else {
        printf("Enter a length on the command line\n");
        return -1;
    }
    if (lengthofavg <1 || lengthofavg >MAXPOINTS) {
        printf("Invalid length\n");
        return -1;
    }



    /* PUT YOUR CODE HERE */
```

Continued on the next page.

```
    printf("t,x,y,z,x avg,y avg,z avg,x max,x min,y max,y min,z max,z min\n");

    double x_inst, y_inst, z_inst = 0.0;
    int b_tri, b_cir, b_x, b_sqr = 0;

    double avg_x, avg_y, avg_z = 0.0;
    double max_x, max_y, max_z;
    double min_x, min_y, min_z;

    int i = 0;

    do {
        // Get new data point
        scanf(
            "%lf,%lf,%lf,%d,%d,%d,%d",
            &x_inst, &y_inst, &z_inst,
            &b_tri, &b_cir, &b_x, &b_sqr
        );

        // Print raw instantaneous readings
        printf("%6.4lf,%6.4lf,%6.4lf,", x_inst, y_inst, z_inst);

        // Update buffers with new readings
        updatebuffer(x, lengthofavg, x_inst);
        updatebuffer(y, lengthofavg, y_inst);
        updatebuffer(z, lengthofavg, z_inst);

        if (i < lengthofavg) {
            i++;
            printf("0.0000,0.0000,0.0000,0.0000,0.0000,0.0000,0.0000,0.0000,0.0000\n");
            continue;
        }
        avg_x = avg(x, lengthofavg);
        avg_y = avg(y, lengthofavg);
        avg_z = avg(z, lengthofavg);
        printf("%6.4lf,%6.4lf,%6.4lf,", avg_x, avg_y, avg_z);

        // Max and min calculations and printouts

        // X
        maxmin(x, lengthofavg, &max_x, &min_x);
        printf("%6.4lf,%6.4lf,", max_x, min_x);

        // Y
        maxmin(y, lengthofavg, &max_y, &min_y);
        printf("%6.4lf,%6.4lf,", max_y, min_y);
```

Continued on the next page.

```
        // Z
        maxmin(z, lengthofavg, &max_z, &min_z);
        printf("%6.4lf,%6.4lf", max_z, min_z);

        // New line at end of row, then flush output
        printf("\n");
        fflush(stdout);


    }
    while (!b_sqr);
}


double avg(double buffer[], int num_items) {
    double t = 0;
    for (int i = 0; i < num_items; i++) {
        t += buffer[i];
    }

    return t / (double)num_items;
}

void maxmin(double array[], int num_items, double* max, double* min) {
    double max_temp = array[0];
    double min_temp = array[0];

    for (int i = 1; i < num_items; i++) {
        if (array[i] > max_temp) {
            max_temp = array[i];
        }

        else if (array[i] < min_temp) {
            min_temp = array[i];
        }
    }

    *max = max_temp;
    *min = min_temp;
}

void updatebuffer(double buffer[], int length, double new_item) {
    for (int i = 0; i < length - 1; i++) {
        buffer[i] = buffer[i + 1];
    }
    buffer[length - 1] = new_item;
}
```