**Moving Averages**

**LAB #7**

**SECTION 5**

**SUBMITTED BY:**

**Jackson Collalti**

**SUBMISSION DATE:**

**11/7/2022**

**Problem**

When collecting data oftentimes a smooth data set is more desirable than rough data. In this lab create a program that averages real-time data. Using data from the gyroscope values from the DS4 calculate the moving average of each axis. The program should also output the min and max values. The length of the moving average should be an input in the command line. The program will end when the square button is pressed.

**Analysis**

The program should print the min, max, and average along with the current value. The output will be for all three axes x, y, and z. The program should end when the square button is pressed. Inputs will be data from the DS4, then the output of max min and moving average. This program will be primarily based on arrays. The moving average of length n computes the average of the last n inputs. For instance, a moving average of length 2 of (1, 3, 5, 6,3) is (2, 4, 5.5, 4.5). A

moving average of length 3 of the same data would be (3, 4.666, 4.666). The n length should be imputed in the command line.

**Design**

I plan creation functions to find the max and mins, a function for average, and a function for updating the buffer. Max and mins functions will compare the current value to the previous max or min. If the current is a new max or min the pointer for max and for min will be updated. To find the average I need a for loop that adds the current buffer[i] to the current sum and returns the sum over num\_items. The update buffer will also use a for loop that will save the current buffer and then add one to the buffer. The source code given gives the function names and inputs required to complete the program.

**Testing**

I plan on testing the program initially with random movements and seeing the results. After a couple of seconds of functioning press the square button to end the program. Once I know that the program runs correctly I plan on collecting data for the following movements.

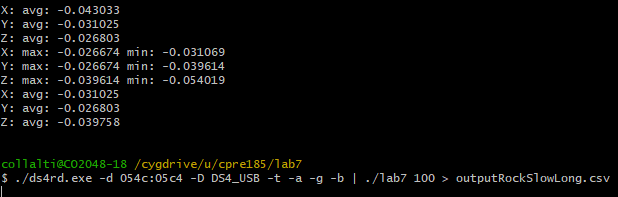
Motion 1: DualShock 4 lying on the table still. (short window length 20 and long window length 100)

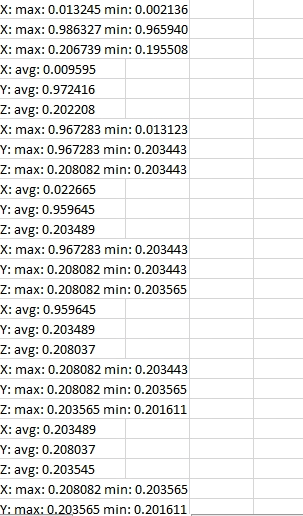
Motion 2: A very fast repetitive rocking motion (short window length 20, and long window length 100)

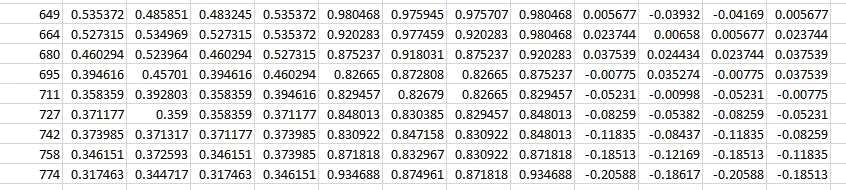
Motion 3: A slow repetitive motion of your choosing (short window length 20, and long window length 100)

**Comments**

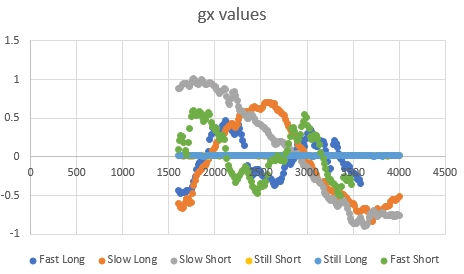
My design worked well the only large issue was with my original output format. When exporting my data I found out that the data was getting put into just one cell meaning I couldn’t easily create a graph without a tremendous amount of work.

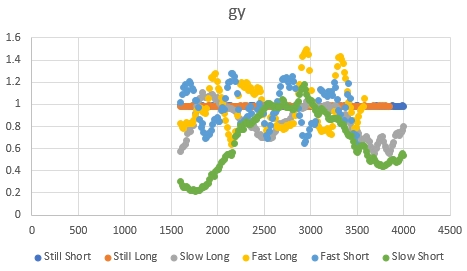


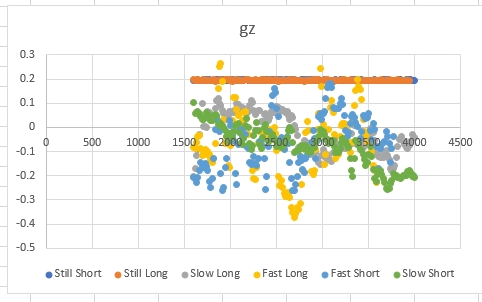


This meant I had to change my print statement so that everything would print in a single line. This wasn’t hard, but I had to rerun all my test since the data wasn’t easily usable. After some work I got my data to print on one line and compatible with excel.

1. Use a spreadsheet to graph the short and long window lengths. Label the data. You will turn in three graphs, one for each axis (X, Y, Z) with the three motions labeled (i.e. one graph will be for the X-axis and only contains the X-axis data for all three motions, one graph will be for the Y-axis and only contains the Y-axis data for all three motions, one graph will be for the Z-axis and only contains the Z-axis data for all three motions.) Compare the short window length data to the longer window length data and the raw data on each. Describe at least 2 things about the graphs that seem to happen when window lengths get longer. Why?.







In all the short vs long windows look very similar as far as smoothness. I feel gx and gy are the best for looking at trends. Gz is very messy and it's hard to see a trend in the motion. In my opinion, I found the long window to be slightly more smooth. I also noticed the gx of the Fast rocker the long window had fewer jumps in data compared to the small window.

2. What happens in comparing Motion 2’s data graph and Motion 3’s data graph? Would you be more likely to use a short window length or a long window length for each of these? Explain your reasoning

I found that the slow motions had a greater amplitude, but I attribute that to personal error. It is difficult since the data I put into the spreadsheets aren’t at the same point in the cycle. From my data, I think the long window worked the best for both motions 2 and 3 since gx and gy graphs had fewer jumps and were generally smoother.

//implementation page

// 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];

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 \*/

int t, b1, b2, b3, b4, i;

double gx, gy, gz, avgX, avgY, avgZ, x1, y1, z1, minX, maxX, minY, maxY, minZ, maxZ;

scanf("%d, %lf, %lf, %lf, %d, %d, %d, %d", &t, &gx, &gy, &gz, &b1, &b2, &b3, &b4);

for (i = 0; i < lengthofavg; i++) {

scanf("%d, %lf, %lf, %lf, %d, %d, %d, %d", &t, &gx, &gy, &gz, &b1, &b2, &b3, &b4);

updatebuffer(x, lengthofavg, gx);

updatebuffer(y, lengthofavg, gy);

updatebuffer(z, lengthofavg, gz);

}

do

{

scanf("%d, %lf, %lf, %lf, %d, %d, %d, %d", &t, &gx, &gy, &gz, &b1, &b2, &b3, &b4);

updatebuffer(x, lengthofavg, gx);

updatebuffer(y, lengthofavg, gy);

updatebuffer(z, lengthofavg, gz);

avgX = avg(x, lengthofavg);

avgY = avg(y, lengthofavg);

avgZ = avg(z, lengthofavg);

maxmin(x, lengthofavg, &maxX, &minX);

maxmin(y, lengthofavg, &maxY, &minY);

maxmin(z, lengthofavg, &maxZ, &minZ);

printf("%d, %10lf, %10lf, %10lf, %10lf, %10lf, %10lf, %10lf, %10lf, %10lf, %10lf, %10lf, %10lf\n",t, gx, avgX, minX, maxX, gy, avgY, minY, maxY, gz, avgZ, minZ, maxZ);

fflush(stdout);

} while (b4 != 1);

}

double avg(double buffer[], int num\_items) {

double sum = 0;

int i = 0;

double Avg = 0;

for (i = 0; i < num\_items; i++) {

sum += buffer[i];

}

return (sum / num\_items);

}

void maxmin(double array[], int num\_items, double\* max, double\* min) {

\*min = array[0];

\*max = array[0];

int i = 0;

for (i = 0; i < num\_items; ++i) {

if (\*min > array[i]) {

\*min = array[i];

}

if (\*max < array[i]) {

\*max = array[i];

}

}

}

//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 i;

for (i = 0; i < length; i++)

{

buffer[i + 1] = buffer[i];

}

buffer[0] = new\_item;