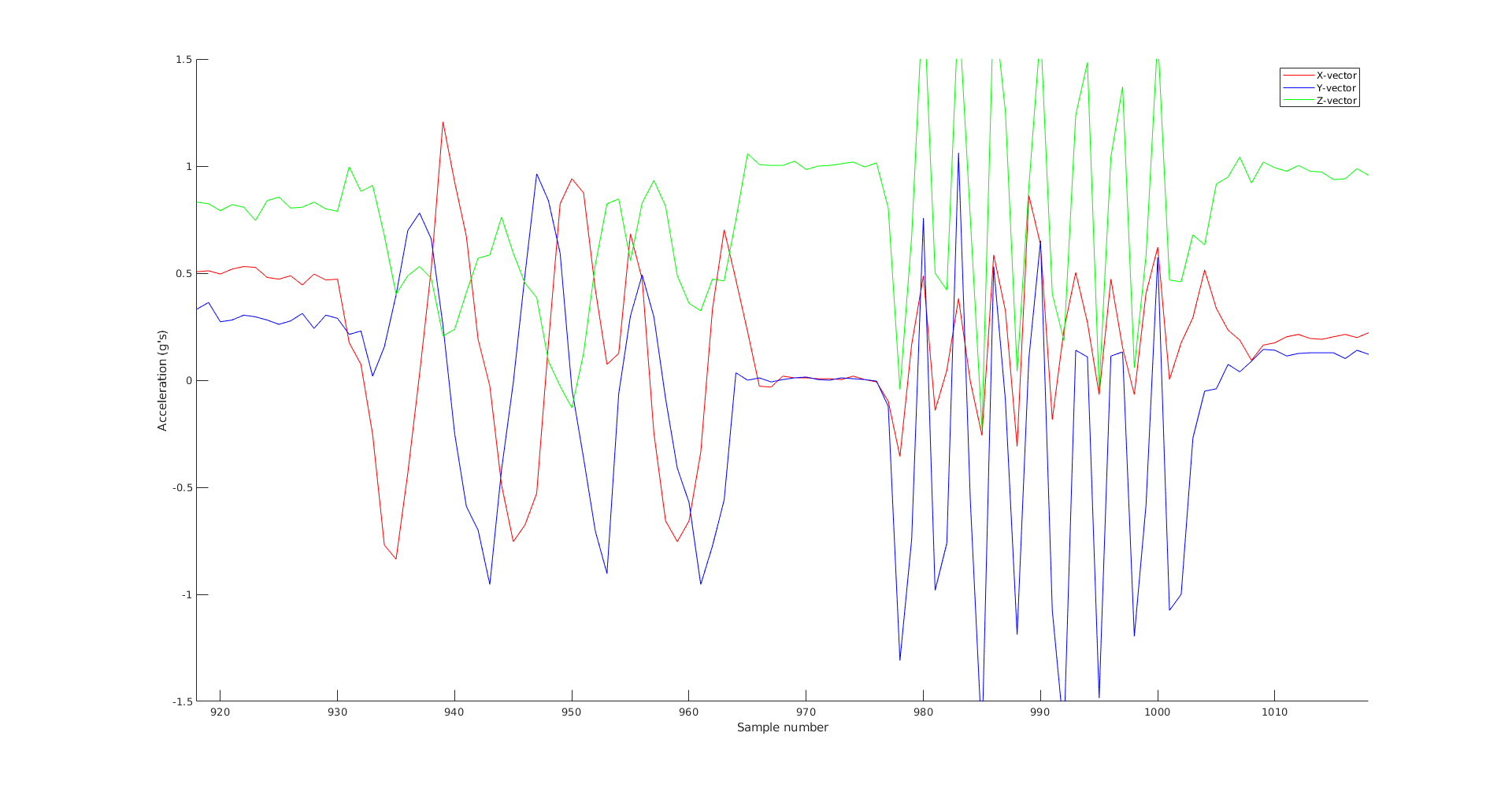
The Serial.c, Serial.h, and RTOS-specific files have been omitted due to no modification deviation from the vendor-provided code. The thdChar.c and thdJoystick.c are omitted due to the threads being unnecessary for completing this assignment.

The graph below shows the acceleration vectors for each real axis in the 3D space of real life, with the smoother oscillations being a gentle roll and rotation of the board, while the more jagged peaks represent the violent tremble of the board.



**C Code:**

//code

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 9

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

#ifndef \_\_lab\_9

#define \_\_lab\_9

#include "stdint.h"

#include "Board\_Joystick.h"

#include "Board\_GLCD.h"

#include "Board\_Accelerometer.h"

#include "rtx\_os.h"

// shared variables

extern uint32\_t treceiveChar;

// mutexes

// Note, actual variable declarations are in main.c, since exactly one file must create the space.

// The extern statements tell other files that the variable already exists, and they are allowed to access it.

extern osMutexId\_t mut1Display;

extern osMutexId\_t mut2Serial;

extern osMutexId\_t mut3Accelerometer;

// flag semaphores

//extern osSemaphoreId\_t semTick;

extern osSemaphoreId\_t semTick3\_interval;

extern osSemaphoreId\_t semTick5\_interval;

//threads

extern osThreadId\_t tid\_thdDisplay;

int Init\_thdDisplay (void);

extern osThreadId\_t tid\_thdTick;

int Init\_thdTick (void);

//extern osThreadId\_t tid\_thdChar;

//int Init\_thdChar (void);

//extern osThreadId\_t tid\_thdJoystick;

//int Init\_thdJoystick (void);

extern osThreadId\_t tid\_thdSerial;

int Init\_thdSerial(void);

//helper functions

extern void int2char(char \*, int32\_t);

#endif

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 9

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

/\*----------------------------------------------------------------------------

\* CMSIS-RTOS 'main' function

\* This is the main function for initializing the serial, display, and ticker

\* threads

\* Mutexes: display, serial, accelerometer

\* Semaphores: tick 1/3 of second, tick 1/5 of second

\*---------------------------------------------------------------------------\*/

#include "RTE\_Components.h"

#include CMSIS\_device\_header

#include "serial.h"

#include "cmsis\_os2.h"

#include "GLCD\_Config.h"

#include "stm32f2xx\_hal.h"

#include "rtx\_os.h"

#include "lab\_9.h" // specific to this project

extern GLCD\_FONT GLCD\_Font\_16x24;

// Note that the main file declares the space for all the system's variables (someone has to),

// and "extern" declarations are in the rtosClockObjects.h file so other files can find them

osMutexId\_t mut1Display;

osMutexId\_t mut2Serial;

osMutexId\_t mut3Accelerometer;

osSemaphoreId\_t semTick3\_interval;

osSemaphoreId\_t semTick5\_interval;

//osSemaphoreId\_t semChar;

uint32\_t treceiveChar;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// The RTOS and HAL need the SysTick for timing. The RTOS wins and gets control

// of SysTick, so we need to route the HAL's tick call to the RTOS's tick.

// Don't mess with this code.

uint32\_t HAL\_GetTick(void) {

return osKernelGetTickCount();

}

/\*----------------------------------------------------------------------------

\* HW Init - since the HAL depends on a periodic timer, we need the RTOS

\* in order for several HW devices to initialize correctly, like the GLCD

\*---------------------------------------------------------------------------\*/

void app\_hw\_init (void \*argument) {

GLCD\_Initialize();

GLCD\_SetBackgroundColor(GLCD\_COLOR\_PURPLE);

GLCD\_SetForegroundColor(GLCD\_COLOR\_WHITE);

GLCD\_ClearScreen();

GLCD\_SetFont(&GLCD\_Font\_16x24);

Joystick\_Initialize(); // Note: Joystick now uses HAL functions

Accelerometer\_Initialize();

SER\_Init(115200); // 115200 baud, 8 data bits, 1 stop bits, no flow control

// configures and enables the interrupt for the USART 3 serial port.

USART3->CR1 |= USART\_CR1\_RXNEIE;

NVIC->ISER[ USART3\_IRQn/32] = (1UL << (USART3\_IRQn%32));

NVIC->IP[USART3\_IRQn] = 0x80;

// Create other threads here so that all initialization is done before others get scheduled.

Init\_thdDisplay();

Init\_thdTick();

//Init\_thdChar();

//Init\_thdJoystick();

Init\_thdSerial();

osThreadExit(); // job is done, thread suicide. There better be other threads created above...

}

int main (void) {

SystemCoreClockUpdate(); // always first, make sure the clock freq. is current

osKernelInitialize(); // Initialize CMSIS-RTOS

HAL\_Init();

mut1Display = osMutexNew(NULL);

if (mut1Display==NULL) while(1){} // failed, scream and die

mut2Serial = osMutexNew(NULL);

if (mut2Serial == NULL) while(1){}

mut3Accelerometer = osMutexNew(NULL);

if (mut3Accelerometer == NULL) while(1) {}

semTick3\_interval = osSemaphoreNew(1000, 1, NULL);

if (semTick3\_interval==NULL) while(1){} // failed, scream and die

semTick5\_interval = osSemaphoreNew(1000, 1, NULL);

if (semTick5\_interval==NULL) while(1){} //AAAAAAAAAAAAAAAAAAAAAAA

//semChar = osSemaphoreNew(1000, 0, NULL);

// if (semChar==NULL) while(1){} // failed, its a comment doh!

osThreadNew(app\_hw\_init, NULL, NULL); // Create application's main thread to init HW now that HAL is running

osKernelStart(); // Start thread execution

for (;;) {} // should never get here

}

// Interrupt service routine for the serial port. It is triggered upon receiving any character

void USART3\_IRQHandler(void)

{

treceiveChar = SER\_GetChar();//value not used, but needed to prevent OS freeze upon character receive

/\*

if (treceiveChar >= 'A' && treceiveChar <= 'Z')

{

osSemaphoreRelease(semChar);

}

\*/

}

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 9

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

#include "cmsis\_os2.h"

#include "rtx\_os.h"

#include "lab\_9.h"

/\*----------------------------------------------------------------------------

\* Thread 1 'Thread\_Display': Display thread

\* Displays the current acceleration vectors on the onboard display

\* Acts on semaphore of tick interval 1/3 second

\*---------------------------------------------------------------------------\*/

void thdDisplay (void \*argument);

osThreadId\_t tid\_thdDisplay;

int Init\_thdDisplay (void) {

tid\_thdDisplay = osThreadNew (thdDisplay, NULL, NULL);

if (!tid\_thdDisplay) return(-1);

return(0);

}

void int2char(char [], int32\_t);

void thdDisplay (void \*argument)

{

ACCELEROMETER\_STATE accel;

char accelx[6] = " ";

char accely[6] = " ";

char accelz[6] = " ";

GLCD\_DrawChar(20, 10, 'x');

GLCD\_DrawChar(20, 40, 'y');

GLCD\_DrawChar(20, 70, 'z');

for(uint32\_t i = 10; i <= 70; i += 30)

{

GLCD\_DrawChar(40, i, ':');

GLCD\_DrawChar(120, i, '.');

GLCD\_DrawChar(200, i, 'g');

}

//TODO draw all axes to onboard display

while (1)

{

osSemaphoreAcquire(semTick3\_interval, osWaitForever);

osMutexAcquire(mut3Accelerometer,osWaitForever);

Accelerometer\_GetState(&accel);

osMutexRelease(mut3Accelerometer);

int2char(accelx, accel.x);

int2char(accely, accel.y);

int2char(accelz, accel.z);

osMutexAcquire(mut1Display,osWaitForever);

for(uint32\_t i = 0; i < 6; i++)

{

if(i < 3)

GLCD\_DrawChar(180 - i \* 20, 10, accelx[i]);

else

GLCD\_DrawChar(100 - (i-3) \* 20, 10, accelx[i]);

}

/\*

GLCD\_DrawChar(60, 10, accelx[5]);

GLCD\_DrawChar(80, 10, accelx[4]);

GLCD\_DrawChar(100, 10, accelx[3]);

GLCD\_DrawChar(140, 10, accelx[2]);

GLCD\_DrawChar(160, 10, accelx[1]);

GLCD\_DrawChar(180, 10, accelx[0]);

\*/

for(uint32\_t i = 0; i < 6; i++)

{

if(i < 3)

GLCD\_DrawChar(180 - i \* 20, 40, accely[i]);

else

GLCD\_DrawChar(100 - (i-3) \* 20, 40, accely[i]);

}

for(uint32\_t i = 0; i < 6; i++)

{

if(i < 3)

GLCD\_DrawChar(180 - i \* 20, 70, accelz[i]);

else

GLCD\_DrawChar(100 - (i-3) \* 20, 70, accelz[i]);

}

osMutexRelease(mut1Display);

}

}

void int2char(char text[], int32\_t data)

{

uint32\_t i;

if (data < 0)

{

text[5] = '-';

data = -data;

}

else

{

text[5] = '+';

}

for (i = 0; i<5; i++)

{

text[i] = (data % 10)+0x30;

data /= 10;

}

}

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 9

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

#include "cmsis\_os2.h"

#include "lab\_9.h"

#define UPDATEFREQ 15 //lcm of 3 and 5

/\*----------------------------------------------------------------------------

\* Thread 2 'Thread\_ticker': Tick generator thread

\* This thread generates a combined tick and flags a semaphore based on the

\* multiple of the master frequency

\*---------------------------------------------------------------------------\*/

void thdTick (void \*argument); // thread function

osThreadId\_t tid\_thdTick; // thread id

int Init\_thdTick (void) {

tid\_thdTick = osThreadNew (thdTick, NULL, NULL);

if (!tid\_thdTick) return(-1);

return(0);

}

//The embedded version of fizzbuzz

void thdTick (void \*argument) {

uint32\_t timerCount = 0;

while (1)

{

osDelay(osKernelGetTickFreq()/UPDATEFREQ);

if(timerCount % 3 == 0)

osSemaphoreRelease(semTick5\_interval);

if(timerCount % 5 == 0)

osSemaphoreRelease(semTick3\_interval);

if(timerCount == UPDATEFREQ)

timerCount = 0;

timerCount++;

}

}

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 9

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

#include "lab\_9.h"

#include "serial.h"

/\*----------------------------------------------------------------------------

\* Thread 3 'Thread\_Serial': Display thread

\* This thread outputs to the serial port in format x,y,z0

\* (no spaces, comma-separated, null (\0) terminated)

\* Acts on semaphore of tick interval 1/5 second

\*---------------------------------------------------------------------------\*/

void thdSerial(void \*argument);

osThreadId\_t tid\_thdSerial;

//uint32\_t sampleCount;

int Init\_thdSerial (void) {

tid\_thdSerial = osThreadNew (thdSerial, NULL, NULL);

if (!tid\_thdSerial) return(-1);

//sampleCount = 1;

return(0);

}

void thdSerial(void \*argument)

{

//declare stuff

ACCELEROMETER\_STATE accel;

char accelx[6] = " ";

char accely[6] = " ";

char accelz[6] = " ";

while(1)

{

osSemaphoreAcquire(semTick5\_interval, osWaitForever);

osMutexAcquire(mut3Accelerometer,osWaitForever);

Accelerometer\_GetState(&accel);

osMutexRelease(mut3Accelerometer);

int2char(accelx, accel.x);

int2char(accely, accel.y);

int2char(accelz, accel.z);

osMutexAcquire(mut2Serial,osWaitForever);

//SER\_PutChar(sampleCount + 0x30);

//SER\_PutChar(',');

for(uint32\_t i = 0; i < 3; i++)

{

SER\_PutChar(accelx[5 - i]);

}

SER\_PutChar('.');

for(uint32\_t i = 3; i < 6; i++)

{

SER\_PutChar(accelx[5 - i]);

}

SER\_PutChar(',');

for(uint32\_t i = 0; i < 3; i++)

{

SER\_PutChar(accely[5 - i]);

}

SER\_PutChar('.');

for(uint32\_t i = 3; i < 6; i++)

{

SER\_PutChar(accely[5 - i]);

}

SER\_PutChar(',');

for(uint32\_t i = 0; i < 3; i++)

{

SER\_PutChar(accelz[5 - i]);

}

SER\_PutChar('.');

for(uint32\_t i = 3; i < 6; i++)

{

SER\_PutChar(accelz[5 - i]);

}

//if needed, add a line ending sequence of your choice here

//SER\_PutChar('\r');

SER\_PutChar('\0');

osMutexRelease(mut2Serial);

}

}

**MATLAB Code:**

%Modified from assignment provided sources:

%Author of modifications: Peter A. Dranishnikov

%Lab #: 9

%Course: EEL4685C

%Due date: April 23, Spring 2019

% SPEC:

% MATLAB must graph the X, Y, and Z components on a graph that updates in

% real time and displays (up to) the last 100 samples.

% Hint: serial ISR can grow an array unitl it reaches 100 samples, then it

% can use last 99 samples and newest sample for the next one.

% The x-axis must be labeled with the number of samples

% (first\_sample-last\_sample)

clear all;

% You will need to change the number of the COM port to match the

% USB-to-Serial cable

%comport = 'COM4';

comport = '/dev/ttyUSB0'; %linux version

instrreset;

figure;

%graph setup

% This \*should\* set up the COM port the same way it was in PuTTY.

sp = serial(comport);

% After the serial port object is created, you can click on it in the

% Workspace window, and MATLAB will show a configuration screen. This is a

% good way to see all the fields that can be modified.

sp.baudrate= 115200;

sp.databits=8;

sp.FlowControl = 'none';

sp.StopBits = 1.0;

sp.ReadAsyncMode = 'continuous';

sp.BytesAvailableFcn = @readport; % This sets up an interrupt in MATLAB for receiving bytes on the serial port.

% The @ symbol states that you are supply the name of the function.

sp.BytesAvailableFcnCount = 24;

sp.BytesAvailableFcnMode = 'byte';

fopen(sp);

char = '0';

% This is the main loop that repeats "forever"...or until the serial port

% dies.

while (sp.Status == 'open')

fprintf(sp,char);

if (char == '9')

char = '0';

else

char = char+1;

end

pause(1);

end

% MATLAB allows multiple interrupts to call the same function. Normally,

% there are no arguments to an interrupt service routine, but since MATLAB

% is running on top of the OS and doesn't have direct access to the

% hardware, it cheats a little by passing a reference to the object that

% generated the interrupt and the type of interrupt that was triggered.

% Therefore, in the ISR below, "port" is a reference to the serial port

% object sp above. Note that this also allows the same ISR to be used for

% multiple devices (i.e. different serial ports could all point to this

% ISR.)

function readport(port,b)

% for a serial port, fread requires the handle to the serial port and the

% number of bytes to read. The char() function converts the numerical bytes

% to integers.

persistent da\_plotx;

persistent da\_ploty;

persistent da\_plotz;

persistent n\_arr;

if(isempty(n\_arr)|n\_arr < 1)

da\_plotx = animatedline('Color', 'red');

da\_ploty = animatedline('Color', 'blue');

da\_plotz = animatedline('Color', 'green');

legend({'X-vector','Y-vector','Z-vector'});

axis([0 100 -1.5 1.5])

xlabel("Sample number");

ylabel("Acceleration (g's)");

end

data = char(fread(port,24));

x\_val = str2double(data(1:7));

y\_val = str2double(data(9:15));

z\_val = str2double(data(17:23));

if(isempty(n\_arr) | isnan(n\_arr))

n\_arr = [1];

else

n\_arr = n\_arr + 1;

end

addpoints(da\_plotx, n\_arr, x\_val);

addpoints(da\_ploty, n\_arr, y\_val);

da\_plotz.addpoints(n\_arr, z\_val);

if(n\_arr > 100)

axis([(n\_arr-100) n\_arr -1.5 1.5]);

end

drawnow();

% Comment out the line above and enable the line below to see the difference that char() makes.

%data = dec2hex(fread(port,3))

end