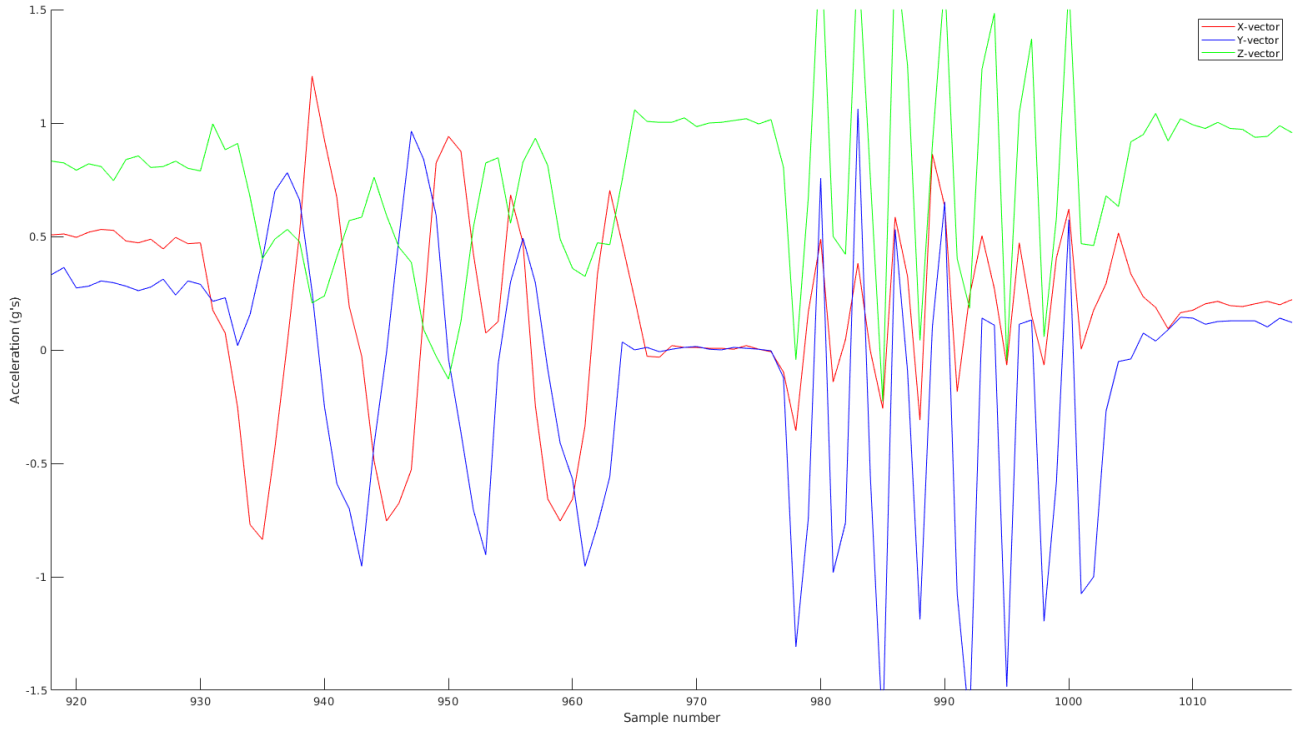
The Serial.c, Serial.h, and RTOS-specific files have been omitted due to no modification deviation from the vendor-provided code. The thdChar.c and thdJoystick.c are omitted due to the threads being unnecessary for completing this assignment.

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){}
//AAAAAAAAAAAAAAAAAAAAAA
     //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
```