Bryan Takemoto
Github root directory: github.com/AkagiCoder/AdvEmbeddedDA

Date Submitted: 11/3

Main CODE:

Youtube Link: https://youtu.be/nueKNUDeisk

```
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <math.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_i2c.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/adc.h"
#include "driverlib/i2c.h"
#include "driverlib/fpu.h"
#include "utils/uartstdio.h"

// Constants
#define MPU_ADDR    0x68    // Address of MPU6050
#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 65.536
#define dt 0.01 // 10 ms sample rate!

// Function prototypes
void MPUInit();
void ftoa(float f,char *buf);
uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg);
void ComplementaryFilter(short accData[3], short gyrData[3], float *pitch, float *roll);

// Raw data from the accelerometer and gyroscope
short accelData[3];
short gyroData[3];

// Floating-point data
volatile float accel[3];
volatile float gyro[3];
float pitch;
float roll;

// Buffer to hold strings of floating-point values
char bufferPitch[10];
char bufferRoll[10];
char bufferAX[10];
char bufferAY[10];
char bufferAZ[10];
char bufferGX[10];
char bufferGY[10];
char bufferGZ[10];

int main(void)
{
    // System clock at 50 MHz
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

    // Set up GPIOA for UART
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    // Enable UART0 so that we can configure the clock.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    // Use the internal 16MHz oscillator as the UART clock source.
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    // Select the alternate (UART) function for these pins.
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
// Initialize the UART for console I/O.
UARTStdioConfig(0, 115200, 16000000);

// Set up I2C
SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);     // Enable I2C hardware
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);    // Enable Pin hardware
GPIOPinConfigure(GPIO_PB3_I2C0SDA);             // Configure GPIO pin for I2C Data line
GPIOPinConfigure(GPIO_PB2_I2C0SCL);             // Configure GPIO Pin for I2C clock line
GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3);  // Set Pin Type
// SDA MUST BE STD
GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD);
// SCL MUST BE OPEN DRAIN
GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_OD);
// The False sets the controller to 100kHz communication
I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);

//clear I2C FIFOs
HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;

// Set up FPU
FPULazyStackingEnable();    // Enables lazy stacking of floating-point registers
FPUEnable();                // Enables the FPU since it's disabled by default

MPUInit();

while(1)
{
    // X-Axis Accel
    accelData[0] = (I2CReceive(MPU_ADDR, 0x3B)) << 8;  // Upper-byte of data
    accelData[0] |= I2CReceive(MPU_ADDR, 0x3C);        // Lower-byte of data
    accel[0] = accelData[0] * 0.00059875;              // Converts raw to floating-point
    ftoa(accel[0], bufferAX);                          // Converts floating-point into string
    // Y-Axis Accel
    accelData[1] = (I2CReceive(MPU_ADDR, 0x3D)) << 8;
    accelData[1] |= I2CReceive(MPU_ADDR, 0x3E);
    accel[1] = accelData[1] * 0.00059875;
    ftoa(accel[1], bufferAY);
    // Z-Axis Accel
    accelData[2] = (I2CReceive(MPU_ADDR, 0x3F)) << 8;
    accelData[2] |= I2CReceive(MPU_ADDR, 0x40);
    accel[2] = accelData[2] * 0.00059875;
    ftoa(accel[2], bufferAZ);
    // X-Axis Gyro
    gyroData[0] = (I2CReceive(MPU_ADDR, 0x43)) << 8;
    gyroData[0] |= I2CReceive(MPU_ADDR, 0x44);
    gyro[0] = gyroData[0] * 0.00059875;
    ftoa(gyro[0], bufferGX);
    // Y-Axis Gyro
    gyroData[1] = (I2CReceive(MPU_ADDR, 0x45)) << 8;
    gyroData[1] |= I2CReceive(MPU_ADDR, 0x46);
    gyro[1] = gyroData[1] * 0.00059875;
    ftoa(gyro[1], bufferGY);
    // Z-Axis Gyro
    gyroData[2] = (I2CReceive(MPU_ADDR, 0x47)) << 8;
    gyroData[2] |= I2CReceive(MPU_ADDR, 0x48);
    gyro[2] = gyroData[2] * 0.00059875;
    ftoa(gyro[2], bufferGZ);
    // Input the raw data into the complementary filter
    ComplementaryFilter(accelData, gyroData, &pitch, &roll);
    ftoa(pitch, bufferPitch);
    ftoa(roll, bufferRoll);
    // Print values to the terminal
    UARTprintf("\033[2J");
    UARTprintf("Pitch: %s\n", bufferPitch);
    UARTprintf("Roll: %s\n", bufferRoll);
    UARTprintf("Accelerometer:\nX = %s\nY = %s\nZ = %s\n", bufferAX, bufferAY, bufferAZ);
    UARTprintf("Gyroscope:\nX = %s\nY = %s\nZ = %s\n\n", bufferGX, bufferGY, bufferGZ);
    SysCtlDelay(500000);    // Delay to prevent the terminal from being flooded
```

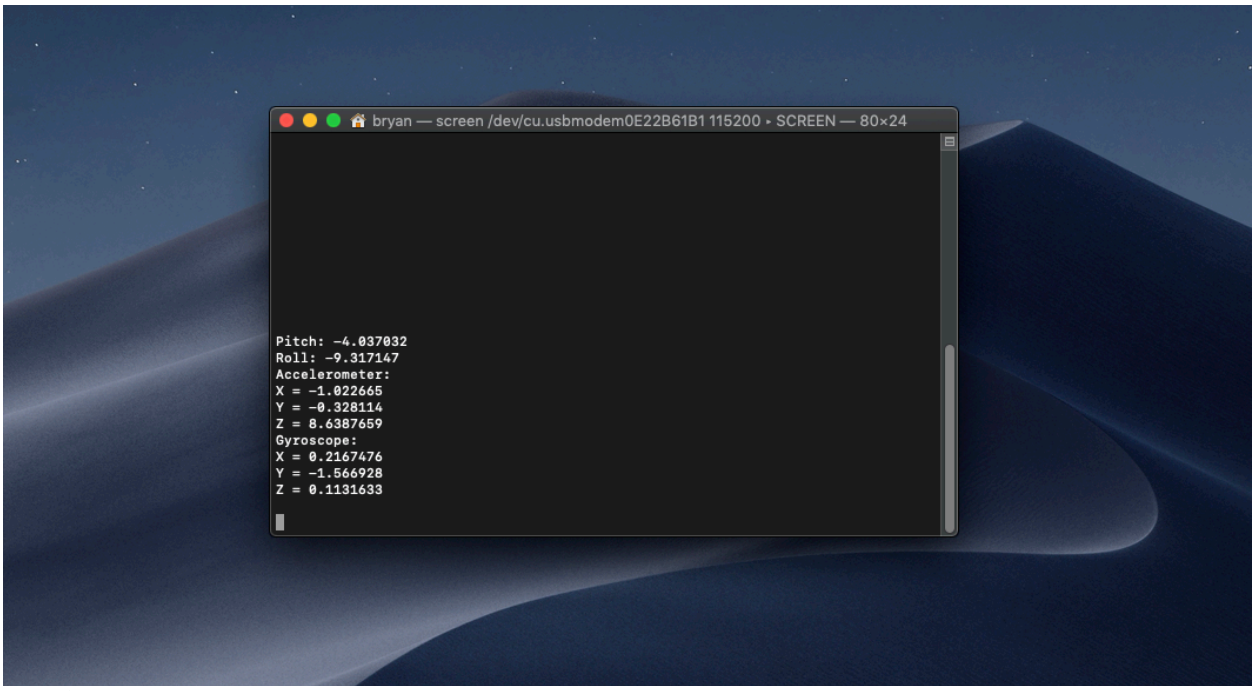
Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    }
    return 0;
}

// Function to initiate the MPU6050
void MPUInit()
{
    I2CMasterSlaveAddrSet(I2C0_BASE, MPU_ADDR, false);                  // False means transmit
    I2CMasterDataPut(I2C0_BASE, 0x6B);                                  // PWR_MGMT_1 reg
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);       // Start condition
    while(I2CMasterBusy(I2C0_BASE));
    I2CMasterDataPut(I2C0_BASE, 0x00);                                  // Wake up the MPU
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
    while(I2CMasterBusy(I2C0_BASE));
}

// I2C function to receive data from the MPU6050
uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg)
{
    //specify that we are writing (a register address) to the
    //slave device
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);

    //specify register to be read
    I2CMasterDataPut(I2C0_BASE, reg);

    //send control byte and register address byte to slave device
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);

    //wait for MCU to finish transaction
    while(I2CMasterBusy(I2C0_BASE));

    //specify that we are going to read from slave device
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, true);

    //send control byte and read from the register we
    //specified
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);

    //wait for MCU to finish transaction
    while(I2CMasterBusy(I2C0_BASE));

    //return data pulled from the specified register
    return I2CMasterDataGet(I2C0_BASE);
}

// Function to convert floating-point to string. This allows the floating-point
// to be printed in the terminal using UARTprintf (by default, does not support
// floating-point).
void ftoa(float f,char *buf)
{
    int pos=0,ix,dp,num;
    if (f<0)
    {
        buf[pos++]='-';
        f = -f;
    }
    dp=0;
    while (f>=10.0)
    {
        f=f/10.0;
        dp++;
    }
    for (ix=1;ix<8;ix++)
    {
        num = (int)f;
        f=f-num;
        if (num>9)
            buf[pos++]='#';
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
        else
            buf[pos++]='0'+num;
        if (dp==0) buf[pos++]='.';
        f=f*10.0;
        dp--;
    }
}

// Complementary filter function to obtain pitch and roll.
void ComplementaryFilter(short accData[3], short gyrData[3], float *pitch, float *roll)
{
    float pitchAcc, rollAcc;
    // Integrate the gyroscope data -> int(angularSpeed) = angle
    // Angle around the X-axis
    *pitch += ((float)gyrData[0] / GYROSCOPE_SENSITIVITY) * dt;
    // Angle around the Y-axis
    *roll -= ((float)gyrData[1] / GYROSCOPE_SENSITIVITY) * dt;
    // Compensate for drift with accelerometer data
    // Sensitivity = -2 to 2 G at 16Bit -> 2G = 32768 && 0.5G = 8192
    int forceMagnitudeApprox = abs(accData[0]) + abs(accData[1]) + abs(accData[2]);
    if (forceMagnitudeApprox > 8192 && forceMagnitudeApprox < 32768)
    {
        // Turning around the X axis results in a vector on the Y-axis
        pitchAcc = atan2f((float)accData[1], (float)accData[2]) * 180 / M_PI;
        *pitch = *pitch * 0.98 + pitchAcc * 0.02;
        // Turning around the Y axis results in a vector on the X-axis
        rollAcc = atan2f((float)accData[0], (float)accData[2]) * 180 / M_PI;
        *roll = *roll * 0.98 + rollAcc * 0.02;
    }
}
```

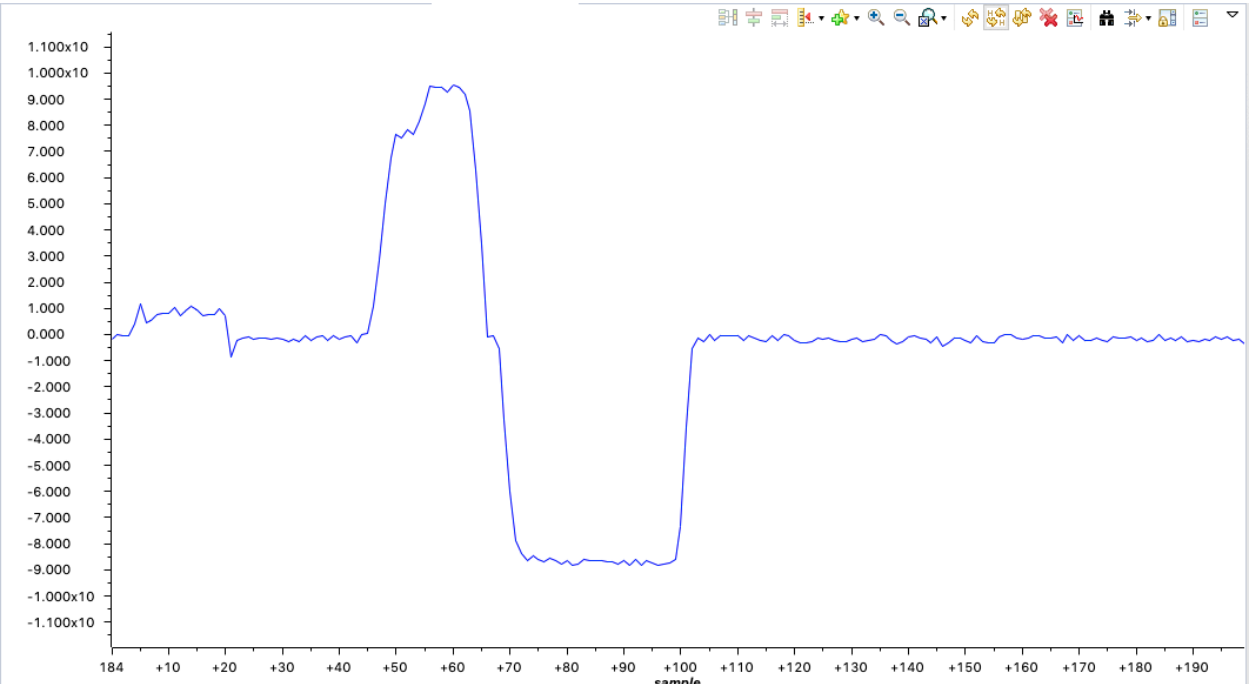
Task Terminal & Graph:



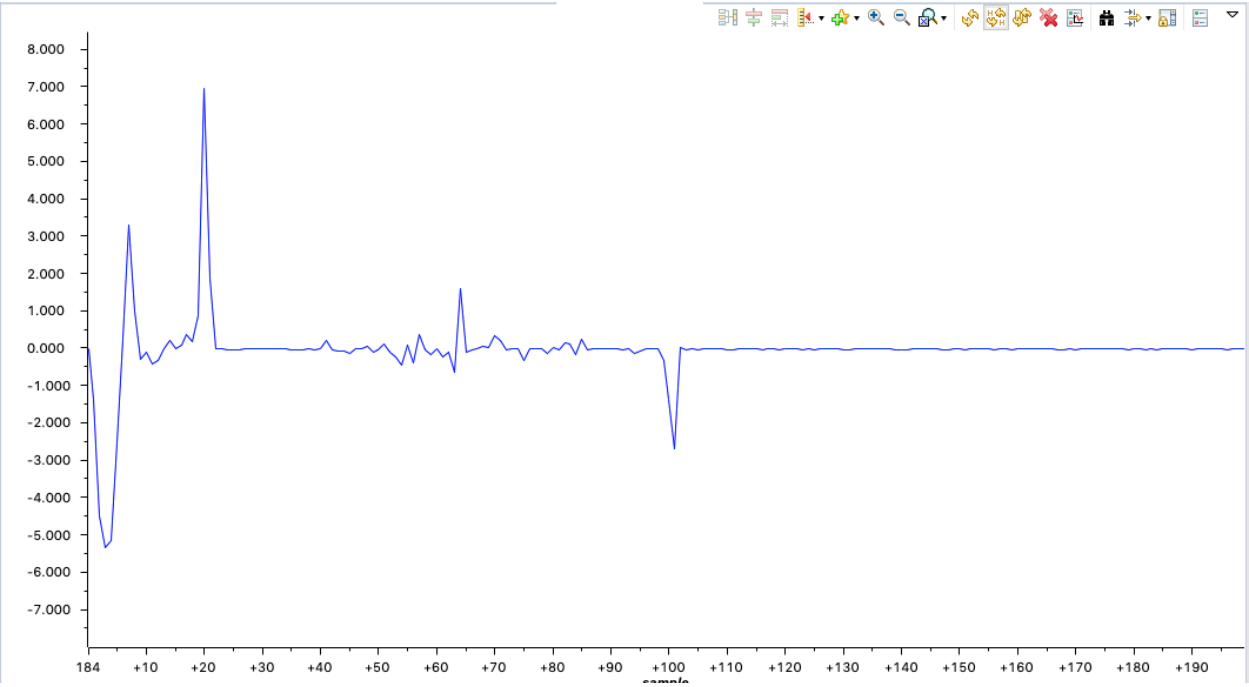
Terminal

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Bryan Takemoto
Github root directory: github.com/AkagiCoder/AdvEmbeddedDA



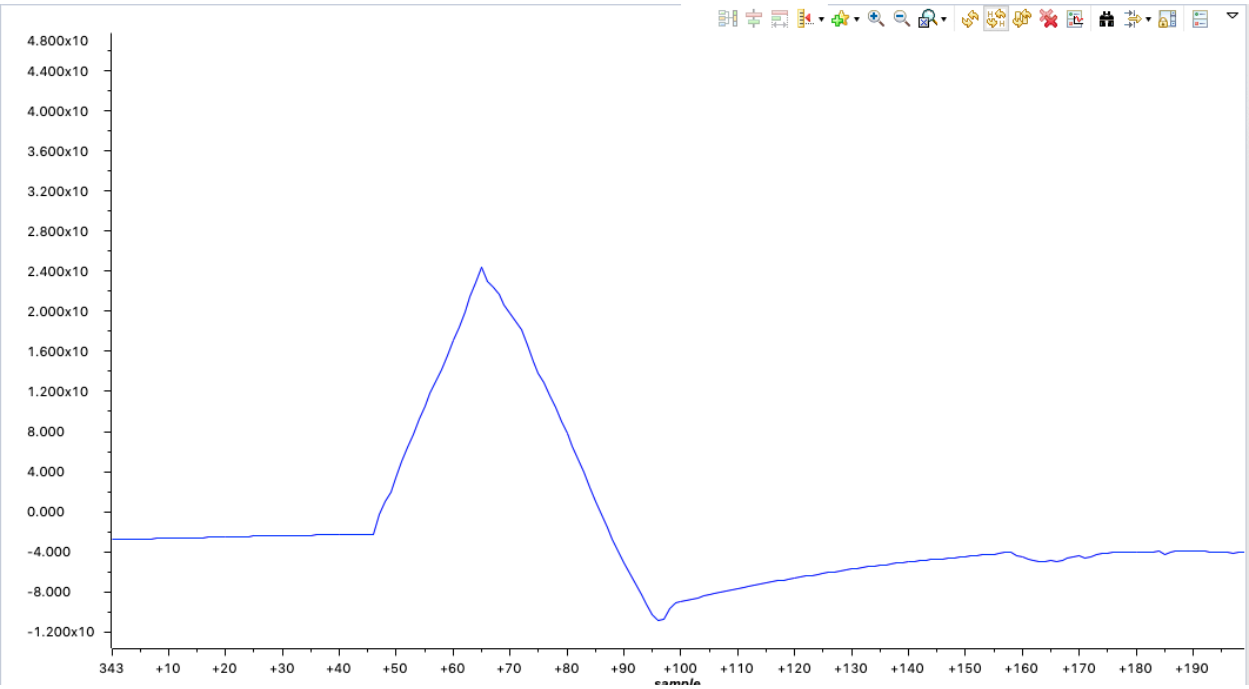
Example graph of accelerometer on the x-axis.



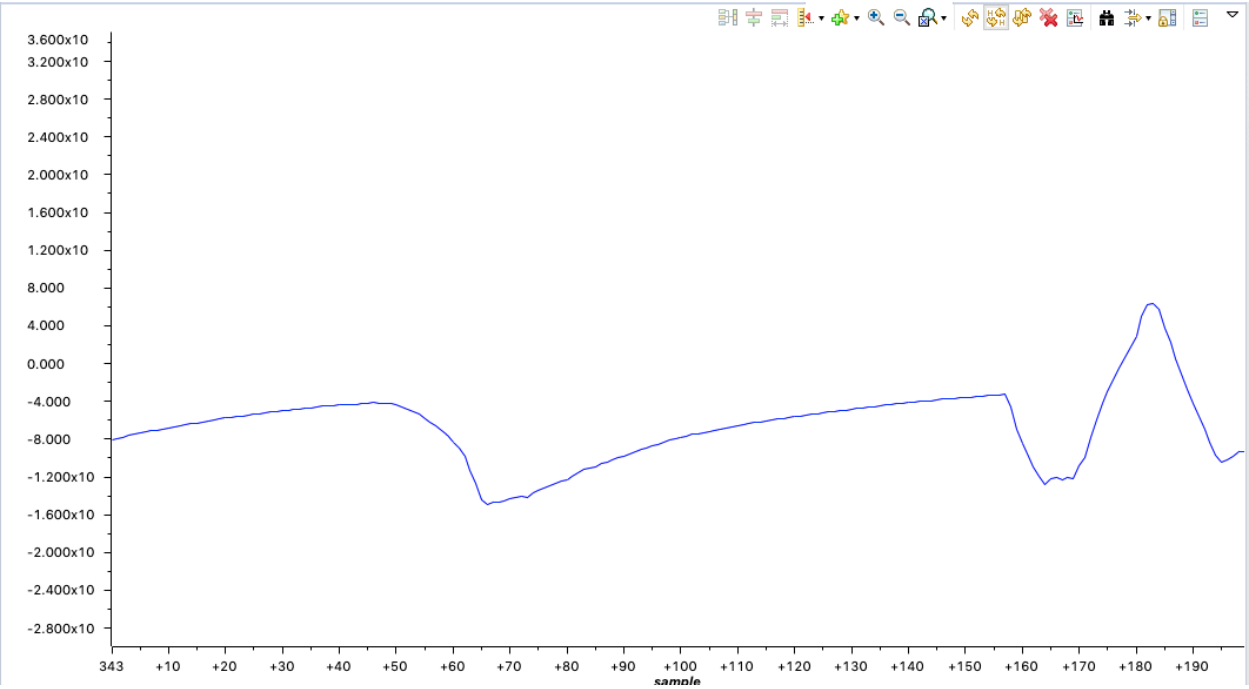
Graph of gyroscope on the x-axis.

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Bryan Takemoto
Github root directory: github.com/AkagiCoder/AdvEmbeddedDA



Graph of the pitch



Graph of the roll.

My IQMath was not working correctly on the TIVA C.

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.