Name: Itzel Becerril

CPE 403 ADV EMB SYS DES F 2019

Repository: https://github.com/HadidBuilds/TivaC_project_labs

TITLE: TIVAC MIDTERM

GOAL:

- The goal of this midterm is to learn how to use a MPU6050 using I2C Protocol to TIVAC.
- The MPU6050 is used to get accelerometer and gyro values.
- Initially get raw values but then implement the complementary filter to give me more accurate values such as the roll and pitch.
- Use IQMath Library
- Each task is either printed on serial terminal or graphed.

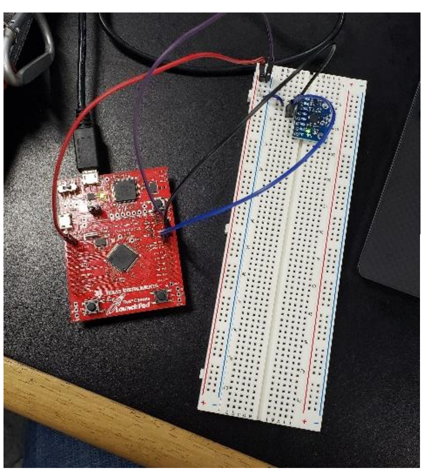
DELIVERABLES:

Task 1 and 3 are heavy in code. It completes the task such as displaying raw values and then the other code incorporates the complementary filter. Will deliver codes, graphs, YouTube video links, screenshots, and images. Will also be added to my GitHub repository.

COMPONENTS:

- Code Composer Studio – used to program code to display values on terminal and graph
- TIVAC- microcontroller/ which was connected to other components using i2c
- I2C- used to communicate between TivaC and MPU6050
- Male to male wires – used for connections
- MPU6050- used to get accelerometer and gyro values
- Micro USB- connection for microcontroller and laptop

SCHEMATICS:



J2 Pin	GPIO	Analog Function / GPIO AMSEL	On-board Function	Tiva C Series MCU Pin	GPIOPCTL Register Setting 1	2	3	4	5	6	7
2.01	GND										
2.02	PB2	–	–	47	–	–	I2C0SCL	–	–	–	T3CCP0
2.03	PE0	AIN3	–	9	U7Rx	–	–	–	–	–	–
			(R12)								
1.03	PB3	–	–	48	–	–	I2C0SDA	–	–		
1.04	PC4	C1–	–	16	U4Rx	U1Rx	–	M0PWM6	–		

PB2 –I2C0SCL
PB3 –I2C0SDA

SCREENSHOTS:

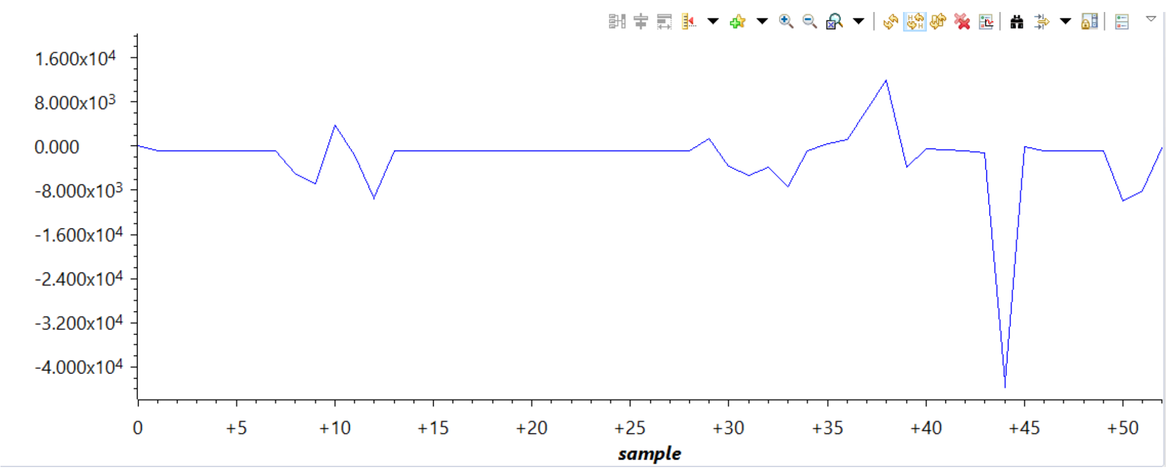
Task 01:

Output of the raw values

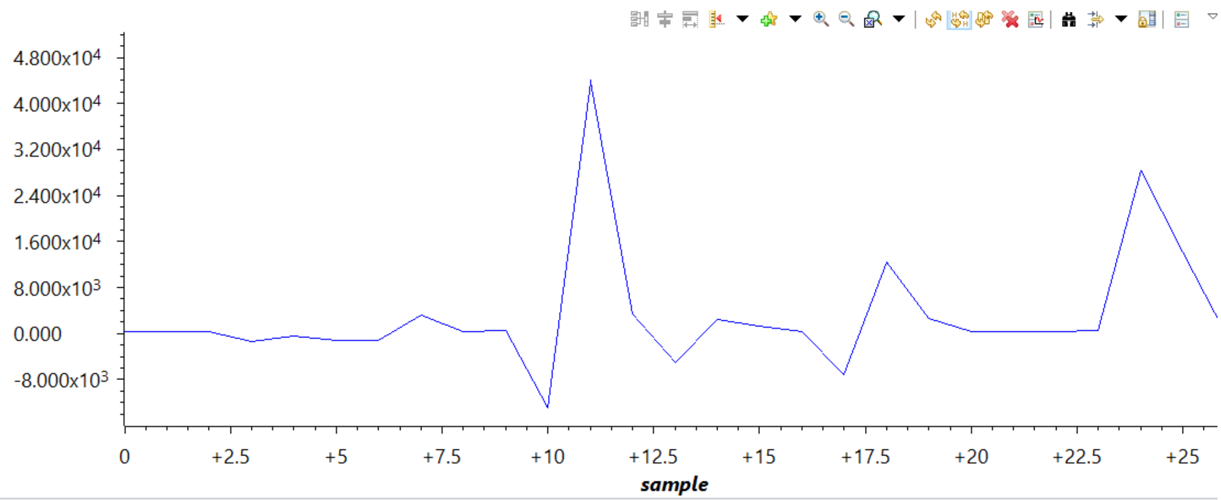
```
GYRO. XX: 1141 | GYRO. YY: 6500 | GYRO. ZZ: -182
ACC.  X: 0  | ACC. Y: 14   | ACC. Z: 44
GYRO. XX: -2543 | GYRO. YY: -14671 | GYRO. ZZ: 327
ACC.  X: 31  | ACC. Y: 3   | ACC. Z: 44
GYRO. XX: 2583 | GYRO. YY: 34113 | GYRO. ZZ: 2884
ACC.  X: -8  | ACC. Y: -20   | ACC. Z: 43
GYRO. XX: -4452 | GYRO. YY: -1004 | GYRO. ZZ: -2466
ACC.  X: -4  | ACC. Y: -14   | ACC. Z: 44
GYRO. XX: 8875 | GYRO. YY: -7991 | GYRO. ZZ: 282
ACC.  X: -3  | ACC. Y: 4   | ACC. Z: 44
GYRO. XX: -774 | GYRO. YY: -77 | GYRO. ZZ: -361
ACC.  X: 0  | ACC. Y: 0   | ACC. Z: 45
GYRO. XX: -1244 | GYRO. YY: 500 | GYRO. ZZ: -187
ACC.  X: 0  | ACC. Y: 0   | ACC. Z: 45
GYRO. XX: -803 | GYRO. YY: 79 | GYRO. ZZ: -171
ACC.  X: -20  | ACC. Y: -6   | ACC. Z: 44
GYRO. XX: -5467 | GYRO. YY: 4503 | GYRO. ZZ: 964
ACC.  X: -2  | ACC. Y: -1   | ACC. Z: 45
GYRO. XX: -893 | GYRO. YY: 229 | GYRO. ZZ: -106
ACC.  X: -2  | ACC. Y: 0   | ACC. Z: 45
GYRO. XX: -842 | GYRO. YY: 378 | GYRO. ZZ: -143
```

Task 02:

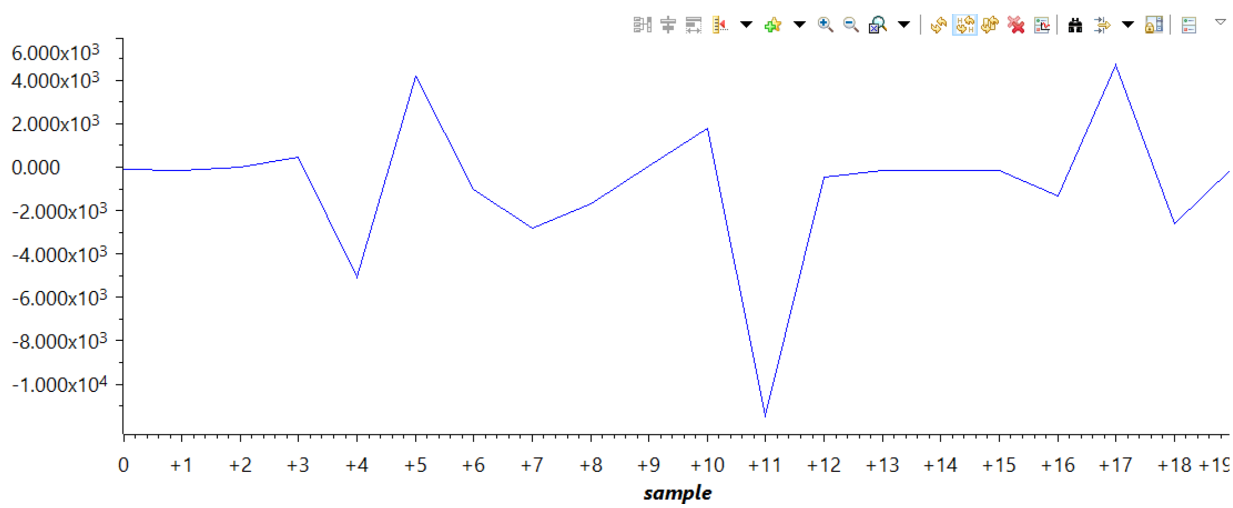
Gyro-X:



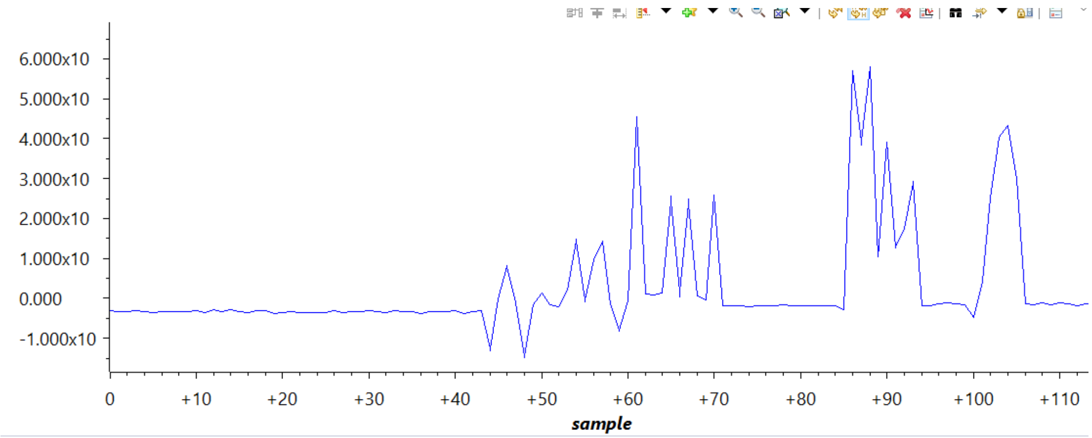
Gyro-Y:


4.800x10^4
4.000x10^4
3.200x10^4
2.400x10^4
1.600x10^4
8.000x10^3
0.000
-8.000x10^3
0
+2.5
+5
+7.5
+10
+12.5
+15
+17.5
+20
+22.5
+25
sample

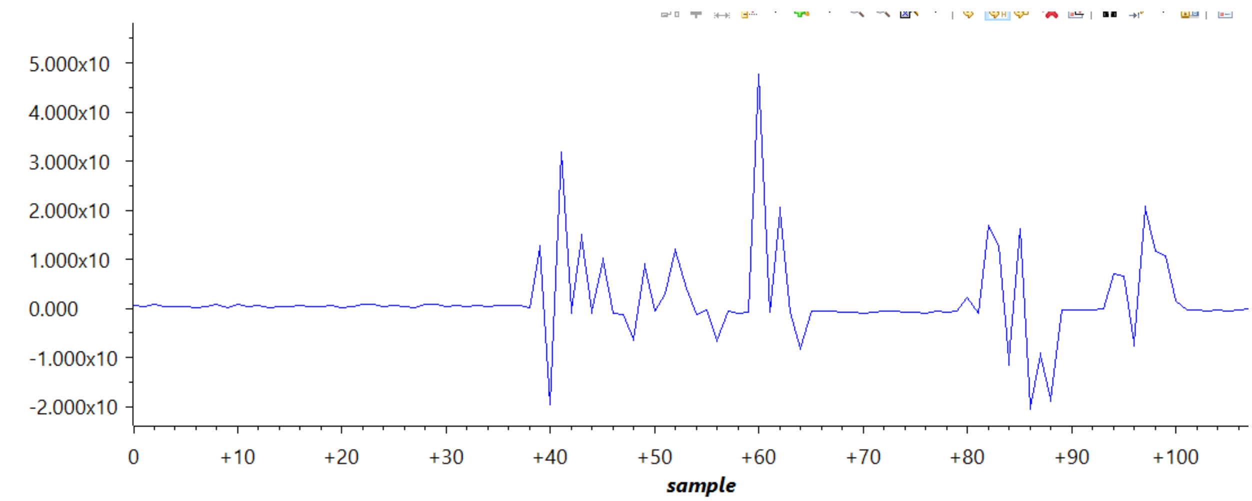
Gyro-Z:


6.000x10^3
4.000x10^3
2.000x10^3
0.000
-2.000x10^3
-4.000x10^3
-6.000x10^3
-8.000x10^3
-1.000x10^4
0
+1
+2
+3
+4
+5
+6
+7
+8
+9
+10
+11
+12
+13
+14
+15
+16
+17
+18
sample

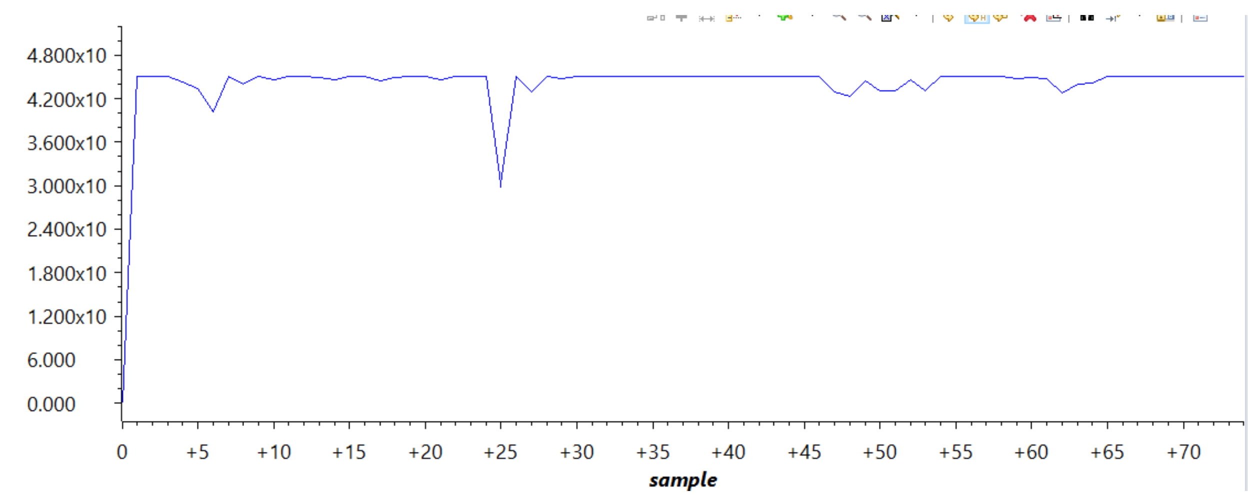
Acc-X:



Acc-Y:

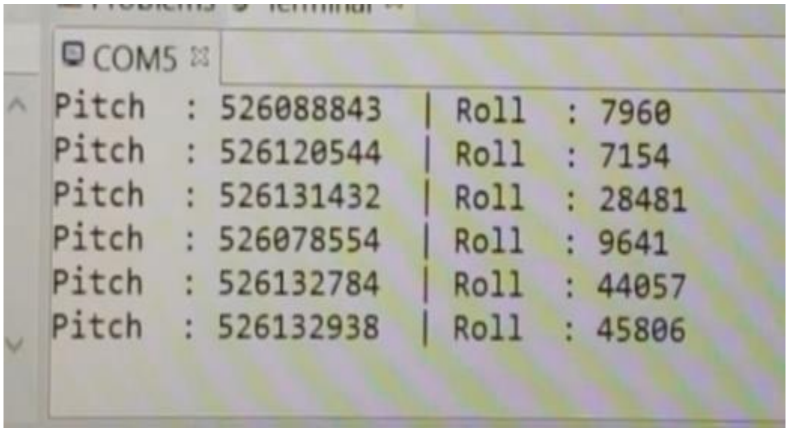


Acc-Z:



Task 03:

Pitch and Roll values printed on the serial monitor

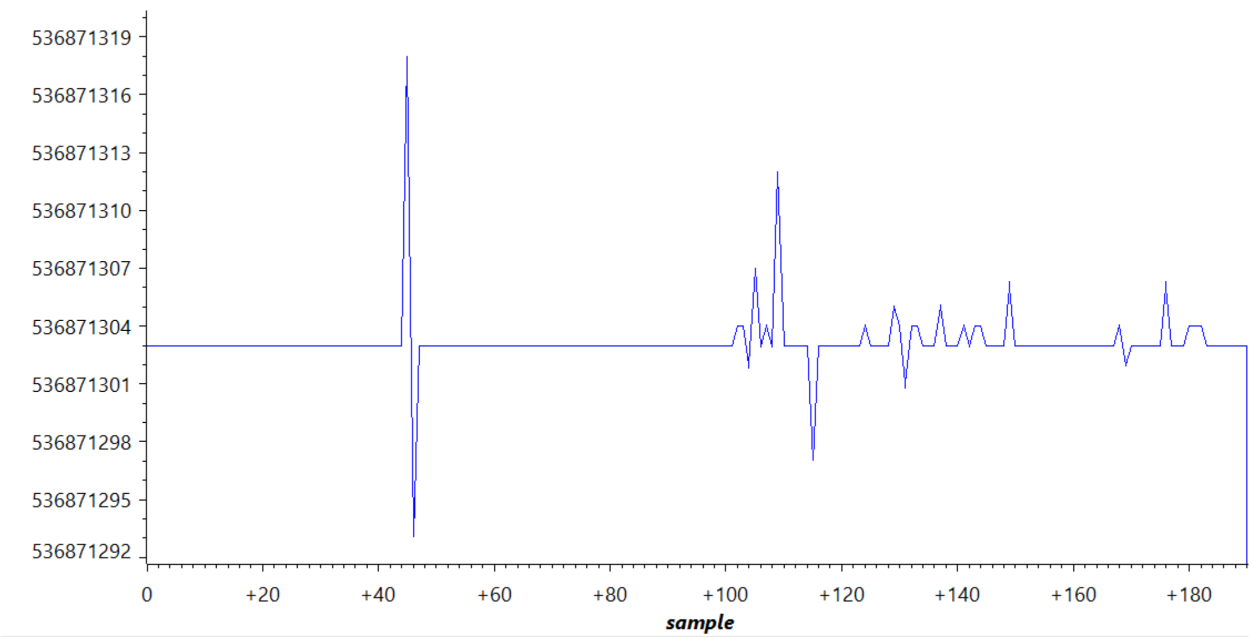

COM5

Pitch : 526088843 | Roll : 7960
Pitch : 526120544 | Roll : 7154
Pitch : 526131432 | Roll : 28481
Pitch : 526078554 | Roll : 9641
Pitch : 526132784 | Roll : 44057
Pitch : 526132938 | Roll : 45806

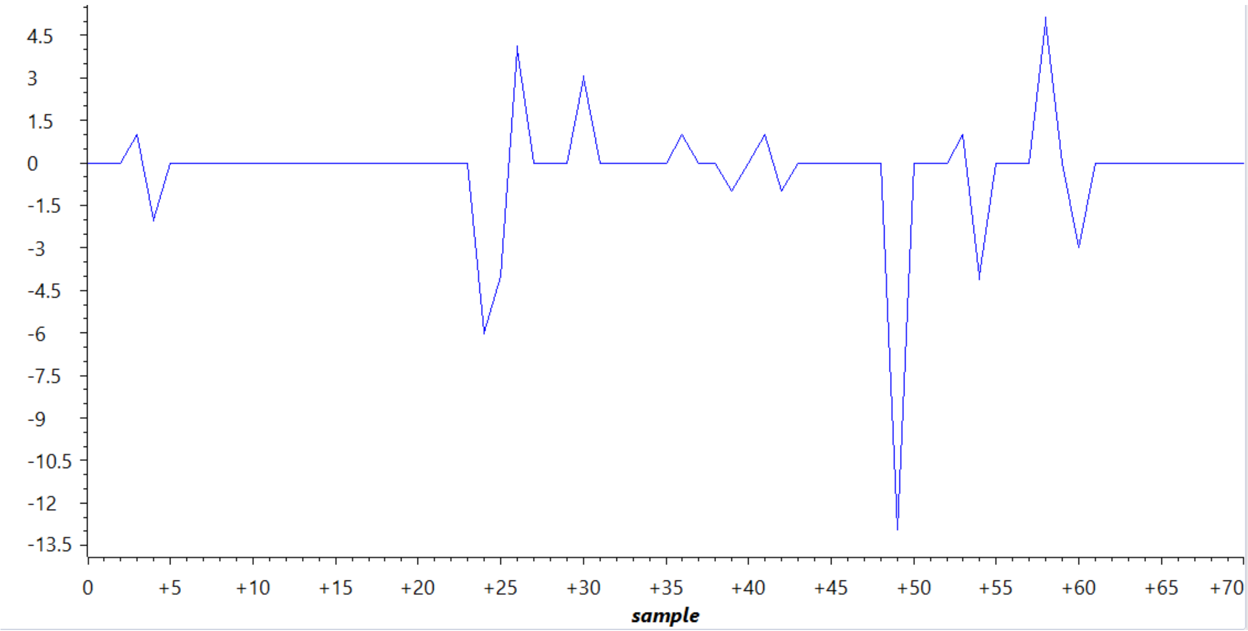
Task 04:

Graphs of the Pitch and Roll

Pitch



Roll



IMPLEMENTATION:

See code comments for step by step

```
InitI2C0();         //Initialize I2C, therefore, call I2C0 function
ConfigUART();   //Configure UART to so I can print on terminal
MPU6050Example(); //Get MPU6050 values
```

VIDEO LINKS:

Task 01: https://www.youtube.com/watch?v=dgCq4ldWung

Task 02: https://www.youtube.com/watch?v=zrL_wCLWnLU

Task 03: https://www.youtube.com/watch?v=M72fJuVoygM

Task 04: https://www.youtube.com/watch?v=FITj9PBKy2M

CODE:

Task 01 && Task02:

Task 02 is just graphing. It still uses the same code.

```
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_i2c.h"
#include "inc/hw_memmap.h"
```

```c
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "sensorlib/i2cm_drv.c"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/mpu6050.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "inc/hw_ints.h"
#include "inc/hw_sysctl.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
#include "IQmath/IQmathLib.h"
#include <math.h>



// A boolean that is set when a MPU6050 command has completed.
volatile bool g_bMPU6050Done;

// I2C master instance
tI2CMInstance g_sI2CMSimpleInst;

//---------------------------------------------------------------------------------
-
int main()
{
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_PLL | SYSCTL_OSC_INT |
SYSCTL_XTAL_16MHZ); //Set clock to 16MHz

    //Referenced past codes and slides for these functions
    InitI2C0();       //Initialize I2C, therefore, call I2C0 function
    ConfigUART();  //Configure UART to so I can print on terminal
    MPU6050Example(); //Get MPU6050 data, retrieved from slides

    return(0);
}

//---------------------------------------------------------------------------------
-
void InitI2C0(void)
{
    //enable I2C module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);

    //reset module
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);

    //enable GPIO peripheral that contains I2C 0
```

```
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

    // Configure the pin muxing for I2C0 functions on port B2 and B3.
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);

    // Select the I2C function for these pins.
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

    // Enable and initialize the I2C0 master module.  Use the system clock for
    // the I2C0 module.
    // I2C data transfer rate set to 400kbps.
    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), true);

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

    // Initialize the I2C master driver.
    I2CMInit(&g_sI2CMSimpleInst, I2C0_BASE, INT_I2C0, 0xff, 0xff, SysCtlClockGet());

}

//---------------------------------------------------------------------------------
-
void ConfigUART(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); //enable UART0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); //enable GPIOA peripherals(the UART
pins are on GPIO Port A)

    //Configure pins for the reciever and transmitter
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    UARTStdioConfig(0, 115200, 16000000);
}

//---------------------------------------------------------------------------------
// The interrupt handler for the I2C module.
void I2CIntHandler(void)
{
    I2CMIntHandler(&g_sI2CMSimpleInst);
}


//---------------------------------------------------------------------------------
// The function that is provided by this example as a callback when MPU6050
// transactions have completed.
void MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status)
{
    // See if an error occurred.
    if (ui8Status != I2CM_STATUS_SUCCESS)
```

```
    {
        // An error occurred, so handle it here if required.
    }
    // Indicate that the MPU6050 transaction has completed.
    g_bMPU6050Done = true;
}

//------------------------------------------------------------------------------------
// The MPU6050 example.
void MPU6050Example(void)
{
    float fAccel[3], fGyro[3];

    tMPU6050 sMPU6050;
    float x = 0, y = 0, z = 0;
    float xx = 0, yy = 0, zz = 0;

    // Initialize the MPU6050. This code assumes that the I2C master instance
    // has already been initialized.
    g_bMPU6050Done = false;
    MPU6050Init(&sMPU6050, &g_sI2CMSimpleInst, 0x68, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    // Configure the MPU6050 for +/- 4 g accelerometer range.
    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_ACCEL_CONFIG,
~MPU6050_ACCEL_CONFIG_AFS_SEL_M,
        MPU6050_ACCEL_CONFIG_AFS_SEL_4G, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_1, 0x00, 0b00000010 &
MPU6050_PWR_MGMT_1_DEVICE_RESET, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_2, 0x00, 0x00,
MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    // Loop forever reading data from the MPU6050.
    while (1)
    {
        // Request another reading from the MPU6050.
        g_bMPU6050Done = false;
        MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050);
        while (!g_bMPU6050Done)
```

```
        {
        }

        // Get the new accelerometer and gyroscope readings.
        MPU6050DataAccelGetFloat(&sMPU6050, &fAccel[0], &fAccel[1], &fAccel[2]);
        MPU6050DataGyroGetFloat(&sMPU6050, &fGyro[0], &fGyro[1], &fGyro[2]);

        // Do something with the new accelerometer and gyroscope readings.
        xx = fGyro[0]*10000;
        yy = fGyro[1]*10000;
        zz = fGyro[2]*10000;

        x = (atan2(fAccel[0], sqrt (fAccel[1] * fAccel[1] + fAccel[2] *
fAccel[2]))*180.0)/3.14;
        y = (atan2(fAccel[1], sqrt (fAccel[0] * fAccel[0] + fAccel[2] *
fAccel[2]))*180.0)/3.14;
        z = (atan2(fAccel[2], sqrt (fAccel[1] * fAccel[1] + fAccel[2] *
fAccel[2]))*180.0)/3.14;

        UARTprintf("Acc. X: %d | Acc. Y: %d | Acc. Z: %d\n", (int)x, (int)y, (int)z);
        UARTprintf("Gyro. XX: %d | Gyro. YY: %d | Gyro. ZZ: %d\n", (int)xx, (int)yy,
(int)zz);
        SysCtlDelay( (SysCtlClockGet()/(3*1000))*1000 ) ;
    }
}
```

Task 03 && Task 04:

Task 04 is just graphing. It still uses the same code.

```
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_i2c.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "sensorlib/i2cm_drv.c"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/mpu6050.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "inc/hw_ints.h"
#include "inc/hw_sysctl.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
#include "IQmath/IQmathLib.h"
```

```c
#include <math.h>



// A boolean that is set when a MPU6050 command has completed.
volatile bool g_bMPU6050Done;

#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 131
#define SAMPLE_RATE 0.01
#define RATIO (180/3.14)

// I2C master instance
tI2CMInstance g_sI2CMSimpleInst;

//Device frequency
int clockFreq;

//-----------------------------------------------------------------------------------
-
int main()
{
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_PLL | SYSCTL_OSC_INT |
SYSCTL_XTAL_16MHZ); //Set clock to 16MHz

    //Referenced past codes and slides for these functions
    InitI2C0();       //Initialize I2C, therefore, call I2C0 function
    ConfigUART();   //Configure UART to so I can print on terminal
    MPU6050Example(); //Get MPU6050 data, retrieved from slides

    return(0);
}

//-----------------------------------------------------------------------------------
-
void InitI2C0(void)
{
    //enable I2C module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);

    //reset module
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);

    //enable GPIO peripheral that contains I2C 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

    // Configure the pin muxing for I2C0 functions on port B2 and B3.
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);

    // Select the I2C function for these pins.
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

    // Enable and initialize the I2C0 master module.  Use the system clock for
```

```
    // the I2C0 module.
    // I2C data transfer rate set to 400kbps.
    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), true);

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

    // Initialize the I2C master driver.
    I2CMInit(&g_sI2CMSimpleInst, I2C0_BASE, INT_I2C0, 0xff, 0xff, SysCtlClockGet());

}

//-----------------------------------------------------------------------------------
-
void ConfigUART(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); //enable UART0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); //enable GPIOA peripherals(the UART
pins are on GPIO Port A)

    //Configure pins for the reciever and transmitter
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    UARTStdioConfig(0, 115200, 16000000);
}

//-----------------------------------------------------------------------------------
-
// The interrupt handler for the I2C module.
void I2CIntHandler(void)
{
    I2CMIntHandler(&g_sI2CMSimpleInst);
}

//-----------------------------------------------------------------------------------
-
void Complementary_Filter(float fAccel[], float fGyro[])
{
    _iq16 ForceMagApprx, PitchAcc, RollAcc, sensitivity, Rat, JKL, LKJ;
    _iq16 GyroVal[3], Acc[3];
    _iq16 Pitch;
    _iq16 Roll;

    Rat = _IQ16(RATIO);
    JKL = _IQ16(0.98);
    LKJ = _IQ16(0.02);
    GyroVal[0] = _IQ16(fGyro[0]);
    GyroVal[1] = _IQ16(fGyro[1]);
    GyroVal[2] = _IQ16(fGyro[2]);
    Acc[0] = _IQ16(fAccel[0]);
    Acc[1] = _IQ16(fAccel[1]);
    Acc[2] = _IQ16(fAccel[2]);
```

```
    sensitivity = _IQ16(GYROSCOPE_SENSITIVITY);

    Pitch += _IQ16mpy(_IQ16div(GyroVal[0],sensitivity), _IQ16(SAMPLE_RATE));
    Roll -= _IQ16mpy(_IQ16div(GyroVal[1],sensitivity), _IQ16(SAMPLE_RATE));
    ForceMagApprx = _IQabs(Acc[0]) + _IQabs(Acc[1]) + _IQabs(Acc[2]);

    if(ForceMagApprx > 1411510 && ForceMagApprx < 4705028)
    {
        PitchAcc = _IQ16mpy(_IQ16atan2(Acc[1],Acc[2]), Rat);
        Pitch = _IQ16mpy(Pitch,JKL) + _IQ16mpy(PitchAcc,LKJ);
        RollAcc = _IQ16mpy(_IQ16atan2(Acc[0],Acc[2]), Rat);
        Roll = _IQ16mpy(Roll,JKL) + _IQ16mpy(RollAcc,LKJ);
        UARTprintf("Pitch  : %d  | Roll  : %d \n", (int)Pitch, (int)Roll);
    }
}

//--------------------------------------------------------------------------------------
-
// The function that is provided by this example as a callback when MPU6050
// transactions have completed.
void MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status)
{
    // See if an error occurred.
    if (ui8Status != I2CM_STATUS_SUCCESS)
    {
        // An error occurred, so handle it here if required.
    }
    // Indicate that the MPU6050 transaction has completed.
    g_bMPU6050Done = true;
}

//--------------------------------------------------------------------------------------
-
// The MPU6050 example.
void MPU6050Example(void)
{
    float fAccel[3], fGyro[3];

    tMPU6050 sMPU6050;
    float x = 0, y = 0, z = 0;
    float xx = 0, yy = 0, zz = 0;

    // Initialize the MPU6050. This code assumes that the I2C master instance
    // has already been initialized.
    g_bMPU6050Done = false;
    MPU6050Init(&sMPU6050, &g_sI2CMSimpleInst, 0x68, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    // Configure the MPU6050 for +/- 4 g accelerometer range.
    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_ACCEL_CONFIG,
~MPU6050_ACCEL_CONFIG_AFS_SEL_M,
        MPU6050_ACCEL_CONFIG_AFS_SEL_4G, MPU6050Callback, &sMPU6050);
```

```
    while (!g_bMPU6050Done)
    {
    }

    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_1, 0x00, 0b00000010 &
MPU6050_PWR_MGMT_1_DEVICE_RESET, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_2, 0x00, 0x00,
MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    // Loop forever reading data from the MPU6050.
    while (1)
    {
        // Request another reading from the MPU6050.
        g_bMPU6050Done = false;
        MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050);
        while (!g_bMPU6050Done)
        {
        }

        // Get the new accelerometer and gyroscope readings.
        MPU6050DataAccelGetFloat(&sMPU6050, &fAccel[0], &fAccel[1], &fAccel[2]);
        MPU6050DataGyroGetFloat(&sMPU6050, &fGyro[0], &fGyro[1], &fGyro[2]);

        // Do something with the new accelerometer and gyroscope readings.
//        xx = fGyro[0]*10000;
//        yy = fGyro[1]*10000;
//        zz = fGyro[2]*10000;
//
//        x = (atan2(fAccel[0], sqrt (fAccel[1] * fAccel[1] + fAccel[2] *
fAccel[2]))*180.0)/3.14;
//        y = (atan2(fAccel[1], sqrt (fAccel[0] * fAccel[0] + fAccel[2] *
fAccel[2]))*180.0)/3.14;
//        z = (atan2(fAccel[2], sqrt (fAccel[1] * fAccel[1] + fAccel[2] *
fAccel[2]))*180.0)/3.14;

        Complementary_Filter(fAccel, fGyro);

//        UARTprintf("ACC.  X: %d  | ACC. Y: %d | ACC. Z: %d\n", (int)x, (int)y,
(int)z);
//        UARTprintf("GYRO. XX: %d | GYRO. YY: %d | GYRO. ZZ: %d\n", (int)xx,
(int)yy, (int)zz);
        SysCtlDelay( (SysCtlClockGet()/(3*1000))*1000 ) ;
    }
}
```

CONCLUSION:

Overall, I believe I was able to successfully print and graph the acc. and gyro values on the serial monitor and graph them. Task 1 and task 3 were code intensive and on just prints raw values and the other prints the filtered values. Task 2 and 4 simply print it on a graph.

Name: Itzel Becerril