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.

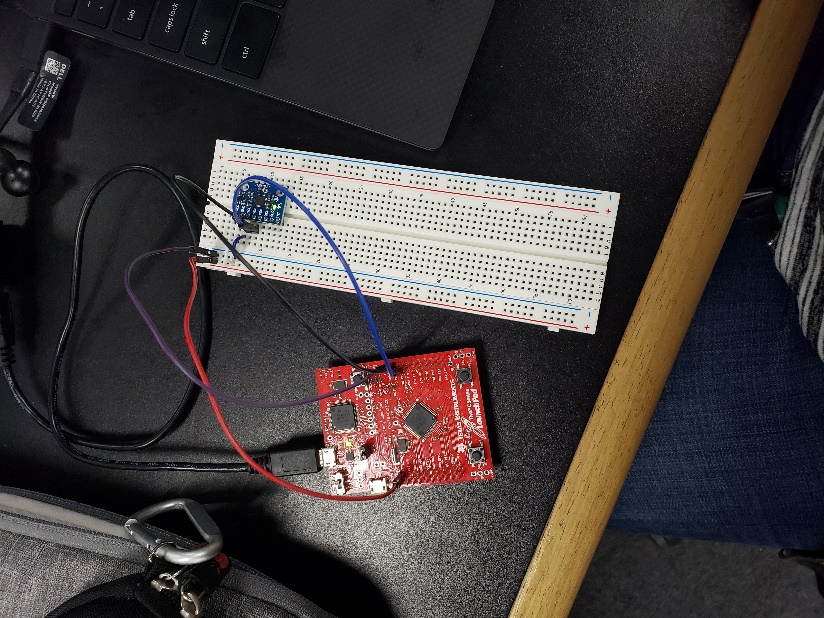
DELIVERABLES:

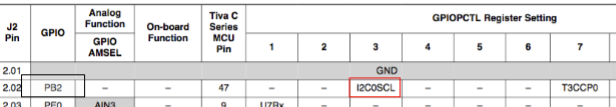
Task 1 and 3 are heavy in code. It completes the task such as displaying raw values and then the other code incorporates the complementary filter. Will deliver codes, graphs, YouTube video links, screenshots, and images. Will also be added to my GitHub repository.

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:







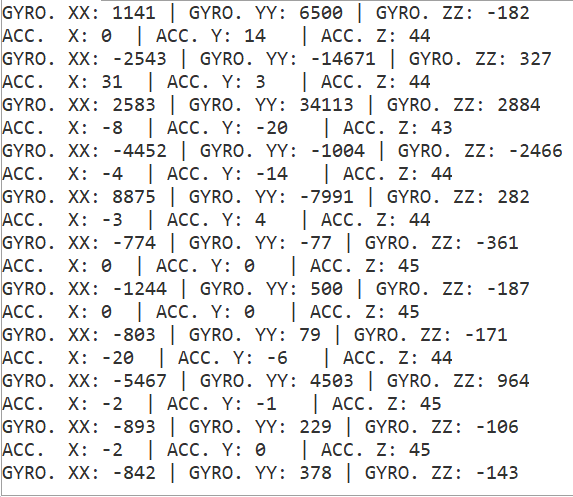
PB2 –I2C0SCL

PB3 –I2C0SDA

SCREENSHOTS:

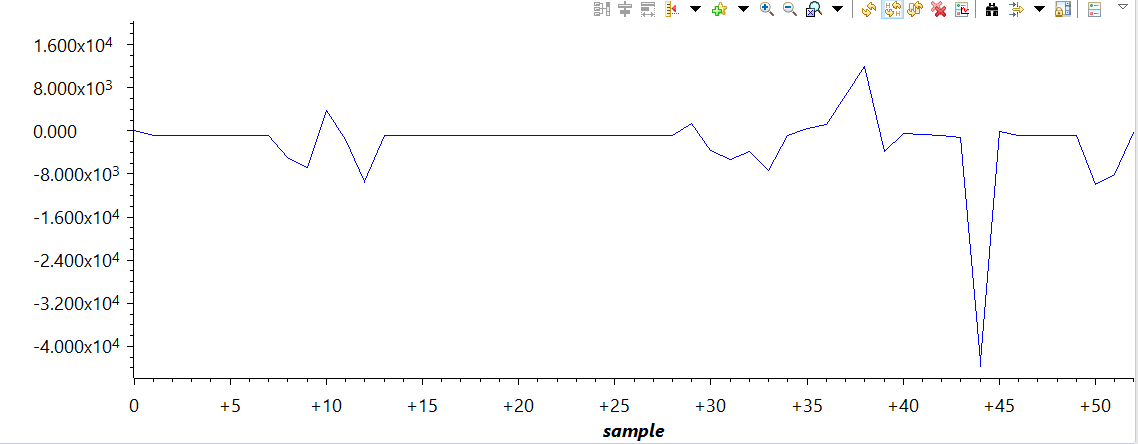
Task 01:

Output of the raw values

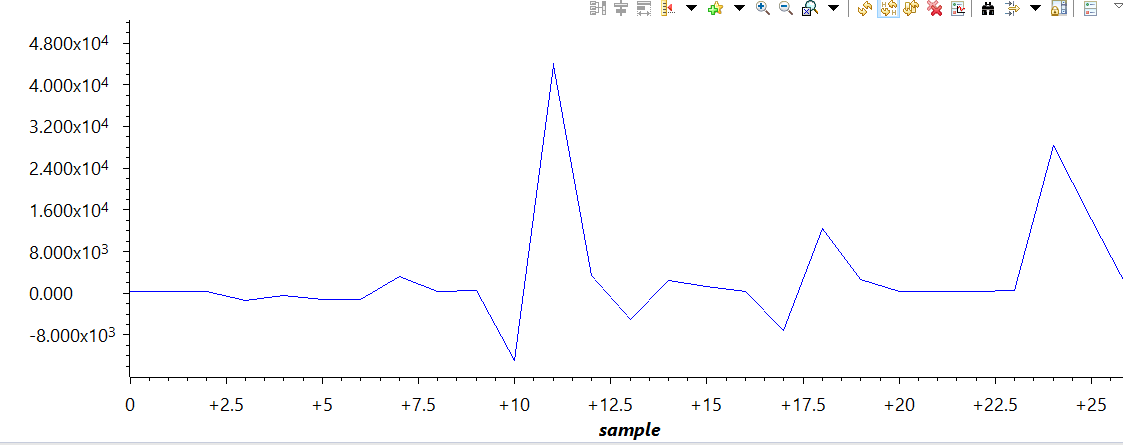


Task 02:

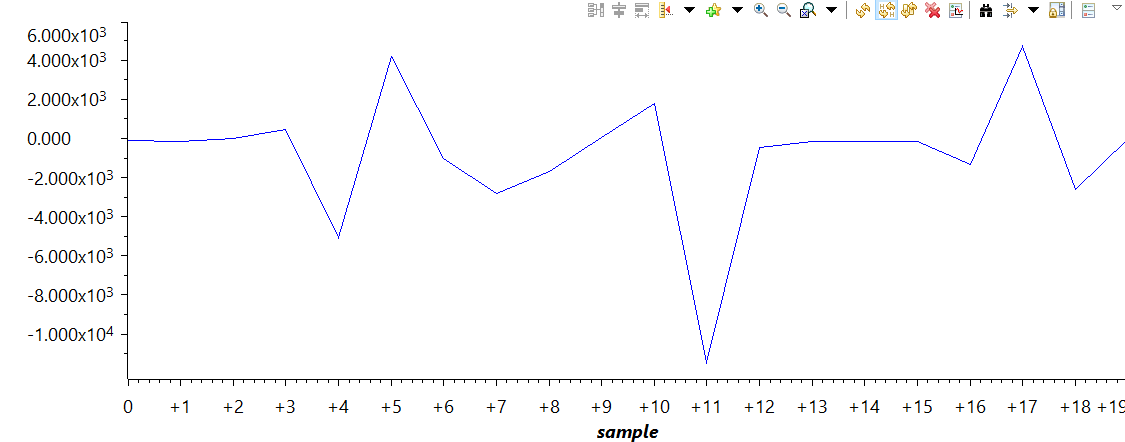
Gyro-X:



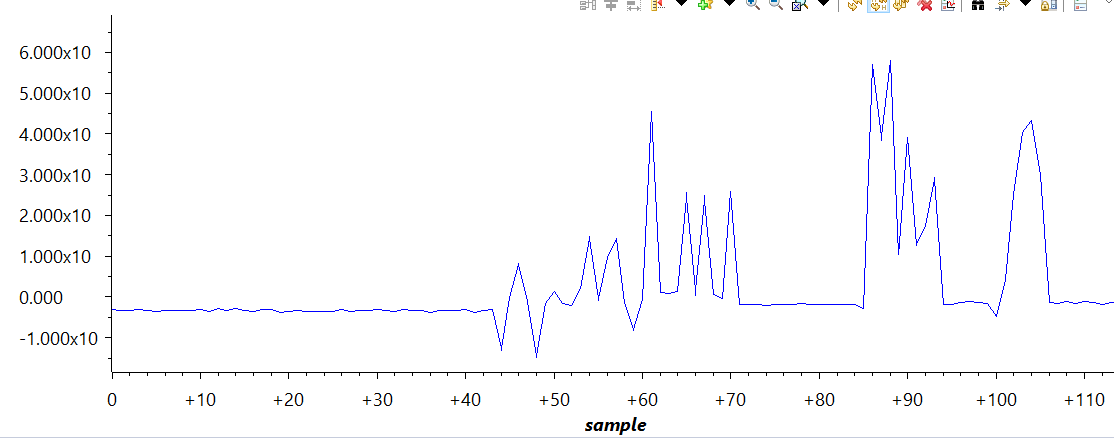
Gyro-Y:



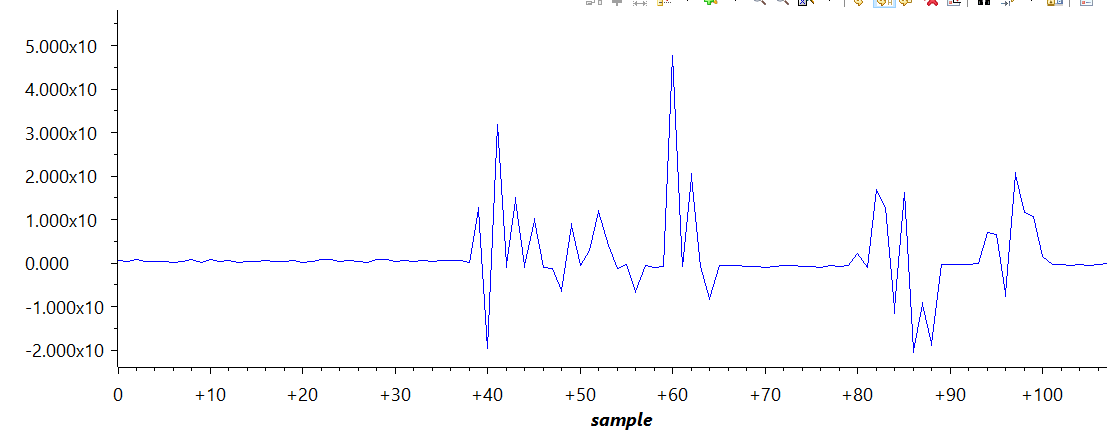
Gyro-Z:



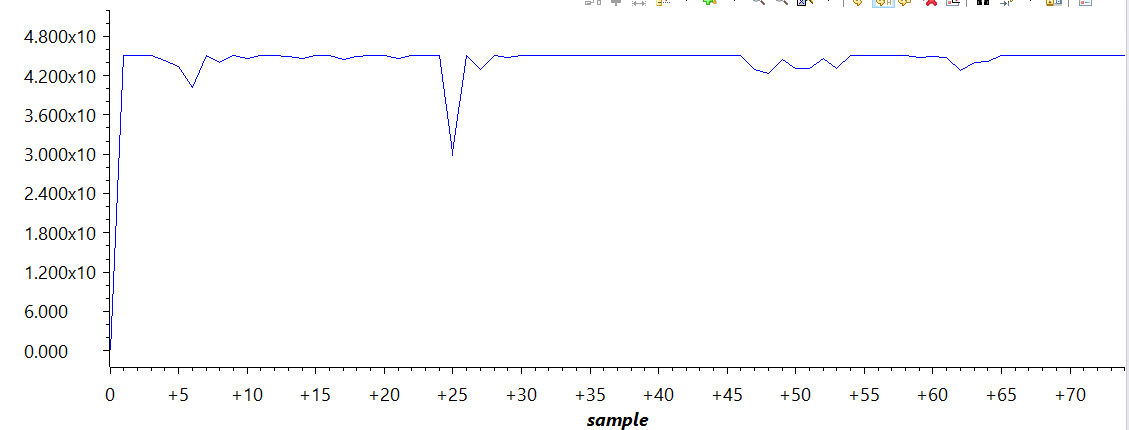
Acc-X:



Acc-Y:

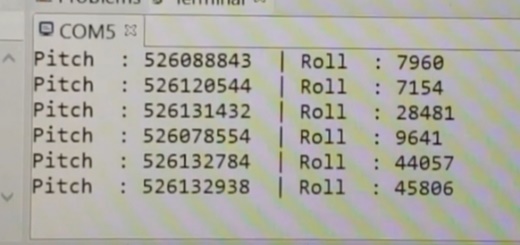


Acc-Z:



Task 03:

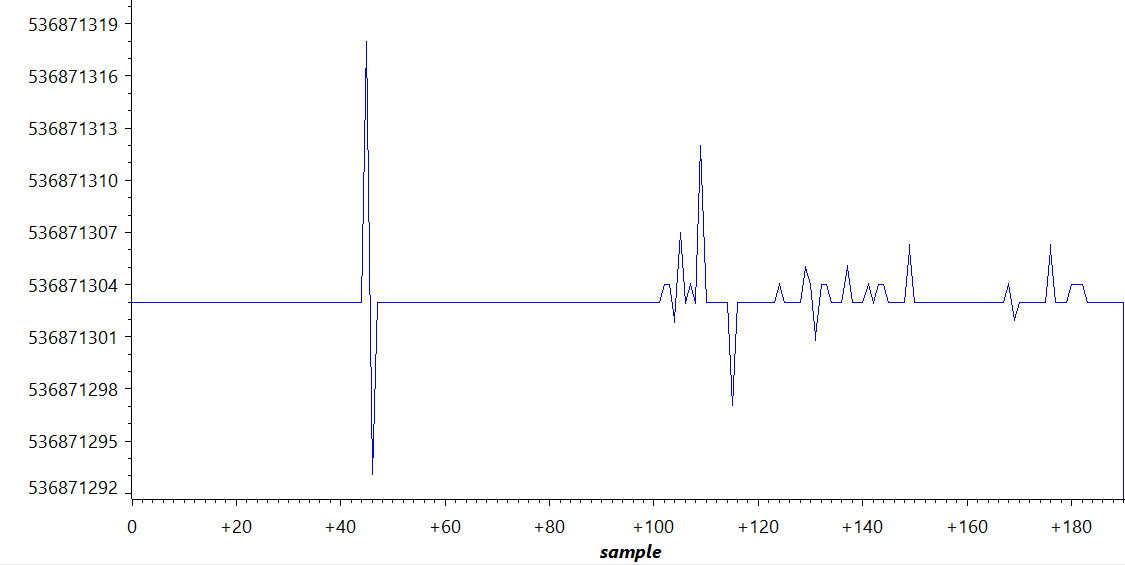
Pitch and Roll values printed on the serial monitor



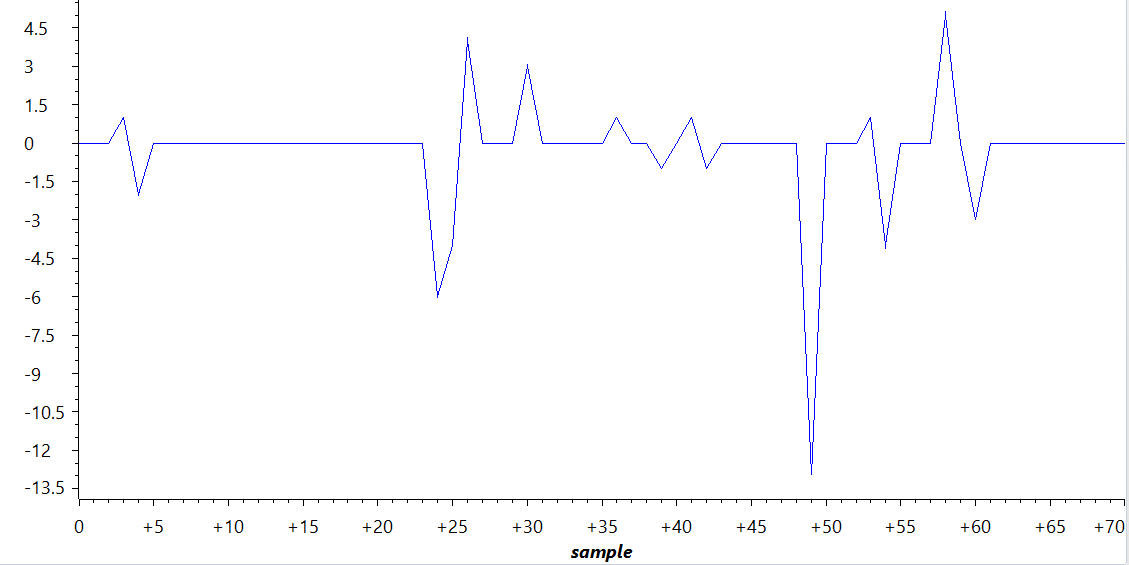
Task 04:

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"

**#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"

**#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