CPE 403 **ADV EMB SYS DES** **Fall 2019**

TITLE: MIDTERM

Name: Jett Guerrero Date: 10/26/2019

Github Link: https://github.com/guerrj1/Advanced-Embedded-Systems

GOAL:

Task1-2: Interface MPU6050 IMU using the I2C protocol onto the TIVA C and display the values of accelerometer and gyro measurements onto the serial terminal and graph using the graph tool.

Task3-4: Implement a complementary filter that will filter the accelerometer and gyro raw values and display the raw and filtered values onto the serial terminal and graph using the graph tool.

DELIVERABLES:

The intended project deliverables are screenshots of the serial terminal and the graphs of the accelerometer, gyro, pitch and roll values as well as the video links of the demo.

Video Links:

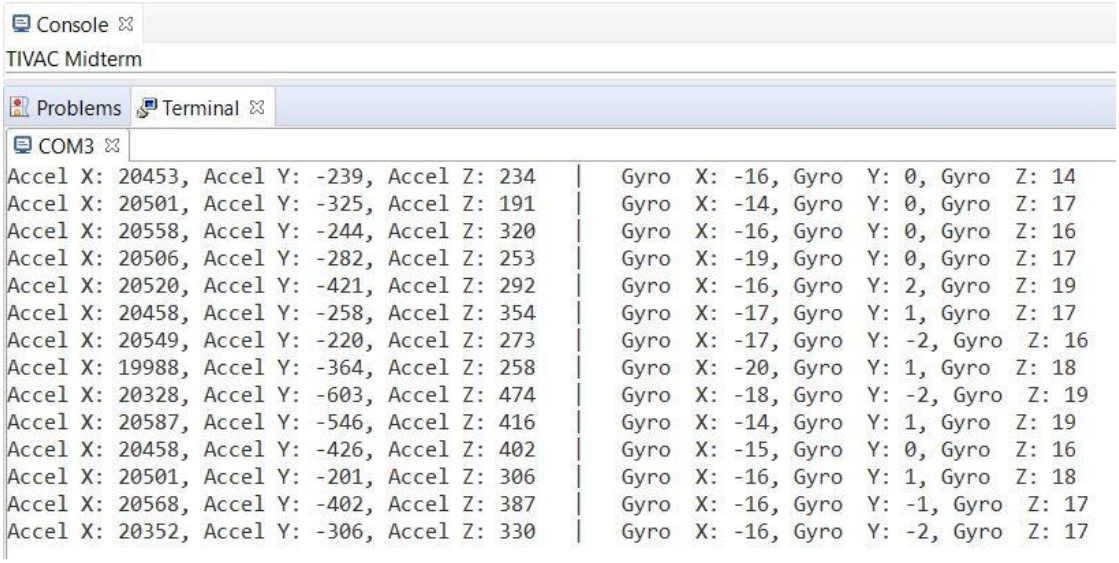
Task 1: https://youtu.be/-bjQqzfrJWU

Task 2: https://youtu.be/mBbOTtiiiLQ

Task 3: https://youtu.be/hV8p-V-xEJU

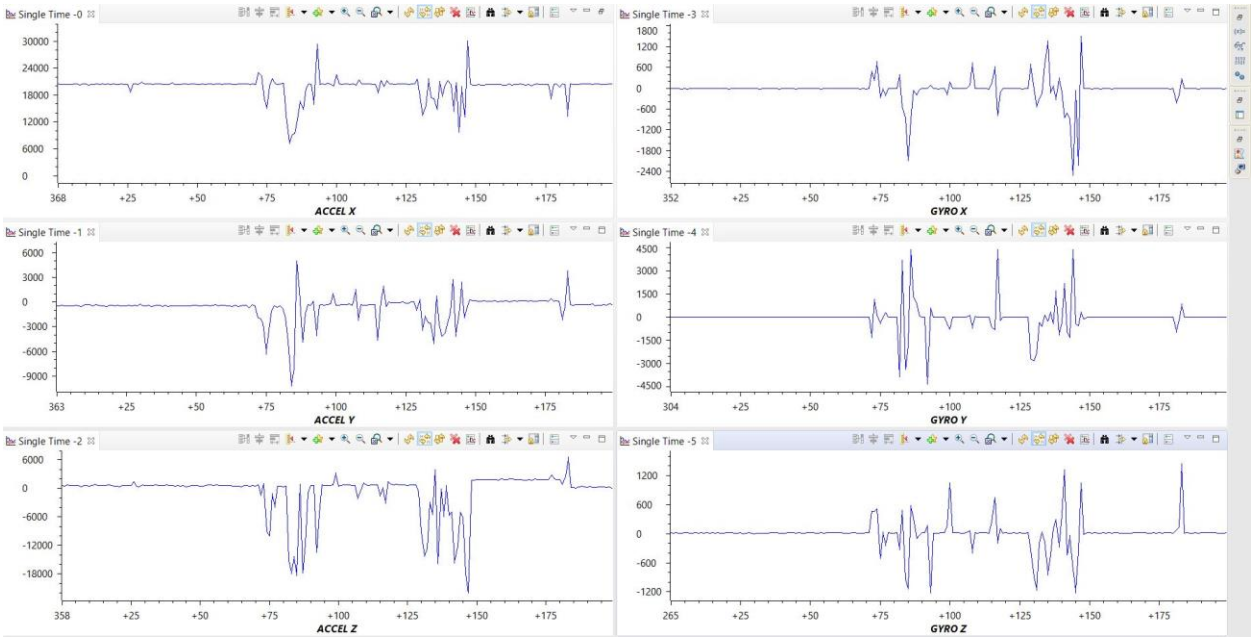
Task 4: https://youtu.be/FBAXTNJCAA0

TASK 1:



Serial Terminal with Accelerometer and Gyro values of X, Y, Z axis

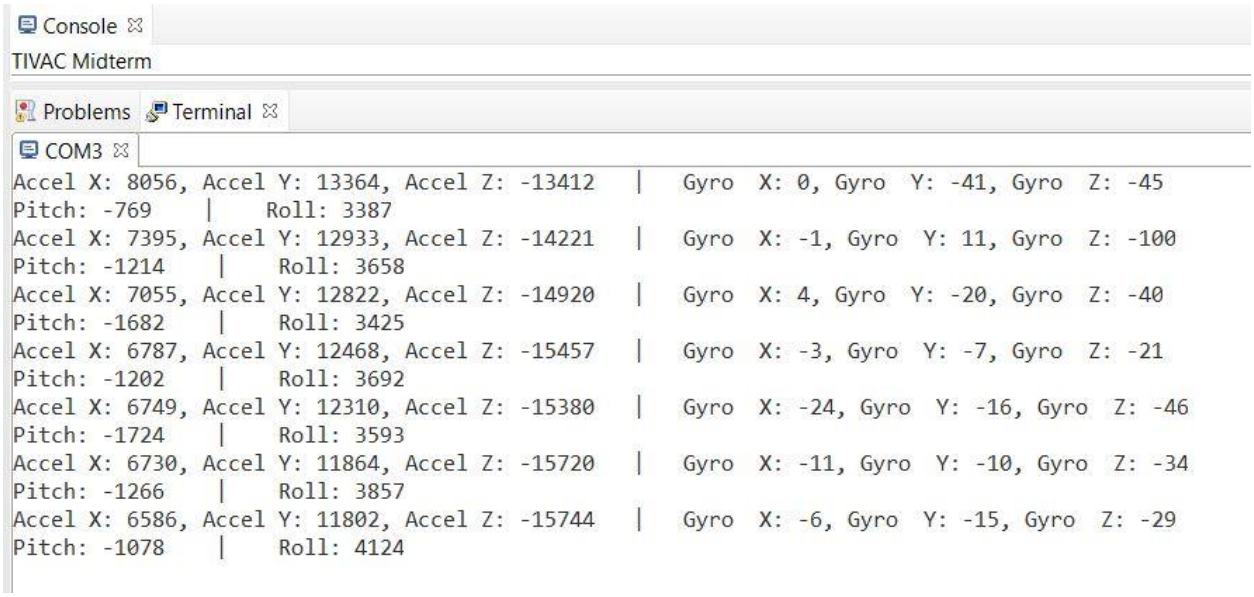
TASK 2:



Graph for Accel X, Y Z

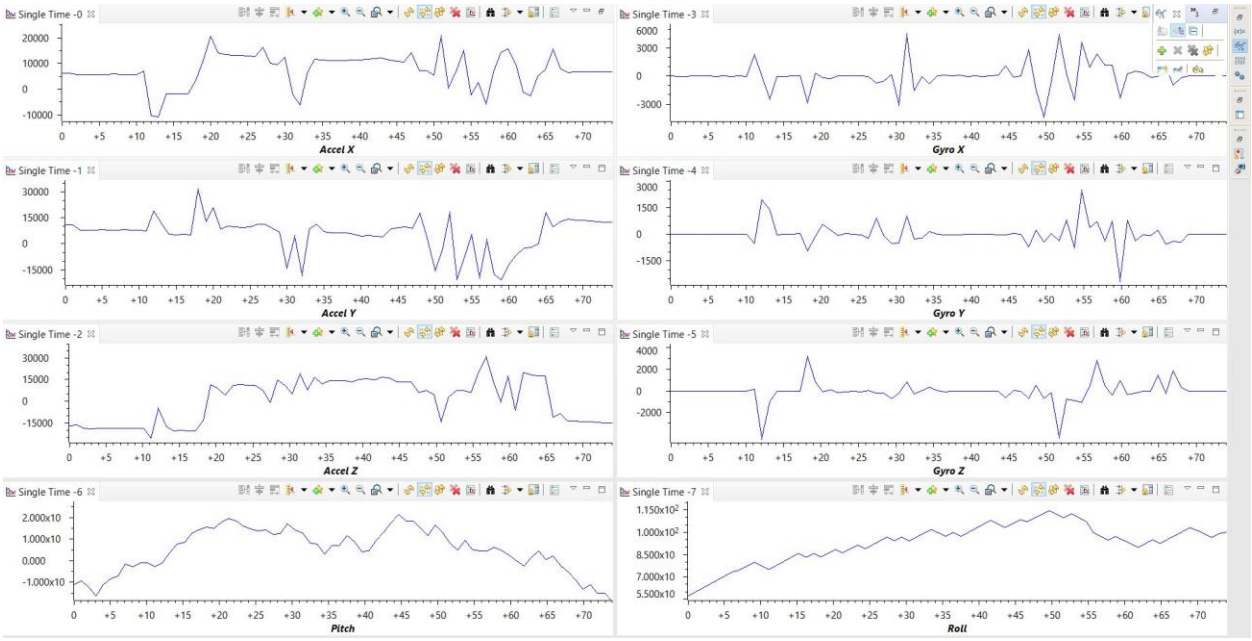
Graph of Gyro X, Y, Z

TASK 3:



Serial Terminal with Accelerometer and Gyro values of X, Y, Z axis, Pitch and Roll

TASK 4:



Graph for Accel X, Y Z

Graph of Gyro X, Y, Z

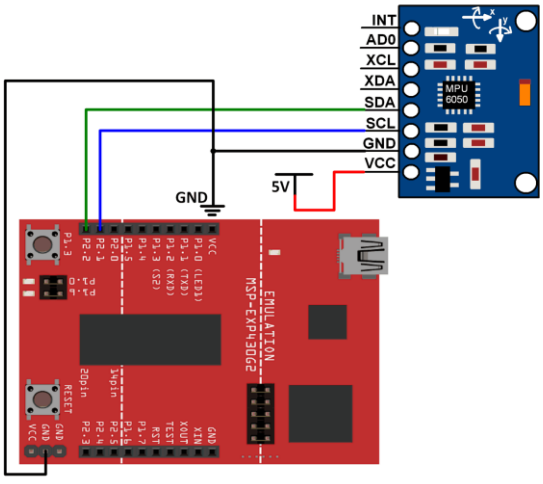
Pitch

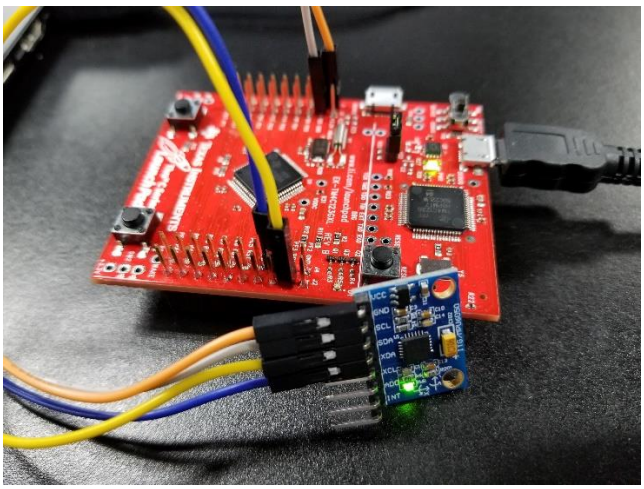
Roll

COMPONENTS:

Components used in this project are the TIVA C launchpad TM4C123GXL and MPU6050

SCHEMATICS:





CODE TASK 1-2:

```
//Midterm TIVA C Task 1 and 2
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include <string.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "sensorlib/i2cm_drv.h"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/mpu6050.h"


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

tI2CMInstance g_sI2CMSimpleInst;  // I2C master instance

int main()
{
        //clock initialization
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

        InitI2C0();  //InitI2C0 function call

        //UART initialization
        SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        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);

        MPU6050Example(); //MPU6050Example function call

        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());

}

// 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.
	}
	g_bMPU6050Done = true;  // Indicate that the MPU6050 transaction has completed.
}


// The interrupt handler for the I2C module.
void I2CMSimpleIntHandler(void)
{
	I2CMIntHandler(&g_sI2CMSimpleInst);     // Call the I2C master driver interrupt handler.
}

// The MPU6050 example.
void MPU6050Example(void)
{
	float fAccel[3], fGyro[3]; //array for acceleration and gyro values
	float Ax = 0;
	float Ay = 0;
	float Az = 0;
	float Gx = 0;
	float Gy = 0;
	float Gz = 0;
	int scale = 1000; //measurement values to be scaled
```

```
        int dispdelay = 10000000; //delay value

        tMPU6050 sMPU6050;

        //
        // 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)
        {
        }

        //Configure MPU6050 settings
        g_bMPU6050Done = false;
        MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_1, 0x00, 0b00000010
        & MPU6050_PWR_MGMT_1_DEVICE_RESET, MPU6050Callback, &sMPU6050);
        while (!g_bMPU6050Done)
        {
        }

        //Configure MPU6050 settings
        g_bMPU6050Done = false;
        MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_2, 0x00, 0x00, MPU6050Callback,
&sMPU6050);
        while (!g_bMPU6050Done)
        {
        }

        //
        // Loop forever reading data from the MPU6050. Typically, this process
        // would be done in the background, but for the purposes of this example,
        // it is shown in an infinite loop.
        //
        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.
                //
                int intAx = Ax;
                int intAy = Ay;
                int intAz = Az;
                int intGx = Gx;
                int intGy = Gy;
                int intGz = Gz;

                //Accelerometer readings
                Ax = fAccel[0] * scale;
                Ay = fAccel[1] * scale;
                Az = fAccel[2] * scale;

                //Gyro readings
                Gx = fGyro[0] * scale;
                Gy = fGyro[1] * scale;
                Gz = fGyro[2] * scale;

                //value print out onto terminal
                UARTprintf("Accel X: %d, Accel Y: %d, Accel Z: %d  |  Gyro  X: %d, Gyro  Y: %d, Gyro  Z: %d\n",
                (int)Ax, (int)Ay, (int)Az, (int) Gx, (int) Gy, (int) Gz);
                SysCtlDelay(dispdelay); //display delay

        }
}
```

CODE TASK 3-4:

```
//Midterm TIVA C Task 3 and 4
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include <string.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include <math.h>
#include "math.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
```

```
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "sensorlib/i2cm_drv.h"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/mpu6050.h"
#include "IQmath/IQmathLib.h"

#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 65.536
#define SAMPLE_RATE 0.01
#define M_PI 3.14159265359
#define dt 0.01   //10 ms sample rate

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

tI2CMInstance g_sI2CMSimpleInst;  // I2C master instance

int main()
{
        //clock initialization
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

        InitI2C0();  //InitI2C0 function call

        //UART initialization
        SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        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);

        MPU6050Example(); //MPU6050Example function call

        return(0);
}

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[0], (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;
        }
}


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());

}

// 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.
        }
        g_bMPU6050Done = true;  // Indicate that the MPU6050 transaction has completed.
}


// The interrupt handler for the I2C module.
void I2CMSimpleIntHandler(void)
{
```

```
        I2CMIntHandler(&g_sI2CMSimpleInst);     // Call the I2C master driver interrupt handler.
}

// The MPU6050 example.
void MPU6050Example(void)
{
        float fAccel[3], fGyro[3]; //array for acceleration and gyro values
        float Ax = 0;
        float Ay = 0;
        float Az = 0;
        float Gx = 0;
        float Gy = 0;
        float Gz = 0;
        float pitch = 0;
        float roll = 0;
        int scale = 1000; //measurement values to be scaled
        int scale2 = 100; //measurment values to be scaled
        int dispdelay = 10000000; //delay value

        tMPU6050 sMPU6050;

        //
        // 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)
        {
        }

        //Configure MPU6050 settings
        g_bMPU6050Done = false;
        MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_1, 0x00, 0b00000010
        & MPU6050_PWR_MGMT_1_DEVICE_RESET, MPU6050Callback, &sMPU6050);
        while (!g_bMPU6050Done)
        {
        }

        //Configure MPU6050 settings
        g_bMPU6050Done = false;
        MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_2, 0x00, 0x00, MPU6050Callback,
&sMPU6050);
        while (!g_bMPU6050Done)
        {
        }

        //
```

```
        // Loop forever reading data from the MPU6050. Typically, this process
        // would be done in the background, but for the purposes of this example,
        // it is shown in an infinite loop.
        //
        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.
                //

                int intAx = Ax;
                int intAy = Ay;
                int intAz = Az;
                int intGx = Gx;
                int intGy = Gy;
                int intGz = Gz;

                //Accelerometer readings
                Ax = fAccel[0] * scale;
                Ay = fAccel[1] * scale;
                Az = fAccel[2] * scale;

                //Gyro readings
                Gx = fGyro[0] * scale;
                Gy = fGyro[1] * scale;
                Gz = fGyro[2] * scale;

                //complementary filter function call
                ComplementaryFilter(fAccel, fGyro, &pitch, &roll);

                //value print out onto terminal
                UARTprintf("Accel X: %d, Accel Y: %d, Accel Z: %d   |   Gyro  X: %d, Gyro  Y: %d, Gyro  Z: %d\n",
                (int)Ax, (int)Ay, (int)Az, (int) Gx, (int) Gy, (int) Gz);

                //filtered value print out onto terminal
                UARTprintf("Pitch: %d    |    Roll: %d\n", (int)(pitch*scale2), (int)(roll*scale2));
                SysCtlDelay(dispdelay); //display delay

        }
}
```

Name: Jett Guerrero Date: 10/26/2019