CAN-RGX: Embedded Subsystem

Hanzhen Lin, Tyler Gamvrelis, Twesh Upadhyaya

April 11, 2018

Introduction

This document describes the embedded subsystem onboard team "FAM"'s (fluids affect by magnetism) experiment for the Canadian Reduced Gravity Experiment Design Challenge.

Contents

1. Overview

The embedded subsystem is responsible for acquiring data from sensors, using this data to initiate control sequences, and streaming data to the onboard PC. The sensors are as follows:

1. MPU9250: 3-axis accelerometer, gyroscope, and magnetometer, 1 in quantity
2. MC9701A: analog temperature sensor

The systems to which control signals will be asserted are as follows:

1. DRV8871: H-bridge, 6 in quantity
2. TEC: thermoelectric cooler, 2 in quantity

The proceeding sections will describe each of these in more detail, and subsequently, their integration will be described.

2. MPU9250

As the MPU9250 measures acceleration, it is capable of automatically triggering the experiment through software once the acceleration magnitude is below a certain threshold. It can also detect other events of interest throughout the flight. The aircraft detected these events according to the following algorithm, where:

1. w_y is the rate of change of the pitch of the aircraft about the y-axis (also called *angular rate*). Note that pitch is the angle between the nose of the aircraft and the horizon in degrees per second.
2. a_z is the vertical acceleration of the aircraft. **Note that $a_z > 0$ implies the aircraft is accelerating downward**.

Algorithm 1 Algorithm used by plane for triggering flight events

```
procedure FlightEventsPlane
    Measure w_y and a_y
    if w_y > 2.5 AND a_z < −14.715 (1.5G) then
        return Transition from Straight and level to Pull Up
    else if w_y < 0 AND a_z > −0.981 (0.1G) then
        return Transition from Pull-up to Reduced Gravity
    else if w_y > 0 AND a_z < −0.981 (0.1G) then
        return Transition from Reduced Gravity to Pull-Out
    else
        return No flight event
```

The algorithm actually implemented on the microcontroller was much more simple in nature and only detected reduced gravity events. It is shown in Algorithm 2.

Algorithm 2 Algorithm used in microcontroller for detecting reduced gravity

```
procedure FlightEventsMicro
    Measure a_z, a_y, a_x
    if ||a|| < 0.981 then
        return Reduced gravity
    else
        return No flight event
```

Note: It was suggested that a manual mechanism be provided to trigger the experiment in addition to the automated software protocol

The MPU9250 is also used to measure the magnetic field generated by the electromagnet. This allows the magnetic field to be generated to its maximum possible value while maintaining safety limits, and also is important for analysis of the experimental results.

At a minimum, the MPU9250 must be used to measure the following:

- a_z (16 bits, full-scale range $\pm 2g$)
- w_y (16 bits, full-scale range ± 250 [deg /s])
- H_x, H_y, H_z (14 bits each, full-scale range ± 4912 [uT])

9.1 Orientation of Axes

The diagram below shows the orientation of the axes of sensitivity and the polarity of rotation. Note the pin 1 identifier (•) in the figure.

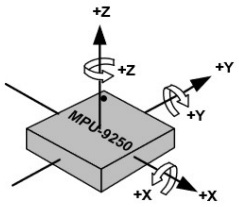

Figure 4. Orientation of Axes of Sensitivity and Polarity of Rotation for Accelerometer and Gyroscope

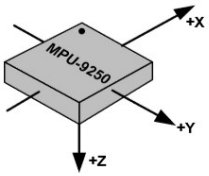

Figure 5. Orientation of Axes of Sensitivity for Compass

Figure 1: Orientation of axes for MPU9250 data (page 38 of datasheet).

Note that these full-scale ranges were the defaults and also the most suitable since they provided the highest resolution for the window of interest. Note that the full-scale range of 4912 for H_i was taken from page 50 of the register map, while the datasheet specific 4800.

Due to the importance of the readings from the MPU9250, it was decided that the motion processing algorithms should be run at a high rate, about 500 [Hz]. Also, note that the MPU9250 features offset registers that can be user-programmed to eliminate dc offsets. It also features an integrated digital low-pass filter whose bandwidth can be set using the `DLPF_CFG` and `A_DLFPCFG` for the gyroscope and accelerometer, respectively. Note that lower bandwidth comes at the cost of increased delay, on the order of milliseconds.

5.6 HXL to HZH: Measurement Data

Addr	Register name	D7	D6	D5	D4	D3	D2	D1	D0
Read-only register									
03H	HXL	HX7	HX6	HX5	HX4	HX3	HX2	HX1	HX0
04H	HXH	HX15	HX14	HX13	HX12	HX11	HX10	HX9	HX8
05H	HYL	HY7	HY6	HY5	HY4	HY3	HY2	HY1	HY0
06H	HYH	HY15	HY14	HY13	HY12	HY11	HY10	HY9	HY8
07H	HZL	HZ7	HZ6	HZ5	HZ4	HZ3	HZ2	HZ1	HZ0
08H	HZH	HZ15	HZ14	HZ13	HZ12	HZ11	HZ10	HZ9	HZ8
Reset		0	0	0	0	0	0	0	0

Measurement data of magnetic sensor X-axis/Y-axis/Z-axis

HXL[7:0]: X-axis measurement data lower 8bit
HXH[15:8]: X-axis measurement data higher 8bit
HYL[7:0]: Y-axis measurement data lower 8bit
HYH[15:8]: Y-axis measurement data higher 8bit
HZL[7:0]: Z-axis measurement data lower 8bit
HZH[15:8]: Z-axis measurement data higher 8bit

Measurement data is stored in two's complement and Little Endian format. Measurement range of each axis is from -32760 ~ 32760 decimal in 16-bit output.

Measurement data (each axis) [15:0]			Magnetic flux density [μT]
Two's complement	Hex	Decimal	
0111 1111 1111 1000	7FF8	32760	4912(max.)
|	|	|	|
0000 0000 0000 0001	0001	1	0.15
0000 0000 0000 0000	0000	0	0
1111 1111 1111 1111	FFFF	-1	-0.15
|	|	|	|
1000 0000 0000 1000	8008	-32760	-4912(min.)

Table 4 Measurement data format

Figure 2: Magnetometer register map and corresponding scales from page 50 of the register map.

Implementation details will now be described. To begin, it was decided that the 400 $[kHz]$ I^2C bus would be used to transfer data to and from the sensor. However, there is no reason why this could not be changed to 1 $[MHz]$ or 20 $[MHz]$ SPI. The code related to the MPU9250 was split into 2 parts:

1. start-up routine, where the peripheral is initialized
2. data-gathering and processing routine used during the experiment

Start-up routine
Initializing the MPU9250 consists of initializing two different devices, the accelerometer and gyroscope (device 1) and the magnetometer (device 2). Although these are integrated as 1 physical package, under the hood they remain distinct. Consequently, the accelerometer and gyroscope have a slave address of 0x68, and the magnetometer has a slave address of 0x0C. Now, there are two important points to make before continuing:

- The magnetometer must be controlled *by proxy* through the accelerometer and gyroscope. Essentially, the accelerometer and gyroscope have the ability to control *slave devices*, and in the MPU9250, "slave 0" is directly connected to the magnetometer.
- ST's HAL libraries for I^2C require the slave address to be shifted 1 bit to the left in calls to I2C functions. This only affects calls to the accelerometer and gyroscope, since as was just noted above, the magnetometer is ultimately controlled by these.

This will become more clear shortly.

The start-up routine is called prior to starting the scheduler. This way, more primitive I/O interfaces (i.e. polled I/O) can be used to avoid complexity. Also, it increases the probability that nothing will interrupt the start-up sequence since there is no context switching.
First, all the fields in the global MPU9250 handle, called `myMPU9250`, are initialized to -1.0. Next, the WHO_AM_I register is read from the accelerometer and gyroscope. This is a read-only register that should always return 0x71, so this check basically ensures that the correct device is connected to the bus. Next, the accelerometer and gyroscope are configured as follows:

1. The best available clock source is selected
2. The I^2C master interface module is enabled
3. The I^2C module is set to 400 $[kHz]$ bus speed
4. The accelerometer and gyroscope are set to ON
5. I^2C bypass is enabled. This is critically important for getting the magnetometer to work

Next, the magnetometer is configured as follows:

1. Check the ID of the magnetometer
2. Reset the magnetometer and delay 10 $[ms]$
3. Enable continuous measurement

If the startup routine makes it to the end, it returns 1. Otherwise, a unique negative integer is returned whose value depends on the cause of failure.

Data-gathering and processing routine
This routine runs after the scheduler is started. It begins by calling `vTaskDelayUntil()` such that it runs every 2 milliseconds (500 $[Hz]$ refresh rate). Next, the accelerometer, gyroscope, and magnetometer are read. Then, the acceleration and angular rate are checked for the experiment event conditions as per Algorithm 2.

The total number of bytes required for these transfers is as follows. Note that write transactions are 2 bytes, while read transactions are 2 bytes plus the number of bytes read back

- Get accelerometer data for a_z: 1 write transaction (accelerometer data address) + 2 bytes returned for the data = 4 bytes
- Get accelerometer data for a_y: 4 bytes
- Get accelerometer data for a_x: 4 bytes
- Get angular rate data for v_z: 1 write transaction (angular rate data address) + 2 bytes returned for the data = 4 bytes
- Get angular rate data for v_y: 4 bytes
- Get angular rate data for v_x: 4 bytes
- Get magnetometer data: 1 call to `magnetometerRead()` = 3 write transactions (I2C_SLV0_ADDR, I2C_SLV0_REG, I2C_SLV0_CTRL) + 1 read transaction of 6 bytes for magnetometer data (starting from EXT_SENS_DATA_00) = 3 * 2 + 2 + 6 = 14 bytes

The total number of bytes transferred to get all the data required for the experiment is thus:

$$4 \times 3 + 4 \times 3 + 14 = 38 \; bytes \tag{1}$$

So the time required for the data transfer is:

$$38 * 8/400 \times 10^3 = 760 \; [\mu s] \tag{2}$$

The acquisition of the accelerometer and gyroscope data is accomplished using DMA. First, `HAL_I2C_Mem_Read_DMA` is called to initiate the data transfers, and then `xSemaphoreTake(semMPU9250Handle, portMAX_DELAY)` is executed. This second line forces the MPU9250 thread to block (i.e. not run) until the semaphore `semMPU9250Handle` is given from the I2C callback, `HAL_I2C_MemRxCpltCallback`. For reading from the magnetometer, interrupt-driven writes were used along with DMA-based reads. Interrupt-driven writes were used instead of DMA-based writes because the callback for DMA-based writes was not working properly.

3. MC9701A

The temperature sensor is used to monitor the temperature of the electromagnet.
Can have up to 6 of these with the current board TODO: Details

4. DRV8871

The H-bridges are used to generate magnetic fields of various strengths in the coils surrounding the fluid cell, and to generate various heat gradients using the TECs. PWM is used in both cases.

5. TEC

The TEC plates are used to supply heat and cooling to the fluid.
Power data should be acquired at about 500 Hz

6. Integration

The aforementioned components were integrated using a STM32F446RE microcontroller on a Nucleo-F446RE development board.

6.1 Data Logging

Data from all sensors were logged and sent to the onboard PC over a 230400 [Hz] UART channel. The UART function used DMA and a semaphore to optimize CPU usage. The packet format was as follows:

Parameter	Data Type	Size (bytes)	Offset (bytes)	Units	Range	Positive Sign Convention
Magnet power	TODO	TODO	TODO	TODO	TODO	TODO
TEC power	TODO	TODO	TODO	TODO	TODO	TODO
Temperature	TODO	TODO	TODO	TODO	TODO	TODO
Acceleration	TODO	TODO	TODO	TODO	TODO	TODO

Table 1: Form of packet to PC

Additionally, the aircraft provided its own data as a UDP packet on port 5124, which could be read by the onboard PC so long as it had an IP address of 132.246.192.[25..50]. A breakdown of the packet can be found in “NRC Nav Packet.xlsx”.