ECED3901 Lab #4: MPU-9150 Interface Experiments

Lab Day: June 15, 2015

Lab Due: June 22, 2015 @ 12:30 PM - Submitted via BBLearn Website (PDF files only), OR printed files in 3901 Mail-Slot at ECED Office

Lab Objective

The goals of this lab are to:

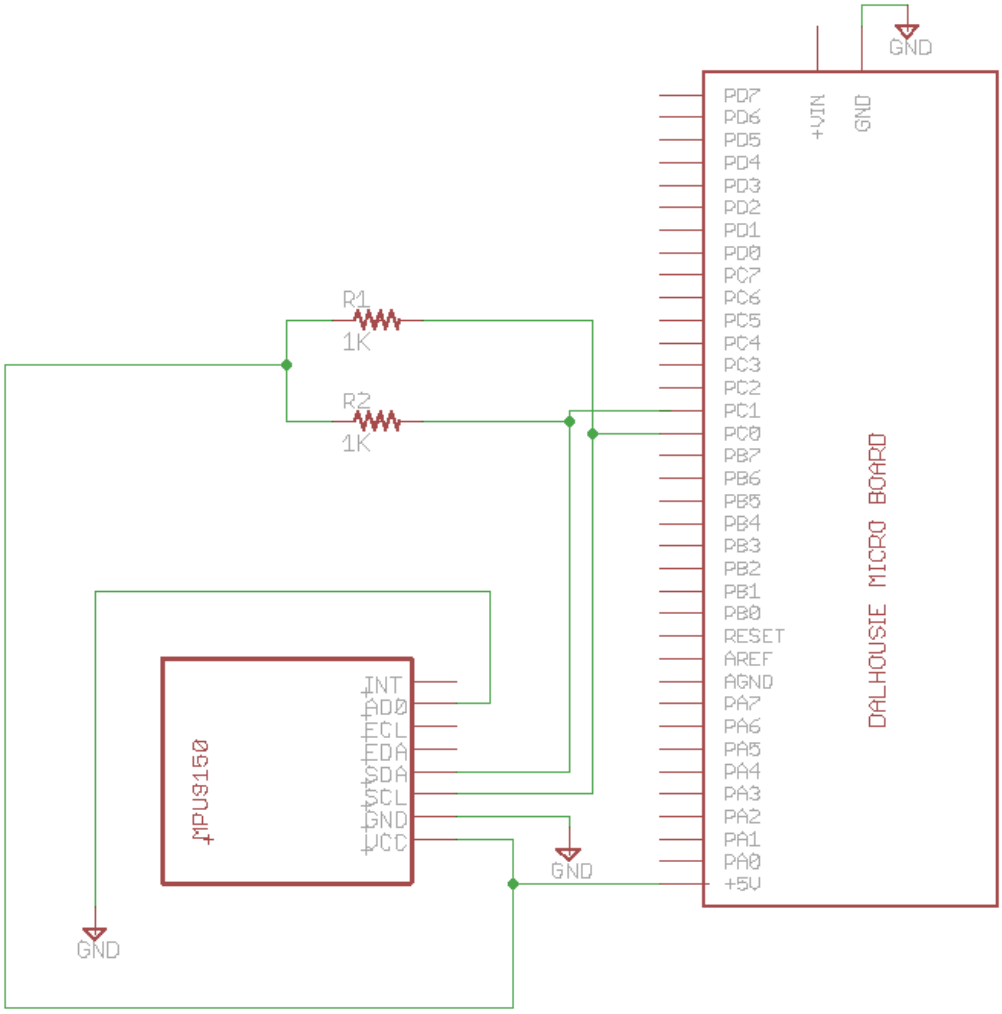
1. Familiarize you with the MPU-9150 Device.
2. Learn about accelerometer, gyros, and magnetic field sensors.
3. Introduce you to plotting and subtracting sensor biases.

Part 1: Interfacing to MPU-9150 and Raw Sensor Readings

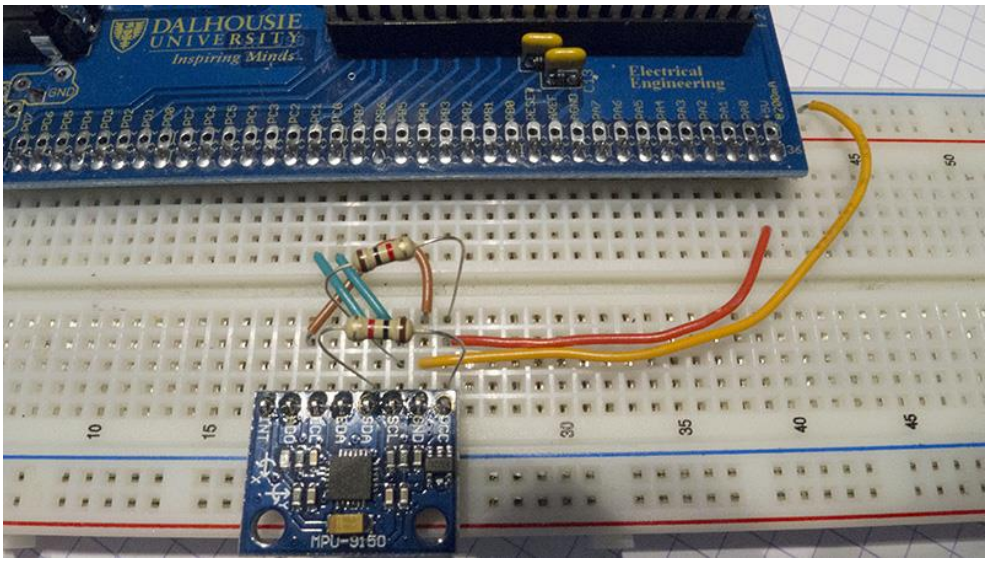
This lab assumes you have completed Lab #2. If not please see the ECED3901 Lab #2, which also references the ECED3204 Lab #1 and Lab #5 (available at https://github.com/colinoflynn/eced3204/blob/master/labs).

Setup / Procedure

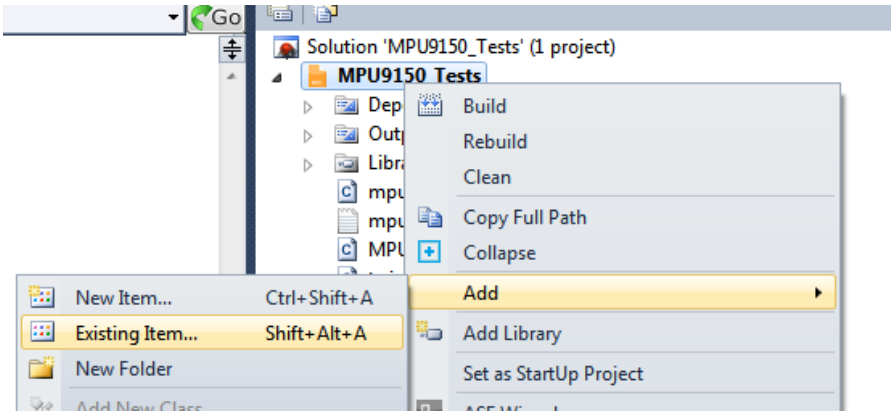
1. You will need to solder the header onto the MPU-9150 board.
2. Mount the MPU-9150 in a breadboard, and wire the following circuit up:



Which might look like this:



3. Generate a new Atmel Studio project, setting the usual ATMega644P setting.
4. In addition to the generated main file (which includes the `main()` function), you will need to download and add several files. You can do this by right-clicking on the solution name and hitting "Add Existing Files":



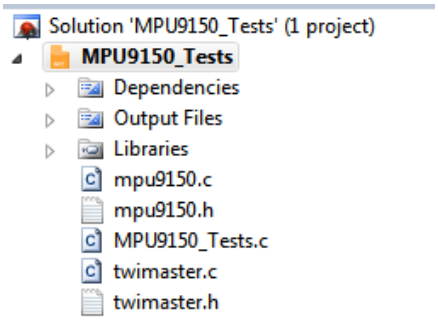
Before doing this, you will need to download the files. You can download the files from one of the following sources:

- BBLearn, called **lab4_part1.zip** and **unzip the files to your project.**
- From the GITHub repo: https://github.com/colinoflynn/eced3901/blob/master/labs_base/lab4_mpu9150/part1_base.zip?raw=true and **unzip the files to your project.**
- Download/view individual files at https://github.com/colinoflynn/eced3901/tree/master/labs_base/lab4_mpu9150/part1_base

The individual files to be added to your project are:

- mpu9150.c
- mpu9150.h
- twimaster.c
- twimaster.h

You should end up with something like this (where MPU9150_Tests.c is my main file, yours may have a different name):



5. In the main file, copy the following to setup the MPU-9150 sensor:

```
#include <stdio.h>
```

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#define F_CPU 14745600UL
#include <util/delay.h>
#include "twimaster.h"
#include "mpu9150.h"

static int uart_putchar(char c, FILE *stream);
static int uart_getchar(FILE *stream);
FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_WRITE);
FILE mystdin = FDEV_SETUP_STREAM(NULL, uart_getchar, _FDEV_SETUP_READ);

static int uart_putchar(char c, FILE *stream)
{
   loop_until_bit_is_set(UCSR0A, UDRE0);
   UDR0 = c;
   return 0;
}

static int uart_getchar(FILE *stream)
{
   loop_until_bit_is_set(UCSR0A, RXC0); /* Wait until data exists. */
   return UDR0;
}

void init_uart(void)
{
   UCSR0B = (1<<RXEN0) | (1<<TXEN0);
   UBRR0 = 7;
   stdout = &mystdout;
   stdin = &mystdin;
}

int main(void)
{
   init_uart();
   printf_P(PSTR("System Booted, built %s on %s\n"), __TIME__, __DATE__);

   i2c_init();

   printf_P(PSTR("MPU9150: Attempting Init Call.. if system hangs here check
I2C connections\n"));
   Initalise_AccelGyro(ACCELRANGE_2g, GYRORANGE_250DPS);
   printf_P(PSTR("MPU9150: Init Done\n"));

   int xacc, yacc, zacc;
   int xgyr, ygyr, zgyr;
   int xmag, ymag, zmag;

   while(1){

          Trigger_Compass();

          xacc = Read_Acc_Gyro(ACCEL_XOUT_H);
          yacc = Read_Acc_Gyro(ACCEL_YOUT_H);
          zacc = Read_Acc_Gyro(ACCEL_ZOUT_H);

          xgyr = Read_Acc_Gyro(GYRO_XOUT_H);
          ygyr = Read_Acc_Gyro(GYRO_YOUT_H);
          zgyr = Read_Acc_Gyro(GYRO_ZOUT_H);

          xmag = Read_Compass(COMP_XOUT_L);
          ymag = Read_Compass(COMP_YOUT_L);
```

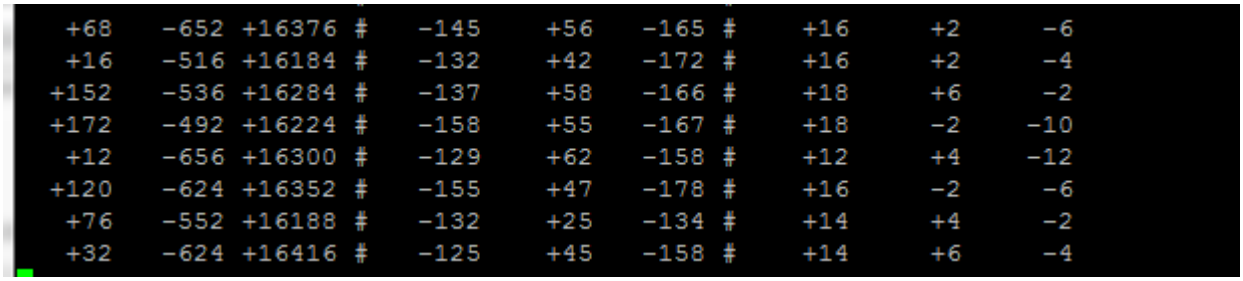
```
            zmag = Read_Compass(COMP_ZOUT_L);

            _delay_ms(200);

            printf("%+6d %+6d %+6d # %+6d %+6d %+6d # %+6d %+6d %+6d\n", xacc,
yacc, zacc, xgyr, ygyr, zgyr, xmag, ymag, zmag);
   }
}
```

6. Configure a terminal emulator (such as putty or termite), as in previous labs. See Lab #2 if you need a refresher on terminal emulator setup!

   Program the device, and it should start reading off sensor values. If the system hangs check your wiring and soldering, as it might not be communicating with the sensor.



Lab Part 1 Questions/Observations

1. Each line contains printed sensor readings. What is the order of sensor readings within the line? You can determine this from the source code.

2. Based on the datasheet and the default configuration used here, the following is maximum values and scale factors for each sensor type.

   Accelerometer: ±32768 counts = ±2g
   Gyro: ±32768 counts = ±250 degrees/seconds
   Magnetometer: ±4096 counts = ±1229 µT

   To convert these to scale factors, simply divide the measurement with units into ‘counts’. For example for the accelerometer:

   Scale = 2g / 32768 count = 0.00006104 g/count

   To use this scale factor, you can simply multiple a reading by that scale factor to get the final reading. Note the following about the units:

   Accelerometer: The units of ‘1g’ means the force of gravity, which is an acceleration of 9.8m/s^2.

   Gyro: These units are degrees/second of angular velocity.

Magnetometer: The units of microteslas are a measure of the *magnetic flux*. The Earth's magnetic field has an approximate strength of around 52 microteslas here in Nova Scotia (the field strength varies on the Earth's surface).

Finally… using the scale factors, fill in the following table to show the readings from your unit (NOTE: they will bounce around a little, simply select one line that seems reasonable):

	Accelerometer			Gyroscope			Magnetometer		
	X	**Y**	**Z**	**X**	**Y**	**Z**	**X**	**Y**	**Z**
Raw Reading									
Scaled Reading									

Part 2: Measuring Gyroscope Bias and Integration

This part of the lab will demonstrate how to measure the Gyroscope 'bias', allowing you to use it for measuring rotational angle.

In Part #1 you should have noticed the fairly large readings from the gyroscope, even when the device is not moving. This will quickly give you an extreme amount of error, but by removing the bias we can use the device freely over a few-minute period.

Procedure

1. Using the code from Part #1 as a base, insert code to measure the average bias over a few measurement cycles. An example of such a chunk of code is the following. This would run before the main while(1) loop:

```
… rest of original code here…
int xmag, ymag, zmag;

printf_P(PSTR("DO NOT MOVE DEVICE\n"));

xgyr = 0;
ygyr = 0;
zgyr = 0;

for(int i = 0; i < 128; i++){
   xgyr += Read_Acc_Gyro(GYRO_XOUT_H);
   ygyr += Read_Acc_Gyro(GYRO_YOUT_H);
   zgyr += Read_Acc_Gyro(GYRO_ZOUT_H);
   _delay_ms(50);
}

int xgyrbias = xgyr / 128;
int ygyrbias = ygyr / 128;
int zgyrbias = zgyr / 128;
```

```
printf_P(PSTR("Cal Done: %d %d %d\n"), xgyrbias, ygyrbias, zgyrbias);
while(1){
… rest of original code here…
```

2. You can then modify the main while(1) loop to remove the calculated bias on every measurement:

```
xgyr = Read_Acc_Gyro(GYRO_XOUT_H) - xgyrbias;
ygyr = Read_Acc_Gyro(GYRO_YOUT_H) - ygyrbias;
zgyr = Read_Acc_Gyro(GYRO_ZOUT_H) - zgyrbias;
```

3. Finally, we can *integrate* the measurement by using a new variable (which will need to be a larger variable such as an int32_t or int64_t to avoid overflow errors). The following shows an example of the main loop which performs this integration:

```
int32_t xgyracc = 0;
int32_t ygyracc = 0;
int32_t zgyracc = 0;

while(1){
          Trigger_Compass();

          xacc = Read_Acc_Gyro(ACCEL_XOUT_H);
          yacc = Read_Acc_Gyro(ACCEL_YOUT_H);
          zacc = Read_Acc_Gyro(ACCEL_ZOUT_H);

          xgyr = Read_Acc_Gyro(GYRO_XOUT_H) - xgyrbias;
          ygyr = Read_Acc_Gyro(GYRO_YOUT_H) - ygyrbias;
          zgyr = Read_Acc_Gyro(GYRO_ZOUT_H) - zgyrbias;

          xmag = Read_Compass(COMP_XOUT_L);
          ymag = Read_Compass(COMP_YOUT_L);
          zmag = Read_Compass(COMP_ZOUT_L);

          xgyracc += xgyr;
          ygyracc += ygyr;
          zgyracc += zgyr;

          _delay_ms(200);
          printf("%+6d %+6d %+6d\n",
          (int)((float)xgyracc*0.2*(250.0/32768.0)),
          (int)((float)ygyracc*0.2*(250.0/32768.0)),
          (int)((float)zgyracc*0.2*(250.0/32768.0))
          );
    }
```

4. Download the program, and try rotating the breadboard. You should see the measurement of the angle corresponding to your movement of the device… for example rotate the device 90 degrees, and check the Z-Axis rotation measurement is approximately 90 degrees.

5. In the conversion of count to angular measurement we have the following conversion:

   0.2*(250.0/32768.0)

   Where 0.2 is the integration interval (200mS or 0.2S). Note this is actually slightly wrong. The code does have a 200mS delay, but there is additional time when the device is performing the

printf() statement and calculations. Try measuring the actual delay be toggling a port pin on each loop through.

6. Change the `int32_t` for the declaration of `zgyracc` to an `int16_t`, and try the rotation again. What happens when you rotate 90 or 180 degrees? Why do you think this happens?

Lab Part 2 Questions/Observations

1. What happens when you changed the int32_t declaration for the accumulation register? Why do you think this happened?
2. Leave the gyro on the table for some time (>2 mins) and measure the drift. This is to say even if you don't move the gyro, how many degrees does it think you have rotated it?

Part 3: Measure Magnetic Sensor Bias

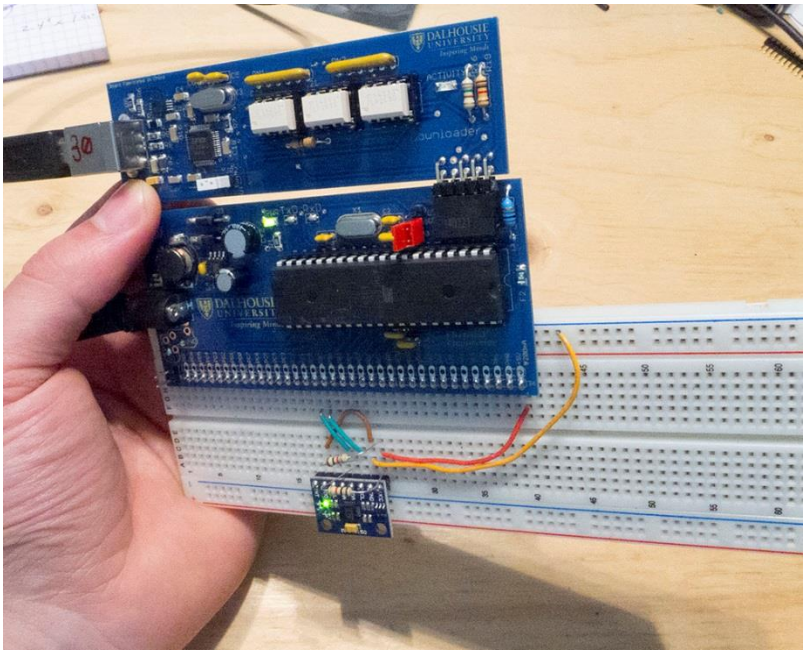
In this part of the lab we will measure the magnetic sensor bias. This allows you to use the magnetic sensor for a compassing application.

Procedure

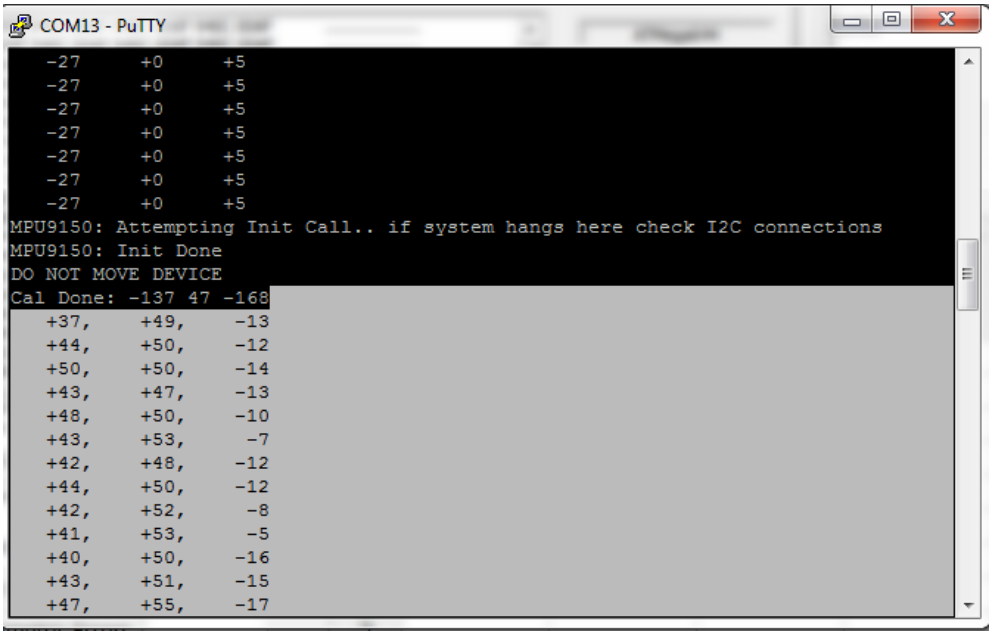
1. The magnetic sensor often has a constant positive offset. We will first visualize the magnetic field sensor.
2. Using the previous code as a base, simply print the magnetic field readings in X,Y,Z axis to the console. Use a comma between readings to allow easier importing to our plotting service. The following shows an example of such a print statement:

```
printf("%+6d, %+6d, %+6d\n", xmag, ymag, zmag);
```

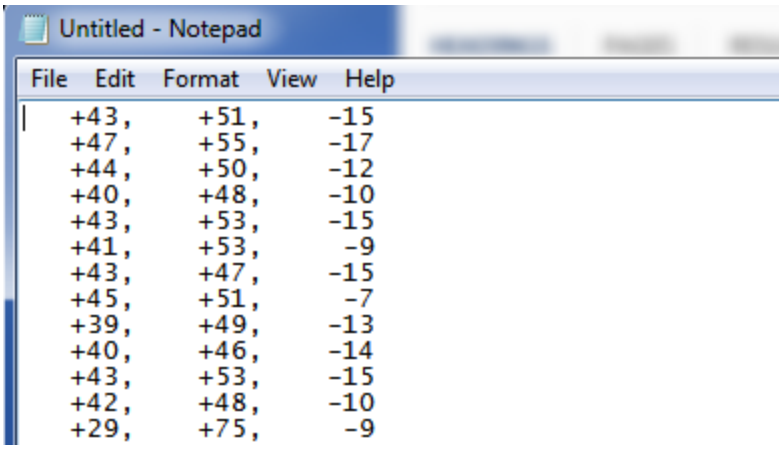
3. Once this is programmed, carefully download and move the sensor around. Rotate it in each direction, but be careful to keep the downloader (which has the serial communication) connected:



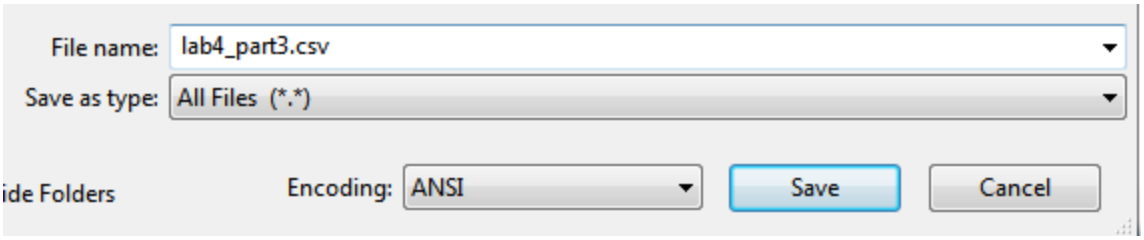
4. Copy the readings from the console into a text file. Note on Putty you can highlight the readings, and right-clicking will automatically copy them to the clipboard. You can unplug the downloader to stop new readings from coming in (which will prevent you from scrolling):



Then paste them into a text document (i.e. as with Notepad):



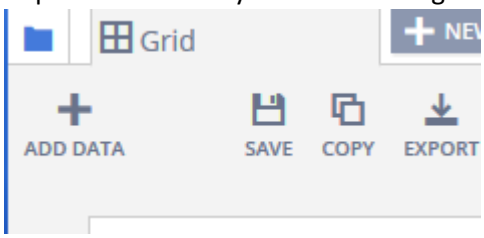
Finally save this as a .csv file, i.e. here is Lab4_Part3.csv:



5. Perform a 3-D scatter plot. You can use a variety of tools for this (such as MATLAB), but in case there is no such tool installed on your computer I'll take you through the use of http://plot.ly web service:

    a. Register for the https://plot.ly service using your email address.
    b. Create a new project:

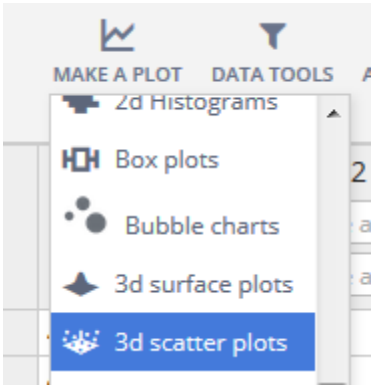
    c. Import the .csv file you created using the "Add Data" button:



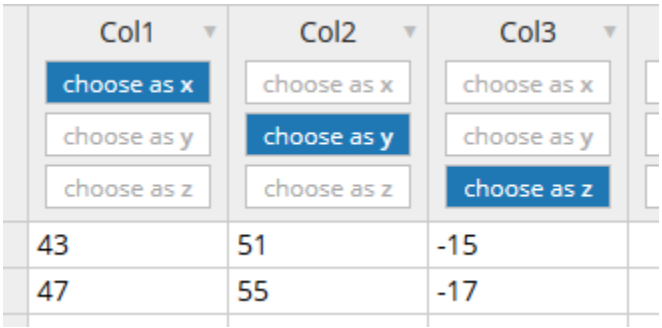
    d. Hopefully your data is imported successfully:

	Col1	Col2	Col3
x	choose as x	choose as x	choose as x
y	choose as y	choose as y	choose as y
1	43	51	-15
2	47	55	-17
3	44	50	-12
4	40	48	-10
5	43	53	-15
6	41	53	-9
7	43	47	-15
8	45	51	-7
9	39	49	-13
10	40	46	-14
11	43	53	-15
12	42	48	-10
13	29	75	-9
14	10	106	-10

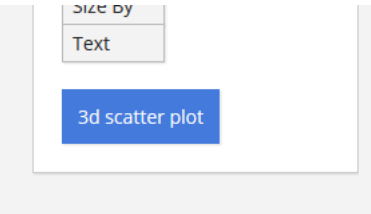
e. Under the “Make a Plot” menu select the 3D Scatterplot option:



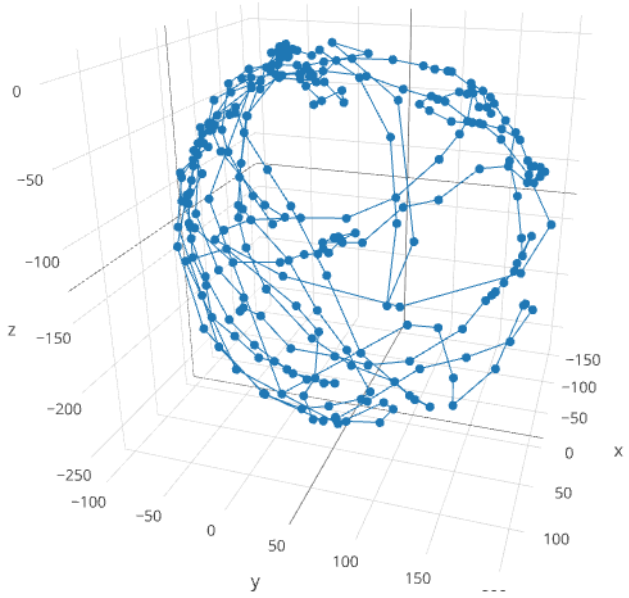
f. Ensure you select the ‘x’, ‘y’, and ‘z’ columns:



g. Hit the 3d scatter plot button, and you should get a plot.



h. You can move the 3D plot around. You should get a spherical looking object – if not you might have not moved the device around enough:



6. The 3D scatter plot *should* be a sphere centered on 0,0,0. You sphere is almost certainly *not* centered on 0,0,0 – you could read an approximate center location from the 3D scatter plot. The next section will determine the most likely center location for each of the x,y,z axis. Save a copy/screen-shot of this sphere for your results.

7. Based on the previous code, you could use something like the following to plot the magnetometer bias:

```
int16_t mag_max[3] =  {0, 0, 0};
int16_t mag_min[3] =  {0, 0, 0};
int16_t mag_temp[3] = {0, 0, 0};
int16_t mag_bias[3] = {0, 0, 0};

while(1){
      Trigger_Compass();

      mag_temp[0] = Read_Compass(COMP_XOUT_L);
      mag_temp[1] = Read_Compass(COMP_YOUT_L);
      mag_temp[2] = Read_Compass(COMP_ZOUT_L);

      for (int jj = 0; jj < 3; jj++) {
```

```
                if(mag_temp[jj] > mag_max[jj]) mag_max[jj] = mag_temp[jj];
                if(mag_temp[jj] < mag_min[jj]) mag_min[jj] = mag_temp[jj];
            }

            mag_bias[0] = (mag_max[0] + mag_min[0])/2;
            mag_bias[1] = (mag_max[1] + mag_min[1])/2;
            mag_bias[2] = (mag_max[2] + mag_min[2])/2;
            printf("Bias: %5d %5d %5d\n", mag_bias[0], mag_bias[1],
mag_bias[2]);
    }
```

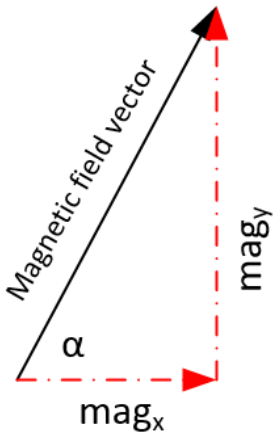
8. As you move the board around, new maximum bias values are calculated. Continue moving the board in every direction/axis until the bias values no longer change, indicating a good estimate of the bias was found for the X, Y, and Z direction.

9. Returning to Step #2, modify your readings to subtract the bias. For example we can subtract them as follows:

   xmag = Read_Compass(COMP_XOUT_L) - (X_BIAS);
   ymag = Read_Compass(COMP_YOUT_L) - (Y_BIAS);
   zmag = Read_Compass(COMP_ZOUT_L) - (Z_BIAS);

10. Re-plot the 3D figure by again moving the device around while plotting the X,Y, and Z axis, which should be now center around 0,0,0. Again save a copy for your lab report.

11. Finally, we will convert the X & Y magnetometer readings into a *compass heading*. The problem can be visualized as using the x and y components of the field vector to determine the angle α in the following figure:



   Which can be accomplished with the following code:

```
azmith = atan((float)ymag / (float)xmag);
azmith *= (180.0 / M_PI);
printf("Heading: %d\n", (int)azmith);
```

Print the heading using the above code as you rotate the sensor through all 360 degrees. You will notice it does not correspond to compass headings (which range from 0 – 360 degrees), but has various “breaks” due to the arctan() function response.

Be careful to keep the breadboard flat on your desk as you rotate it. Tilting the module will cause incorrect results.

You can solve the problem of jumps in the readings by using absolute values for the arctan() input, and manually “rotating” the output of the arctan() function into one of the four possible output quadrants. This will be done as part of Assignment #4, but you might consider how to perform this in your code and testing it during the lab.

Lab Part 3 Questions/Observations

1. Include your 3D scatter plot of magnetometer response without bias compensation

2. Include your 3D scatter plot of magnetometer response with bias compensation.

3. Include a drawing or note of the responses of the heading calculation for various rotations. For example an easy way to accomplish this is to mark readings on the following diagram as you rotate the module: