Trevor Smith, A11051971
Ryan Collins, A11739931

Lab 3 Report - The IMU, the Gimbals, and the Motor

Introduction

In this lab we experimented with the LSM9DS0 IMU breakout board, reading analog values from a gimbal, and driving a motor through a MOSFET using a PWM output.

Analysis

Communicating with the IMU

The IMU library was added to the Arduino Library directory. We connected the wires (3.3v, GND, SCL, SDA) from the IMU to the corresponding pins on the ATmega128rfa1.

Using the lsm9doftest code provided in the Lab 3 repository as a reference, we wrote a program to display the IMU sensor data on the serial port. The sensor values are integer parameters of a structure named lsm. They are accessed with lsm.read() and printed to the serial port.

The data was saved as comma delimited values, imported to excel, and plotted. The three sensors and their corresponding x, y, z axis values are displayed in figures 1, 2, and 3.

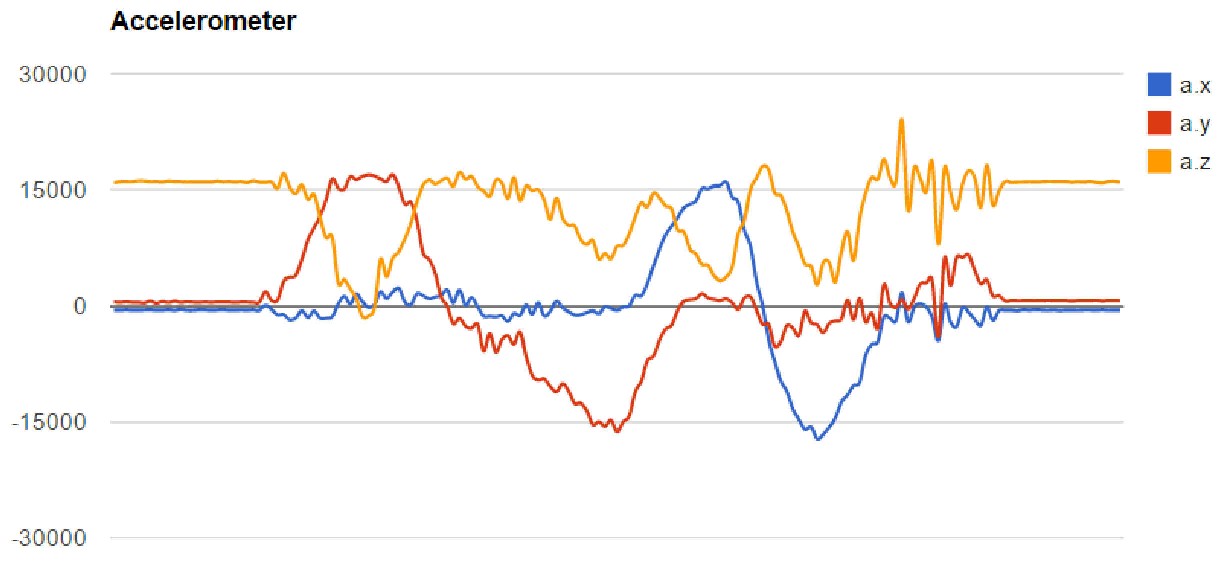

Figure 1 - Accelerometer data plotted in time.

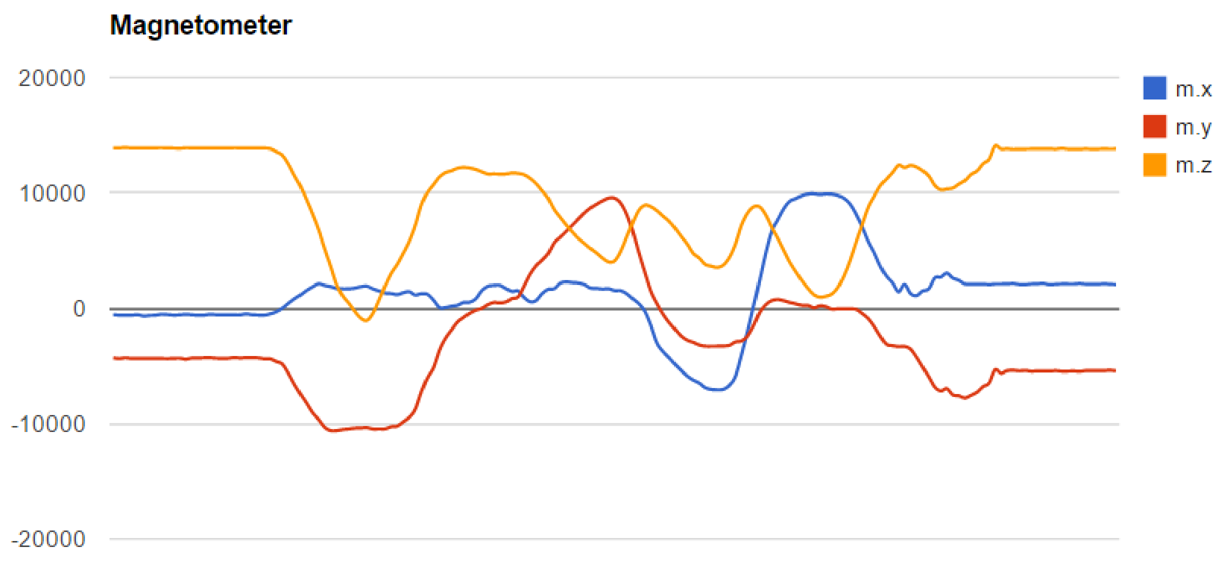

Figure 2 - Magnetometer data plotted in time.

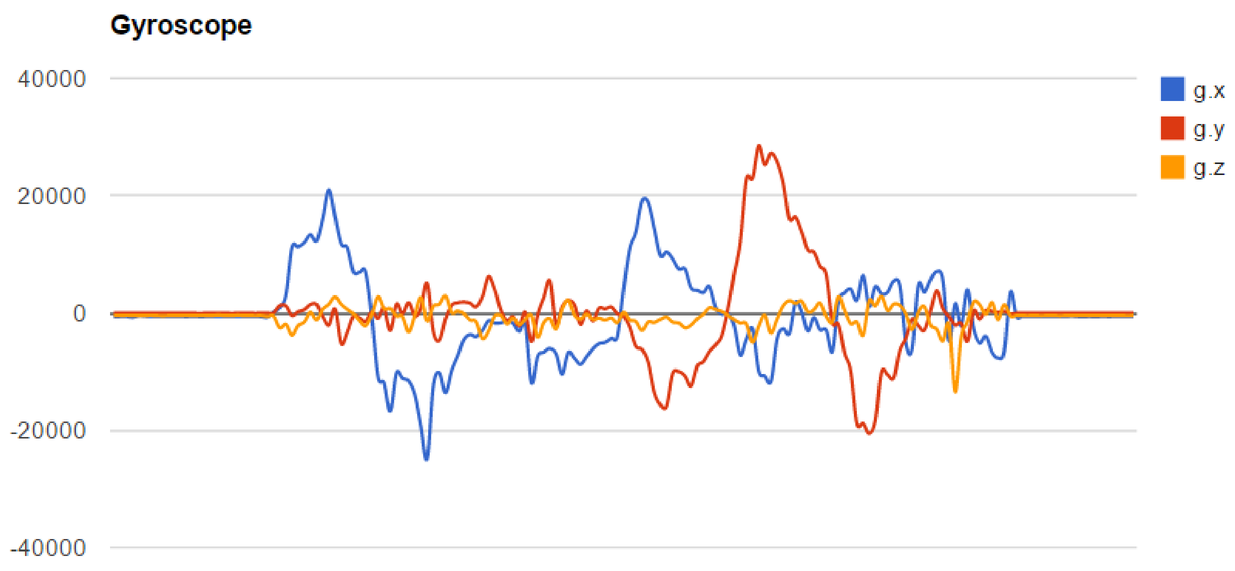

Figure 3 - Gyroscope data plotted in time.

Interfacing the Gimbal

The Gimbal works by using a potentiometer to divide voltage which can be read by an analog input of the microcontroller. If connecting the full range of the gimbal from 3.3V to GND, the output will give a range between .761V and 2.50V. The default reference for analog pins is 1.6V, so we put a potentiometer in series with the gimbal in order to bring the max gimbal voltage to 1.6V.

The gimbal was set up as follows:

	Amin	Amax	Pot R (Ω)
Vertical	270	1023	2.87k
Horizontal	211	1020	3.55k

As an alternative to using the potentiometer, you can configure the Arduino to use the external analog reference (AREF pin) set to a voltage greater than or equal to the largest value from the gimbal. We were able to connect the AREF pin to the 3.3V line, which gave us a range of analog values from about 230 to 770.

Also of note, even after adding the potentiometer to the circuit, we were unable to use the full range of the analog input because the voltage across the gimbal never drops below .761V. We were unable to think of a way to solve this other than by using a voltage lower than GND at the negative terminal of the gimbal.

Driving the Motor

The motor was set up so that the gimbal input controlled the speed of the motor. In order to control the speed of the motor, we needed to use a PWM output since we don't have a true analog output from the board. The PWM pin cannot source enough current to the motor alone, so instead it drives the mosfet, which acts as a switch between the motor and 3.3V.

A flyback diode is used to protect the mosfet from voltage surges induced by the motor. A pulldown resistor is also used between the gate and source to ensure the mosfet gate is default GND.

The analog input from the gimbal is a 10 bit number and the PWM only accepts 8 bit values. In order to fix this, we mapped the 1023 possible analog inputs to a range of 0 to 255 (corresponding to an 8 bit value). Then we could control the speed of the motor using the gimbal. Figure 4 shows the setup.

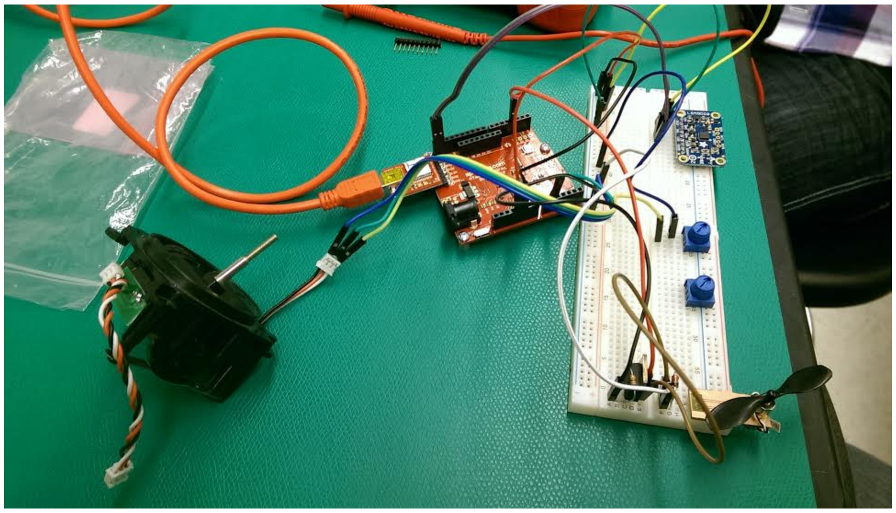

Figure 4 - The gimbal attached to the ATmega, controlling the motor using PWM.

We encountered strange behavior when displaying the input values from the gimbal while the motor was running. The analog input was no longer a steady value, it would jump to a higher or lower value at some fixed frequency. The maximum and minimum also changed from around 1020 to around 800. We are uncertain of the exact source but think it may be a fluctuation in the reference voltage due to the high current draw of the motor.