MPU-9250 Hookup Guide

Introduction

The MPU-9250 is the latest 9-axis MEMS sensor from InvenSense®. This replaces the popular EOL'd MPU-9150. InvenSense® lowered power consumption and decreased the size by 44% compared to the MPU-9150. "Gyro noise performance is 3x better, and compass full scale range is over 4x better than competitive offerings." The MPU-9250 uses 16-bit analog-to-digital converters (ADCs) for digitizing all 9 axes.

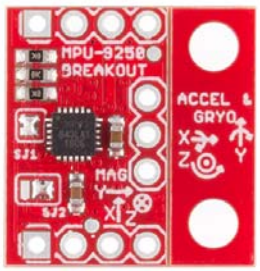
The **S**ystem **i**n **P**ackage (SiP) combines two chips: the MPU-6500, which contains a 3-axis gyroscope, a 3-axis accelerometer, and the AK8963, a 3-axis magnetometer.

Suggested Reading

Before getting started, you may find the following links useful:

- I²C Protocol
- Logic Levels
- Installing an Arduino Library
- What are Pull-up Resistors?
- How to use a Breadbaord

Board Overview



The MPU-9250 Breakout as you will receive it

The board is designed to be smaller than some of our other offerings to fit in smaller projects. To achieve this, the PTHs are wrapped around the boarder of the PCB in three rows of three or four. The top row (J1) is all one need to get most of the functionality of the IMU. These include the I²C and power interface. If space were really tight, one could take a saw and carefully remove all of the other PTHs.

The second most likely to be used set of PTHs are found along the bottom (J3). This includes the address pin, the interrupt pin, and the IO voltage supply for easy interface with a more modern 1.8V processor.

The third, non-breadboard-compatible row (J2) is used for features like running other I²C devices as slaves to this one. For prototyping with these connections, throw your connections on top like you would with an Arduino Pro Mini or similar product.

PTH Connections

The following table summarizes all of the plated through hole (PTH) connections on the breakout board in order found on the board stating in the upper-left corner and wrapping clockwise:

Pin Label	Pin Function	Notes
SCL	I²C serial clock SPI serial port clock	100 or 400 kHz I²C Up to 1 MHz SPI (20 MHz in certain cases)
SDA	I²C serial data	Can also be used for SPI serial data input (SDI)
VDD	Power supply	+2.4V to +3.6V
GND	Ground reference	+0V
AUXDA	Ground reference	I²C master serial data, for connecting to external sensors
FSYNC	Ground reference	Frame synchronization digital input. Connect to GND if unused.
AUXCL	Ground reference	I²C Master serial clock, for connecting to external sensors
INT	Interrupt signal	Interrupt digital output (totem pole or open-drain)
$\overline{\text{CS}}$	Chip select	Chip select (SPI mode only)
VDDIO	Power supply for I/O pins	+1.71V up to VDD

Pin Label	Pin Function	Notes
AD0/ SDO	Address selection	I²C Slave Address LSB (AD0): Low: 0b1101000 ⇨ 0x68 High: 0b1101001 ⇨ 0x69 SPI serial data output (SDO)

Jumpers

The MPU-9250 Breakout has two solder jumpers, SJ1 and SJ2.

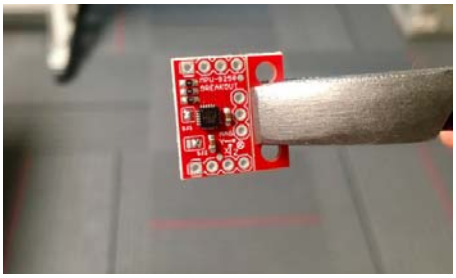
SJ1 comes pre-soldered to short V_{DD} and V_{DDIO}. This reduces the number of power supplies to one with out requiring an external jumper. If the core and IO need to be supplied with different voltages, remove the solder from SJ1.

SJ2 is a two way jumper that comes pre-soldered to connect AD0 to ground. This sets the I²C address to 0x68. It also leaves the PTH for AD0 disconnected and floating. If the solder is moved to connect the center pad with the pad on the left, then the AD0 PTH needs to be connected high or low to chose the I²C address.

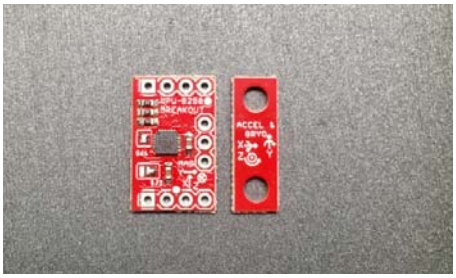
Reducing size

As stated earlier, one of the design goals for this breakout was to make the board small. Some projects will require mounting holes, so we threw them on the right side of some v-score on this board. Since the board is only ⅔“ wide and there isn’t enough mass to the left of the mounting holes, there isn’t much of a bending moment.

If you plan to use a breadboard, or secure the IMU securely to a project with something like epoxy, the mounting holes can be snapped off. As shown in the following image. The pliers I had on hand made super easy work of this. The edge of a table should work fine too.



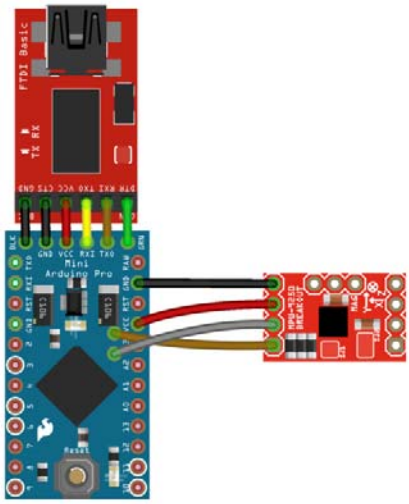
Board was held with pliers and easily broke hand pressing the other side



PCB snapped in two parts

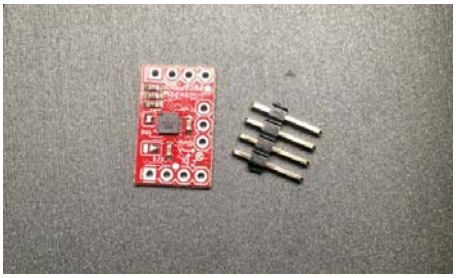
Hardware Connections

The MPU-9250 breakout board runs on 3.3 VDC, so a 3.3V USB to UART bridge such as the SparkFun FTDI Basic Breakout - 3.3V or the SparkFun Beefy 3 - FTDI Basic Breakout can be used to power and bridge communication with a micro controller. In this case an Arduino Pro Mini 328 - 3.3V/8MHz was chosen so logic level translation isn't needed.



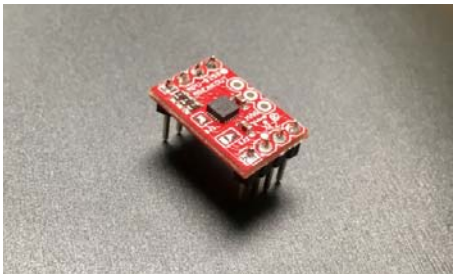
Fritzing diagram of setup

Only 4 connections are needed for I²C communication.



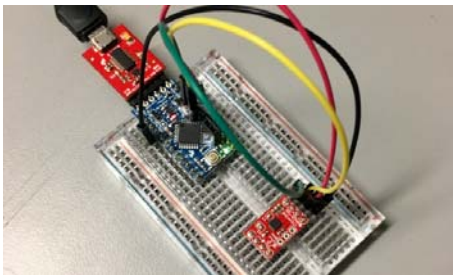
Minimum parts for a breadboard compatible setup

For stability in the breadboard, another four pins were soldered on: V_{DDIO}, AD0/SDO, $\overline{CS}$, and INT.



PCB with breadboard compatible male headers soldered on

Here is the final setup used for testing.



Setup used for testing

Library and Example Code

The example sketch and library are *HEAVILY* based on the work of Kris Winer. His original work can be found here on GitHub. Our version is mostly a conversion to make it follow the Arduino library format. This is a link to information on the algorithms Kris used for the **a**ttitude and **h**eading **r**eference **s**ystem (AHRS).

To install our library manually grab a copy here, or just use the library manager as detailed in this tutorial.

```
/* MPU9250 Basic Example Code
 by: Kris Winer
 date: April 1, 2014
 license: Beerware - Use this code however you'd like. If you
 find it useful you can buy me a beer some time.
 Modified by Brent Wilkins July 19, 2016

 Demonstrate basic MPU-9250 functionality including parameteri
zing the register
 addresses, initializing the sensor, getting properly scaled a
ccelerometer,
 gyroscope, and magnetometer data out. Added display function
s to allow display
 to on breadboard monitor. Addition of 9 DoF sensor fusion usi
ng open source
 Madgwick and Mahony filter algorithms. Sketch runs on the 3.
3 V 8 MHz Pro Mini
 and the Teensy 3.1.

 SDA and SCL should have external pull-up resistors (to 3.3V).
 10k resistors are on the EMSENSR-9250 breakout board.

 Hardware setup:
 MPU9250 Breakout --------- Arduino
 VDD ---------------------- 3.3V
 VDDI --------------------- 3.3V
 SDA ---------------------- A4
 SCL ---------------------- A5
 GND ---------------------- GND
 */

#include "quaternionFilters.h"
#include "MPU9250.h"

#ifdef LCD
#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>

// Using NOKIA 5110 monochrome 84 x 48 pixel display
// pin 9 - Serial clock out (SCLK)
// pin 8 - Serial data out (DIN)
// pin 7 - Data/Command select (D/C)
// pin 5 - LCD chip select (CS)
// pin 6 - LCD reset (RST)
Adafruit_PCD8544 display = Adafruit_PCD8544(9, 8, 7, 5, 6);
#endif // LCD

#define AHRS true         // Set to false for basic data read
#define SerialDebug true  // Set to true to get Serial output
for debugging

// Pin definitions
int intPin = 12;  // These can be changed, 2 and 3 are the Ard
uinos ext int pins
int myLed  = 13;  // Set up pin 13 led for toggling

MPU9250 myIMU;

void setup()
{
  Wire.begin();
  // TWBR = 12;  // 400 kbit/sec I2C speed
  Serial.begin(38400);
```

```
  // Set up the interrupt pin, its set as active high, push-pu
ll
  pinMode(intPin, INPUT);
  digitalWrite(intPin, LOW);
  pinMode(myLed, OUTPUT);
  digitalWrite(myLed, HIGH);

#ifdef LCD
  display.begin(); // Ini8ialize the display
  display.setContrast(58); // Set the contrast

  // Start device display with ID of sensor
  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(0,0); display.print("MPU9250");
  display.setTextSize(1);
  display.setCursor(0, 20); display.print("9-DOF 16-bit");
  display.setCursor(0, 30); display.print("motion sensor");
  display.setCursor(20,40); display.print("60 ug LSB");
  display.display();
  delay(1000);

  // Set up for data display
  display.setTextSize(1); // Set text size to normal, 2 is twi
ce normal etc.
  display.setTextColor(BLACK); // Set pixel color; 1 on the mo
nochrome screen
  display.clearDisplay();   // clears the screen and buffer
#endif // LCD

  // Read the WHO_AM_I register, this is a good test of commun
ication
  byte c = myIMU.readByte(MPU9250_ADDRESS, WHO_AM_I_MPU9250);
  Serial.print("MPU9250 "); Serial.print("I AM "); Serial.prin
t(c, HEX);
  Serial.print(" I should be "); Serial.println(0x71, HEX);

#ifdef LCD
  display.setCursor(20,0); display.print("MPU9250");
  display.setCursor(0,10); display.print("I AM");
  display.setCursor(0,20); display.print(c, HEX);
  display.setCursor(0,30); display.print("I Should Be");
  display.setCursor(0,40); display.print(0x71, HEX);
  display.display();
  delay(1000);
#endif // LCD

  if (c == 0x71) // WHO_AM_I should always be 0x68
  {
    Serial.println("MPU9250 is online...");

    // Start by performing self test and reporting values
    myIMU.MPU9250SelfTest(myIMU.SelfTest);
    Serial.print("x-axis self test: acceleration trim withi
n : ");
    Serial.print(myIMU.SelfTest[0],1); Serial.println("% of fa
ctory value");
    Serial.print("y-axis self test: acceleration trim withi
n : ");
    Serial.print(myIMU.SelfTest[1],1); Serial.println("% of fa
ctory value");
    Serial.print("z-axis self test: acceleration trim withi
n : ");
    Serial.print(myIMU.SelfTest[2],1); Serial.println("% of fa
ctory value");
```

```
    Serial.print("x-axis self test: gyration trim within : ");
    Serial.print(myIMU.SelfTest[3],1); Serial.println("% of fa
ctory value");
    Serial.print("y-axis self test: gyration trim within : ");
    Serial.print(myIMU.SelfTest[4],1); Serial.println("% of fa
ctory value");
    Serial.print("z-axis self test: gyration trim within : ");
    Serial.print(myIMU.SelfTest[5],1); Serial.println("% of fa
ctory value");

    // Calibrate gyro and accelerometers, load biases in bias
registers
    myIMU.calibrateMPU9250(myIMU.gyroBias, myIMU.accelBias);

#ifdef LCD
    display.clearDisplay();

    display.setCursor(0, 0); display.print("MPU9250 bias");
    display.setCursor(0, 8); display.print(" x   y   z  ");

    display.setCursor(0,  16); display.print((int)(1000*accelB
ias[0]));
    display.setCursor(24, 16); display.print((int)(1000*accelB
ias[1]));
    display.setCursor(48, 16); display.print((int)(1000*accelB
ias[2]));
    display.setCursor(72, 16); display.print("mg");

    display.setCursor(0,  24); display.print(myIMU.gyroBias
[0], 1);
    display.setCursor(24, 24); display.print(myIMU.gyroBias
[1], 1);
    display.setCursor(48, 24); display.print(myIMU.gyroBias
[2], 1);
    display.setCursor(66, 24); display.print("o/s");

    display.display();
    delay(1000);
#endif // LCD

    myIMU.initMPU9250();
    // Initialize device for active mode read of acclerometer, gyroscope, and
    // temperature
    Serial.println("MPU9250 initialized for active data mode....");

    // Read the WHO_AM_I register of the magnetometer, this is a good test of
    // communication
    byte d = myIMU.readByte(AK8963_ADDRESS, WHO_AM_I_AK8963);
    Serial.print("AK8963 "); Serial.print("I AM "); Serial.print(d, HEX);
    Serial.print(" I should be "); Serial.println(0x48, HEX);

#ifdef LCD
    display.clearDisplay();
    display.setCursor(20,0); display.print("AK8963");
    display.setCursor(0,10); display.print("I AM");
    display.setCursor(0,20); display.print(d, HEX);
    display.setCursor(0,30); display.print("I Should Be");
    display.setCursor(0,40); display.print(0x48, HEX);
    display.display();
    delay(1000);
#endif // LCD
```

```
    // Get magnetometer calibration from AK8963 ROM
    myIMU.initAK8963(myIMU.magCalibration);
    // Initialize device for active mode read of magnetometer
    Serial.println("AK8963 initialized for active data mod
e....");
    if (SerialDebug)
    {
      //  Serial.println("Calibration values: ");
      Serial.print("X-Axis sensitivity adjustment value ");
      Serial.println(myIMU.magCalibration[0], 2);
      Serial.print("Y-Axis sensitivity adjustment value ");
      Serial.println(myIMU.magCalibration[1], 2);
      Serial.print("Z-Axis sensitivity adjustment value ");
      Serial.println(myIMU.magCalibration[2], 2);
    }

#ifdef LCD
    display.clearDisplay();
    display.setCursor(20,0); display.print("AK8963");
    display.setCursor(0,10); display.print("ASAX "); display.s
etCursor(50,10);
    display.print(myIMU.magCalibration[0], 2);
    display.setCursor(0,20); display.print("ASAY "); display.s
etCursor(50,20);
    display.print(myIMU.magCalibration[1], 2);
    display.setCursor(0,30); display.print("ASAZ "); display.s
etCursor(50,30);
    display.print(myIMU.magCalibration[2], 2);
    display.display();
    delay(1000);
#endif // LCD
  } // if (c == 0x71)
  else
  {
    Serial.print("Could not connect to MPU9250: 0x");
    Serial.println(c, HEX);
    while(1) ; // Loop forever if communication doesn't happen
  }
}

void loop()
{
  // If intPin goes high, all data registers have new data
  // On interrupt, check if data ready interrupt
  if (myIMU.readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)
  {
    myIMU.readAccelData(myIMU.accelCount);  // Read the x/y/z
adc values
    myIMU.getAres();

    // Now we'll calculate the accleration value into actual
g's
    // This depends on scale being set
    myIMU.ax = (float)myIMU.accelCount[0]*myIMU.aRes; // - acc
elBias[0];
    myIMU.ay = (float)myIMU.accelCount[1]*myIMU.aRes; // - acc
elBias[1];
    myIMU.az = (float)myIMU.accelCount[2]*myIMU.aRes; // - acc
elBias[2];

    myIMU.readGyroData(myIMU.gyroCount);  // Read the x/y/z ad
c values
    myIMU.getGres();
```

```
    // Calculate the gyro value into actual degrees per second
    // This depends on scale being set
    myIMU.gx = (float)myIMU.gyroCount[0]*myIMU.gRes;
    myIMU.gy = (float)myIMU.gyroCount[1]*myIMU.gRes;
    myIMU.gz = (float)myIMU.gyroCount[2]*myIMU.gRes;

    myIMU.readMagData(myIMU.magCount);  // Read the x/y/z adc
values
    myIMU.getMres();
    // User environmental x-axis correction in milliGauss, sho
uld be
    // automatically calculated
    myIMU.magbias[0] = +470.;
    // User environmental x-axis correction in milliGauss TOD
O axis??
    myIMU.magbias[1] = +120.;
    // User environmental x-axis correction in milliGauss
    myIMU.magbias[2] = +125.;

    // Calculate the magnetometer values in milliGauss
    // Include factory calibration per data sheet and user env
ironmental
    // corrections
    // Get actual magnetometer value, this depends on scale be
ing set
    myIMU.mx = (float)myIMU.magCount[0]*myIMU.mRes*myIMU.magCa
libration[0] -
               myIMU.magbias[0];
    myIMU.my = (float)myIMU.magCount[1]*myIMU.mRes*myIMU.magCa
libration[1] -
               myIMU.magbias[1];
    myIMU.mz = (float)myIMU.magCount[2]*myIMU.mRes*myIMU.magCa
libration[2] -
               myIMU.magbias[2];
  } // if (readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)

  // Must be called before updating quaternions!
  myIMU.updateTime();

  // Sensors x (y)-axis of the accelerometer is aligned with t
he y (x)-axis of
  // the magnetometer; the magnetometer z-axis (+ down) is opp
osite to z-axis
  // (+ up) of accelerometer and gyro! We have to make some al
lowance for this
  // orientationmismatch in feeding the output to the quaterni
on filter. For the
  // MPU-9250, we have chosen a magnetic rotation that keeps t
he sensor forward
  // along the x-axis just like in the LSM9DS0 sensor. This ro
tation can be
  // modified to allow any convenient orientation convention.
This is ok by
  // aircraft orientation standards! Pass gyro rate as rad/s
//  MadgwickQuaternionUpdate(ax, ay, az, gx*PI/180.0f, gy*PI/1
80.0f, gz*PI/180.0f,  my,  mx, mz);
  MahonyQuaternionUpdate(myIMU.ax, myIMU.ay, myIMU.az, myIMU.g
x*DEG_TO_RAD,
                         myIMU.gy*DEG_TO_RAD, myIMU.gz*DEG_TO_
RAD, myIMU.my,
                         myIMU.mx, myIMU.mz, myIMU.deltat);

  if (!AHRS)
  {
    myIMU.delt_t = millis() - myIMU.count;
```

```
    if (myIMU.delt_t > 500)
    {
      if(SerialDebug)
      {
        // Print acceleration values in milligs!
        Serial.print("X-acceleration: "); Serial.print(1000*my
IMU.ax);
        Serial.print(" mg ");
        Serial.print("Y-acceleration: "); Serial.print(1000*my
IMU.ay);
        Serial.print(" mg ");
        Serial.print("Z-acceleration: "); Serial.print(1000*my
IMU.az);
        Serial.println(" mg ");

        // Print gyro values in degree/sec
        Serial.print("X-gyro rate: "); Serial.print(myIMU.gx,
3);
        Serial.print(" degrees/sec ");
        Serial.print("Y-gyro rate: "); Serial.print(myIMU.gy,
3);
        Serial.print(" degrees/sec ");
        Serial.print("Z-gyro rate: "); Serial.print(myIMU.gz,
3);
        Serial.println(" degrees/sec");

        // Print mag values in degree/sec
        Serial.print("X-mag field: "); Serial.print(myIMU.mx);
        Serial.print(" mG ");
        Serial.print("Y-mag field: "); Serial.print(myIMU.my);
        Serial.print(" mG ");
        Serial.print("Z-mag field: "); Serial.print(myIMU.mz);
        Serial.println(" mG");

        myIMU.tempCount = myIMU.readTempData();  // Read the a
dc values
        // Temperature in degrees Centigrade
        myIMU.temperature = ((float) myIMU.tempCount) / 333.8
7 + 21.0;
        // Print temperature in degrees Centigrade
        Serial.print("Temperature is ");  Serial.print(myIMU.t
emperature, 1);
        Serial.println(" degrees C");
      }

#ifdef LCD
      display.clearDisplay();
      display.setCursor(0, 0); display.print("MPU9250/AK896
3");
      display.setCursor(0, 8); display.print(" x   y   z  ");

      display.setCursor(0,  16); display.print((int)(1000*myIM
U.ax));
      display.setCursor(24, 16); display.print((int)(1000*myIM
U.ay));
      display.setCursor(48, 16); display.print((int)(1000*myIM
U.az));
      display.setCursor(72, 16); display.print("mg");

      display.setCursor(0,  24); display.print((int)(myIMU.g
x));
      display.setCursor(24, 24); display.print((int)(myIMU.g
y));
      display.setCursor(48, 24); display.print((int)(myIMU.g
z));
```

```
      display.setCursor(66, 24); display.print("o/s");

      display.setCursor(0,  32); display.print((int)(myIMU.m
x));
      display.setCursor(24, 32); display.print((int)(myIMU.m
y));
      display.setCursor(48, 32); display.print((int)(myIMU.m
z));
      display.setCursor(72, 32); display.print("mG");

      display.setCursor(0,  40); display.print("Gyro T ");
      display.setCursor(50,  40); display.print(myIMU.temperat
ure, 1);
      display.print(" C");
      display.display();
#endif // LCD

      myIMU.count = millis();
      digitalWrite(myLed, !digitalRead(myLed));  // toggle led
    } // if (myIMU.delt_t > 500)
  } // if (!AHRS)
  else
  {
    // Serial print and/or display at 0.5 s rate independent o
f data rates
    myIMU.delt_t = millis() - myIMU.count;

    // update LCD once per half-second independent of read rat
e
    if (myIMU.delt_t > 500)
    {
      if(SerialDebug)
      {
        Serial.print("ax = "); Serial.print((int)1000*myIMU.a
x);
        Serial.print(" ay = "); Serial.print((int)1000*myIMU.a
y);
        Serial.print(" az = "); Serial.print((int)1000*myIMU.a
z);
        Serial.println(" mg");

        Serial.print("gx = "); Serial.print( myIMU.gx, 2);
        Serial.print(" gy = "); Serial.print( myIMU.gy, 2);
        Serial.print(" gz = "); Serial.print( myIMU.gz, 2);
        Serial.println(" deg/s");

        Serial.print("mx = "); Serial.print( (int)myIMU.mx );
        Serial.print(" my = "); Serial.print( (int)myIMU.my );
        Serial.print(" mz = "); Serial.print( (int)myIMU.mz );
        Serial.println(" mG");

        Serial.print("q0 = "); Serial.print(*getQ());
        Serial.print(" qx = "); Serial.print(*(getQ() + 1));
        Serial.print(" qy = "); Serial.print(*(getQ() + 2));
        Serial.print(" qz = "); Serial.println(*(getQ() + 3));
      }

// Define output variables from updated quaternion---these ar
e Tait-Bryan
// angles, commonly used in aircraft orientation. In this coor
dinate system,
// the positive z-axis is down toward Earth. Yaw is the angle
between Sensor
// x-axis and Earth magnetic North (or true North if correcte
d for local
```

```
// declination, looking down on the sensor positive yaw is cou
nterclockwise.
// Pitch is angle between sensor x-axis and Earth ground plan
e, toward the
// Earth is positive, up toward the sky is negative. Roll is a
ngle between
// sensor y-axis and Earth ground plane, y-axis up is positiv
e roll. These
// arise from the definition of the homogeneous rotation matri
x constructed
// from quaternions. Tait-Bryan angles as well as Euler angle
s are
// non-commutative; that is, the get the correct orientation t
he rotations
// must be applied in the correct order which for this configu
ration is yaw,
// pitch, and then roll.
// For more see
// http://en.wikipedia.org/wiki/Conversion_between_quaternions
_and_Euler_angles
// which has additional links.
      myIMU.yaw   = atan2(2.0f * (*(getQ()+1) * *(getQ()+2) +
*getQ() *
                    *(getQ()+3)), *getQ() * *getQ() + *(getQ()
+1) * *(getQ()+1)
                    - *(getQ()+2) * *(getQ()+2) - *(getQ()+3)
* *(getQ()+3));
      myIMU.pitch = -asin(2.0f * (*(getQ()+1) * *(getQ()+3) -
*getQ() *
                    *(getQ()+2)));
      myIMU.roll  = atan2(2.0f * (*getQ() * *(getQ()+1) + *(ge
tQ()+2) *
                    *(getQ()+3)), *getQ() * *getQ() - *(getQ()
+1) * *(getQ()+1)
                    - *(getQ()+2) * *(getQ()+2) + *(getQ()+3)
* *(getQ()+3));
      myIMU.pitch *= RAD_TO_DEG;
      myIMU.yaw   *= RAD_TO_DEG;
      // Declination of SparkFun Electronics (40°05'26.6"N 10
5°11'05.9"W) is
      //    8° 30' E  ± 0° 21' (or 8.5°) on 2016-07-19
      // - http://www.ngdc.noaa.gov/geomag-web/#declination
      myIMU.yaw   -= 8.5;
      myIMU.roll  *= RAD_TO_DEG;

      if(SerialDebug)
      {
        Serial.print("Yaw, Pitch, Roll: ");
        Serial.print(myIMU.yaw, 2);
        Serial.print(", ");
        Serial.print(myIMU.pitch, 2);
        Serial.print(", ");
        Serial.println(myIMU.roll, 2);

        Serial.print("rate = ");
        Serial.print((float)myIMU.sumCount/myIMU.sum, 2);
        Serial.println(" Hz");
      }

#ifdef LCD
      display.clearDisplay();

      display.setCursor(0, 0); display.print(" x   y   z  ");

      display.setCursor(0,  8); display.print((int)(1000*myIM
```

```
U.ax));
      display.setCursor(24, 8); display.print((int)(1000*myIM
U.ay));
      display.setCursor(48, 8); display.print((int)(1000*myIM
U.az));
      display.setCursor(72, 8); display.print("mg");

      display.setCursor(0,  16); display.print((int)(myIMU.g
x));
      display.setCursor(24, 16); display.print((int)(myIMU.g
y));
      display.setCursor(48, 16); display.print((int)(myIMU.g
z));
      display.setCursor(66, 16); display.print("o/s");

      display.setCursor(0,  24); display.print((int)(myIMU.m
x));
      display.setCursor(24, 24); display.print((int)(myIMU.m
y));
      display.setCursor(48, 24); display.print((int)(myIMU.m
z));
      display.setCursor(72, 24); display.print("mG");

      display.setCursor(0,  32); display.print((int)(myIMU.ya
w));
      display.setCursor(24, 32); display.print((int)(myIMU.pit
ch));
      display.setCursor(48, 32); display.print((int)(myIMU.rol
l));
      display.setCursor(66, 32); display.print("ypr");

    // With these settings the filter is updating at a ~145 H
z rate using the
    // Madgwick scheme and >200 Hz using the Mahony scheme eve
n though the
    // display refreshes at only 2 Hz. The filter update rate
is determined
    // mostly by the mathematical steps in the respective algo
rithms, the
    // processor speed (8 MHz for the 3.3V Pro Mini), and the
magnetometer ODR:
    // an ODR of 10 Hz for the magnetometer produce the above
rates, maximum
    // magnetometer ODR of 100 Hz produces filter update rate
s of 36 - 145 and
    // ~38 Hz for the Madgwick and Mahony schemes, respectivel
y. This is
    // presumably because the magnetometer read takes longer t
han the gyro or
    // accelerometer reads. This filter update rate should be
fast enough to
    // maintain accurate platform orientation for stabilizatio
n control of a
    // fast-moving robot or quadcopter. Compare to the update
rate of 200 Hz
    // produced by the on-board Digital Motion Processor of In
vensense's MPU6050
    // 6 DoF and MPU9150 9DoF sensors. The 3.3 V 8 MHz Pro Min
i is doing pretty
    // well!
      display.setCursor(0, 40); display.print("rt: ");
      display.print((float) myIMU.sumCount / myIMU.sum, 2);
      display.print(" Hz");
      display.display();
#endif // LCD
```

```
      myIMU.count = millis();
      myIMU.sumCount = 0;
      myIMU.sum = 0;
    } // if (myIMU.delt_t > 500)
  } // if (AHRS)
}
```

Full demo sketch

Some configuration can be found at the beginning of the sketch. Here is where you can turn on or off AHRS calculations, and serial debugging:

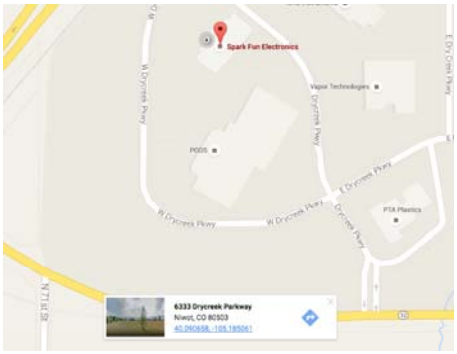
```
#define AHRS true         // Set to false for basic data read
#define SerialDebug true  // Set to true to get Serial output
for debugging
```

Some of the settings used by library exposed in the sketch

Magnetic declination

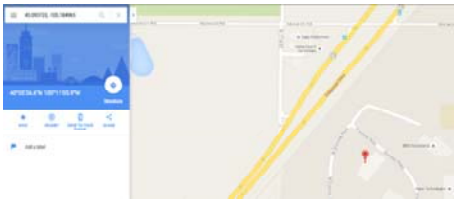
The AHRS needs to know where you are located to convert magnetic north to true north. I used the combination of two services to find the current declination of our offices; Google, and NOAA. This may or may not be ideal in your country.

The first thing I needed were the latitude and longitude of the office. Searching for SparkFun in Google Maps shows a red pin on our building. Clicking near the front door, but not on the existing pin dropped a new pin which also made a card appear bottom center. The bottom of this card contains a link to a search of the latitude and longitude marked by the new pin.



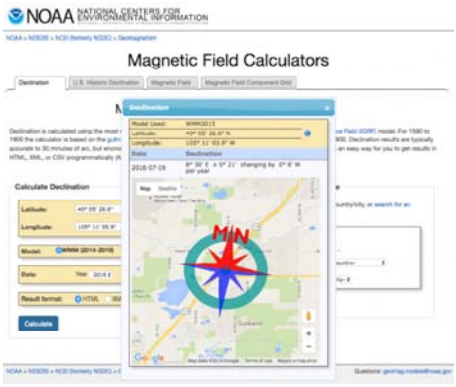
Google Maps showing the latitude and longitude of SparkFun Electronics

Clicking on the link in the card brings up a page showing the latitude and longitude in both the **d**ecimal **d**egrees (DD) and **d**egrees **m**inutes **s**econds (DMS) forms. I find sign errors easier to avoid by using DMS form Google provides with the since I'm not intimately familiar with the WGS 84 **c**oordinate **r**eference **s**ystem (CRS). The included cardinal directions are handy and required by the next tool.



Search for SparkFun Electronics' coordinates

The second tool is the default magnetic field calculator on NOAA's website. Enter the coordinates found in the previous step into the latitude and longitude inputs. The *Calculate* button will trigger a dialog box with the results to appear. If the results appear on the Google map where you are expecting them, then you chose the correct directions with the radio inputs!



Magnetic declination of SparkFun Electronics' Colorado office

Note that the declination here is currently 8˚ 30' E. Also note that the image on the map shows magnetic north to be east of true north. The provided DM format needs to be converted to DD format for the code. There are 60 minutes per degree, 30 of them is $^{30}/_{60}$ of a degree, or 0.5˚. The declination in DD format is thus 8.5˚ E. Update the code in the example sketch around line 391 with the declination for your desired location.

```
myIMU.yaw -= 8.5;
```

The library left a lot of the math in the example sketch in part to make things like this easier to access.

Here is an example of what the output of the demo sketch should look like:

```
MPU9250 I AM 71 I should be 71
MPU9250 is online...
x-axis self test: acceleration trim within : 2.4% of factory value
y-axis self test: acceleration trim within : 0.5% of factory value
z-axis self test: acceleration trim within : -0.0% of factory value
x-axis self test: gyration trim within : -0.3% of factory value
y-axis self test: gyration trim within : -1.0% of factory value
z-axis self test: gyration trim within : 0.5% of factory value
MPU9250 initialized for active data mode....
AK8963 I AM 48 I should be 48
AK8963 initialized for active data mode....
X-Axis sensitivity adjustment value 1.21
Y-Axis sensitivity adjustment value 1.22
Z-Axis sensitivity adjustment value 1.17
ax = -70.19 ay = -70.56 az = 931.03 mg
gx = 0.02 gy = 0.00 gz = 0.02 deg/s
mx = -189 my = 355 mz = 127 mG
q0 = 0.97 qx = -0.04 qy = 0.03 qz = 0.22
Yaw, Pitch, Roll: 16.45, 4.22, -4.14
rate = 140.15 Hz
```

Resources and Going Further

For more information about the MPU-9250 Breakout, check out the links below.

- MPU-9250 Datasheet
- MPU-9250 Register Map
- MPU-9250 Breakout Schematic

Many of the advanced features of the MPU-9250 are only accessable by agreeing to a pages of licensing terms and logging in as a developer to get access to Embedded MotionDriver 6.12. This approach isn't super Arduino friendly. At power up 3062 bytes of undocumented hex needs to be loaded into the MPU-9250.

```
#define DMP_CODE_SIZE          (3062)

static const unsigned char dmp_memory[DMP_CODE_SIZE] = {
    /* bank # 0 */
    0x00, 0x00, 0x70, 0x00, 0x00, 0x00, 0x00, 0x24, 0x00, 0x0
0, 0x00, 0x02, 0x00, 0x03, 0x00,
    0x00, 0x00, 0x65, 0x00, 0x54, 0xff, 0xef, 0x00, 0x00, 0xf
a, 0x80, 0x00, 0x0b, 0x12, 0x82,...
```

Code snippet of MPU-9250 Embedded MotionDriver firmware

That binary combined with the driver and any code that does something with the sensor data quickly maxes out smaller microcontrollers. Feel free to register and play around if you want to take this product further.

A customer of ours shared his great tutorial on Affordable 9-DoF Sensor Fusion. Check it out for more info on sensor fusion.

For more SparkFun sensor fun, check out these other great tutorials. (This was fun, right??)