

Making an Arduino Compass



Knowledge is a process of piling up facts; wisdom lies in their simplification.
Martin H Fischer

Modification Record

Issue	Date	Author	Changes (Including the change authority)
A	10 March 2020	ZizWiz	Original from various notes
B			

Contents

1. Introduction	4
2. True North vs Magnetic North.....	4
2.1. What causes this movement of Magnetic North?.....	4
3. C# App	5
4. Parts list	6
5. Libraries	6
6. Magnetometer Sensor.....	7
7. Find I2C address.....	8
8. QMC5883L.....	9
9. MPU6050 Accelerometer and Gyroscope.....	10
9.1. Calibrate the MPU6050.....	10
9.2. About the MPU6050	11
10. Arduino Nano	13
11. I²C LCD.....	14
11.1. I ² C Address.....	14
12. Appendix A – Find I2C addresses	15
13. Appendix – B – Calibrate MPU6050	16
14. Appendix C – Clinometer Code	24

1. Introduction

This project builds an Arduino Compass that will display onto a C# Winform App. The compass will be able to display the magnetic or true North for a given point. For True North the user will need to provide further info all of which is described below.

2. True North vs Magnetic North

A magnetic compass points to Magnetic North which is not the same as True North Pole. The difference between the two is called the **Magnetic Declination**. This Magnetic Declination differs depending on where you are on Earth. The figure also changes on a regular basis due to complex fluid motion in the Earth's outer core.

You can find out the Declination for your position from the website @ <http://www.magnetic-declination.com/>

2.1. What causes this movement of Magnetic North?

Complex fluid motion in the outer core of the Earth (the molten metallic region that lies from 2800 to 5000 km below the Earth's surface) causes the magnetic field to change slowly with time. This change is known as secular variation. Because of secular variation, declination values shown on old topographic, marine and aeronautical charts need to be updated if they are to be used without large errors. Unfortunately, the annual change corrections given on most of these maps cannot be applied reliably if the maps are more than a few years old since the secular variation also changes with time in an unpredictable manner.

If you point the North of your compass to True North but the needle points to the left then you have a negative declination which means you are in the west.



While the needle pointing to the right means we are in the east.



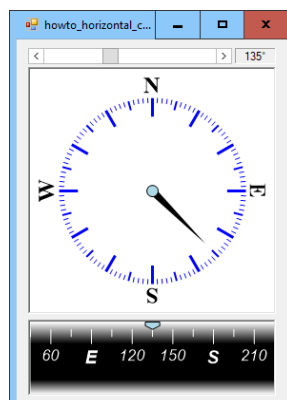
During this project I will accept the magnetic declination to be positive, so I am in the East and have a value of $0^{\circ} 6'$ East.

There are 60 minutes of arc in a single degree. This means the $0^{\circ} 6'$ gives us a decimal value of:

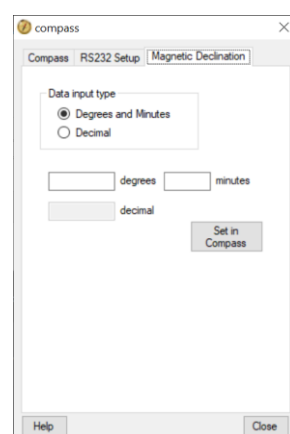
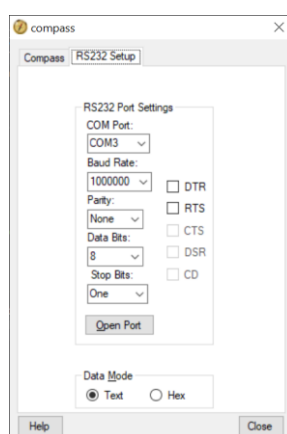
$$6'/60\text{min of Arc} = 0,1 \text{ degrees.}$$

3. C# App

The starter for this part comes from Rod Stephens C# Helper website @ <http://csharpHelper.com/blog/2020/03/draw-a-horizontal-compass-in-c/>



I have modified this to accept data coming in from an RS232 connection to an Arduino. Manual input of data is also possible and a help file containing this document is also available. Hopefully I have done it justice as Rod is a very eminent C# programmer.



4. Parts list

For this project we will use:

- Arduino Nano
- HMC5883L Triple Axis Compass Magnetometer Sensor

5. Libraries

To make this work a few libraries are needed. Using them makes this a lot simpler.

- HMC5883L Triple Axis Compass Magnetometer Sensor
- <https://github.com/mprograms/QMC5883LCompass>
 - <https://github.com/jarzebski/Arduino-HMC5883L>

Once you have the files unzipped find the file HMC5883L.h. Open it and then find the register definitions and delete them replacing them with those below.

<https://osoyoo.com/2017/09/14/qmc5883l-electronic-compass-for-arduino/>
https://osoyoo.com/driver/DFRobot_QMC5883.zip

https://github.com/DFRobot/DFRobot_QMC5883

```
#define HMC5883L_ADDRESS          (0x0D)
#define HMC5883L_REG_CONFIG_A    (0x0A)
#define HMC5883L_REG_CONFIG_B    (0x0B)
#define HMC5883L_REG_MODE        (0x09)
#define HMC5883L_REG_OUT_X_M     (0x01)
#define HMC5883L_REG_OUT_X_L     (0x00)
#define HMC5883L_REG_OUT_Z_M     (0x05)
#define HMC5883L_REG_OUT_Z_L     (0x04)
#define HMC5883L_REG_OUT_Y_M     (0x03)
#define HMC5883L_REG_OUT_Y_L     (0x02)
#define HMC5883L_REG_STATUS      (0x0C)
#define HMC5883L_REG_IDENT_A     (0x06)
#define HMC5883L_REG_IDENT_B     (0x07)
#define HMC5883L_REG_IDENT_C     (0x08)
```

Place the whole folder @ C:\Users\Your_Name\Documents\Arduino\libraries



6. Magnetometer Sensor

There are two common chips that are used for the purposes of a magnetic compass. They are

- Honeywell 3-Axis Digital Compass IC = HMC5883L
- 3-Axis Magnetic Sensor = QMC5883L. Datasheet is from Osoyoo who is owned by Pinetree Electronics Ltd Canada although I think this is a generic manufactured chip.

The QMC5883L is often referred to as a HMC5883L and because of this many call it a fake chip. In my mind it is not it is an entirely different generic chip doing a similar job.

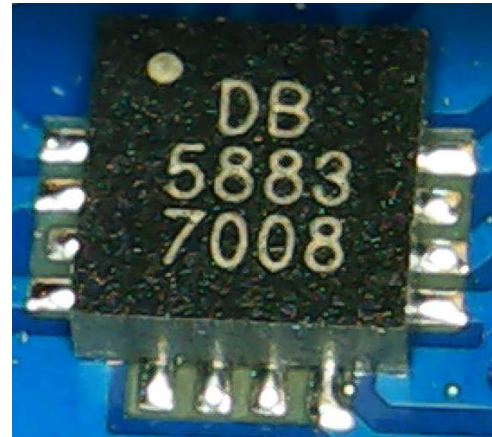
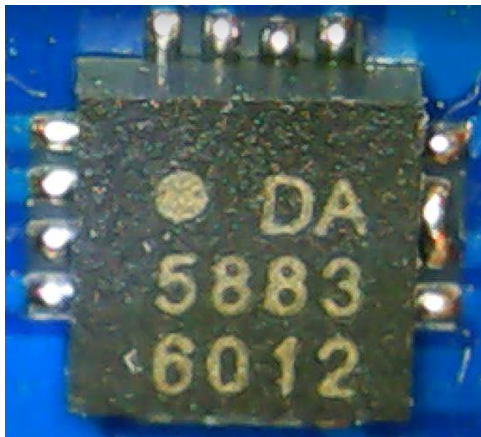
How do you know which chip you have as the PCB it is attached to will probably say they are all HMC5883L as we say below. Look at the chip and see what is written in the area shown in Red below.

	
QMC5883L Never called L883	HMC5883L Always called L883

Now we know there are two different chips we need to look at how we can use them.

The Honeywell 3-Axis Digital Compass IC, HMC5883L, is end of life and so may no longer be found. A replacement for this chip can be the Freescale Semiconductor's Xtrinsic MAG3110 Three-Axis, Digital Magnetometer. The MAG3110 has an I2C address of 0E₁₆.

After a somewhat torturous investigation, it turns out there's an A and B version of device, labelled DA5883 and DB5883 on the chip respectively.



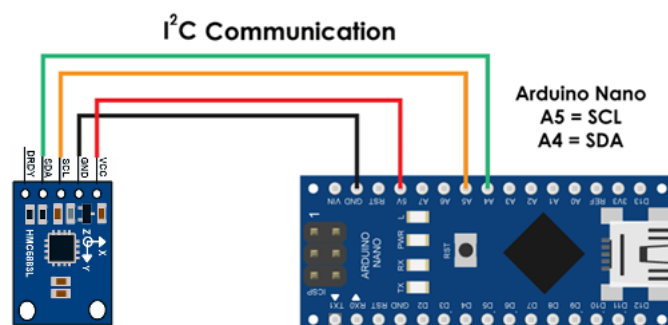
The DA5883 is identical to the HMC5883L, except (and I kid you not) that the status register doesn't work. In this case it's possible to use a HMC5883L library, but it's necessary to comment out lines that poll the data ready (RDY) bit in the status register. Instead just read the data after a sample time delay of so many milliseconds in your sketch.

The DB5883 on the other hand is a completely different device and functions according the the QMC5883L datasheet. In this case, as jremington suggests just use a QMC5883L library instead.

It's rather frustrating, as these changes are completely undocumented and have caused a good deal of confusion for those of us using this device.

7. Find I2C address

First attach the PCB to an Arduino as show



Now using the code in Appendix A, we can check the I2C address of the chip.

- QMC5883L = 0D₁₆
- HMC5883L = 0E₁₆

8. QMC5883L

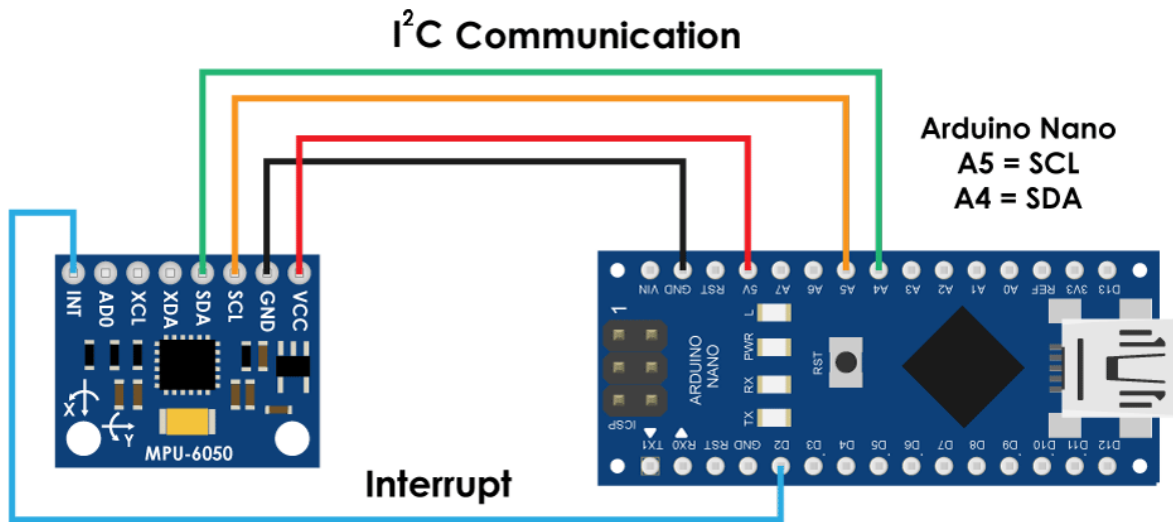
The QMC5883L chip has a different way to calibrate to the HMC5883L so be careful the code in Appendix B can be used to calibrate it.

The driver for QMC588L is not the same as the HMC5883L as some of the registers are different. You can use the HMC5883L driver but you will have to correct the drivers. This document will show you how to achieve this.

NOTE: Still to finish.

9. MPU6050 Accelerometer and Gyroscope

This device will read Gyro, acceleration and temperature.



Note this device can have either address 68_{16} or 69_{16} so check it.

9.1. Calibrate the MPU6050

If an MPU6050

- * is an ideal member of its tribe,
- * is properly warmed up,
- * is at rest in a neutral position,
- * is in a location where the pull of gravity is exactly 1g, and
- * has been loaded with the best possible offsets,

then it will report 0 for all accelerations and displacements, except for Z acceleration, for which it will report 16384 (that is 2^{14}). Most device probably won't do quite this well, but good offsets will all get the baseline outputs close to these target values.

Accelerate			Gyro		
X	Y	Z	X	Y	Z
0	0	16384	0	0	0

To calibrate put the MPU6050 on a flat and horizontal surface, and leave it operating for 5-10 minutes so its temperature gets stabilized.

Run the program in Appendix B. A "----- done -----" line will indicate that it has done its best. With the current accuracy-related constants (NFast = 1000, NSlow = 10000), it will take a few minutes to get there.

Along the way, it will generate a dozen or so lines of output, showing that for each of the 6 desired offsets, it is

- * first, trying to find two estimates, one too low and one too high, and
- * then, closing in until the bracket can't be made smaller.

The line just above the "done" line will look something like

```
[567,567] --> [-1,2]   [-2223,-2223] --> [0,1] [1131,1132] --> [16374,16404]
[155,156] --> [-1,1]   [-25,-24] --> [0,3] [5,6] --> [0,4]
```

As will have been shown in interspersed header lines, the six groups making up this line describe the optimum offsets for the X acceleration, Y acceleration, Z acceleration, X gyro, Y gyro, and Z gyro, respectively. In the sample shown just above, the trial showed that +567 was the best offset for the X acceleration, -2223 was best for Y acceleration. For those that are not equal choose the one closest to the offset value discussed earlier.

Once you decide on your values plug them into the code you find under Appendix C as shown below.

```
////////////////////
//Setup MPU
////////////////////

mpu.initialize();
pinMode(INTERRUPT_PIN, INPUT);
devStatus = mpu.dmpInitialize();
// supply your own gyro offsets here, scaled for min sensitivity

mpu.setXAccelOffset(-2627);
mpu.setYAccelOffset(-2040);
mpu.setZAccelOffset(1419);
mpu.setXGyroOffset(128);
mpu.setYGyroOffset(7);
mpu.setZGyroOffset(-57);
```

The code in Appendix C can now be uploaded to the Arduino and when it reboots the Roll, Pitch and Yaw should all be set at Zero. Now you know you have got good offsets.

Next move the MPU6050 so the figures are no longer at zero. Reboot the Arduino and they should return to zero. If they do not return to zero then recalibrate the MPU 6050.

9.2. About the MPU6050

MPU-6050 is an IMU Sensor that contains a MEMS (Microelectromechanical System) Accelerometer and MEMS Gyroscope on a single chip.

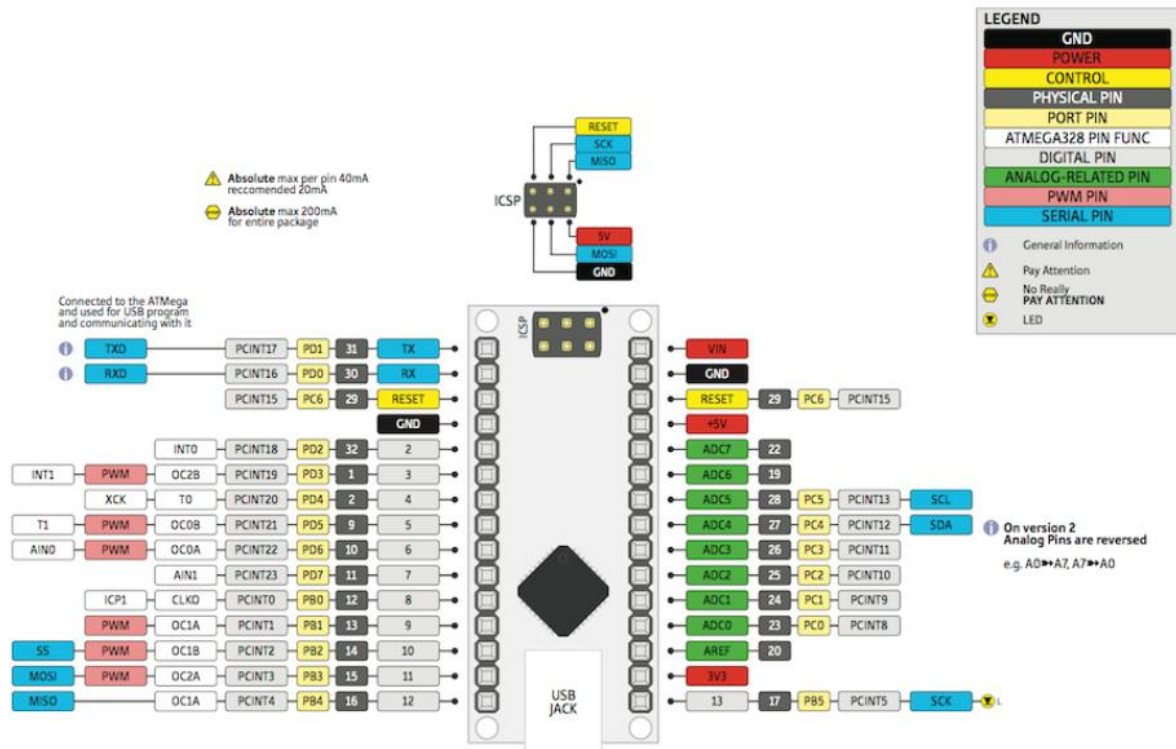
Here, IMU Sensor, where IMU stands for Inertial Measurement Unit, is a device that measures the specific force using Accelerometer, angular rate using Gyroscope and magnetic field using Magnetometers.

IMU Sensors are used in self-balancing robots, aircrafts, mobile phones, tablets, spacecraft, satellites, drones, UAVs (unmanned aerial vehicles) etc. for guidance, position detection, orientation detection, motion tracking and flight control.

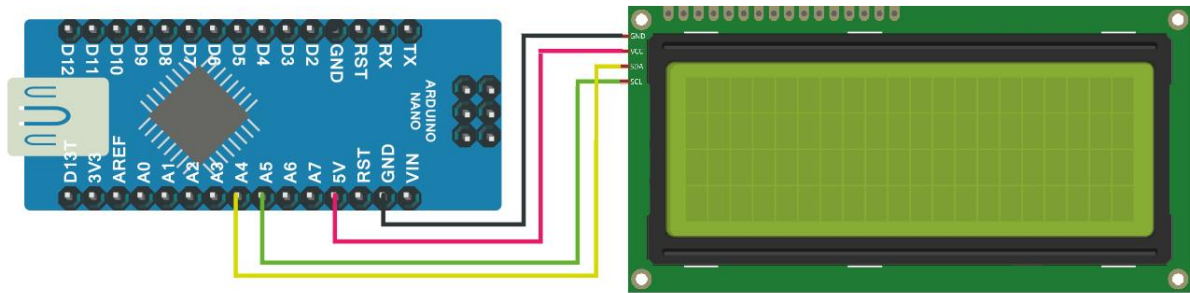
The MPU-6050, it is a six-axis motion tracking device that combines a 3-axis Accelerometer and a 3-axis Gyroscope on a single chip.

10. Arduino Nano

The pinouts for the Arduino are shown below.

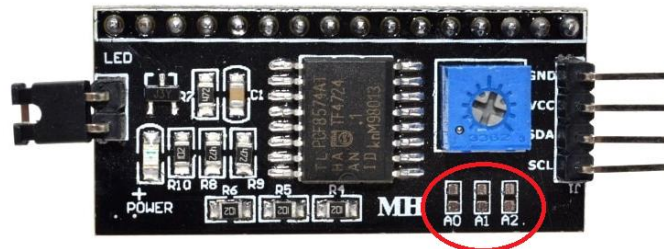


II. I²C LCD



11.1. I²C Address

Some I²C LCD module have an address selector solder pad.



They are usually labelled with A0, A1, and A2. The following table show you how to interpret the selector. “1” = Not Connected, “0” = Connected.

A0	A1	A2	HEX Address
1	1	1	27
0	1	1	26
1	0	1	25
0	0	1	24
1	1	0	23
0	1	0	22
1	0	0	21
0	0	0	20

12. Appendix A – Find I2C addresses

```
#include <Wire.h>

void setup() {
  Serial.begin (115200);

  // wait for serial port to connect
  while (!Serial)
  {
  }

  Serial.println ();
  Serial.println ("I2C scanner. Scanning ...");
  byte count = 0;

  Wire.begin();
  for (byte i = 8; i < 120; i++)
  {
    Wire.beginTransmission (i);
    if (Wire.endTransmission () == 0)
    {
      Serial.print ("Found address: ");
      Serial.print (i, DEC);
      Serial.print (" (0x");
      Serial.print (i, HEX);
      Serial.println (")");
      count++;
      delay (1); // maybe unneeded?
    } // end of good response
  } // end of for loop
  Serial.println ("Done.");
  Serial.print ("Found ");
  Serial.print (count, DEC);
  Serial.println (" device(s).");
} // end of setup

void loop() {}
```

13. Appendix – B – Calibrate MPU6050

```
// MPU6050 offset-finder, based on Jeff Rowberg's MPU6050_RAW
// 2016-10-19 by Robert R. Fenichel (bob@fenichel.net)

// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class
// 10/7/2011 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//   2019-07-11 - added PID offset generation at beginning Generates first offsets
//               - in @ 6 seconds and completes with 4 more sets @ 10 seconds
//               - then continues with original 2016 calibration code.
//   2016-11-25 - added delays to reduce sampling rate to ~200 Hz
//               added temporizing printing during long computations
//   2016-10-25 - requires inequality (Low < Target, High > Target) during expansion
//               dynamic speed change when closing in
//   2016-10-22 - cosmetic changes
//   2016-10-19 - initial release of IMU_Zero
//   2013-05-08 - added multiple output formats
//               - added seamless Fastwire support
//   2011-10-07 - initial release of MPU6050_RAW
```

```
/* =====
I2Cdev device library code is placed under the MIT license
Copyright (c) 2011 Jeff Rowberg
```

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

If an MPU6050

- * is an ideal member of its tribe,
- * is properly warmed up,
- * is at rest in a neutral position,
- * is in a location where the pull of gravity is exactly 1g, and
- * has been loaded with the best possible offsets,

then it will report 0 for all accelerations and displacements, except for Z acceleration, for which it will report 16384 (that is, 2^{14}). Your device probably won't do quite this well, but good offsets will all get the baseline outputs close to these target values.

Put the MPU6050 on a flat and horizontal surface, and leave it operating for 5-10 minutes so its temperature gets stabilized.

Run this program. A "----- done -----" line will indicate that it has done its best. With the current accuracy-related constants (NFast = 1000, NSlow = 10000), it will take a few minutes to get there.

Along the way, it will generate a dozen or so lines of output, showing that for each of the 6 desired offsets, it is

- * first, trying to find two estimates, one too low and one too high, and
- * then, closing in until the bracket can't be made smaller.

The line just above the "done" line will look something like

```
[567,567] --> [-1,2] [-2223,-2223] --> [0,1] [1131,1132] --> [16374,16404] [155,156] --> [-1,1] [-25,-24] --> [0,3] [5,6] --> [0,4]
```

As will have been shown in interspersed header lines, the six groups making up this line describe the optimum offsets for the X acceleration, Y acceleration, Z acceleration, X gyro, Y gyro, and Z gyro, respectively. In the sample shown just above, the trial showed that +567 was the best offset for the X acceleration, -2223 was best for Y acceleration, and so on.

The need for the delay between readings (usDelay) was brought to my attention by Nikolaus Doppelhammer.

```
=====
*/
```

```
// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050.h"
```

```
// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
```

```

#include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 accelgyro;
//MPU6050 accelgyro(0x69); // <-- use for AD0 high

const char LBRACKET = '[';
const char RBRACKET = ']';
const char COMMA = ',';
const char BLANK = ' ';
const char PERIOD = '.';

const int iAx = 0;
const int iAy = 1;
const int iAz = 2;
const int iGx = 3;
const int iGy = 4;
const int iGz = 5;

const int usDelay = 3150; // empirical, to hold sampling to 200 Hz
const int NFast = 1000; // the bigger, the better (but slower)
const int NSlow = 10000; // ..
const int LinesBetweenHeaders = 5;
    int LowValue[6];
    int HighValue[6];
    int Smoothed[6];
    int LowOffset[6];
    int HighOffset[6];
    int Target[6];
    int LinesOut;
    int N;

void ForceHeader()
{ LinesOut = 99; }

void GetSmoothed()
{ int16_t RawValue[6];
  int i;
  long Sums[6];
  for (i = iAx; i <= iGz; i++)
    { Sums[i] = 0; }
  // unsigned long Start = micros();

  for (i = 1; i <= N; i++)

```

```

    { // get sums
      accelgyro.getMotion6(&RawValue[iAx], &RawValue[iAy], &RawValue[iAz],
                          &RawValue[iGx], &RawValue[iGy], &RawValue[iGz]);
      if ((i % 500) == 0)
        Serial.print(PERIOD);
      delayMicroseconds(usDelay);
      for (int j = iAx; j <= iGz; j++)
        Sums[j] = Sums[j] + RawValue[j];
    } // get sums
// unsigned long usForN = micros() - Start;
// Serial.print(" reading at ");
// Serial.print(1000000/((usForN+N/2)/N));
// Serial.println(" Hz");
for (i = iAx; i <= iGz; i++)
  { Smoothed[i] = (Sums[i] + N/2) / N ; }
} // GetSmoothed

void Initialize()
{
  // join I2C bus (I2Cdev library doesn't do this automatically)
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
  #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
  #endif

  Serial.begin(9600);

  // initialize device
  Serial.println("Initializing I2C devices...");
  accelgyro.initialize();

  // verify connection
  Serial.println("Testing device connections...");
  Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050
connection failed");
  Serial.println("PID tuning Each Dot = 100 readings");
  /*A tidbit on how PID (PI actually) tuning works.
  When we change the offset in the MPU6050 we can get instant results. This allows us to use
  Proportional and
  integral of the PID to discover the ideal offsets. Integral is the key to discovering these
  offsets, Integral
  uses the error from set-point (set-point is zero), it takes a fraction of this error (error * ki)
  and adds it
  to the integral value. Each reading narrows the error down to the desired offset. The greater
  the error from
  set-point, the more we adjust the integral value. The proportional does its part by hiding the
  noise from the

```

integral math. The Derivative is not used because of the noise and because the sensor is stationary. With the

noise removed the integral value lands on a solid offset after just 600 readings. At the end of each set of 100

readings, the integral value is used for the actual offsets and the last proportional reading is ignored due to

the fact it reacts to any noise.

*/

```
    accelgyro.CalibrateAccel(6);
    accelgyro.CalibrateGyro(6);
    Serial.println("\nat 600 Readings");
    accelgyro.PrintActiveOffsets();
    Serial.println();
    accelgyro.CalibrateAccel(1);
    accelgyro.CalibrateGyro(1);
    Serial.println("700 Total Readings");
    accelgyro.PrintActiveOffsets();
    Serial.println();
    accelgyro.CalibrateAccel(1);
    accelgyro.CalibrateGyro(1);
    Serial.println("800 Total Readings");
    accelgyro.PrintActiveOffsets();
    Serial.println();
    accelgyro.CalibrateAccel(1);
    accelgyro.CalibrateGyro(1);
    Serial.println("900 Total Readings");
    accelgyro.PrintActiveOffsets();
    Serial.println();
    accelgyro.CalibrateAccel(1);
    accelgyro.CalibrateGyro(1);
    Serial.println("1000 Total Readings");
    accelgyro.PrintActiveOffsets();
```

Serial.println("\n\n Any of the above offsets will work nice \n\n Lets proof the PID tuning using another method:");

} // Initialize

```
void SetOffsets(int TheOffsets[6])
{ accelgyro.setXAccelOffset(TheOffsets [iAx]);
  accelgyro.setYAccelOffset(TheOffsets [iAy]);
  accelgyro.setZAccelOffset(TheOffsets [iAz]);
  accelgyro.setXGyroOffset (TheOffsets [iGx]);
  accelgyro.setYGyroOffset (TheOffsets [iGy]);
  accelgyro.setZGyroOffset (TheOffsets [iGz]);
} // SetOffsets
```

```
void ShowProgress()
{ if (LinesOut >= LinesBetweenHeaders)
  { // show header
    Serial.println("\tXAccel\t\tYAccel\t\tZAccel\t\tXGyro\t\tYGyro\t\tZGyro");
```

```

    LinesOut = 0;
    } // show header
    Serial.print(BLANK);
    for (int i = iAx; i <= iGz; i++)
    { Serial.print(LBRACKET);
      Serial.print(LowOffset[i]),
      Serial.print(COMMA);
      Serial.print(HighOffset[i]);
      Serial.print("] --> [");
      Serial.print(LowValue[i]);
      Serial.print(COMMA);
      Serial.print(HighValue[i]);
      if (i == iGz)
        { Serial.println(RBRACKET); }
      else
        { Serial.print("]\t"); }
    }
    LinesOut++;
  } // ShowProgress

void PullBracketsIn()
{ boolean AllBracketsNarrow;
  boolean StillWorking;
  int NewOffset[6];

  Serial.println("\nclosing in:");
  AllBracketsNarrow = false;
  ForceHeader();
  StillWorking = true;
  while (StillWorking)
  { StillWorking = false;
    if (AllBracketsNarrow && (N == NFast))
      { SetAveraging(NSlow); }
    else
      { AllBracketsNarrow = true; } // tentative
    for (int i = iAx; i <= iGz; i++)
      { if (HighOffset[i] <= (LowOffset[i]+1))
        { NewOffset[i] = LowOffset[i]; }
        else
          { // binary search
            StillWorking = true;
            NewOffset[i] = (LowOffset[i] + HighOffset[i]) / 2;
            if (HighOffset[i] > (LowOffset[i] + 10))
              { AllBracketsNarrow = false; }
          } // binary search
      }
    SetOffsets(NewOffset);
    GetSmoothed();
    for (int i = iAx; i <= iGz; i++)

```

```

    { // closing in
      if (Smoothed[i] > Target[i])
      { // use lower half
        HighOffset[i] = NewOffset[i];
        HighValue[i] = Smoothed[i];
      } // use lower half
    }
    else
    { // use upper half
      LowOffset[i] = NewOffset[i];
      LowValue[i] = Smoothed[i];
    } // use upper half
  } // closing in
  ShowProgress();
} // still working

} // PullBracketsIn

void PullBracketsOut()
{ boolean Done = false;
  int NextLowOffset[6];
  int NextHighOffset[6];

  Serial.println("expanding:");
  ForceHeader();

  while (!Done)
  { Done = true;
    SetOffsets(LowOffset);
    GetSmoothed();
    for (int i = iAx; i <= iGz; i++)
    { // got low values
      LowValue[i] = Smoothed[i];
      if (LowValue[i] >= Target[i])
      { Done = false;
        NextLowOffset[i] = LowOffset[i] - 1000;
      }
    }
    else
    { NextLowOffset[i] = LowOffset[i]; }
  } // got low values

  SetOffsets(HighOffset);
  GetSmoothed();
  for (int i = iAx; i <= iGz; i++)
  { // got high values
    HighValue[i] = Smoothed[i];
    if (HighValue[i] <= Target[i])
    { Done = false;
      NextHighOffset[i] = HighOffset[i] + 1000;
    }
  }
}

```

```

        else
            { NextHighOffset[i] = HighOffset[i]; }
        } // got high values
    ShowProgress();
    for (int i = iAx; i <= iGz; i++)
        { LowOffset[i] = NextLowOffset[i]; // had to wait until ShowProgress done
          HighOffset[i] = NextHighOffset[i]; // ..
        }
    } // keep going
} // PullBracketsOut

void SetAveraging(int NewN)
{ N = NewN;
  Serial.print("averaging ");
  Serial.print(N);
  Serial.println(" readings each time");
} // SetAveraging

void setup()
{ Initialize();
  for (int i = iAx; i <= iGz; i++)
      { // set targets and initial guesses
        Target[i] = 0; // must fix for ZAccel
        HighOffset[i] = 0;
        LowOffset[i] = 0;
      } // set targets and initial guesses
  Target[iAz] = 16384;
  SetAveraging(NFast);

  PullBracketsOut();
  PullBracketsIn();

  Serial.println("----- done -----");
} // setup

void loop()
{
} // loop

```

14. Appendix C – Clinometer Code

```
////////////////////
//Includes
////////////////////

#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include <Wire.h>
#include <Adafruit_BMP280.h>
#include <LiquidCrystal_I2C.h>

////////////////////
//Create objects
////////////////////

MPU6050 mpu;           // I2C MPU6050
Adafruit_BMP280 bmp;   // I2C BMP280
LiquidCrystal_I2C lcd(0x27, 20, 4); // I2C LCD Screen

////////////////////
// Defines
////////////////////
#define INTERRUPT_PIN 2 // use pin 2 on Arduino Uno & most boards

////////////////////
// Create variables
////////////////////
float correct;

double StartPressure;

char foot = ' ';

int j = 0;
int Temperature = 0;
int Pressure = 0;
int Altitude = 0;
int Previous_Temperature = 0;
int Previous_Pressure = 0;
int Previous_Altitude = 0;

bool blinkState = false;

// MPU control/status vars
bool dmpReady = false; // set true if DMP init was successful
uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU
uint8_t devStatus; // return status after each device operation (0 = success, !=0 = error)
uint16_t packetSize; // expected DMP packet size (default is 42 bytes)
```



```

uint16_t fifoCount; // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q; // [w, x, y, z] quaternion container
VectorInt16 aa; // [x, y, z] accel sensor measurements
VectorInt16 aaReal; // [x, y, z] gravity-free accel sensor measurements
VectorInt16 aaWorld; // [x, y, z] world-frame accel sensor measurements
VectorFloat gravity; // [x, y, z] gravity vector
float euler[3]; // [psi, theta, phi] Euler angle container
float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

// =====
// ===          INTERRUPT DETECTION ROUTINE          ===
// =====

volatile bool mpuInterrupt = false; // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}

// =====
// ===          INITIAL SETUP          ===
// =====

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation
        difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    ////////////
    //Setup MPU
    ////////////

    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);
    devStatus = mpu.dmpInitialize();
    // supply your own gyro offsets here, scaled for min sensitivity
    /* mpu.setXGyroOffset(17);
    mpu.setYGyroOffset(-69);
    mpu.setZGyroOffset(27);
    mpu.setZAccelOffset(1551); // 1688 factory default for my test chip*/

    /*

```

```

mpu.setXGyroOffset(127);
mpu.setYGyroOffset(7);
mpu.setZGyroOffset(-57);
mpu.setZAccelOffset(1410);
*/

mpu.setXAccelOffset(-2627);
mpu.setYAccelOffset(-2040);
mpu.setZAccelOffset(1419);
mpu.setXGyroOffset(128);
mpu.setYGyroOffset(7);
mpu.setZGyroOffset(-57);


// make sure it worked (returns 0 if so)
if (devStatus == 0) {
    // turn on the DMP, now that it's ready
    mpu.setDMPEnabled(true);

    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

    // set our DMP Ready flag so the main loop() function knows it's okay to use it
    dmpReady = true;

    // get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOPacketSize();
} else {
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    // Serial.print(F("DMP Initialization failed (code "));
    //Serial.print(devStatus);
    //Serial.println(F(""));
}
////////////////////
// setup I2C LCD
////////////////////

lcd.begin();
lcd.backlight(); //put light on

////////////////////
// setup LCD text
////////////////////

lcd.setCursor(0, 0);

```

```

lcd.print(F("Eurostar"));
lcd.setCursor(0, 1);
lcd.print(F("EV97"));
lcd.setCursor(0, 2);
lcd.print(F("Owned By"));
lcd.setCursor(0, 3);
lcd.print(F("CFI Dave Garrison"));

delay(2000);
lcd.clear();

lcd.setCursor(0, 0);
lcd.print(F("Roll"));
lcd.setCursor(6, 0);
lcd.print(F("Pitch"));
lcd.setCursor(14, 0);
lcd.print(F("Yaw"));
lcd.setCursor(0, 2);
lcd.print(F("Temp"));
lcd.setCursor(6, 2);
lcd.print(F("Press"));
lcd.setCursor(14, 2);
lcd.print(F("Altude"));

//////////
// setup bmp280
//////////

if (!bmp.begin()) {
  Serial.println(F("Could not find a valid BMP280 sensor, check wiring!"));
  while (1);
}

/* Default settings from datasheet. */
bmp.setSampling(Adafruit_BMP280::MODE_NORMAL, /* Operating Mode. */
  Adafruit_BMP280::SAMPLING_X2, /* Temp. oversampling */
  Adafruit_BMP280::SAMPLING_X16, /* Pressure oversampling */
  Adafruit_BMP280::FILTER_X16, /* Filtering. */
  Adafruit_BMP280::STANDBY_MS_500); /* Standby time. */

StartPressure = bmp.readPressure() / 100; //get starting pressure to know what is 0

}
// =====
// ===          MAIN PROGRAM LOOP          ===
// =====

void loop() {
  // if programming failed, don't try to do anything

```

```

if (!dmpReady) return;

// wait for MPU interrupt or extra packet(s) available
while (!mpuInterrupt && fifoCount < packetSize) {
    if (mpuInterrupt && fifoCount < packetSize) {
        // try to get out of the infinite loop
        fifoCount = mpu.getFIFOCount();
    }
}

// reset interrupt flag and get INT_STATUS byte
mpuInterrupt = false;
mpuIntStatus = mpu.getIntStatus();

// get current FIFO count
fifoCount = mpu.getFIFOCount();

// check for overflow (this should never happen unless our code is too inefficient)
if ((mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) || fifoCount >=
1024) {
    // reset so we can continue cleanly
    mpu.resetFIFO();
    fifoCount = mpu.getFIFOCount();

    // otherwise, check for DMP data ready interrupt (this should happen frequently)
} else if (mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)) {
    // wait for correct available data length, should be a VERY short wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
    fifoCount -= packetSize;

    // Get Yaw, Pitch and Roll values
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

    // Yaw, Pitch, Roll values - Radians to degrees
    ypr[0] = ypr[0] * 180 / M_PI;
    ypr[1] = ypr[1] * 180 / M_PI;
    ypr[2] = ypr[2] * 180 / M_PI;

    // Skip 300 readings (self-calibration process)
    if (j <= 300) {
        correct = ypr[0]; // Yaw starts at random value, so we capture last value after 300 readings
    }
}

```

```

    j++;
}
// After 300 readings
else {
    ypr[0] = ypr[0] - correct; // Set the Yaw to 0 deg - subtract the last random Yaw value from
the current value to make the Yaw 0 degrees
    // Map the values of the MPU6050 sensor from -90 to 90
    int Yaw = map(ypr[0], -90, 90, 90, -90); // when neg kick with left foot
    int Pitch = map(ypr[1], -90, 90, -90, 90); // when neg you are coming down
    int Roll = map(ypr[2], -90, 90, -90, 90);

    // if negative force positive
    if (Roll < 0) Roll = Roll * -1;

    if (Yaw < 0)
    {
        foot = 'L';
        Yaw = Yaw * -1;
    }
    else if (Yaw > 0)
    {
        foot = 'R';
    }
    else
    {
        foot = ' ';
    }

    //write data to screen
    clearLCDLine(1);
    lcd.setCursor(0, 1);
    lcd.print(Roll); //roll angle
    lcd.print(char(0xDF));
    lcd.setCursor(6, 1);
    lcd.print(Pitch); //pitch angle
    lcd.print(char(0xDF));
    lcd.setCursor(14, 1);
    lcd.print(foot);
    lcd.print(F(" "));
    lcd.print(Yaw); //yaw angle
    lcd.print(char(0xDF));

    Temperature = round(bmp.readTemperature());
    Pressure = round((bmp.readPressure()) / 100);
    Altitude = round(bmp.readAltitude(StartPressure + 2.2));

    // Only write this data to screen if there is a change to stop flicker.

```

```

    if ((Previous_Temperature != Temperature) || (Previous_Pressure != Pressure) ||
        (Previous_Altitude != Altitude))
    {
        clearLCDLine(3);
        lcd.setCursor(0, 3);
        lcd.print(Temperature); //current temperature
        lcd.setCursor(2, 3);
        lcd.print(char(0xDF));
        lcd.print("C");

        lcd.setCursor(6, 3);
        lcd.print(Pressure); //current pressure
        lcd.print("hPa");

        lcd.setCursor(14, 3);
        lcd.print(Altitude); //rounded starting pressure for 18m elevation
        lcd.print("m");

        Previous_Temperature = Temperature;
        Previous_Pressure = Pressure;
        Previous_Altitude = Altitude;
    }
    delay (500);
}
}
}

//clear only lines to be rewritten
void clearLCDLine(int line)
{
    lcd.setCursor(0, line);
    for (int n = 0; n < 20; n++) // 20 indicates 0-20 symbols in line.
    {
        lcd.print(" ");
    }
}
}

```