# Primer Método

Abrir examples en arduino, luego MPU 6050 y abrir el sketch “IMU\_Zero”

// 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 begninning Generates first offsets

// - in @ 6 seconds and completes with 4 more sets @ 10 seconds

// - then continues with origional 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\t\tYAccel\t\t\t\tZAccel\t\t\tXGyro\t\t\tYGyro\t\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

# Segundo Método

Este método involucra a los offsets en la calibración. Durante la calibración deberemos mantener el sensor sin moverlo en la posición de trabajo habitual, entonces el programa empieza por leer los offsets y nos pide que enviemos un carácter por el puerto serie. El programa trata de corregir los errores de las medidas, para ello modifica constantemente el offest, usando un filtro y cada 100 lecturas comprueba los valores si se acercan a los que deseamos leer, aumentando o disminuyendo los offsets. Esto hará que las lecturas filtradas se acerquen a:

*-aceleración: p\_ax=0 , p\_ay=0 , p\_az=+16384*

*-velocidad angular: p\_gx=0 , p\_gy=0 , p\_gz=0*

*Tal como indica en el monitor debemos anotar las compensaciones obtenidas para configurarlas en nuestros proyectos, usando la función mpu.setXAccelOffset(), como se indica a continuación.*

*Abriremos el archivo MPU6050\_PMT y buscaremos la parte de Calibration results como se aprecia en la figura que sigue.*

// Librerias I2C para controlar el mpu6050

// la libreria MPU6050.h necesita I2Cdev.h, I2Cdev.h necesita Wire.h

#include "I2Cdev.h"

#include "MPU6050.h"

#include "Wire.h"

// La dirección del MPU6050 puede ser 0x68 o 0x69, dependiendo

// del estado de AD0. Si no se especifica, 0x68 estará implicito

MPU6050 sensor;

// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z

int ax, ay, az;

int gx, gy, gz;

//Variables usadas por el filtro pasa bajos

long f\_ax,f\_ay, f\_az;

int p\_ax, p\_ay, p\_az;

long f\_gx,f\_gy, f\_gz;

int p\_gx, p\_gy, p\_gz;

int counter=0;

//Valor de los offsets

int ax\_o,ay\_o,az\_o;

int gx\_o,gy\_o,gz\_o;

void **setup**() {

**Serial**.begin(57600); //Iniciando puerto serial

Wire.begin(); //Iniciando I2C

sensor.initialize(); //Iniciando el sensor

if (sensor.testConnection()) **Serial**.println("Sensor iniciado correctamente");

// Leer los offset los offsets anteriores

ax\_o=sensor.getXAccelOffset();

ay\_o=sensor.getYAccelOffset();

az\_o=sensor.getZAccelOffset();

gx\_o=sensor.getXGyroOffset();

gy\_o=sensor.getYGyroOffset();

gz\_o=sensor.getZGyroOffset();

**Serial**.println("Offsets:");

**Serial**.print(ax\_o); **Serial**.print("\t");

**Serial**.print(ay\_o); **Serial**.print("\t");

**Serial**.print(az\_o); **Serial**.print("\t");

**Serial**.print(gx\_o); **Serial**.print("\t");

**Serial**.print(gy\_o); **Serial**.print("\t");

**Serial**.print(gz\_o); **Serial**.print("\t");

**Serial**.println("nnEnvie cualquier caracter para empezar la calibracionnn");

// Espera un caracter para empezar a calibrar

while (true){if (**Serial**.available()) break;}

**Serial**.println("Calibrando, no mover IMU");

}

void **loop**() {

// Leer las aceleraciones y velocidades angulares

sensor.getAcceleration(&ax, &ay, &az);

sensor.getRotation(&gx, &gy, &gz);

// Filtrar las lecturas

f\_ax = f\_ax-(f\_ax>>5)+ax;

p\_ax = f\_ax>>5;

f\_ay = f\_ay-(f\_ay>>5)+ay;

p\_ay = f\_ay>>5;

f\_az = f\_az-(f\_az>>5)+az;

p\_az = f\_az>>5;

f\_gx = f\_gx-(f\_gx>>3)+gx;

p\_gx = f\_gx>>3;

f\_gy = f\_gy-(f\_gy>>3)+gy;

p\_gy = f\_gy>>3;

f\_gz = f\_gz-(f\_gz>>3)+gz;

p\_gz = f\_gz>>3;

//Cada 100 lecturas corregir el offset

if (counter==100){

//Mostrar las lecturas separadas por un [tab]

**Serial**.print("promedio:"); **Serial**.print("t");

**Serial**.print(p\_ax); **Serial**.print("\t");

**Serial**.print(p\_ay); **Serial**.print("\t");

**Serial**.print(p\_az); **Serial**.print("\t");

**Serial**.print(p\_gx); **Serial**.print("\t");

**Serial**.print(p\_gy); **Serial**.print("\t");

**Serial**.println(p\_gz);

//Calibrar el acelerometro a 1g en el eje z (ajustar el offset)

if (p\_ax>0) ax\_o--;

else {ax\_o++;}

if (p\_ay>0) ay\_o--;

else {ay\_o++;}

if (p\_az-16384>0) az\_o--;

else {az\_o++;}

sensor.setXAccelOffset(ax\_o);

sensor.setYAccelOffset(ay\_o);

sensor.setZAccelOffset(az\_o);

//Calibrar el giroscopio a 0º/s en todos los ejes (ajustar el offset)

if (p\_gx>0) gx\_o--;

else {gx\_o++;}

if (p\_gy>0) gy\_o--;

else {gy\_o++;}

if (p\_gz>0) gz\_o--;

else {gz\_o++;}

sensor.setXGyroOffset(gx\_o);

sensor.setYGyroOffset(gy\_o);

sensor.setZGyroOffset(gz\_o);

counter=0;

}

counter++;

}

# Tercer Método

Es el ángulo, pero no tengo mucha idea de cómo funciona

//MPU6050 Auto-Calibration

// Arduino sketch that returns calibration offsets for MPU6050

// Version 1.1 (31th January 2014)

// Done by Luis Ródenas <luisrodenaslorda@gmail.com>

// Based on the I2Cdev library and previous work by Jeff Rowberg <jeff@rowberg.net>

// Updates (of the library) should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib

// These offsets were meant to calibrate MPU6050's internal DMP, but can be also useful for reading sensors.

// The effect of temperature has not been taken into account so I can't promise that it will work if you

// calibrate indoors and then use it outdoors. Best is to calibrate and use at the same room temperature.

/\* ========== LICENSE ==================================

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.

=========================================================

\*/

// I2Cdev and MPU6050 must be installed as libraries

#include "I2Cdev.h"

#include "MPU6050.h"

#include "Wire.h"

/////////////////////////////////// CONFIGURATION /////////////////////////////

//Change this 3 variables if you want to fine tune the skecth to your needs.

int buffersize=1000; //Amount of readings used to average, make it higher to get more precision but sketch will be slower (default:1000)

int acel\_deadzone=8; //Acelerometer error allowed, make it lower to get more precision, but sketch may not converge (default:8)

int giro\_deadzone=1; //Giro error allowed, make it lower to get more precision, but sketch may not converge (default:1)

// 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(0x68); // <-- use for AD0 high

int16\_t ax, ay, az,gx, gy, gz;

int mean\_ax,mean\_ay,mean\_az,mean\_gx,mean\_gy,mean\_gz,state=0;

int ax\_offset,ay\_offset,az\_offset,gx\_offset,gy\_offset,gz\_offset;

/////////////////////////////////// SETUP ////////////////////////////////////

void setup() {

// join I2C bus (I2Cdev library doesn't do this automatically)

Wire.begin();

// COMMENT NEXT LINE IF YOU ARE USING ARDUINO DUE

TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz). Leonardo measured 250kHz.

// initialize serial communication

Serial.begin(19200);

// initialize device

accelgyro.initialize();

// wait for ready

while (Serial.available() && Serial.read()); // empty buffer

while (!Serial.available()){

Serial.println(F("Send any character to start sketch.\n"));

delay(1500);

}

while (Serial.available() && Serial.read()); // empty buffer again

// start message

Serial.println("\nMPU6050 Calibration Sketch");

delay(2000);

Serial.println("\nYour MPU6050 should be placed in horizontal position, with package letters facing up. \nDon't touch it until you see a finish message.\n");

delay(3000);

// verify connection

Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050 connection failed");

delay(1000);

// reset offsets

accelgyro.setXAccelOffset(0);

accelgyro.setYAccelOffset(0);

accelgyro.setZAccelOffset(0);

accelgyro.setXGyroOffset(0);

accelgyro.setYGyroOffset(0);

accelgyro.setZGyroOffset(0);

}

/////////////////////////////////// LOOP ////////////////////////////////////

void loop() {

if (state==0){

Serial.println("\nReading sensors for first time...");

meansensors();

state++;

delay(1000);

}

if (state==1) {

Serial.println("\nCalculating offsets...");

calibration();

state++;

delay(1000);

}

if (state==2) {

meansensors();

Serial.println("\nFINISHED!");

Serial.print("\nSensor readings with offsets:\t");

Serial.print(mean\_ax);

Serial.print("\t");

Serial.print(mean\_ay);

Serial.print("\t");

Serial.print(mean\_az);

Serial.print("\t");

Serial.print(mean\_gx);

Serial.print("\t");

Serial.print(mean\_gy);

Serial.print("\t");

Serial.println(mean\_gz);

Serial.print("Your offsets:\t");

Serial.print(ax\_offset);

Serial.print("\t");

Serial.print(ay\_offset);

Serial.print("\t");

Serial.print(az\_offset);

Serial.print("\t");

Serial.print(gx\_offset);

Serial.print("\t");

Serial.print(gy\_offset);

Serial.print("\t");

Serial.println(gz\_offset);

Serial.println("\nData is printed as: acelX acelY acelZ giroX giroY giroZ");

Serial.println("Check that your sensor readings are close to 0 0 16384 0 0 0");

Serial.println("If calibration was succesful write down your offsets so you can set them in your projects using something similar to mpu.setXAccelOffset(youroffset)");

while (1);

}

}

/////////////////////////////////// FUNCTIONS ////////////////////////////////////

void meansensors(){

long i=0,buff\_ax=0,buff\_ay=0,buff\_az=0,buff\_gx=0,buff\_gy=0,buff\_gz=0;

while (i<(buffersize+101)){

// read raw accel/gyro measurements from device

accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

if (i>100 && i<=(buffersize+100)){ //First 100 measures are discarded

buff\_ax=buff\_ax+ax;

buff\_ay=buff\_ay+ay;

buff\_az=buff\_az+az;

buff\_gx=buff\_gx+gx;

buff\_gy=buff\_gy+gy;

buff\_gz=buff\_gz+gz;

}

if (i==(buffersize+100)){

mean\_ax=buff\_ax/buffersize;

mean\_ay=buff\_ay/buffersize;

mean\_az=buff\_az/buffersize;

mean\_gx=buff\_gx/buffersize;

mean\_gy=buff\_gy/buffersize;

mean\_gz=buff\_gz/buffersize;

}

i++;

delay(2); //Needed so we don't get repeated measures

}

}

void calibration(){

ax\_offset=-mean\_ax/8;

ay\_offset=-mean\_ay/8;

az\_offset=(16384-mean\_az)/8;

gx\_offset=-mean\_gx/4;

gy\_offset=-mean\_gy/4;

gz\_offset=-mean\_gz/4;

while (1){

int ready=0;

accelgyro.setXAccelOffset(ax\_offset);

accelgyro.setYAccelOffset(ay\_offset);

accelgyro.setZAccelOffset(az\_offset);

accelgyro.setXGyroOffset(gx\_offset);

accelgyro.setYGyroOffset(gy\_offset);

accelgyro.setZGyroOffset(gz\_offset);

meansensors();

Serial.println("...");

if (abs(mean\_ax)<=acel\_deadzone) ready++;

else ax\_offset=ax\_offset-mean\_ax/acel\_deadzone;

if (abs(mean\_ay)<=acel\_deadzone) ready++;

else ay\_offset=ay\_offset-mean\_ay/acel\_deadzone;

if (abs(16384-mean\_az)<=acel\_deadzone) ready++;

else az\_offset=az\_offset+(16384-mean\_az)/acel\_deadzone;

if (abs(mean\_gx)<=giro\_deadzone) ready++;

else gx\_offset=gx\_offset-mean\_gx/(giro\_deadzone+1);

if (abs(mean\_gy)<=giro\_deadzone) ready++;

else gy\_offset=gy\_offset-mean\_gy/(giro\_deadzone+1);

if (abs(mean\_gz)<=giro\_deadzone) ready++;

else gz\_offset=gz\_offset-mean\_gz/(giro\_deadzone+1);

if (ready==6) break;

}

}