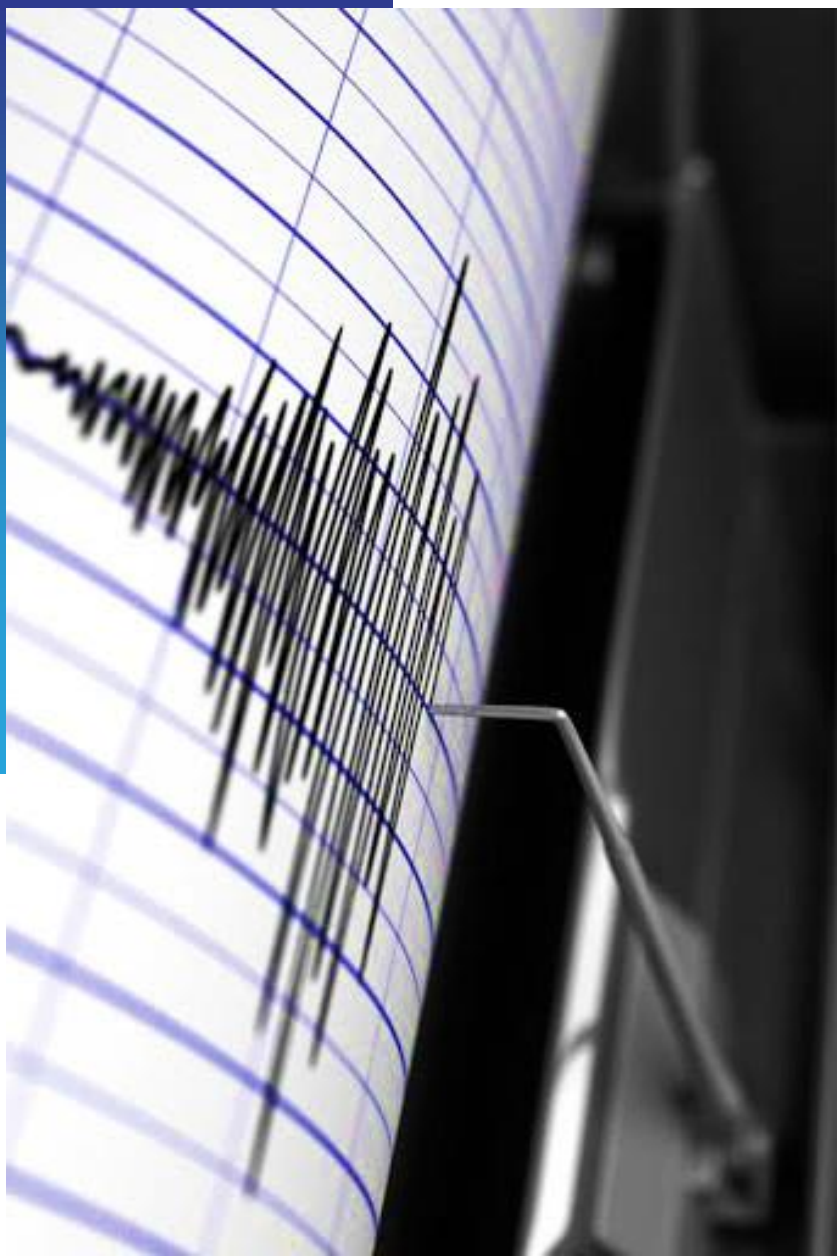
USAC
Centro Universitario de Occidente
División de Ciencias de la Ingeniería



MANUAL DE USO Y ENSAMBLAJE

ACELERÓMETRO TRIAXIAL DE BAJO COSTO

UNIVERSIDAD SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE
DIVISIÓN CIENCIAS DE LA INGENIERÍA

MANUAL DE USO Y ENSAMBLAJE

ACELERÓMETRO TRIAXIAL DE BAJO COSTO

EDITORES
ING. MARIO CIFUENTES JACOBS
DANIEL A. RODAS VELÁSQUEZ
JUAN A. ORÓZCO COLOMA

ACLARACIONES

La generación del acelerómetro que se presenta es el resultado de un proceso de investigación y calibración que se realizó en la División de Ciencias de la Ingeniería, este modelo tomo como fundamento el crear un prototipo de sensor sísmico de bajo costo para medir la respuesta dinámica del suelo.

Edición digital.

Quetzaltenango, Guatemala.
Noviembre 2020

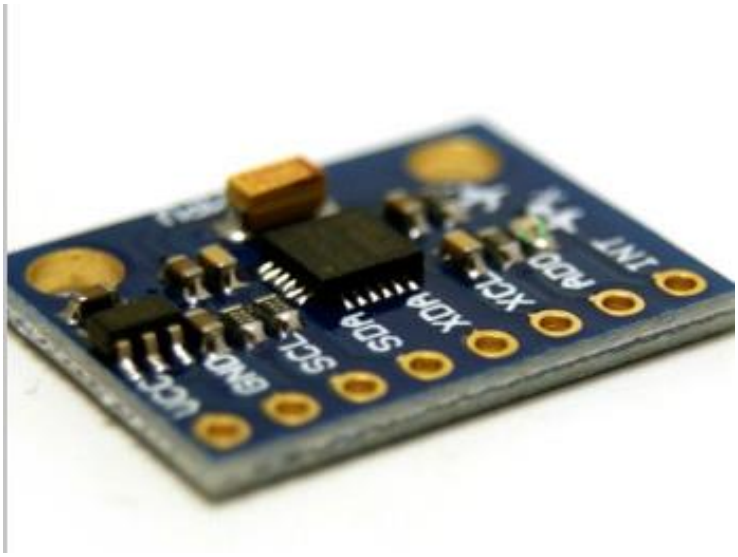
CONTENIDO

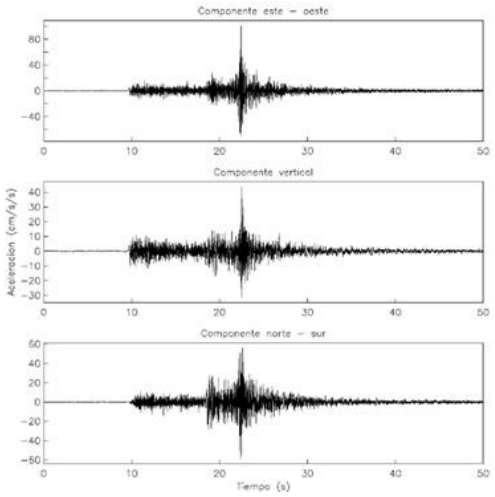
CONCEPTOS

Lo primero es ver qué son estos tipos de sensores que son capaces de detectar aceleración y giros, como se puede deducir de sus propios nombres.

- Acelerómetro: mide aceleración, es decir, la variación de velocidad por unidad de tiempo. En física, el cambio de velocidad respecto del tiempo (a=dV/dt) es la definición de aceleración. Según la Segunda Ley de Newton, también tenemos que a=F/m, y eso es lo que usan los acelerómetros para funcionar, es decir, usan parámetros de fuerza y masa del objeto. Para que eso se pueda implementar en la electrónica, se usan técnicas de MEMS (Micro Electro Mechanical Systems), que difieren en las técnicas de fabricación de chips electrónicos convencionales, ya que en un MEMS se crean partes mecánicas. En este caso, se crean unas pistas o elementos capaces de medir aceleraciones. Eso implica que se pueden sacar otras muchas unidades, como velocidad (si se integra la aceleración en el tiempo), si se integra nuevamente se tiene el desplazamiento, etc. Es decir, parámetros muy interesantes para saber la posición o detectar movimiento de un objeto

  Giroscopio: también llamado giróscopo, es un dispositivo que mide la velocidad angular de un objeto, es decir, el desplazamiento angular por unidad de tiempo o lo rápido que gira un cuerpo alrededor de su eje. En este caso, también se emplean técnicas MEMS para medir dicha velocidad usando un efecto conocido como Coriolis. Gracias a eso se puede medir la velocidad angular o, integrando la velocidad angular respecto al tiempo, se puede obtener el desplazamiento angular.





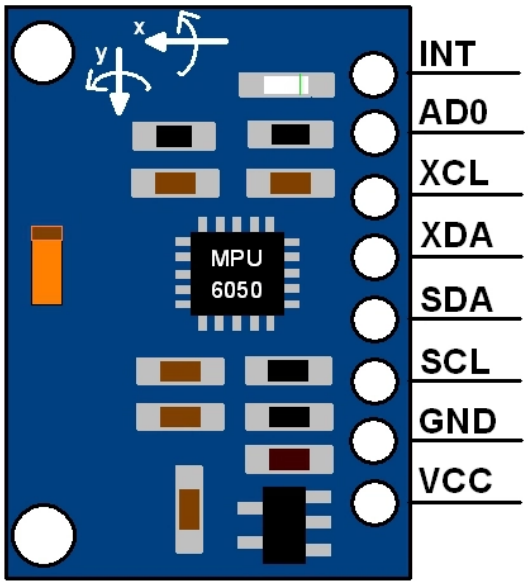
SENSOR DE ACELERACIÓN

MPU-6050

El sensor sísmico es un dispositivo que mide la vibración o la aceleración del movimiento del suelo o estructura. La fuerza generada por la vibración o el cambio en el movimiento (aceleración) hace que la masa "comprima" el material piezoeléctrico del sensor, generando una carga eléctrica que es proporcional a la fuerza ejercida sobre él.

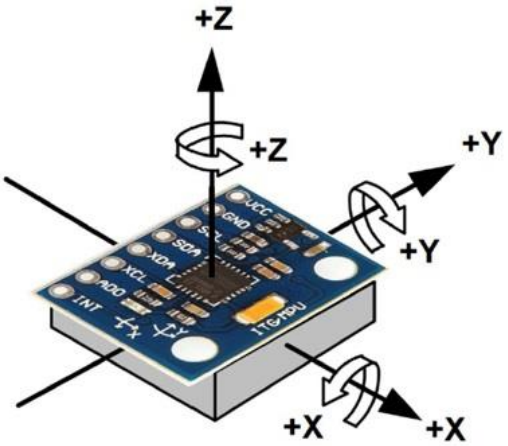
Se desarrolla un sensor sísmico con un circuito integrado MPU-6050 que contiene un acelerómetro y giroscopio MEMS en un solo empaque. Cuenta con una resolución de 16-bits, lo cual significa que divide el rango dinámico en 65536 fracciones, estos aplican para cada eje X, Y y Z al igual que en la velocidad angular. El sensor es ideal para diseñar control, medición de vibración, sistemas de medición inercial (IMU), detector de caídas, sensor de distancia y velocidad, y muchas cosas más. El MPU-6050 contiene un giroscópico, un acelerómetro, además de un sensor de temperatura, mediante I2C regresa unos valores conocidos como raw o “crudos” según el registro seleccionado.

MODULO MPU 6050



Ahora que se conoce qué es el acelerómetro y el giroscopio, el módulo MPU6050 es una placa electrónica que integra estos dos elementos para permitirte medir estos cambios de posición de un elemento y así poder generar una reacción. Por ejemplo, que cuando un objeto se mueva se encienda un LED, u otras cosas mucho más complejas.

Como se ha mencionado, tiene 6 ejes de libertad, DoF, 3 ejes para el acelerómetro X, Y, y Z de aceleración, y otros 3 ejes del giroscopio para medir la velocidad angular. Hay que tener en cuenta no equivocarse en la forma en la que posicionas el módulo y el sentido del giro para las mediciones, ya que si llegara a colocarse en una posición diferente se tendrá que modificar la programación del sensor. En la siguiente imagen donde especifica el sentido de los ejes (no obstante, la propia PCB también lo tiene impreso en un lateral):



ESPECIFICACIONES TÉCNICAS

- Sensor: MPU6050
- Voltaje de operación: 3V/3.3V~5V DC
- Regulador de voltaje en placa
- Grados de libertad (DoF): 6
- Rango Acelerómetro: 2g/4g/8g/16g
- Rango Giroscopio: 250Grad/Seg, 500Grad/Seg, 1000Grad/Seg, 2000Grad/Seg
- Sensibilidad Giroscopio: 131 LSBs/dps
- Interfaz: I2C
- Conversor AD: 16 Bits (salida digital)
- Tamaño: 2.0cm x 1.6cm x 0.3cm

Para más información ingresar al Datasheet:

https://www.alldatasheet.com/view.jsp?Searchword=MPU-6050&sField=4

LIBRERÍA DE COMUNICACIÓN PARA MPU6050

La comunicación del módulo es por I2C, esto le permite trabajar con la mayoría de microcontroladores. Los pines SCL y SDA tienen una resistencia pull-up en placa para una conexión directa al microcontrolador o Arduino.

Se tienen dos direcciones I2C para poder trabajar:

Pin AD0	Dirección I2C
AD0=HIGH (5V)	0x69
AD0=LOW (GND o NC)	0x68

El pin ADDR internamente en el módulo tiene una resistencia a GND, por lo que, si no se conecta, la dirección por defecto será 0x68.

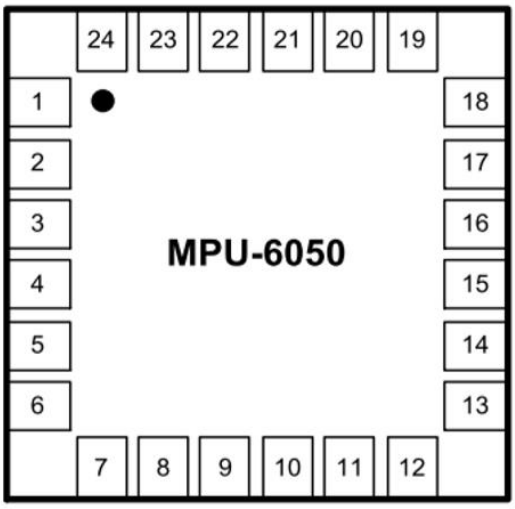
El módulo tiene un regulador de voltaje en placa de 3.3V, el cual se puede alimentar con los 5V del Arduino.

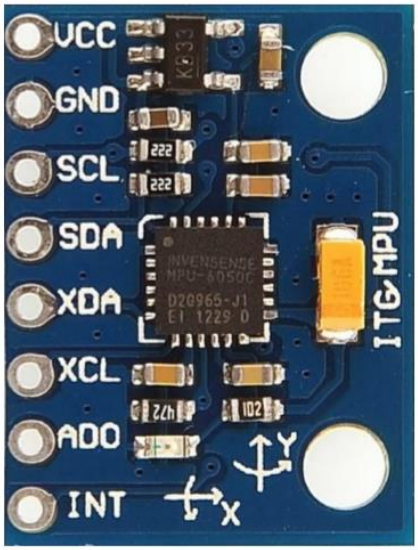
Esta librería trabaja con una librería adicional para la comunicación I2C:

- https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/I2Cdev

Para trabajar este tipo de sensor es necesario instalar las librerías en el IDE Arduino.

PIN OUT Y CONEXIONES





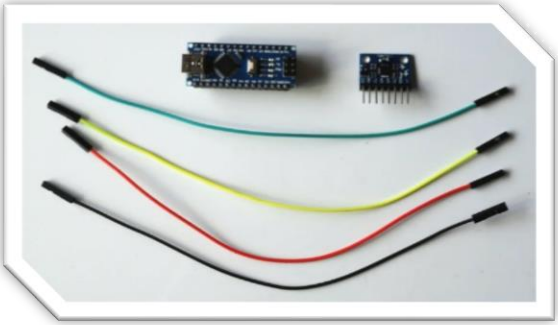
El acelerómetro MPU6050 se encuentra montado en una placa de expansión que facilita su conexionado y montado. La misma ordena los pines de la salida del sensor y los presenta en una fila de 8 pines. Así mismo, la placa viene integrada con un regulador KB33, para adaptar las tensiones de alimentación y referencias necesarias para el sensor, y las resistencias de pull-up, queridas para la comunicación I2C.

Placa de expansión	MPU 6050		Descripción
-	1	CLKIN	Entrada de referencia externa de clock (opcional)
XDA	6	AUX_DA	BUS maestro de datos I2C para conectar un sensor externo
XCL	7	AUX_CL	BUS maestro de datos I2C para conectar un sensor externo
Vcc	8	VLOGIC	Referencia para salida digital
AD0	9	AD0	LSB para la dirección de esclavo en el protocolo I2C
-	10	REGOUT	Conector para filtro regulador
-	11	FSYNC	Sincronización
INT	12	INT	Interrupción digital
Vcc	13	VDD	Tensión de alimentación
GND	18	GNA	Tierra
-	19	RESV	Reservado. No conectar
-	20	CPOUT	Capacitor
-	21	RESV	Reservado. No conectar
-	22	REV	Reservado. No conectar
SCL	23	SCL	Bus de entrada de clock I2C esclavo
SDA	24	SDA	Bus de datos I2C esclavo
2,3,4,5,14,15,16,17	2	NC	Sin conexión interna

ACELERÓMETRO CON SENSOR MPU 6050

MATERIALES

- ARDUINO UNO (SE PUEDE UTILIZAR OTROS TIPO DE MICROCONTROLADORES).
- CABLE MINI USB.
- JUMPERS.
- MPU 6050.
- CAJA IMPRESA 3D.
- TORNILLOS 23.
- TUERCAS.
- ESTAÑO.
- PASTA PARA SOLDAR.

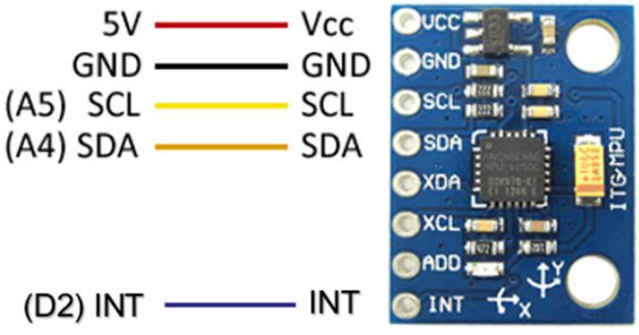


HERRAMIENTAS

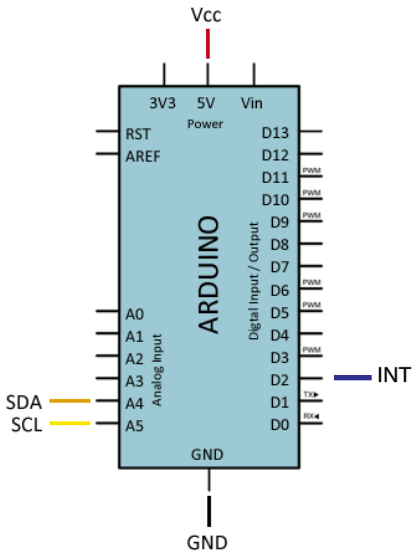
- CAUTIN.
- DESTORNILLADORES (PRINCIPALMENTE CRUZ PEQUEÑO).
- PINZAS.
- EQUIPO DE PROTECCIÓN PERSONAL.

ESQUEMA DE MONTAJE

La conexión es sencilla, simplemente alimentamos el módulo desde Arduino mediante GND y 5V y se conecta el pin SDA y SCL de Arduino con los pines correspondientes del sensor.

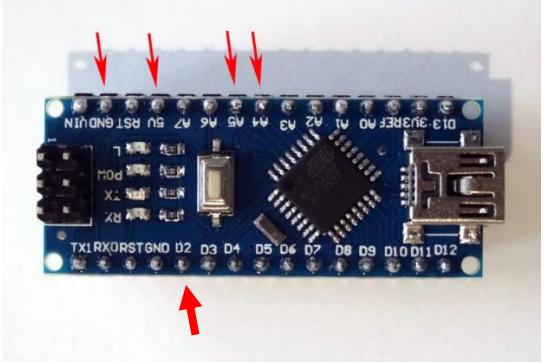


Mientras que la conexión vista desde el lado de Arduino quedaría así.



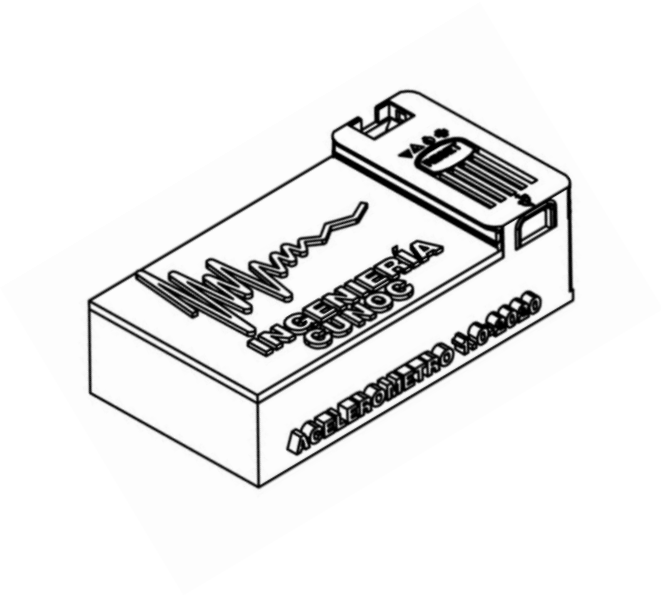
CONEXIONES

1. Conecte la alimentación VCC de 5 V (cable rojo), tierra (cable negro), SCL (cable amarillo), SDA (cable verde) y INT (cable Azul) al módulo MPU6050.
2. Conecte el otro extremo del cable de tierra (cable negro) a un pin de tierra de la placa Arduino Nano.
3. Conecte el otro extremo del cable de alimentación de 5V VCC (cable rojo) a la 5V pin de alimentación del tablero nano.
4. Conecte el otro extremo del cable SDA (cable verde) al pin 4 SDA / Analógico de la placa Arduino Nano.
5. Conecte el otro extremo del cable SCL (cable amarillo) al pin 5 de SCL / Analog de la placa Arduino Nano.
6. Conecte el otro extremo del cable INT (cable azul) al pin D2 de la placa Arduino Nano.
7. Se muestra dónde están el pin de tierra, 5V de alimentación, SDA / pin analógico 4 y SCL / pin analógico 5, y INT pines del Arduino Nano

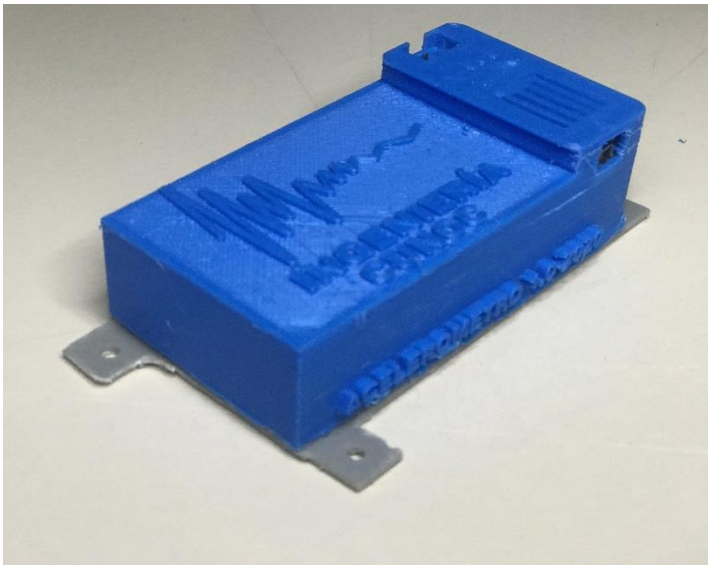


DISEÑO DE CAJA PARA MONTAJE

Se diseñó una caja especial para la protección del para la protección del sensor MPU 6050 y el Arduino NANO, también cumple con la función de anclaje. Esta caja fue diseñada en el software CAD de SolidWorks. La caja cuenta con una estructura principal de montaje, la cubierta principal, una cubierta ICSP y los tornillos niveladores.



Este diseño se replicó con una impresora 3D.



Los planos de la caja diseñada se presentan como anexo a este manual.

PROGRAMACIÓN

Para comenzar a programar el Arduino, deberá tener el IDE de Arduino instalado desde aquí: http://www.arduino.cc/ .

Para observar el código del graficador tendrá que tener NetBeans instalado: https://netbeans.org/ y para ejecutar el graficador deberá tener Java https://www.java.com/es/ o ejecutarlo desde su puerto .

NOTA: Toda programación deberá de transferir a 38400 baudios para que el graficador pueda leer de forma correcta los datos.

Se presenta la programación del sensor sísmico, se deberán de seguir los pasos para poder ejecutar de forma correcta el sensor:

CALIBRADOR

Al iniciar el sensor MPU6050 se cuenta un problema y es que presenta muchas vibraciones y ruido en las medidas, además cuando se tenga instalado el módulo MPU6050 en el proyecto, siempre puede haber un desnivel en sus componentes, motivo por el cual debemos calibrar el módulo, asegurándonos de que no haya un error de desnivel agregado en cada componente.

Podemos solucionar estos problemas al configurar el módulo MPU6050 OFFSETS, para compensar dichos errores.

Este código sirve para calibrar los offset del MPU6050.

```
// Sensor Sísmico Ingeniería CUNOC -PROGRAMACION CALIBRADOR-
// Librerías I2C para controlar el mpu6050
// la librería 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 implícito
MPU6050 sensor;

// Valores RAW (sin procesar) del acelerómetro 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 giroscopio
int ax_o,ay_o,az_o;
int gx_o,gy_o,gz_o;

void setup() {
  Serial.begin(38400);   //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 0°/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++;
}
```

Durante la calibración se debe mantener el sensor sin moverlo en la posición de trabajo habitual según se indicó en el segmento anterior, 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: ax=0 , ay=0 , az=+16384

-velocidad angular: gx=0 , gy=0 , gz=0

La calibración solo es necesario hacerla una vez y será el paso inicial antes de cualquier medición.

ESCALADOR Y TRANSFERENCIA DE DATOS

Luego de calibrar se debe de permitir la lectura de los datos para graficar y tenemos que escalar los datos a dimensiones conocidas por lo que utilizamos el siguiente código.

NOTA: Por lo objetivos del alcance del proyecto se descartarán los valores del giroscopio en la programación, pero si es útil podrá modificar el código para ver los datos.

```
// Sensor Sísmico Ingeniería CUNOC -PROGRAMACION ESCALADOR -
// librerías I2Cdev y MPU 6050
#include "I2Cdev.h"
#include "MPU6050.h"

// Se requiere la libreria Arduino Wire de I2Cdev
// se utiliza en I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// la dirección I2C predeterminada es de 0x68
// Aquí se pueden pasar direcciones I2C específicas como parámetro
// AD0 bajo = 0x68 (predeterminado para placa de evaluación IvenSense)
// AD0 alto = 0x69
MPU6050 accelgyro;
//MPU6050 accelgyro(0x69); // <-- usar pára AD0 alto

int16_t ax, ay, az;
int16_t gx, gy, gz;

// Descomente "OUTPUT_READABLE_ACCELGYRO" si desea ver una pestañaba separada
// lista de valores  X/Y/Z y luego giroscopio X/Y/Z valores en decimales.
Fácil de leer
// no es tan fácil de analizar si los valores son transferidos muy rápidos.
#define OUTPUT_READABLE_ACCELGYRO

#define LED_PIN 13
bool blinkState = false;

void setup() {
    // Ingreso de I2C bus (I2Cdev library no es automatica)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // inicial la comunicacion serial
    // (38400 escogido por que trabajamos con  8MHz y no ha 16MHz, pero
    // esto dependera del proyecto, tambien se coloca 38400 baudios para que
el graficador lea los datos)
    Serial.begin(38400);

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

    // verificar la conexion
    //Serial.println("Testing device connections...");
    //Serial.println(accelgyro.testConnection() ? "MPU6050 connection
successful" : "MPU6050 connection failed");
```

```
    //Usar el codigo para ver datos del giroscopio e ingresar los offsets
encontrados en la calibracion
    /*
    Serial.println("Updating internal sensor offsets...");
    // -76     -2359   1688    0       0       0
    Serial.print(accelgyro.getXAccelOffset()); Serial.print("\t"); // -76
    Serial.print(accelgyro.getYAccelOffset()); Serial.print("\t"); // -2359
    Serial.print(accelgyro.getZAccelOffset()); Serial.print("\t"); // 1688
    Serial.print(accelgyro.getXGyroOffset()); Serial.print("\t"); // 0
    Serial.print(accelgyro.getYGyroOffset()); Serial.print("\t"); // 0
    Serial.print(accelgyro.getZGyroOffset()); Serial.print("\t"); // 0
    Serial.print("\n");
    accelgyro.setXGyroOffset(220);
    accelgyro.setYGyroOffset(76);
    accelgyro.setZGyroOffset(-85);
    Serial.print(accelgyro.getXAccelOffset()); Serial.print("\t"); // -76
    Serial.print(accelgyro.getYAccelOffset()); Serial.print("\t"); // -2359
    Serial.print(accelgyro.getZAccelOffset()); Serial.print("\t"); // 1688
    Serial.print(accelgyro.getXGyroOffset()); Serial.print("\t"); // 0
    Serial.print(accelgyro.getYGyroOffset()); Serial.print("\t"); // 0
    Serial.print(accelgyro.getZGyroOffset()); Serial.print("\t"); // 0
    Serial.print("\n");
    */
    pinMode(LED_PIN, OUTPUT);
}
void loop() {
    // Escalador de los datos, teniendo en cuenta que la resolución de las
lecturas es de 16 bits por lo que el rango de lectura es de -16384 a 16384
para el acelerómetro y de -32768 a 32767 para giroscopio//

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

    float ax_m_s2 = ax * (9.81/16384.0);
    float ay_m_s2 = ay * (9.81/16384.0);
    float az_m_s2 = az * (9.81/16384.0);
    float gx_deg_s = gx * (250.0/32768.0);
    float gy_deg_s = gy * (250.0/32768.0);
    float gz_deg_s = gz * (250.0/32768.0);

    String dataOutput = (String(ax_m_s2,4) + "," + String(ay_m_s2,4) + "," +
String(az_m_s2,4) + "," + String(gx_deg_s,4) + "," + String(gy_deg_s,4) + ","
+ String(gz_deg_s,4));
    // presentacion de los datos:
    //accelgyro.getAcceleration(&ax, &ay, &az);
    #ifdef OUTPUT_READABLE_ACCELGYRO
        // display separado por tab accel/gyro x/y/z
        Serial.print(dataOutput+"\n");
    #endif

    #ifdef OUTPUT_BINARY_ACCELGYRO
        Serial.write((uint8_t)(ax_m_s2 >> 8)); Serial.write((uint8_t)(ax_m_s2
& 0xFF));
        Serial.write((uint8_t)(ay_m_s2 >> 8)); Serial.write((uint8_t)(ay_m_s2
& 0xFF));
        Serial.write((uint8_t)(az_m_s2 >> 8)); Serial.write((uint8_t)(az_m_s2
& 0xFF));
    #endif
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
    delay(20);
    //velocidad de lectura de los datos en milisegundos
}
```

Los valores obtenidos ya están escalados a unidades de aceleración y velocidad angular, y se ha convertido la aceleración a valores en m/s^2 por lo que se reemplazó el valor de g = 9.81 si el sensor se mantiene en posición horizontal se deben obtener mediciones cercanas a 9.8 m/s^2 (aceleración de la gravedad terrestre) es la componente z de la aceleración.

Para las primeras dos partes de la programación si utiliza otro tipo de microcontrolador deberá de verificar que las conexiones en los puertos coincidan con las especificaciones del sensor MPU6050 y el programa Arduino especificar el tipo de microcontrolador utilizado.

GRAFICADOR

Para poder ver el comportamiento de las aceleraciones se realizó un graficador per permite visualizar 4 graficas. La principal une las aceleraciones de los ejes X, Y y Z. Luego existe una gráfica individual para cada eje.

Es importante que en el código se introduzca el puerto al que está conectado el Arduino nano.

Es importante verificar el puerto de entrada en el graficador, ya que puede variar según el sistema operativo IOS o Windows.

El siguiente código nos sirve para ejecutar el graficador, programado en lenguaje JAVA en la plataforma de NetBeans, para esto utilizamos dos source packages:

GraphApp

```
package com.usac.cunoc.acelerografo2.GUI;

import com.panamahitek.ArduinoException;
import com.panamahitek.PanamaHitek_Arduino;
import com.usac.cunoc.acelerografo2.file.ManejadorArchivo;
import java.io.File;
import java.io.IOException;
import java.time.LocalDateTime;
import java.util.Random;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.event.ActionEvent;
```

```
import javafx.scene.Scene;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.LineChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.stage.FileChooser;
import javafx.stage.FileChooser.ExtensionFilter;
import javafx.stage.Stage;
import jssc.SerialPortEvent;
import jssc.SerialPortEventListener;
import jssc.SerialPortException;

public class GraphApp extends Application {

    /*Conector al Arduino */
    private PanamaHitek_Arduino ino = new PanamaHitek_Arduino();

    final int WINDOW_SIZE = 20;
    final String dimAcc ="m/s^2";
    final String dimTime ="Tiempo";
    private ScheduledExecutorService scheduledExecutorService;

    /*Principal's Graph elements*/
    private final CategoryAxis xAxisPrincipal = new CategoryAxis();
    private final NumberAxis yAxisPrincipal = new NumberAxis();
    private XYChart.Series<String, Number> seriesAccXPrincipal;
    private XYChart.Series<String, Number> seriesAccYPrincipal;
    private XYChart.Series<String, Number> seriesAccZPrincipal;

    /*Individual Graph Elements (X,Y,Z)*/
    private final CategoryAxis xAxisX = new CategoryAxis();
    private final NumberAxis yAxisX = new NumberAxis();
    private final CategoryAxis xAxisY = new CategoryAxis();
    private final NumberAxis yAxisY = new NumberAxis();
    private final CategoryAxis xAxisZ = new CategoryAxis();
    private final NumberAxis yAxisZ = new NumberAxis();
    private XYChart.Series<String, Number> seriesAccX;
    private XYChart.Series<String, Number> seriesAccY;
    private XYChart.Series<String, Number> seriesAccZ;

    /*Menu*/
    MenuBar menuBar = new MenuBar();
    Menu menu = new Menu("Ejecucion");
    MenuItem initMenuItem = new MenuItem("Inciar");
    MenuItem infoMenuItem = new MenuItem("Instrucciones");

    /*líneas de control*/
    ManejadorArchivo fileManager = new ManejadorArchivo();
    private String path = "";

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("Acelerografo CUNOC");
```

```
        /*Estructura del menu*/
        initMenuItem.setOnAction(e -> {
            try {
                initMenuItemAction(e, primaryStage);
            } catch (IOException ex) {
                Alert a = new Alert(AlertType.ERROR);
                a.setTitle("Error en el archivo");
                a.setHeaderText("Ha surgido un error al crear el archivo");
                a.show();
            }
        });

        infoMenuItem.setOnAction(e -> {
            infoMenuItemAction(e);
        });

        menu.getItems().add(initMenuItem);
        menu.getItems().add(infoMenuItem);

        menuBar.getMenus().add(menu);

        /*Configuracion de elementos de la grafica*/
        xAxisPrincipal.setLabel(dimTime);
        xAxisPrincipal.setAnimated(false);
        yAxisPrincipal.setLabel(dimAcc);
        yAxisPrincipal.setAnimated(false);

        seriesAccXPrincipal = new XYChart.Series<>();
        seriesAccXPrincipal.setName("Eje X");
        seriesAccYPrincipal = new XYChart.Series<>();
        seriesAccYPrincipal.setName("Eje Y");
        seriesAccZPrincipal = new XYChart.Series<>();
        seriesAccZPrincipal.setName("Eje Z");

        /*Grafica general X,Y,Z*/
        final LineChart<String, Number> lineChart = new
LineChart<>(xAxisPrincipal, yAxisPrincipal);
        lineChart.setAnimated(false);

        /*Self Graph's elements configuration*/
        xAxisX.setLabel(dimTime);
        xAxisX.setAnimated(false);
        yAxisX.setLabel(dimAcc);
        yAxisX.setAnimated(false);

        xAxisY.setLabel(dimTime);
        xAxisY.setAnimated(false);
        yAxisY.setLabel(dimAcc);
        yAxisY.setAnimated(false);

        xAxisZ.setLabel(dimTime);
        xAxisZ.setAnimated(false);
        yAxisZ.setLabel(dimAcc);
        yAxisZ.setAnimated(false);

        seriesAccX = new XYChart.Series<>();
        seriesAccX.setName("Eje X");
        seriesAccY = new XYChart.Series<>();
        seriesAccY.setName("Eje Y");
        seriesAccZ = new XYChart.Series<>();
        seriesAccZ.setName("Eje Z");

        /*Graficas individuales*/
        final LineChart<String, Number> xLineChart = new LineChart<>(xAxisX,
yAxisX);
        xLineChart.setAnimated(false);
```

```
        final LineChart<String, Number> yLineChart = new LineChart<>(xAxisY,
yAxisY);
        yLineChart.setAnimated(false);
        final LineChart<String, Number> zLineChart = new LineChart<>(xAxisZ,
yAxisZ);
        zLineChart.setAnimated(false);

        /*Unión de todas la variables*/
        lineChart.getData().addAll(seriesAccXPrincipal, seriesAccYPrincipal,
seriesAccZPrincipal);
        xLineChart.getData().add(seriesAccX);
        yLineChart.getData().add(seriesAccY);
        zLineChart.getData().add(seriesAccZ);

        /*Insertar elementos de paneles*/
        BorderPane secondPane = new BorderPane();
        secondPane.setLeft(xLineChart);
        secondPane.setCenter(yLineChart);
        secondPane.setRight(zLineChart);

        BorderPane bPane = new BorderPane();
        bPane.setTop(menuBar);
        bPane.setCenter(lineChart);
        bPane.setBottom(secondPane);

        /* Cree un panel principal e inserte paneles con elementos*/
        Scene scene = new Scene(bPane, 1500, 900, false);
        scene.getStylesheets().add("style.css");
        primaryStage.setScene(scene);

        primaryStage.show();

    }

    /*Leer datos del puerto de aurduino*/
    public void leerDatos() {
        try {
            ino.arduinoRX("/dev/cu.usbserial-1420", 38400, listener);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //Metodo especifico para arduino
    private final SerialPortEventListener listener = new
SerialPortEventListener() {
        @Override
        public void serialEvent(SerialPortEvent spe) {
            try {
                if (ino.isMessageAvailable()) {
                    String datos = ino.printMessage();
                    System.out.println(datos);
                    addData(LocalDateTime.now(),
                            Double.parseDouble(datos.split(",")[0]),
                            Double.parseDouble(datos.split(",")[1]),
                            Double.parseDouble(datos.split(",")[2])
                    );
                }
            } catch (SerialPortException ex) {
                Alert a = new Alert(AlertType.ERROR);
                a.setTitle("Comunicación");
                a.setHeaderText("Ha surgido un error en la comunicacion con
el arduino.");
                a.show();
            } catch (ArduinoException e) {
                Alert a = new Alert(AlertType.ERROR);
```

```
                a.setTitle("Error Arduino");
                a.setHeaderText("Arduino ha fallado, verificarlo.");
                a.show();
            }
        }
    };

    /* Reciba los datos, configurarlos en Graph y guardarlos en un archivo*/
    private void addData(LocalDateTime time, Double accX, Double accY, Double
accZ) {
        String textTime = time.getHour() + ":" + time.getMinute() + ":" +
time.getSecond() + "::" + (time.getNano() / 1000000);

        /*Insertar datos a graficas*/
        seriesAccXPrincipal.getData().add(new XYChart.Data<>(textTime,
accX));
        seriesAccYPrincipal.getData().add(new XYChart.Data<>(textTime,
accY));
        seriesAccZPrincipal.getData().add(new XYChart.Data<>(textTime,
accZ));
        seriesAccX.getData().add(new XYChart.Data<>(textTime, accX));
        seriesAccY.getData().add(new XYChart.Data<>(textTime, accY));
        seriesAccZ.getData().add(new XYChart.Data<>(textTime, accZ));

        if (seriesAccXPrincipal.getData().size() > WINDOW_SIZE) {
            seriesAccXPrincipal.getData().remove(0);
        }
        if (seriesAccYPrincipal.getData().size() > WINDOW_SIZE) {
            seriesAccYPrincipal.getData().remove(0);
        }
        if (seriesAccZPrincipal.getData().size() > WINDOW_SIZE) {
            seriesAccZPrincipal.getData().remove(0);
        }
        if (seriesAccX.getData().size() > WINDOW_SIZE) {
            seriesAccX.getData().remove(0);
        }
        if (seriesAccY.getData().size() > WINDOW_SIZE) {
            seriesAccY.getData().remove(0);
        }
        if (seriesAccZ.getData().size() > WINDOW_SIZE) {
            seriesAccZ.getData().remove(0);
        }

        if (!path.replace(" ", "").isEmpty()) {
            fileManager.addRow(time, accX, accY, accZ, path);
        }
    }

    private void prueba(int millisecons, int range) {
        scheduledExecutorService =
Executors.newSingleThreadScheduledExecutor();
        scheduledExecutorService.scheduleAtFixedRate(() -> {

            Random random = new Random();

            Platform.runLater(() -> {
                addData(LocalDateTime.now(),
                        (random.nextInt(range) + random.nextDouble()),
                        (random.nextInt(range) + random.nextDouble()),
                        (random.nextInt(range) + random.nextDouble()));
            });
        }, 0, millisecons, TimeUnit.MILLISECONDS);
    }

    @Override
    public void stop() throws Exception {
```

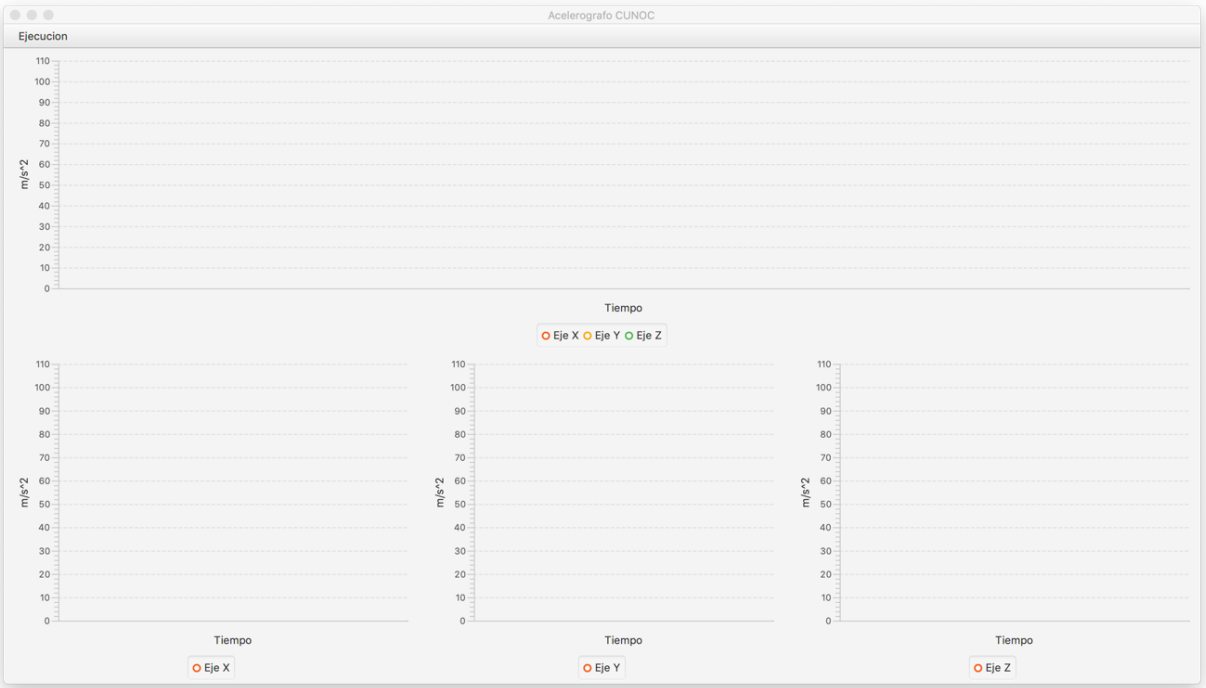
```
        super.stop();
//        scheduledExecutorService.shutdownNow();
    }

    /*Logica para inicio del programa*/
    private void initMenuItemAction(ActionEvent e, Stage primaryStage) throws
IOException {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Guardar Archivo");
        fileChooser.getExtensionFilters().addAll(new ExtensionFilter("CSV
Files", "*.csv"));
        File selectedFile = fileChooser.showSaveDialog(primaryStage);

        if (selectedFile != null) {
            path = selectedFile.getAbsolutePath();

            if (!path.contains(".csv")) {
                path = path + ".csv";
            }

            System.out.println("Path: " + path);
            Alert a = new Alert(AlertType.INFORMATION);
            a.setTitle("Información");
            a.setHeaderText("Archivo creado");
            a.show();
            fileManager.guardarArchivo(path);

            /*Comando para probar sin tener conenctado el arduino*/
//            prueba(200, 20);

            /*Comando para iniciar conexión con Arduino*/
            leerDatos();
        } else {
            Alert a = new Alert(AlertType.ERROR);
            a.setTitle("Sin archivo");
            a.setHeaderText("Debe crear o elegir un archivo donde se
almacenara la información");
            a.show();
        }
    }

    private void infoMenuItemAction(ActionEvent e) {
        Alert a = new Alert(AlertType.INFORMATION);
        a.setTitle("Instrucciones");
        a.setHeaderText("Para la ejecucion de este programa\n"
                + "se requiere crear un archivo CSV,\n"
                + "de no crearse no iniciara la ejecucion\n"
                + "del programa.");
        a.show();
    }
}
```

MANEJADOR DE ARCHIVO

```
package com.usac.cunoc.acelerografo2.file;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDateTime;

public class ManejadorArchivo {

    public void guardarArchivo(String path) throws IOException {
```

```
        System.out.println("Path: " + path);
        FileWriter fichero = null;
        File file = new File(path);
        fichero = new FileWriter(file);
        fichero.write("'Fecha y Hora','Aceleracion X','Aceleracion
Y','AceleracionZ'\n");
        fichero.close();
    }

    public void addRow(LocalDateTime time, Double accX, Double accY, Double
accZ, String path) {
        String row = "'" + time.toString() + "'," + accX + "," + accY + "," +
accZ;
        appendToFile(path, row);
    }

    private void appendToFile(String path, String texto) {
        File file = new File(path);

        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter(file,
true));
            writer.append((texto + "\n"));
            writer.close();
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

}
```

La pantalla del graficador ya ejecutado quedaría así.



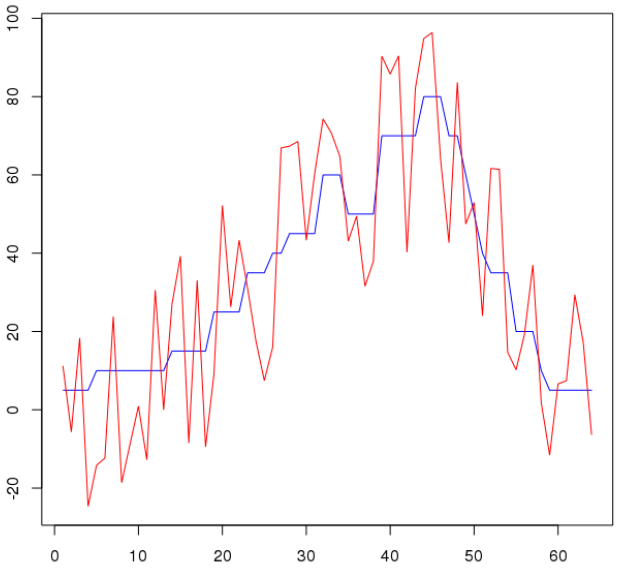
CALIBRACIÓN CON DISPOSITIVO COMERCIAL

MANEJO DE RUIDO Y FUENTES DE ERROR

Existen dos problemas importantes en este tipo de dispositivos: el ruido y los errores.

El ruido son todas aquellas interferencias que afectan a los dispositivos electrónicos. El acelerómetro es capaz de medir cualquier aceleración, sin embargo, sus lecturas son ruidosas y tienen un cierto margen de error.

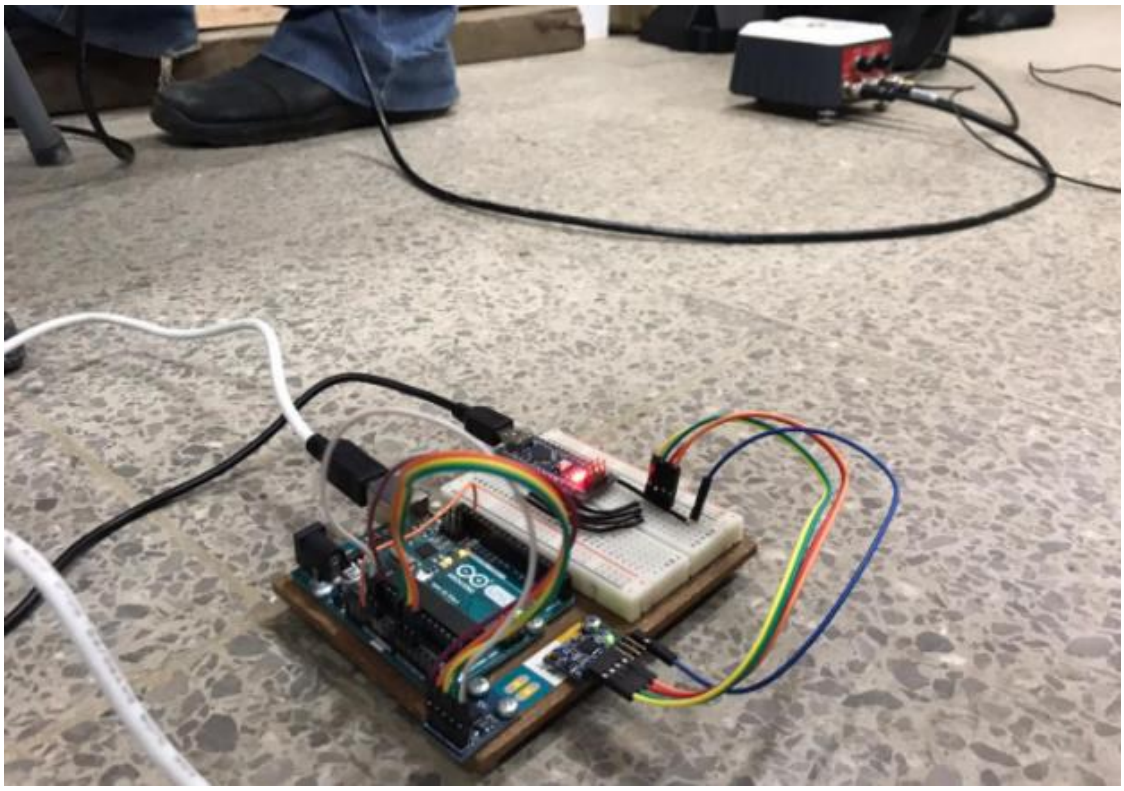
Al dibujar un gráfico de las medidas de un acelerómetro en función del tiempo, se obtendrá:



La aceleración real (ideal) está marcado en azul, y las lecturas reales del sensor están en rojo. Se Puedo afirmar que no cumple a cabalidad la definición de "preciso".

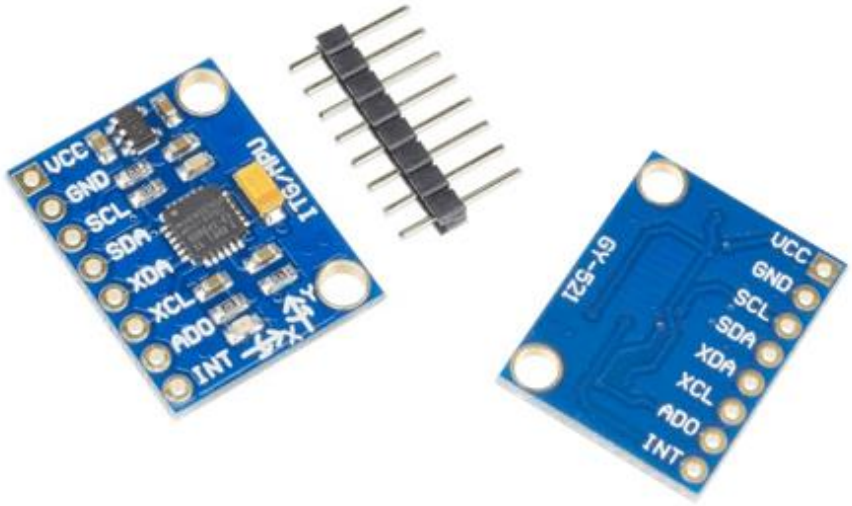
Y al realizar los cálculos de escala es inevitable que se produzca un pequeño error, que con el tiempo va acumulándose hasta que cualquier similitud con la realidad es pura coincidencia. Esto en inglés se llama drift (errores aleatorios). Por lo que se necesita un **FACTOR DE CORRECCION**, la idea es muy simple: hay que conseguir eliminar el ruido, el drift y conseguir que el acelerómetro sea lo más preciso posible.

PRUEBAS CON EL ACELERÓGRAFO KINEMETRICS ETNA 2



PRESUPUESTO

El acelerómetro con sensor MPU-6050 es un dispositivo excelente en relación al precio debido, entre otros factores, y es ampliamente utilizado por lo que es fácil encontrarlo.



Costo del acelerómetro:

DESCRIPCIÓN	COSTO
ARDUINO NANO	Q68.00
MPU650	Q35.00
CABLES TIPO JUMPERS (10 piezas)	Q7.00
CABLE MINI USB	Q10.00
CAJA BASE IMPRESIÓN 3D	Q200.00
TOTAL	Q320.00

Presupuesto elaborado en La Electrónica Guatemala octubre 2020. Sujeto a variaciones.

Cabe resaltar que para este dispositivo se puede utilizar cualquier otro tipo de microcontrolador, pero es necesario revisar el Data del microcontrolador para verificar los puertos de entrada y salida. Además, la programación debe de ser configurada, pero en teoría son los mismos principios.

El costo de este dispositivo es relativamente bajo respecto a los acelerómetros que se encuentran en el mercado.

REFERENCIA BIBLIOGRÁFICA

Tutorial MPU6050, Acelerómetro y Giroscopio. (n.d.). Retrieved November 3, 2020, from https://naylampmechatronics.com/blog/45_Tutorial-MPU6050-Acelerómetro-y-Giroscopio.html

MPU6050 Arduino, Acelerómetro y Giroscopio - HETPRO/TUTORIALES. (n.d.). Retrieved November 3, 2020, from https://hetpro-store.com/TUTORIALES/modulo-acelerometro-y-giroscopio-mpu6050-i2c-twi

MPU6050: módulo para posicionamiento con Arduino. (n.d.). Retrieved November 3, 2020, from https://www.hwlibre.com/mpu6050/

Tutorial de Arduino y MPU-6050 | robologs. (n.d.). Retrieved November 3, 2020, from https://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/

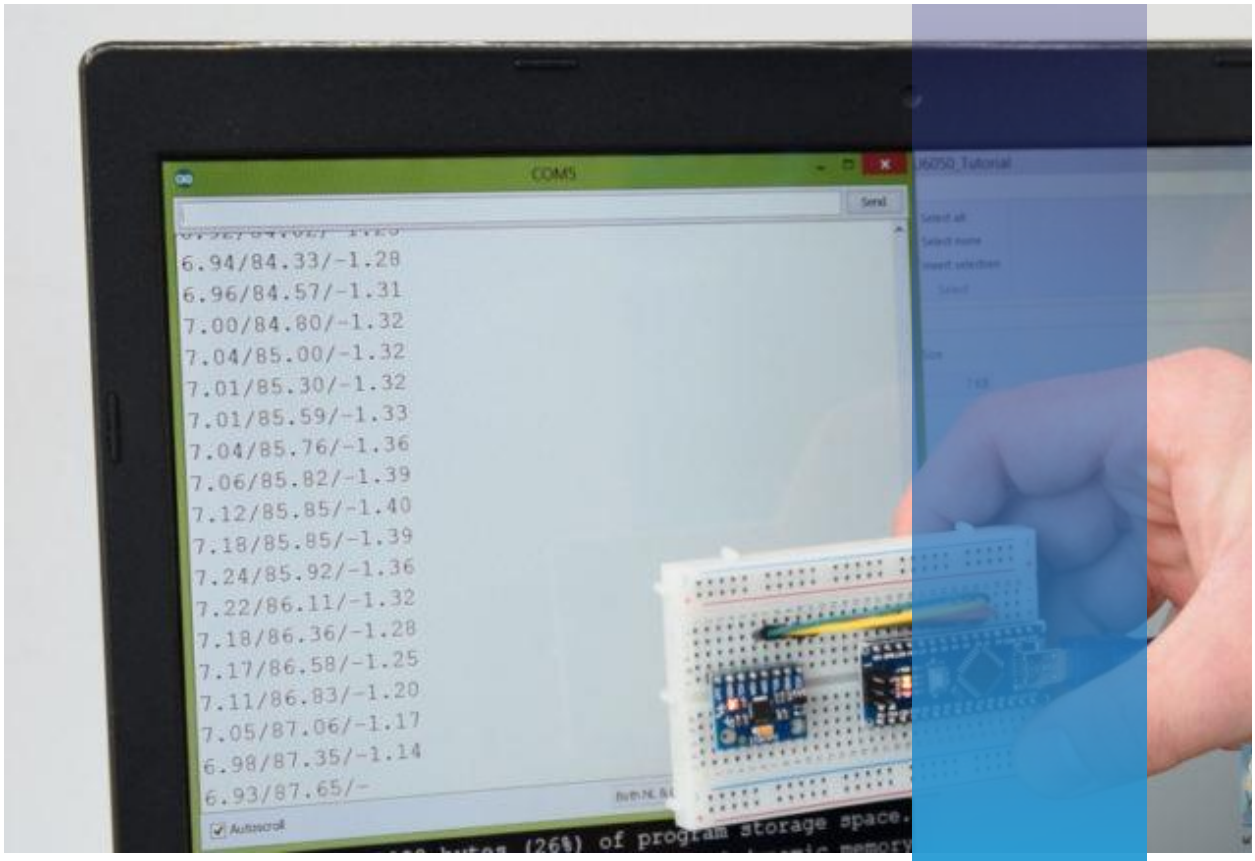
Tutorial MPU6050, Acelerómetro y Giroscopio. (n.d.). Retrieved November 3, 2020, from https://naylampmechatronics.com/blog/45_Tutorial-MPU6050-Acelerómetro-y-Giroscopio.html

Moreano, R., & Caiza, P. (2018). Experimental validation of the use of the Arduino Uno sensor for the determination of accelerations and displacements in structures. *Revista Internacional de Ingeniería de Estructuras*, *23*(3), 341–356.

Na, Y., El-Tawil, S., Ibrahim, A., & Eltawil, A. (2020). Automated Assessment of Building Damage from Seismic Events Using Smartphones. *Journal of Structural Engineering (United States)*, *146*(5), 1–14. https://doi.org/10.1061/(ASCE)ST.1943-541X.0002618

Soler-Llorens, J. L., Galiana-Merino, J. J., Nassim-Benabdeloued, B. Y., Rosa-Cintas, S., Zamora, J. O., & Giner-Caturla, J. J. (2019). Design and implementation of an arduino-based plug-and-play acquisition system for seismic noise measurements. *Electronics (Switzerland)*, *8*(9). https://doi.org/10.3390/electronics8091035

Tijuana, Z. R. Í. O. D. E., López, C. I. H., Salvador, D., Limón, L., Fuentes, M. E., & Espinoza, F. (2009). *Edificio De Nueve Niveles Localizado En La Vecindad De La Numerical and Experimental Vibration Frequencies of a Nine-Story Building Constructed Near the Tijuana B . C ., Mexico River Zone*. *9*, 97–114.



ACERCA DE

Este modelo pretende colocar al alcance de profesores y estudiantes de ingeniería, así como público en general la generación de dispositivos que pueden ser usados para la determinación de parámetros de propagación de ondas sísmicas, así como al ensamblarlo con otros elementos la generación de sistemas de alarma o de alerta temprana.

El trabajo de investigación y desarrollo del dispositivo fue realizado por la División de Ciencias de la Ingeniería del Centro Universitario de Occidente de la Universidad de San Carlos de Guatemala.

Equipo de trabajo:

- Ing. Mario Cifuentes Jacobs (Coordinación)
- Daniel Alberto Rodas Velásquez (Programación y calibración)
- Juan Adolfo Orozco Coloma (Modelado)

Agradecimientos:
Ing. Pavel Cifuentes / Laboratorio Ingeniería Sísmica
Ing. Nery Pérez / Coordinador Ingeniería Civil
MBA. Ing. Victor Carol Hernández / Director División Ciencias de la Ingeniería

Contacto: mariocifuentesjacobs@cunoc.edu.gt
Ingeniería.cunoc.usac.edu.gt