UNIVERSIDAD CARLOS III DE MADRID

ESCUELA TÉCNICA SUPERIOR

DEPARTAMENTO DE INGENIERÍA TÉRMICA Y DE FLUIDOS

PROYECTO DE FIN DE CARRERA

WAVEDUINO: DISPOSITIVO MEDIDOR DE OLAS MEDIANTE TECNOLOGÍA ARDUINO

TUTOR: Javier Rodríguez Rodríguez

DEPARTAMENTO DE INGENIERÍA TÉRMICA Y DE FLUIDOS

AUTOR: David Huertas Ruiz

INGENIERÍA INDUSTRIAL

A mi padre

ÍNDICE

PRÓLOGO

Cuando terminé mi último examen de la carrera no tenía ni idea de qué iba a hacer como proyecto de final de carrera. Ya había tenido varios intentos anteriores de comenzar proyectos fallidos que no iban a ningún sitio. Fue Javier Rodríguez el que me sugirió trastear con Arduino, buscando unir en un mismo proyecto las dos temáticas que más me han gustado de la carrera: electrónica y fluidos.

Pero para poder decidir el tema final estuve meses aprendiendo todo lo que pude sobre Arduino, unidades de medida inercial (IMU, siglas en inglés), Kalman y sus filtros de estimación y corrección, comunicaciones puerto-serie con periféricos en MATLAB, e incluso algo de programación en C para la configuración del microcontrolador Arduino.

La idea inicial de crear un dispositivo de flotabilidad neutra que surcara las distintas corrientes de un lecho fluido midiendo posición, presión y temperatura, fue descartada poco después dado que la duración del proyecto habría sido superior a un año y no quería eternizarme. Pero si se suprime la flotabilidad neutra de esta idea (de manera que el dispositivo flote sin más), eliminando a su vez la detección de la presión y la temperatura, ¿qué queda? Un medidor de olas. En la superficie del agua, el movimiento del dispositivo es bastante predecible por su considerable periodicidad, facilitando así los cálculos y el filtrado de los datos, ya que los errores de integración y de deriva de los sensores son más fáciles de predecir y corregir.

Sin embargo la intención de este proyecto no es la de crear un dispositivo medidor de olas sin más, sino demostrar que con los avances tecnológicos de hoy en día, se pueden desarrollar dispositivos de cierta complejidad sin muchas dificultades, y lo más importante, sin un esfuerzo económico elevado ya que estas tecnologías, aparte de ocupar muy poco, son de bajo coste.

He de añadir que Waveduino (nombre final del proyecto), es un nombre que ya han empleado distintas organizaciones para comercializar diversos productos basados en Arduino. Sin embargo no creo que haya mejor nombre que éste para describir un medidor de olas basado en Arduino.

ARDUINO Y PERIFÉRICOS

CONCEPTOS PREVIOS

Arduino es conocido cada vez más por ser uno de los fabricantes de hardware libre más solicitados. Sin embargo, bajo el nombre de Arduino se engloban tres conceptos distintos[1]:

- Una placa de hardware libre, basada en un microcontrolador que el usuario programa desde su PC, y una serie de pines hembra para conectar a la placa distintos periféricos.

- Un entorno de desarrollo (software) gratis, libre y multiplataforma mediante el que se escribe el programa y se carga en el microcontrolador.

- Un lenguaje de programación libre, basado en una forma bastante sencilla de C, el cual se explicará más adelante con la descripción del programa a cargar en nuestro microcontrolador.

PROGRAMAR EN ARDUINO

Arduino tiene una estructura de programación muy sencilla y dividida en tres partes: inicialización de variables y de registros, una función de setup que ejecuta sentencias del programa al arrancar el microcontrolador (encabezado con la sentencia void setup()), y una tercera parte en la que se repiten las mismas sentencias en bucle hasta que el dispositivo se desconecte o se reinicie (encabezado con la sentencia void loop())[2].

Todo esto tiene un nivel muy básico para que cualquier usuario con conocimientos básicos sobre programación sea capaz de crear pequeños programas con utilidades sencillas. Como ya se ha mencionado, el lenguaje Arduino es una selección de comandos básicos extraídos de C, pero el microcontrolador será capaz de reconocer y ejecutar cualquier programa escrito en C.

[1] *Arduino. Curso práctico de formación.* Oscar Torrente Artero (Ed. RC Libros), pág. 63
[2] *Arduino. Curso práctico de formación.* Oscar Torrente Artero (Ed. RC Libros), pág. 154

En el entorno de Arduino a los programas que se cargan en el microcontrolador se les denomina "sketches", así que en lo sucesivo, un sketch es un programa escrito en Arduino. Así podremos diferenciarlo del programa de MATLAB.

COMUNICACIÓN DE ARDUINO CON PERIFÉRICOS

Arduino se comunica con los distintos componentes electrónicos periféricos mediante puerto serie, esto es, los paquetes de información son transmitidos bit a bit por el mismo canal. Existen muchos protocolos de comunicación, Arduino puede trabajar con dos de ellos, que son el I^2C (Inter-Integrated Circuit), también conocido como TWI (Two WIres); y el SPI (Serial Peripheral Interface). Empleamos el I^2C ya que alguno de los periféricos con los que se trabaja en este proyecto sólo tolera éste protocolo[3].

El protocolo de comunicación empleado es I2C, se basa en dos canales de transmisión de datos, ambos de doble sentido (emisión y recepción). Uno de ellos es el SCL (Serial CLock), a través del cual se transmite la señal del reloj del microcontrolador para sincronizar la comunicación entre los dispositivos que se comunican. El otro es el SDA (Serial DAta), por el que circulan los datos (bits) en ambos sentidos.

En la mayoría de las placas de Arduino, la comunicación I^2C se realiza a través de los pines A4 (SDA) y A5 (SCL), dicha comunicación se programa mediante una serie de librerías oficiales de las que se hablará más adelante en el desarrollo del sketch.

MATERIAL EMPLEADO

El microcontrolador empleado para este proyecto ha sido el Arduino Nano, dado su reducido tamaño, su capacidad de procesado de datos y por su versatilidad modular (incluye unos pines macho como prolongación de los pines hembra para poder conectar en una protoboard o en demás dispositivos con capacidad de inserción.

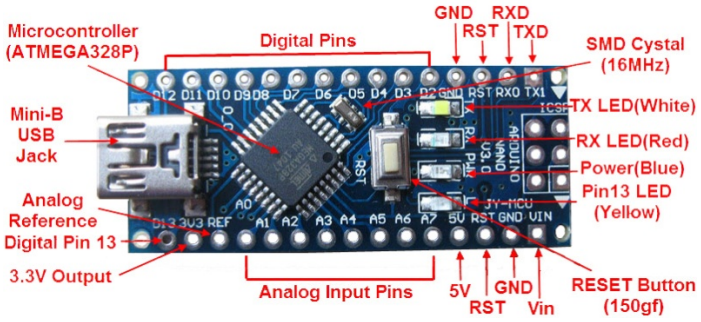
A este controlador se ha conectado una unidad de medida inercial (Inertial Measurement Unit, IMU), de nueve grados de libertad: el IMU Sensor Stick 9DOF de Sparkfun. Éste reúne en un solo dispositivo el acelerómetro ADXL345, el giróscopo ITG3200 y el magnetoscópio HMC5883L. Se accede a los distintos datos que éstos proporcionan mediante el protocolo de comunicaciones I2C, a través de las distintas direcciones de los registros de cada uno de ellos (se explicará más adelante).

[3] *Arduino. Curso práctico de formación.* Oscar Torrente Artero (Ed. RC Libros), pág. 79

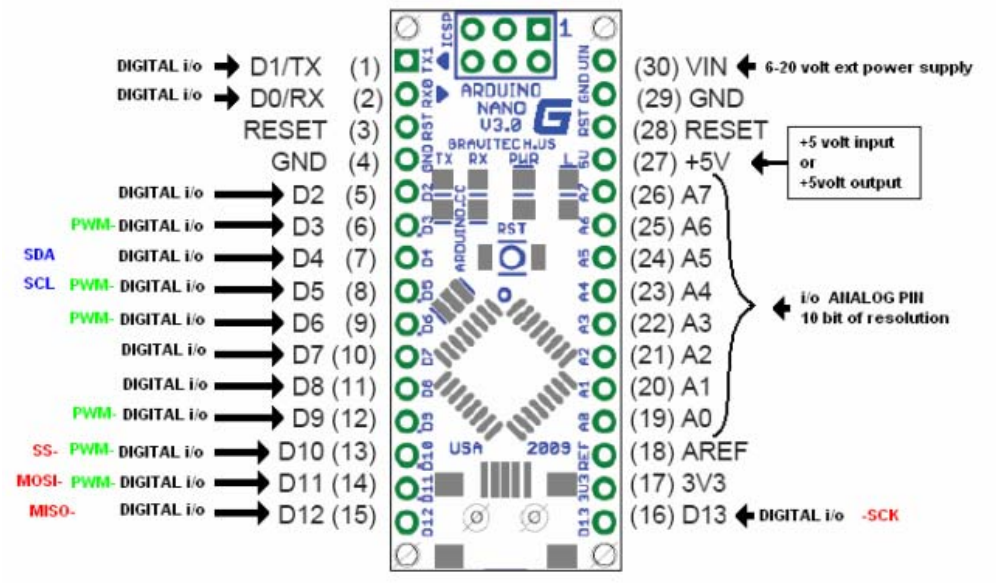
Éstos fueron los únicos dispositivos empleados inicialmente, teniendo en cuenta que el coste del IMU es de unos 100€ y el de Arduino Nano de unos 30€, hace un coste inicial de 130€. Sin embargo, dejaba mucho que desear el hecho de que hubiera cables de por medio, ya que esto no permite al dispositivo una plena libertad de movimiento. Es por ello que posteriormente se han añadido a la lista dos módulos wifi Xbee (emisor y receptor), un intérprete para el receptor (Xbee USB Explorer), y un soporte-puente (conocido en el entorno de Arduino como "shield") para el Arduino Nano. Esta ampliación supone un coste de 80€. La suma total es de 210€, un coste muy reducido si consideramos el resultado final.

ARDUINO NANO REV 3.0

Dentro de la amplia gama de placas que ofrece Arduino, ha sido elegida Arduino Nano por su tamaño (es una de las placas con mejor relación entre su tamaño y su procesador). Las versiones de Arduino Nano anteriores a 3.0 tenían el microcontrolador ATmega168, pero la 3.0 tiene el ATmega328 (propiedades en la página siguiente)



ESQUEMA DE ARDUINO NANO



ESQUEMA DE LA DISTRIBUCIÓN Y FUNCIÓN DE LOS DISTINTOS PINES DE ARDUINO NANO

ESPECIFICACIONES[4]

Microcontrolador	Atmel ATmega168 o ATmega328
Tensión de Operación (nivel lógico)	5 V
Tensión de Entrada (recomendado)	7-12 V
Tensión de Entrada (límites)	6-20 V
Pines E/S Digitales	14 (de los cuales 6 proveen de salida PWM)
Entradas Analógicas	8
Corriente máx por cada PIN de E/S	40 mA
Memoria Flash	16 KB (ATmega168) o 32 KB (ATmega328) de los cuales 2KB son

[4]http://www.arduino.cc/es/Main/ArduinoBoardNano

	usados por el bootloader
SRAM	1 KB (ATmega168) o 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) o 1 KB (ATmega328)
Frecuencia de reloj	16 MHz
Dimensiones	18,5mm x 43.2mm

ALIMENTACIÓN

Arduino Nano tiene distintas formas de alimentación:

-Mediante el cable USB Mini-B, a través del cual se aportan 5V exactos

-Mediante una fuente externa no regulada de 6 a 20 V (recomendado de 7 a 12V) a través del pin 30

-Mediante una fuente externa regulada de 5V a través del pin 27.

La placa selecciona automáticamente la entrada de mayor tensión.[5]

MEMORIA

El ATmega328tiene una memoria flash de 32KB para almacenar código (de los cuales 2KB son usados por el bootloader). Además, tiene 2KB de SRAM y 1KB de EEPROM (dicha memoria puede ser controlada, es decir, leída y escrita, mediante la librería EEPROM).[6]

ENTRADA Y SALIDA

Arduino Nano tiene 14 pines digitales (pueden ser empleados como entrada o salida), controlables mediante las funcionespinMode(), digitalWrite(), y digitalRead(). Estos pines trabajan a 5 voltios y pueden proveer o recibir un máximo de 40mA. Tienen una

[5]http://www.arduino.cc/es/Main/ArduinoBoardNano
[6]http://www.arduino.cc/es/Main/ArduinoBoardNano

resistencia de pull-up (desconectada por defecto) de 20 a 50 kΩ. Enumeramos en adelante algunos de estos pines con funciones especiales:

- **Serial: 0 (RX) y 1 (TX).** (RX) Actúa como receptor y (TX) que actúa como transmisor de datos TTL vía serie. Estos pines están conectados a los pines correspondientes del chip USB-a-TTL de FTDI. Así mismo, son los pines que empleamos para conectar Arduino Nano al módulo Xbee para la transmisión Wireless de los datos.
- **Interrupciones Externas: pines 2 y 3.** Estos pines pueden ser configurados para activar una interrupción por paso a nivel bajo, por flanco de bajada o flanco de subida, o por un cambio de valor. Son controlados mediante la funciónattachInterrupt().
- **PWM: pines 3, 5, 6, 9, 10, y 11.** Proveen de una salida PWM (Pulse-width modulation) de 8-bits cuando se usa la función analogWrite().
- **SPI: pines 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** Estos pines soportan la comunicación SPI (protocolo de comunicación de cuatro canales). Recientemente se ha creado una nueva librería de Arduino para controlar esta comunicación.
- **LED: Pin 13.** Existe un LED conectado al pin digital 13. Cuando el pin se encuentra en nivel alto, el LED está encendido, cuando el pin está a nivel bajo, el LED estará apagado.

De todos los pines descritos anteriormente, los únicos que se han empleado en este proyecto han sido los pines de serial 0 y 1. El resto de pines empleados son los pines de tierra, de tensión (Vin y 3V3) y los pines para la comunicación I^2C descritos a continuación.

Arduino Nano tiene 8 entradas analógicas que proveen de 10 bits de resolución (1024 valores diferentes). Por defecto miden entre 5 voltios y masa, sin embargo es posible cambiar el rango superior usando la función analogReference() (que utiliza la tensión del pin AREF como tensión de referencia). De todos los pines analógicos, los empleados en nuestro proyecto han sido los necesarios para la comunicación I^2C: Pines 4 (SDA) y 5 (SCL). Soporta comunicación I^2C (TWI) usando la librería Wire (se verá más adelante en la programación).

Otros pines de la placa son:

- **AREF.** Tensión de referencia por las entradas analógicas. Se configura con la función analogReference().
- **Reset.** Pon esta linea a baja tensión para resetear el microcontrolador. Normalmente se usa para añadir un botón de Reset que mantiene a nivel alto el pin Reset mientras no es pulsado. Para hacer que funcione el botón, se conecta entre el pin de Reset y un pin de tierra (GND).
- **GND.** Hay dos pines GND que sirven como conexión a tierra.[7]

COMUNICACIÓN

Arduino Nano puede comunicarse con un PC, otro Arduino o con otros microcontroladores. ElATmega328tiene un módulo UART que funciona con TTL (Transistos-Transistor Logic) a 5V, el cual permite una comunicación vía serie que está disponible usando los pines 0 (RX) y 1 (TX). El chip FTDI FT232RL en la placa hace de puente a través de USB para la comunicación serial y los controladores FTDI (incluidos con el software de Arduino) proveen al PC de un puerto com vitual para el software en el PC. El software Arduino incluye un monitor serial que permite visualizar en forma de texto los datos enviados desde y hacia la placa Arduino. Los LEDs RX y TX en la placa parpadearán cuando los datos se estén enviando a través del chip FTDI y la conexión USB con el PC (pero no para la comunicación directa a través de los pines 0 y 1).

La librería SoftwareSerial permite llevar a cabo una comunicación serie usando cualquiera de los pines digitales del Nano.

A su vez, y como ya se ha indicado, elATmega328 también soporta comunicación I2C (TWI) y SPI. El software Arduino incluye la librería Wire para simplificar el uso del bus I2C. [8]

PROGRAMACIÓN

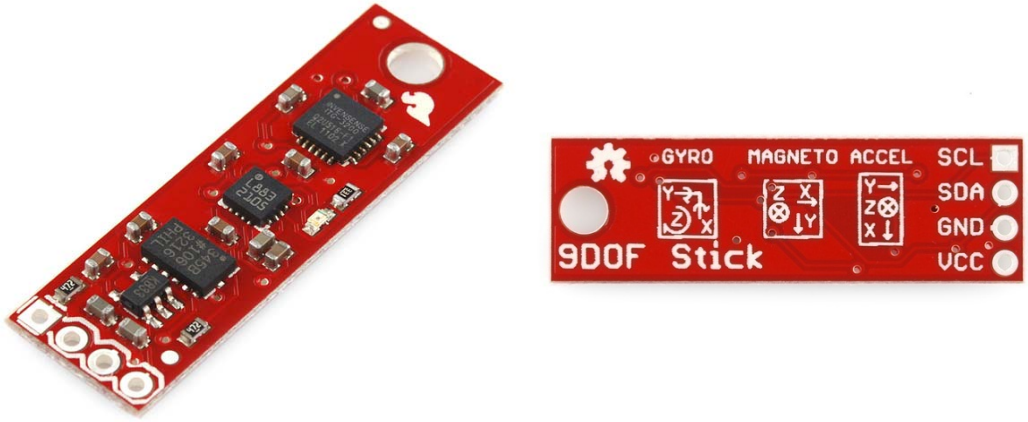
El Arduino Nano puede ser programado con el software de Arduino. El ATmega328 del Arduino Nano viene preprogramado con un bootloader que te permite subir tu código al Arduino sin la necesidad de un programador externo. Se comunica usando el protocolo STK500.

[7]http://www.arduino.cc/es/Main/ArduinoBoardNano
[8]http://www.arduino.cc/es/Main/ArduinoBoardNano

También se puede programar el microcontrolador usando un programador ICSP (In-Circuit Serial Programming, Pogramación Serie En-Circuito[9].

IMU SENSOR STICK 9DOF

El IMU Sensor Stick 9DOF contiene, como ya se ha dicho con anterioridad, un acelerómetro ADXL345, un giróscopo ITG3200 y un magnetoscopio HMC5883L. Cada uno mide aceleración, velocidad angular y campo magnético en tres ejes. Además, dada la sensibilidad de la ganancia del giróscopo a la temperatura, éste mide dicha temperatura con la intención de recalcular constantemente su ganancia como función de la temperatura. Son en total diez datos que el sensor mide en forma de variaciones de tensión eléctrica según sus distintos mecanismos.



SENSOR STICK 9 DOF

El sensor tiene cuatro conexiones: tensión (obligatoriamente 3V3 regulados), tierra, SDA y SCL (para la comunicación I^2C). En cuanto se establecen las conexiones de tensión y tierra, los tres sensores incluidos en este dispositivo establecen su configuración inicial y comienzan a enviar datos al canal SDA. Para poder configurar dichos sensores existen varios registros hexadecimales a los que se accede mediante la programación en Arduino. Los registros de escritura (para configurar los sensores) y de lectura (para leer los datos de cada sensor) vienen dados en las datasheets de los tres sensores incluidos en nuestro dispositivo. Dichas datasheets vienen adjuntas en el anexo de este trabajo, aunque

[9]http://www.arduino.cc/es/Main/ArduinoBoardNano

los registros empleados son enunciados y especificados en el apartado de la programación de nuestro sistema electrónico en su conjunto.

Por desgracia, y como se ha podido entrever en la explicación del funcionamiento de los pines analógicos de Arduino Nano, cuando éste recibe los datos del IMU, no tienen nada que ver con lo que buscamos, son diferencias de tensión causadas por los movimientos de los distintos sistemas electromecánicos de los sensores.

Para realizar la conversión de estos valores a lecturas válidas de los sensores, se deben aplicar una serie de ganancias y offsets, algunos obtenidos en las datasheets de los sensores y otros experimentalmente. Este proceso de interpretación de datos se podría hacer en el Arduino Nano, pero le restaríamos capacidad y el proyecto se extendería sobremanera. Por ello se ha preferido enviar los datos en bruto leídos en el IMU directamente al PC y procesarlos e interpretarlos mediante MATLAB.

MÓDULOS XBEE

Aunque la primera versión del dispositivo fue básicamente la conexión del IMU con Arduino Nano, conectado al PC mediante un cable USB que cumplía las funciones tanto de alimentación del dispositivo como de comunicación con éste, se insistió en la implementación de dos módulos Wireless que dotaran al dispositivo de la capacidad de comunicarse con el PC sin la necesidad de haber cables de por medio, ya que éste reducía la libertad del dispositivo a moverse y avanzar con las olas.

Los módulos XBee empleados han sido los más simples que hay, para reducir al máximo el consumo de las pilas empleadas. Éstos son los XBee 802.15.4 Low Power Module con antena de cable (nombre oficial del producto XB24-AWI-001).

ESPECIFICACIONES

Fabricante	Digi Internacional
Categoría de productos	Módulos Zigbee / 802.15.4
Banda de frecuencia	2.4 GHz
Rango de línea visual	100 ft, 300 ft
Frecuencia de datos	250 kbps

Sensibilidad	-92dBm
Voltaje operativo de suministro	2.8 V a 3.4 V
Corriente de suministro en recepción	45 mA
Corriente de suministro en transmisión	50 mA
Energía de salida	1 mW
Temperatura operativa máxima	+ 85 ºC
Temperatura operativa minima	-40 ºC
Dimensiones	27.61 mm x 24.38 mm

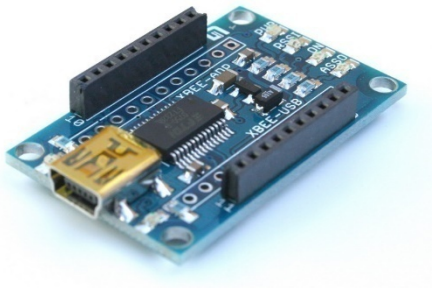
MÓDULO XBEE

Ya que la configuración de los módulos XBee se realiza conectado cada módulo al XBee USB Adapter y programándolo con su software específico, explicaremos la configuración de éstos en el siguiente apartado.

XBEE USB ADAPTER

El XBee USB Adapter es el dispositivo que sirve para conectar el módulo XBee al PC para programarlo y configurarlo. X-CTU es el software mediante el cual se programa cada módulo XBee. La programación de ambos es bien sencilla: el módulo XBee conectado al

dispositivo inalámbrico medidor de olas será el emisor, para ello habrá que configurarlo como “End-device”, y le especificaremos el número de serie del otro módulo XBee para que éstos se encuentren automáticamente al conectarse. Además debemos especificar la velocidad de transmisión de datos (Baudrate), que se ha establecido en el máximo permitido por nuestro dispositivo final montado (57600 baudios).



XBEE USB ADAPTER

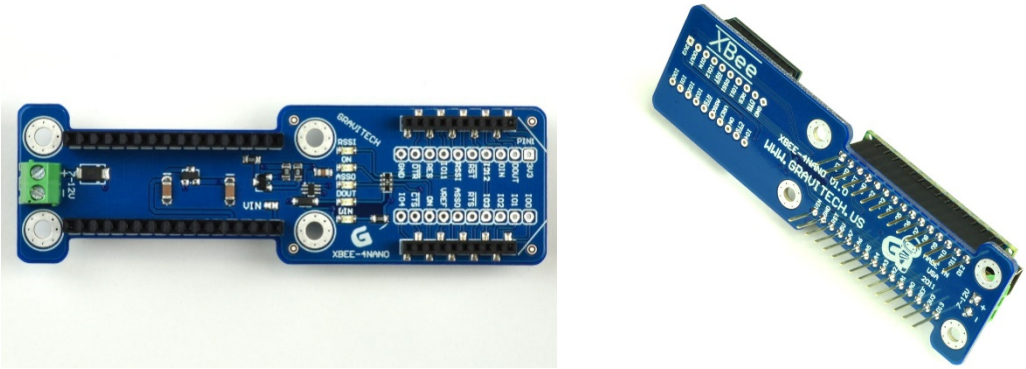
Así mismo, el XBee receptor, que irá conectado al XBee USB Adapter y éste directamente al PC para comunicar los datos, tendrá la configuración de dispositivo “Coordinator”, con el número de serie del XBee emisor, y la velocidad de transmisión de datos que es la misma en ambos casos (57600 baudios).

Finalmente, el intérprete de los datos puede ser desde el mismo IDE (Integrated Development Environment) de Arduino, el software X-CTU de configuración de los módulos XBee, o directamente en nuestro caso, MATLAB, es decir, cualquier programa con capacidad de comunicación puerto-serie será capaz de leer los datos recibidos por el XBee USB Adapter.

Por si la conexión de los dispositivos no ha quedado clara, más adelante hay un esquema con todos los dispositivos conectados, formando el sistema medidor de olas final.

XBEE ADD ON FOR ARDUINO NANO

Para conectar el módulo XBee al conjunto Arduino + IMU, ha sido necesario lo que en el entorno de Arduino se conoce como Shield (Escudo), es decir, una placa que conecta los dispositivos mencionados entre sí.

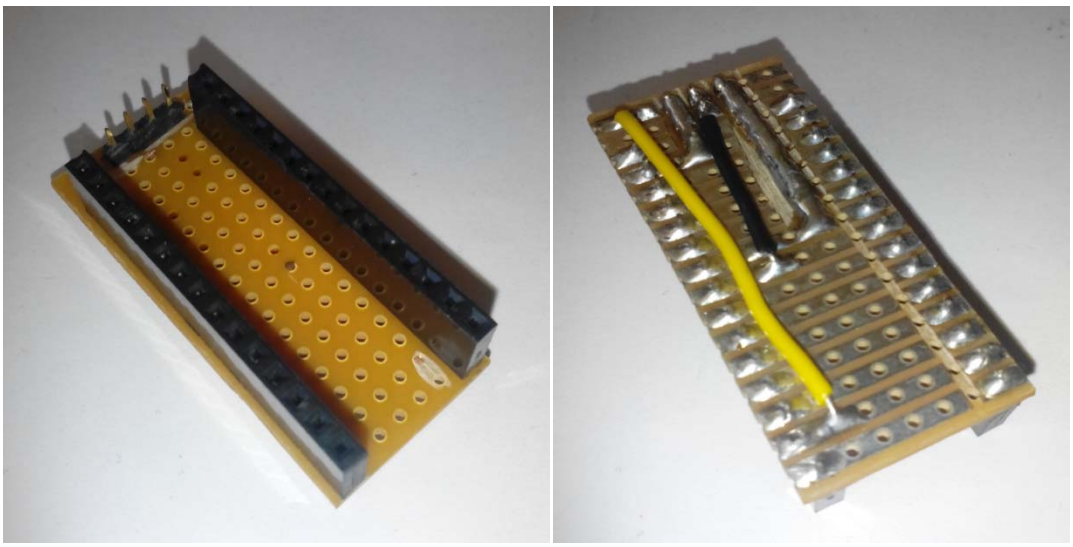


XBEE ADD ON FOR ARDUINO NANO

Éste shield es plug&play, es decir, no necesita ningún tipo de programación.

PCB DE CONEXIÓN ARDUINO + IMU

Para conectar el IMU al Arduino Nano tuve que diseñar y crear una PCB casera (buscando la mayor profesionalidad posible) que hiciera de soporte. Sus dimensiones son las mismas que las de Arduino Nano.



PCB DE CONEXIÓN ARDUINO NANO + IMU, DE FABRICACIÓN CASERA

CONEXIÓN DE LOS DISPOSITIVOS: DIAGRAMA Y FOTOS

La conexión entre todos los dispositivos principales se muestra en el siguiente diagrama:

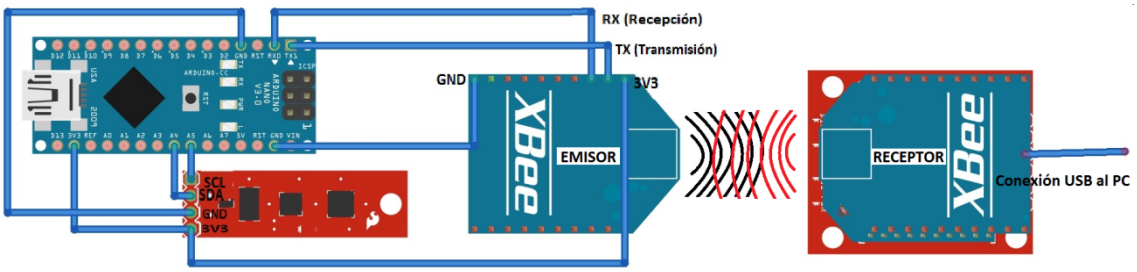
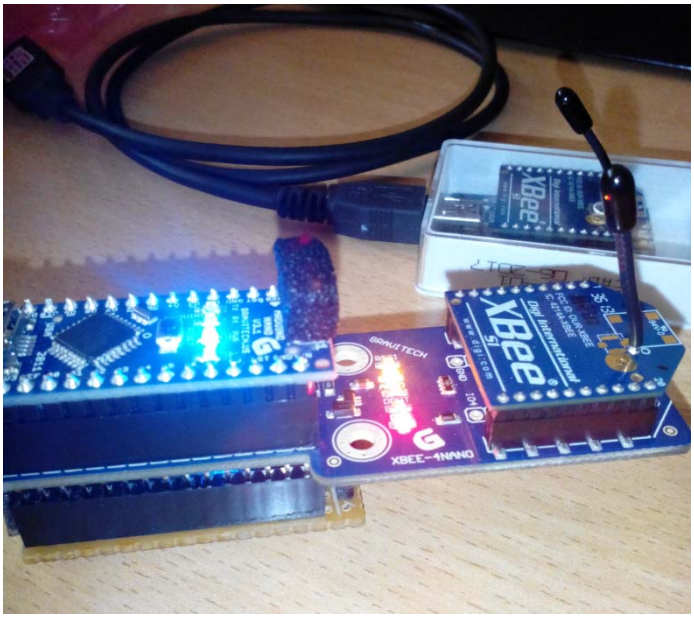


DIAGRAMA DE CONEXIONES[10]

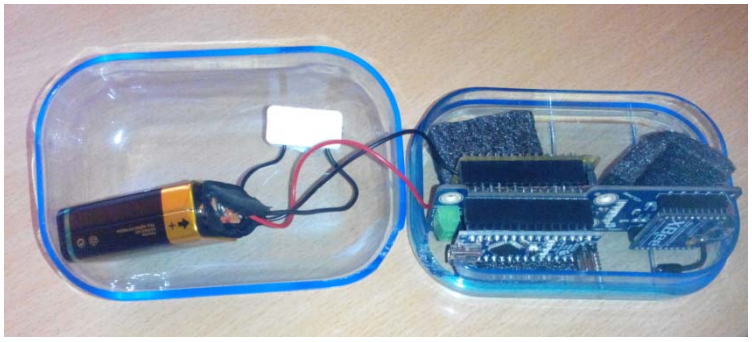
Sin embargo, estas conexiones se hacen prácticamente solas gracias a que todos los dispositivos son modulares. A continuación adjuntaré una serie de fotos del proyecto:



Delante se puede observar el dispositivo medidor al completo: Arduino Nano, Módulo XBee, Shield para el XBee, y el IMU oculto abajo a la izquierda tras los pines hembras de la PCB de conexión Arduino + IMU.

Además, se puede apreciar en la misma foto que justo detrás se encuentra el receptor (Módulo XBee ensamblado sobre el XBee USB Adapter), en su carcasa casera (fabricada a partir de una caja de tapones para los oídos). Los orificios para el cable y la antena fueron realizados con una taladradora pequeña (Dremel).

[10]Diagrama realizado con el programa de software libre Fritzing.

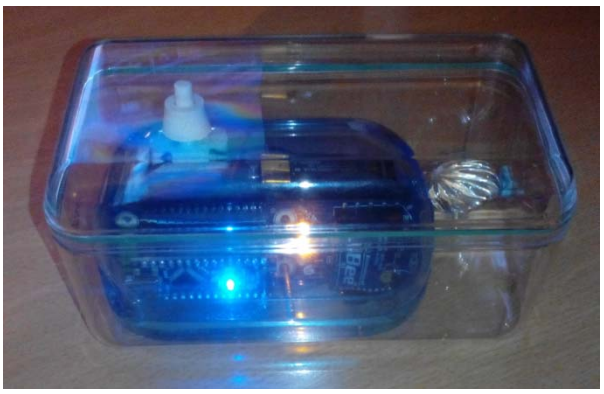


Así quedó el dispositivo final con su carcasa abierta (incluye pila 9V e interruptor externo).

Dispositivo con la carcasa cerrada y encendido. Se aprecian las distintas luces que indican el encendido y los distintos canales de transmisión de datos activos.

Aquí se puede apreciar el tamaño final del dispositivo, del tamaño de la palma de mi mano.

Finalmente, una última foto con el dispositivo introducido en la que es su carcasa externa: una caja de bombones que hará la función de “barco” a la hora de situar sobre la superficie del agua. Se encuentra desplazado para alcanzar el equilibrio y que no se desbalancee en el agua.



MATLAB

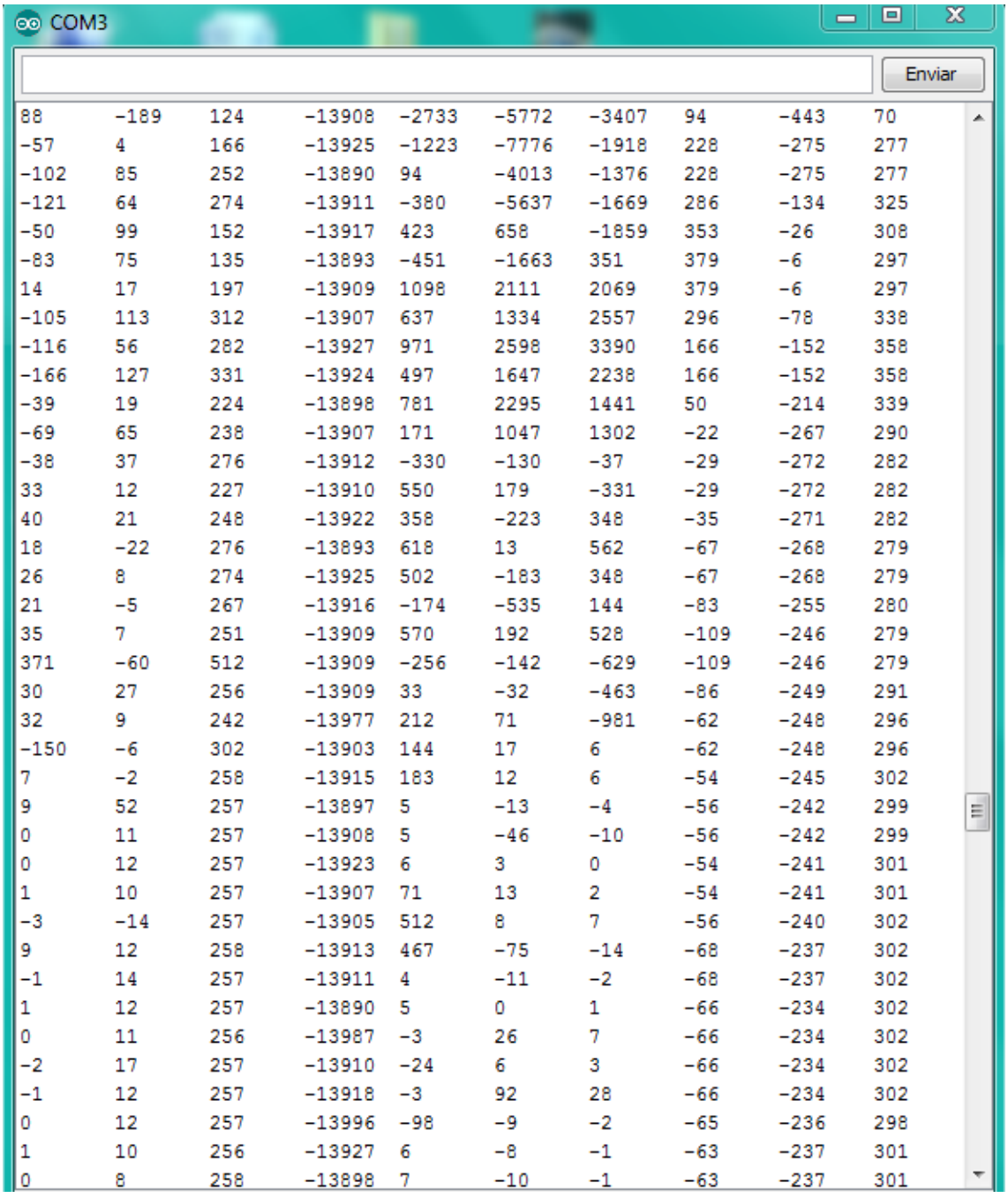
Como se ha dicho anteriormente, una de las opciones para procesar los datos era realizarlo desde el mismo Arduino Nano, pero habría reducido notablemente la velocidad a la que se procesarían los datos de los sensores. Es por ello que en el diseño final se transmiten directamente al PC sin procesar. Sin embargo, se barajó una opción diferente: desde Arduino Nano se podían enviar los datos sin procesar a otro Arduino, Arduino Due, que tiene un microcontrolador mucho más rápido y potente (32 bits frente a los 8 bits de Arduino Nano), que los procesara y enviara al PC para que MATLAB los graficara directamente.

¿Por qué no se hizo así? Demasiado excesivo para obtener los mismos resultados procesando los datos directamente desde el PC.

FUNCIONAMIENTO DEL PROGRAMA

Para entender el código en MATLAB lo primero que debemos entender es qué datos nos está enviando el IMU. Es por ello que vamos a proceder al análisis de varias secuencias de datos interpretados por el IDE de Arduino.

Son de una de las primeras versiones delsketchde Arduino, más preparada para entender los datos cuando el proyecto estaba en sus inicios que para la comunicación puerto-serie con MATLAB. En esta pantalla se aprecian algunos datos, resultado de agitar el IMU rápidamente y dejarlo en reposo:



DATOS ENVIADOS POR EL DISPOSITIVO

Cada fila es un paquete de datos, de izquierda a derecha son aceleraciones (x, y, z), temperatura del giróscopo, velocidades angulares (x, y, z), y campo magnético (x, y, z). Nueve datos en total (en las primeras versiones del sketch también se leen los datos del magnetoscopio porque no sabía que no iban a ser necesarios para el desarrollo del programa, aunque lo consideré necesario en una primera instancia). Se puede ver que los datos son números enteros que nada tienen que ver con las unidades que nosotros conocemos, en la última parte (reposo), se ve que la aceleración en z vale 258, que no corresponde a la aceleración de la gravedad ni medida en m/s^2, ni medida en g; al igual que

la temperatura del giróscopo ni las velocidades angulares medidas tampoco se corresponden con las unidades conocidas.

¿A qué corresponden exactamente estos números? Pues a la relación directamente proporcional entre el rango de cada sensor y la capacidad de transmisión de este. Veamos el ejemplo del acelerómetro: sabemos que en reposo está midiendo 1g en vertical (9'81 m/s²) y 0g en "x" y en "y", y sin embargo, aunque está imprimiendo prácticamente 0 en las aceleraciones horizontales, en lugar de 1g en la aceleración vertical está imprimiendo aproximadamente 256. Esto es completamente lógico si nos damos cuenta de que el acelerómetro está configurado a un rango de ±2g, ocupando 10 bits, es decir, 2^10=1024 posibles valores, que van desde -512 hasta 512. Por lo que si 2g corresponde a 512, -2g corresponde a -512, 256 corresponde a 1g.

En el apartado de ganancias y offsets explicaremos cómo se ha realizado el resto de análisis del resto de datos de sensores.

En realidad, los paquetes de datos no se envían uno detrás de otro en forma de filas tabuladas como en el sketch anterior. En el sketch final se inicia y finaliza el envío de los datos enviando desde el PC una "a" y una "b" respectivamente (a modo de interruptor), y se envía un dato por fila para optimizar la velocidad de los datos.

En lo sucesivo veremos cómo MATLAB solicita dichos paquetes y los procesa en los sucesivos apartados.

ESTABLECER LA CONEXIÓN PUERTO – SERIE

El fragmento de código que abre el puerto serie es el siguiente:

```
%% ABRIR EL PUERTO SERIE

puerto=get(handles.puerto,'String');
baudrate = cellstr(get(handles.baudrate,'String'));
baudrate = fix(str2double(baudrate(get(handles.baudrate,'Value'))));
s = serial (puerto,'DataBits',8,'StopBits',1,...
'BaudRate',baudrate,'Parity','none');
```

La variable "puerto" almacena la cadena de caracteres que contiene el nombre del puerto al que está conectado el receptor de datos XBee USB Adapter. "s" es la variable que almacena los datos del puerto que vamos a abrir: el nombre del puerto, el número de bits que se transmiten en serie (8 bits implica que los bits se transmiten de 8 en 8), el número

de bits que indican el final de un byte, la velocidad a la que se transmiten los bytes (el BaudRate)... Nosotros sólo vamos a variar el puerto y el BaudRate, aunque lo recomendado para nuestro programa serán 57600 baudios.

RECOGIDA DE DATOS

El código comienza inicializando una serie de variables: n es el tamaño máximo que obtendrá el vector de lectura de los datos, t es el tiempo durante el cual se recibirán los paquetes de datos pertinentes. Las variables que van desde ax hasta gz son los siete datos que envía el IMU. El bucle while se encarga de almacenar los datos recibidos en sus respectivas variables, se puede apreciar que el bucle comienza enviando una "a" (para establecer la comunicación con el emisor), seguido de la lectura del puerto serie siete veces. Dicha lectura viene dada en forma de cadenas de caracteres numéricos que hay que convertir con la sentencia str2double. Finalizada la recogida de datos, la última parte de este trozo de código se encarga de comprobar si el código ha leído los datos durante el tiempo estimado (en cuyo caso redimensiona las variables y el valor de n como el total de datos leídos por cada variable) o ha finalizado antes por haber alcanzado todas las variables su tamaño máximo "n".

```
%% LECTURA DE DATOS

n=50000;
t=get(handles.tiempomedirbar,'Value');

fopen(s);

i=1;
j=1;
ax=zeros(n,1);
ay=zeros(n,1);
az=zeros(n,1);
temp=zeros(n,1);
gx=zeros(n,1);
gy=zeros(n,1);
gz=zeros(n,1);

disp('Recibiendo datos en')
disp('3')
pause(1)
disp('2')
pause(1)
disp('1')
```

```
pause(1)
disp('Recibiendo...')
tic

fprintf(s,'%c','a');

while (i<=n && toc<=(1.1*t))

ax(i)=str2double(fgetl(s));
ay(i)=str2double(fgetl(s));
az(i)=str2double(fgetl(s));
temp(i)=str2double(fgetl(s));
gx(i)=str2double(fgetl(s));
gy(i)=str2double(fgetl(s));
gz(i)=str2double(fgetl(s));
i=i+1;

end

disp('Finalizada la recogida de datos')

if i==n
    t=toc;
else
ax=ax(1:i-1);
ay=ay(1:i-1);
az=az(1:i-1);
gx=gx(1:i-1);
gy=gy(1:i-1);
gz=gz(1:i-1);
temp=temp(1:i-1);
    n=i-1;
end
if ~isempty(instrfind)
tic
while (toc<=3)
fprintf(instrfind,'%c','b');
end
fclose(instrfind);
delete(instrfind);
end
```

PROCESADO: GANANCIAS Y OFFSETS

En esta parte del código vemos cómo obtenemos (dentro de la sección de calibrado) el diferencial de tiempo (tiempo transcurrido entre datos) que será necesario para realizar las integraciones pertinentes a la aceleración y a la velocidad, las ganancias por las que hay que dividir los datos recibidos del acelerómetro, la transformación de la temperatura a grados centígrados para poder utilizarla en la obtención de la ganancia del giróscopo, la propia ganancia del giróscopo en función de la temperatura y los offsets que hay que restar a las velocidades del giróscopo:

```
dt=(1.1*t)/n;

set(handles.diftemp, 'String', num2str(dt))

set(handles.frecmuest, 'String', num2str(1/dt))

temp=35-(-13200-temp)./280;

gain_gyr=14.375+((15.3733-14.375)/(32-25)).*(temp-25);

gain_ac=str2double(get(handles.gananciaacel,'String'));

ax=ax/gain_ac;
ay=ay/gain_ac;
az=az/gain_ac;
gx=(gx-offset_gyr(1))./gain_gyr;
gy=(gy-offset_gyr(2))./gain_gyr;
gz=(gz-offset_gyr(3))./gain_gyr;
```

Para entender mejor este código debemos entender que los datos obtenidos para la calibración correcta del IMU es necesario que éste esté en reposo y en posición completamente horizontal.

GANANCIA DEL ACELERÓMETRO

La ganancia del acelerómetro es bastante sencilla, ya que al estar en reposo y en posición horizontal, las componentes x e y de la aceleración deben ser nulas, y la componente z debe medir 1g, o lo que es lo mismo, 9'81 m/s^2. Aún para no despreciar el error humano que cometemos al posicionar el dispositivo en horizontal de forma casi precisa, tenemos en cuenta las aceleraciones horizontales para obtener la ganancia: al realizar la norma de la media de las tres componentes, obtenemos la ganancia, que debe

dar un valor similar a 256 (1g, valor de la gravedad como norma). Sólo es cuestión de dividir entre dicho valor los tres componentes de la aceleración para obtener los valores reales de la aceleración medida en "g" (para obtenerla aceleración relativa al IMU). Más adelante, tras aplicar la transformación de orientación y obtener la aceleración medida en ejes absolutos, se procederá a multiplicar por 9'81 m/s^2 antes de integrarla para obtener la velocidad.

OBTENCIÓN DE LA TEMPERATURA

Antes de obtener la ganancia del giróscopo, debemos obtener previamente la temperatura. La temperatura sigue una regresión lineal de la cual nos dan en el datasheet del giróscopo tanto pendiente como el valor de la temperatura a 35ºC. Según el datasheet, la pendiente de la recta es 1/280, y el valor que imprime el sensor a 35º es 13200. Si denominamos Traw a la temperatura en bruto, la que envía el sensor, en LSB (Less Significant Bit), y Tc a la temperatura transformada a grados centígrados (la temperatura real):

$$(T_{ºC} - 35ºC) = (\frac{1}{280\frac{LSB}{ºC}}) * (T_{raw} + 13200\ LSB)$$

$$T_{ºC} = 35ºC + (\frac{1}{280\frac{LSB}{ºC}}) * (T_{raw} + 13200\ LSB)$$

GANANCIA DEL GIRÓSCOPO

La ganancia del giroscopio ha sido un poco más enrevesada de obtener. Dicho valor es muy sensible a la temperatura, y por desgracia, en todo el rango de temperatura en el que el sensor funciona (-40ºC, 80ºC), su dependencia respecto a esta no es lineal: varía mucho a muy altas temperaturas y muy poco a temperaturas bajas. Además, la ganancia también depende de la frecuencia de muestreo que hayamos seleccionado (en nuestro caso 8KHz) y del ancho del filtro de paso bajo que hayamos seleccionado dentro del sensor para filtrar sus datos (nuestro ancho de banda es de 256HZ).

Por suerte, después de muchas pruebas con el dispositivo, metiéndolo y sacándolo del congelador de mi casa (mi madre no entendía por qué debían compartir cajón los guisantes y mi proyecto) y haciendo mediciones, descubrí que en el rango de temperaturas en el que iba a actuar el sensor (entre 20ºC y 35ºC por lo que se llega a calentar el dispositivo cuando está conectado), con la frecuencia de muestreo y el ancho de banda establecidos, el comportamiento de la ganancia era lineal respecto de la temperatura.

Tan sólo me faltaban un par de valores para crear la regresión lineal. Uno de ellos lo aportaba la datasheet: la ganancia figura con un valor de 14'375 LSB/(º/s) a una temperatura de 25ºC, valor que experimentalmente se aproximó de forma bastante precisa (14'38 – 14'4). El otro valor lo obtuve tras alcanzar la temperatura máxima estacionaria; a 32ºC, con un valor de 15'3733 LSB/(º/s).

Antes de seguir explicando cómo hallar la función que relaciona ganancia y temperatura debo explicar primero cómo hallo datos exactos de ganancia en función de la temperatura. Para poder hallar dichas ganancias, la única manera que se me ocurrió fue crear un programa que obtenía la ganancia mediante un método bastante sencillo: girar con velocidad rápida y constante sobre un eje del dispositivo 180º exactos. El programa está configurado para medir velocidad angular con la ganancia estándar de la datasheet (14'375 LSB/(º/s)). Con el ángulo girado medido a la ganancia de 14'375 LSB/(º/s) y los 180º reales, se realiza una simple regla de tres:

$$GananciaGiróscopo = 14'375\,\frac{\text{LSB}}{(º/\text{s})} * \frac{GiroObtenido(º)}{180(º)}$$

A su vez, se debe leer la temperatura con el sensor del giroscopio para relacionarla con la ganancia.

Finalmente, la relación lineal ganancia – temperatura queda de la siguiente manera:

$$\left(GananciaGirós. - 14'375\,\frac{\text{LSB}}{º/\text{s}}\right) = \left(\frac{15'3733\text{LSB}/(º/\text{s}) - 14'375\text{LSB}/(º/\text{s})}{32ºC - 25ºC}\right) * (T - 25ºC)$$

$$GananciaGirós. = 14'375\,\frac{\text{LSB}}{º/\text{s}} + \left(\frac{15'3733\text{LSB}/(º/\text{s}) - 14'375\text{LSB}/(º/\text{s})}{32ºC - 25ºC}\right) * (T - 25ºC)$$

$$GananciaGirós. = 14'375\,\frac{\text{LSB}}{º/\text{s}} + 0'1426\,\frac{\text{LSB}}{(º/\text{s})ºC} * (T - 25ºC)$$

ORIENTACIÓN: CÓDIGO AHRS

PROBLEMAS EN LA ORIENTACIÓN

Cuando pensé por primera vez en el proyecto, supuse que la programación no sería tan complicada ya que, habiendo aprobado 5º de Ingeniería Industrial, integrar las aceleraciones para obtener la posición del dispositivo sería de lo más sencillo. Pero el día que caí en la cuenta de que el IMU medía tanto aceleraciones como giros relativos a sí mismo, y no a unos ejes absolutos casi dejo el proyecto. Estuve días pensando ¿cómo voy a obtener la orientación del IMU para hallar la aceleración en ejes absolutos?

El principal problema se plantea en que, si queremos resolverlo por el camino más simple, nos encontramos con lo siguiente: para obtener el giro del dispositivo se puede integrar el giro en los tres ejes y obtener el ángulo girado en cada eje:

- En cada paso de integración, el ángulo girado por cada eje lo hace en una dirección distinta, por lo que no sólo hay que sumar el ángulo girado a todos los anteriores, sino que encima hay que aplicarle la transformación de los giros acumulados. ¡Y sólo estamos hablando del giróscopo, no he mentado el acelerómetro!
- Cada paso de integración acumula un "pequeño" error, que pasados los 10 pasos de integración, dicho error se dispara y, de unos 3200 datos tomados en 20 segundos, sólo los 10 primeros se aproximan a la realidad.

Fue entonces cuando comencé a investigar sobre filtros de estimación y corrección. Aunque pueda parecer sencillo, fueron los dos peores meses de investigación sobre el proyecto porque no tuve en absoluto la sensación de avanzar: por cada día que leía sobre el tema descubría que necesitaba saber algo completamente diferente para entender dichos filtros. Entender, por ejemplo, cómo funciona un filtro de Kalman aplicado a un oscilador autobalanceable me llevó a un programa que aplicaba Kalman a una sóla variable (velocidad de giro) y para la cual necesitaba ¡otras dos variables más para estimar el error y corregirlo! Si yo tenía tres velocidades de giro (x, y, z) que integrar, estimar sus respectivos errores, y corregirlos, iba a necesitar otras seis variables más, lo que implicaba utilizar sólo para calcular la orientación (hablamos de rotaciones, no hemos entrado en traslaciones) los nueve grados de libertad que me aportaba el Sensor Stick 9DOF (Degrees Of Freedom).

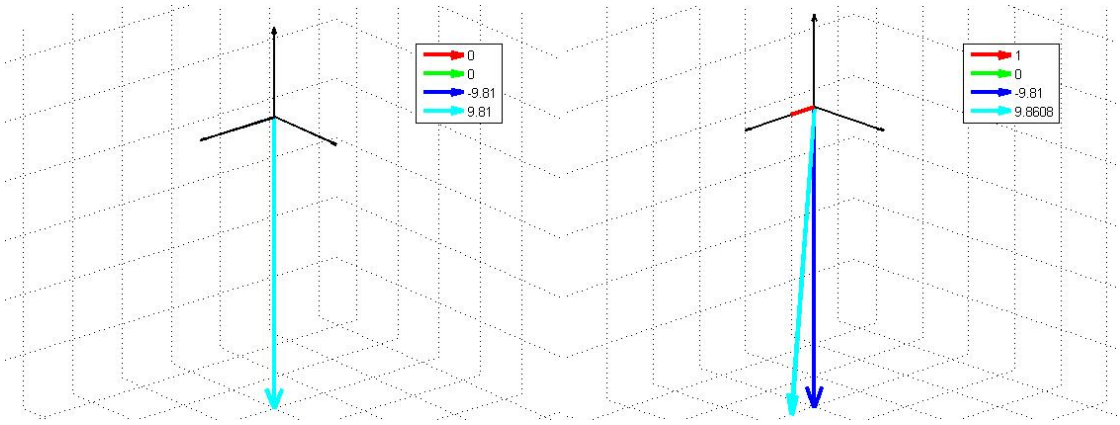
Programar aquello parecía insano, así que seguí investigando y descubrí que para poder llevar a cabo aquello no necesitaba los tres datos del magnetoscopio si utilizaba bien los datos del acelerómetro y del giroscopio para estimar la orientación: la clave estaba en entender los cuaterniones, que son un tipo de números hipercomplejos (números complejos en tres dimensiones).

Y finalmente descubrí que, si las transformaciones de giro que hay que hacer por cada paso de integración, las hacía por cuaterniones en lugar de por matrices de giro, por ángulos de Euler o cualquier otro método; dichas transformaciones de giro quedaban reducidas al producto de dos cuaterniones (que aunque parezca fácil tiene su complejidad, aunque nada tiene que ver comparada con las complicaciones de las matrices rotacionales).

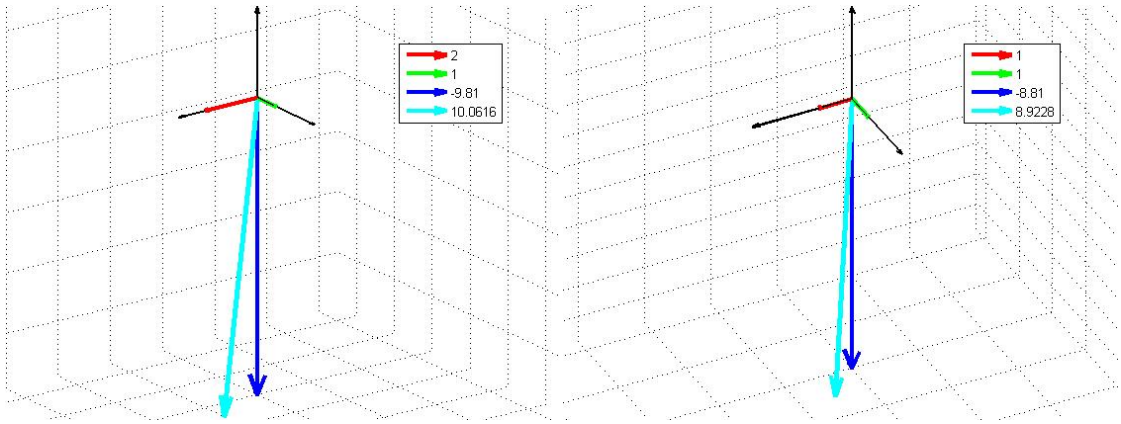
ORIENTACIÓN DEL DISPOSITIVO: ESTIMACIÓN, MEDICIÓN Y CORRECCIÓN

Para poder hablar de de orientación, debemos hablar de AHRS (Attitude Heading Reference System). Se refiere al sistema formado por una serie de sensores microelectromecánicos (MEMS) como acelerómetro, giróscopo y magnetoscopio, que proporcionan la orientación de estos en todo momento. La diferencia entre un AHRS y un IMU es que el segundo no nos da la orientación, sólo los datos que leen los sensores. Así que está en mi mano conseguir un código que interprete los datos del IMU para transformarlos a AHRS.

Profundicemos un poco más en la matemática oculta tras el código para la orientación del IMU. Si situamos el medidor de olas sobre la superficie acuática del generador de olas, sabemos de antemano que las aceleraciones máximas que el dispositivo puede alcanzar no superarán los 2 m/s^2 (realmente eso sería demasiado). Si tenemos en cuenta que la aceleración de la gravedad está entorno a los 10 m/s^2, que es un 500% de lo que acabamos de considerar la aceleración máxima del dispositivo, podemos considerar en la mayoría de los casos, que el sentido de la aceleración será cercano a la vertical hacia abajo:

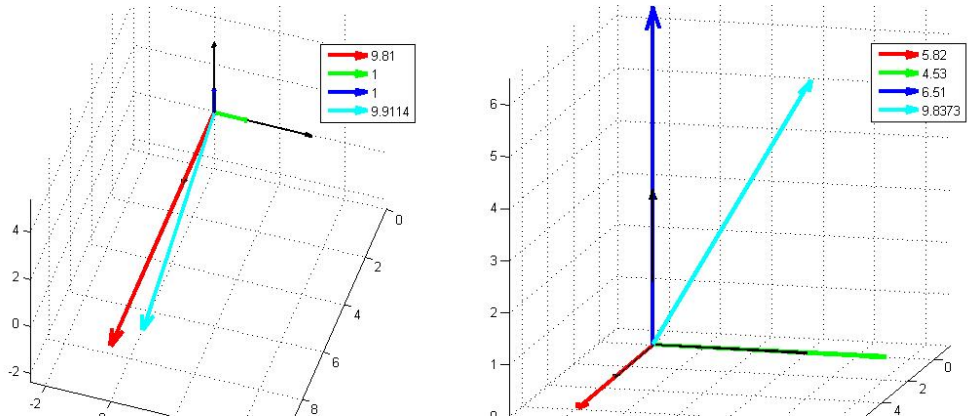


IMU EN REPOSO $AX=1M/S^2$



$AX=1M/S^2$ $AY=1M/S^2$ $AX=1M/S^2$ $AY=1M/S^2$ $AZ=1M/S^2$

De esta manera, el sentido de la aceleración indicará prácticamente la dirección vertical:



EJEMPLOS DE IMU GIRADO: A LA IZQUIERDA HA GIRADO 90º SOBRE EL EJE Y, A LA DERECHA UN GIRO BASTANTE ALEATORIO

Se podría crear un código que supusiera que obteniendo la aceleración total del dispositivo (módulo, dirección y sentido), podríamos conseguir automáticamente la orientación del IMU mediante matrices rotacionales ó ángulos de Euler. Sin embargo, el error que induce el hecho de despreciar las aceleraciones provocadas por las traslaciones del IMU no es despreciable, por lo que debemos tener más factores en cuenta.

El giróscopo es relativo a sí mismo, por lo que no tiene sentido integrarlo respecto al tiempo para obtener sus ángulos girados. Sin embargo, cada medida está referida a la anterior; si dicha medida de ángulos girados mediante giróscopo de un paso al otro se compara mediante un algoritmo a la medida de los ángulos girados del acelerómetro entre los mismos pasos de tiempo que el giróscopo, evaluando su similitud y comparando con los pasos anteriores cuán cerca del valor real están cada uno de ellos, se conseguiría la orientación de forma muy aproximada. Por supuesto, hay que tener en cuenta que cada paso será distinto del anterior, y la fiabilidad tanto del acelerómetro como del giróscopo variará.

Para realizar este algoritmo es necesario tener en cuenta que las mediciones del acelerómetro serán utilizadas como nuestra medida principal de observación. Mediante un análisis de Lyapunov[11] aseguramos la estabilidad y la corrección del error de observación (error cometido por el acelerómetro) reescribiendo la orientación mediante las mediciones de las velocidades angulares (giróscopo). Dicho análisis consta de lo siguiente (en adelante, una pequeña explicación por partes):

Inicialmente el código lee los datos del IMU de un instante dado (giroscopio y Acelerómetro). La variable obj es un handle que se realimenta a sí mismo (así se tienen en cuenta los datos anteriores).

```
function obj = UpdateIMU(obj, Gyroscope, Accelerometer)
```

Se procede a normalizar los datos del acelerómetro:

```
Accelerometer = Accelerometer / norm(Accelerometer);
```

[11] Ver http://es.wikipedia.org/wiki/Funci%C3%B3n_de_Lyapunov, y también http://es.wikipedia.org/wiki/Estabilidad_de_Lyapunov

Calculamos el error entre la estimación y la medición de la dirección de la gravedad, siendo éste el resultado del producto vectorial entre la medición del acelerómetro y la del cuaternion en el paso anterior:

```
v = [2*(obj.q(2)*obj.q(4) - obj.q(1)*obj.q(3))
    2*(obj.q(1)*obj.q(2) + obj.q(3)*obj.q(4))
    obj.q(1)^2 - obj.q(2)^2 - obj.q(3)^2 + obj.q(4)^2];

error = cross(v, Accelerometer');
```

Dicho error se suma al error de integración acumulado:

```
obj.IntError = obj.IntError + error;
```

Restamos a los datos medidos por el giróscopo el error medido en este paso y el que se obtuviera en la medición anterior. Pero no se restan en la misma medida, Kp y Ki son las constantes que amplían y/o reducen el error de este paso y del paso anterior en función de la exactitud de la medición actual:

```
Ref = Gyroscope - (obj.Kp*error + obj.Ki*obj.IntError)';
```

Calculamos el cambio en el cuaternión del paso anterior a este, integrándolo, normalizándolo y conjugándolo para obtener en forma de cuaternión la posición final del IMU:

```
pDot = 0.5 * obj.quaternProd(obj.q, [0 Ref(1) Ref(2)
Ref(3)]);
obj.q = obj.q + pDot * obj.SamplePeriod;

obj.q = obj.q / norm(obj.q);

obj.Quaternion = obj.quaternConj(obj.q);

end
```

Programé por mi cuenta este código que procesa los datos del IMU para calcular la orientación, y aunque funcionaba bien era especialmente lento, por lo que finalmente

encontré en internet uno ya hecho (opensource)[12] bastante aceptable que adapté para que formara parte de mi código. Los fragmentos de código anteriores están tomados de dicha modificación.

DETECCIÓN DE LAS ACELERACIONES POR DEBAJO DE UN UMBRAL

Antes de aplicar la transformación de giro de las aceleraciones, vamos a buscar todas aquellas que estén por debajo de la suma de la media y la desviación típica de la norma de los valores del acelerómetro en reposo, con la intención de que todos los datos de la velocidad que coincidan con los instantes umbral se impongan nulos y reducir así los posibles errores de integración:

```
%% DETECTAR ACELERACIONES NULAS

% Busca las aceleraciones cercanas al cero (dentro del umbral de
% aceleración nula), y crea una variable lógica que avisa de si el muestreo
% está o no en la zona del umbral

% Cálculo del módulo de la aceleración
ac_mod = sqrt(ax.*ax + ay.*ay + az.*az);

% Filtro Paso Alto a la aceleración
frecCorte = 0.001;
[b, a] = butter(1, (2*frecCorte)/(1/dt), 'high');
ac_modFilt = filtfilt(b, a, ac_mod);

% Cálculo del valor absoluto de la aceleración filtrada
ac_modFilt = abs(ac_modFilt);

% Filtro Paso Bajo a la aceleración
frecCorte = 5;
[b, a] = butter(1, (2*frecCorte)/(1/dt), 'low');
ac_modFilt = filtfilt(b, a, ac_modFilt);

% Detección de los datos por debajo del umbral
ac_umbral = mean(ac_modFilt) + std(ac_modFilt);
```

[12] http://www.x-io.co.uk/open-source-imu-and-ahrs-algorithms/

REDUCCIÓN DE ERRORES EN LA LECTURA DE DATOS

En muchas ocasiones se da el caso de que un dato concreto en la lectura de los datos se sale de forma exagerada de las mediciones normales. Este tipo de datos, por muy aislados que fueran, causaban errores constantes al código, por lo que me vi obligado a eliminar dichos valores igualándolos al siguiente para reducir considerablemente la cantidad de errores que daba el programa:

```
%Eliminar errores en giroscopio:

for i=1:n-1
if((abs(gx(i+1)-gx(i))>500) || isnan(gx(i)))
gx(i+1)=gx(i);
end
if((abs(gy(i+1)-gy(i))>500) || isnan(gz(i)))
gy(i+1)=gy(i);
end
if((abs(gz(i+1)-gz(i))>500) || isnan(gz(i)))
gz(i+1)=gz(i);
end
end


%Eliminar errores en acelerómetro:

for i=1:n-1
if((abs(ax(i+1)-ax(i))>20) || isnan(ax(i)))
ax(i+1)=ax(i);
end
if((abs(ay(i+1)-ay(i))>20) || isnan(ay(i)))
ay(i+1)=ay(i);
end
if((abs(az(i+1)-az(i))>20) || isnan(az(i)))
az(i+1)=az(i);
end
if(norm([ax(i) ay(i) az(i)])==0)
ax(i)=ax(i+1);
ay(i)=ay(i+1);
az(i)=az(i+1);
end
end
```

TRANSFORMACIÓN DE LA ACELERACIÓN A EJES ABSOLUTOS

Ahora sí, se aplica el código del AHRS a las aceleraciones para transformarla a ejes absolutos. Se puede observar que en los momentos umbral se considera la ganancia de proporción definida anteriormente como nula. Eso no significa que la aceleración lo sea. Finalmente se convierten todas las aceleraciones a m/s^2, y se sustrae la gravedad de la componente z, ahora que ya está en ejes absolutos:

```
%% CALCULAR ORIENTACIÓN

quat = zeros(length(tiempo), 4);
AHRSalgorithm = AHRS('SamplePeriod', 1/256, 'Kp', 1, 'KpInit', 1);

% Orientación de todos los datos
for t = 1:length(tiempo)
if(umbral(t))
        AHRSalgorithm.Kp = 0.5;
else
        AHRSalgorithm.Kp = 0;
end
AHRSalgorithm.UpdateIMU(deg2rad([gx(t) gy(t) gz(t)]), [ax(t) ay(t)...az(t)]);
quat(t,:) = AHRSalgorithm.Quaternion;
end

%% CALCULAR ACELERACIONES DE TRASLACIÓN EN EJES ABSOLUTOS

% Aplicar rotación por cuaterniones
acel = quaternRotate([ax ay az], quaternConj(quat));

% Convertir a m/s²
acel = acel * 9.81;

% Sustraer la gravedad de la aceleración en z
acel(:,3) = acel(:,3) - 9.81;
```

INTEGRACIÓN DE LA ACELERACIÓN PARA OBTENER LA VELOCIDAD

En esta parte del código no sólo se integra la aceleración para obtener la velocidad, también se imponen como nulas las velocidades en los valores por debajo del umbral establecido en la aceleración, así como procesar los errores de integraciónen los tramos de las velocidades que sí están por encima de los valores establecidos como umbral:

```
% Integrar la aceleración para obtener la velocidad. Iguala a cero los
% valores situados por debajo del umbral de aceleración nula.

vel = zeros(size(acel));
for t = 2:length(vel)
vel(t,:) = vel(t-1,:) + acel(t,:) * dt;
if(umbral(t) == 1)
vel(t,:) = [0 0 0];
end
end

% Calcular el error de integración de las velocidades tras el umbral

errorVel = zeros(size(vel));
inicioUmbral = find([0; diff(umbral)] == -1);
finUmbral = find([0; diff(umbral)] == 1);

for i = 1:numel(finUmbral)
errorEstim = vel(finUmbral(i)-1, :) /...
        (finUmbral(i) - inicioUmbral(i));
enum = 1:(finUmbral(i) - inicioUmbral(i));
error = [enum'*errorEstim(1) enum'*errorEstim(2) enum'*errorEstim(3)];
errorVel(inicioUmbral(i):finUmbral(i)-1, :) = error;
end


% Eliminar el error de integración
vel = vel - errorVel;
```

INTEGRACIÓN DE LA VELOCIDAD PARA OBTENER LA POSICIÓN

Al integrar la velocidad para obtener la posición, nos encontramos con que el error de integración acumulado es tan grande que a los dos segundos la posición se dispara muy lejos de los valores reales. La solución para esto fue emplear un filtro de paso bajo. La determinación de la frecuencia de corte de dicho filtro fue experimental.

```
% Integrar la velocidad para obtener la posición
pos = zeros(size(vel));
for t = 2:length(pos)
pos(t,:) = pos(t-1,:) + vel(t,:) * dt;
end

% Aplica un filtro de paso alto para eliminar el error. Ésto hace que
% siempre que se quede quieto se desplace lentamente hacia el origen.
```

```
% El filtro de paso alto se aplica con mayor frecuencia de corte en la
% dirección vertical que en las horizontales, ya que el error de
% integración es distinto en ambas direcciones:

orden = 1;
frecCorte = 0.1;
[b, a] = butter(orden, (2*frecCorte)/(1/dt), 'high');
posHP(:,1:2) = filtfilt(b, a, pos(:,1:2));

orden = 1;
frecCorte = 0.5;
[b, a] = butter(orden, (2*frecCorte)/(1/dt), 'high');
posHP(:,3) = filtfilt(b, a, pos(:,3));
```

REPRESENTACIÓN GRÁFICA DEL MOVIMIENTO REGISTRADO Y PROCESADO

Para la representación en 3D del movimiento del IMU fue necesario adaptar otro código abierto obtenido en Internet. En concreto, la función SixDOFanimation[13]. Hay adjuntas imágenes más adelante, en el apartado de mediciones.

DIFERENCIAS ENTRE CALIBRADO Y LECTURA

He implementado en la interfaz de usuario dos botones distintos para la obtención de datos: el primero es el de "Calibrar", que puede durar entre 2 y 8 segundos (a elección del usuario), y que evalúa los datos del IMU para obtener media y desviación típica de los valores de las aceleraciones y los giroscopios, así como ganancias y offsets iniciales. Es obligatorio que, durante el calibrado, el IMU esté situado en reposo sobre un plano horizontal.

El segundo botón es el de "Medición", que mide los datos (partiendo del reposo) del IMU situado en el generador de olas. Para ello, he impuesto los límites de selección del tiempo de medición entre 20 y 180 segundos.

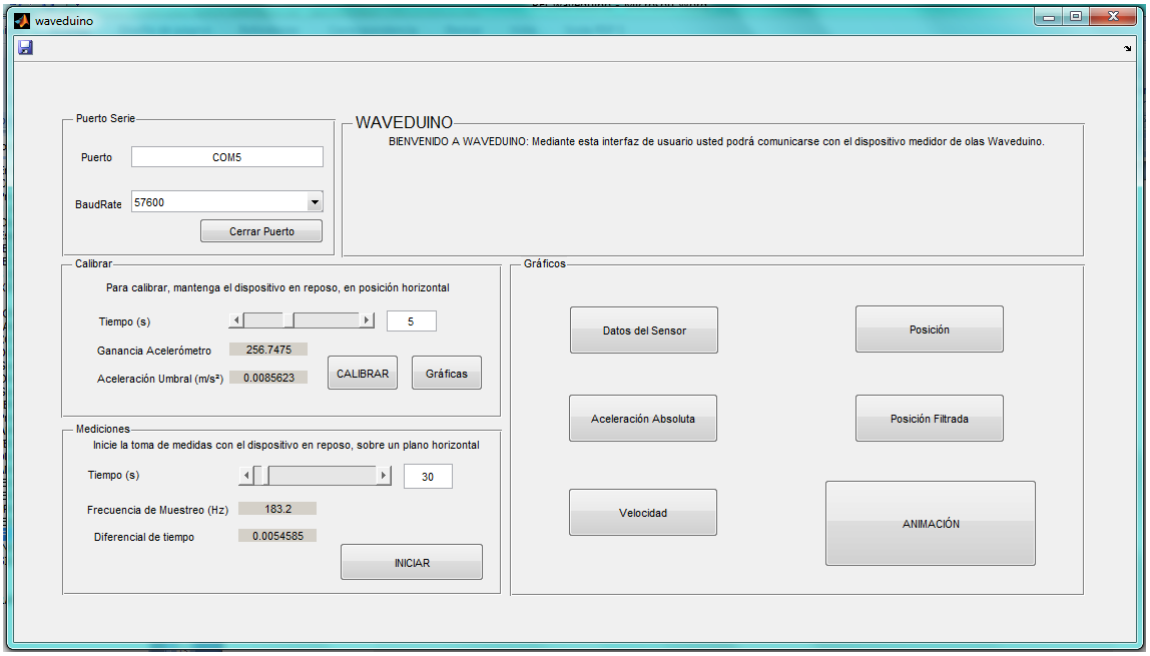
INTERFAZ DE USUARIO

De la interfaz de usuario no hay demasiado que explicar, aprendí a usar el GUIDE (Graphical User Interface Development Environment) de MATLAB y creé una pequeña

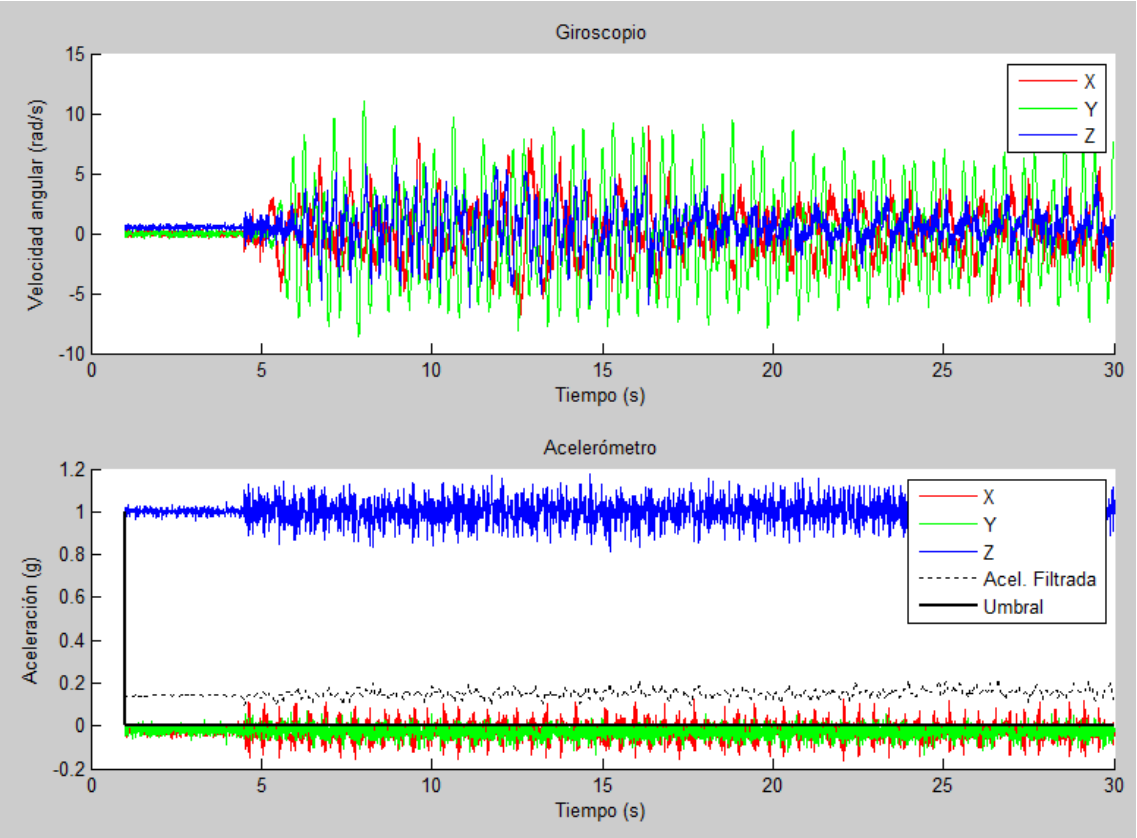
[13] https://github.com/xioTechnologies/Oscillatory-Motion-Tracking-With-x-IMU/blob/master/SixDOFanimation.m

ventana con botones y slides para seleccionar los parámetros de calibrado y de lectura de datos:

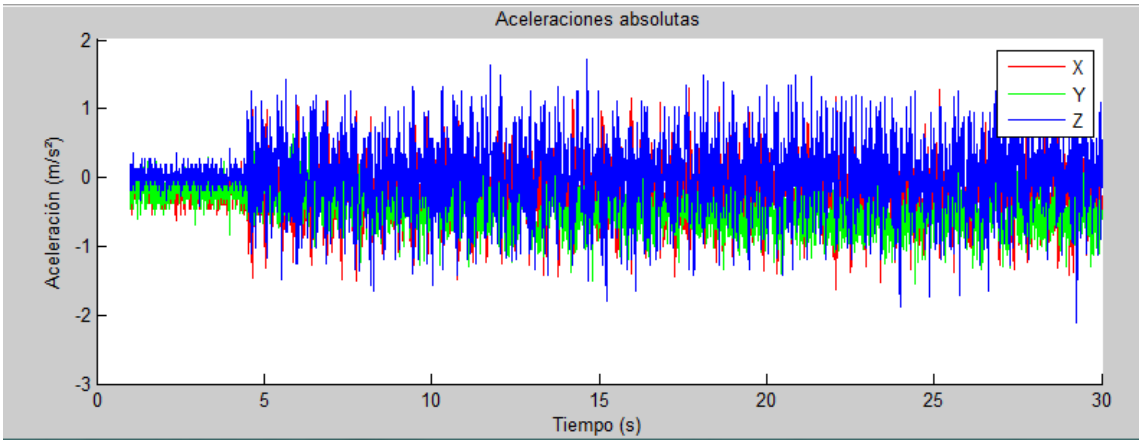
Ya que el código de la interfaz incluye en su totalidad el código empleado en el desarrollo del programa en MATLAB de este proyecto con explicaciones y comentarios, éste se incluirá en el cd de datos ajunto a este proyecto.



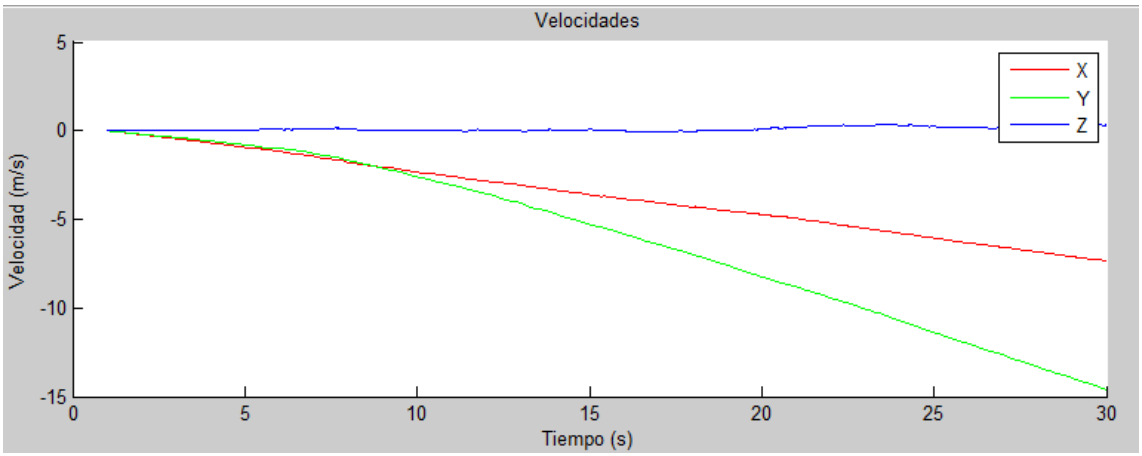
INTERFAZ GRÁFICA DE USUARIO



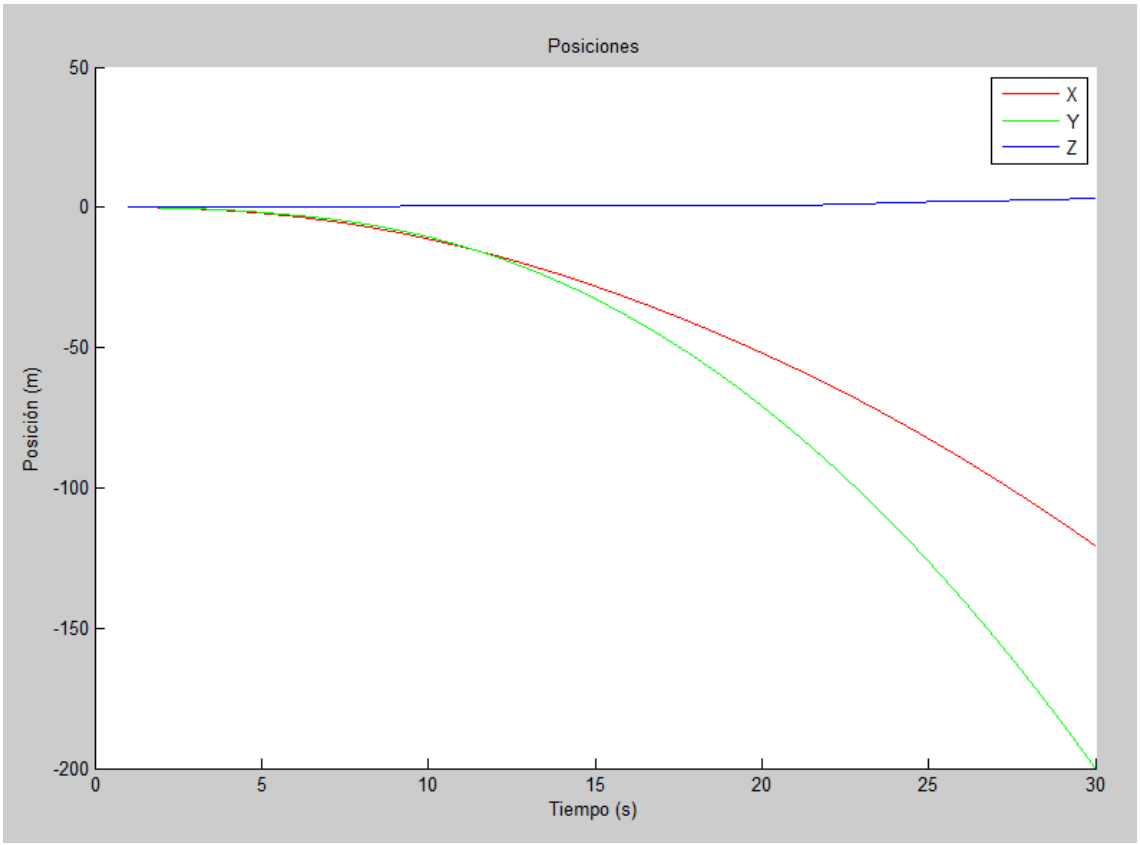
DATOS DEL SENSOR



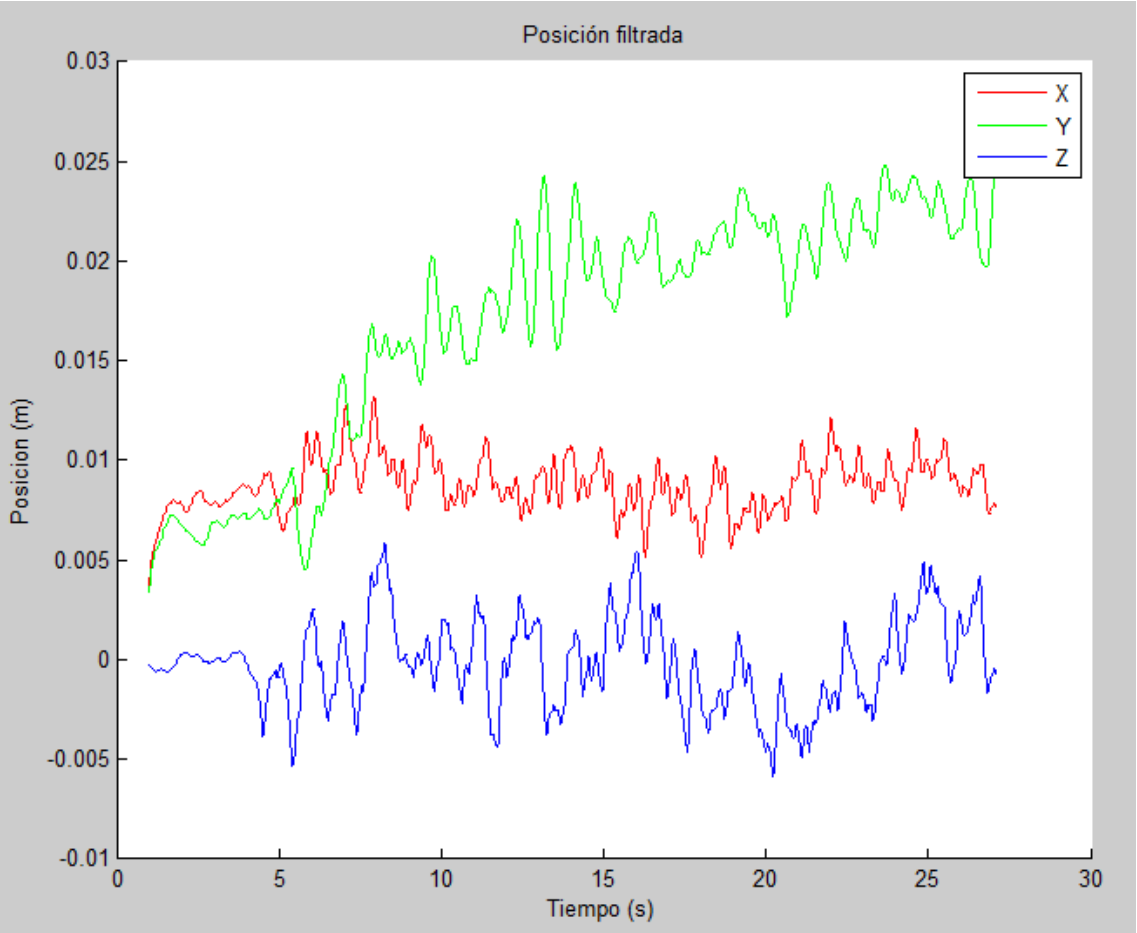
ACELERACIONES ABSOLUTAS



VELOCIDADES



POSICIONES



POSICIONES FILTRADAS: SE PUEDE OBSERVAR QUE EL AVANCE SE REALIZA EN DIRECCIÓN Y.

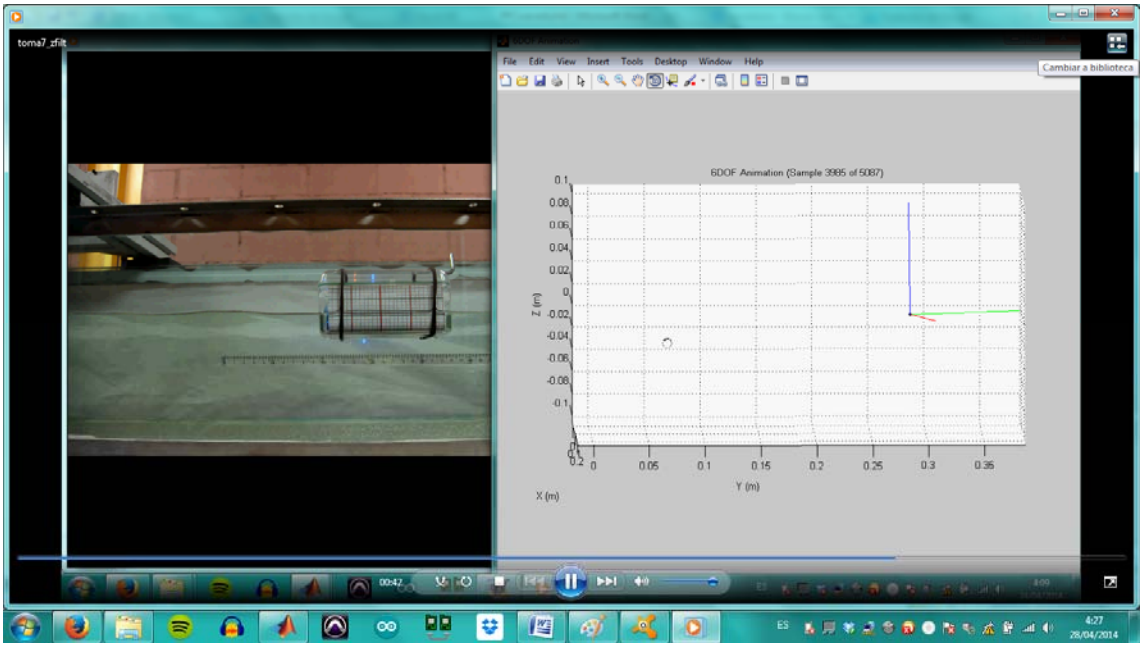
CARGA DE DATOS EN EL PROGRAMA

El programa suele cargar y guardar datos en dos archivos diferentes que se pueden ver en la carpeta del programa. Uno de ellos es “calibrado.mat”, y no debe modificarse ya que registra los datos del último calibrado. El otro es “medicion.mat”, y es el archivo que contiene los datos de la última medición. Si se quiere cargar un archivo de datos tomado ya con anterioridad, debe sustituirse por “medición.mat” bajo el mismo nombre. La carpeta “Datos” contiene algunas de las mediciones realizadas en el laboratorio.

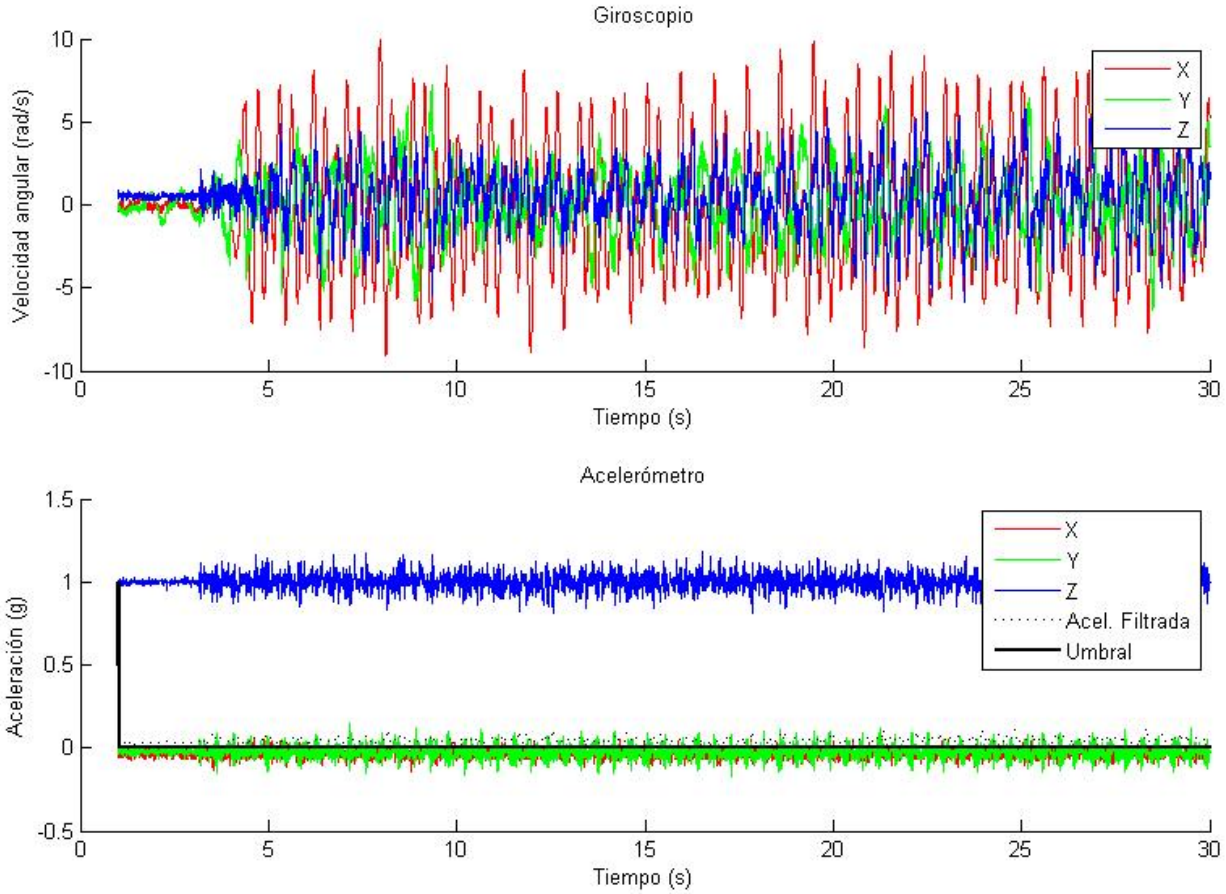
MEDICIONES EN EL GENERADOR DE OLAS

He de comenzar explicando que, debido al uso continuado en el tiempo del prototipo de mi proyecto, para cuando ha estado terminado todo el proyecto, realizar las mediciones finales ha sido un calvario: debido al continuado uso del dispositivo para programarlo y reprogramarlo, conectarlo y desconectarlo una y otra vez para las distintas modificaciones sufridas, las conexiones de éste han quedado bastante dañadas por lo que con movimientos no demasiado bruscos el contacto se perdía y con él el envío de datos al PC. Es por ello que sólo adjunto fotos y videos de las dos mediciones realizadas con mayor exactitud.

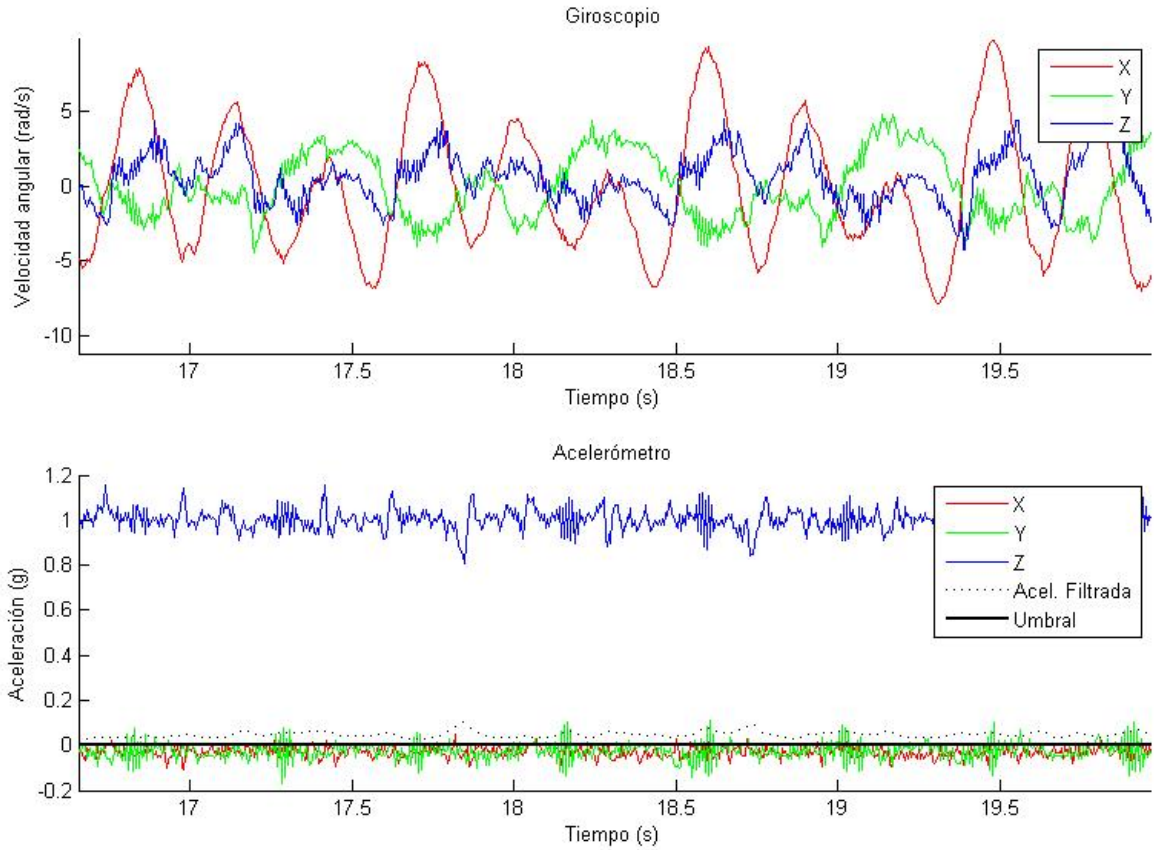
PRIMERA MEDICIÓN



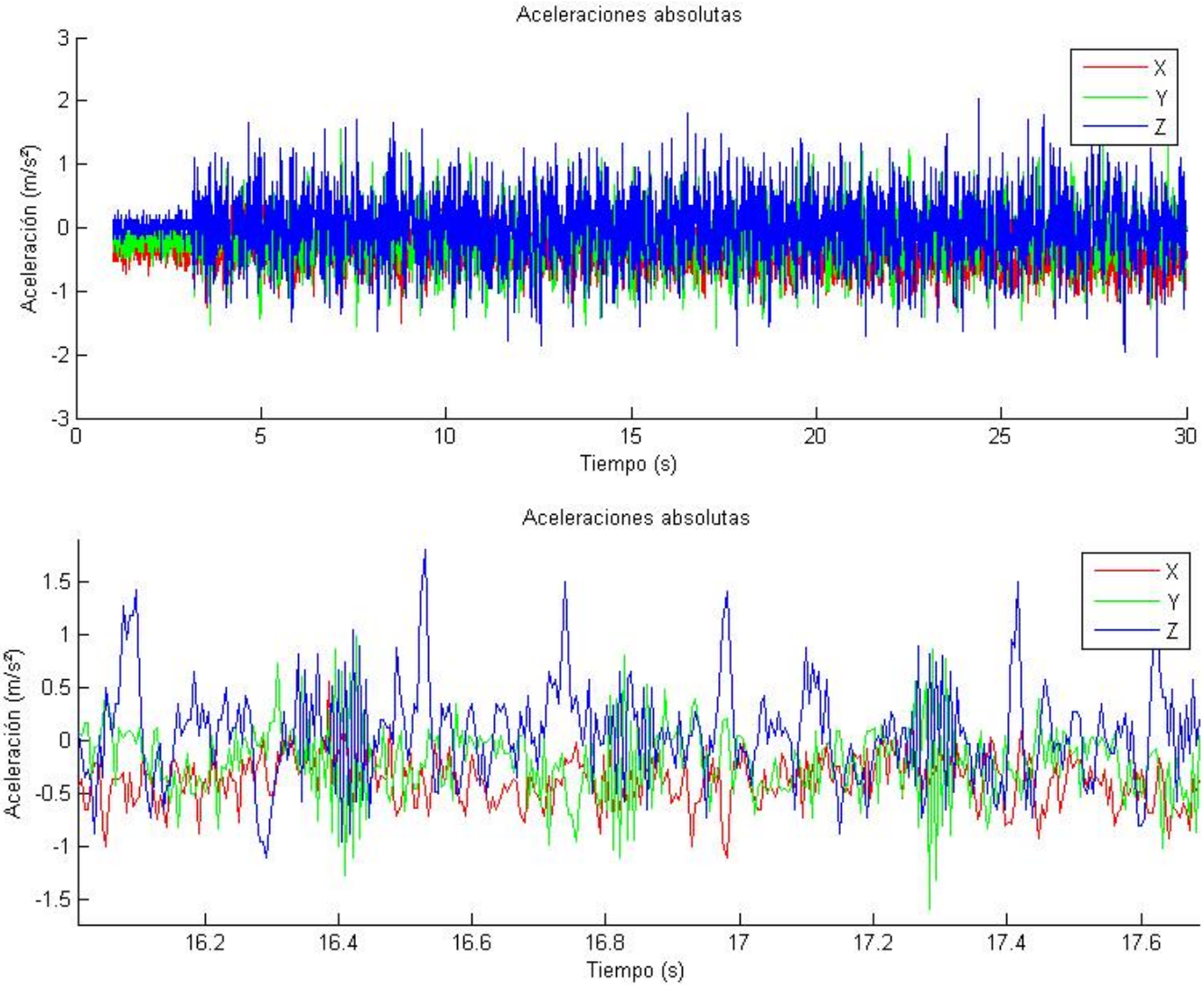
En el video se observa un desplazamiento inicial que se debe al filtro de paso alto empleado en la eliminación del error de integración de la velocidad para la obtención de la posición. Veamos ahora los gráficos obtenidos:



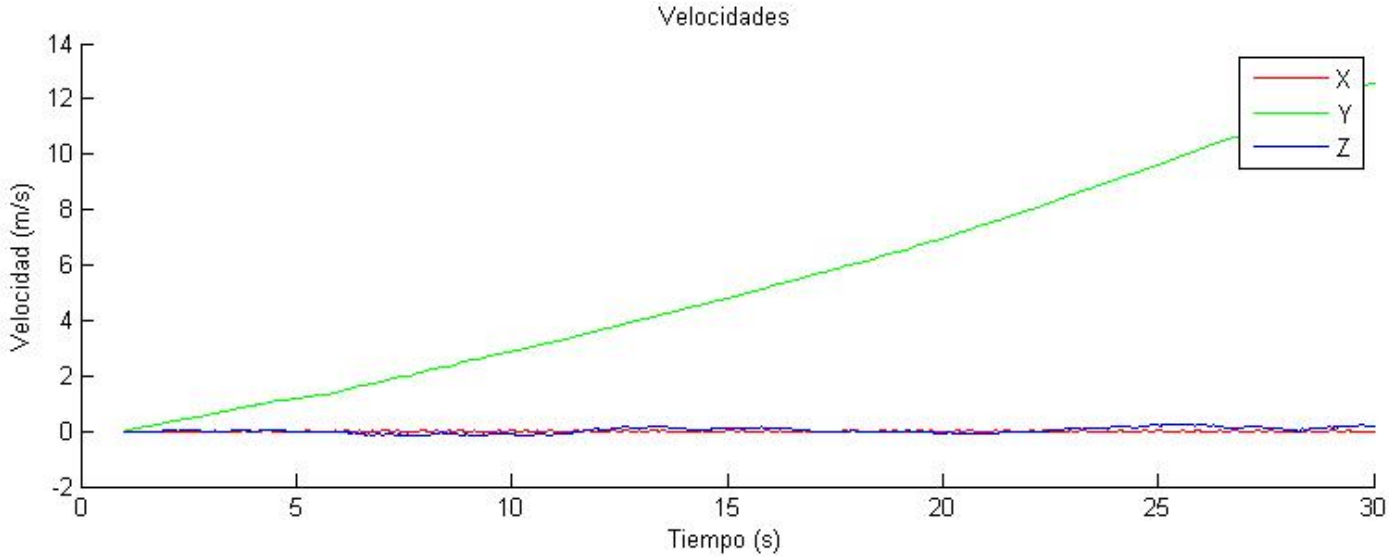
DATOS RECIBIDOS DEL SENSOR, CONVERTIDOS A RAD/S Y G MEDIANTE OFFSETS Y GANANCIAS.

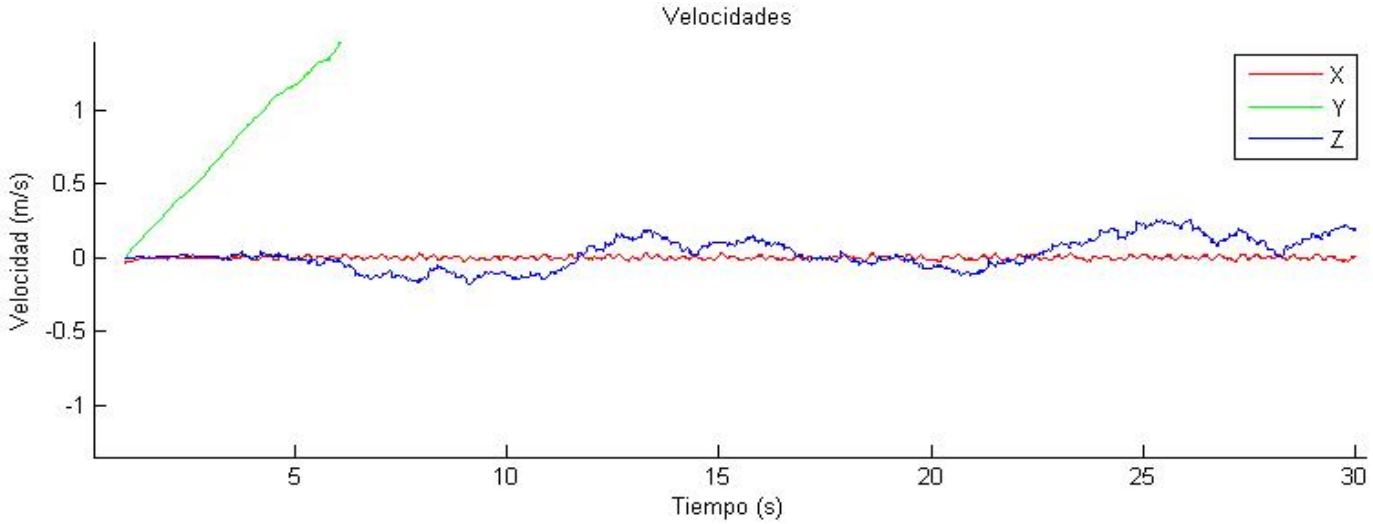


SI AMPLIAMOS LA IMAGEN, SE PUEDE APRECIAR QUE EN EL GIROSCOPIO, EL GIRO SOBRE EL EJE X ES MAYOR Y MÁS NÍTIDO QUE EN EL RESTO, MIENTRAS QUE EN EL ACELERÓMETRO LA ACELERACIÓN EN Z ES MAYOR QUE EN Y, Y ESTA MAYOR QUE EN X. ESTO SE DEBE A QUE EL GIRO DEL DISPOSITIVO ES MAYORITARIAMENTE EN X, LAS SUBIDAS Y BAJADAS DEL DISPOSITIVO SON EN Z Y EL AVANCE EN Y.

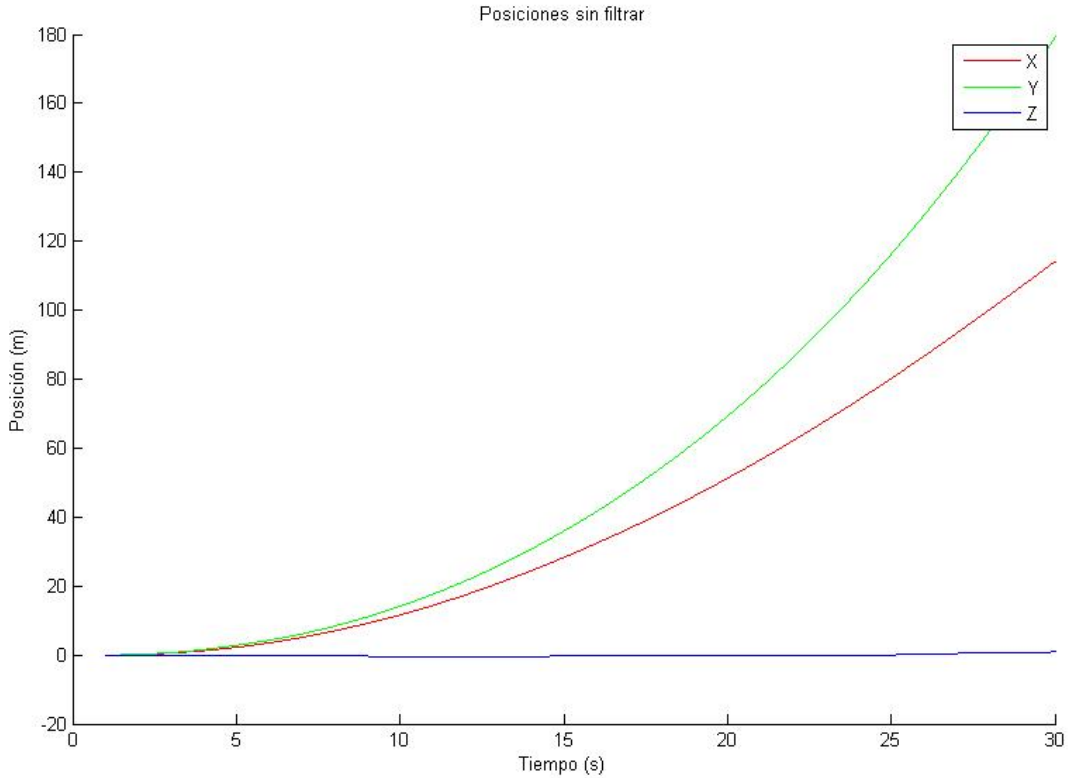


EN ESTE CASO VEMOS LAS ACELERACIONES REFERIDAS AL SISTEMA DE REFERENCIA FIJO, Y HABIENDO EXTRAÍDO LA GRAVEDAD. SE APRECIA QUE LAS ACELERACIONES EN Z SON MAYORES QUE EN X E Y YA QUE EL DISPOSITIVO SUBE Y BAJA CON EL VAIVÉN DE LAS OLAS CON MAYOR ACELERACIÓN QUE EL AVANCE DEL DISPOSITIVO.

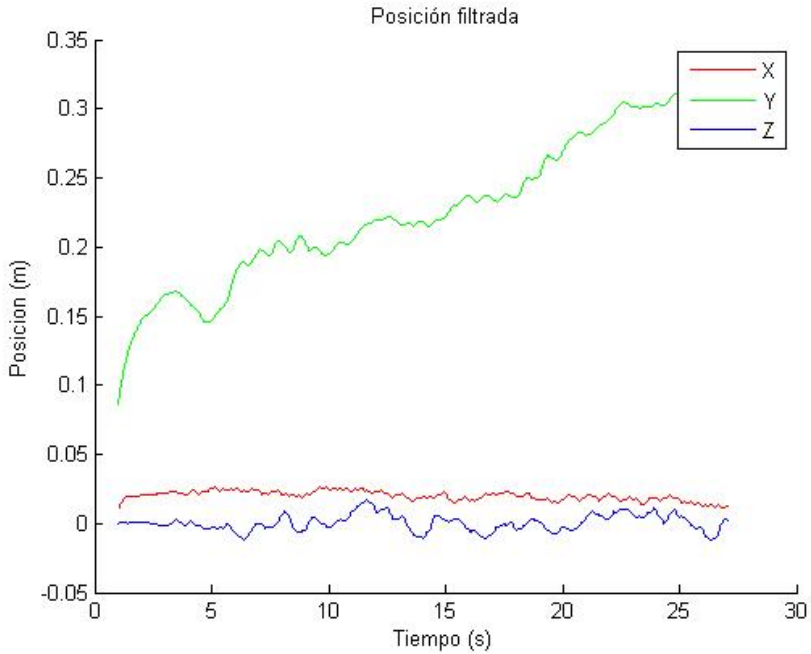


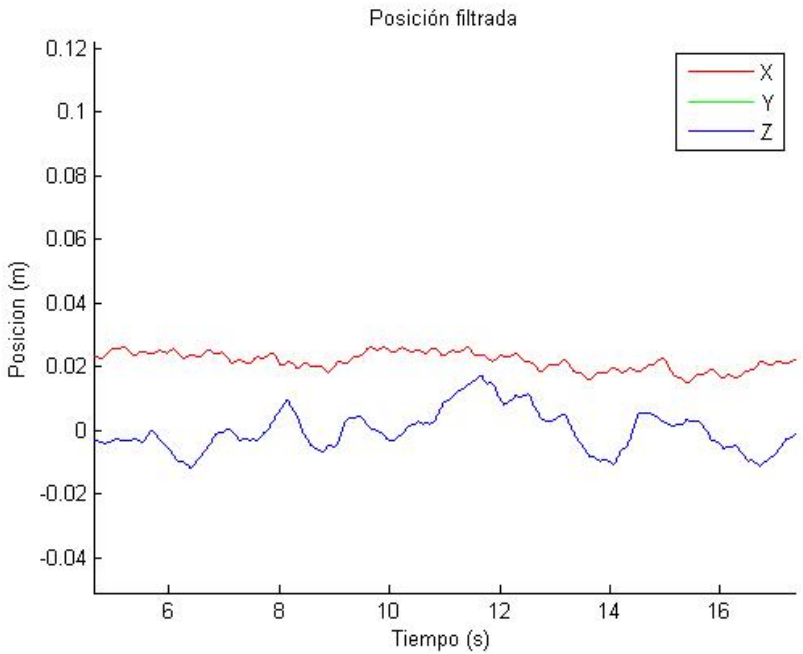


EN LAS VELOCIDADES PODEMOS VER QUE EL AVANCE EN Y ES MUCHO MAYOR. SIN EMBARGO NO TANTO COMO PARA ALCANZAR OBVIAMENTE LOS 13 M/S. ESTO SE DEBE AL ERROR DE INTEGRACIÓN, QUE NO SERÁ CORREGIDO HASTA QUE SEA TRATADO EN EL FILTRO DE LA POSICIÓN.

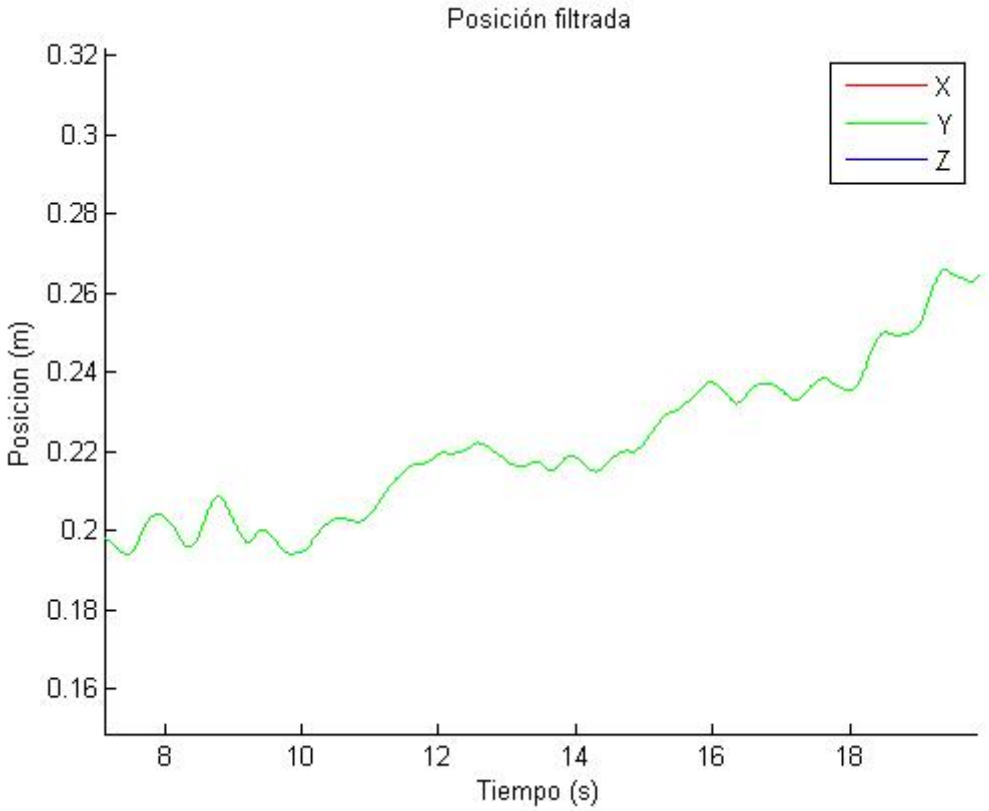


EN LA POSICIÓN SIN FILTRAR SE PUEDE APRECIAR QUE LA VELOCIDAD SE DISPARA POR EL ERROR DE INTEGRACIÓN TANTO EN X COMO EN Y.



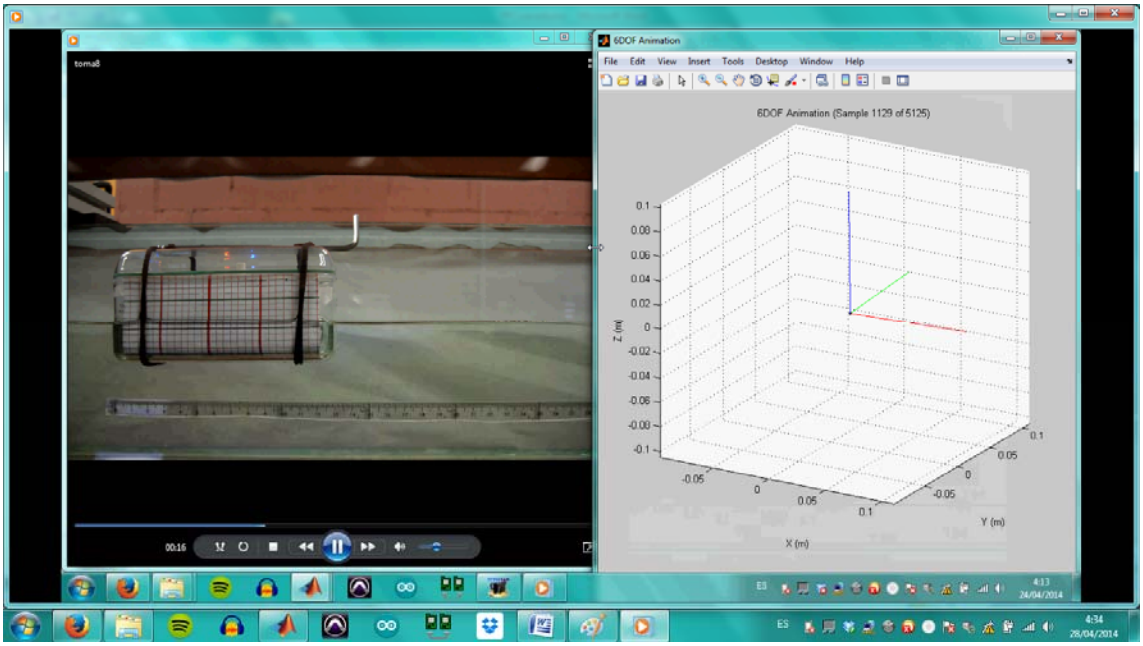


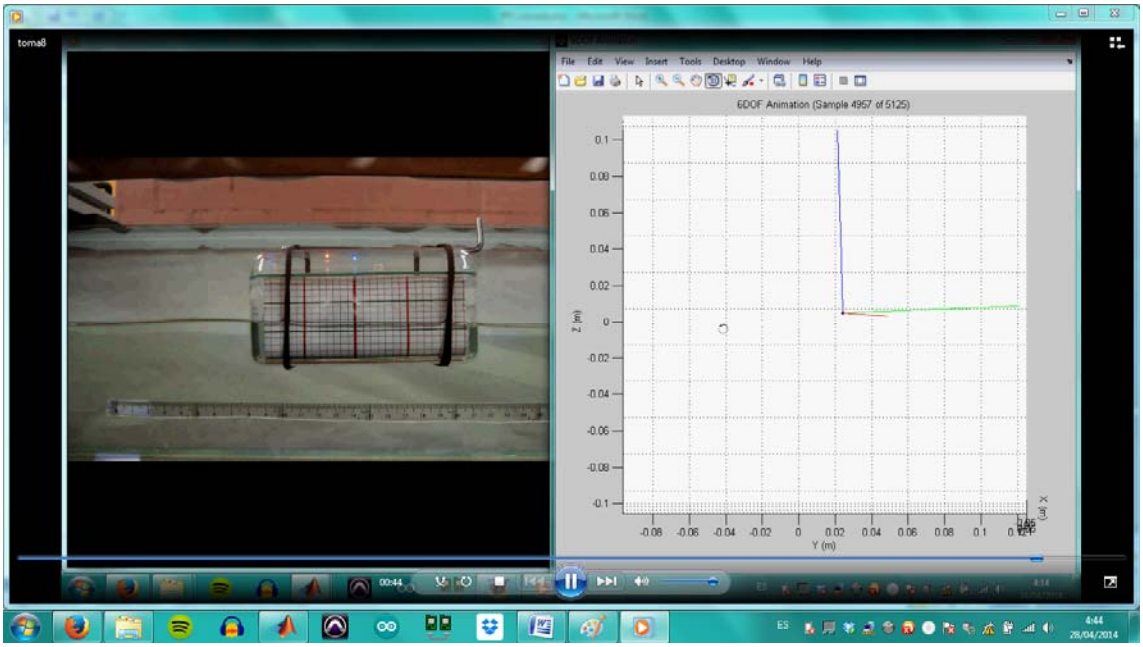
SE PUEDE APRECIAR CÓMO EL AVANCE SE REALIZA EN LA DIRECCIÓN Y, EL VAIVÉN DE LAS OLAS SE VE REFLEJADO EN Z Y LA X SE MANTIENE PRÁCTICAMENTE CONSTANTE (AUNQUE ALGO DESPLAZADA RESPECTO DEL ORIGEN, DEBIDO AL DE PASO ALTO).



EN LA DIRECCIÓN Y SE APRECIA EL AVANCE.

SEGUNDA MEDICIÓN



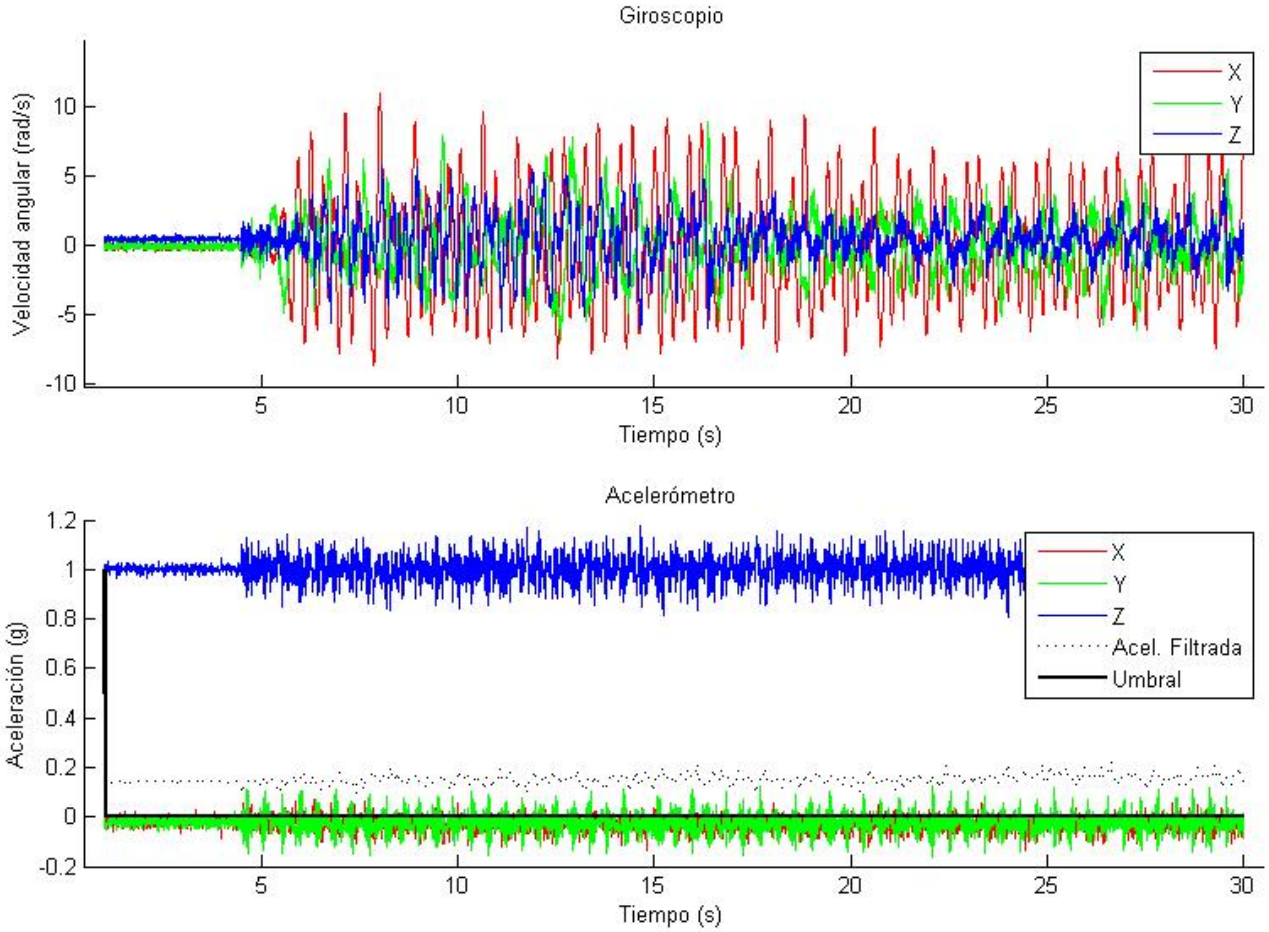


Se puede apreciar en las mediciones que la orientación es prácticamente idéntica a la real, y que la traslación se aproxima bastante, aunque no se ajusta tan bien como la orientación. Esto es debido a dos motivos:

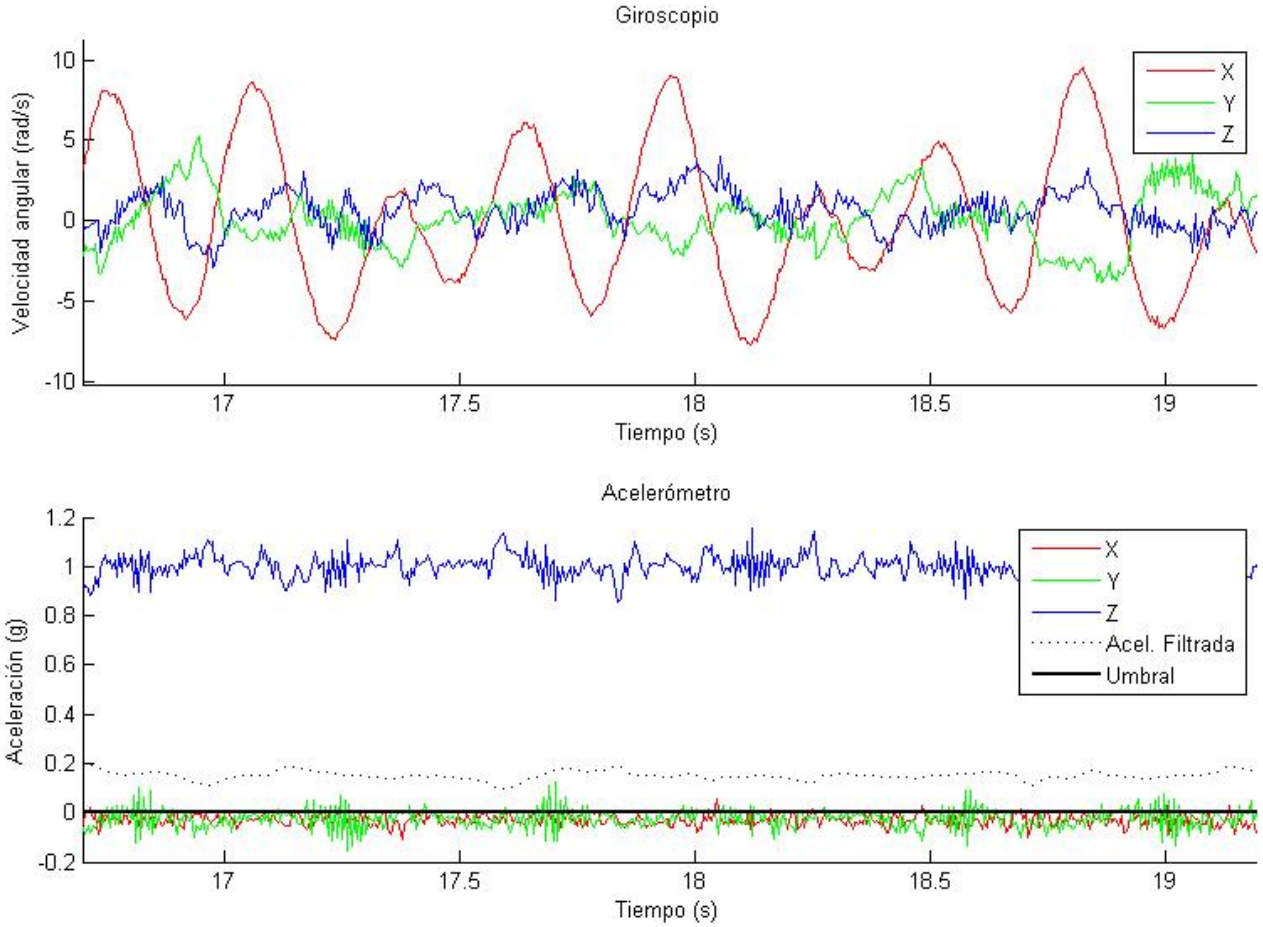
-El angular de la cámara con la que se han realizado las medidas hace que la traslación parezca mucho mayor de lo que realmente es. Y he de añadir que lo hice alejando la cámara suficiente como para que se vieran las divisiones de la regla y sin emplear zoom.

-El ya mencionado filtro de paso bajo aplicado a la posición para eliminar el error de integración realiza un desplazamiento inicial del dispositivo que implica que no parta del origen.

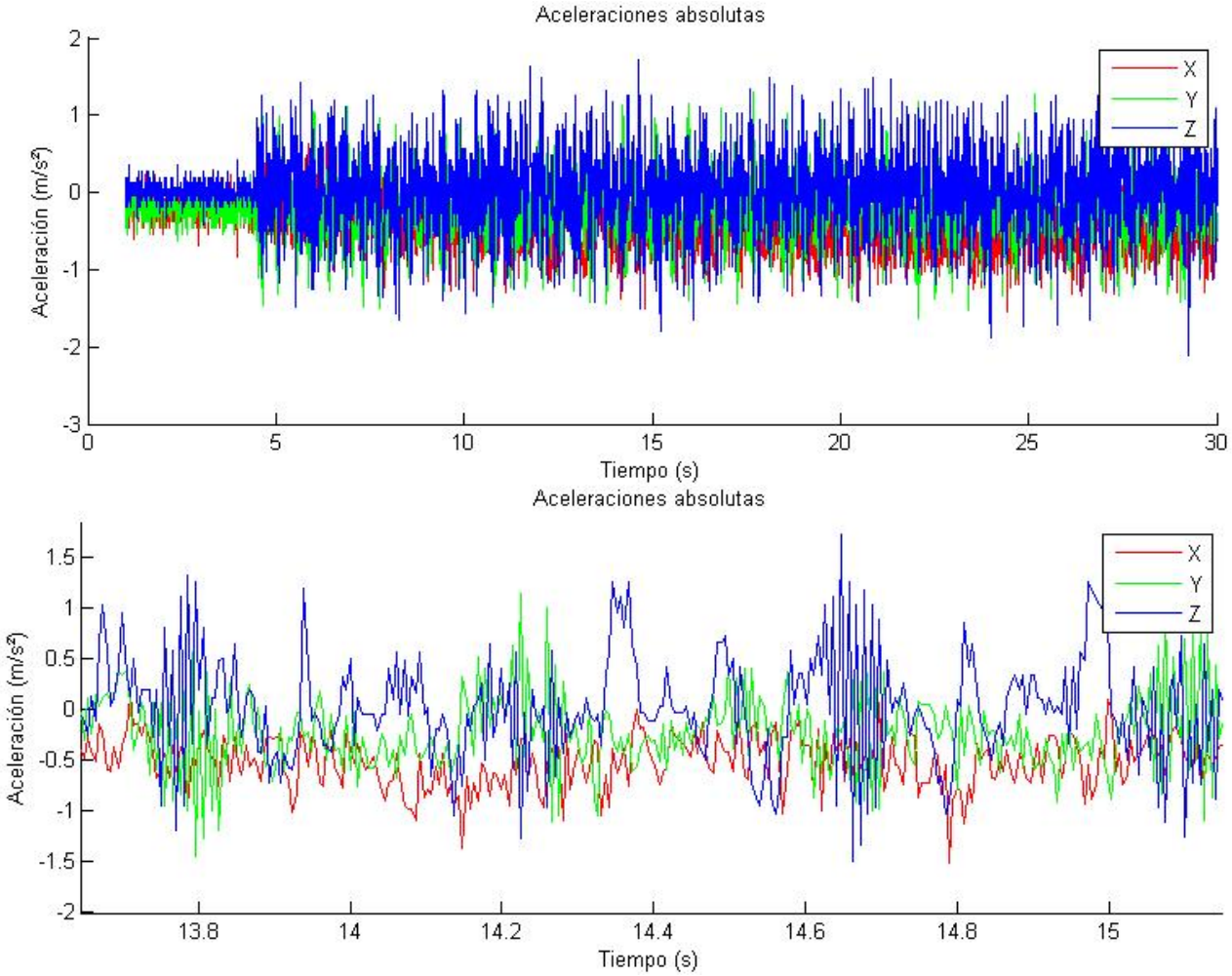
Veamos en esta toma de datos las diferentes gráficas de aceleración, velocidad y posición:



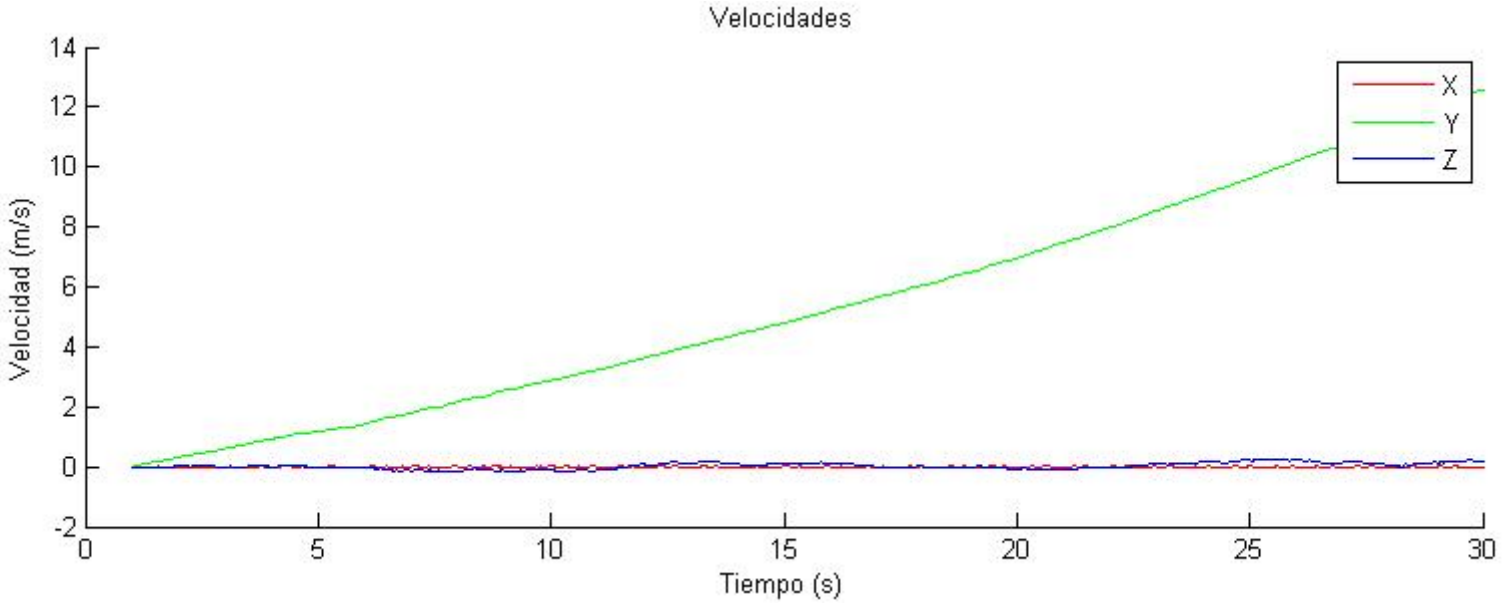
DATOS RECIBIDOS DEL SENSOR, CONVERTIDOS A RAD/S Y G MEDIANTE OFFSETS Y GANANCIAS.



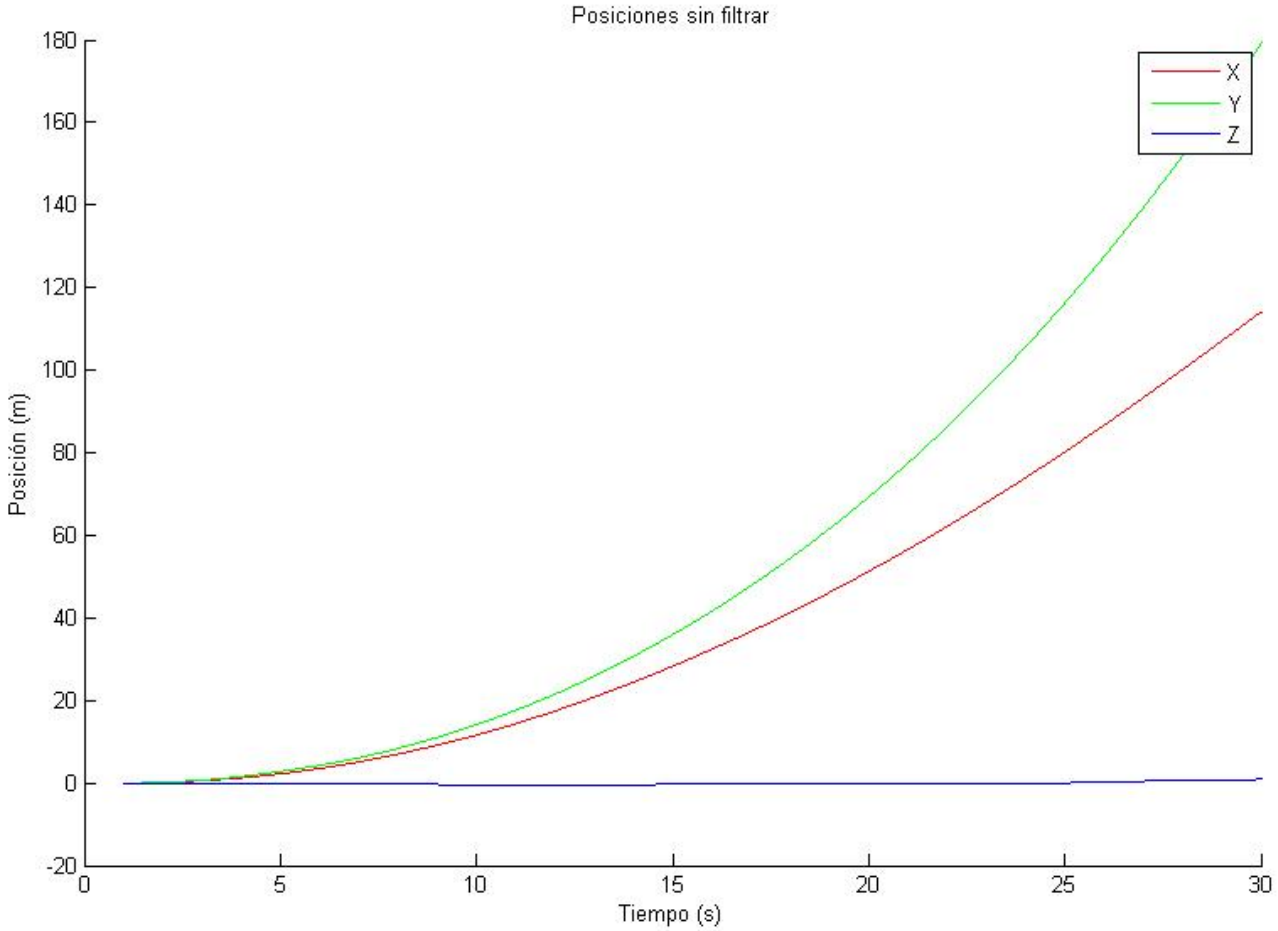
AMPLIACIÓN DE LOS DATOS RECIBIDOS DEL SENSOR, CONVERTIDOS A RAD/S Y G MEDIANTE OFFSETS Y GANANCIAS.



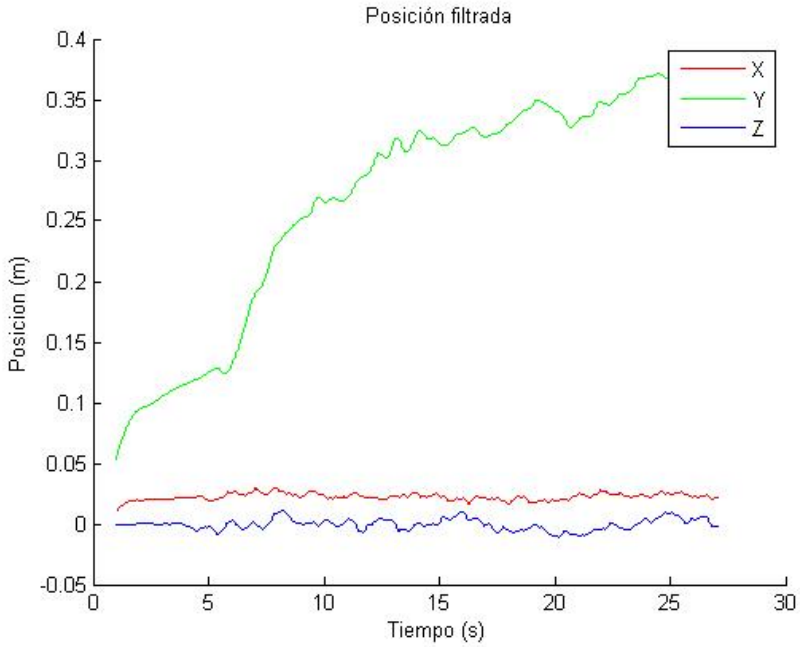
ACELERACIONES REFERIDAS AL SISTEMA FIJO, SIN LA ACELERACIÓN DE LA GRAVEDAD.

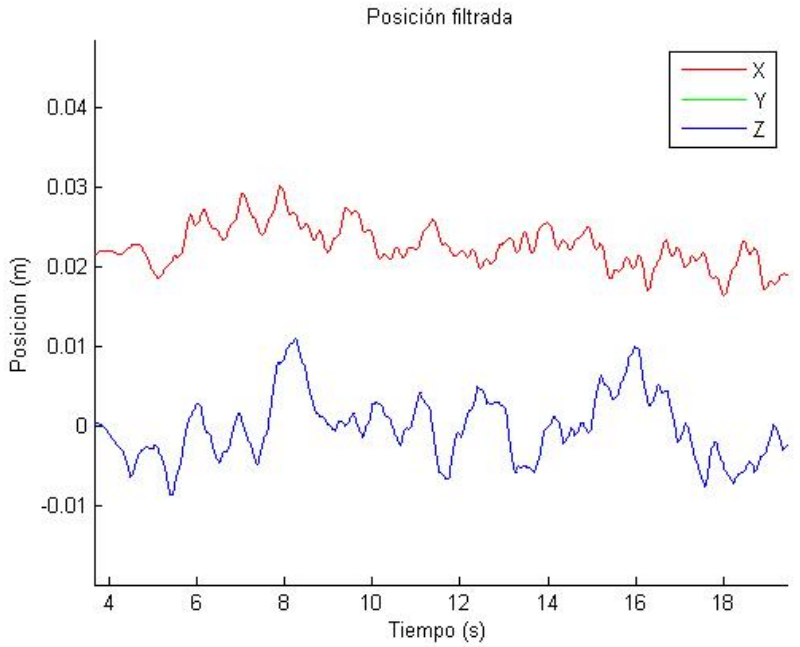


REPRESENTACIÓN DE LAS VELOCIDADES EN X, Y, Z.

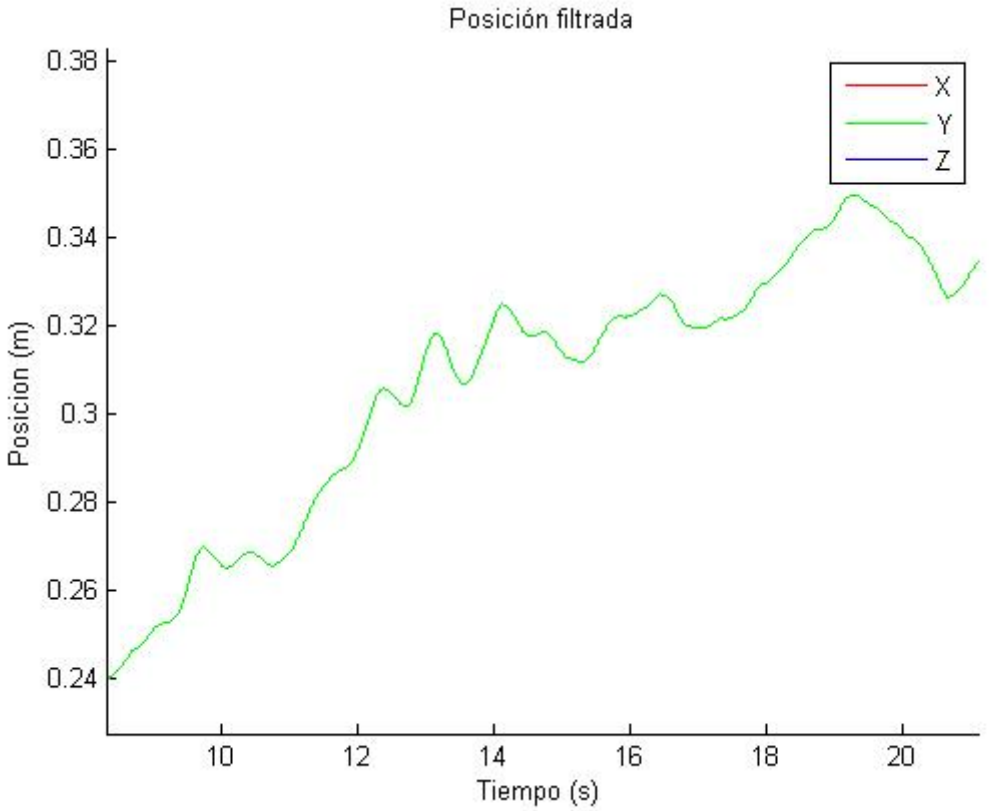


REPRESENTACIÓN DE LAS VELOCIDADES EN X, Y, Z.





REPRESENTACIÓN GRÁFICA DE LAS POSICIONES YA FILTRADAS.



UNA VEZ MÁS, SE PUEDE APRECIAR QUE EL AVANCE SE REALIZA EN LA DIRECCIÓN Y.

PRESUPUESTO

El coste de este proyecto reside, como ya se ha dicho anteriormente, principalmente en el tiempo invertido, ya que la ventaja de trabajar con los sensores y dispositivos elegidos y el microcontrolador de Arduino es que son muy baratos.

Para calcular el coste de ingeniería he preguntado en varias empresas en España y el coste estándar de ingeniero oscila entre los 10 y los 50 €/h, en función de la empresa y de la experiencia del ingeniero. Es por ello que se ha establecido una media de 30 €/h. Ya que el proyecto comenzó en septiembre y acabó a finales de abril, consideramos 7 meses trabajados, 5 días semanales, 3 horas por día. En total son 420 horas trabajadas lo que supone un coste de ingeniería de 12600 €.

NOMBRE	**PRECIO (IVA INCLUIDO)**
Arduino Nano Rev 3.0	46.50 €
IMU Sensor Stick 9DOF	95.95 €
Módulo Xbee 802.15.4 Low Power Module (Emisor)	18.40 €
Módulo Xbee 802.15.4 Low Power Module (Receptor)	18.40 €
Xbee Add-on For Arduino Nano	22.5 €
Xbee USB Adapter	27 €
Pilas 9V, Interruptor, Carcasas y Material para la PCB	10€
Coste de Ingeniería	12600 €
COSTE TOTAL DEL PROYECTO	12838.75 € [14]

[14] Los precios de los distintos dispositivos se pueden ver en las dos páginas en las que fueron adquiridos: http://www.electronicaembajadores.com y http://www.mouser.com .

RESUMEN Y CONCLUSIONES

RESUMEN

Finalizamos este proyecto realizando un repaso de lo aprendido y realizado:

1) Arduino: tanto hardware libre como software libre (IDE) y lenguaje de programación (basado en C).

2) Unidades de medición inercial (IMU), distintos dispositivos de hardware para Arduino (shields, módulos wifi…) y su programación.

3) Código AHRS para orientación de dispositivos basado en filtros de estimación y corrección de errores.

4) Códigos de integración de aceleración y velocidad para obtener traslación, así como de filtros de paso alto para eliminar errores de integración.

5) Uso del editor de interfaces gráficas de usuario de MATLAB (GUIDE) para el desarrollo de un programa con interfaz gráfica.

CONCLUSIONES

Tras finalizar el proyecto, se pueden realizar una serie de observaciones relativamente importantes sobre el mismo:

- Aprender sobre microcontroladores, Arduino, programar en C y en MATLAB, o incluso el empleo del GUIDE de MATLAB son tareas que bien pueden estar al alcance de cualquier persona con unos conocimientos mínimos de hardware y de programación. Sin embargo, las unidades de medición inercial, su lectura y procesado de datos puede resultar de una complejidad mucho mayor.

- El seguimiento de un dispositivo mediante sistemas de medición inercial (IMU) y de referencia de actitud y rumbo (AHRS) conlleva muchas horas de comprensión y de programación. Además, la principal observación sobre estos sistemas es que, a la hora de obtener la posición del dispositivo basada en la integración doble de la aceleración se induce en ésta automáticamente dos

errores distintos difíciles de separar: el error cometido por el dispositivo y el error de integración. El primero depende de la magnitud de las aceleraciones y giros que el dispositivo experimenta, así como de la temperatura y otros factores menos importantes. El segundo depende del tiempo ya que el error de integración acumulado se incrementa en cada paso de integración, y al ser esta integración doble se debe corregir el error de la primera antes de realizar la segunda.

La cantidad de materia aprendida a lo largo de este proyecto me ha despertado cierto interés en el hardware libre, crear mis propios proyectos y ayudar a otros a crear los suyos. Ya estoy visualizando mi próximo proyecto: construir un dron… ¡Las posibilidades parecen infinitas!

Espero que el lector haya quedado satisfecho con la lectura del proyecto tanto como yo lo he quedado realizándolo, y que si tiene alguna idea sobre algún proyecto que se pueda realizar mediante Arduino no se lo piense dos veces y siga adelante.

GLOSARIO DE TÉRMINOS

A lo largo del proyecto se ha hecho referencia en a distintas siglas que, pese a que se han definido la primera vez que han sido mencionadas, el lector puede haber olvidado y no encontrado su definición. Algunas de ellas son:

- AHRS: Attitude and Heading Reference System (Sistema de Referencia de Actitud y Rumbo).
- IMU: Inertial Measurement Unit (Unidad de Medición Inercial).
- MEMS: Microelectromechanical Systems (Sistemas Microelectromecánicos).
- Sketch: Programa desarrollado para Arduino.
- SDA: Serial Data (Canal de datos del protocolo de comunicación I^2C).
- SCL: Serial Clock (Canal del reloj del protocolo de comunicación I^2C).
- X-CTU: Programa para la configuración de los módulos XBee.

ANEXOS

ANEXO I: PROGRAMA ARDUINO EXPLICADO

```
//CÓDIGO ArduIMU

#include <Wire.h>
```

Incluye la librería necesaria para la comunicación I^2C

```
#define ADXL345_ADDRESS (0xA6 >> 1)

#define ADXL345_REGISTER_XLSB (0x32)

#define ADXL345_BW_RATE (0x2C)

#define ADXL345_3200HZ (0x0F)

#define ADXL_REGISTER_PWRCTL (0x2D)

#define ADXL_PWRCTL_MEASURE (1 << 3)
```

Definimos las diferentes direcciones de acceso a los distintos registros del acelerómetro, que nos servirán para modificar su configuración y leer sus lecturas.

Los distintos registros se han obtenido de la datasheet del acelerómetro.

```
#define ITG3200_ADDRESS (0xD0 >> 1)

#define ITG3200_REGISTER_XMSB (0x1B)

#define ITG3200_REGISTER_DLPF_FS (0x16)

#define ITG3200_FULLSCALE (0x03 << 3)

#define ITG3200_256HZ (0x00)
```

Realizamos la definición de direcciones de registros, esta vez del giróscopo.

Comprobar dichas direcciones en la datasheet del giróscopo.

```
int accelerometer_data[3];

int gyro_data[3];

int a;
```

Definimos algunas variables, tanto enteras como de caracteres.

```
int b;

char c;
```

Inicio del setup del sketch

```
void setup() {

 Wire.begin();

 Serial.begin(57600);
```

Arrancamos la librería para la comunicación I^2C, y abrimos a su vez el puerto serie con un baudrate de 57600 Baudios, que es el máximo permitido por el shield XBee Add-on for Nano.

```
 for(int i = 0; i < 3; ++i) {

  accelerometer_data[i] = gyro_data[i] = 0;

 }
```

Inicializamos las variables de almacenamiento de la lectura

```
 init_adxl345();

 init_itg3200();

 a=0

}
```

Iniciamos el setup del acelerómetro y del giróscopo. Éstas funciones "init" están definidas más adelante.

```
void loop() {
```

Inicio de la repetición en bucle (loop) del sketch

```
 a=Serial.read();

 if (a=='a'){

  while (a!='b')

  {
```

Leemos el puerto serie de conexión al PC esperando leer la variable "a". Llegados a este punto, hasta que Arduino no lea dicho carácter ("a") del PC, se estanca ya que la variable "a" funciona como interruptor de encendido del programa, enviar "a" desde el PC equivale a establecer la conexión y el envío de datos.

```
  read_adxl345();

  read_itg3200();
```

Leemos los datos del acelerómetro y del giróscopo. Las funciones "read" quedan definidas más adelante.

```
  Serial.println(-accelerometer_data[1]);

  Serial.println(-accelerometer_data[2]);

  Serial.println(-accelerometer_data[0]);

  Serial.println(gyro_data[0]);

  Serial.println(-gyro_data[2]);

  Serial.println(-gyro_data[3]);

  Serial.println(-gyro_data[1]);
```

Enviamos al PC los datos del sensor en orden:

ax, ay, az, temperatura del giróscopo, gx, gy, gz;

siendo a y g las aceleraciones y velocidades angulares respectivamente.

```
  a=Serial.read();
```

Código interruptor para pausar la lectura y envío de datos. El carácter "b" actúa como interruptor de apagado.

```
  if (a=='b'){

    break;

  }

}

}

}
```

```
void i2c_write(int address, byte reg, byte data) {
```

```
 Wire.beginTransmission(address);
 Wire.write(reg);
 Wire.write(data);
 Wire.endTransmission();
}
```

El programa i2c_write interactua con la dirección del registro definido, escribiendo en él los datos pertinentes.

```
void i2c_read(int address, byte reg, int count, byte* data) {
 int i = 0;
 Wire.beginTransmission(address);
 Wire.write(reg);
 Wire.endTransmission();
 Wire.beginTransmission(address);
 Wire.requestFrom(address,count);
 while(Wire.available()){
  c = Wire.read();
  data[i] = c;
  i++;
 }
 Wire.endTransmission();
}
```

El programa i2c_read interactua con la dirección del registro definido, leyendo en él los datos pertinentes.

```
void init_adxl345() {
  byte data = 0;
```

El programa init_adxl345 es el programa que inicia el acelerómetro. Ésta es la puesta a punto inicial del acelerómetro, escribiendo y leyendo los registros necesarios. Éste código sólo se lee una vez en todo el programa ya que está en la sección “setup” del sketch.

```
 i2c_write(ADXL345_ADDRESS, ADXL_REGISTER_PWRCTL, ADXL_PWRCTL_MEASURE);
 i2c_write(ADXL345_ADDRESS, ADXL345_BW_RATE, ADXL345_3200HZ);

 i2c_read(ADXL345_ADDRESS, ADXL_REGISTER_PWRCTL, 1, &data);
 Serial.println((unsigned int)data);
}
```

El programa read_adxl345 es el programa que lee los datos del acelerómetro. Éste código se lee una constantemente ya que está en la sección "loop" del sketch.

```
void read_adxl345() {
 byte bytes[6];
 memset(bytes,0,6);

 i2c_write(ADXL345_ADDRESS, ADXL_REGISTER_PWRCTL, ADXL_PWRCTL_MEASURE);

 i2c_read(ADXL345_ADDRESS, ADXL345_REGISTER_XLSB, 6, bytes);

 for (int i=0;i<3;++i) {
 accelerometer_data[i] = (int)bytes[2*i] + (((int)bytes[2*i + 1]) << 8);
 }
}

void init_itg3200() {
  byte data = 0;
```

El programa init_itg3200 es el programa que inicia el giróscopo. Ésta es la puesta a punto inicial del giróscopo, escribiendo y leyendo los registros necesarios. Éste código sólo se lee una vez en todo el programa ya que está en la sección "setup" del sketch.

```
  i2c_write(ITG3200_ADDRESS, ITG3200_REGISTER_DLPF_FS, ITG3200_FULLSCALE |
ITG3200_256HZ);


  i2c_read(ITG3200_ADDRESS, ITG3200_REGISTER_DLPF_FS, 1, &data);


  Serial.println((unsigned int)data);
}


void read_itg3200() {
  byte bytes[8];
  memset(bytes,0,8);


  i2c_read(ITG3200_ADDRESS, ITG3200_REGISTER_XMSB, 8, bytes);
  for (int i=0;i<4;++i) {
  gyro_data[i] = (int)bytes[2*i + 1] + (((int)bytes[2*i]) << 8);
  }
}
```

El programa read_itg_3200 es el programa que lee los datos del giróscopo. Éste código se lee una constantemente ya que está en la sección “loop” del sketch.

ANEXOS II Y III: DATASHEETS DE ADXL345 E ITG3200

Three-Axis, ±2/4/8/16*g* Digital Accelerometer

Preliminary Technical Data

ADXL345

FEATURES

Ultra low power: 25 to 130 µA at V_S = 2.5 V (typ)
Power consumption scales automatically with bandwidth
User selectable fixed 10-bit resolution or 4mg/LSB scale factor in all *g*-ranges, up to 13-bit resolution at ±16 *g*
32 level output data FIFO minimizes host processor load
Built in motion detection functions

- **Tap/Double Tap detection**
- **Activity/Inactivity monitoring**
- **Free-Fall detection**

Supply and I/O voltage range: 1.8 V to 3.6 V
SPI (3 and 4 wire) and I²C digital interfaces
Flexible interrupt modes – Any interrupt mappable to either interrupt pin
Measurement ranges selectable via serial command
Bandwidth selectable via serial command
Wide temperature range (-40 to +85°C)
10,000 *g* shock survival
Pb free/RoHS compliant
Small and thin: 3 × 5 × 1 mm LGA package

APPLICATIONS

Handsets
Gaming and pointing devices
Personal navigation devices
HDD protection
Fitness equipment
Digital cameras

GENERAL DESCRIPTION

The ADXL345 is a small, thin, low power, three-axis accelerometer with high resolution (13-bit) measurement up to ±16 *g*. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I²C digital interface.

The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4mg/LSB) enables resolution of inclination changes of as little as 0.25°.

Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion and if the acceleration on any axis exceeds a user-set level. Tap sensing detects single and double taps. Free-Fall sensing detects if the device is falling. These functions can be mapped to interrupt output pins. An integrated 32 level FIFO can be used to store data to minimize host processor intervention.

Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation.

The ADXL345 is supplied in a small, thin 3 mm × 5 mm × 1 mm, 14-lead, plastic package.

FUNCTIONAL BLOCK DIAGRAM

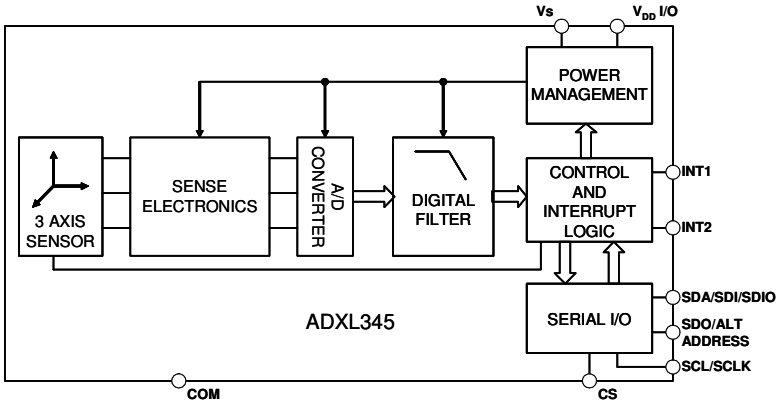


Figure 1. ADXL345 Simplified Block Diagram

Rev. PrA

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.461.3113

TABLE OF CONTENTS

REVISION HISTORY

11/08—Rev. PrA - Initial Version

SPECIFICATIONS

T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V, Acceleration = 0 *g*, unless otherwise noted.

Table 1. Specifications[1]

Parameter	Conditions	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range	User Selectable		±2, 4, 8, 16		*g*
Nonlinearity	Percentage of full scale		±0.5		%
Inter-Axis Alignment Error			±0.1		Degrees
Cross-Axis Sensitivity[2]			±1		%
OUTPUT RESOLUTION	Each axis				
All g-ranges	10-bit mode		10		Bits
±2 *g* range	Full-Resolution		10		Bits
±4 *g* range	Full-Resolution		11		Bits
±8 *g* range	Full-Resolution		12		Bits
±16 *g* range	Full-Resolution		13		Bits
SENSITIVITY	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	V_S = 2.5 V, ±2 *g* 10-bit or Full-Resolution	232	256	286	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	V_S = 2.5 V, ±2 *g* 10-bit or Full-Resolution	3.5	3.9	4.3	m*g*/LSB
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	V_S = 2.5 V, ±4 *g* 10-bit mode	116	128	143	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	V_S = 2.5 V, ±4 *g* 10-bit mode	7.0	7.8	8.6	m*g*/LSB
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	V_S = 2.5 V, ±8 *g* 10-bit mode	58	64	71	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	V_S = 2.5 V, ±8 *g* 10-bit mode	14.0	15.6	17.2	m*g*/LSB
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	V_S = 2.5 V, ±16 *g* 10-bit mode	29	32	36	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	V_S = 2.5 V, ±16 *g* 10-bit mode	28.1	31.2	34.3	m*g*/LSB
Sensitivity Change due to Temperature			±0.02		%/°C
0 *g* BIAS LEVEL	Each axis				
0 *g* Output (X_{OUT}, Y_{OUT}, Z_{OUT})	V_S = 2.5 V, T_A = 25°C	-150	0	+150	m*g*
0 *g* Offset vs. Temperature			<±1		m*g*/°C
NOISE PERFORMANCE					
Noise (x-, y-axes)	Data Rate = 100 Hz, ±2 *g* 10-bit or Full-Res.		<1		LSB RMS
Noise (z-axis)	Data Rate = 100 Hz, ±2 *g* 10-bit or Full-Res.		<1.5		LSB RMS
OUTPUT DATA RATE / BANDWIDTH	User Selectable				
Measurement Rate[3]		0.1		3200	Hz
SELF TEST					
Output Change X		+0.31		+1.02	*g*
Output Change Y		-0.31		-1.02	*g*
Output Change Z		+0.46		+1.64	*g*
POWER SUPPLY					
Operating Voltage Range (V_S)		2.0	2.5	3.6	V
Interface Voltage Range ($V_{DD\ I/O}$)		1.7	1.8	V_S	V
Supply Current	Data Rate > 100 Hz		130	150	µA
Supply Current	Data Rate < 10 Hz		25		µA
Standby Mode Leakage Current			0.1	2	µA
Turn-On Time[4]	Data Rate = 3200 Hz		1.4		ms
TEMPERATURE					
Operating Temperature Range		−40		85	°C
WEIGHT					
Device Weight			20		mgrams

[1] All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.
[2] Cross-axis sensitivity is defined as coupling between any two axes.
[3] Bandwidth is half the output data rate.
[4] Turn-on and wake-up times are determined by the user defined bandwidth. At 100 Hz data rate the turn-on/wake-up time is approximately 11.1 ms. For additional data rates the turn-on/wake-up time is approximately τ + 1.1 in milliseconds, where τ is 1/(Data Rate).

ABSOLUTE MAXIMUM RATINGS

Table 2. Absolute Maximum Ratings

Parameter	Rating
Acceleration (Any Axis, Unpowered)	10,000 *g*
Acceleration (Any Axis, Powered)	10,000 *g*
V_S	−0.3 V to 3.6 V
$V_{DD\ I/O}$	−0.3 V to 3.6
All Other Pins	−0.3 V to 3.6
Output Short-Circuit Duration (Any Pin to Ground)	Indefinite
Temperature Range (Powered)	−40°C to +105°C
Temperature Range (Storage)	−40°C to +105°C

Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Electrostatic charges as high as 4000 V readily accumulate on the human body and test equipment and can discharge without detection. Although this product features proprietary ESD protection circuitry, permanent damage may occur on devices subjected to high energy electrostatic discharges. Therefore, proper ESD precautions are recommended to avoid performance degradation or loss of functionality.

PIN CONFIGURATION AND DESCRIPTIONS

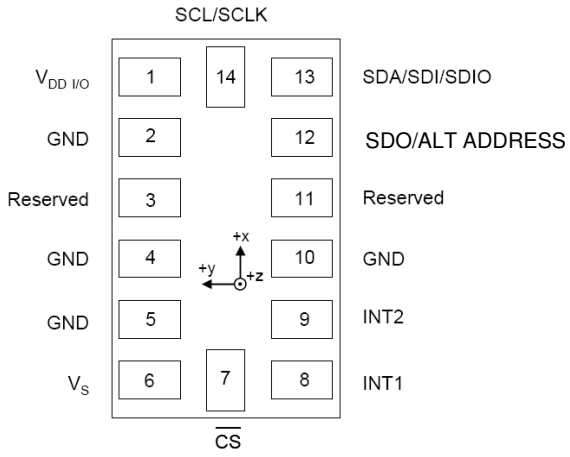


Figure 2. Pin Configuration (TOP VIEW)

Figure 2. Pin Configuration (Top View)

Table 3. Pin Descriptions

Pin No.	Mnemonic	Description
1	$V_{DD\ I/O}$	Digital Interface Supply Voltage
2	GND	Must be connected to ground
3	Reserved	Reserved, must be connected to V_S or left open
4	GND	Must be connected to ground
5	GND	Must be connected to ground
6	V_S	Supply Voltage
7	$\overline{CS}$	Chip Select
8	INT1	Interrupt 1 Output
9	INT2	Interrupt 2 Output
10	GND	Must be connected to ground
11	Reserved	Reserved, must be connected to GND or left open
12	SDO/ALT ADDRESS	Serial Data Out, Alternate I^2C Address Select
13	SDA/SDI/SDIO	Serial Data (I^2C), Serial Data In (SPI 4-Wire), Serial Data In/Out (SPI 3-Wire)
14	SCL/SCLK	Serial Communications Clock

TYPICAL PERFORMANCE CHARACTERISTICS

FUNCTIONAL DESCRIPTION

DEVICE OPERATION

The ADXL345 is a complete three-axis acceleration measurement system with a selectable measurement range of either ±2 *g*, ±4 *g*, ±8 *g*, *or* ±16 *g*. It measures both dynamic acceleration resulting from motion or shock and static acceleration, such as gravity, which allows it to be used as a tilt sensor. The sensor is a polysilicon surface-micromachined structure built on top of a silicon wafer. Polysilicon springs suspend the structure over the surface of the wafer and provide a resistance against acceleration forces. Deflection of the structure is measured using differential capacitors that consist of independent fixed plates and plates attached to the moving mass. Acceleration deflects the beam and unbalances the differential capacitor, resulting in a sensor output whose amplitude is proportional to acceleration. Phase-sensitive demodulation is used to determine the magnitude and polarity of the acceleration.

POWER SEQUENCING

Power may be applied to V_S or $V_{DD\ I/O}$ in any sequence without damaging the ADXL345. All possible power on states are summarized in Table 4. The interface voltage level is set with the interface supply voltage $V_{DD\ I/O}$, which must be present to ensure that the ADXL345 does not create a conflict on the communications bus. For single-supply operation, $V_{DD\ I/O}$ can be the same as the main supply, V_S. Conversely, in a dual-supply application, $V_{DD\ I/O}$ can differ from V_S to accommodate the desired interface voltage. Once V_S is applied, the device enters standby state, where power consumption is minimized and the device waits for $V_{DD\ I/O}$ to be applied and a command to enter measurement state (setting the MEASURE bit in the **POWER_CTL** register). Clearing the MEASURE bit returns the device to standby state.

Table 4. Power Sequencing

Condition	V_S	$V_{DD\ I/O}$	Description
Power Off	Off	Off	Completely off, potential for communications bus conflict.
Bus Enabled	Off	On	No functions available, but will not create conflict on communications bus.
Standby or Measurement	On	On	At power up the device is in Standby mode awaiting a command to enter measurement mode and all sensor functions are off. Once instructed to enter Measurement mode, all sensor functions are available.

POWER SAVING

Power Modes

The ADXL345 automatically modulates its power consumption proportionally with its output data rate as shown in Table 5. If additional power savings is desired, a lower power mode is available. In this mode, the internal sampling rate is reduced allowing for power savings in the 12.5 to 400Hz data rate range at the expense of slightly greater noise. To enter lower power mode, set the LOW_POWER bit(D4) in the **BW_RATE** register.

Table 5. Current Consumption versus Data Rate

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (µA)
3200	1600	1111	130
1600	800	1110	80
800	400	1101	130
400	200	1100	130
200	100	1011	130
100	50	1010	130
50	25	1001	80
25	12.5	1000	55
12.5	6.25	0111	37
6.25	3.125	0110	25
3.125	1.563	0101	25
1.563	0.782	0100	25
0.782	0.39	0011	25
0.39	0.195	0010	25
0.195	0.098	0001	25
0.098	0.048	0000	25

The current consumption in Low Power Mode is shown in Table 6. Cases where there is no advantage to using Low Power Mode are shaded.

Table 6. Current Consumption versus Data Rate in Low Power Mode

Output Data Rate	Bandwidth (Hz)	Rate Code	I_{DD} (µA)
3200	1600	1111	130
1600	800	1110	80
800	400	1101	130
400	200	1100	80
200	100	1011	55
100	50	1010	37
50	25	1001	30
25	12.5	1000	25
12.5	6.25	0111	25
6.25	3.125	0110	25
3.125	1.563	0101	25
1.563	0.782	0100	25
0.782	0.39	0011	25
0.39	0.195	0010	25
0.195	0.098	0001	25
0.098	0.048	0000	25

Auto Sleep Mode

Additional power can be saved by having the ADXL345 automatically switch to sleep mode during periods of inactivity. To enable this feature set the **THRESH_INACT** register to an acceleration value that signifies no activity (this value will depend on the application), set **TIME_INACT** to an appropriate inactivity time period (again, this will depend on the application), and set the AUTO_SLEEP bit and the LINK bit in the **POWER_CTL** register. Current consumption at the sub-8Hz data rates used in this mode is typically 25 μA.

Standby Mode

For even lower power operation Standby Mode can be used. In Standby Mode current consumption is reduced to 2μA (typical). In this mode no measurements are made and communication with the ADXL345 is limited to single-byte read or writes. Standby Mode is entered by clearing the MEASURE bit (D3) in the **POWER_CTL** register. Placing the device into Standby Mode will preserve the contents of the FIFO.

SERIAL COMMUNICATIONS

I²C and SPI digital communications are available. In both cases, the ADXL345 operates as a slave. I²C mode is enabled if the $\overline{CS}$ pin is tied high to $V_{DD\ I/O}$. In SPI mode, the $\overline{CS}$ pin is controlled by the bus master. In both SPI and I²C modes of operation, data transmitted from the ADXL345 to the master device should be ignored during writes to the ADXL345.

SPI

For SPI, either 3-wire or 4-wire configuration is possible, as shown in the connection diagrams in Figure 3 and Figure 4. Clearing the SPI bit in the **DATA_FORMAT** register selects 4-wire mode while setting the SPI bit selects 3-wire mode. The maximum SPI clock speed is 5 MHz, with 12 pF maximum loading and the timing scheme follows CPOL = 1, CPHA = 1.

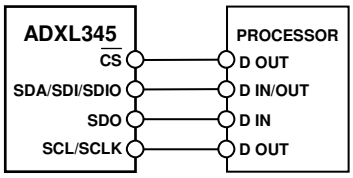


Figure 3. 4-Wire SPI connection

$\overline{CS}$ is the serial port enable line, and is controlled by the SPI master. It must go low at the start of transmissions and back high at the end as shown in Figure 5. SCLK is the serial port clock and is supplied by the SPI master. It is stopped high when $\overline{CS}$ is high, during period of no transmission. SDI and SDO are the serial data in and out respectively. Data should be sampled at the rising edge of SCLK.

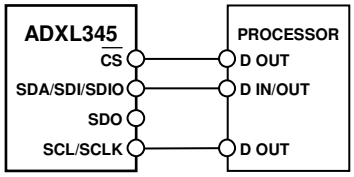


Figure 4. 3-Wire SPI connection

To read or write multiple bytes in a single transmission, the Multi-Byte bit, located after the R/W bit in the first byte transfer, must be set. After the register addressing and the first byte of data, continued clock pulses will cause the ADXL345 to point to the next register for read or write. Continued clock pulses will continue to shift the register that is pointed to until the clock pulses are ceased and $\overline{CS}$ is de-asserted. To perform reads or writes on different, non-sequential registers, $\overline{CS}$ must be de-asserted between transmissions and the new register must be addressed separately.

The timing diagram for 3-wire SPI reads or writes is shown in Figure 5. The 4-wire equivalents for SPI reads and writes are shown in Figure 6 and Figure 7 respectively.

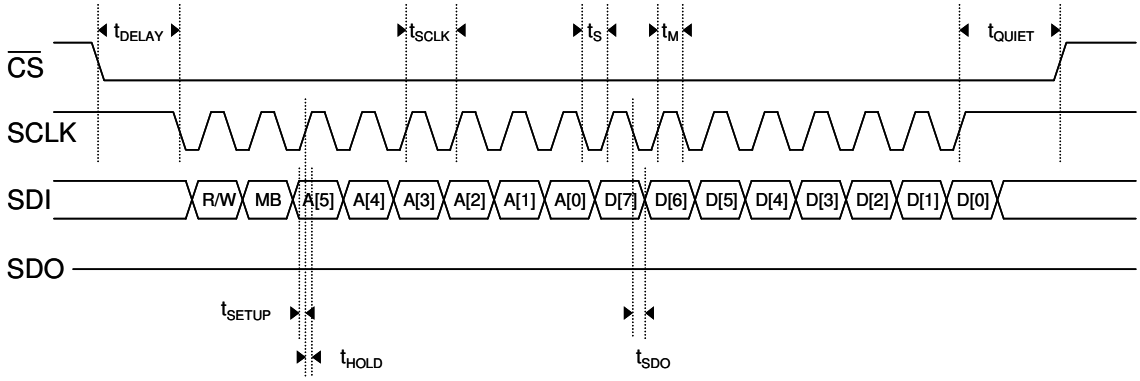


Figure 5. SPI 3-wire Timing Diagram

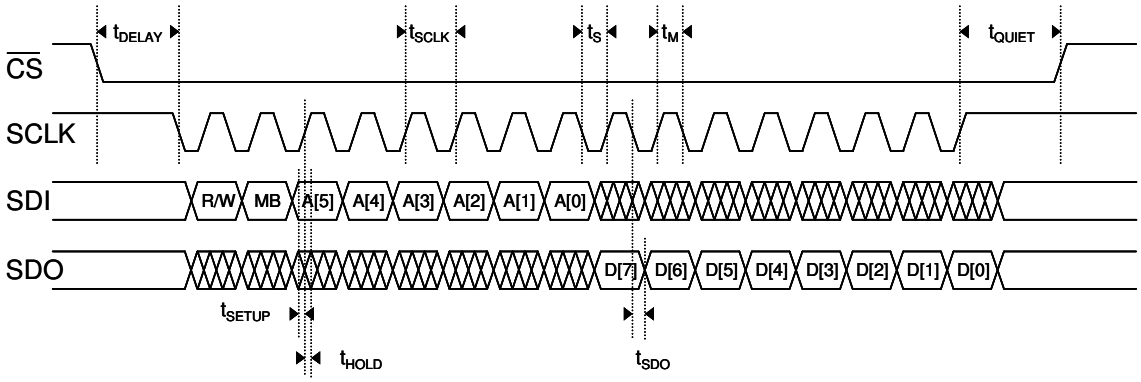


Figure 6. SPI 4-wire Read Timing Diagram

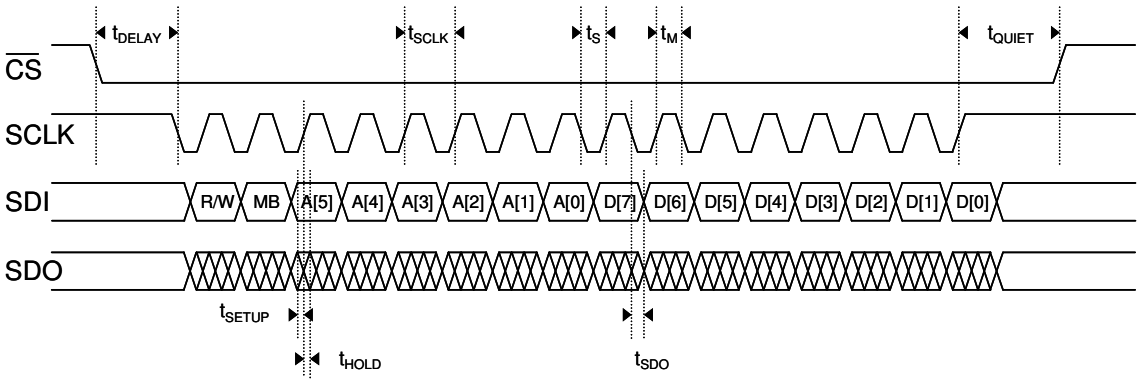


Figure 7. SPI 4-wire Write Timing Diagram

Table 7. SPI Timing Specifications
(TA = 25°C, VS = 2.5V, VDD I/O = 1.8V)

Parameter	Limit	Unit	Description
f_{SCLK}	5	MHz max	SPI clock frequency
t_{SCLK}	200	ns min	1/(SPI clock frequency) Mark/space ratio for the SCLK input is 40/60 to 60/40
t_{DELAY}	200	ns min	falling edge to SCLK falling edge
t_{QUIET}	200	ns min	SCLK rising edge to rising edge
t_S	0.4 × t_{SCLK}	ns min	SCLK low pulse width (space)
t_M	0.4 × t_{SCLK}	ns min	SCLK high pulse width (mark)
t_{SDO}	8	ns max	SCLK falling edge to SDO transition
t_{SETUP}	10	ns min	SDI valid before SCLK rising edge
t_{HOLD}	10	ns min	SDI valid after SCLK rising edge

I²C

With $\overline{CS}$ tied high to $V_{DD\ I/O}$, the ADXL345 is in I²C mode, requiring a simple 2-wire connection as shown in Figure 8. The ADXL345 conforms to *The I²C Bus Specification*, Version 2.1, January 2000, available from Phillips Semiconductor. It supports standard (100 kHz) and fast (400 kHz) data transfer modes. Single or multiple byte read/writes are supported as shown in Figure 9. With the SDO pin high the 7 bit I²C address for the device is 0x1D, followed by the read/write bit. This translates to 0x3A for write, 0x3B for read. An alternate I²C address of 0x53 (followed by the read/write bit) may be chosen by grounding the SDO pin (pin 12). This translates to 0xA6 for write, 0xA7 for read.

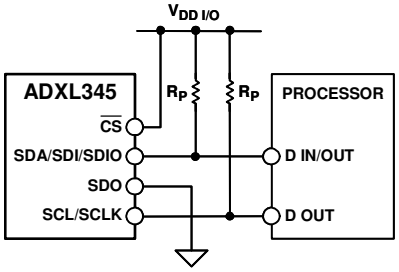


Figure 8. I²C Connection Diagram (Address = 0x53)

If other devices are connected to the same I²C bus, the nominal operating voltage level of these other devices cannot exceed $V_{DD\ I/O}$ by more than 0.3 V. Pull-up resistors, R_P, should be in the range of 1k to 20kΩ.

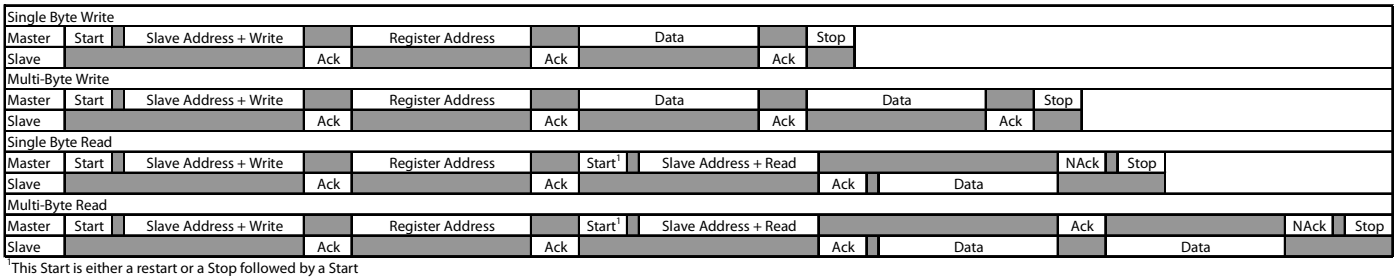


Figure 9. I²C Timing Diagram

INTERRUPTS

The ADXL345 provides two output pins for driving interrupts: INT1 and INT2. Each of the interrupt functions are described in detail below. All functions can be used simultaneously, with the only limiting feature being that some functions may need to share interrupt pins. Interrupts are enabled by setting the appropriate bit in the **INT_ENABLE** register and are mapped to either the INT1 or INT2 pins based on the contents of the **INT_MAP** register. It is recommended that interrupts be configured with the interrupts disabled, preventing interrupts from being accidentally triggered during configuration. This can be done by writing a value of 0x00 to the **INT_ENABLE** register.

DATA_READY

DATA_READY is set when new data is available and cleared when no new data is available.

SINGLE_TAP

SINGLE_TAP is set when single acceleration event that is greater than the value in the **THRESH_TAP** register occurs for a time shorter than specified in the **DUR** register.

DOUBLE_TAP

DOUBLE_TAP is set when two acceleration events that are greater than the value in the **THRESH_TAP** register occur that are shorter than the time specified in the **DUR** register, with the second tap starting after the time specified by the **LATENT** register and within the time specified in the **WINDOW** register. See the Tap Detection description in the Application section for more details.

ACTIVITY

ACTIVITY is set when acceleration greater than the value stored in **THRESH_ACT** is experienced.

INACTIVITY

INACTIVITY is set when acceleration of less than the value stored in the **THRESH_INACT** register is experienced for longer than the time specified in the **TIME_INACT** register. The maximum value for **TIME_INACT** is 255 s.

FREE_FALL

FREE_FALL is set when acceleration of less than the value stored in the **THRESH_FF** register is experienced for longer than the time specified in the **TIME_FF** register. The FREE_FALL interrupt differs from INACTIVITY interrupt in that all axes always participate, the timer period is much smaller (1.28 s maximum) and it is always DC coupled.

WATERMARK

WATERMARK is set when the FIFO has filled up to the value stored in SAMPLES. It is cleared automatically when the FIFO is read and its content emptied below the value stored in SAMPLES.

OVERRUN

OVERRUN is set when new data has replaced unread data. The precise operation of OVERRUN depends on the FIFO mode. In Bypass Mode, OVERRUN is set when new data replaces unread data in the **DATAX**, **DATAY**, and **DATAZ** registers. In all other modes, OVERRUN is set when the FIFO is filled. OVERRUN is cleared by reading the FIFO contents, and is automatically cleared when the data is read.

FIFO

The ADXL345 contains a 32 level FIFO that can be used to minimize host processor intervention. The FIFO has four modes as described in Table 15 in the Register Definitions section. Mode selection is made by setting the appropriate MODE bits in the **FIFO_CTL** register. Each FIFO mode is described below.

Bypass Mode

In Bypass Mode the FIFO is not operational and remains empty.

FIFO Mode

In FIFO Mode data from X, Y, and Z measurements go into the FIFO. When the FIFO is filled to the level specified in SAMPLES (in the **FIFO_CTL** register), the WATERMARK interrupt is set. The FIFO will continue filling until it is full (32 X, Y, and Z samples) and then stop collecting data. After the FIFO has stopped collecting data the device still continues to operate, so features like Tap detection, for example, may still be used once the FIFO is full. The WATERMARK interrupt will continue to occur until the number of samples in the FIFO is less than the value of SAMPLES in the **FIFO_CTL** register.

Stream Mode

In Stream Mode data from X, Y, and Z measurements go into the FIFO. When the FIFO is filled to the level specified in SAMPLES (in the **FIFO_CTL** register), the WATERMARK interrupt is set. The FIFO will continue filling, and will hold the latest 32 X, Y, and Z samples, discarding older data as new data arrives. The WATERMARK interrupt will continue to occur until the number of samples in the FIFO is less than the value of SAMPLES in the **FIFO_CTL** register.

Trigger Mode

In Trigger Mode, the FIFO fills and holds the latest 32 X, Y, and Z samples. Once a trigger event occurs (as described by the TRIG_SOURCE bit in the **FIFO_CTL** register), the FIFO will keep the last n samples (where n is the value specified by SAMPLES in the **FIFO_CTL** register) and then operate in FIFO mode, collecting new samples only when the FIFO is not full. Additional trigger events will not be recognized until Trigger Mode is reset. This can be done by setting the device in Bypass Mode, reading the FIFO_STATUS register and then setting the device back into Trigger Mode. The FIFO data should be read first, as placing the device into Bypass Mode will clear the FIFO.

Retrieving Data from the FIFO

FIFO data is read through the **DATAX**, **DATAY** and **DATAZ** registers. When the FIFO is in FIFO, Stream, or Trigger Modes, reads to the **DATAX**, **DATAY**, and **DATAZ** registers read data stored in the FIFO. Each time any data is read from the FIFO the oldest X, Y, and Z data is placed into the **DATAX**, **DATAY** and **DATAZ** registers. If a single byte read operation is performed, the remaining bytes worth of data will be lost. Therefore, all axes of interest should be read in a burst (or multi-byte) read operation. To ensure that the FIFO has completely popped, there must be at least 5 µs between the end of reading the data registers, signified by the transition to register 0x38 from 0x37 or the $\overline{CS}$ pin going high, and the start of new reads of the FIFO or reading the **FIFO_STATUS** register. For SPI operation at 1.5 MHz or lower, the register addressing portion of the transmission is sufficient delay to ensure the FIFO has completely popped. It is necessary for SPI operation greater than 1.5 MHz to de-assert the $\overline{CS}$ pin to ensure a total of 5 µs, which is at most 3.4 µs at 5 MHz operation. This is not a concern when using I^2C, as the communication rate is low enough to ensure a sufficient delay between FIFO reads.

SELF TEST

The ADXL345 incorporates a Self Test feature that effectively tests both its mechanical and electronic systems. When the Self Test function is enabled (via the SELF_TEST bit in the **DATA_FORMAT** register), an electrostatic force is exerted on the mechanical sensor. This electrostatic force moves the mechanical sensing element in the same manner as acceleration, and it is additive to the acceleration experienced by the device. This added electrostatic force results in an output change in the X, Y, and Z-axes. Because the electrostatic force is proportional to V_S^2, the output change varies with V_S.

The Self Test feature on the ADXL345 exhibits a bi-modal behavior which depends upon which phase of the clock Self Test is enabled. Due to this, a typical value for Self Test is not reported; however, the limits shown in Table 1 and below are valid for both potential values.

Table 8. Self Test output in LSB for 2 *g* and Full-Resolution

	Vs = 2.5 V			Vs = 3.3 V		
	Min.	Typ.	Max.	Min.	Typ.	Max.
X-Axis	+80		+260	+140		+455
Y-Axis	-80		-260	-140		-455
Z-Axis	+120		+420	+210		+730

Table 9. Self Test output in LSB for 4 *g* 10-bit

	Vs = 2.5 V			Vs = 3.3 V		
	Min.	Typ.	Max.	Min.	Typ.	Max.
X-Axis	+40		+130	+70		+225
Y-Axis	-40		-130	-70		-225
Z-Axis	+60		+210	+105		+365

Table 10. Self Test output in LSB for 8 *g* 10-bit

	Vs = 2.5 V			Vs = 2.5 V		
	Min.	Typ.	Max.	Min.	Typ.	Max.
X-Axis	+20		+65	+35		+113
Y-Axis	-20		-65	-35		-113
Z-Axis	+30		+105	+52		+183

Table 11. Self Test output in LSB for 16 *g* 10-bit

	Vs = 2.5 V			Vs = 2.5 V		
	Min.	Typ.	Max.	Min.	Typ.	Max.
X-Axis	+10		+33	+17		+57
Y-Axis	-10		-33	-17		-57
Z-Axis	+15		+53	+26		+92

REGISTER MAP

Table 12. Register Map

Hex	Dec	Name	Type	Reset Value	Description
0	0	DEVID	R	11100101	Device ID.
1	1	Reserved			Reserved. Do not access.
to					Reserved. Do not access.
1C	28	Reserved			Reserved. Do not access.
1D	29	THRESH_TAP	R/W	00000000	Tap threshold
1E	30	OFSX	R/W	00000000	X axis offset
1F	31	OFSY	R/W	00000000	Y axis offset
20	32	OFSZ	R/W	00000000	Z axis offset
21	33	DUR	R/W	00000000	Tap duration
22	34	LATENT	R/W	00000000	Tap latency
23	35	WINDOW	R/W	00000000	Tap window
24	36	THRESH_ACT	R/W	00000000	Activity threshold
25	37	THRESH_INACT	R/W	00000000	Inactivity threshold
26	38	TIME_INACT	R/W	00000000	Inactivity time
27	39	ACT_INACT_CTL	R/W	00000000	Axis enable control for ACT/INACT
28	40	THRESH_FF	R/W	00000000	Free-fall threshold
29	41	TIME_FF	R/W	00000000	Free-fall time
2A	42	TAP_AXES	R/W	00000000	Axis control for Tap/Double Tap
2B	43	ACT_TAP_STATUS	R	00000000	Source of Tap/Double Tap
2C	44	BW_RATE	R/W	00001010	Data Rate and Power Mode control
2D	45	POWER_CTL	R/W	00000000	Power Save features control
2E	46	INT_ENABLE	R/W	00000000	Interrupt enable control
2F	47	INT_MAP	R/W	00000000	Interrupt mapping control
30	48	INT_SOURCE	R	00000000	Source of interrupts
31	49	DATA_FORMAT	R/W	00000000	Data format control
32	50	DATAX0	R	00000000	X axis data
33	51	DATAX1	R	00000000	
34	52	DATAY0	R	00000000	Y axis data
35	53	DATAY1	R	00000000	
36	54	DATAZ0	R	00000000	Z axis data
37	55	DATAZ1	R	00000000	
38	56	FIFO_CTL	R	00000000	FIFO control
39	57	FIFO_STATUS	R/W	00000000	FIFO status

REGISTER DEFINITIONS

0x00 DEVID (read-only)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	0	0	1	0	1

DEVID holds a fixed device ID code of 0xE5 (345 octal).

0x1D THRESH_TAP (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
MSB	D6	D5	D4	D3	D2	D1	LSB

THRESH_TAP holds the threshold value for tap interrupts. The data format is unsigned, so the magnitude of the tap event is compared to **THRESH_TAP**. The scale factor is 62.5 mg/LSB (i.e. 0xFF = +16 g). A zero value may result in undesirable behavior if Tap/Double Tap interrupts are enabled.

0x1E, 0x1F, 0x20 OFSX, OFSY, OFSZ (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
MSB	D6	D5	D4	D3	D2	D1	LSB

OFSX/OFSY/OFSZ offer user offset adjustments in twos-compliment form with a scale factor of 15.6 mg/LSB (i.e. 0x7F = +2 g).

0x21 DUR (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
MSB	D6	D5	D4	D3	D2	D1	LSB

DUR is an unsigned time value representing the maximum time that an event must be above the **THRESH_TAP** threshold to qualify as a tap event. The scale factor is 625 µs/LSB. A zero value will prevent Tap/Double Tap functions from working.

0x22 LATENT (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
MSB	D6	D5	D4	D3	D2	D1	LSB

LATENT is an unsigned time value representing the wait time from the detection of a tap event to the opening of the time window **WINDOW** for a possible second tap event. The scale factor is 1.25 ms/LSB. A zero value will disable the Double Tap function.

0x23 WINDOW (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
MSB	D6	D5	D4	D3	D2	D1	LSB

WINDOW is an unsigned time value representing the amount of time after the expiration of **LATENT** during which a second tap can begin. The scale factor is 1.25 ms/LSB. A zero value will disable the Double Tap function.

0x24 THRESH_ACT (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
MSB	D6	D5	D4	D3	D2	D1	LSB

THRESH_ACT holds the threshold value for activity detection. The data format is unsigned, so the magnitude of the activity event is compared to **THRESH_ACT**. The scale factor is 62.5 mg/LSB. A zero value may result in undesirable behavior if Activity interrupt is enabled.

0x25 THRESH_INACT (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
MSB	D6	D5	D4	D3	D2	D1	LSB

THRESH_INACT holds the threshold value for inactivity detection. The data format is unsigned, so the magnitude of the inactivity event is compared to **THRESH_INACT**. The scale factor is 62.5 mg/LSB. A zero value may result in undesirable behavior if Inactivity interrupt is enabled.

0x26 TIME_INACT (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
MSB	D6	D5	D4	D3	D2	D1	LSB

TIME_INACT is an unsigned time value representing the amount of time that acceleration must be below the value in **THRESH_INACT** for inactivity to be declared. The scale factor is 1 second/LSB. Unlike the other interrupt functions, which operate on unfiltered data(See Threshold description in Application section), the inactivity function operates on the filtered output data. At least one output sample must be generated for the inactivity interrupt to be triggered. This will result in the function appearing un-responsive if the **TIME_INACT** register is set with a value less than the time constant of the Output Data Rate. A zero value will result in an interrupt when the output data is below **THRESH_INACT**.

0x27 ACT_INACT_CONTROL (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
ACT AC/DC	ACT_X Enable	ACT_Y Enable	ACT_Z Enable	INACT AC/DC	INACT_X Enable	INACT_Y Enable	INACT_Z Enable

X/Y/Z Enable: A '1' enables X, Y, or Z participation in activity or inactivity detection. A '0' excludes the selected axis from participation. If all of the axes are excluded, the function is disabled.

AC/DC: A '0' = DC coupled operation and a '1' = AC coupled operation. In DC coupled operation, the current acceleration is compared with **THRESH_ACT** and **THRESH_INACT** directly to determine whether activity or inactivity is detected. In AC coupled operation for activity detection, the acceleration value at the start of activity detection is taken as a reference value. New samples of acceleration are then compared to this reference value and if the magnitude of the difference exceeds **THRESH_ACT** the device will trigger an activity interrupt. In AC coupled operation for inactivity detection, a reference value is used again for comparison and is updated whenever the device exceeds the inactivity threshold. Once the reference value is selected, the device compares the magnitude of the difference between the reference value and the current acceleration with **THRESH_INACT**. If the difference is below **THRESH_INACT** for a total of **TIME_INACT**, the device is considered inactive and the inactivity interrupt is triggered.

0x28 THRESH_FF (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
MSB	D6	D5	D4	D3	D2	D1	LSB

THRESH_FF holds the threshold value for Free-Fall detection. The data format is unsigned. The root-sum-square(RSS) value of all axes is calculated and compared to the value in **THRESH_FF** to determine if a free fall event may be occurring. The scale factor is 62.5 mg/LSB. A zero value may result in undesirable behavior if Free-Fall interrupt is enabled. Values between 300 and 600 mg (0x05 to 0x09) are recommended.

0x29 TIME_FF (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
MSB	D6	D5	D4	D3	D2	D1	LSB

TIME_FF is an unsigned time value representing the minimum time that the RSS value of all axes must be less than **THRESH_FF** to generate a Free-Fall interrupt. The scale factor is 5 ms/LSB. A zero value may result in undesirable behavior if Free-Fall interrupt is enabled. Values between 100 to 350 ms (0x14 to 0x46) are recommended.

0x2A TAP_AXES (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	SUPPRESS	TAP_X Enable	TAP_Y Enable	TAP_Z Enable

TAP_X/Y/Z Enable: A '1' in TAP_X, Y, or Z Enable enables X, Y, or Z participation in Tap detection. A '0' excludes the selected axis from participation in Tap detection.

Setting the SUPPRESS bit suppresses Double Tap detection if acceleration greater than **THRESH_TAP** is present between taps. See Tap Detection in the Application Section for more details.

0x2B ACT_TAP_STATUS (read)

D7	D6	D5	D4	D3	D2	D1	D0
X	ACT_X Source	ACT_Y Source	ACT_Z Source	ASLEEP	TAP_X Source	TAP_Y Source	TAP_Z Source

X/Y/Z Source: Indicate the first axis involved in a Tap or Activity event. A '1' corresponds to involvement in the event and a '0' corresponds to no involvement. They are not cleared, but overwritten by new data. **ACT_TAP_STATUS** should be read before clearing the interrupt. Disabling an axis from participation will clear the corresponding Source bit when the next Activity or Tap/Double Tap event occurs.

ASLEEP: A '1' indicates that the part is in the Auto Sleep Mode. A '0' indicates that the part is not using Auto Sleep Mode. See the **POWER_CTL** description for more information on Auto Sleep Mode.

0x2C BW_RATE (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	LOW_POWER	RATE			

LOW_POWER: A '0' = Normal operation and a '1' = Reduced power operation with somewhat higher noise. (See Power Modes section for details).

RATE: Selects device bandwidth and output data rate. See Table 5 and Table 6 for details. Default value is 0x0A, or 100 Hz Output Data Rate. An Output Data Rate should be selected that is appropriate for the communication protocol and frequency selected. Selecting too high of an Output Data Rate with a low communication speed will result in samples being discarded.

0x2D POWER_CTL (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
X	X	LINK	AUTO_SLEEP	MEASURE	SLEEP	WAKEUP	

LINK: A '1' with both the activity and inactivity functions enabled will delay the start of the activity function until inactivity is detected. Once activity is detected, inactivity detection will begin and prevent the detection of activity. This bit serially links the activity and inactivity functions. When '0' the inactivity and activity functions are concurrent. Additional information can be found in the Application section under Link Mode.

AUTO_SLEEP: A '1' sets the ADXL345 to switch to Sleep Mode when inactivity (acceleration has been below THRESH_INACT for at least TIME_INACT) is detected and the LINK bit is set. A '0' disables automatic switching to Sleep Mode. See SLEEP for further description.

MEASURE: A '0' places the part into standby mode and a '1' places the part into measurement mode. The ADXL345 powers up in standby mode with minimum power consumption.

SLEEP: A '0' puts the part into a normal mode of operation. A '1' places the part into Sleep Mode. This suppresses DATA_READY, stops sending data to the FIFO, and switches the sampling rate to one specified by the WAKEUP bits. In Sleep Mode, only the Activity function can be used.

When clearing the LINK, AUTO_SLEEP, or SLEEP bits, it is recommended that the part be placed into Standby when clearing the bits and then re-enabling Measurement mode during a following write. This is done to ensure the device is properly biased if Sleep mode is manually disabled. Not toggling Measurement mode may result in the first few after LINK, AUTO_SLEEP, or SLEEP is cleared having additional noise, especially if the device was asleep when the bits were cleared.

WAKEUP: Controls the frequency of readings in Sleep Mode as shown in

Table 13 below:

Table 13. WAKEUP Rates

D1	D0	Frequency (Hz)
0	0	8
0	1	4
1	0	2
1	1	1

0x2E INT_ENABLE (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
DATA_READY	SINGLE_TAP	DOUBLE_TAP	ACTIVITY	INACTIVITY	FREE_FALL	WATERMARK	OVERRUN

Setting bits with a value of '1' in this register to enable their respective functions and generate interrupts. A value of '0' will prevent the functions from generating an interrupt. DATA_READY, WATERMARK, and OVERRUN bits only enable the interrupt output; the functions are always enabled. It is recommended that interrupts be configured before enabling their outputs.

0x2F INT_MAP (read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
DATA_READY	SINGLE_TAP	DOUBLE_TAP	ACTIVITY	INACTIVITY	FREE_FALL	WATERMARK	OVERRUN

Any '0' bits in this register send their respective interrupts to the INT1 pin. Bits set with a '1' send their respective interrupts to the INT2 pin. All selected interrupts for a given pin are ORed.

0x30 INT_SOURCE (read)

D7	D6	D5	D4	D3	D2	D1	D0
DATA_READY	SINGLE_TAP	DOUBLE_TAP	ACTIVITY	INACTIVITY	FREE_FALL	WATERMARK	OVERRUN

Bits set with a '1' in this register indicate that their respective functions have triggered. A value of '0' indicates that the corresponding event has not occurred. DATA_READY, WATERMARK and OVERRUN bits will always be set if corresponding event occurs, regardless of **INT_ENABLE**, and are cleared by reading data from the **DATAX/Y/Z** registers. DATA_READY and WATERMARK may require multiple reads, as per the FIFO Mode descriptions in the FIFO section. Other bits are cleared by reading **INT_SOURCE**.

0x31 DATA_FORMAT (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	X	FULL_RES	JUSTIFY	RANGE	

DATA_FORMAT controls the presentation of data at registers 0x32 to 0x37. All data, except ±16 g range, must be clipped to avoid rollover.

SELF_TEST: A '1' applies a Self Test force to the sensor causing a shift in the output data. A value of '0' disable Self Test.

SPI: A value of '1' sets the device to 3-wire SPI and a value of '0' sets the device to 4-wire SPI.

INT_INVERT: A value of '0' sets the interrupts to Active High while a value of '1' sets the interrupts to Active Low.

FULL_RES: When this bit is set with a value of '1' the device is in Full-Resolution Mode, where the output resolution increases with RANGE to maintain a 4 mg/LSB scale factor. When this bit is '0' the device is in 10-bit Mode and RANGE determine the maximum g-Range and scale factor.

JUSTIFY: A '1' = Left (MSB) justified and a '0' = Right justified with sign extension.

RANGE: Sets the g-Range based on Table 14 below.

Table 14. g-Range Setting

D1	D0	g-Range
0	0	±2 g
0	1	±4 g
1	0	±8 g
1	1	±16 g

0x32, 0x33 DATAX0, DATAX1 (read only)

D7	D6	D5	D4	D3	D2	D1	D0

0x34, 0x35 DATAY0, DATAY1 (read only)

D7	D6	D5	D4	D3	D2	D1	D0

0x36, 0x37 DATAZ0, DATAZ1 (read only)

D7	D6	D5	D4	D3	D2	D1	D0

These six bytes hold the output data for each axis. The output data is two's complement with **DATAx0** as the LSByte and **DATAx1** as the MSByte. The **DATA_FORMAT** register (0x31) controls the format of the data. It is recommended that a burst read of all of the registers is performed to prevent the change of data between reads of sequential registers.

0x38 FIFO_CTL (read/write)

D7	D6	D5	D4	D3	D2	D1	D0
FIFO_MODE		TRIGGER	SAMPLES				

FIFO_MODE: Corresponds to the FIFO Mode as shown in Table 15 below.

TRIGGER: A value of '0' sets the trigger event of Trigger Mode to INT1 and a value of '1' sets the trigger event to INT2.

SAMPLES: Function depends on the FIFO Mode as shown in Table 16 below. Entering a value of zero in SAMPLES will immediately set the WATERMARK status bit in **INT_SOURCE**, regardless of FIFO mode. Undesirable operation may occur if a value of zero is used for SAMPLES when Trigger Mode is used.

Table 15. FIFO Modes

D7	D6	MODE	Function
0	0	Bypass	The FIFO is bypassed
0	1	FIFO	FIFO collects up to 32 values then stops collecting data
1	0	Stream	FIFO holds the last 32 data values. Once full, the FIFO's oldest data is lost as it is replaced with newer data
1	1	Trigger	When triggered by the TRIGGER the FIFO holds the last 32 data values and stops when full.

Table 16. SAMPLES Functions

FIFO Mode	Samples Function
Bypass	None
FIFO	Specifies how many FIFO entries are need to trigger a Watermark interrupt
Stream	Specifies how many FIFO entries are need to trigger a Watermark interrupt
Trigger	Specifies how many FIFO samples *before* the trigger event are retained in the FIFO buffer

0x39 FIFO_STATUS (read)

D7	D6	D5	D4	D3	D2	D1	D0
FIFO_TRIG	X	ENTRIES					

FIFO_TRIG: A '1' corresponds to a trigger event occurring while a '0' means that a FIFO trigger event has not yet occurred.

ENTRIES: Reports how many data values are stored in the FIFO. To collect the data from the FIFO, access is through the standard **DATAX**, **DATAY**, and **DATAZ** registers. FIFO reads must be done in burst, or multi-byte, mode as each FIFO level is cleared after any read, single- or multi-byte, of the FIFO. The FIFO stores a maximum of 32 entries, which equates to a maximum of 33 entries available at any given time, due to the fact that an additional entry is available at the output filter of device.

APPLICATION

POWER SUPPLY DECOUPLING

In many applications, a 0.1 µF capacitor at V_S and $V_{DD\ I/O}$ placed close to the ADXL345 supply pins adequately decouples the accelerometer from noise on the power supply. However, in applications where noise is present at the 50 kHz internal clock frequency, or any harmonic thereof, additional care in power supply bypassing is required because this noise may cause errors in acceleration measurement. If additional decoupling is necessary, a 10 Ω resistor or ferrite in series with V_S and an additional larger bypass capacitor (2.2 µF or greater) at V_S may be helpful.

Care should be taken that the connection from the ADXL345 ground to the power supply ground be low impedance because noise transmitted through ground has an effect similar to noise transmitted through V_S.

MECHANICAL CONSIDERATIONS FOR MOUNTING

The ADXL345 should be mounted on the PCB in a location close to a hard mounting point of the PCB to the case. Mounting the ADXL345 at an unsupported PCB location (that is, at the end of a "lever," or in the middle of a "trampoline"), as shown in Figure 10, may result in large apparent measurement errors because the accelerometer will see the resonant vibration of the PCB. Locating the accelerometer near a hard mounting point ensures that any PCB resonances at the accelerometer are above the accelerometer's mechanical sensor resonant frequency and, therefore, effectively invisible to the accelerometer.

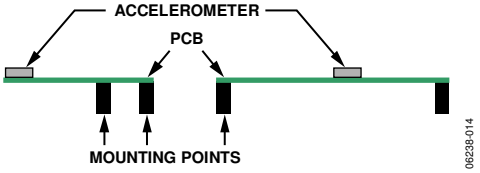


Figure 10. Where Not to Mount an Accelerometer

TAP DETECTION

The tap interrupt function is capable of detecting either single or double taps. The following parameters are shown graphically in Figure 11 for a valid single and valid double tap event:

- Tap detection threshold is defined by the **THRESH_TAP** register.
- Maximum tap duration time is defined by the **DUR** register.
- Tap latency time is defined by the **LATENT** register and is the waiting period from the end of the first tap until the opening of the time window, whose value is contained in the **WINDOW** register, for a possible second tap.
- The interval time after the expiration of **LATENT** is defined by the **WINDOW** register and is the period of time during which a second tap must begin. The second tap need not finish before the end of **WINDOW**.

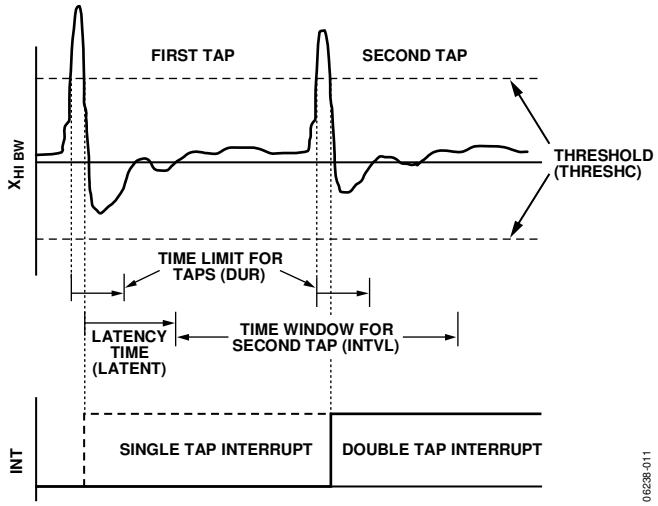


Figure 11. Tap Interrupt Function with Valid Single and Double Taps

If only the single tap function is in use, the single tap interrupt will trigger at the point that the acceleration goes below the threshold as long as **DUR** is not exceeded. If both single and double tap functions are in use the single tap interrupt will trigger once the double tap event has been either validated or invalidated.

Several events can occur to invalidate the second tap of a double tap event. First, if the SUPPRESS bit in the **TAP_AXES** register is set, any acceleration spikes above the threshold during the **LATENT** time window will invalidate the double tap as seen in Figure 12.

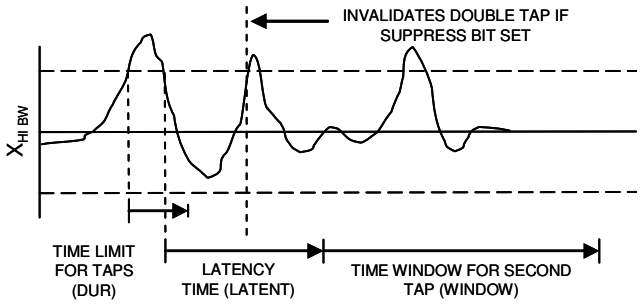


Figure 12. Double Tap event invalid due to high-g event with SUPPRESS set

A double tap event can also be invalidated if an acceleration above the threshold is detected at the start of **WINDOW**, resulting in an invalid double tap at the start of **WINDOW**, shown in Figure 13. Additionally, a double tap event can be invalidated by having an acceleration exceed **DUR**, resulting in an invalid double tap at the end of **DUR** for the second tap event, also seen in Figure 13.

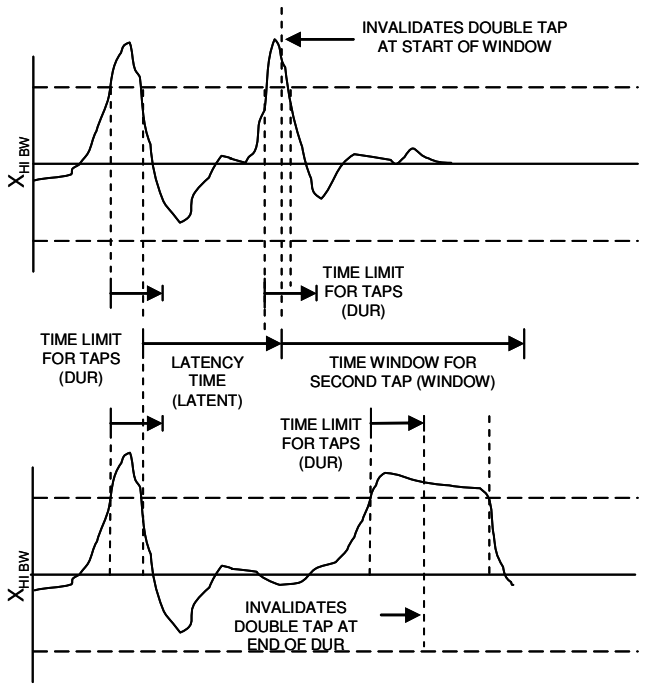


Figure 13. Tap Interrupt Function with Invalid Double Taps

Single taps, double taps, or both may be detected by setting their respective bits in the **INT_ENABLE** register. Control over participation of each of the three axes in tap/double tap detection is exerted by setting the appropriate bits in the **TAP_AXES** register. For the double tap function to operate, both **LATENT** and **WINDOW** must be non-zero.

Every mechanical system will have somewhat different tap/double tap response based on the system's mechanical characteristics, so some experimentation with values for the **LATENT**, **WINDOW**, and **THRESH_TAP** registers will be required. In general a good starting point is **LATENT**>0x10, **WINDOW**>0x10, and **THRESH_TAP**>3g. Setting very low values in the **LATENT**, **WINDOW**, and/or **THRESH_TAP** registers may result in unpredictable response due to the accelerometer picking up "echoes" of the tap inputs.

After a tap interrupt is received, the first axis to exceed the **THRESH_TAP** level is reported in the **ACT_TAP_STATUS** register. This register is never cleared, but overwritten with new data.

THRESHOLD

The lower Output Data Rates are achieved by decimation of a common sampling frequency inside the device. The activity, free-fall and tap/double tap detection functions are performed using the un-filtered data. Since the output data is filtered, the high frequency and high-*g* data that is used to determine activity, free-fall and tap/double tap events may not be present if the output of the accelerometer is examined. This may result in trigger events appearing to occur when acceleration does not appear to trigger an event, such as exceeding a threshold or remaining below a threshold for a certain period of time.

LINK MODE

The LINK function can be used to reduce the number of activity interrupts the processor must service by only looking for activity after inactivity. For proper operation of the link feature, the processor must still respond to the activity and inactivity interrupts by reading the **INT_SOURCE** register to clear them. If the activity interrupt is not cleared, the part will not go into Auto Sleep Mode. The ASLEEP bit in the **ACT_TAP_STATUS** register indicates if the part is in Auto Sleep Mode.

RECOMMENDED PWB LAND PATTERN

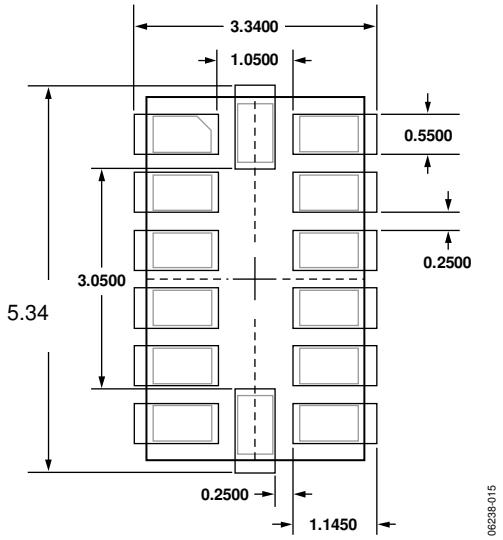


Figure 14. Recommended Printed Wiring Board Land Pattern (Dimensions Shown in Millimeters)

RECOMMENDED SOLDERING PROFILE

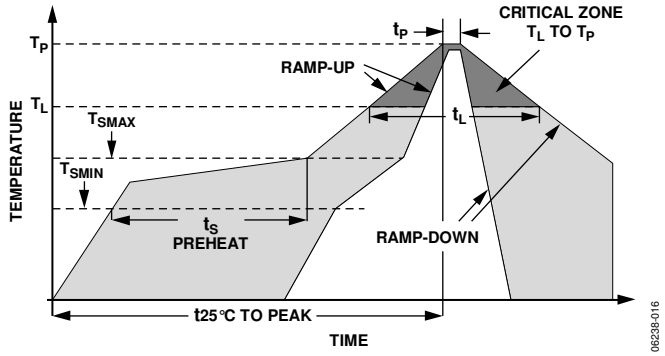
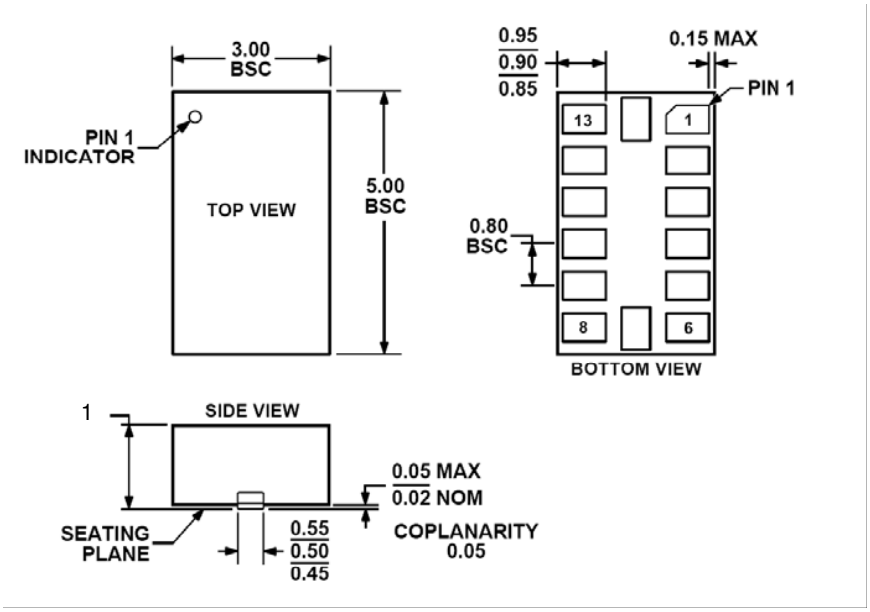


Figure 15. Recommended Soldering Profile

Table 17. Recommended Soldering Profile[1]

	Condition	
Profile Feature	**Sn63/Pb37**	**Pb-Free**
Average Ramp Rate (T_L to T_P)	3°C/sec max	3°C/sec max
Preheat		
Minimum Temperature (T_{SMIN})	100°C	150°C
Maximum Temperature (T_{SMAX})	150°C	200°C
Time (T_{SMIN} to T_{SMAX})(t_S)	60 sec to 120 sec	60 sec to 180 sec
T_{SMAX} to T_L		
Ramp-Up Rate	3°C/sec max	3°C/sec max
Time Maintained Above Liquidous (T_L)		
Liquidous Temperature (T_L)	183°C	217°C
Time (t_L)	60 sec to 150 sec	60 sec to 150 sec
Peak Temperature (T_P)	240 + 0/−5°C	260 + 0/−5°C
Time Within 5°C of Actual Peak Temperature (t_P)	10 sec to 30 sec	20 sec to 40 sec
Ramp-Down Rate	6°C/sec max	6°C/sec max
Time 25°C to Peak Temperature	6 minutes max	8 minutes max

OUTLINE DIMENSIONS



ORDERING GUIDE

Model	Measurement Range	Specified Voltage (V)	Temperature Range	Package Description	Package Option
ADXL345BCCZ[1]	±2, 4, 8, 16*g*	2.5	−40°C to +85°C	14-Lead Land Grid Array Package [LGA]	TBD
ADXL345BCCZ–RL[1]	±2, 4, 8, 16*g*	2.5	−40°C to +85°C	14-Lead Land Grid Array Package [LGA]	TBD
ADXL345BCCZ–RL7[1]	±2, 4, 8, 16*g*	2.5	−40°C to +85°C	14-Lead Land Grid Array Package [LGA]	TBD
EVAL-ADXL345Z[1]				Evaluation Board	

[1] Z = Pb-free part.

NOTES

NOTES

PR07925-0-11/08(PrA)

InvenSense Inc.
1197 Borregas Ave, Sunnyvale, CA 94089 U.S.A.
Tel: +1 (408) 988-7339 Fax: +1 (408) 988-8104
Website: www.invensense.com

Document Number: PS-ITG-3200A-00-01.4
Revision: 1.4
Release Date: 03/30/2010

ITG-3200
Product Specification
Revision 1.4

CONTENTS

1 Document Information

1.1 Revision History

Revision Date	Revision	Description
10/23/09	1.0	Initial Release
10/28/09	1.1	Edits for readability
02/12/2010	1.2	• Changed full-scale range and sensitivity scale factor (Sections 2, 3.1, 5.3, and 8.3) • Changed sensitivity scale factor variation over temperature (Section 3.1) • Changed total RMS noise spec (Section 3.1) • Added range for temperature sensor (Section 3.1) • Updated VDD Power-Supply Ramp Rate specification (Sections 3.2 and 4.4) • Added VLOGIC Voltage Range condition (Section 3.2) • Added VLOGIC Reference Voltage Ramp Rate specification (Sections 3.2 and 4.4) • Updated Start-Up Time for Register Read/Write specification (Section 3.2) • Updated Input logic levels for AD0 and CLKIN (Section 3.2) • Updated Level I_{OL} specifications for the I^2C interface (Section 3.3) • Updated Frequency Variation Over Temperature specification for internal clock source (Section 3.4) • Updated VLOGIC conditions for I^2C Characterization (Section 3.5) • Updated ESD specification (Section 3.6) • Added termination requirements for CLKIN if unused (Section 4.1) • Added recommended power-on procedure diagram (Section 4.4) • Changed DLPF_CFG setting 7 to reserved (Section 8.3) • Changed Reflow Specification description (Section 9.12) • Removed errata specifications
03/05/2010	1.3	• Updated temperature sensor linearity spec (Section 3.1) • Updated VDD Power-Supply Ramp Rate timing figure (Sections 3.2 and 4.4) • Updated VLOGIC Reference Voltage timing figure (Section 4.4) • Added default values to registers (all of Section 8) • Updated FS_SEL description (Section 8.3) • Updated package outline drawing and dimensions (Section 9.2) • Updated Reliability (Section 10.1 and 10.2) • Removed Environmental Compliance (Section 11)
03/30/2010	1.4	• Removed confidentiality mark

1.2 Purpose and Scope

This document is a preliminary product specification, providing a description, specifications, and design related information for the ITG-3200™. Electrical characteristics are based upon simulation results and limited characterization data of advanced samples only. Specifications are subject to change without notice. Final specifications will be updated based upon characterization of final silicon.

1.3 Product Overview

The ITG-3200 is the world's first single-chip, digital-output, 3-axis MEMS gyro IC optimized for gaming, 3D mice, and 3D remote control applications. The part features enhanced bias and sensitivity temperature stability, reducing the need for user calibration. Low frequency noise is lower than previous generation devices, simplifying application development and making for more-responsive remote controls.

The ITG-3200 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyro outputs, a user-selectable internal low-pass filter bandwidth, and a Fast-Mode I^2C (400kHz) interface. Additional features include an embedded temperature sensor and a 2% accurate internal oscillator. This breakthrough in gyroscope technology provides a dramatic 67% package size reduction, delivers a 50% power reduction, and has inherent cost advantages compared to competing multi-chip gyro solutions.

By leveraging its patented and volume-proven Nasiri-Fabrication platform, which integrates MEMS wafers with companion CMOS electronics through wafer-level bonding, InvenSense has driven the ITG-3200 package size down to a revolutionary footprint of 4x4x0.9mm (QFN), while providing the highest performance, lowest noise, and the lowest cost semiconductor packaging required for handheld consumer electronic devices. The part features a robust 10,000*g* shock tolerance, as required by portable consumer equipment.

For power supply flexibility, the ITG-3200 has a separate VLOGIC reference pin, in addition to its analog supply pin, VDD, which sets the logic levels of its I^2C interface. The VLOGIC voltage may be anywhere from 1.71V min to VDD max.

1.4 Applications

- Motion-enabled game controllers
- Motion-based portable gaming
- Motion-based 3D mice and 3D remote controls
- "No Touch" UI
- Health and sports monitoring

2 Features

The ITG-3200 triple-axis MEMS gyroscope includes a wide range of features:

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyros) on one integrated circuit with a sensitivity of 14.375 LSBs per °/sec and a full-scale range of ±2000°/sec
- Three integrated 16-bit ADCs provide simultaneous sampling of gyros while requiring no external multiplexer
- Enhanced bias and sensitivity temperature stability reduces the need for user calibration
- Low frequency noise lower than previous generation devices, simplifying application development and making for more-responsive motion processing
- Digitally-programmable low-pass filter
- Low 6.5mA operating current consumption for long battery life
- Wide VDD supply voltage range of 2.1V to 3.6V
- Flexible VLOGIC reference voltage allows for I^2C interface voltages from 1.71V to VDD
- Standby current: 5µA
- Smallest and thinnest package for portable devices (4x4x0.9mm QFN)
- No high pass filter needed
- Turn on time: 50ms
- Digital-output temperature sensor
- Factory calibrated scale factor
- 10,000 *g* shock tolerant
- Fast Mode I^2C (400kHz) serial interface
- On-chip timing generator clock frequency is accurate to +/-2% over full temperature range
- Optional external clock inputs of 32.768kHz or 19.2MHz to synchronize with system clock
- MEMS structure hermetically sealed and bonded at wafer level
- RoHS and Green compliant

3 Electrical Characteristics

3.1 Sensor Specifications

Typical Operating Circuit of Section 4.2, VDD = 2.5V, VLOGIC = 1.71V to VDD, T_A=25°C.

Parameter	Conditions	Min	Typical	Max	Unit	Note
GYRO SENSITIVITY						
Full-Scale Range	FS_SEL=3		±2000		º/s	4
Gyro ADC Word Length			16		Bits	3
Sensitivity Scale Factor	FS_SEL=3		14.375		LSB/(º/s)	3
Sensitivity Scale Factor Tolerance	25°C	-6		+6	%	1
Sensitivity Scale Factor Variation Over Temperature			±10		%	2
Nonlinearity	Best fit straight line; 25°C		0.2		%	6
Cross-Axis Sensitivity			2		%	6
GYRO ZERO-RATE OUTPUT (ZRO)						
Initial ZRO Tolerance			±40		º/s	1
ZRO Variation Over Temperature	-40°C to +85°C		±40		º/s	2
Power-Supply Sensitivity (1-10Hz)	Sine wave, 100mVpp; VDD=2.2V		0.2		º/s	5
Power-Supply Sensitivity (10 - 250Hz)	Sine wave, 100mVpp; VDD=2.2V		0.2		º/s	5
Power-Supply Sensitivity (250Hz - 100kHz)	Sine wave, 100mVpp; VDD=2.2V		4		º/s	5
Linear Acceleration Sensitivity	Static		0.1		º/s/*g*	6
GYRO NOISE PERFORMANCE	**FS_SEL=3**					
Total RMS noise	100Hz LPF (DLPFCFG=2)		0.38		º/s-rms	1
Rate Noise Spectral Density	At 10Hz		0.03		º/s/√Hz	2
GYRO MECHANICAL FREQUENCIES						
X-Axis		30	33	36	kHz	1
Y-Axis		27	30	33	kHz	1
Z-Axis		24	27	30	kHz	1
Frequency Separation	Between any two axes	1.7			kHz	1
GYRO START-UP TIME	**DLPFCFG=0**					
ZRO Settling	to ±1º/s of Final		50		ms	6
TEMPERATURE SENSOR						
Range			-30 to +85		ºC	2
Sensitivity			280		LSB/ºC	2
Temperature Offset	35ºC		-13,200		LSB	1
Initial Accuracy	35ºC		TBD		°C	
Linearity	Best fit straight line (-30°C to +85°C)		±1		°C	2, 5
TEMPERATURE RANGE						
Specified Temperature Range		-40		85	ºC	

Notes:

1. Tested in production
2. Based on characterization of 30 pieces over temperature on evaluation board or in socket
3. Based on design, through modeling and simulation across PVT
4. Typical. Randomly selected part measured at room temperature on evaluation board or in socket
5. Based on characterization of 5 pieces over temperature
6. Tested on 5 parts at room temperature

3.2 Electrical Specifications

Typical Operating Circuit of Section 4.2, VDD = 2.5V, VLOGIC = 1.71V to VDD, T_A=25°C.

Parameters	Conditions	Min	Typical	Max	Units	Notes
VDD POWER SUPPLY						
Operating Voltage Range		2.1		3.6	V	2
Power-Supply Ramp Rate	Monotonic ramp. Ramp rate is 10% to 90% of the final value (see Figure in Section 4.4)	0		5	ms	2
Normal Operating Current			6.5		mA	1
Sleep Mode Current			5		µA	5
VLOGIC REFERENCE VOLTAGE						
Voltage Range	VLOGIC must be ≤VDD at all times	1.71		VDD	V	
VLOGIC Ramp Rate	Monotonic ramp. Ramp rate is 10% to 90% of the final value (see Figure in Section 4.4)			1	ms	6
Normal Operating Current			100		µA	
START-UP TIME FOR REGISTER READ/WRITE			20		ms	5
I^2C ADDRESS	AD0 = 0		1101000			6
	AD0 = 1		1101001			6
DIGITAL INPUTS (AD0, CLKIN)						
V_{IH}, High Level Input Voltage		0.9*VLOGIC			V	5
V_{IL}, Low Level Input Voltage				0.1*VLOGIC	V	5
C_I, Input Capacitance				5	pF	7
DIGITAL OUTPUT (INT)						
V_{OH}, High Level Output Voltage	OPEN=0, Rload=1MΩ	0.9*VLOGIC			V	2
V_{OL}, Low Level Output Voltage	OPEN=0, Rload=1MΩ			0.1*VLOGIC	V	2
$V_{OL.INT1}$, INT Low-Level Output Voltage	OPEN=1, 0.3mA sink current			0.1	V	2
Output Leakage Current	OPEN=1		100		nA	4
t_{INT}, INT Pulse Width	LATCH_INT_EN=0		50		µs	4

Notes:

1. Tested in production
2. Based on characterization of 30 pieces over temperature on evaluation board or in socket
4. Typical. Randomly selected part measured at room temperature on evaluation board or in socket
5. Based on characterization of 5 pieces over temperature
6. Guaranteed by design

3.3 Electrical Specifications, continued

Typical Operating Circuit of Section 4.2, VDD = 2.5V, VLOGIC = 1.71V to VDD, T_A=25°C.

Parameters	Conditions	Typical	Units	Notes
I^2C I/O (SCL, SDA)				
V_{IL}, LOW-Level Input Voltage		-0.5 to 0.3*VLOGIC	V	2
V_{IH}, HIGH-Level Input Voltage		0.7*VLOGIC to VLOGIC + 0.5V	V	2
V_{hys}, Hysteresis		0.1*VLOGIC	V	2
V_{OL1}, LOW-Level Output Voltage	3mA sink current	0 to 0.4	V	2
I_{OL}, LOW-Level Output Current	V_{OL} = 0.4V V_{OL} = 0.6V	3 6	mA mA	2 2
Output Leakage Current		100	nA	4
t_{of}, Output Fall Time from V_{IHmax} to V_{ILmax}	Cb bus cap. in pF	20+0.1Cb to 250	ns	2
C_I, Capacitance for Each I/O pin		10	pF	5

Notes:

2. Based on characterization of 5 pieces over temperature.
4. Typical. Randomly selected part measured at room temperature on evaluation board or in socket
5. Guaranteed by design

3.4 Electrical Specifications, continued

Typical Operating Circuit of Section 4.2, VDD = 2.5V, VLOGIC = 1.71V to VDD, T_A=25°C.

Parameters	Conditions	Min	Typical	Max	Units	Notes
INTERNAL CLOCK SOURCE	**CLKSEL=0, 1, 2, or 3**					
Sample Rate, Fast	DLPFCFG=0 SAMPLERATEDIV = 0		8		kHz	4
Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1		kHz	4
Clock Frequency Initial Tolerance	CLKSEL=0, 25°C	-2		+2	%	1
	CLKSEL=1,2,3; 25°C	-1		+1	%	1
Frequency Variation over Temperature	CLKSEL=0		-15 to +10		%	2
	CLKSEL=1,2,3		+/-1		%	2
PLL Settling Time	CLKSEL=1,2,3		1		ms	3
EXTERNAL 32.768kHz CLOCK	**CLKSEL=4**					
External Clock Frequency			32.768		kHz	3
External Clock Jitter	Cycle-to-cycle rms		1 to 2		µs	3
Sample Rate, Fast	DLPFCFG=0 SAMPLERATEDIV = 0		8.192		kHz	3
Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1.024		kHz	3
PLL Settling Time			1		ms	3
EXTERNAL 19.2MHz CLOCK	**CLKSEL=5**					
External Clock Frequency			19.2		MHz	3
Sample Rate, Fast	DLPFCFG=0 SAMPLERATEDIV = 0		8		kHz	3
Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1		kHz	3
PLL Settling Time			1		ms	3
Charge Pump Clock Frequency						
Frequency	1st Stage, 25°C		8.5		MHz	5
	2nd Stage, 25°C		68		MHz	5
	Over temperature		+/-15		%	5

Notes:

1. Tested in production
2. Based on characterization of 30 pieces over temperature on evaluation board or in socket
3. Based on design, through modeling and simulation across PVT
4. Typical. Randomly selected part measured at room temperature on evaluation board or in socket
5. Based on characterization of 5 pieces over temperature.

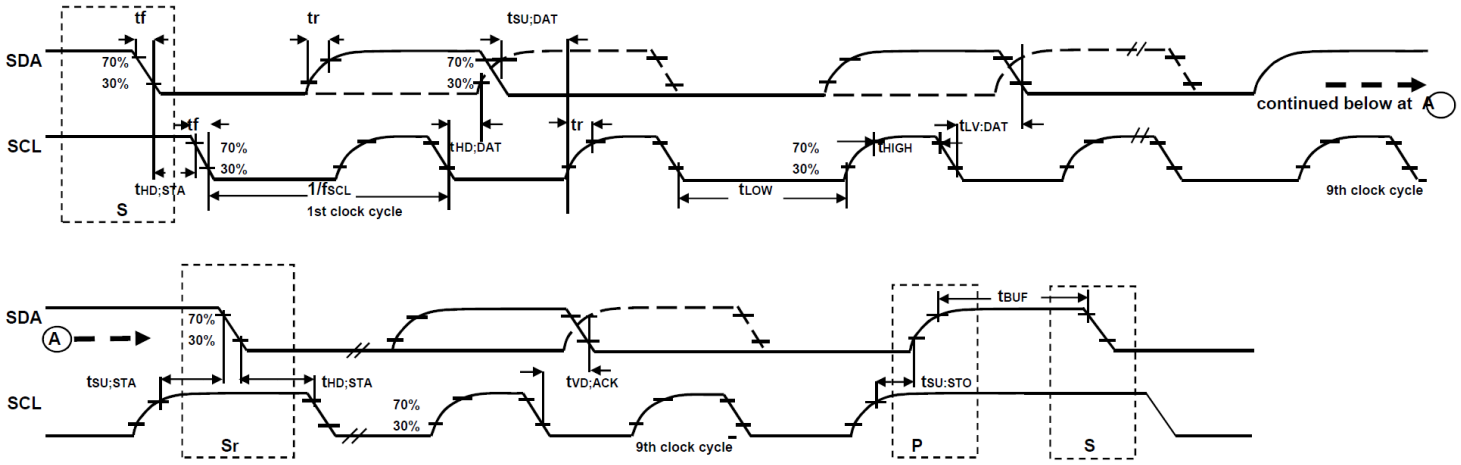
3.5 I²C Timing Characterization

Typical Operating Circuit of Section 4.2, VDD = 2.5V, VLOGIC = 1.8V±5%, 2.5V±5%, 3.0V±5%, or 3.3V±5%, T_A=25°C.

Parameters	Conditions	Min	Typical	Max	Units	Notes
I²C TIMING	**I²C FAST-MODE**					
f_{SCL}, SCL Clock Frequency		0		400	kHz	1
$t_{HD.STA}$, (Repeated) START Condition Hold Time		0.6			us	1
t_{LOW}, SCL Low Period		1.3			us	1
t_{HIGH}, SCL High Period		0.6			us	1
$t_{SU.STA}$, Repeated START Condition Setup Time		0.6			us	1
$t_{HD.DAT}$, SDA Data Hold Time		0			us	1
$t_{SU.DAT}$, SDA Data Setup Time		100			ns	1
t_r, SDA and SCL Rise Time	Cb bus cap. from 10 to 400pF	20+0.1Cb		300	ns	1
t_f, SDA and SCL Fall Time	Cb bus cap. from 10 to 400pF	20+0.1Cb		300	ns	1
$t_{SU.STO}$, STOP Condition Setup Time		0.6			us	1
t_{BUF}, Bus Free Time Between STOP and START Condition		1.3			us	1
C_b, Capacitive Load for each Bus Line				400	pF	2
$t_{VD.DAT}$, Data Valid Time				0.9	us	1
$t_{VD.ACK}$, Data Valid Acknowledge Time				0.9	us	1

Notes:

1. Based on characterization of 5 pieces over temperature on evaluation board or in socket
2. Guaranteed by design



I²C Bus Timing Diagram

3.6 Absolute Maximum Ratings

Stresses above those listed as "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these conditions is not implied. Exposure to the absolute maximum ratings conditions for extended periods may affect device reliability.

Absolute Maximum Ratings

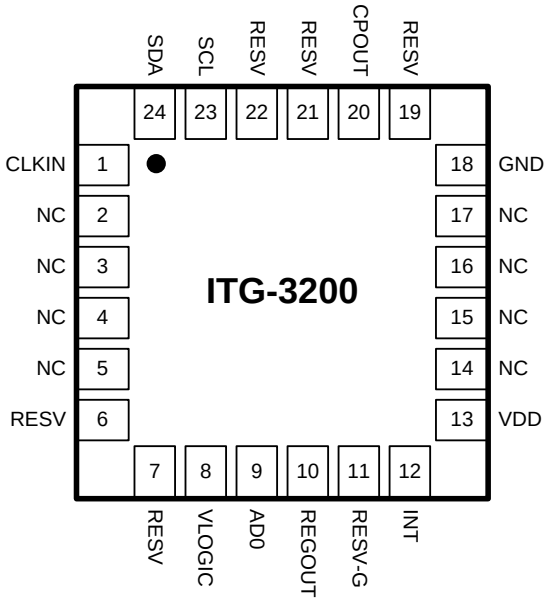
Parameter	Rating
Supply Voltage, VDD	-0.5V to +6V
VLOGIC Input Voltage Level	-0.5V to VDD + 0.5V
REGOUT	-0.5V to 2V
Input Voltage Level (CLKIN, AD0)	-0.5V to VDD + 0.5V
SCL, SDA, INT	-0.5V to VLOGIC + 0.5V
CPOUT (2.1V ≤ VDD ≤ 3.6V)	-0.5V to 30V
Acceleration (Any Axis, unpowered)	10,000*g* for 0.3ms
Operating Temperature Range	-40°C to +105°C
Storage Temperature Range	-40°C to +125°C
Electrostatic Discharge (ESD) Protection	1.5kV (HBM); 200V (MM)

4 Applications Information

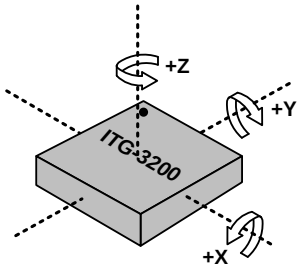
4.1 Pin Out and Signal Description

Number	Pin	Pin Description
1	CLKIN	Optional external reference clock input. Connect to GND if unused.
8	VLOGIC	Digital IO supply voltage. VLOGIC must be ≤ VDD at all times.
9	AD0	I^2C Slave Address LSB
10	REGOUT	Regulator filter capacitor connection
12	INT	Interrupt digital output (totem pole or open-drain)
13	VDD	Power supply voltage
18	GND	Power supply ground
11	RESV-G	Reserved - Connect to ground.
6, 7, 19, 21, 22	RESV	Reserved. Do not connect.
20	CPOUT	Charge pump capacitor connection
23	SCL	I^2C serial clock
24	SDA	I^2C serial data
2, 3, 4, 5, 14, 15, 16, 17	NC	Not internally connected. May be used for PCB trace routing.

Top View

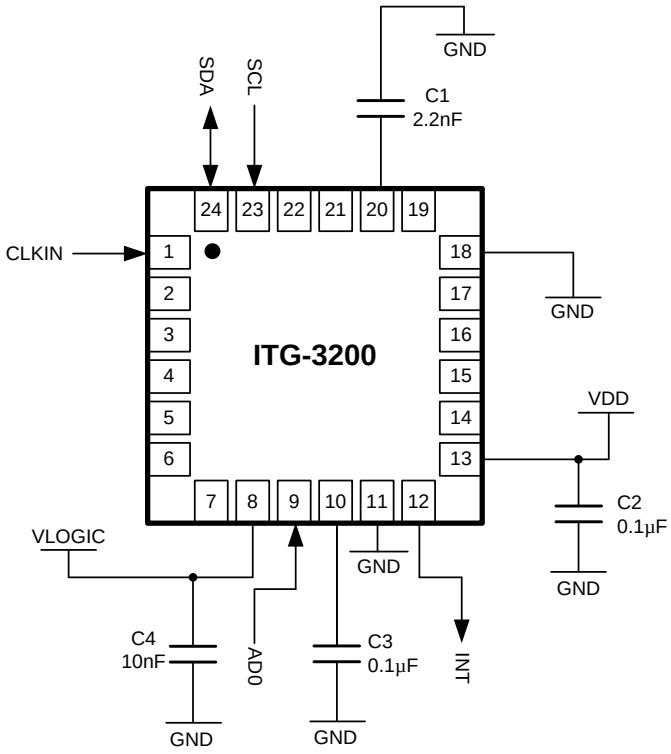


QFN Package
24-pin, 4mm x 4mm x 0.9mm



Orientation of Axes of Sensitivity and Polarity of Rotation

4.2 Typical Operating Circuit

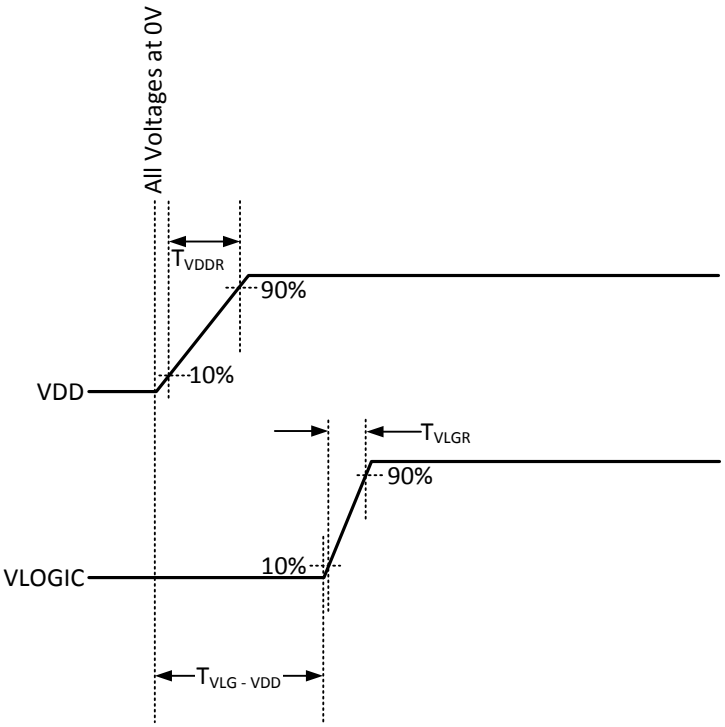


Typical Operating Circuit

4.3 Bill of Materials for External Components

Component	Label	Specification	Quantity
Charge Pump Capacitor	C1	Ceramic, X7R, 2.2nF ±10%, 50V	1
VDD Bypass Capacitor	C2	Ceramic, X7R, 0.1µF ±10%, 4V	1
Regulator Filter Capacitor	C3	Ceramic, X7R, 0.1µF ±10%, 2V	1
VLOGIC Bypass Capacitor	C4	Ceramic, X7R, 10nF ±10%, 4V	1

4.4 Recommended Power-On Procedure

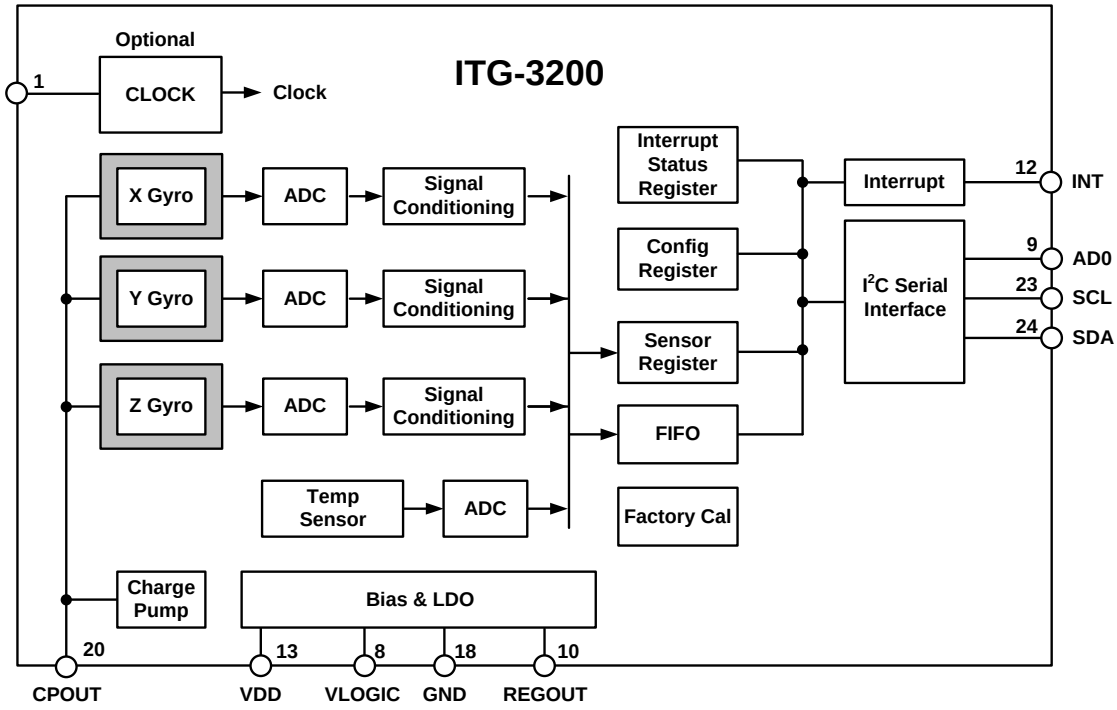


Power-Up Sequencing

1. T_{VDDR} is VDD rise time: Time for VDD to rise from 10% to 90% of its final value
2. T_{VDDR} is ≤5msec
3. T_{VLGR} is VLOGIC rise time: Time for VLOGIC to rise from 10% to 90% of its final value
4. T_{VLGR} is ≤1msec
5. $T_{VLG-VDD}$ is the delay from the start of VDD ramp to the start of VLOGIC rise
6. $T_{VLG-VDD}$ is 0 to 20msec but VLOGIC amplitude must always be ≤VDD amplitude
7. VDD and VLOGIC must be monotonic ramps

5 Functional Overview

5.1 Block Diagram



5.2 Overview

The ITG-3200 consists of the following key blocks and functions:

- Three-axis MEMS rate gyroscope sensors with individual 16-bit ADCs and signal conditioning
- I^2C serial communications interface
- Clocking
- Sensor Data Registers
- Interrupts
- Digital-Output Temperature Sensor
- Bias and LDO
- Charge Pump

5.3 Three-Axis MEMS Gyroscope with 16-bit ADCs and Signal Conditioning

The ITG-3200 consists of three independent vibratory MEMS gyroscopes, which detect rotational rate about the X (roll), Y (pitch), and Z (yaw) axes. When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a deflection that is detected by a capacitive pickoff. The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate. This voltage is digitized using individual on-chip 16-bit Analog-to-Digital Converters (ADCs) to sample each axis.

The full-scale range of the gyro sensors is preset to ±2000 degrees per second (°/s). The ADC output rate is programmable up to a maximum of 8,000 samples per second down to 3.9 samples per second, and user-selectable low-pass filters enable a wide range of cut-off frequencies.

5.4 I²C Serial Communications Interface

The ITG-3200 communicates to a system processor using the I²C serial interface, and the device always acts as a slave when communicating to the system processor. **The logic level for communications to the master is set by the voltage on the VLOGIC pin.** The LSB of the of the I²C slave address is set by pin 9 (AD0).

5.5 Clocking

The ITG-3200 has a flexible clocking scheme, allowing for a variety of internal or external clock sources for the internal synchronous circuitry. This synchronous circuitry includes the signal conditioning, ADCs, and various control circuits and registers. An on-chip PLL provides flexibility in the allowable inputs for generating this clock.

Allowable internal sources for generating the internal clock are:

- An internal relaxation oscillator (less accurate)
- Any of the X, Y, or Z gyros' MEMS oscillators (with an accuracy of ±2% over temperature)

Allowable external clocking sources are:

- 32.768kHz square wave
- 19.2MHz square wave

Which source to select for generating the internal synchronous clock depends on the availability of external sources and the requirements for clock accuracy. There are also start-up conditions to consider. When the ITG-3200 first starts up, the device operates off of its internal clock until programmed to operate from another source. This allows the user, for example, to wait for the MEMS oscillators to stabilize before they are selected as the clock source.

5.6 Sensor Data Registers

The sensor data registers contain the latest gyro and temperature data. They are read-only registers, and are accessed via the Serial Interface. Data from these registers may be read at any time, however, the interrupt function may be used to determine when new data is available.

5.7 Interrupts

Interrupt functionality is configured via the Interrupt Configuration register. Items that are configurable include the INT pin configuration, the interrupt latching and clearing method, and triggers for the interrupt. Items that can trigger an interrupt are (1) Clock generator locked to new reference oscillator (used when switching clock sources); and (2) new data is available to be read from the Data registers. The interrupt status can be read from the Interrupt Status register.

5.8 Digital-Output Temperature Sensor

An on-chip temperature sensor and ADC are used to measure the ITG-3200 die temperature. The readings from the ADC can be read from the Sensor Data registers.

5.9 Bias and LDO

The bias and LDO sections take in an unregulated VDD supply from 2.1V to 3.6V and generate the internal supply and the references voltages and currents required by the ITG-3200. The LDO output is bypassed by a capacitor at REGOUT. Additionally, the part has a VLOGIC reference voltage which sets the logic levels for its I²C interface.

5.10 Charge Pump

An on-board charge pump generates the high voltage (25V) required to drive the MEMS oscillators. Its output is bypassed by a capacitor at CPOUT.

6 Digital Interface

6.1 I²C Serial Interface

The internal registers and memory of the ITG-3200 can be accessed using I²C at up to 400kHz.

Serial Interface

Pin Number	Pin Name	Pin Description
8	VLOGIC	Digital IO supply voltage. VLOGIC must be ≤ VDD at all times.
9	AD0	I²C Slave Address LSB
23	SCL	I²C serial clock
24	SDA	I²C serial data

6.1.1 I²C Interface

I²C is a two wire interface comprised of the signals serial data (SDA) and serial clock (SCL). In general, the lines are open-drain and bi-directional. In a generalized I²C interface implementation, attached devices can be a master or a slave. The master device puts the slave address on the bus, and the slave device with the matching address acknowledges the master.

The ITG-3200 always operates as a slave device when communicating to the system processor, which thus acts as the master. SDA and SCL lines typically need pull-up resistors to VDD. The maximum bus speed is 400kHz.

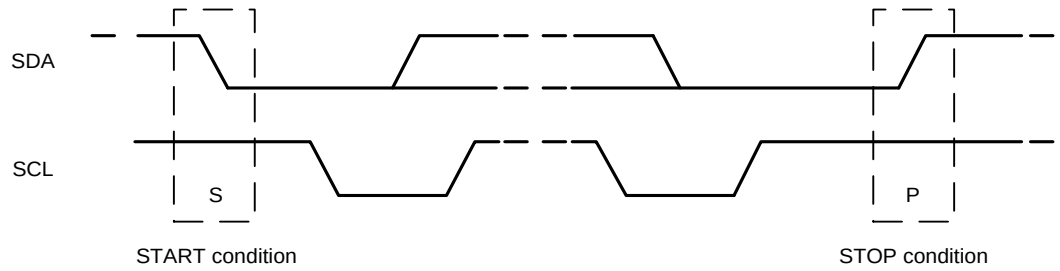
The slave address of the ITG-3200 devices is b110100X which is 7 bits long. The LSB bit of the 7 bit address is determined by the logic level on pin 9. This allows two ITG-3200 devices to be connected to the same I²C bus. When used in this configuration, the address of the one of the devices should be b1101000 (pin 9 is logic low) and the address of the other should be b1101001 (pin 9 is logic high). The I²C address is stored in register 0 (WHO_AM_I register).

I²C Communications Protocol

START (S) and STOP (P) Conditions

Communication on the I²C bus starts when the master puts the START condition (S) on the bus, which is defined as a HIGH-to-LOW transition of the SDA line while SCL line is HIGH (see figure below). The bus is considered to be busy until the master puts a STOP condition (P) on the bus, which is defined as a LOW to HIGH transition on the SDA line while SCL is HIGH (see figure below).

Additionally, the bus remains busy if a repeated START (Sr) is generated instead of a STOP condition.

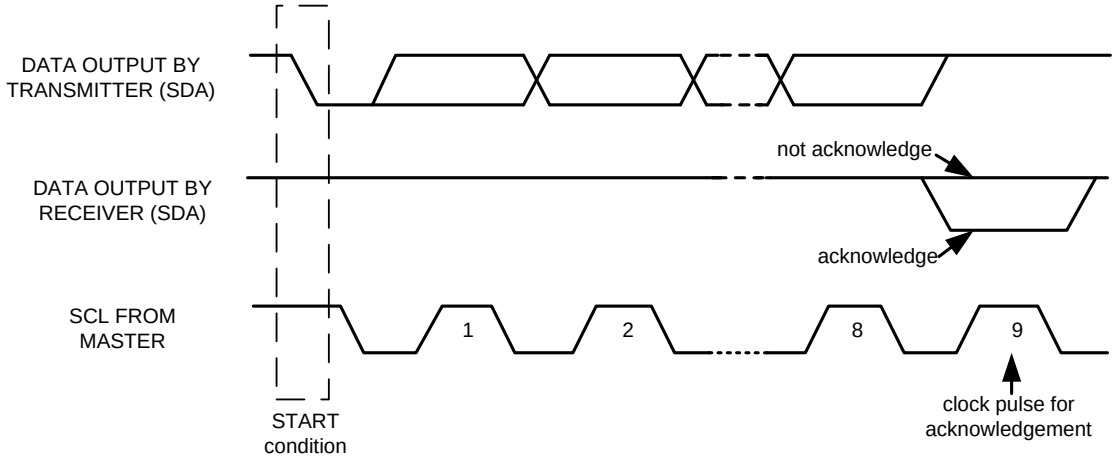


START and STOP Conditions

Data Format / Acknowledge

I²C data bytes are defined to be 8 bits long. There is no restriction to the number of bytes transmitted per data transfer. Each byte transferred must be followed by an acknowledge (ACK) signal. The clock for the acknowledge signal is generated by the master, while the receiver generates the actual acknowledge signal by pulling down SDA and holding it low during the HIGH portion of the acknowledge clock pulse.

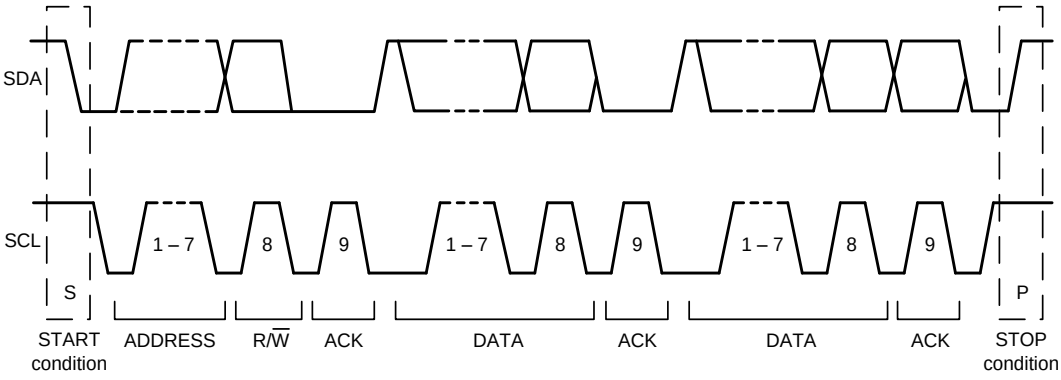
If a slave is busy and cannot transmit or receive another byte of data until some other task has been performed, it can hold SCL LOW, thus forcing the master into a wait state. Normal data transfer resumes when the slave is ready, and releases the clock line (see figure below).



Acknowledge on the I²C Bus

Communications

After beginning communications with the START condition (S), the master sends a 7-bit slave address followed by an 8th bit, the read/write bit. The read/write bit indicates whether the master is receiving data from or is writing to the slave device. Then, the master releases the SDA line and waits for the acknowledge signal (ACK) from the slave device. Each byte transferred must be followed by an acknowledge bit. To acknowledge, the slave device pulls the SDA line LOW and keeps it LOW for the high period of the SCL line. Data transmission is always terminated by the master with a STOP condition (P), thus freeing the communications line. However, the master can generate a repeated START condition (Sr), and address another slave without first generating a STOP condition (P). A LOW to HIGH transition on the SDA line while SCL is HIGH defines the stop condition. All SDA changes should take place when SCL is low, with the exception of start and stop conditions.



Complete I²C Data Transfer

To write the internal ITG-3200 device registers, the master transmits the start condition (S), followed by the I^2C address and the write bit (0). At the 9th clock cycle (when the clock is high), the ITG-3200 device acknowledges the transfer. Then the master puts the register address (RA) on the bus. After the ITG-3200 acknowledges the reception of the register address, the master puts the register data onto the bus. This is followed by the ACK signal, and data transfer may be concluded by the stop condition (P). To write multiple bytes after the last ACK signal, the master can continue outputting data rather than transmitting a stop signal. In this case, the ITG-3200 device automatically increments the register address and loads the data to the appropriate register. The following figures show single and two-byte write sequences.

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

To read the internal ITG-3200 device registers, the master first transmits the start condition (S), followed by the I^2C address and the write bit (0). At the 9th clock cycle (when clock is high), the ITG acknowledges the transfer. The master then writes the register address that is going to be read. Upon receiving the ACK signal from the ITG-3200, the master transmits a start signal followed by the slave address and read bit. As a result, the ITG-3200 sends an ACK signal and the data. The communication ends with a not acknowledge (NACK) signal and a stop bit from master. The NACK condition is defined such that the SDA line remains high at the 9th clock cycle. To read multiple bytes of data, the master can output an acknowledge signal (ACK) instead of a not acknowledge (NACK) signal. In this case, the ITG-3200 automatically increments the register address and outputs data from the appropriate register. The following figures show single and two-byte read sequences.

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

I²C Terms

Signal	Description
S	Start Condition: SDA goes from high to low while SCL is high
AD	Slave I²C address
W	Write bit (0)
R	Read bit (1)
ACK	Acknowledge: SDA line is low while the SCL line is high at the 9th clock cycle
NACK	Not-Acknowledge: SDA line stays high at the 9th clock cycle
RA	ITG-3200 internal register address
DATA	Transmit or received data
P	Stop condition: SDA going from low to high while SCL is high

7 Register Map

Addr Hex	Addr Decimal	Register Name	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	WHO_AM_I	R/W	-	ID						-
15	21	SMPLRT_DIV	R/W	SMPLRT_DIV							
16	22	DLPF_FS	R/W	-	-	-	FS_SEL		DLPF_CFG		
17	23	INT_CFG	R/W	ACTL	OPEN	LATCH_ INT_EN	INT_ ANYRD_ 2CLEAR	-	ITG_RDY _EN	-	RAW_ RDY_ EN
1A	26	INT_STATUS	R	-	-	-	-	-	ITG_RDY	-	RAW_ DATA_ RDY
1B	27	TEMP_OUT_H	R	TEMP_OUT_H							
1C	28	TEMP_OUT_L	R	TEMP_OUT_L							
1D	29	GYRO_XOUT_H	R	GYRO_XOUT_H							
1E	30	GYRO_XOUT_L	R	GYRO_XOUT_L							
1F	31	GYRO_YOUT_H	R	GYRO_YOUT_H							
20	32	GYRO_YOUT_L	R	GYRO_YOUT_L							
21	33	GYRO_ZOUT_H	R	GYRO_ZOUT_H							
22	34	GYRO_ZOUT_L	R	GYRO_ZOUT_L							
3E	62	PWR_MGM	R/W	H_RESET	SLEEP	STBY_XG	STBY_YG	STBY_ZG	CLK_SEL		

8 Register Description

This section details each register within the InvenSense ITG-3200 gyroscope. Note that any bit that is not defined should be set to zero in order to be compatible with future InvenSense devices.

The register space allows single-byte reads and writes, as well as burst reads and writes. When performing burst reads or writes, the memory pointer will increment until either (1) reading or writing is terminated by the master, or (2) the memory pointer reaches certain reserved registers between registers 33 and 60.

8.1 Register 0 – Who Am I

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	-	ID						-

Description:
This register is used to verify the identity of the device.
Parameters:
ID Contains the I^2C address of the device, which can also be changed by writing to this register.

The Power-On-Reset value of Bit6: Bit1 is 110 100.

8.2 Register 21 – Sample Rate Divider

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
15	21	SMPLRT_DIV								00h

Description:
This register determines the sample rate of the ITG-3200 gyros. The gyros outputs are sampled internally at either 1kHz or 8kHz, determined by the *DLPF_CFG* setting (see register 22). This sampling is then filtered digitally and delivered into the sensor registers after the number of cycles determined by this register. The sample rate is given by the following formula:

$F_{sample} = F_{internal} / (divider+1)$, where $F_{internal}$ is either 1kHz or 8kHz

As an example, if the internal sampling is at 1kHz, then setting this register to 7 would give the following:

$F_{sample} = 1kHz / (7 + 1) = 125Hz$, or 8ms per sample

Parameters:
SMPLRT_DIV Sample rate divider: 0 to 255

8.3 Register 22 – DLPF, Full Scale

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
16	22	-			FS_SEL		DLPF_CFG			00h

Description:

This register configures several parameters related to the sensor acquisition.

The *FS_SEL* parameter allows setting the full-scale range of the gyro sensors, as described in the table below. The power-on-reset value of *FS_SEL* is 00h. **Set to 03h for proper operation.**

FS_SEL

FS_SEL	Gyro Full-Scale Range
0	Reserved
1	Reserved
2	Reserved
3	±2000°/sec

The *DLPF_CFG* parameter sets the digital low pass filter configuration. It also determines the internal sampling rate used by the device as shown in the table below.

DLPF_CFG

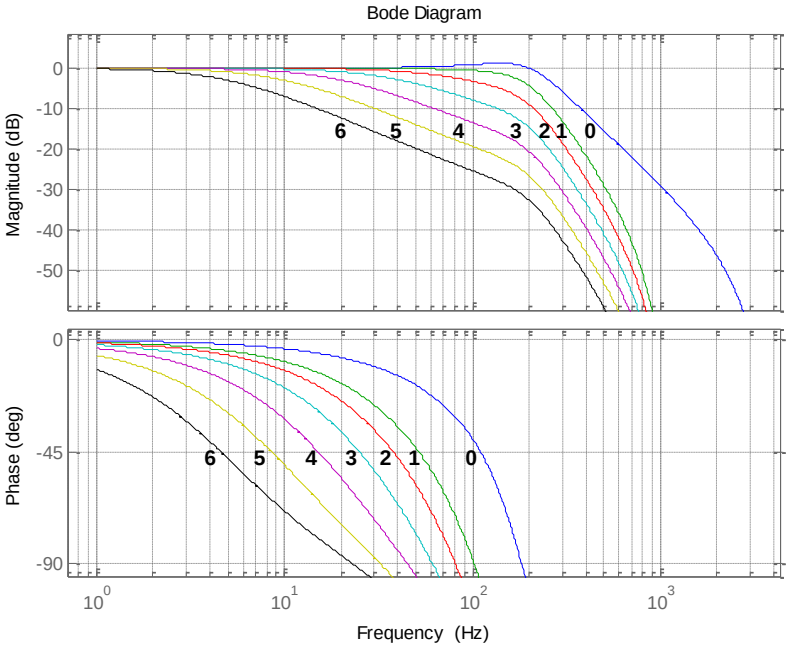
DLPF_CFG	Low Pass Filter Bandwidth	Internal Sample Rate
0	256Hz	8kHz
1	188Hz	1kHz
2	98Hz	1kHz
3	42Hz	1kHz
4	20Hz	1kHz
5	10Hz	1kHz
6	5Hz	1kHz
7	Reserved	Reserved

Parameters:

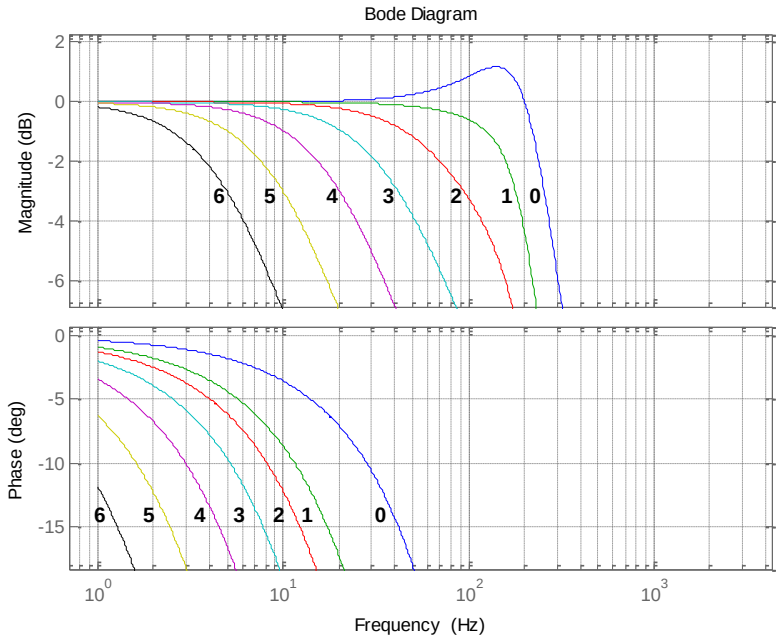
FS_SEL Full scale selection for gyro sensor data

DLPF_CFG Digital low pass filter configuration and internal sampling rate configuration

DLPF Characteristics: The gain and phase responses of the digital low pass filter settings (*DLPF_CFG*) are shown below:



Gain and Phase vs. Digital Filter Setting



Gain and Phase vs. Digital Filter Setting, Showing Passband Details

8.4 Register 23 – Interrupt Configuration

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
17	23	ACTL	OPEN	LATCH_ INT_EN	INT_ ANYRD_ 2CLEAR	0	ITG_RDY_ EN	0	RAW_ RDY_ EN	00h

Description:

This register configures the interrupt operation of the device. The interrupt output pin (INT) configuration can be set, the interrupt latching/clearing method can be set, and the triggers for the interrupt can be set.

Note that if the application requires reading every sample of data from the ITG-3200 part, it is best to enable the raw data ready interrupt (*RAW_RDY_EN*). This allows the application to know when new sample data is available.

Parameters:

ACTL	Logic level for INT output pin – 1=active low, 0=active high
OPEN	Drive type for INT output pin – 1=open drain, 0=push-pull
LATCH_INT_EN	Latch mode – 1=latch until interrupt is cleared, 0=50us pulse
INT_ANYRD_2CLEAR	Latch clear method – 1=any register read, 0=status register read only
ITG_RDY_EN	Enable interrupt when device is ready (PLL ready after changing clock source)
RAW_RDY_EN	Enable interrupt when data is available
0	Load zeros into Bits 1 and 3 of the Interrupt Configuration register.

8.5 Register 26 – Interrupt Status

Type: Read only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
1A	26	-	-	-	-	-	ITG_RDY	-	RAW_ DATA_ RDY	00h

Description:

This register is used to determine the status of the ITG-3200 interrupts. Whenever one of the interrupt sources is triggered, the corresponding bit will be set. The polarity of the interrupt pin (active high/low) and the latch type (pulse or latch) has no affect on these status bits.

Use the Interrupt Configuration register (23) to enable the interrupt triggers. If the interrupt is not enabled, the associated status bit will not get set.

In normal use, the *RAW_DATA_RDY* interrupt is used to determine when new sensor data is available in either the sensor registers (27 to 32).

Interrupt Status bits get cleared as determined by INT_ANYRD_2CLEAR in the interrupt configuration register (23).

Parameters:

ITG_RDY	PLL ready
RAW_DATA_RDY	Raw data is ready

8.6 Registers 27 to 34 – Sensor Registers

Type: Read only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	TEMP_OUT_H							
1C	28	TEMP_OUT_L							
1D	29	GYRO_XOUT_H							
1E	30	GYRO_XOUT_L							
1F	31	GYRO_YOUT_H							
20	32	GYRO_YOUT_L							
21	33	GYRO_ZOUT_H							
22	34	GYRO_ZOUT_L							

Description:
These registers contain the gyro and temperature sensor data for the ITG-3200 parts. At any time, these values can be read from the device; however it is best to use the interrupt function to determine when new data is available.

Parameters:
TEMP_OUT_H/L 16-bit temperature data (2's complement format)
GYRO_XOUT_H/L 16-bit X gyro output data (2's complement format)
GYRO_YOUT_H/L 16-bit Y gyro output data (2's complement format)
GYRO_ZOUT_H/L 16-bit Y gyro output data (2's complement format)

8.7 Register 62 – Power Management

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
3E	62	H_RESET	SLEEP	STBY _XG	STBY _YG	STBY _ZG	CLK_SEL			00h

Description:
This register is used to manage the power control, select the clock source, and to issue a master reset to the device.

Setting the *SLEEP* bit in the register puts the device into very low power sleep mode. In this mode, only the serial interface and internal registers remain active, allowing for a very low standby current. Clearing this bit puts the device back into normal mode. To save power, the individual standby selections for each of the gyros should be used if any gyro axis is not used by the application.

The *CLK_SEL* setting determines the device clock source as follows:

CLK_SEL

CLK_SEL	Clock Source
0	Internal oscillator
1	PLL with X Gyro reference
2	PLL with Y Gyro reference
3	PLL with Z Gyro reference
4	PLL with external 32.768kHz reference
5	PLL with external 19.2MHz reference
6	Reserved
7	Reserved

On power up, the ITG-3200 defaults to the internal oscillator. It is highly recommended that the device is configured to use one of the gyros (or an external clock) as the clock reference, due to the improved stability.

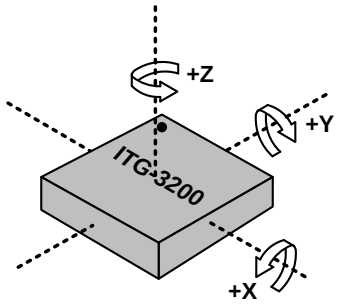
Parameters:

H_RESET	Reset device and internal registers to the power-up-default settings
SLEEP	Enable low power sleep mode
STBY_XG	Put gyro X in standby mode (1=standby, 0=normal)
STBY_YG	Put gyro Y in standby mode (1=standby, 0=normal)
STBY_ZG	Put gyro Z in standby mode (1=standby, 0=normal)
CLK_SEL	Select device clock source

9 Assembly

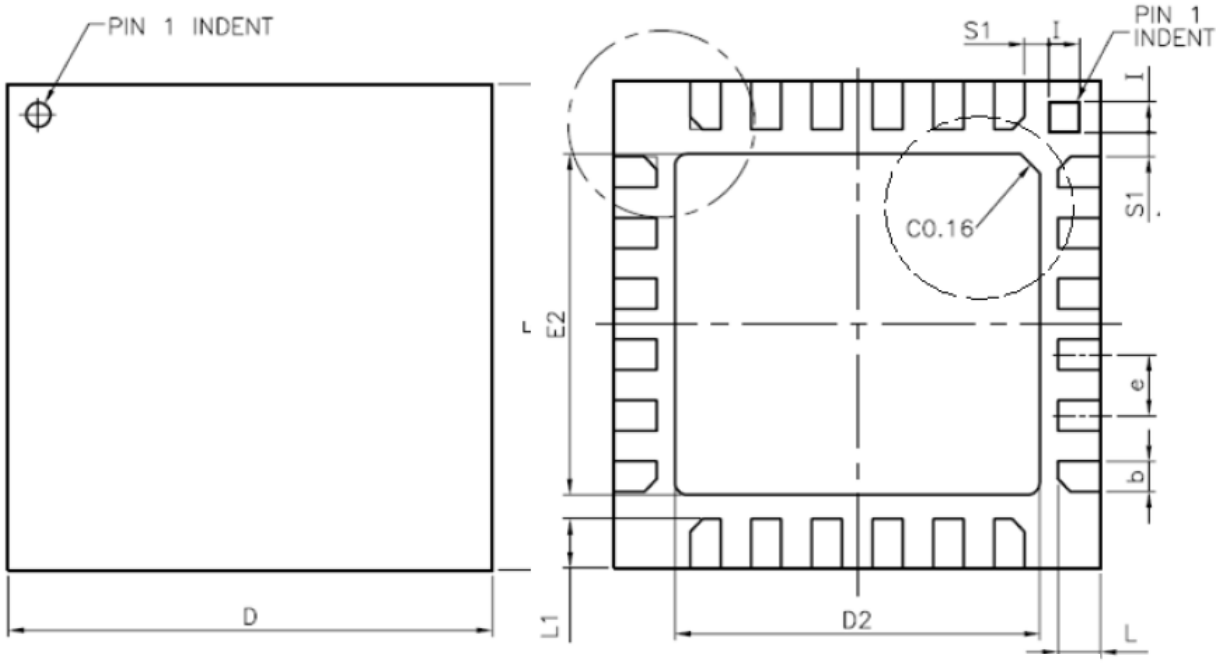
9.1 Orientation

The diagram below shows the orientation of the axes of sensitivity and the polarity of rotation.



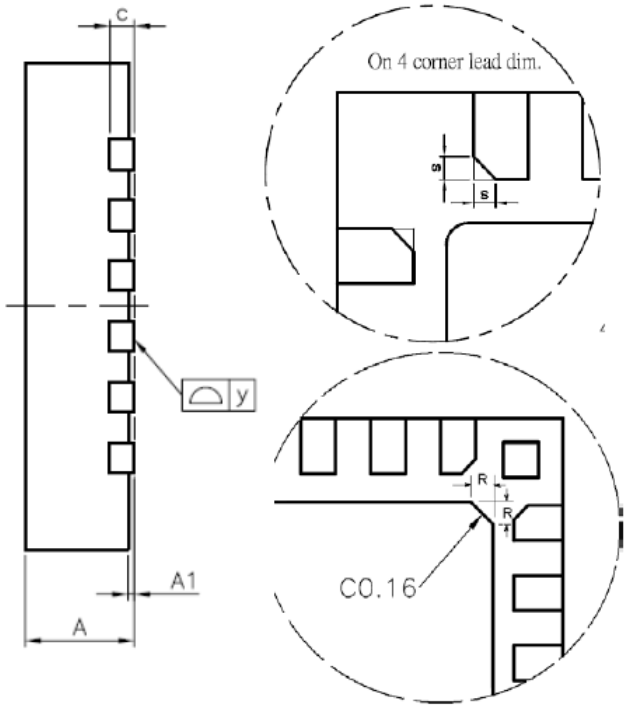
Orientation of Axes of Sensitivity and Polarity of Rotation

9.2 Package Dimensions



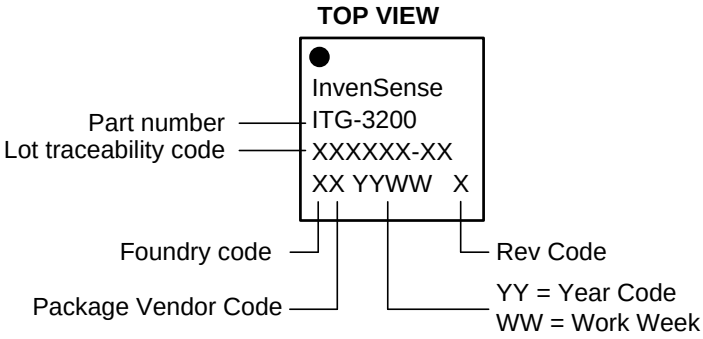
Top View **Bottom View**

SYMBOLS	DIMENSIONS IN MILLIMETERS		
	MIN	NOM	MAX
A	0.85	0.90	0.95
A1	0.00	0.02	0.05
b	0.18	0.25	0.30
c	---	0.20 REF.	---
D	3.90	4.00	4.10
D2	2.95	3.00	3.05
E	3.90	4.00	4.10
E2	2.75	2.80	2.85
e	---	0.50	---
L	0.30	0.35	0.40
L1	0.35	0.40	0.45
I	0.20	0.25	0.30
R	0.085	---	0.145
s	0.05	---	0.15
S1	0.15	0.20	0.25
y	0.00	---	0.075



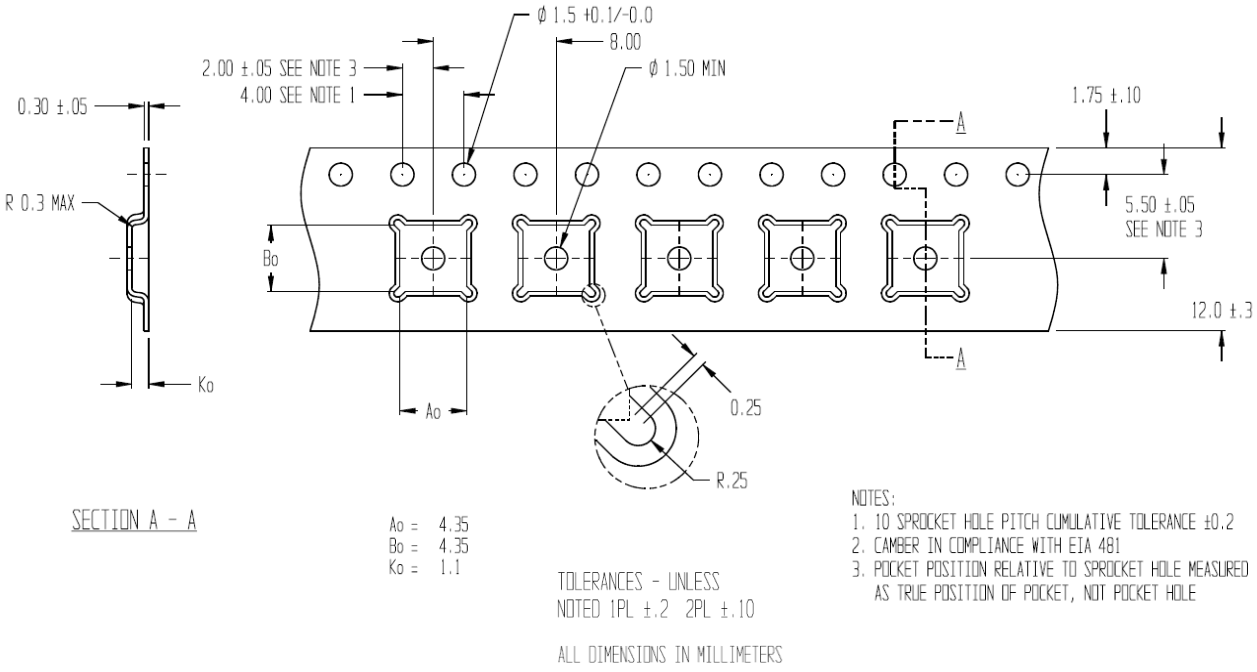
Package Dimensions

9.3 Package Marking Specification

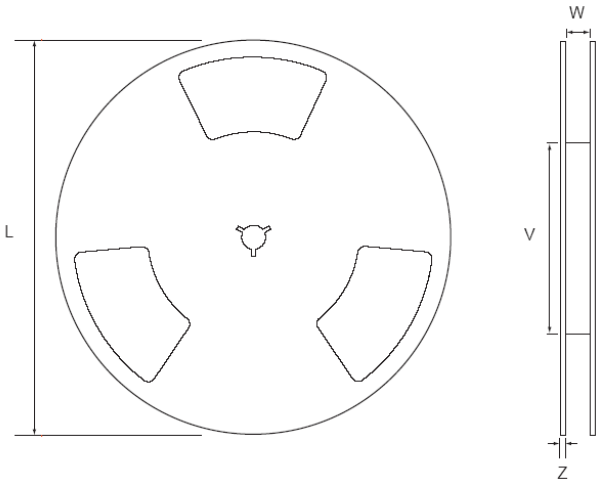


Package Marking Specification

9.4 Tape & Reel Specification



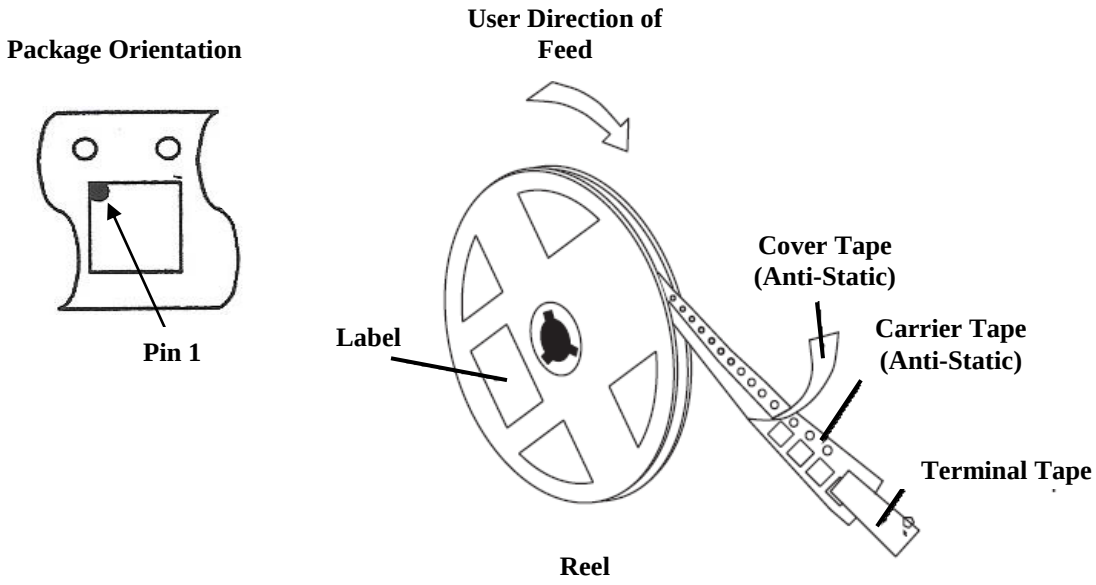
Tape Dimensions



Reel Outline Drawing

Reel Dimensions and Package Size

PKG SIZE	REEL (mm)			
	L	V	W	Z
4x4	330	100	13.2	2.2

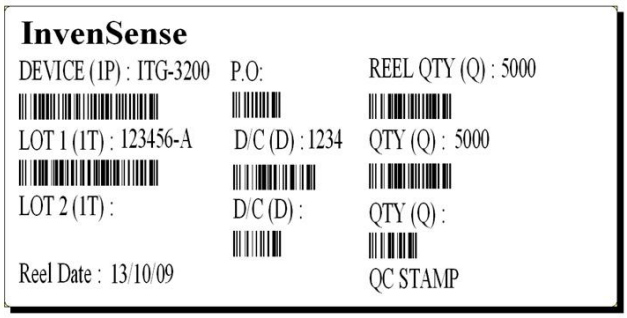


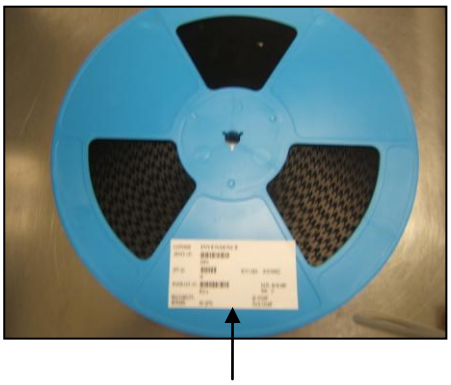
Tape and Reel Specification

Reel Specifications

Quantity Per Reel	5,000
Reels per Box	1
Boxes Per Carton (max)	3 full pizza boxes packed in the center of the carton, buffered by two empty pizza boxes (front and back).
Pcs/Carton (max)	15,000

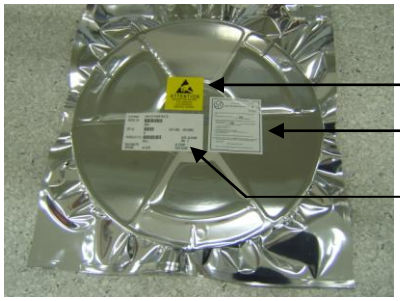
9.5 Label



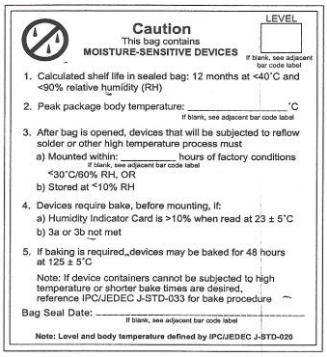


Location of Label

9.6 Packaging



Moisture Barrier Bag With Labels



Moisture-Sensitive Caution Label

Reel in Box

Box with Tape & Reel Label

9.7 Soldering Exposed Die Pad

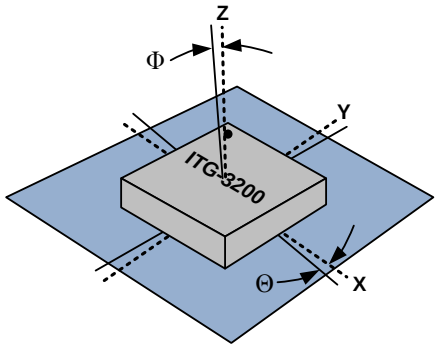
The ITG-3200 has very low active and standby current consumption. The exposed die pad is not required for heat sinking, and should not be soldered to the PCB since soldering to it contributes to performance changes due to package thermo-mechanical stress.

9.8 Component Placement

Testing indicates that there are no specific design considerations other than generally accepted industry design practices for component placement near the ITG-3200 multi-axis gyroscope to prevent noise coupling, and thermo-mechanical stress.

9.9 PCB Mounting and Cross-Axis Sensitivity

Orientation errors of the gyroscope mounted to the printed circuit board can cause cross-axis sensitivity in which one gyro responds to rotation about another axis, for example, the X-axis gyroscope responding to rotation about the Y or Z axes. The orientation mounting errors are illustrated in the figure below.



Package Gyro Axes (·····) Relative to PCB Axes (——) with Orientation Errors (Θ and Φ)

The table below shows the cross-axis sensitivity as a percentage of the specified gyroscope's sensitivity for a given orientation error.

Cross-Axis Sensitivity vs. Orientation Error

Orientation Error (θ or Φ)	Cross-Axis Sensitivity (sinθ or sinΦ)
0º	0%
0.5º	0.87%
1º	1.75%

The specification for cross-axis sensitivity in Section 3 includes the effect of the die orientation error with respect to the package.

9.10 MEMS Handling Instructions

MEMS (Micro Electro-Mechanical Systems) are a time-proven, robust technology used in hundreds of millions of consumer, automotive and industrial products. MEMS devices consist of microscopic moving mechanical structures. They differ from conventional IC products even though they can be found in similar packages. Therefore, MEMS devices require different handling precautions than conventional ICs prior to mounting onto printed circuit boards (PCBs).

The ITG-3200 gyroscope has a shock tolerance of 10,000*g*. InvenSense packages its gyroscopes as it deems proper for protection against normal handling and shipping. It recommends the following handling precautions to prevent potential damage.

- Individually packaged or trays of gyroscopes should not be dropped onto hard surfaces. Components placed in trays could be subject to *g*-forces in excess of 10,000*g* if dropped.
- Printed circuit boards that incorporate mounted gyroscopes should not be separated by manually snapping apart. This could also create *g*-forces in excess of 10,000*g*.

9.11 Gyroscope Surface Mount Guidelines

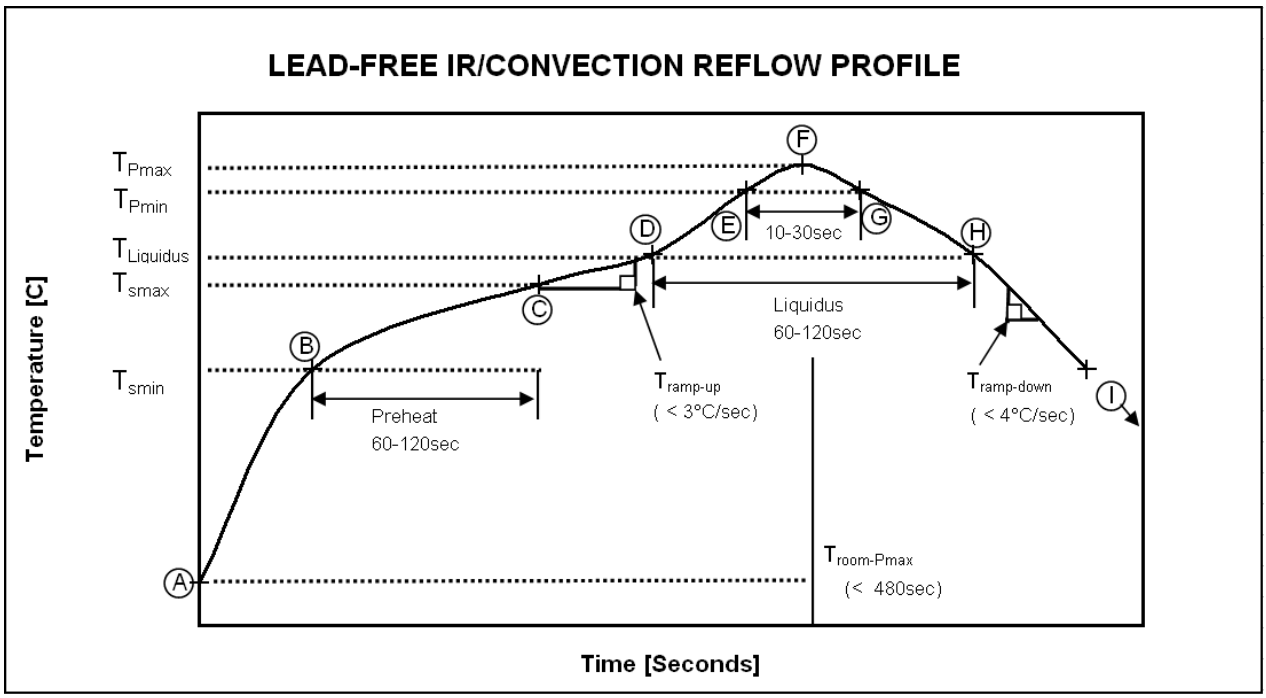
Any material used in the surface mount assembly process of the MEMS gyroscope should be free of restricted RoHS elements or compounds. Pb-free solders should be used for assembly.

In order to assure gyroscope performance, several industry standard guidelines need to be considered for surface mounting. These guidelines are for both printed circuit board (PCB) design and surface mount assembly and are available from packaging and assembly houses.

When using MEMS gyroscope components in plastic packages, package stress due to PCB mounting and assembly could affect the output offset and its value over a wide range of temperatures. This is caused by the mismatch between the Coefficient Temperature Expansion (CTE) of the package material and the PCB. Care must be taken to avoid package stress due to mounting.

9.12 Reflow Specification

The approved solder reflow curve shown in the figure below conforms to IPC/JEDEC J-STD-020D.01 (Moisture/Reflow Sensitivity Classification for Nonhermetic Solid State Surface Mount Devices) with a maximum peak temperature (Tc = 260°C). This is specified for component-supplier reliability qualification testing using lead-free solder for package thicknesses less than 1.6 mm. The reliability qualification pre-conditioning used by InvenSense incorporates three of these conforming reflow cycles. All temperatures refer to the topside of the QFN package, as measured on the package body surface. Customer solder-reflow processes should use the solder manufacturer's recommendations, making sure to never exceed the constraints listed in the table and figure below, as these represent the maximum tolerable ratings for the device. For optimum results, production solder reflow processes should use lower temperatures, reduced exposure times to high temperatures, and lower ramp-up and ramp-down rates than those listed below.



Approved IR/Convection Solder Reflow Curve

Temperature Set Points for IR / Convection Reflow Corresponding to Figure Above

Step	Setting	CONSTRAINTS		
		Temp (°C)	Time (sec)	Rate (°C/sec)
A	T_{room}	25		
B	T_{Smin}	150		
C	T_{Smax}	200	$60 < t_{BC} < 120$	
D	$T_{Liquidus}$	217		$r_{(TLiquidus-TPmax)} < 3$
E	T_{Pmin} [< TPmax-5°C, 250°C]	255		$r_{(TLiquidus-TPmax)} < 3$
F	T_{Pmax} [< TPmax, 260°C]	260	$t_{AF} < 480$	$r_{(TLiquidus-TPmax)} < 3$
G	T_{Pmin} [< TPmax-5°C, 250°C]	255	$t_{EG} < 30$	$r_{(TPmax-TLiquidus)} < 4$
H	$T_{Liquidus}$	217	$60 < t_{DH} < 120$	
I	T_{room}	25		

9.13 Storage Specifications

The storage specification of the ITG-3200 gyroscope conforms to IPC/JEDEC J-STD-020C Moisture Sensitivity Level (MSL) 3.

Storage Specifications for ITG-3200

Calculated shelf-life in moisture-sealed bag	12 months -- Storage conditions: <40°C and <90% RH
After opening moisture-sealed bag	168 hours -- Storage conditions: ambient ≤30°C at 60% RH

10 Reliability

10.1 Qualification Test Policy

InvenSense's products complete a Qualification Test Plan before being released to production. The Qualification Test Plan follows the JEDEC 47D Standards, "Stress-Test-Driven Qualification of Integrated Circuits," with the individual tests described below.

10.2 Qualification Test Plan

Accelerated Life Tests

TEST	Method/Condition	Lot Quantity	Sample / Lot	Acc / Reject Criteria
High Temperature Operating Life (HTOL/LFR)	JEDEC JESD22-A108C, Dynamic, 3.63V biased, Tj>125°C [read-points 168, 500, 1000 hours]	3	77	(0/1)
Steady-State Temperature Humidity Bias Life (1)	JEDEC JESD22-A101C, 85°C/85%RH [read-points 168, 500 hours], Information Only 1000 hours]	3	77	(0/1)
High Temperature Storage Life	JEDEC JESD22-A103C, Cond. A, 125°C Non-Bias Bake [read-points 168, 500, 1000 hours]	3	77	(0/1)

Device Component Level Tests

TEST	Method/Condition	Lot Quantity	Sample / Lot	Acc / Reject Criteria
ESD-HBM	JEDEC JESD22-A114F, Class 2 (1.5KV)	1	3	(0/1)
ESD-MM	JEDEC JESD22-A115-A, Class B (200V)	1	3	(0/1)
Latch Up	JEDEC JESD78B Level 2, 125C, +/- 100mA	1	6	(0/1)
Mechanical Shock	JEDEC JESD22-B104C, Mil-Std-883, method 2002, Cond. D, 10,000*g's*, 0.3ms, ±X,Y,Z – 6 directions, 5 times/direction	3	5	(0/1)
Vibration	JEDEC JESD22-B103B, Variable Frequency (random), Cond. B, 5-500Hz, X,Y,Z – 4 times/direction	3	5	(0/1)
Temperature Cycling (1)	JEDEC JESD22-A104D Condition N, -40°C to +85°C, Soak Mode 2, 100 cycles	3	77	(0/1)

Board Level Tests

TEST	Method/Condition/	Lot Quantity	Sample / Lot	Acc / Reject Criteria
Board Mechanical Shock	JEDEC JESD22-B104C,Mil-Std-883, method 2002, Cond. D, 10,000*g's*, 0.3ms, +-X,Y,Z – 6 directions, 5 times/direction	1	5	(0/1)
Board T/C	JEDEC JESD22-A104D Condition N, -40°C to +85°C, Soak Mode 2, 100 cycles	1	40	(0/1)

(1) – Tests are preceded by MSL3 Preconditioning in accordance with JEDEC JESD22-A113F

BIBLIOGRAFÍA

ARDUINO Y PERIFÉRICOS:

"Arduino. Curso práctico de formación". Oscar Torrente Artero (Ed. RC Libros)

http://www.arduino.cc

http://chionophilous.wordpress.com/2012/02/10/connecting-to-sparkfuns-9dof-sensor-stick-i2c-access-to-adxl345-itg-3200-and-hmc5843/

MATLAB:

http://www.mathworks.com

http://www.matlabarduino.org/

http://geekytheory.com/matlab-arduino-serial-port-communication/

OTROS LINKS CONSULTADOS:

http://www.patagoniatecnology.com/producto/arduino-nano-r3/

http://www.fact4ward.com/blog/ic-if/itg3200/

https://learn.adafruit.com/adxl345-digital-accelerometer/programming

http://www.den-uijl.nl/electronics/gyro.html

http://www.ijicic.org/sss10-15.pdf

http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/