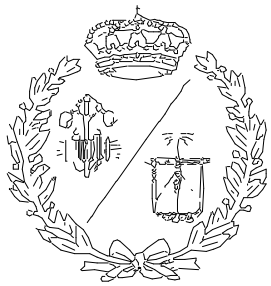
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Proyecto Fin de Máster

FUSIÓN SENSORIAL PARA LA ESTIMACIÓN DEL ESTADO DE UN VEHÍCULO AUTÓNOMO

(Sensory fusion for the state estimation of an autonomous vehicle)

Para acceder al Título de

MÁSTER UNIVERSITARIO EN INGENIERIA INDUSTRIAL

Autor: Daniel López Montes

Julio-2016

INDICE GENERAL

Índice de figuras

Índice de tablas

CAPÍTULO 1: INTRODUCCIÓN

1.1 FINALIDAD DEL PROYECTO

El objetivo del proyecto es recoger y procesar información de sensores para conseguir el funcionamiento autónomo de un vehículo eléctrico, dotando de información útil a un ordenador de abordo que controlará el vehículo. Este ordenador necesitará información de varios tipos, en el caso de este proyecto se tratará la información obtenida de sensores inerciales y un receptor GPS, dando información relativa al movimiento del vehículo da tiempo real.

En cuanto a los datos que se un vehículo autónomo necesita, una parte de ellos es información externa al vehículo, y tiene que ver con las condiciones o el entorno que lo rodea, como los obstáculos con los que se pueda encontrar, esta información es totalmente necesaria para que el ordenador pueda trazar una ruta segura. Para conseguir esta información se pueden utilizar muchos tipos de sensores, como por ejemplo cámaras o sensores de ultrasonidos.

También es necesario que el controlador trabaje con información relativa al movimiento del vehículo en si, como la orientación, la velocidad, o la posición. Hay 2 formas complementarias de conseguir esto, una es utilizando datos del funcionamiento del vehículo, como pueden ser la velocidad angular o el giro de las ruedas. Esto puede acarrear error debido al deslizamiento de los neumáticos o fallos en los sensores.

Otra forma de obtener información sobre el movimiento del vehículo es utilizando sensores inerciales (acelerómetros y giróscopos), que pueden medir la aceleración o el giro absolutos del vehículo, esta opción puede ser combinada con la anterior para tener mejores y más fiables resultados.

Figura 1.1: Vehículo autónomo

En la figura 1.1 se puede ver un vehículo autónomo que dispone de un sistema de sensores que utiliza para recoger información.

Este método de los sensores inerciales es más utilizado en tecnologías para dispositivos voladores, como pueden ser los drones, ya que estos carecen de la posibilidad de utilizar otro tipo de sensores que un coche por ejemplo si puede. Además el hecho de utilizar información de las 3 dimensiones del espacio es mucho más importante en el caso de un objeto volador que en un vehículo terrestre, en el cual se podría conseguir buenos resultados utilizando únicamente 2 ejes, suponiendo que no se desplace por terrenos muy inclinados.

Además de los datos obtenidos de sensores inerciales también se utilizarán un magnetómetro (contenido en un mismo dispositivo junto al acelerómetro y el giróscopo), y un módulo GPS para conocer información absoluta de la orientación y la posición del vehículo, estos datos también se tratarán de fusionar con el resto para tener una precisión mayor en los resultados.

Para recopilar y procesar esta información se utilizará Arduino, un microprocesador programable, el cual se conectará con los sensores y con el ordenador de abordo que ejecuta el control del vehículo.

La comunicación entre Arduino y el ordenador de abordo será mediante cable Ethernet, debido a que se prevé que los puertos USB del ordenador puedan estar ocupados por otras conectividades necesarias para el control del vehículo.

El objetivo es poder conseguir un seguimiento preciso a tiempo real del estado del vehículo, para poder aplicarlo al control que realizará el ordenador de a bordo, permitiendo una conducción autónoma precisa y segura, ya que con la fusión sensorial el funcionamiento no se basa únicamente en un único sensor, que puede fallar, sino que se combinan todos.

1.2 VEHÍCULOS AUTÓNOMOS

La investigación de los vehículos autónomos en los últimos años ha conseguido que ya existan muchos modelos diferentes desarrollados por distintas empresas o universidades, algunos de ellos incluso han sido certificados para poder circular legalmente en ciertas regiones donde la normativa ya lo permite. Para llegar a esto se ha tenido que recorrer un largo camino con muchos proyectos y pruebas.

La primera presentación de un vehículo autónomo fue la realizada por Norman Bel Geddes en la feria de muestras "Futurama", en la presentación de 1939 de la exposición universal, esta presentación consistía en un vehículo eléctrico controlado por un circuito eléctrico en el pavimento de la carretera.

Posteriormente, en 1980, una furgoneta Mercedes-Benz, diseñada por la universidad de Múnich, alcanzo los 100km/h sin tráfico. Gracias a esta demostración la Comisión Europea invirtió 800 millones de euros con el objetivo de desarrollar un vehículo autónomo.

En la figura 1.2 se muestras uno de los primeros experimentos para la creación de un vehículo autónomo.

Figura 1.2: Primeras pruebas de vehículos autónomos

En el mismo año, la DARPA construyó el primer vehículo que funcionaba mediante visión computarizada y radar laser. En 1987, se demostró que era posible diseñar un vehículo que creaba su propia ruta, consiguiendo desplazarse más de 600 metros a través de un terreno complejo con obstáculos.

Años más tarde, en 1994, dos vehículos robots VaMP y Vita-2, circularon de forma prácticamente autónoma a lo largo de más de mil kilómetros en una autovía de Paris con 3 carriles con tráfico a velocidades de hasta 130km/h. Demostraron que la circulación en carriles libres, en convoy e incluso realizando cambios de carril, podían realizarse de forma autónoma.

Un año más tarde, en 1995, el equipo de Dickmanns modifico un Mercedes para que hiciera un trayecto entre Múnich y Copenhague ida y vuelta, utilizando una visión computarizada y un ordenador que utilizaba datos a tiempo real, alcanzando velocidades de más de 175km/h, con pequeñas intervenciones humanas. En este caso la conducción autónoma supuso un 95% del tiempo de circulación, consiguiendo realizar maniobras de adelantamiento a otros vehículos.

En España también se han realizado experimentos con vehículos autónomos circulando por las carreteras, diversas universidades tienen programas para tratar de conseguirlo. En la universidad de Cantabria se

trabaja para conseguirlo, y este proyecto trata de ser un paso adelante para cumplir con este objetivo.

El desarrollo de estos vehículos ha sido realizado principalmente por las mayores empresas fabricantes de vehículos, universidades, y otras empresas del sector tecnológico. Las cuales han colaborado en muchos casos entre ellas para conseguir evolucionar más rápido, utilizando los conocimientos que cada una posee en su campo.

Google ha sido y es una de las principales empresas desarrolladoras de vehículos autónomos, dispone de un proyecto muy importante el cual lleva años llevándolo a cabo que ha dado muy buenos resultados. Tienen vehículos certificados por las autoridades para poder circular autónomamente donde la ley lo permite, siendo los primeros en conseguir circular por los Estados Unidos con tráfico normal.

Figura 1.3: Uno de los vehículos autónomos de Google

En la figura 1.3 se puede ver el vehículo que Google pretende sacar al mercado en los próximos años.

En la actualidad existen vehículos autónomos muy complejos que pueden circular sin peligro por las carreteras, y cuyo mayor problema es que la legislación lo permita.

Estados Unidos es el país donde más avances se han hecho al respecto y donde la legislación da más posibilidades de pruebas en situaciones reales a las empresas fabricantes, hay varios estados como California, Nevada o Florida en los cuales está permitida la circulación de vehículos autónomo por la vía pública. En Europa sin embargo es ilegal la circulación normal de un vehículo sin conductor, aunque si se han hecho excepciones para la realización de pruebas o demostraciones controladas y autorizadas por los organismos competentes.

La seguridad en la conducción siempre ha sido el factor que más ha afectado al desarrollo de los vehículos autónomos, las dudas de la sociedad en general en cuanto a la capacidad de estos para circular de forma segura ha hecho que los vehículos hayan tenido que superar diferentes y exigentes pruebas de seguridad. Sin embargo cuando se han probado en la vía pública con circulación normal, la inmensa mayoría de los accidentes en los que se ha visto envuelto un vehículo autónomo han sido provocados por un error humano, ya sea por un ocupante de este, o el conductor de otro vehículo.

Algunas características de los vehículos autónomos ya han sido llevadas a la práctica en vehículos disponibles en el mercado desde hace tiempo, por ejemplo la capacidad de auto estacionamiento que permite a los vehículos realizar la maniobra de estacionamiento sin que el conductor toque el volante.

Los vehículos autónomos consiguen conocer el entorno mediante técnicas complejas como el láser, el radar, el lidar, el GPS, o la visión computerizada. Sistemas avanzados de control fusionan e interpretan toda la información que les llega para identificar una ruta segura. Esto vehículos son capaces de crear mapas propios si se encuentran en ternos en donde no tienen cartográfica disponible y deben construir sus propias rutas.

Además los vehículos autónomos son capaces de leer e interpretar las señales de tráfico como pueden ser los límites de velocidad o los semáforos.

La figura 1.4 muestra una representación que sirve para hacerse una idea de cómo puede funcionar el sistema de control de un vehículo autónomo.

Figura 1.4: Representación ilustrativa del funcionamiento de un vehículo autónomo

Según previsiones de las empresas involucradas en el desarrollo de estas tecnologías, en un futuro relativamente cercano, el tráfico estará dominado por vehículos autónomos, ya que tendrán importantes ventajas sobre los vehículos convencionales, algunas de estas ventajas son las siguientes:

- Aumento de la seguridad vial, debido a la reducción de las posibilidades accidente debido a un error humano.
- Accesibilidad a las personas que no pueden conducir.
- Mejora del tráfico.
- Aumento de la eficiencia energética, ya que los vehículos utilizarán algoritmos para realizar una conducción eficiente minimizando el gasto energético.
- Eficiencia económica, reducción de gastos debido a la disminución del consumo energético, reducción de gastos en multas y seguros, y reducción de gastos en señalización de las vías.

1.3 VEHÍCULO EXPERIMENTAL

El vehículo al cual se pretende implantar el sistema de control es un pequeño coche eléctrico biplaza, de un tamaño y aspecto similar a un carrito de golf, y que es denominado cuadriciclo. No se trata de un vehículo comercial, sino que es un vehículo experimental utilizado para investigaciones o proyectos.

El fabricante del vehículo es la empresa catalana "Teycars", y el modelo es el ALSO2 de 2008.

Este coche es de uso urbano, ya que no llega a alcanzar los 50 km/h, lo cual condicionará este proyecto. La autonomía de la que dispone es de más o menos 90 km, lo cual lo hace inviable para realizar viajes largos. La figura 1.5 muestra una fotografía de este vehículo.

Para realizar el control del vehículo se utilizará un ordenador a bordo que recibirá datos de todos los sensores de los que dispongan y tras procesarlos, enviará señales a todos los actuadores necesarios para controlar la velocidad y el giro del vehículo.

En la tabla 1.1 se pueden ver algunas características adicionales del vehículo.

Figura 1.5: Vehículo experimental

Peso Vacío	600 Kg
Peso Máximo	950 Kg
Largo x Alto x Ancho	2740 x 1490 x 1950 mm
Altura al suelo	180 mm
Ancho de ejes (Delantero/Trasero)	1280 / 1260 mm
Distancia entre ejes	1900 mm
Motor	4.3 KW, 48 V
Batería	210 AH, 6V
Frenos Delanteros	Disco
Frenos Traseros	Tambor
Distancia de frenado	≤ 4 m
Tiempo de carga de la batería	8-10 Horas
Autonomía	80-90 Km
Velocidad Máxima	45-50 Km/h
Neumáticos	Kebler 165/70R13

Tabla 1.1: Características del vehículo experimental

Conocer las características de este vehículo permite simplificar el sistema de control que se realizará, ya que contemplar la reducida velocidad máxima de este implicará por ejemplo que la frecuencia de muestreo no deba que ser tan alta como en el caso de un dron o una avioneta, manteniéndose la calidad del control. Los giros serán relativamente lentos, por lo que se reduce el error de las mediciones de los sensores. También podremos realizar otras simplificaciones como se verán en apartados posteriores.

1.4 ORDENADOR DE ABORDO

Para recoger la información sensorial y aplicar el control del vehículo se utilizará un ordenador en el interior del vehículo, por este ordenador deberá pasar todos los datos recogidos, y con ellos el ordenador realizará los cálculos necesarios para obtener las señales necesarias para realizar el control del vehículo.

Este ordenador no es un ordenador convencional de los utilizados para uso personal, sino que es de tipo industrial compacto ("PC BOX FANLESS") en concreto el NISE 3500P2. Es de menor tamaño que un ordenador normal, ya que por su utilidad es importante que sea pequeño, sin embargo tiene una gran potencia de procesamiento. Está dotado de múltiples conectores USB y LAN para comunicarse con dispositivos externos. En las figuras 1.6 y 1.7 se ven unas fotografías del ordenador.

Figura 1.6: Ordenador de abordo NISA 3500P

Tiene unas dimensiones de 195 mm de ancho, 268 mm de largo, y 101 mm de alto (7,7" x 10,5" x 3,98"), y se ha fabricado con un chasis de aluminio, este chasis tiene mejor resistencia a los golpes que un ordenador de mesa normal. Para reducir su tamaño se ha diseñado sin ventilador interno, lo cual consigue su propósito de reducir las dimensiones y el peso, pero podría provocar problemas de sobrecalentamiento si se utiliza a pleno rendimiento en un ambiente cerrado. Tal y como se puede ver en las imágenes, dispone de una base para poder fijarlo a una superficie mediante unos pequeños tornillos.

Puede ser utilizado para multitud de usos y aplicaciones, incluido como un ordenador normal de uso personal, ya que se le puede instalar un sistema operativo como el Windows. Sin embargo está orientado a su utilización en aplicaciones especiales como por ejemplo las siguientes: equipos de diagnóstico médico, tratamiento de imágenes médicas, almacenamiento de datos, automatización industrial, aplicaciones públicas de información y entretenimiento, seguridad de vigilancia.

Figura 1.7: Conexiones disponibles del NISE 3500P

Como un ordenador normal también puede ser conectado a dispositivos de control para manejarlo, como pueden ser un ratón, un teclado, o una pantalla.

A continuación se enumeran las características técnicas de las que dispone según la página web del vendedor, en el anexo donde se encuentran las Datasheet se incluye uno de este dispositivo con más información.

SISTEMA:

- Procesador: Intel i7-620M 2,66GHz, Intel i5-520M 2,4GHz, Intel P4500 1,86GHz
- Memoria RAM: 2 x 240pin DIMM DDR3 (hasta 4 GB)
- Almacenamiento: 1 x SATA 2,5" HDD, 2 x eSATA

INTERFACES:

- PS2: 1 x PS2 (teclado y ratón)
- USB: 6 x USB 2.0
- LAN: 2 x 10/100/1000 Ethernet
- COM: 1 x DB44 para 4x RS232 (COM2: RS232/422/485), 1 x DB9 para RS232
- Display: 1 x VGA, 1 x DVI-I
- Audio: 1 x Line-out, 1x Mic-in
- Expansión: 2 x PCI
- GPIO: opcional 4 x Input, 4 x Output (COM5)

OTROS:

- Certificaciones: CE approval, FCC Class B, UL/ cUL, e13
- Alimentación: 9V ~ 30V DC
- Temp. de trabajo: -5°C ~ 55°C (airflow)
- Humedad relativa: 10% to 93% sin condensación
- Dimensiones: 195 x 268 x 80 mm

Para manejar el ordenador desde el interior del vehículo también se dispone de una pequeña pantalla táctil de 7 pulgadas similar a la mostrada en la figura 1.8. Esta pantalla se conectará al ordenador y realizará tanto la visualización de la información del ordenador, como el manejo y la introducción de datos a este.

Figura 1.8: Pantalla táctil

CAPÍTULO 2: ARDUINO

Arduino es un producto de hardware libre que consiste en una placa electrónica con un microprocesador y un entorno de desarrollo propio. Una de sus principales características que lo han llevado a ser el principal competidor en su mercado es su facilidad de uso, tanto en hardware como con software, ya que al poco tiempo de adquirir un Arduino puedes crear un programa sencillo sin necesidad de tener grandes conocimientos de electrónica ni de programación.

Uno de los Arduinos de uso más típico es el Arduino UNO, que se puede ver en la figura 2.1.

Figura 2.1: Arduino UNO

Arduino dispone de un conjunto de puertos tanto digitales como analógicos con entradas y salidas, que se utilizan para conectarlo con otros dispositivos, tales como circuitos eléctricos, LEDs, sensores, etc… Lo cual permite realizar una infinidad de proyectos diferentes. Además Arduino se puede conectar a otras placas de expansión o "shields" que amplían las posibilidades de Arduino y que se comentarán posteriormente.

En internet existe una gran comunidad de usuarios que comparten sus programas y sus problemas con el resto del mundo, haciendo que todos puedan progresar con ayuda del resto, aportando ideas y soluciones. El hecho de que Arduino sea una plataforma "Open-Source" es una de las principales características y ventajas de Arduino, motivando su uso respecto a otros dispositivos similares en el mercado.

Para cargar un programa en Arduino es necesario utilizar el entorno de desarrollo integrado de Arduino, el "Arduino IDE", un programa de PC en el cual podrás crear el código del programa, compilarlo, cargarlo en la

placa, y observar los resultados enviados por el puerto serie que pueden ser mostrados por el llamado monitor serie. La forma de conectar Arduino con el PC es mediante un cable USB, dependiendo del modelo de Arduino se necesitará uno u otro tipo de cable USB, una vez cargado un programa el Arduino se puede mantener en funcionamiento con un adaptador de corriente o con el mismo cable USB.

La historia de Arduino es reciente, ya que se inició en 2005, y fue creado como un proyecto estudiantil en un instituto italiano con el objetivo de utilizar microprocesadores a un bajo precio. Poco a poco este proyecto fue tomando forma y tras involucrar cada vez a más gente a medida que iba haciéndose más grande, consiguió salir al mercado global, y ser conocido por todos las personas involucradas en este sector tecnológico, lo cual no era algo previsto en los primeros momentos de Arduino.

Figura 2.2: Logo de Arduino

Existen distintos y variados productos Arduino, empezando por las placas, elemento principal de cualquier proyecto Arduino. Existen multitud de placas, cada una con unas características distintas de sus predecesoras, ya sean de distinto tamaño, más potentes o simplemente más modernas y adaptadas al presente y futuro.

Algunas de las placas más utilizadas son las siguientes: Arduino UNO, Arduino Leonardo, Arduino Mega, Arduino Due, Arduino Nano…

Además existen unas placas de expansión que permiten amplían la conectividad de Arduino, dotándoles por ejemplo conectividad GPRS, WIFI o Ethernet.

También se han creado accesorios para aumentar la funcionalidad de Arduino, tales como una pantalla TFT LCD para mostrar imágenes, o una impresora 3D para crear objetos diseñados por los usuarios, como la mostrada en la figura 2.3.

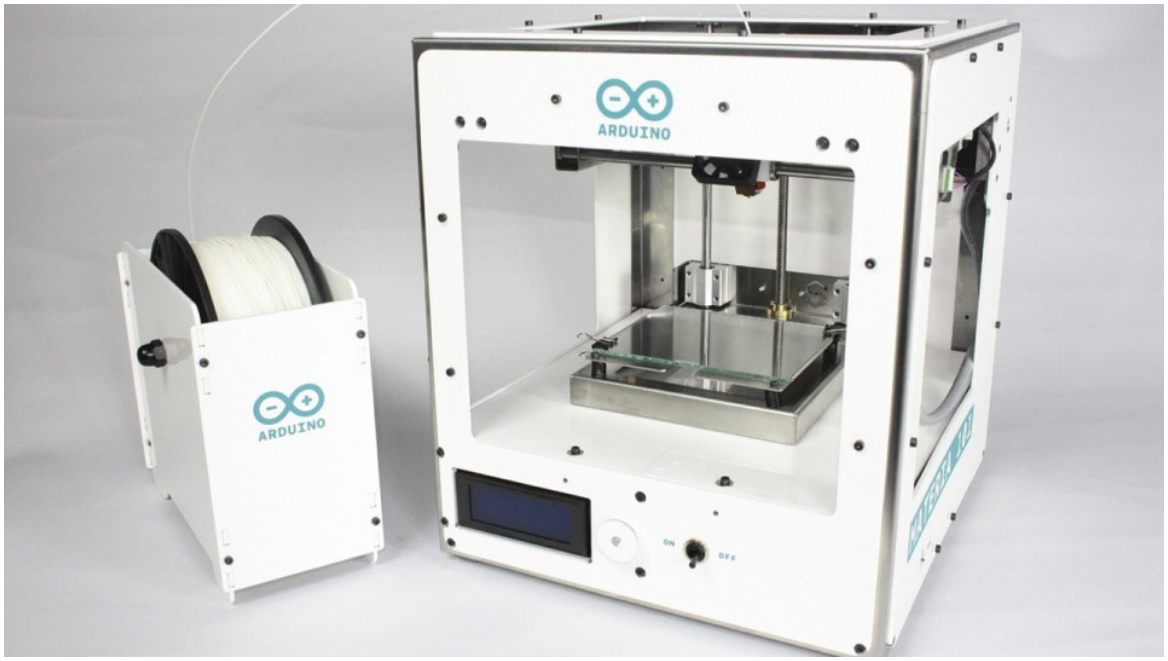


Figura 2.3: Impresora 3D para Arduino

Además de estos productos propios del proyecto Arduino, se pueden utilizar miles elementos que no están creados específicamente para funcionar con Arduino, pero que se pueden utilizar sin ningún problema, simplemente conectando a Arduino por medio de los puertos o pines de conexión.

Todo este conjunto de productos aumenta las posibilidades de Arduino, creando una amplia variedad de aplicaciones, proyectos y utilidades que se pueden llevar a cabo.

1.1 PROGRAMACIÓN EN ARDUINO

Para programar en Arduino se debe utilizar un lenguaje de programación propio de Arduino, que es muy similar a C++. La principal diferencia entre este lenguaje de programación con otros es que en Arduino deben existir 2 funciones principales para su funcionamiento, como explicaremos posteriormente.

El lenguaje de programación de Arduino está basado en Processing, un software del que también escribiremos después y que es muy útil para combinarlo con ciertas aplicaciones desarrolladas en Arduino.

Este lenguaje utiliza las mismas expresiones y sintaxis que se usan en C++, con algunas excepciones debidas al carácter práctico de Arduino.

Un programa de Arduino tiene una extensión ".ino" y es única para Arduino, además puede utilizar otros archivos que sirven como ayuda o

complemento (véase el caso de las librerías) que tengan extensión ".h" como se hace en C++.

1.1.1 Funciones Setup y Loop

Estas funciones deben estar presentes en todos y cada uno de los programas de Arduino, ambas son funciones principales y son las aplicaciones que Arduino ejecuta por defecto.

En el caso de la función "Setup", solo se ejecuta una vez, y será al arrancar el Arduino, y en ella deben estar todos los inicializadores de funciones o elementos (Cuando se resetea el Arduino el programa vuelve a inicializarse desde el principio, ejecutándose por lo tanto la función Setup).

La función "Loop" sin embargo se ejecuta constantemente como lo hace un bucle infinito, la primera vez que se ejecuta es justo tras acabar la función Setup. La función Loop no para de ejecutarse una y otra vez mientras el Arduino esté funcionando adecuadamente, y en ella se debe llamar a todas las funciones que el usuario quiera que se ejecuten a cada iteración.

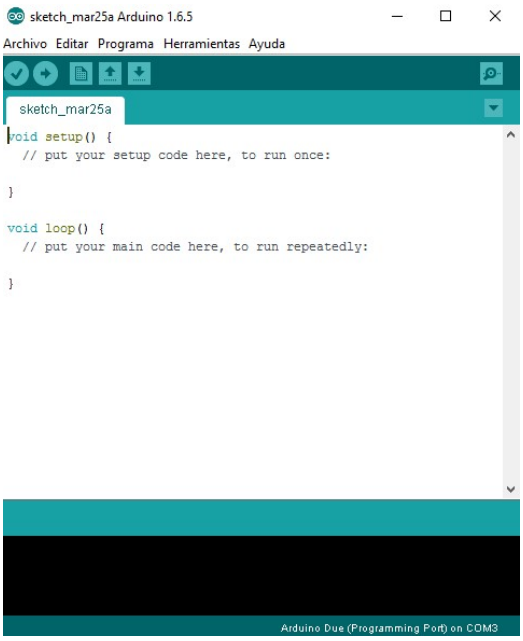


Figura 2.4: Pantalla de Arduino IDE al crear un nuevo proyecto

Al iniciar un nuevo proyecto de Arduino IDE las funciones Setup y Loop estarán ya creadas, como se ve en la figura 2.4.

Esta separación de las dos funciones principales es muy útil, ya que en caso de no llevarse a cabo debiéramos incluir en el lazo principal funciones que solo necesitaríamos que se ejecutasen una vez, perdiendo así potencia y velocidad de procesamiento.

1.1.2 Puerto y Monitor Serie

Una de las librerías principales específicas de Arduino es la "Serial", esta librería te permite mostrar resultados en una pantalla que tiene el Arduino IDE y que denomina "Monitor Serie". Se puede ver el monitor serie en la figura 2.5.

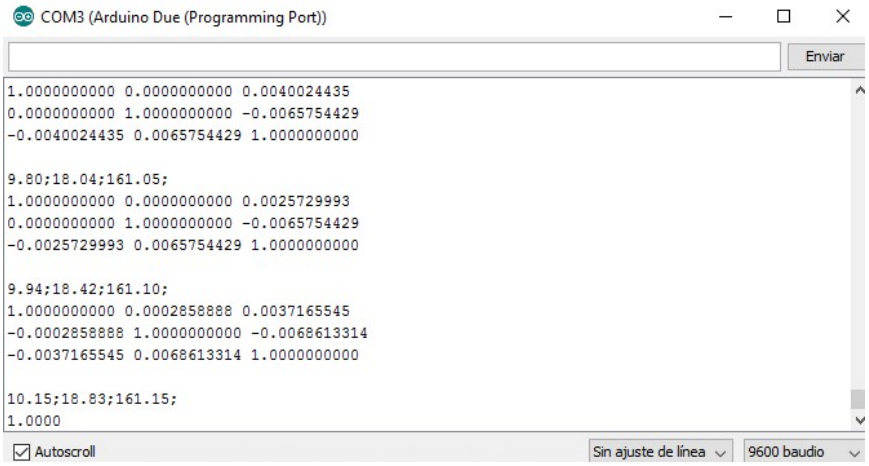


Figura 2.5: Monitor serie

La comunicación serie se lleva a cabo principalmente mediante el puerto de programación USB de Arduino, que comunica Arduino y PC, aunque también se pueden realizar más conexiones serie mediante los pines TX y RX de los que dispone el Arduino.

Para inicializar la comunicación serie es necesario incluir en el Setup una línea que ejecute la función "Serial.Begin()", la cual necesita la introducción de un parámetro, este parámetro son los bits por segundo (o baudios) que se pueden enviar con esta comunicación. Para poder leer los datos mediante el monitor serie será necesario que en este se especifique también los baudios con los que se envían los datos, en el caso de que los baudios especificados en la función "Begin", y los especificados en el monitor serie sean distintos, la información no llegará correctamente al monitor serie.

Las elección de los baudios dependerá mucho del programa a ejecutar, ya que dependiendo de la velocidad de ejecución necesaria será necesario utilizar mayor o menor velocidad en la comunicación. La velocidad en la comunicación es el factor más influyente en el tiempo de ejecución de un programa. En este caso se ha determinado que una velocidad de 19200 baudios será suficiente para este programa, llevando a un periodo de muestreo de unos 20ms.

La gran mayoría de las placas Arduino solo disponen de un puerto serie, por lo que no se pueden realizar varias comunicaciones serie a menos que

se utilice otra librería llamada "SoftwareSerial", la cual permite utilizar pines de entradas/salidas normales como pines serie RX y TX. En el caso de las placas Arduino mas grandes como puede ser la Mega o la Due, se disponen de varios puertos serie, evitando así la necesidad de utilizar la librería "SoftwareSerial" y simplificando la programación.

1.1.3 Librerías

Arduino dispone de varias librerías (o bibliotecas) básicas que se pueden utilizar para realizar ciertas funciones útiles. Algunas de estas librerías son las comentadas anteriormente, como la "Serial" o la "SoftwareSerial", otras que también utilizaremos en este proyecto son las librerías "Wire" para la comunicación I2C con la IMU, o la librería "Ethernet" que usaremos para la comunicación UDP mediante un cable de red Ethernet, .

Además de las librerías que vienen ya con el Arduino, podemos instalar otras librerías que usuarios de todo el mundo han colgado en internet, para ello Arduino IDE tiene un buscador de librerías integrado con el cual se podrán buscar y descargar directamente estas librerías.

En el caso de que no haya ninguna librería específica para nuestro proyecto también podemos crear una librería propia, con las funciones necesarias para simplificar el código del archivo ".ino".

Las librerías de Arduino son como las librerías en C++, utilizan normalmente 2 archivos, un archivo principal ".h" donde se encuentran las constantes, direcciones y la definición de todas las funciones y variables, y otro archivo ".cpp" donde se encuentra toda la programación de estas funciones. En el caso de ser una librería sencilla es posible utilizar un único archivo ".h".

Para utilizar una librería creada por el usuario hace falta cargarla en el Arduino IDE y añadir la llamada en el archivo de programación ".ino".

1.2 ARDUINO DUE

La placa Arduino con las que se va a realizar este proyecto es la denominada Arduino Due, uno de los Arduino más potentes y con más posibilidades que a fecha de hoy existen. En la figura 2.6 se muestra una fotografía de la placa, se puede ver en ella todas los puertos de conexión de los que dispone.

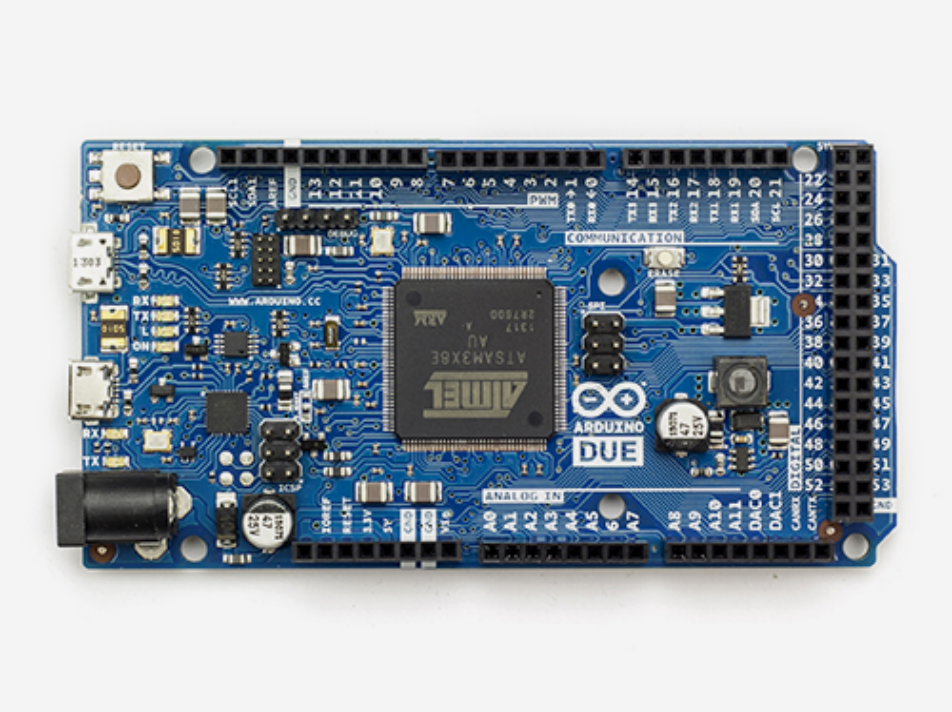


Figura 2.6: Arduino Due

Arduino Due salió al mercado en 2012 como el mejor Arduino del momento, gracias a su gran procesador de 32 bits a 84MHz ARM Cortex-M3, que consigue un funcionamiento mucho mejor en programas complejos que podían tener problemas en otro Arduino de menor potencia.

Una característica singular de esta placa es que a diferencia de la mayoría de ellas, esta trabaja con un nivel de tensión de 3.3 V, por lo cual hay que tener especial cuidado en utilizar esta tensión y no la típica utilizada en otros modelos de 5V que podría dañar este procesador.

Además del procesador también cuenta con varias ventajas con respecto a otros Arduino, como por ejemplo la existencia de un segundo puerto micro USB, que puede ser utilizado como cliente o como host, o incluso para conectar un teclado o un ratón. También cuenta con un números de pines de entradas/salidas superior a la mayoría de Arduinos, lo cual da la posibilidad utilizar más conexiones simultáneamente. Entre los pines hay 8 utilizados para la conexión en serie (4 puertos serie, de RX0-TX0 a RX3-TX3), eliminando la necesidad de utilizar la librería "SoftwareSerial".

Tiene un gran números de pines que se pueden utilizar como entradas o salidas digitales (54 pines concretamente), y 12 entradas analógicas. También dispone de 2 pines (SDA y SCL) para comunicación I2C, 2 conversores analógico-digital, conector SPI, pines para comunicación

CAN, un LED conectado directamente a una salida, lo que posibilita conocer el valor de esta.

1.3 PLACAS DE EXPANSIÓN: ETHERNET SHIELD

Las placas de expansión son unas placas similares a las principales de un Arduino, que se conectan mediante los pines de conexión a una placa principal de Arduino. En la figura 2.7 se puede ver una placa Arduino con 2 placas de expansión sobre ella.

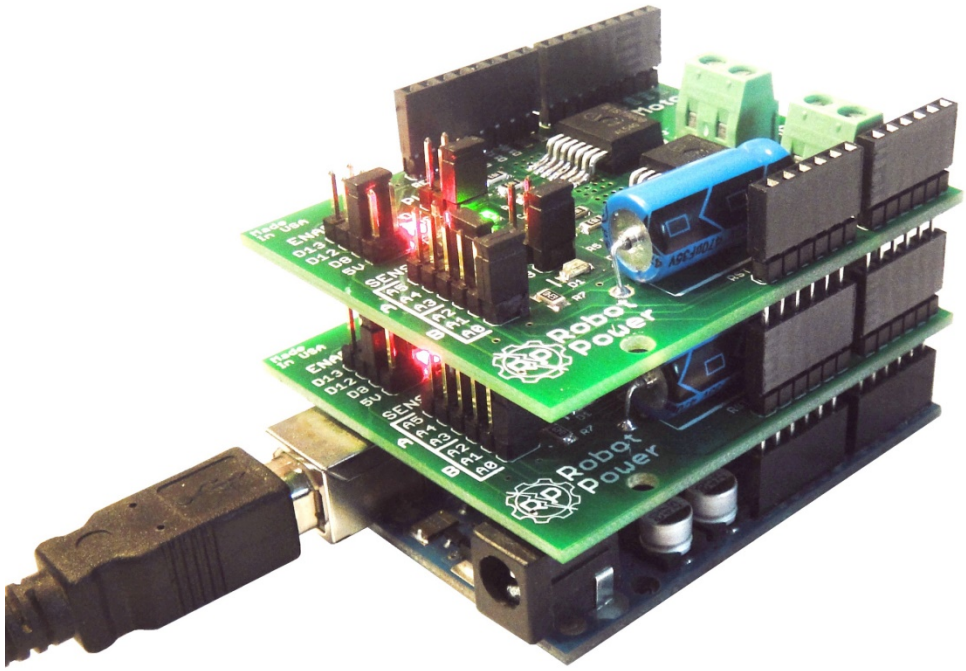


Figura 2.7: Ejemplo de utilización de varias placas de expansión

Existen multitud de placas de expansión diferentes que añaden a Arduino distintas posibilidades, algunos ejemplos de ellas son las de comunicación Wifi, comunicación GPRS, cargador solar, display TFT, etc…

En este proyecto se utilizará la placa Ethernet, la cual tiene un puerto que permite conectar un cable Ethernet a Arduino, además esta placa permite también la utilización de una tarjeta de memoria SD para almacenar información, aunque en este proyecto no será necesaria.

Un cable Ethernet es un cable utilizado generalmente para la conexión a internet, dispone de 4 pares trenzados (8 hilos) y está diseñado para transmitir datos, por lo que no se utiliza para que pasen grandes intensidades por él.

Esta placa se puede utilizar para conectar el Arduino a un modem, lo cual dotaría a Arduino de acceso a internet, dando entrada a un gran número

de posibilidades, como por ejemplo conectar indirectamente Arduino con dispositivos en el otro lado del mundo.

Se puede ver una placa de expansión Ethernet Shield en la figura 2.8.

Figura 2.8: Ethernet Shield

En el caso de este proyecto se utilizará para conectar a Arduino con un ordenador, enviando datos mediante una comunicación UDP. Para ello será necesario utilizar la librería "Ethernet" de la cual dispone Arduino IDE.

1.3.1 Comunicación UDP

UDP ("User Datagram Protocol") es un protocolo de comunicación que permite un sencillo transporte de información a una buena velocidad.

Este protocolo realiza envíos de "datagramas", o paquetes de datos sin necesidad de realizar ninguna sincronización entre origen y destino, lo cual facilita la conexión.

El principal problema de la comunicación UDP es su poca fiabilidad, ya que no controla los envíos ni ordena los paquetes, lo cual provoca posibles errores en la calidad de la información recibida. Esto es más notable cuando la longitud de los paquetes de datos es variable, por eso se tratará de enviar siempre cadenas de caracteres de la misma longitud, ya que de no ser así la información transmitida no sería del todo fiable.

Por eso la conexión UDP se utiliza para casos en los que cobra más importancia la velocidad de la comunicación que el hecho de que todos los bytes lleguen correctamente siempre. En este caso se tratará de no forzar

al máximo los envíos de paquetes para que la información enviada sea lo más correcta posible.

Para establecer una conexión UDP en Arduino es necesario definir una dirección IP y el puerto con los cuales funcionara el Arduino.

La mayoría de los programas mediante los cuales se puede realizar programación son capaces de utilizar la comunicación UDP, como por ejemplo Matlab, que tiene funciones para crear y utilizar este protocolo, y se utilizará para realizar las pruebas del correcto funcionamiento de la comunicación.

En el siguiente ejemplo se comprueba la funcionalidad de la comunicación UDP, a través de Arduino recogemos y procesamos una serie de datos (que se verán en apartados posteriores), y los enviamos en forma de paquetes de datos a través del cable Ethernet. Utilizando Matlab se reciben utilizando un bucle infinito que se limita a buscar datos entrantes por el puerto Ethernet.

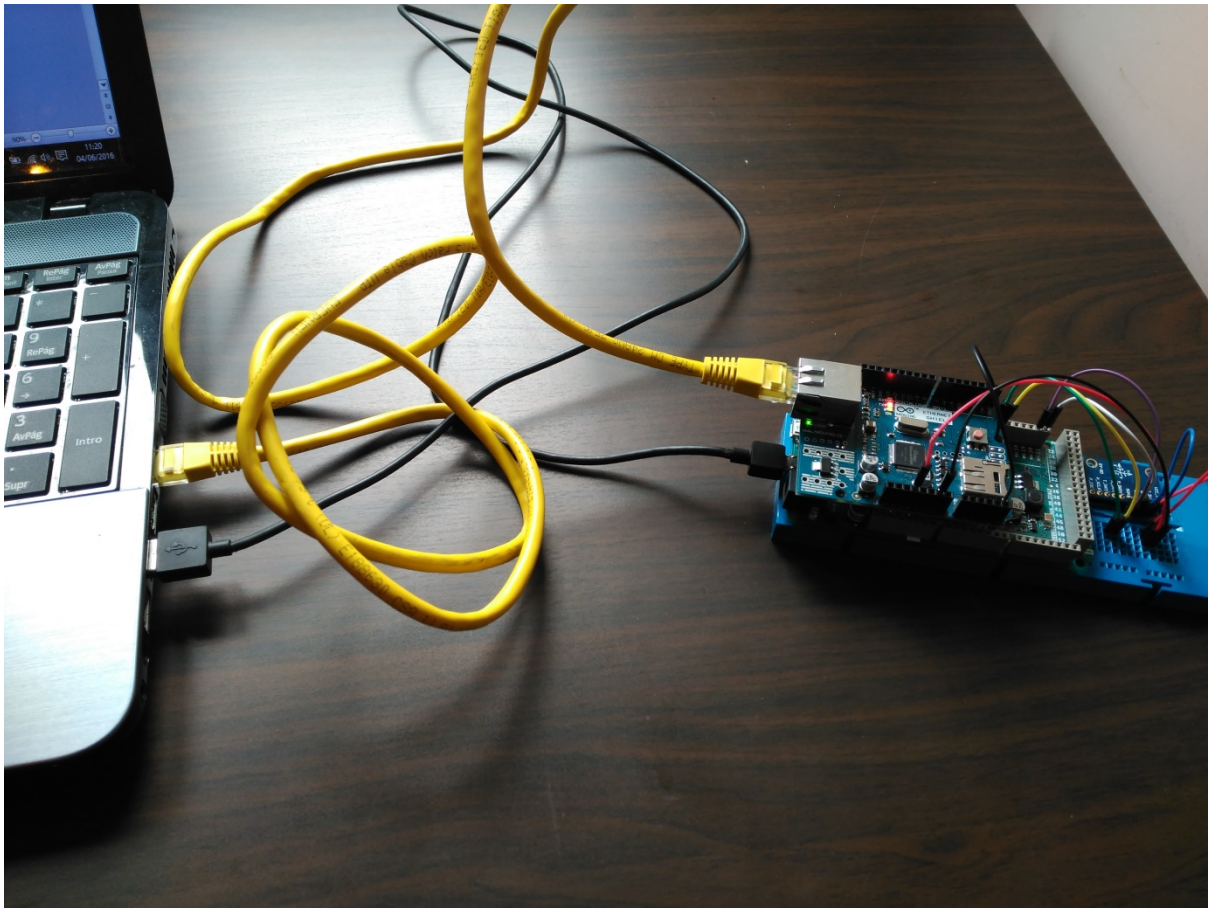


Figura 2.9: Comunicación UDP Arduino-PC

Un código simple que se puede utilizar en Matlab para establecer la comunicación UDP y esperar para recibir los paquetes de datos es el siguiente:

```
clear; close all; clc;
PortLocal = 8000;
PortHost = 4000;
IPLocal = '192.168.1.2';
IPHost = '192.168.1.177';
u = udp(IPHost, PortHost);
u.Timeout = 5;
u.LocalHost = IPLocal;
u.LocalPort = PortLocal;
fopen(u);
while (1)
fscanf(u)
end
```

Como se puede ver, se especifican las direcciones IP y los puertos tanto del Arduino como del PC, aunque la dirección y el puerto del PC no son imprescindibles para establecer una comunicación correcta.

En la siguiente figura se pueden apreciar los resultados obtenidos en Matlab, los cuales son los esperados. Estos datos conforman una cadena de caracteres, por lo tanto habrá que procesarlos para dividirlos y almacenarlos en las diferentes variables necesarias para su posterior procesamiento para el control por parte del ordenador de a bordo del vehículo.

Figura 2.10: Resultados obtenidos utilizando la comunicación UDP con Matlab

CAPÍTULO 3: UNIDAD DE MEDICIÓN INERCIAL (IMU)

3.1 DESCRIPCIÓN

Una unidad de medición inercial o IMU (de sus siglas en inglés, "Inercial Mesurement Unit") es un dispositivo dotado de varios sensores que miden variables tales como la velocidad angular, o las fuerzas soportadas en las distintas direcciones del espacio. Los principales sensores de los que dispone una IMU son acelerómetros y giróscopos, y además a veces también pueden usar magnetómetros o barómetros (que en realidad no son sensores inerciales pero se pueden combinar sus funcionalidades).

Las IMU son ampliamente utilizadas en la navegación de vehículos, principalmente de aviones o barcos, y son de importancia capital en vehículos o aparatos no tripulados, tales como naves espaciales, satélites, misiles guiados o drones como el de la figura 3.1. Además también se usan en otras aplicaciones como pueden ser los vehículos "equilibrados" como un Segway.

Figura 3.1: Un Quadcopter también utiliza una IMU para su funcionamiento

Otro ámbito para el cual puede ser utilizada una IMU es para la medicina, ya que se puede registrar los movimientos de una persona y a partir de esos datos tomar medidas para mejorar su diagnóstico o tratamiento.

Los datos recogidos por los sensores de la IMU pueden ser enviados a un computador, ayudando a conocer en todo momento la posición y la orientación de un vehículo. A este método de posicionamiento se le llama "navegación por estima".

Si se conoce en todo momento la orientación que tiene el dispositivo (gracias a los acelerómetros y/o a los magnetómetros) y la velocidad de

giro en cualquier dirección, se puede calcular el movimiento relativo de la IMU con respecto a la posición inicial. Combinado además con un sistema de posicionamiento global como puede ser un GPS, tendremos también la posición absoluta del dispositivo en todo momento de forma más precisa que sin la combinación de estos sensores.

El principal problema de la navegación por estima mediante una IMU se debe al error acumulado (también denominado deriva o drift) con cada medición, ya que inevitablemente las medidas no son 100% precisas, el error de cada medición se va sumando al error acumulado, y cuando más tiempo pasa, mayor puede ser el error cometido en el cálculo de la posición.

Por esto es recomendable usar otros dispositivos que ayuden a corregir el error que pueda tener una IMU, como por ejemplo un GPS.

En la práctica se habla de que una IMU tiene un determinado números de grados de libertad en función del número de medidas independientes que obtiene. Las más habituales son la de 6 grados de libertad, que están dotadas de 3 acelerómetros (también se puede hablar de acelerómetros de 3 ejes, que dan 3 grados de libertad), y 3 giróscopos (otros 3 grados de libertad). Si además disponen sensores magnéticos de 3 magnetómetros se añaden otros 3 grados de libertad, y menos comúnmente, cuando de forma adicional utilizan un barómetro, se habla de IMUs de 10 grados de libertad, como la que se utilizará en este proyecto.

Cada sensor está orientado ortogonalmente al resto de sensores de su mismo tipo, es decir, un acelerómetro está colocado ortogonalmente a los otros 2 acelerómetros. De esta forma se puede medir la aceleración del dispositivo en cualquier dirección del espacio de una forma lo más sencilla posible. Además. la orientación de uno de los ejes del acelerómetro siempre coincide con la de un eje del giróscopo, de manera que podemos tener los 3 ejes de la IMU, llamados normalmente X, Y, y Z, que son coincidente para cada conjunto de sensores, ayudando posteriormente a la fusión de estos.

3.2 COMPONENTES

Tal y como se ha dicho antes una IMU está compuesta habitualmente por un conjunto de acelerómetros, giróscopos y magnetómetros, y en la práctica muchos de estos dispositivos electrónicos disponen de un barómetro que puede medir temperatura y presión. A continuación se hablará de cada uno de estos sensores, la ciencia en la que se basan, que hacen y que aportan.

Técnicamente los magnetómetros y los barómetros no son sensores inerciales aunque se implementen en una IMU, a los sensores con magnetómetros también se les puede denominar MARG (Magnetic, Angular Rate, Gravity) aunque es menos común encontrarlos con este nombre.

Actualmente, la tecnología ha conseguido construir conjuntos de 3 sensores en chips de pequeñas dimensiones, a esta tecnología se la conoce como MEMS (o Sistemas Microelectromecánicos), esta posibilidad de tener un sensor inercial de gran precisión y de pequeñas dimensiones es un gran avance para por ejemplo la tecnología de los drones, ayudando a controlar su estabilidad y mejorando su funcionamiento. Además el bajo precio de estos dispositivos ha motivado su extensión en el mercado y la industrial, ya que en el pasado los sensores inerciales únicamente se podían utilizar para proyectos con grandes presupuestos.

3.2.1 Acelerómetro

Un acelerómetro, como se puede deducir por su nombre, es un instrumento que sirve para medir aceleraciones, aunque al contrario de lo que se puede pensar en un primer momento, no solo mide las aceleraciones a las que podemos someter al dispositivo provocando cambios en su posición y velocidad (fuerzas dinámicas), sino que también mide la aceleración que de forma constante provoca la gravedad (fuerzas estáticas), es decir, un acelerómetro siempre va a tener en cuenta una aceleración de 9.8 m/s^2 en sentido hacia el centro de la tierra, esta característica tiene una grandísima importancia en cómo se puede utilizar un acelerómetro.

Figura 3.2: Diferentes modelos de acelerómetros

Existen varias formas de funcionamiento para que un acelerómetro pueda medir la aceleración, típicamente funciona mediante una masa interna (por ejemplo, una bola minúscula) que se permite su movimiento en una única dirección, esa dirección será en la cual se medirá la aceleración. En cada uno de los 2 sentidos que se puede mover, a la bola se le impedirá parcialmente el movimiento mediante unas placas o muelles, cuanto mayor sea la aceleración más se desplazará la bola (gracias a un resorte elástico). El sensor puede medir cuanto se ha desplazado la placa que impide el movimiento a la bola, en función a ese desplazamiento se calcula la aceleración a la que está sometida la bola en esa dirección gracias a la segunda ley de Newton.

$$F=m\cdot a \tag{3.1}$$

La masa (m) es conocida de antemano, la fuerza provocada (F) es medida por un dinamómetro, y la aceleración es calculada por la ecuación:

$$a = \frac{F}{m} \tag{3.2}$$

Las unidades que se manejan con un acelerómetro son las de aceleración, que pueden ser los metros por segundo al cuadrado, o las llamadas fuerzas "G", una fuerza G equivale a la fuerza de la gravedad en la tierra ($9.8 m/s^2$), y el rango valores que puede medir un acelerómetro MEMS normalmente varía entre los ±2G y los ±16G.

Existen otros tipos de acelerómetros que utilizan diferentes métodos para medir la aceleración, como por ejemplo los que utilizan elementos piezoeléctricos para medir la fuerza con la que son presionados,

acelerómetros de efecto Hall (variaciones magnéticas), o acelerómetros de condensador (condensadores con placas móviles y capacidad variable).

En cuanto a cómo se puede utilizar un acelerómetro, se tiene por ejemplo la posibilidad de calcular la orientación en el espacio del dispositivo que utilice 3 sensores con 3 ejes ortogonales entre sí, siempre y cuando este no esté sometido a un movimiento que genere aceleraciones. Teniendo en cuenta que el dispositivo está midiendo únicamente a la aceleración que provoca su peso, y esta es de 9.8m/s^2 en dirección hacia el centro de la tierra, y sabiendo también las medidas de los acelerómetros en 3 ejes ortogonales. Mediante fórmulas trigonométricas sencillas se puede obtener la inclinación del dispositivo respecto a unos ejes de referencia. Aunque obviamente no se puede conocer la orientación geográfica, para ello será necesario un magnetómetro.

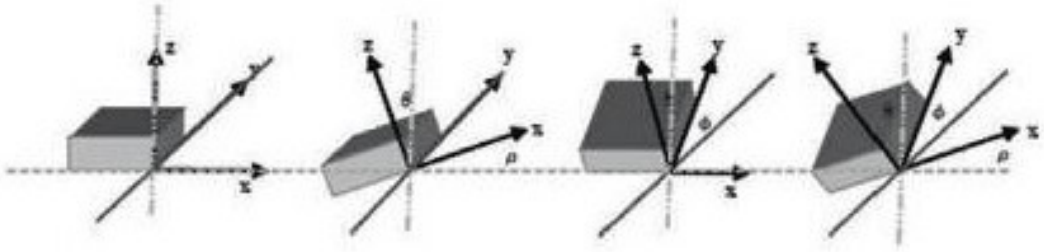


Figura 3.3: Rotación en los 3 ejes del acelerómetro

Este método es utilizado por ejemplo en la inmensa mayoría de nuestros teléfonos móviles o "smartphones", para aplicaciones tales como girar la pantalla automáticamente en función de la orientación (vertical/horizontal) del móvil, o para determinados juegos que pueden ser controlados mediante el giro del móvil.

En cuanto al problema que surge cuando los acelerómetros se ven sometidos a una aceleración dinámica (ya que no es posible calcular su orientación), se puede arreglar utilizando otros sensores tales como giróscopos, combinando la información en lo que se conoce como una fusión sensorial. De esta forma la información suministrada es más fiable y por lo tanto más útil.

Si mediante la fusión con otros sensores se calcula la orientación de la IMU a tiempo real, entonces teóricamente, será posible obtener, mediante integración de los datos leídos por los acelerómetros, la velocidad y el movimiento de la IMU, conociendo la aceleración en todos los ejes y eliminando la aceleración estática provocado por la gravedad (al conocer la orientación de la IMU sabemos cómo debe ser el vector gravitacional), obtenemos la aceleración dinámica del sistema.

En la práctica esto es bastante más complejo, debido a los errores de medición de los acelerómetros y a la integración de estos. Al realizar una integración se genera un error de deriva, provocado la suma continua de los pequeños errores de las mediciones del acelerómetro, este error de deriva va en aumento con el tiempo, ampliando la incertidumbre de los resultados obtenidos. En el caso del cálculo de la posición se necesita una doble integración, por lo tanto los errores de deriva son mucho mayores.

Hay varios métodos para reducir este error, el más simple de ellos es tratar de eliminar los errores en las mediciones, para ello hay que calibrar el acelerómetro en sus tres ejes, así reducimos la velocidad de la deriva.

También existen métodos para compensar o filtrar el error, como puede ser la realización de un filtro utilizando sensores de posicionamiento global como puede ser un GPS. Fusionando los datos obtenidos por el GPS y por los acelerómetros obtendremos resultados más fieles a la realidad.

3.2.2 Giróscopo

Un giróscopo (o también denominado giroscopio) es un dispositivo que mide la velocidad angular a la que este gira respecto a un eje, las unidades con las que se suele medir esta velocidad angular son los grados por segundo.

Existen 3 principales tipos de giróscopos, clasificados en función de su forma constructiva: Giróscopos mecánicos, ópticos, o los MEMS. Los primeros en utilizarse fueron los mecánicos, pero después se crearon los ópticos y los MEMS los cuales tienen mayor utilidad práctica. Los giróscopos de mayor precisión son los ópticos, que son utilizados para la navegación en barcos o aviones, en cambio los MEMS son muy utilizados por su bajo precio y su facilidad de uso.

Básicamente un giróscopo mecánico es un mecanismo simétrico respecto a un eje (un disco o un aro, por ejemplo) que gira respecto a su eje de simetría, y se mantiene girando respecto a ese mismo eje a pesar de que se le someta a fuerzas que intenten cambiar su orientación. Esto es debido al efecto de la conservación del momento angular, que hace que mientras el giróscopo se encuentre girando, este se resista a los cambios de orientación.

Esa característica hace posible que midiendo la diferencia angular existente entre el momento inicial conocido, y en otro momento cualquiera, puedas conocer la orientación en cualquier momento, ya que sabes que el eje del giróscopo estará de forma permanente con la misma orientación.

Para que se pueda producir este efecto es necesario que el rozamiento que se produzca sea mínimo, por eso se utilizan soportes especiales de alta calidad, que permiten la rotación sin rozamiento en todas direcciones, como por ejemplo el "soporte de Cardano", mostrado en la figura 3.4.

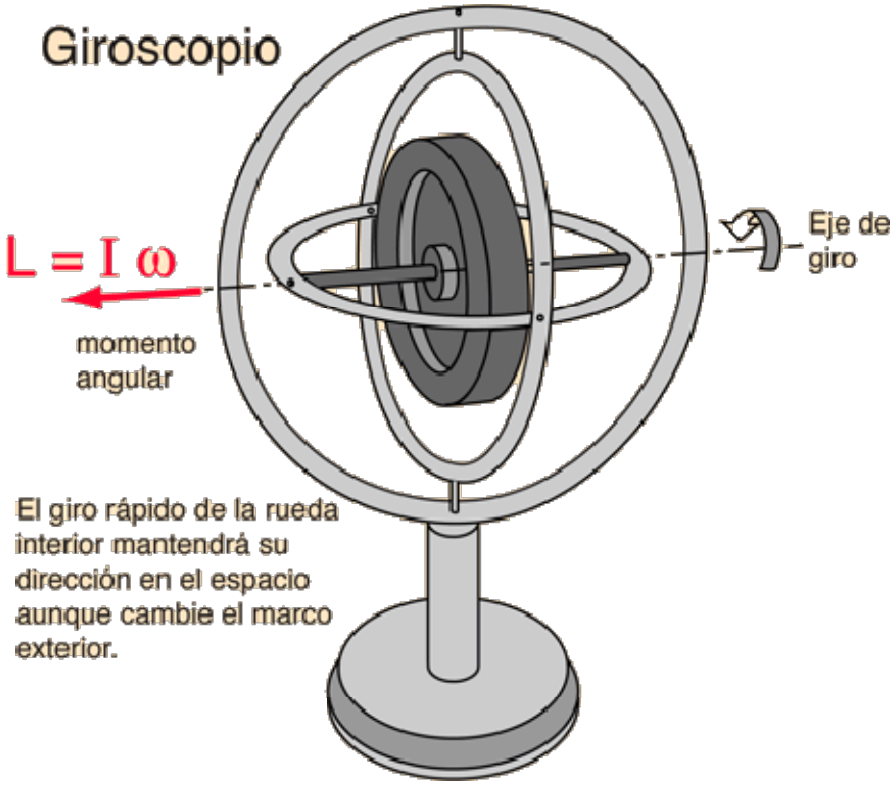


Figura 3.4: Giróscopo en un soporte de Cardano

A los 3 anillos que forman el llamado soporte de Cardano se les denomina "cardanes", y a diferencia del disco del giróscopo estos si giran cambiando su orientación. Los cardanes disponen de cojinetes de gran calidad que aseguran rozamientos ínfimos y ayudan a que el giróscopo tenga una gran estabilidad y mantenga la dirección del eje.

Pese a que los rozamientos sean mínimos, existen, y por lo tanto poco a poco van generando un error con el tiempo.

Los efectos de un giróscopo se pueden observar en diferente medida en objetos cotidianos, como las peonzas o en las ruedas de las bicicletas o motocicletas.

En los dispositivos electrónicos de pequeño tamaño como el que se utilizará en este proyecto, no es posible utilizar un giróscopo como los anteriores, por lo tanto los giróscopos que se utilizarán serán los MEMS, ya que caben en las dimensiones de un pequeño chip. Su funcionamiento es diferente al de un giróscopo tradicional. Siguen basándose en un disco que gira alrededor de un eje, pero en este caso el disco tiene una masa

interior (y por lo tanto no es inercialmente simétrico) que dependiendo del movimiento del disco vibra de una manera determinada, esa vibración es medida y a partir de ella se calcula la velocidad de giro instantánea en ese eje.

Esta vibración es debida al efecto Coriolis, que genera una fuerza en sentido perpendicular al eje de giro y cuya magnitud depende de la velocidad angular de giro.

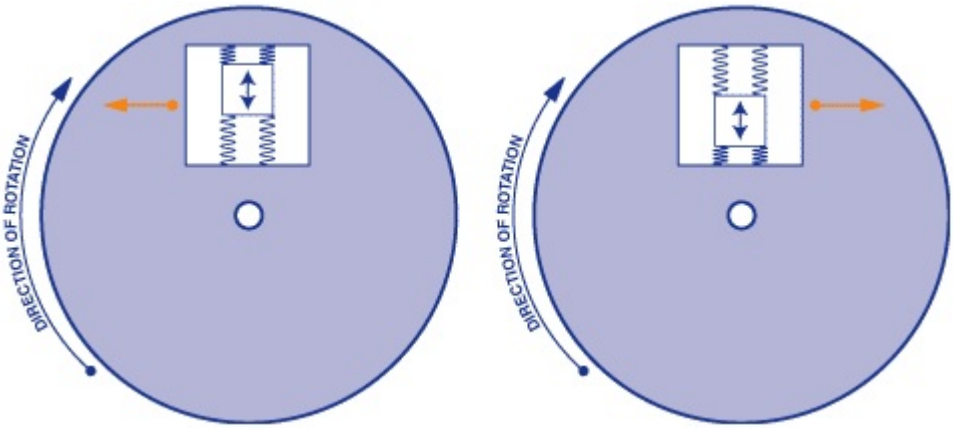


Figura 3.5: Funcionamiento interno de un giróscopo MEMS

El mayor problema que existe al calcular el ángulo girado mediante giróscopos es la deriva (o drift en inglés), la deriva es el error acumulado en las mediciones, ya que para calcular el ángulo girado hay que integrar las medidas que nos dan los giróscopos, y por lo tanto si todas estas medidas tienen un pequeño error, este se va acumulando con el tiempo.

Una de las maneras de tratar de eliminar o disminuir la deriva es eliminar los errores en las mediciones. Para ello será necesario realizar una calibración del giróscopo, como se verá en apartados posteriores.

Otra forma de solucionar el problema de la deriva es utilizar un filtro usando también las mediciones de acelerómetros o magnetómetros.

Los giróscopos son de una gran utilidad en un gran número de ámbitos. Su principal uso fue y sigue siendo en la navegación inercial o navegación por estima, para aviones, naves espaciales o misiles. Actualmente los teléfonos móviles también los usan de forma similar y complementaria a la que se comentó de los acelerómetros, ya que estos 2 dispositivos se complementan muy bien, como en el caso de una IMU.

3.2.3 Magnetómetro

Un magnetómetro es un dispositivo sensible a los campos magnéticos que sirve para medir la magnitud de estos, también pueden ser llamados brújulas digitales. Una de sus principales funciones es medir el campo magnético terrestre para poder conocer la orientación, debido a su fiabilidad son ampliamente usados para esta utilidad.

Figura 3.6: Magnetómetro de 3 ejes

Existen dos tipos de magnetómetros, dependiendo si pueden medir la dirección del campo magnético en una dirección particular o no. En el caso de que un magnetómetro solo pueda medir la intensidad total de un campo magnético en un punto determinado se le denomina magnetómetro escalar, y en el caso de que el magnetómetro pueda medir la intensidad de un campo magnético en una determinada dirección, se le denomina magnetómetro vectorial.

Por lo tanto, al igual que ocurría con los anteriores componentes de la IMU, se necesitan 3 magnetómetros vectoriales (o un magnetómetro de 3 ejes) para conocer perfectamente como es el campo magnético en un punto. Podría decirse que un magnetómetro de 3 ejes sería una brújula que funciona en las 3 dimensiones.

La unidades utilizadas para medir el campo magnético son los Teslas (T) en el sistema internacional, o los Gauss (G), Un Tesla es una unidad bastante grande, ya que 1T equivale a 10000G, y el campo magnético de la tierra tiene una intensidad aproximada de 0.5G (0.00005T), por eso se suelen utilizar unidades más pequeñas como lo mT.

Evidentemente los campos magnéticos provocados por la cercanía de una corriente eléctrica o por un imán también tienen influencia en las mediciones de un magnetómetro, y pueden ser medidos por ellos. Por eso si el objetivo es medir el campo magnético terrestre, hay que evitar toda posible interferencia que pueda provocar mediciones erróneas, como podría ser utilizar el magnetómetro cerca de un objeto metálico.

Esta propiedad no siempre es negativa, puede ser utilizada para buscar anomalías que indiquen la presencia de, por ejemplo, objetos metálicos en un espacio determinado, esto los hace particularmente eficaces en la detección de metales, son muy utilizados para uso militar en este aspecto, para la detección de submarinos u otros vehículos. También se utilizan en la minería para la búsqueda de minerales.

Los magnetómetros, al igual que el resto de los sensores pueden tener un Offset, el cual hay que contrarrestar para obtener mejores medidas. Para calcular el Offset en los 3 ejes hay que realizar 3 mediciones, cada una de ellas consiste en girar el magnetómetro 360º según uno de los ejes, esto nos dará un Offset aproximado de los otros 2 ejes.

Dibujando en una gráfica los puntos de cada medida (valor medido en un eje respecto valor medido en el otro eje), debiéramos obtener un circulo, y el Offset en cada eje es la desviación del centro del circulo con respecto el origen de coordenadas.

Un problema que surge al medir los campos magnéticos terrestres es que estos no son constantes, pueden variar su intensidad y orientación en función del lugar en el que nos encontremos en la Tierra. Y además estos campos magnéticos varían levemente a lo largo de los años.

En la figura 3.7 se muestra un mapa de la Tierra con la representación superpuesta del campo magnético de esta para los años entre el 2015 y el 2019.

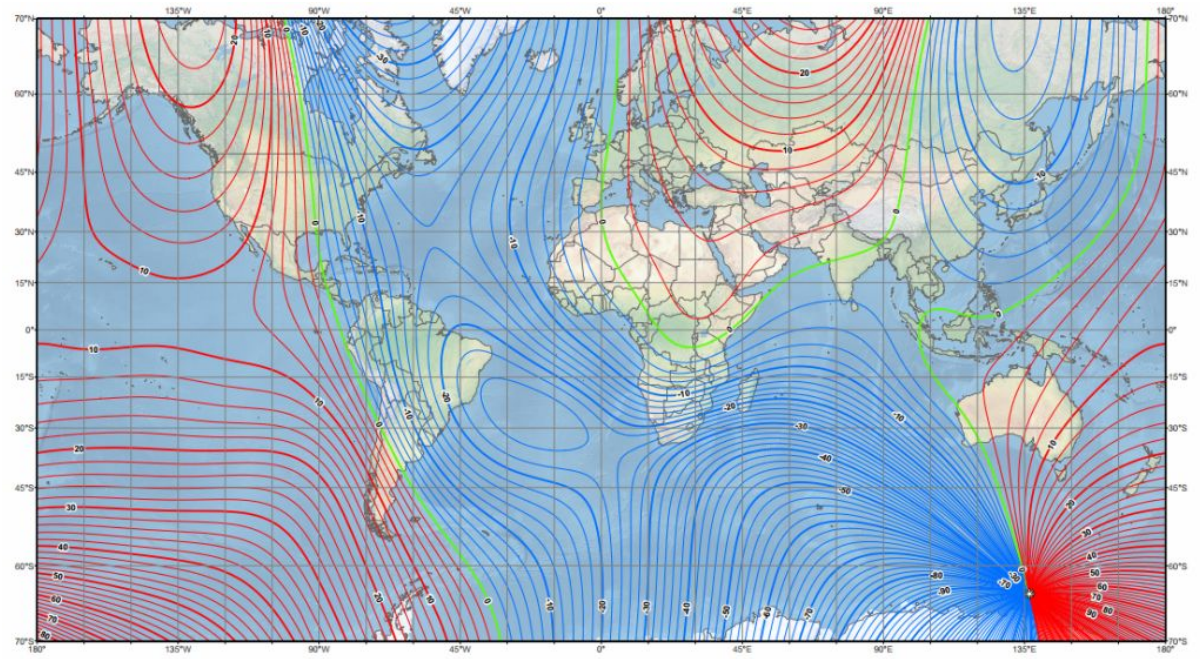


Figura 3.7: Modelo declinación magnética 2015-2019 de la NOAA

Para tener mayor precisión en el cálculo de la orientación hay que sumar a la medición la declinación magnética, esto compensa la diferencia existente entre los polos magnéticos y los polos geográficos. La declinación también incluye los pequeños cambios temporales anteriormente comentados, por eso hay programas en internet para calcular la declinación en función de la ubicación y de la fecha.

El funcionamiento de un magnetómetro "flux-gate" (los más comunes) es bastante simple comparado con el de un acelerómetro o un giróscopo. Consiste en dos bobinados en un mismo núcleo magnético, un bobinado de excitación y otro de medición. La excitación satura el núcleo, y el de medición detecta las variaciones producidas por un campo magnético externo. Después esta variación es traducida a unidades de campo magnético mediante un bloque electrónico. En la figura 3.8 se muestra un esquema del funcionamiento de uno de estos magnetómetros.

Otros magnetómetros más recientes pueden medir las variaciones del campo mediante resistencias sensibles a estos.

Gracias a la precisión de un magnetómetro se puede llegar a medir una variación de un giro de 1 segundo (como medida angular).

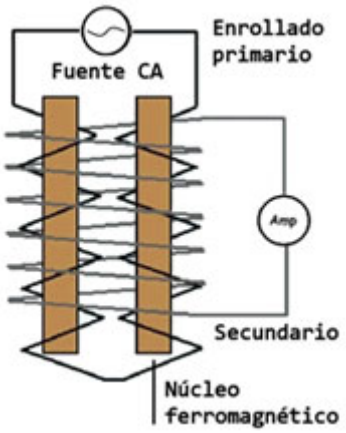


Figura 3.8: Esquema eléctrico simplificado de un magnetómetro

3.2.4 Barómetro

Un barómetro es un instrumento que sirve para medir la presión atmosférica. Los barómetros tienen un origen bastante antiguo y los primeros en fabricarse consistían en una columna de líquido en un tubo cerrado por su parte superior, cuanto mayor era la presión, mayor nivel alcanza el líquido en la columna.

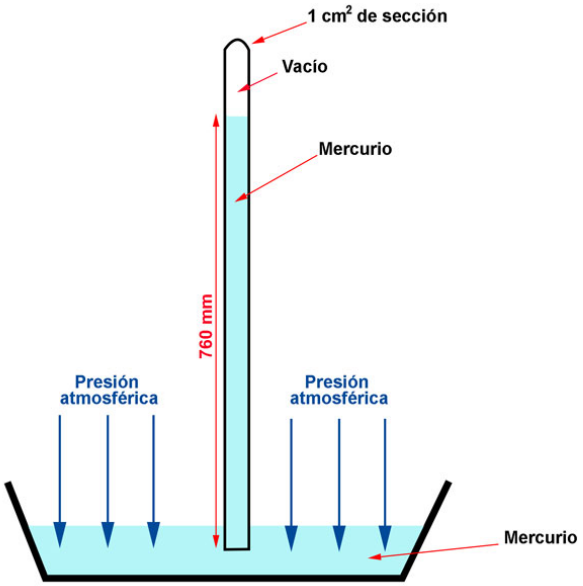


Figura 3.9: Esquema del funcionamiento de un barómetro de mercurio

Hay barómetros de diferentes tipos, siendo el más conocido el barómetro de mercurio como el de la figura 3.9. Sin embargo, en la actualidad, la mayoría de los barómetros son digitales y son capaces de medir la presión absoluta con gran precisión.

En el caso del barómetro del que dispone la IMU y que se va a utilizar en este proyecto también es capaz de medir la temperatura.

Aunque bastantes IMU lo usan, un barómetro no es un sensor inercial como el acelerómetro o el giróscopo, sin embargo se suelen implementar en las IMU debido a su sencillez y a su utilidad en dispositivos voladores para calcular la altitud de forma aproximada.

El cálculo de la altitud sin embargo es bastante poco preciso para la utilidad que en este proyecto se le puede dar, ya que en un vehículo la altura no varía mucho, y se ve muy afectada por las variaciones de la presión atmosférica, generando un margen de error considerable y haciendo la medición poco fiable.

3.3 IMU GY-80

La IMU GY-80 es una unidad de medición inercial de 10 grados de libertad (10 DOF), la cual cuenta con un acelerómetro de 3 ejes (ADXL345), un giróscopo de 3 ejes (L3G4200D), un magnetómetro de 3 ejes (HMC5883L) y un barómetro (BMP085).

Figura 3.10: Vista frontal de la IMU GY-80

Figura 3.11: Vista posterior de la IMU GY-80

Como se puede ver en las figuras 3.10 y 3.11, esta IMU tiene 10 pines para su conexión, Solo se necesitaran conectar 4 de ellos. "VCC_3.3V" se utilizará para la entrada de alimentación positiva, que serán los 3.3V del Arduino DUE (con otro Arduino que funcionara a otro nivel de tensión sería necesario utilizar el pin "VCC_IN"), el pin "GND" será la conexión a la tierra del Arduino, y SLC y SDA que son los pines para la comunicación I2C, serán conectados a los pines correspondientes de los que dispone el Arduino. El resto de pines no será necesario conectarlos.

La GY-80 utiliza una comunicación I2C, que sirve para comunicar circuitos integrados, necesita 4 conexiones para funcionar, 2 conexiones son para alimentar y establecer un nivel de tensión (la GY-80 puede funcionar a 3.3V o a 5V), y los otros 2 hilos son para el transporte de información, estos dos hilos son el SCL (System Clock), y el SDA (System Data).

La comunicación I2C necesita de al menos 2 dispositivos para que se comuniquen, uno actuará como maestro y otro como esclavo, el maestro llama a una dirección de 2 bytes hexadecimal del esclavo, e intercambiará información con ella.

Arduino posee una librería para la comunicación I2C, se trata de la librería "Wire.h" que se utilizará para poder comunicar con la IMU y sus componentes.

Cada uno de los sensores de la GY-80 tiene una dirección diferente, Arduino los llamará en un orden según se programe y obtendrá la información de los sensores.

Una vez se accede a cada sensor, estos tienen unos registros con direcciones de 2 dígitos hexadecimales lo cuales pueden ser de escritura o de lectura. Al comienzo del programa se inicializará cada sensor escribiendo en el registro correspondiente para configurarlo. Posteriormente para leer la información del sensor se accederá a los registros donde se encuentran los datos de salida.

A continuación se hablará sobre los diferentes sensores que tiene la GY-80. En el Anexo 2 se adjuntan todas las datasheets de estos sensores, las cuales han sido de vital ayuda a la hora de realizar la programación, dando la información necesaria para utilizar correctamente los registros y direcciones.

3.3.1 ADXL345

El ADXL345 es un pequeño chip (3mm x 5mm x 1mm) que entra dentro de la categoría de sensores MEMS, y que tiene un acelerómetro de 3 ejes de alta resolución, de hasta 13 bits, para medidas que pueden llegar a rangos de ±16g de aceleración (unos ±157 m/s^2).

El rango de valores medidos por el acelerómetro se puede establecer mediante programación, el ADXL345 tiene 4 rangos posibles de funcionamiento, ±2g (10 bits), ±4g (11 bits), ±8g (12 bits), y ±16g (13 bits).

El formato de los datos que podemos obtener de ella son 2 bytes en complemento a 2, para cada acelerómetro tiene 2 registros de 8 bits que hay que concatenar adecuadamente para obtener el valor de 16 bits con la medida del acelerómetro. Esta medida necesitará ser escalada posteriormente mediante una constante que dependerá del rango de valores utilizados, y que se obtendrá de la Datasheet del ADXL345.

La frecuencia de salida de datos es variable y programable, su rango varía entre 0.1 Hz y 3200 Hz, cuanto menor sea la frecuencia menor será el consumo y menor será el ruido de los datos de salida. Por lo tanto las altas frecuencias solo se recomiendan usar para aplicaciones en las que se necesitan una gran frecuencia de datos, en aplicaciones normales será recomendable usar frecuencias en torno a los 100 Hz, las cuales tienen un ruido notablemente menor.

La comunicación con este dispositivo se puede realizar mediante SPI o mediante I2C.

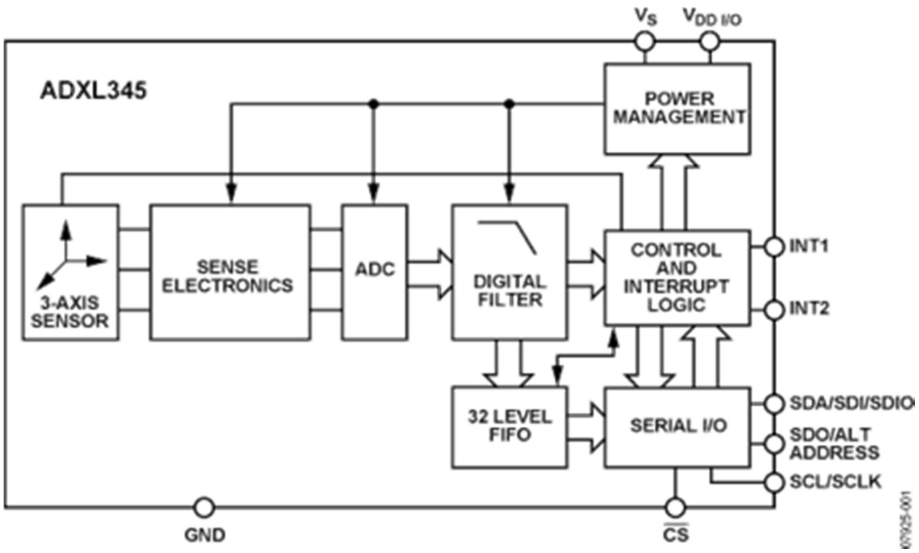


Figura 3.12: Esquema funcional del ADXL345

El pequeño tamaño del chip lo hace propicio para ser utilizado en aplicaciones móviles, puede medir la aceleración estática y las aceleraciones dinámicas a las que se vea sometido. Su alta resolución hace que pueda medir variaciones de la inclinación con ángulos menores de 1º.

Además de dar los datos de la aceleración en tres ejes perpendiculares también tiene algunas funciones especiales, como pueden ser la de detección de movimiento, la de detección de pequeños golpes individuales (tap) o dobles (double tap), o la detección de caída libre.

Dispone también de un modo de bajo consumo para aplicaciones donde necesita estar siempre encendido pero no siempre en uso.

Los registros que se utilizarán para configurar el sensor son los siguientes:

- BW RATE (0x2C): Con este registro se puede iniciar un modo de bajo consumo, el cual no se utilizará porque aumenta el ruido de las mediciones. También es el registro que se utiliza para seleccionar el ancho de banda de los datos de salida. Se configurará como 0x09, lo cual significa que se utilizará una frecuencia de 50Hz (ancho de banda de 25Hz).
- POWER CTL (0x2D): Este registro se utiliza para encender el sensor y asegurarse que funciona en modo normal. Se ha configurado en 0x08, lo cual implica que está en modo de toma de mediciones.
- DATA FORMAT (0x31): Con este registro fijamos el rango de valores que los 3 ejes del acelerómetro medirán, que también van ligados con la sensibilidad del sensor. El valor escrito en el registro será

0x01, lo cual es un rango de +-4 g, y una sensibilidad aproximada de 128 LSB/g.

- DATA OUTPUTS (0x32-0x37): Estos 6 registros de lectura poseen los datos de los acelerómetros, 2 registros (16 bits) para cada eje del acelerómetro.

3.3.2 L3G4200D

El L3G4200D es un dispositivo digital de baja potencia que dispone de un giróscopo de 3 ejes, sus características son, una alta estabilidad y una buena sensibilidad.

Tiene unas dimensiones de 4mm x 4mm x 1.1 mm, por lo que es muy pequeño y se puede utilizar en multitud de aplicaciones.

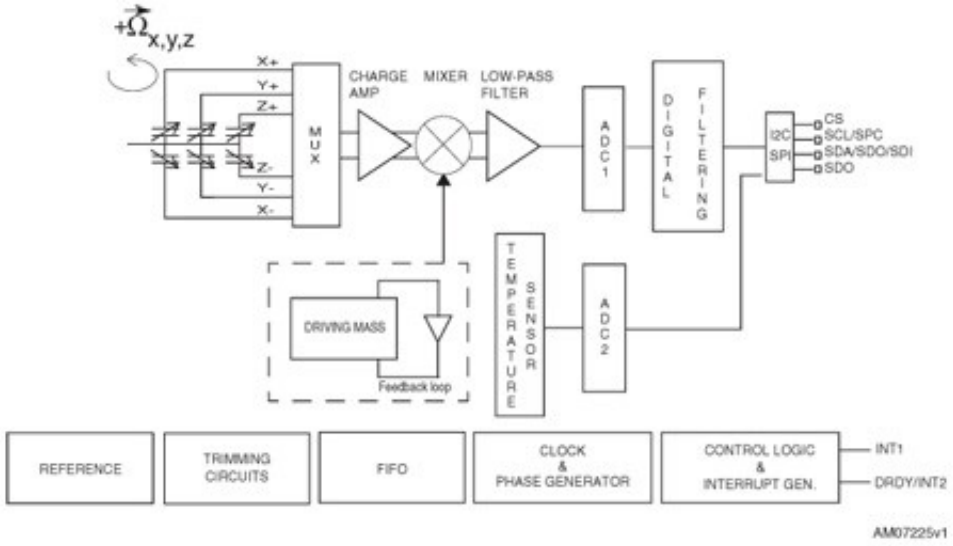


Figura 3.13: Diagrama funcional del L3G4200D

Las formas de comunicación que soporta son las mismas que el ADXL345, comunicación I2C y comunicación SPI.

Tiene hasta 3 escalas de datos con las que puede funcionar, 250 mº/s, 500 mº/s, y 2000 mº/s. La escala se escogerá en función de los valores que se vayan a manejar, ya que al utilizar el mismo número de bits para todas las escalas (16 bits), cuanto mayor sea la escala, peor será la sensibilidad del sensor.

Tiene incluido la posibilidad de activar un modo de funcionamiento de bajo consumo ("sleep-mode"). Y tiene un filtro de paso bajo y otro de paso alto con el ancho de banda seleccionable por programación.

Además de medir la velocidad de giro en cada eje, este chip también tiene un sensor temperatura que nos da la temperatura mediante 8 bits.

En cuanto a los registros que se utilizan para configurar el sensor y obtener los datos de los giróscopos:

- REG 1 (0x20): Este registro es necesario configurarlo para encender el dispositivo en modo normal, y habilitar los 3 sensores. Además con él se selecciona la frecuencia de salida y el ancho de banda, los cuales se escogen los más bajos ya que serán suficientes y reducen el ruido. Este registro se ha decidido configurar en 0x0F.
- REG 2 (0x21): Registro para inicializar los filtros que tiene disponibles el sensor, como no se va a utilizar ninguno se configura como 0x00.
- REG 3 (0x22): Con este registro se configuran diferentes opciones para el correcto funcionamiento del dispositivo, se configura como 0x08.
- REG 4 (0x23): Registro utilizado para fijar la escala de datos utilizada, se configura como 0x30 para utilizar la máxima escala (2000º/s). Esta configuración también fija la sensibilidad de los 3 ejes del giróscopo.
- REG 5 (0x24): Otro registro para activar funciones que en este caso no se utilizarán, por lo que se escribe en el registro 0x00.
- DATA OUTPUT (0x28-0x2D): Datos de salida, 6 registros los cuales se utilizan 2 para cada eje del giróscopo.

3.3.3 HMC5883L

Este chip MEMS es capaz de medir campos magnéticos en 3 ejes, incluyendo el campo magnético terrestre. Tiene unas dimensiones de 3mm x 3mm x 0.9mm que al igual que los chips anteriores lo convierten en una buena opción para incluirlo en dispositivos móviles.

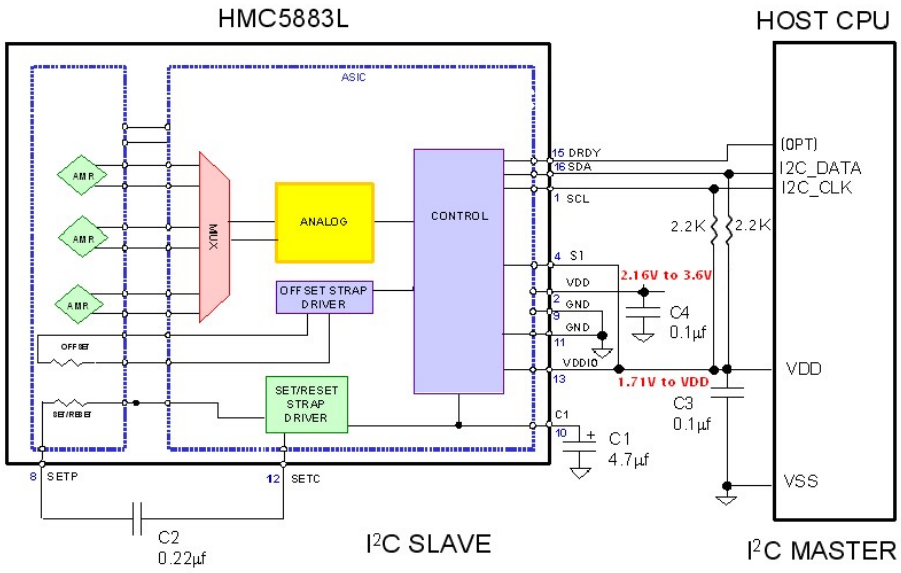


Figura 3.14: Diagrama funcional del HMC5883L

Posee un convertidor digital-analógico para las mediciones, y tiene una alta resolución ya que utiliza 16 bits para cada medición. Como el ADXL345 y el L3G4200D, tiene 2 salidas de 8 bits para cada uno de sus 3 ejes, estos 2 bytes hay que concatenarlos para obtener la medida en 16 bits.

Esta resolución llega a ser de 0.92 mili gauss para el rango mínimo que es de ±0.88 gauss. El rango máximo de campos magnéticos que este sensor puede medir es de ±8 gauss, para el cual tiene una resolución de 4.35 mili gauss.

Como los otros sensores de la IMU GY-80, la comunicación que utiliza es I2C. Y tiene un bajo consumo.

En cuanto a los registros que se configurarán para hacer funcionar el magnetómetro:

- CONFIG A (0x00): En este registro se puede cambiar la frecuencia de datos de salida, la cual utilizaremos la que viene por defecto (15 Hz), además se puede hacer que el dispositivo devuelva a la salida un valor medio entre varias mediciones tomadas entre cada iteración del programa. Se configurará como 0x10.
- CONFIG B (0x01): Este registro es utilizado únicamente para seleccionar el rango de valores a medir por el magnetómetro, desde los 0.88 Gauss a los 8.1. Esta selección también implica cambiar la sensibilidad del sensor. Se utilizará la configuración 0xE0 para obtener el máximo rango de datos.

- MODE (0x02): Con este registro se pondrá el magnetómetro en modo normal, obteniendo una medición cada vez que se consulte el registro debido. Para tal objetivo se configura en 0x00.
- DATA OUTPUT (0x03 – 0x08): Al igual que en el resto de sensores se tienen 6 registros para los datos de salida, 2 para cada eje, la peculiaridad del HMC5883L es que están ordenador de forma distinta, siendo los 2 primeros registros para el eje X, los 2 siguientes para el eje Z, y los 2 últimos para el eje Y.

3.3.4 BMP085

El BMP085 es un barómetro digital de pequeño tamaño (5mm x 5mm x 1.2mm), que también tiene un sensor de temperatura.

Su tamaño y su bajo consumo lo optimizan para el uso en dispositivos móviles. Y como en todos los chips de la GY-80 utiliza comunicación I2C para transmitir los datos.

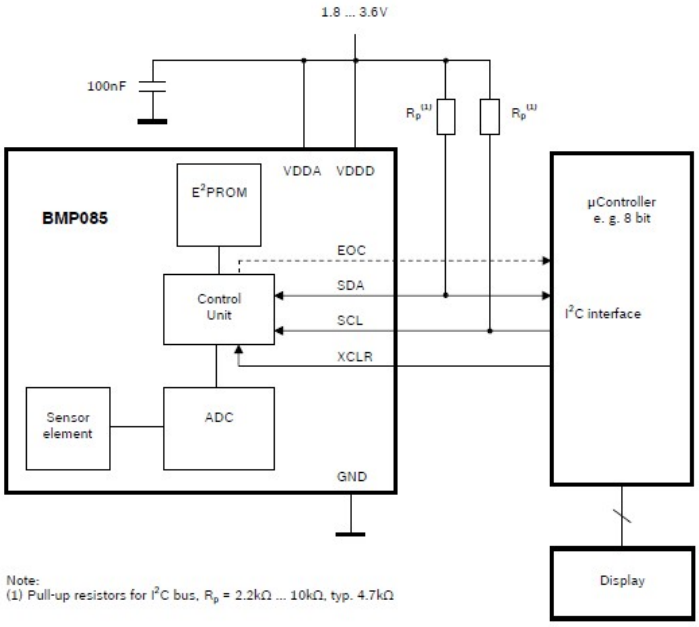


Figura 3.15: Diagrama funcional del BMP085

A diferencia de los sensores anteriores, este dispositivo también realiza unas lecturas necesarias para que el usuario pueda hacer una calibración mediante programación.

Debido a esto su programación es más compleja que las anteriores, ya que se necesitan medir 11 parámetros adicionales para la calibración o compensación de las medidas.

Las salidas de este sensor son 3 bytes para la presión sin compensar, y 2 bytes para la temperatura sin compensar.

En el caso del BMP085 no hay registros de escritura para configurar el dispositivo, sino que todos son de lectura, se utilizan muchos registros para obtener unos datos que luego se utilizan en el cálculo de la presión y la temperatura. Estos registros van desde el 0xAA al 0xBF tal y como se puede ver en la figura 3.16.

	BMP085 reg adr	
Parameter	**MSB**	**LSB**
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

Figura 3.16: Registro para la compensación del BMP085

También las lecturas funcionan de manera diferente, ya que por ejemplo los datos de presión y temperatura (antes de la compensación) se obtienen de los mismos registros, pero unos milisegundos (hay que programar un retraso) antes de realizar la lectura se debe escribir en el registro CONTROL (0xF4).

Es por este motivo que el uso del BMP085 conlleva un retraso al programa, ya que se deben incluir "delays" en la programación. Estos retrasos tienen una magnitud de unos 8ms, que aunque pueda parecer poco, es un tiempo considerable comparado con lo que tarda el programa en realizar una iteración.

Debido a esto y a la poca o nula utilidad de las mediciones se decide no utilizar las mediciones del barómetro en el proyecto, aunque si estará programado por si en el futuro se necesitaran.

3.4 ERRORES Y CALIBRACIÓN DE SENSORES

Las medidas obtenidas por cualquier tipo de sensor pueden tener errores, estos errores pueden ser constantes o variables, y dependiendo del tipo pueden generar problemas en la utilización de los resultados obtenidos.

Todo sensor necesita ser calibrado para reducir los errores, para calibrar un sensor lo que se hace es tomar unas medidas en una situación

determinada en la que se conoce qué resultado se debiera obtener. A partir de la diferencia entre lo que se ha medido y lo que se debiera de haber medido se modifica posteriormente matemáticamente las mediciones para que se acerquen más a su valor real.

Los errores constantes se pueden deber a un offset, este error supone que cada medición esta desviada del valor real un cierto valor constante, y la solución es calcular este valor constante y restarlo en todas las mediciones futuras. Los errores constantes también pueden ser debidos a un error en la ganancia, lo cual se solucionaría multiplicando las mediciones futuras por una ganancia que se debe calcular. Estos 2 tipos de errores son los que se trataran de reducir mediante las pruebas de calibración.

Los errores variables son debidos al ruido propio de los sensores, este ruido hace que la medición se desvíe indistintamente hacia arriba o hacia abajo en un determinado rango. Este tipo de errores se suelen eliminar mediante un filtro, que reduce las desviaciones.

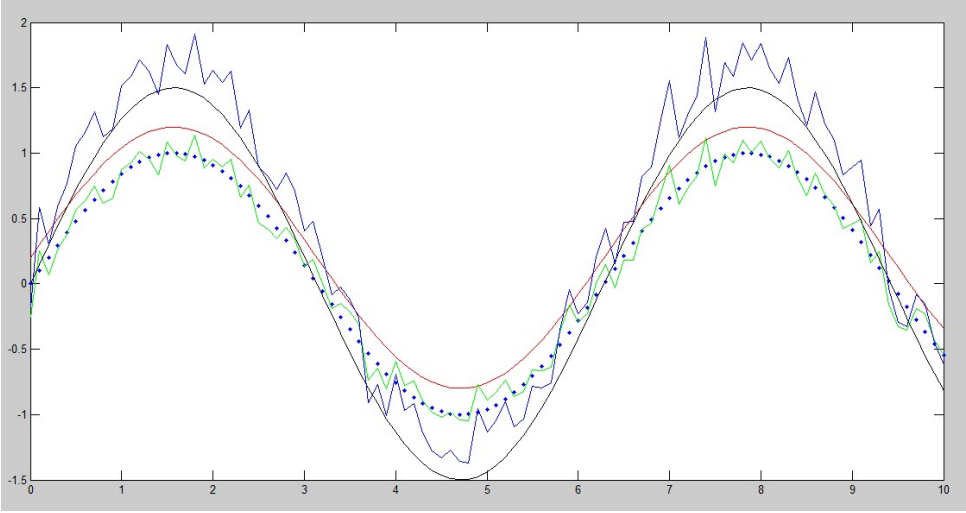


Figura 3.17: Representación de los tipos de errores

En la figura 3.17 se pueden ver los diferentes tipos de errores de los sensores, la señal real es la representada con puntos azules, que es una función seno.

La línea roja representa una medición de la variable real con offset, la línea negra representa la medición con un error de ganancia, y la línea verde es una medición con ruido. Por último la línea azul representa una medición con la suma de todos los errores anteriormente expuestos.

Además las medidas de los sensores son influidas por variables externas como la temperatura, dependiendo de la temperatura las medidas se desviaran más o menos del valor real. Por este motivo es recomendable

recalibrar los sensores cada cierto tiempo, sobre todo cuando se vaya a utilizar en condiciones de humedad o temperatura diferentes a las que con las que se calibró anteriormente.

3.4.1 Calibración del acelerómetro

El acelerómetro utilizado (ADXL345) tiene una buena calibración de fábrica, pero aun eso será necesario que se realice una para modificar la ganancia en cada eje.

El problema del ruido es fácilmente solucionable eligiendo un OSD (Output Data Rate) intermedio que no tenga mucho ruido.

Aunque el acelerómetro calcula automáticamente el offset de los 3 ejes y lo elimina en los datos que devuelve, se calculará un offset para restárselo a cada una de las mediciones hechas. Se puede comprobar el error de offset realizando mediciones alineando el vector gravitacional con uno de los ejes del acelerómetro, en sus 2 direcciones, si la medida positiva no coincide en valor absoluto con la negativa entonces es que hay un offset.

También habrá que calcular la ganancia para cada uno de los 3 ejes, ya que las medidas del acelerómetro utilizando la constante que nos da como típica la Datasheet no es del todo precisa. En las pruebas realizadas se comprueba que la ganancia en los tres ejes no es igual, y por lo tanto hay que obtener las 3 ganancias.

Los datos obtenidos antes de realizar la calibración tienen valores mayores de 10 m/s^2 para un estado estático, cuando en teoría el valor máximo debiera ser de 9'81 m/s^2.

La técnica de calibración que se realizará será tomar medidas en posiciones estáticas donde uno de los ejes es coincidente con el vector gravitacional (6 posiciones en total), midiendo así los valores máximos y mínimos en cada eje. En la figura 3.18 se pueden ver las aceleraciones mínimas y máximas obtenidas durante una prueba de calibración.

```
Aceleraciones minimas (X,Y,Z): -10.111; -9.958; -9.958;
Aceleraciones maximas (X,Y,Z): 9.728;   9.958;  9.498;
```

Figura 3.18: Valores máximos y mínimos del acelerómetro

Una vez se tengan estos valores se calculan los valores del offset y de compensación de la ganancia.

$$OffsetX = \frac{(MaxX+MinX)}{2} \quad (3.3)$$

$$OffsetY = \frac{(MaxY+MinY)}{2} \quad (3.4)$$

$$OffsetZ = \frac{(MaxZ+MinZ)}{2} \quad (3.5)$$

$$GananciaX = \frac{|g|}{(MaxX-OffsetX)} \quad (3.6)$$

$$GananciaY = \frac{|g|}{(MaxY-OffsetY)} \quad (3.7)$$

$$GananciaZ = \frac{|g|}{(MaxZ-OffsetZ)} \quad (3.8)$$

Una vez tengamos estos valores, se deben compensar los datos obtenidos por los acelerómetros.

$$AccX = (X - OffsetX) * GananciaX \quad (3.9)$$

$$AccY = (Y - OffsetY) * GananciaY \quad (3.10)$$

$$AccZ = (Z - OffsetZ) * GananciaZ \quad (3.11)$$

Donde X, Y y Z son los datos de las aceleraciones en cada eje sin aplicar la calibración del sensor, es decir, las obtenidas del ADXL345.

3.4.2 Calibración del giróscopo

Las medidas por los giróscopos tienen un error de offset, esto provoca que por ejemplo cuando el sensor este totalmente quieto se obtengan unas medidas de cierto movimiento, que evidentemente es incorrecto.

El método de calibración que utilizaremos para eliminar el offset será tomar un número relativamente alto de medidas mientras la IMU está totalmente estática, y hacer la media aritmética para cada eje de estas mediciones, hecho esto determinamos que el resultado obtenido es el offset para cada eje, y por lo tanto habrá que restárselo a cada medición futura.

Las medidas así obtenidas serán de mayor calidad y los resultados obtenidos serán más fiables.

Por ejemplo el giróscopo del eje x tiene un offset algo más de 1º, por lo tanto aun estando la IMU totalmente quieta, el programa siempre detecta un giro en el eje x inexistente, que no es demasiado, pero al ser un valor que debe integrarse posteriormente, que cada vez que el programa realiza una iteración este error se va acumulando y creciendo. Con el cálculo y eliminación del offset esto se reduce a un 10%, quedando ahora un ruido con valores positivos y negativos que más o menos se pueden ir compensando en cada iteración.

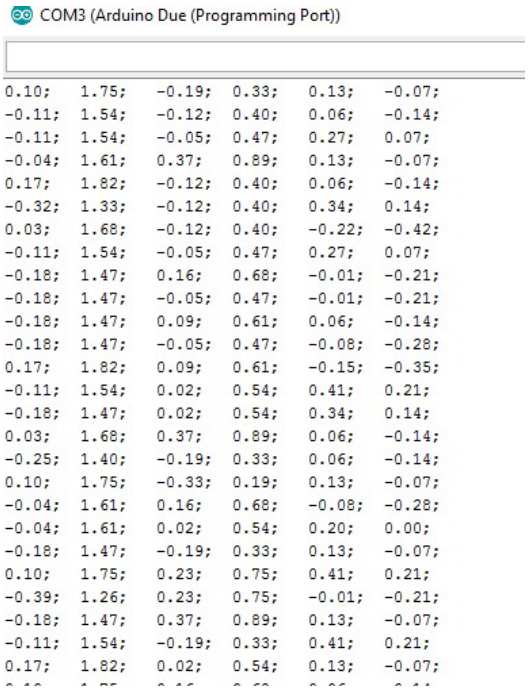


Figura 3.19: Datos giróscopo con/sin offset

En la figura 3.19 se pueden ver los datos obtenidos por el giróscopo antes y después de aplicar la reducción del offset, en la primera columna están las mediciones del giro en el eje X eliminando el offset, y en la segunda columna los datos obtenidos directamente por el eje X del giróscopo. En el resto de columnas se hace lo mismo pero con los ejes Y y Z.

Como se puede apreciar los valores absolutos de las columnas impares son mucho menores que los de las columnas pares, esto confirma que los resultados (al menos para movimientos pequeños o nulos serán en su mayoría los que se llevaran a cabo en el vehículo autónomo), serán de mayor precisión.

En este caso solo se podrá calibrar el offset y no la ganancia, ya que para calibrar esta habría que realizar giros a una velocidad constante conocida, y no se dispone de medios para realizar esta prueba, por lo tanto se utilizará la ganancia dada en la Datasheet del giróscopo.

3.4.3 Calibración del magnetómetro

El magnetómetro también es un sensor que sufre problemas con el offset si no está calibrado, se puede comprobar haciendo mediciones y viendo que el máximo positivo es distinto al máximo negativo de los valores medidos por un eje del magnetómetro, cuando debieran ser iguales.

Además también se puede comprobar que existe una diferencia entre los valores máximos y mínimos en los 3 ejes, siendo el que campo magnético

debiera afectar de igual manera a los 3, por lo tanto también hace falta calibrar con una ganancia distinta los 3 ejes del magnetómetro.

La prueba de calibración consiste en tomar medidas del campo magnético mientras giramos la IMU, intentando tener medidas en todas las orientaciones posibles. Para realizar estas mediciones se debe tratar de evitar las interferencias de otros campos magnéticos como los que pueden provocar el Arduino o el ordenador.

Al igual que hicimos con el acelerómetro se calculara el offset y la ganancia con los máximos y mínimos obtenidos.

$$OffsetX = \frac{(MaxX+MinX)}{2} \tag{3.12}$$

$$OffsetY = \frac{(MaxY+MinY)}{2} \tag{3.13}$$

$$OffsetZ = \frac{(MaxZ+Min\ )}{2} \tag{3.14}$$

$$GananciaX = \frac{|\boldsymbol{m}|}{(MaxX-OffsetX)} \tag{3.15}$$

$$GananciaY = \frac{|\boldsymbol{m}|}{(MaxY-Offse\ \ )} \tag{3.16}$$

$$GananciaZ = \frac{|\boldsymbol{m}|}{(MaxZ-OffsetZ)} \tag{3.17}$$

Donde |**m**| es el valor máximo del campo magnético terrestre que en teoría debemos obtener, que es de unos 0.65 Gauss.

De igual manera que se hace con el acelerómetro se calculan los valores del campo magnético:

$$MagX = (X - OffsetX) * GananciaX \tag{3.18}$$

$$MagY = (Y - OffsetY) * GananciaY \tag{3.19}$$

$$MagZ = (Z - OffsetZ) * GananciaZ \tag{3.20}$$

Para la prueba de calibración del magnetómetro se ha almacenado los datos obtenidos por Arduino para graficarlos en matlab. Haciendo esto y si las medidas fueran del todo precisas, debiéramos observar como los puntos se posicionan formando una esfera con centro en el punto (0,0,0), pero no es así, el centro esta desplazado (debido al offset), y los puntos no forman una esfera perfecta (debido a la imprecisión de las medidas).

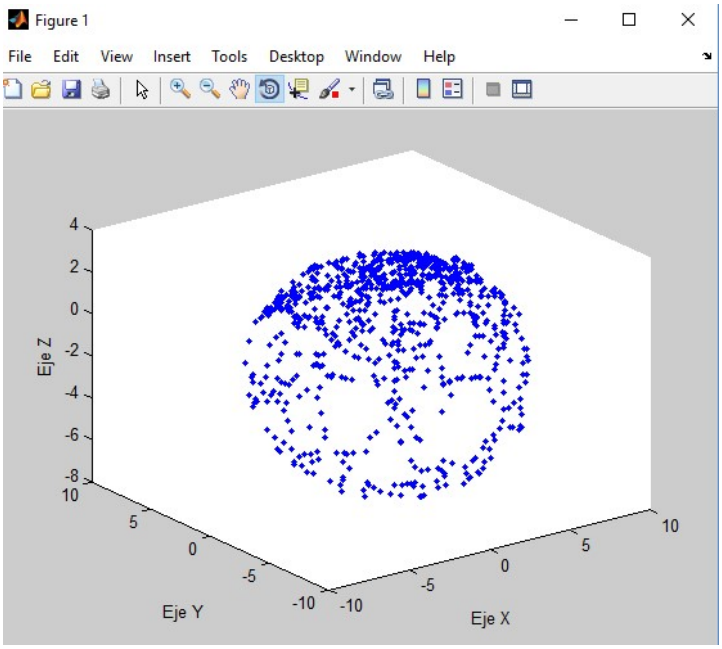


Figura 3.20: Representación de los datos del magnetómetro sin calibrar

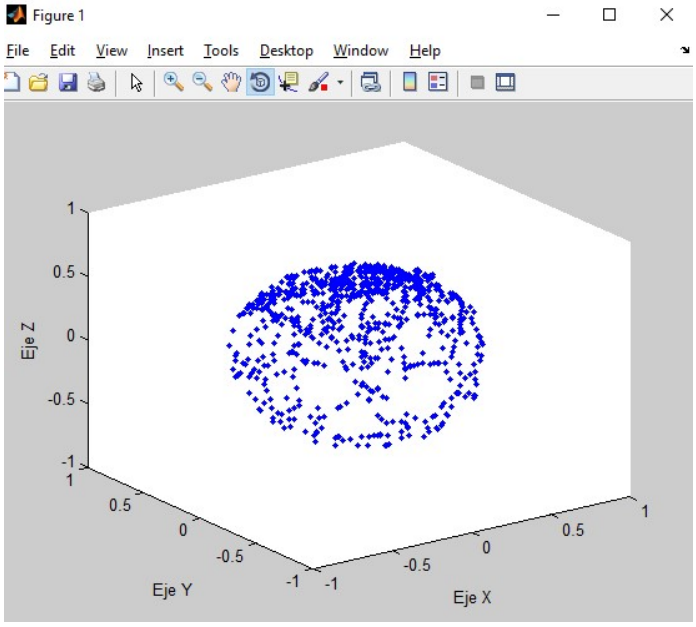


Figura 3.21: Representación de los datos del magnetómetro calibrados

En las figuras 3.20 y 3.21 se muestran las representaciones de los datos recogidos de los magnetómetros, antes y después de la calibración. Después de realizar la calibración la esfera se puede ver que está centrada en el (0, 0, 0).

CAPÍTULO 4: ÁNGULOS DE EULER

Los ángulos de navegación son una variante de los ángulos de Euler que son utilizados principalmente para la aeronáutica, ya que representan de la forma más sencilla la orientación de un avión en el espacio. Estos ángulos se llaman guiñada, cabeceo y alabeo (Yaw, pitch y roll respectivamente), se hablará de ellos con sus nombres en inglés, ya que son mucho más utilizados en la bibliografía. En la figura 4.1 se pueden apreciar cuales serían los ángulos de Euler en un avión. En el caso de un coche son los mismos.

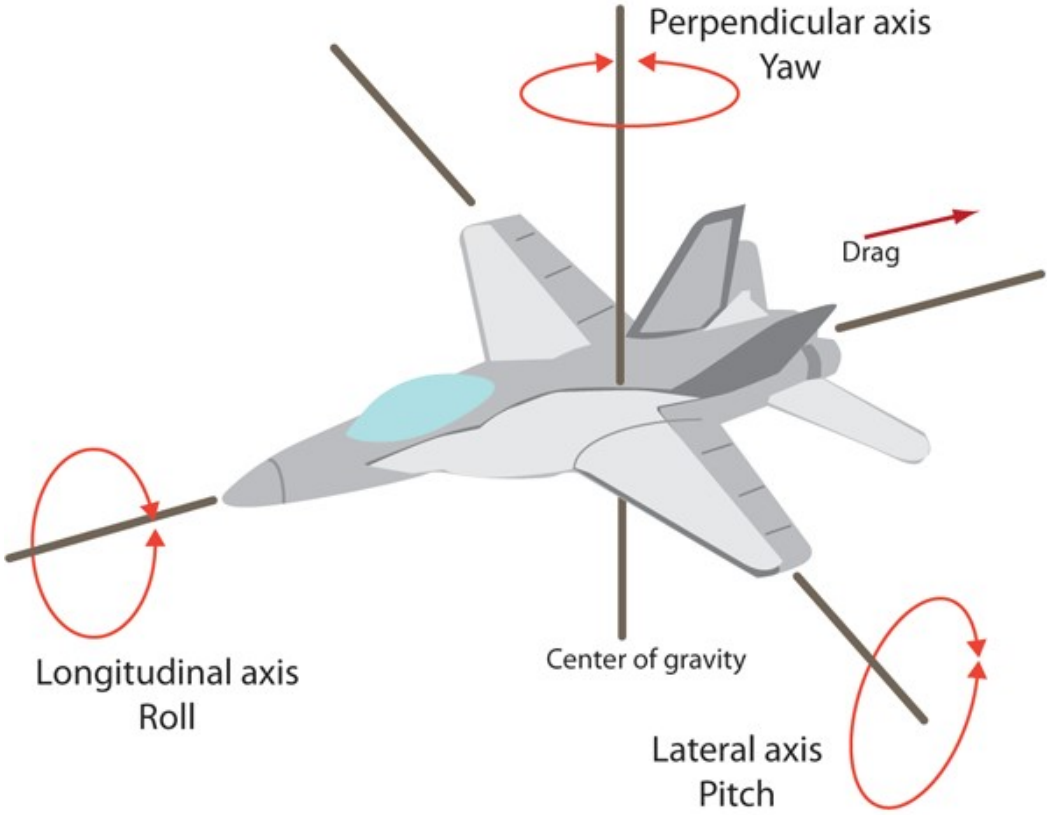


Figura 4.1: Representación de los ángulos de Euler en un avión

Cada uno de los 3 ángulos de navegación tiene un rango de datos distinto, debido a sus peculiaridades.

El pitch es un ángulo que solo varía entre -90 y 90º, ya que no se considera que el vehículo realice un giro mayor en ese eje, en el caso de que lo haga se consideraría como una combinación de giros respecto al roll y/o al yaw.

El rango del roll sin embargo puede considerarse desde los -180º a los 180º. Es evidente que en el caso de un vehículo terrestre los ángulos de pitch y roll variarán entre rangos mucho menores que estos.

El ángulo del yaw tiene el mismo rango que el roll (-180º,180º) aunque también se puede utilizar otro rango distinto de valores, desde los 0º, hasta los 360º. El yaw es el ángulo que podríamos obtener con una brújula y nos mide el ángulo con respecto el norte.

Los ángulos de Euler son una forma de representación de la orientación en un espacio tridimensional con solo 3 componentes, a diferencia de otras formas de representación de la orientación, como son las matrices de rotación o matrices de cosenos directores que utilizan 9 componentes, y los cuaterniones que utilizan 4 parámetros, los ángulos de Euler entonces suponen una simplificación considerable en cuanto a la representación, aunque implican problemas en cuando se realizan operaciones de giros, como se verá posteriormente.

Todo sistema OUVW solidario a un cuerpo cuya orientación se busca definir, se puede hacer respecto a otro sistema OXYZ mediante tres ángulos Φ, θ, y ψ denominados los ángulos de Euler. Estos ángulos representan los valores de los giros que se realizan sobre tres ejes ortogonales entre sí, de modo que si giramos el sistema OXYZ sobre estos ejes los valores de los ángulos de Euler, obtendremos el sistema OUVW. Por eso es necesario además de los valores de los ángulos, conocer los ejes sobre los que se realizan las rotaciones, debido a esto se consideran varias variantes de los ángulos de Euler, siendo una de ellas la de los ángulos de navegación que en este proyecto se utilizarán.

Los ángulos de navegación también son llamados ángulos de Euler "XYZ" para diferenciarlos de las otras variantes, como son los ángulos de Euler "WUW" o los "WVW".

Uno de los problemas que suelen surgir al realizar operaciones de orientaciones mediante los ángulos de Euler es de la aparición de singularidades, las cuales se dan cuando el "pitch" alcanza los valores de ±90º, entonces el eje longitudinal del objeto es paralelo al eje del Yaw, comportándose de la misma manera, esto es lo que se conoce como el "bloqueo de Cardan" o el "Gimbal Lock", donde se pierde un grado de libertad y que provoca grandes problemas en la representación de la orientación. En la figura 4.2 se muestra una imagen de la situación denominada Gimbal Lock.

Es por este motivo que para los cálculos se suele acudir a la utilización de métodos más complejos los cuales evitan la aparición de singularidades, como se verá posteriormente.

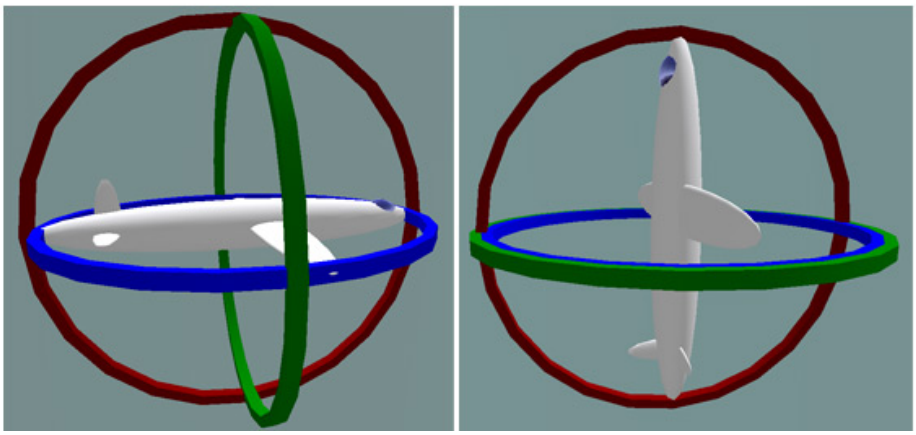


Figura 4.2: Situación de un avión en posición normal (izquierda), y con "Gimbal Lock" (derecha)

Para utilizar la IMU de forma más intuitiva y que quede la cara frontal del dispositivo hacia arriba cuando el pitch y el roll son de 0º, se han cambiado los ejes de a GY80 en la programación, de modo que se utilizará el eje X como el eje Y, y viceversa, y se ha cambiado el sentido al eje Z. Para esto únicamente se ha modificado en el código los datos obtenidos de los sensores.

Para el cálculo de los 3 ángulos de Euler será necesaria la utilización de los datos obtenidos a partir de los sensores de la IMU, hay varias formas de operar con estos ángulos, a continuación se explicarán dos de ellas: La matriz de cosenos directores y los cuaterniones.

Al ejercicio de utilizar los sensores de una IMU para calcular la orientación de esta (y por lo tanto de un objeto ligado a la IMU) se le llama AHRS (Attitude and Heading Reference Systems).

5.1 MATRIZ DE COSENOS DIRECTORES (DCM)

La matriz de cosenos directores (DCM de sus siglas en inglés, Direction Cosine Matrix), es una forma habitual de representar la orientación de un objeto en el espacio utilizando un sistema de referencia, de esta matriz de 3x3 se puede obtener cual es la orientación de un objeto con respecto a un sistema de referencia fijado.

Los puntos fuertes de la matriz DCM son su capacidad de cálculo, su facilidad de realizar operaciones matriciales y la ausencia de singularidades que por ejemplo presenta una representación mediante los ángulos de Euler.

El principal problema de la DCM es que no es intuitiva, y por ello a partir de la matriz DCM se suelen obtener los ángulos de Euler para representar una orientación.

En este método se utilizan también unas matrices de rotación 3x3 que se utilizan para "girar" la matriz DCM en el espacio. Esta rotación consistirá en un conjunto de 3 rotaciones alrededor de cada uno de los ejes de los que cuenta la IMU, que multiplicando la matriz DCM por esta matriz de rotación, se obtendrá la nueva matriz DCM que contiene los nuevos valores de los ángulos de Euler.

Las matrices de rotación utilizadas incluyen 3 rotaciones respecto a los 3 ejes, estas 3 rotaciones se tienen que realizar en un orden determinado, cambiar ese orden significa cambiar la matriz de rotación final. Esto establece una jerarquía entre los 3 ejes, dependiendo de la secuencia elegida, uno de los ejes tendrá mayor importancia que los otros 2, y otro de los restantes tendrá mayor importancia que el tercero.

El motivo de que el resultado depende del orden de las rotaciones debido a que tanto las rotaciones como las multiplicaciones de matrices no son operaciones conmutativas, y por tanto el orden es muy influyente, esto es más notable cuanto mayores sean las rotaciones, por lo tanto será importante tratar de reducir el periodo de muestreo reduciendo el código del programa que se aplica en cada iteración del "Loop".

Dado un vector que defina la orientación inicial de un objeto, mediante la matriz de rotación adecuada se puede calcular la nueva orientación del objeto.

La matriz DCM es una matriz de rotación, esta rotación es la que se lleva a cabo para girar desde la posición original hasta la posición que la DCM representa.

Las matrices de rotación básicas respecto a los 3 ejes que se utilizarán para rotar la matriz DCM son las siguientes:

$$\boldsymbol{R_x}(\Phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\Phi) & -\sin(\Phi) \\ 0 & \sin(\Phi) & \cos(\Phi) \end{pmatrix} \tag{4.1}$$

$$\boldsymbol{R_y}(\theta) = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \tag{4.2}$$

$$\boldsymbol{R_x}(\Phi) = \begin{pmatrix} \cos(\Phi) & -\sin(\Phi) & 0 \\ \sin(\Phi) & \cos(\Phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

Existen 6 posibilidades distintas de realizar la rotación, y como se ha comentado antes, todas las opciones dan un resultado distinto, la secuencia que se utilizará es la de los ejes Z, Y, y X. Con esta secuencia la matriz de rotación resultante es la siguiente:

$$\boldsymbol{DCM} = \boldsymbol{R} = \boldsymbol{R_z}(\psi)\,\boldsymbol{R_y}(\theta)\,\boldsymbol{R_x}(\Phi) =$$

$$= \begin{pmatrix} \cos(\Phi)\cdot\cos(\psi) & cos(\psi)\cdot\sin(\theta)\cdot sin(\Phi) - \sin(\psi)\cdot\cos(\Phi) & \cos(\psi)\cdot\sin(\theta)\cdot\cos(\Phi) + \sin(\psi)\cdot\sin(\Phi) \\ \cos(\Phi)\cdot\sin(\psi) & \cos(\psi)\cdot\cos(\Phi) + \sin(\psi)\cdot\sin(\theta)\cdot\sin(\Phi) & \sin(\psi)\cdot\sin(\theta)\cdot\cos(\Phi) - \cos(\psi)\cdot\sin(\Phi) \\ -\sin(\Phi) & \cos(\theta)\cdot\sin(\Phi) & \cos(\theta)\cdot\cos(\Phi) \end{pmatrix} \tag{4.4}$$

Nótese que la última fila es totalmente de independiente del ángulo del Yaw (ψ), lo cual será de gran importancia a la hora de aplicar un filtro con los valores de los acelerómetros.

Es importante diferenciar entre los ángulos de Euler que son el objetivo calcular, y los ángulos de rotación de la IMU que son los que se pueden obtener a partir de los giróscopos, y que serán el punto de partida en los cálculos.

La matriz DCM es una matriz que tiene información de la orientación relativa entre el sistema de referencia absoluto (ángulos de Euler) y el sistema de referencia de la IMU.

Con la matriz DCM se tienen 9 parámetros para representar una orientación que puede ser representado con solo 3, por lo cual es intuitivo pensar que no cualquier matriz 3x3 cumple los requisitos para ser una DCM. Las 2 propiedades que debe cumplir toda matriz DCM son:

-Es una matriz ortogonal.

$$\boldsymbol{DCM}^{-1} = \boldsymbol{DCM}^{T} \tag{4.5}$$

-El determinante de la matriz DCM es ± 1. Por lo tanto sus filas son vectores linealmente independientes.

$$\det(\boldsymbol{DCM}) = 1 \tag{4.6}$$

Estas propiedades habrá que tenerlas en cuenta a la hora de operar con las matrices, y serán útiles pare reacondicionar la matriz en cada iteración.

4.2.1 Cálculo inicial

El primer cálculo de los ángulos de Euler se realizará únicamente con los datos obtenidos por los acelerómetros y los magnetómetros, y servirá como punto de partida para después actualizar los ángulos mediante los giróscopos.

Ya que no se conoce la matriz de cosenos directores, este cálculo llevará a obtener la primera matriz DCM, para después ir actualizándola en cada paso. Teniendo así los ángulos de Euler actuales en cada paso.

Debido a las propiedades de los acelerómetros, solo es posible calcular con ellos el Pitch y el Roll, y siempre en un estado estático, ya que si la IMU está en movimiento, las fuerzas dinámicas de aceleración, a las que estará sometida perturbaran la medición de los acelerómetros dando lugar mediciones incorrectas.

Para calcular el Pitch y el Roll con los acelerómetros hay que realizar una rotación teórica desde la posición inicial de reposo donde los ángulos de Euler son nulos, a una posición cuya orientación es desconocida, pero se conocen los valores de los acelerómetros. Para ello hay que multiplicar la matriz de rotación por el vector de aceleraciones inicial, cuya única componente es la del eje Z, siendo el resultado de esta multiplicación el vector de aceleraciones **A**.

$$\boldsymbol{A_{inic}} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \tag{4.7}$$

$$\boldsymbol{A} = \begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix} \tag{4.8}$$

La ecuación resulta la siguiente:

$$\frac{\boldsymbol{A}}{|A|} = \boldsymbol{R} \cdot \frac{\boldsymbol{A_{inic}}}{|A|} = \boldsymbol{R} \cdot \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \end{pmatrix} = \begin{pmatrix} \cos(\psi) \cdot \sin(\theta) \cdot \cos(\Phi) + \sin(\psi) \cdot \sin(\Phi) \\ \sin(\psi) \cdot \sin(\theta) \cdot \cos(\Phi) - \cos(\psi) \cdot \sin(\Phi) \\ \cos(\theta) \cdot \cos(\Phi) \end{pmatrix} \tag{4.9}$$

Como esta rotación parte de un momento en el que el sistema de referencia es el mismo, los ángulos de la IMU y los ángulos de Euler coinciden.

Pudiendo resolver únicamente los 2 valores del Pitch y el Roll, despejando la ecuación se obtienen las siguientes fórmulas para calcularlos a partir de los valores del acelerómetro.

$$\Phi = \operatorname{atan}\left(\frac{-A_x}{\sqrt{A_y^2 + A_z^2}}\right) \tag{4.10}$$

$$\theta = \operatorname{atan}\left(\frac{A_y}{Az}\right) \tag{4.11}$$

Estas 2 ecuaciones también pueden ser obtenidas por trigonometría.

Nótese que en las ecuaciones anteriores, el pitch estará siempre comprendido en el rango de -90º a 90º, mientras que el roll tendrá valores entre -180º y 180º.

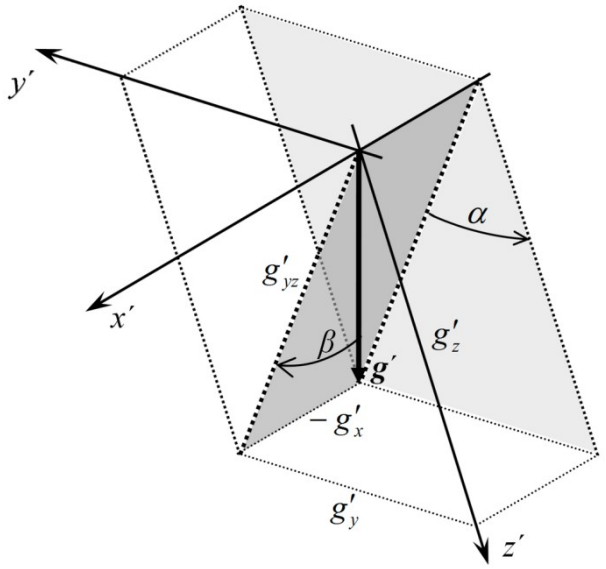


Figura 4.3: Representación del vector gravitacional en un sistema

En la figura 4.3 se pueden ver los ángulos del pitch y del roll formados al tener una inclinación con respecto el eje gravitacional.

Para el cálculo del Yaw se necesitarán utilizar además los valores obtenidos por los magnetómetros, en modo de brújula o compas magnético, obtendremos la orientación respecto los puntos cardinales.

$$\psi = -\text{atan}\left(\frac{-M_y \cdot \cos(\theta) + M_z \cdot \sin(\theta)}{M_x \cdot \cos(\Phi) + M_y \cdot \sin(\theta) \cdot \sin(\Phi) + M_z \cdot \cos(\theta) \cdot \sin(\Phi)}\right) \qquad (4.12)$$

Nótese en la ecuación 4.12 que en el caso de que la IMU estuviera en posición horizontal (sin inclinación), los valores de los senos del pitch y del roll de la ecuación anterior serían nulos y por lo tanto quedaría una ecuación más sencilla:

$$\psi = -\text{atan}\left(\frac{-M_y}{M_x}\right) \qquad (4.13)$$

Este cálculo del Yaw se puede compensar utilizando la declinación magnética que depende de la ubicación geográfica de donde se realicen las pruebas. En este caso debido a que no es necesaria una precisión tan elevada no se compensará la declinación magnética y este paso será obviado.

Con estos 3 ángulos ya se puede definir totalmente la primera matriz DCM que será la matriz de rotación con respecto a la matriz identidad.

$$\boldsymbol{DCM} = \\ = \begin{pmatrix} \cos(\Phi)\cdot\cos(\psi) & cos(\psi)\cdot\sin(\theta)\cdot sin(\Phi) - \sin(\psi)\cdot\cos(\Phi) & \cos(\psi)\cdot\sin(\theta)\cdot\cos(\Phi) + \sin(\psi)\cdot\sin(\Phi) \\ \cos(\Phi)\cdot\sin(\psi) & \cos(\psi)\cdot\cos(\Phi) + \sin(\psi)\cdot\sin(\theta)\cdot\sin(\Phi) & \sin(\psi)\cdot\sin(\theta)\cdot\cos(\Phi) - \cos(\psi)\cdot\sin(\Phi) \\ -\sin(\Phi) & \cos(\theta)\cdot\sin(\Phi) & \cos(\theta)\cdot\cos(\Phi) \end{pmatrix}$$

(4.14)

4.2.2 Actualización de la matriz con los giróscopos

La matriz DCM calculada en un primer momento solo se usará para establecer la orientación inicial de la IMU, a partir de ahí cada vez que el programa cargado en Arduino realiza el código que hay en el lazo principal, la matriz DCM se actualiza utilizando los datos que se obtienen de los giróscopos.

Como la continua integración de estos datos llevará a un error cada vez mayor con el paso del tiempo (deriva o drift), también será necesario utilizar el resto de sensores inerciales de la IMU corrigiendo el valor que obtenemos con los giróscopos, esto se mostrará en el siguiente apartado. En este apartado se verá cómo sería utilizando únicamente los giróscopos, lo cual es una forma más sencilla para realizar este programa.

Como se dijo cuándo se habló de los giróscopos, una forma de reducir el error acumulado es tratar de calcular el offset de nuestros sensores y restárselo a los valores que vamos midiendo, de forma que la deriva será notablemente menor.

El primer paso es calcular la matriz de rotación de cada giro, para después multiplicarla por la matriz DCM previa, obteniendo la nueva matriz DCM que contiene los nuevos ángulos de Euler.

Siendo el vector **G** el vector con los datos obtenidos por los giróscopos, que son los 3 valores de la velocidad de giro en los 3 ángulos de la IMU.

$$\boldsymbol{G} = \begin{pmatrix} GIR_x \\ GIR_y \\ GIR_z \end{pmatrix} \tag{4.15}$$

Como **G** tiene unidades de velocidad de rotación (º/s), para obtener el ángulo girado se debe multiplicar ***G*** por una unidad de tiempo, este tiempo (Dt), es el periodo de muestreo, que será el que pasa entre cada actuación del bucle principal del programa cargado en el Arduino, y que es el tiempo entre cada cálculo de la matriz DCM. Este tiempo puede ser ligeramente variable a lo largo del funcionamiento del programa, así que es necesario calcularlo nuevamente para cada iteración. Cuanto menor sea este tiempo más preciso será el cálculo del ángulo girado en cada eje.

$$\boldsymbol{GIR} = \boldsymbol{G}\cdot Dt \tag{4.16}$$

La matriz de rotación necesaria para actualizar la matriz DCM se calcula de la forma siguiente:

$$\boldsymbol{ROT} = \begin{pmatrix} 1 & -GIR_z & GIR_y \\ GIR_z & 1 & -GIR_x \\ -GIR_y & GIR_x & 1 \end{pmatrix} \tag{4.17}$$

Y por lo tanto, la nueva matriz DCM será:

$$\boldsymbol{DCM}_i = \boldsymbol{DCM}_{i-1} \cdot \boldsymbol{ROT}_i \tag{4.18}$$

Siendo i el número de cada iteración o cada paso.

Este proceso, además de los errores generados por imprecisiones de los giróscopos, genera pequeños errores a cada paso, estos errores son errores de integración y errores cuantitativos, que provocarán que la matriz no sea exactamente la debida, y por lo tanto no cumpla las condiciones que debe cumplir una matriz DCM, por esto será necesario reacondicionar la matriz a cada iteración.

Los errores de integración son debidos a que comentemos la presunción que durante el tiempo de muestreo Dt, el objeto ha estado girando a una velocidad constante, esto no es así, la velocidad de giro es una función continua y sin saltos, de la que nosotros solo conocemos un valor cada un determinado tiempo, y asumimos que ese valor se mantiene durante Dt. Como esto no es así se genera un error de cálculo. Este efecto puede apreciarse en la figura 4.4.

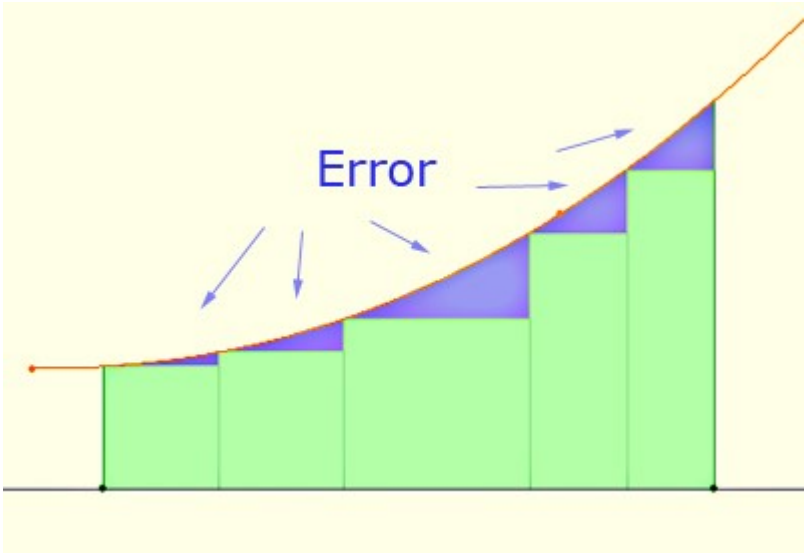


Figura 4.4: Error de integración

Por su parte el error cuantitativo se debe a que utilizamos valores finitos presentados por valores digitales que no son exactos, debido a la inexactitud de la conversión analógica digital.

Como se dijo anteriormente la DCM tiene 2 propiedades, la ortogonalidad y que las filas son linealmente independientes. Esto implica que todas las filas son perpendiculares entre si y que cada fila tiene longitud unidad. Los errores anteriormente citados pueden provocar que estas propiedades no se cumplan, por lo tanto, sabiendo esto, se debe forzar a que la nueva matriz DCM calculada cumpla ambas propiedades.

Para esta explicación se denominarán las filas de la DCM de la siguiente manera:

$$\boldsymbol{DCM} = \begin{pmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \\ \boldsymbol{Z} \end{pmatrix} \tag{4.19}$$

Donde **X**, **Y** y **Z** son las 3 filas de la matriz DCM.

Como la matriz tiene 9 elementos pero solo 3 pueden ser independientes, necesitaremos 6 restricciones para acondicionar la matriz. Estas Restricciones son las siguientes:

$$|\boldsymbol{X}| = 1 \tag{4.20}$$

$$|\boldsymbol{Y}| = 1 \tag{4.21}$$

$$|\boldsymbol{Z}| = 1 \tag{4.22}$$

$$\boldsymbol{X} \perp \boldsymbol{Y} \tag{4.23}$$

$$\boldsymbol{X} \perp \boldsymbol{Z} \tag{4.24}$$

$$\boldsymbol{Y} \perp \boldsymbol{Z} \tag{4.25}$$

El primer paso es calcular lo que se puede llamar el error de perpendicularidad entre las 2 primeras filas, para ello hacemos el producto escalar de ellas que si (el producto escalar es el coseno entre ambos vectores, en el caso de ser perpendiculares este debiera ser nulo).

Por lo tanto el error es:

$$error = \boldsymbol{X} \cdot \boldsymbol{Y} \tag{4.26}$$

Suponemos que el error se divide de igual manera entre los dos vectores, rotando estos en la dirección contraria al error, se obtienen entonces los nuevos valores para las 2 primeras filas (**X**′, **Y**′):

$$\begin{pmatrix} \boldsymbol{X}' \\ \boldsymbol{Y}' \end{pmatrix} = \begin{pmatrix} \boldsymbol{X} - \frac{error}{2} * \boldsymbol{Y} \\ \boldsymbol{Y} - \frac{error}{2} * \boldsymbol{X} \end{pmatrix} \tag{4.27}$$

El próximo paso será calcular la fila **Z** para que sea ortogonal a las anteriores, usando esta propiedad sabemos que las 3 filas son perpendiculares, por lo tanto la fila **Z** se obtiene haciendo el producto vectorial de las filas **X** e **Y** calculadas anteriormente.

$$\boldsymbol{Z'} = \boldsymbol{X'} \, x \, \boldsymbol{Y'} \tag{4.28}$$

Con esto se asegura que la matriz sea ortogonal.

El último paso será escalar la matriz para que todas las filas tengan longitud unidad. La forma más habitual de hacer esto sería dividiendo cada fila entre la su longitud, pero hay una forma más fácil, suponiendo que la desviación con respecto a la matriz que se busca es mínima, podemos utilizar la siguiente ecuación a partir de una expansión de Taylor:

$$\boldsymbol{DCM} = \begin{pmatrix} \boldsymbol{X''} \\ \boldsymbol{Y''} \\ \boldsymbol{Z''} \end{pmatrix} = \begin{pmatrix} 0.5 * (3 * (\boldsymbol{X'} \cdot \boldsymbol{X'})) * \boldsymbol{X'} \\ 0.5 * (3 * (\boldsymbol{Y'} \cdot \boldsymbol{Y'})) * \boldsymbol{Y'} \\ 0.5 * (3 * (\boldsymbol{Z'} \cdot \boldsymbol{Z'})) * \boldsymbol{Z'} \end{pmatrix} \tag{4.29}$$

Obteniendo así la nueva matriz DCM.

4.2.3 Filtro para eliminar el Drift

El "Drift" (o deriva) es la acumulación de error debido a las medidas ligeramente incorrectas de los giróscopos, en una iteración el drift es inapreciable, pero cuando se acumulan muchas iteraciones el error se va haciendo más notable según va pasando el tiempo.

Para evitar esto lo que se hace es utilizar otros sensores para compensar las mediciones de los giróscopos, estos sensores que se utilizarán serán los acelerómetros y los magnetómetros.

El método utilizado para esta compensación será un regulador PI, que ejercerá su regulación creando un giro adicional en la matriz de rotación, para ello se tiene que calcular la diferencia entre los ángulos que tenemos en la matriz DCM y los ángulos de Euler calculados con los acelerómetros y los magnetómetros a cada nueva iteración.

Se llevarán a cabo 2 regulaciones, una para los ángulos del pitch y el roll, y otro para el yaw. Cada una tendrá 2 constantes, una para la regulación proporcional y otro para la regulación integral, cuanto mayores sean estas constantes más rápido cambiará la matriz por los cambios medidos por acelerómetros y magnetómetros, o dicho de otra forma, más se "confiará" en los ángulos medidos por estos y menos en los medidos por los giróscopos.

Tal y como se hizo anteriormente en el primer cálculo de la matriz DCM, los acelerómetros servirán para ayudar en el cálculo del Pitch y del Roll, mientras que los magnetómetros se usarán en el reajuste del Yaw.

También sería posible calcular el Yaw mediante el GPS, ya que se puede suponer que el eje longitudinal coincide con la dirección de

desplazamiento que podemos obtener con el GPS, esto es más común realizarlo en proyectos con drones o aviones, ya que su velocidad a la que se desplazan es mayor e influye menos los errores de posicionamiento del GPS, en el caso de nuestro proyecto, con un vehículo que se desplaza a poca velocidad, los resultados serán mejores utilizando los magnetómetros de los que se dispone.

- **Ajuste del Pitch y del Roll**

Los acelerómetros son un buen método para la eliminación del drift de los giróscopos ya que estos no tienen drift, aunque su principal problema es la influencia de las fuerzas dinámicas que puede sufrir la IMU y el ruido de las mediciones. En el caso de que los acelerómetros midieran únicamente la gravedad y no otras aceleraciones, o el caso particular de que las aceleraciones de la IMU sean despreciables, podrían servir de para calcular los ángulos del pitch y el roll por si solos, pero no es así, por lo que se deben usar otros sensores como los giróscopos y utilizar los acelerómetros como ayuda.

El primer paso será calcular el error de la medición del vector gravitacional, para ello se utiliza la siguiente formula, que divide la longitud del vector de aceleraciones que nos dan los acelerómetros, entre el valor de la gravedad en la tierra (9,81 m/s^2):

$$ErAc = \frac{|\boldsymbol{A}|}{g} \tag{4.30}$$

Después se modifica este valor del error para convertirlo en un valor que se pueda multiplicar por la ganancia que se necesitará posteriormente.

$$ErAc' = 1 - (2 * |1 - ErAc|) \tag{4.31}$$

Y se limitará este valor anterior para que este siempre entre 0 y 1.

Posteriormente se hace el productor vectorial entre el vector de aceleraciones y la última fila de la matriz DCM (que como se comentó anteriormente es la fila que únicamente depende del pitch y del roll), y además se multiplica por el error previamente calculado y la constante proporcional del PI para el ajuste del roll y el pitch (Kp1).

$$\boldsymbol{\omega p_{pitch-roll}} = (\boldsymbol{A}\ x\ \boldsymbol{Z}) * ErAc * Kp1 \tag{4.32}$$

Como resultado obtendremos el vector $\boldsymbol{\omega p_{pitch-roll}}$.

De forma similar se obtiene el vector $\boldsymbol{\omega i_{pitch-roll}}$, con la diferencia de que para este último hay que sumarle el vector $\boldsymbol{\omega i}$, de la anterior iteración, y utiliza una constante diferente, que es la constante integral del PI para el ajuste del roll y el pitch (Ki1).

$$\boldsymbol{\omega i_{pitch-roll}} = \boldsymbol{\omega i} + (\boldsymbol{A}\ x\ \boldsymbol{Z}) * ErAc * Ki1 \tag{4.33}$$

- **Ajuste del Yaw**

Para tener en cuenta el Yaw en el ajuste del drift lo primero que se debe hacer es calcular el Yaw mediante los magnetómetros y los acelerómetros, tal y como se hizo en el cálculo de la primera iteración de la matriz DCM.

$$\psi = -\text{atan}\left(\frac{-M_y \cdot \cos(\theta) + M_z \cdot \sin(\theta)}{M_x \cdot \cos(\Phi) + M_y \cdot \sin(\theta) \cdot \sin(\Phi) + M_z \cdot \cos(\theta) \cdot \sin(\Phi)}\right) \tag{4.34}$$

Posteriormente se realiza la siguiente operación para obtener un valor proporcional al error existente, utilizando también valores de la última matriz DCM calculada.

$$ErrorYaw = DCM(0,0) * \sin(\psi) - \text{DCM}(1,0) * \cos(\psi) \tag{4.35}$$

Escalando la última fila de la matriz DCM por el error calculado, obtenemos un vector proporcional al error del yaw.

$$\boldsymbol{VecErrorYaw} = \boldsymbol{Z} * ErrorYaw \tag{4.36}$$

Ahora solo queda añadir este error a los vectores $\boldsymbol{\omega p_{pitch-roll}}$ y $\boldsymbol{\omega i_{pitch-rol}}$ calculados anteriormente.

$$\boldsymbol{\omega p} = \boldsymbol{\omega p_{pitch-roll}} + \boldsymbol{VecErrorYaw} * Kp2 \tag{4.37}$$

$$\boldsymbol{\omega i} = \boldsymbol{\omega i_{pitch-roll}} + \boldsymbol{VecErrorYaw} * Ki2 \tag{4.38}$$

Donde Kp2 y Ki2 son las constantes proporcional e integral para la compensación del yaw.

Los vectores $\boldsymbol{\omega p}$ y $\boldsymbol{\omega i}$ representan la corrección de la velocidad angular en los ejes x, y, y z de la IMU, y por eso se añaden directamente al vector de los datos que provienen de los giróscopos.

$$\boldsymbol{\omega} = \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \boldsymbol{GIR} + \boldsymbol{\omega p} + \boldsymbol{\omega i} = \begin{pmatrix} GIR_x \\ GIR_y \\ GIR_z \end{pmatrix} + \begin{pmatrix} \omega p_x \\ \omega p_y \\ \omega p_z \end{pmatrix} + \begin{pmatrix} \omega i_x \\ \omega i_y \\ \omega i_z \end{pmatrix} \tag{4.39}$$

Cambiando entonces la matriz de rotación utilizada anteriormente, que quedara como:

$$\boldsymbol{ROT} = \begin{pmatrix} 1 & -\omega_z & \omega_y \\ \omega_z & 1 & -\omega_x \\ -\omega_y & \omega_x & 1 \end{pmatrix} \tag{4.40}$$

Una vez realizado todo este proceso se puede representarlo como un esquema de flujo como se hace en la figura 4.5.

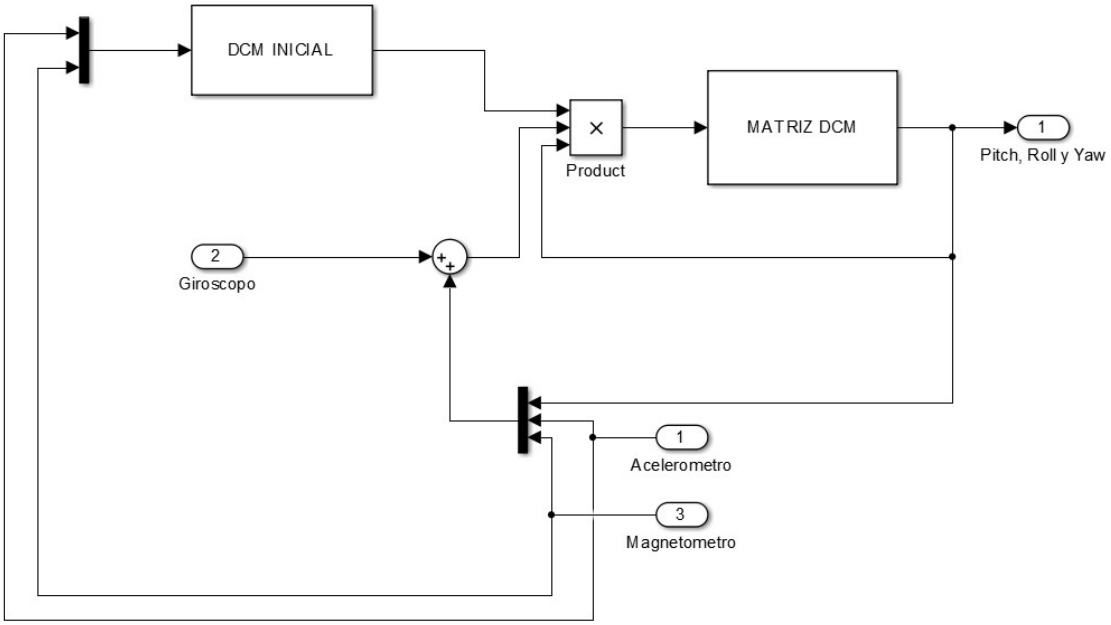


Figura 4.5: Proceso de obtención de los ángulos de Euler

Los datos que provienen de la matriz DCM inicial únicamente se utilizan en la primera iteración, a partir de ese momento no tiene ninguna influencia sobre el ciclo.

4.3 CUATERNIONES

Un cuaternion, o como también se puede llamar, un cuaternio, es un número el cual tiene hasta 3 unidades imaginarias distintas (puede verse como una extensión de los números complejos, que utilizan únicamente una unidad imaginaria "i"). La forma de un cuaternion seria del tipo:

$$Q = A + B\boldsymbol{i} + C\boldsymbol{j} + D\boldsymbol{k} \tag{4.41}$$

Siendo A, B, C y D números reales, y **i**, **j** y **k** números imaginarios.

Los cuaterniones fueron creados por William Rowan Hamilton en 1843 y pueden ser definidos por la siguiente expresión:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{4.42}$$

Estos números imaginarios cumplen ciertas propiedades que extienden las de los números complejos, en la tabla 4.1, llamada tabla de Cayley, se puede ver el resultado de multiplicar las diferentes unidades imaginarias.

	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

Tabla 4.1: Tabla de Cayley

Una propiedad importante de los cuaterniones es que son grupos no conmutativos en las multiplicaciones, pero sin embargo sí que son asociativos.

También se pueden representar los cuaterniones como matrices 2x2 o matrices 4x4, pero en la aplicación práctica que tendrán en este proyecto no será necesario y se utilizarán en su forma vectorial como un vector de 4 componentes, esta forma también se suele ver como un valor escalar "A", y un vector de 3 componentes, "B, C y D".

Los cuaterniones constituyen una notación matemática cuya aplicación práctica más común es la de representar las orientaciones y las rotaciones de objetos en tres dimensiones. Al igual que las matrices de cosenos directores, los cuaterniones evitan el problema del bloqueo de Cardan (Gimbal Lock) que aparece cuando se utilizan los ángulos de Euler.

En comparación con las matrices de rotación, los cuaterniones son más eficientes y más estables numéricamente. Los cuaterniones son útiles en aplicaciones de gráficos por ordenador, robótica, navegación y mecánica orbital de satélites.

Un cuaternion tiene la capacidad de almacenar una orientación o un giro en las 3 dimensiones del espacio, y por lo tanto se pueden realizar operaciones que implican cambios en estas orientaciones. Mediante la suma de un cuaternion que almacena una orientación, y otro cuaternion que representa un giro, se obtiene un tercer cuaternion del cual se puede obtener una nueva orientación.

La suma de cuaterniones consiste simplemente en la suma de cada una de sus componentes.

El caso de la multiplicación es más complejo, se puede definir la multiplicación entre 2 cuaterniones ($p=a+b\mathbf{i}+c\mathbf{j}+d\mathbf{k}$, y $q=w+x\mathbf{i}+y\mathbf{j}+z\mathbf{k}$) como se hace en la ecuación 4.43.

$$pq = aw - (bx + cy + dz) + a(x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k}) + w(b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}) + \boldsymbol{i}(cz - dy) + \boldsymbol{j}(dx - bz) + \boldsymbol{k}(by - cx) \quad (4.43)$$

Se puede apreciar la simetría entre las componentes imaginarias en contraposición con la componente escalar.

4.3.1 Cálculo inicial

Al igual que se hace con la matriz DCM, es necesario crear un cuaternion inicial a partir de únicamente los datos obtenidos por el acelerómetro y el magnetómetro, obteniendo los ángulos de Euler para ese instante inicial, y a partir de ellos formar un cuaternion.

Las ecuaciones para el cálculo del pitch, del roll y del yaw serán exactamente las mismas que las utilizadas en el apartado 4.1.1.

Si el cuaternion donde se almacena toda la información tiene la estructura de "Q = Q[0]+Q[1]**i**+Q[2]**j**+Q[3]**k**", entonces los números reales Q[0], Q[1], Q[2] y Q[3] serán:

$$\begin{aligned} Q[0] &= cos(\theta/2) * cos(\Phi/2) * cos(\psi/2) + sin(\theta/2) * sin(\Phi/2) * sin(\psi/2) \\ Q[1] &= sin(\theta/2) * \cos(\Phi/2) * cos(\psi/2) - cos(\theta/2) * sin(\Phi/2) * sin(\psi/2) \\ Q[2] &= cos(\theta/2) * sin(\Phi/2) * cos(\psi/2) + sin(\theta/2) * cos(\Phi/2) * sin(\psi/2) \\ Q[3] &= cos(\theta/2) * cos(\Phi/2) * sin(\psi/2) - sin(\theta/2) * sin(\Phi/2) * cos(\psi/2) \end{aligned} \quad (4.44)$$

Una vez calculado este cuaternion será necesario normalizarlo para reducir el error que puede tener, para ello se divide cada uno de sus componentes entre la magnitud del cuaternion (se toma el cuaternion como un vector de 4 componentes). Este paso será poco importante en la composición del cuaternion inicial, pero posteriormente es de gran importancia en las actualizaciones de este.

4.3.2 Actualización del cuaternion con los giróscopos

Para comprobar que los giros del cuaternion funcionan correctamente, primero se realizará el cálculo de los ángulos de Euler sin utilizar ningún filtro es decir, utilizando el cuaternion inicial y actualizándolo únicamente con los datos de los giróscopos.

Como se comentó anteriormente, es sencillo sumar ambos cuaterniones para obtener la suma de las orientaciones de ambos, obteniendo un tercer cuaternion con la posición actual.

Para esto se utiliza un cuaternion adicional que se llamará Q_Giro, el cual almacenará información del giro que es necesario realizar desde la orientación anterior a la siguiente. Para formar este cuaternion será necesario utilizar los datos de los giróscopos y el cuaternion principal Q.

El cuaternion Q_Giro tiene los siguientes componentes:

$$\begin{aligned} Q_{Giro}[0] &= 0.5 * (-Q[1] * GIR_x - Q[2] * GIR_y - Q[3] * GIR_z) \\ Q_{Giro}[1] &= 0.5 * (Q[0] * GIR_x + Q[2] * \ GIR_z - Q[3] * GIR_y) \\ Q_{Giro}[2] &= 0.5 * (Q[0] * GIR_y - Q[1] * GIR_z + Q[3] * GIR_x) \\ Q_{Giro}[3] &= 0.5 * (Q[0] * GIR_z + Q[1] * GIR_y - Q[2] * GIR_x) \end{aligned} \quad (4.45)$$

Posteriormente cada una de las componentes del cuaternion se multiplican por el periodo de muestreo, que se calculará de la misma forma que para la matriz DCM, y que representa el tiempo transcurrido entre cada iteración (en segundos). El resultado de estas operaciones se sumará a la componente correspondiente del cuaternion Q.

$$\begin{aligned} Q[0] &= Q[0] + Q_{Giro}[0] * Dt \\ Q[1] &= Q[1] + Q_{Giro}[1] * Dt \\ Q[2] &= Q[2] + Q_{Giro}[2] * Dt \\ Q[3] &= Q[3] + Q_{Giro}[3] * Dt \end{aligned} \quad (4.46)$$

El cuaternion Q resultante será necesario que se le realice una operación de normalización en cada iteración para que no se acumulen errores de cálculo.

El último paso consistirá en obtener los valores para los ángulos de Euler, que se utilizarán como salida del programa, ya que es una forma más visual y con menos parámetros de mostrar los datos. Para realizar esta transformación es necesario utilizar las siguientes ecuaciones:

$$\Phi = \sin^{-1}(-2 * Q[1] * Q[3] + 2 * Q[0] * Q[2]) \quad (4.48)$$

$$\theta = \tan^{-1} \frac{(2*Q[2]*Q[3]+2*Q[0]*Q[1])}{(1-2*Q[1]*Q[1]-2*Q[2]*Q[2])} \quad (4.49)$$

$$\psi = \tan^{-1} \frac{(2*Q[1]*Q[2]+2*Q[0]*Q[3])}{(1-2*Q[2]*Q[2]+Q[3]*Q[3])} \quad (4.50)$$

4.3.3 Filtro complementario para eliminar el Drift

Tal y como se hizo con la matriz DCM, se utilizará un filtro, que en este caso será un filtro complementario, para utilizar en cada iteración los datos obtenidos por el acelerómetro y por el magnetómetro, evitando depender únicamente de los giróscopos que tienen deriva.

Para ello se modificaran los valores del cuaternion Q_Giro mediante un filtrado entre este y un nuevo cuaternion que se llamará Q_Filtro. Este cuaternion se obtendrá mediante operaciones que utilizarán los valores obtenidos por los sensores del acelerómetro y del magnetómetro, además de utilizar los componentes del cuaternion Q.

El cálculo del cuaternion Q_Filtro es bastante complejo y se ha obtenido a partir del AHRS de Madgwick que se puede ver en la bibliografía y en el código correspondiente en los anexos de este proyecto.

Una vez calculado Q_Filtro se normaliza tal y como se hizo anteriormente para otros cuaterniones.

El cuaternion Q_Giro que se utiliza para realizar en giro del cuaternion Q, será el resultado de realizar el filtro complementario entre el cuaternion Q_Giro obtenido por los giróscopos, y el cuaternion Q_Filtro.

$$\begin{aligned} Q_{Giro}[0] &= 0.98 * Q_{Giro}[0] + 0.02 * Q_{Filtro}[0] \\ Q_{Giro}[1] &= 0.98 * Q_{Giro}[1] + 0.02 * Q_{Filtro}[1] \\ Q_{Giro}[2] &= 0.98 * Q_{Giro}[2] + 0.02 * Q_{Filtro}[2] \\ Q_{Giro}[3] &= 0.98 * Q_{Giro}[3] + 0.02 * Q_{Filtro}[3] \end{aligned} \quad (4.51)$$

Este cuaternion se utiliza de la misma forma que en el apartado anterior se utilizaba el Q_Giro, para obtener el nuevo Q que contiene toda la información necesaria para obtener la orientación actual.

4.4 COMPARACIÓN MATRIZ DCM-CUATERNIONES

Ambos métodos tienen similitudes, y su principal diferencia consiste en su forma de almacenar los datos, donde el primero utiliza matrices de dimensión 3x3 (9 parámetros), el segundo utiliza vectores de 4 parámetros. Además de la matriz o del cuaternion principal, también se deben utilizar otros complementarios para realizar los giros y filtros.

Por lo tanto en cuanto a cantidad de parámetros los cuaterniones tienen cierta ventaja en el procesamiento de datos.

El periodo de muestreo es muy similar en ambos casos, siendo de uno 30ms, y depende principalmente de la longitud de la información que utilicemos en la salida de datos. Por lo que respecto a este importante factor ninguno tiene ventaja.

Respecto a los resultados obtenidos es notablemente mejor el método de los cuaterniones, ya que utilizando la matriz DCM se tienen ciertos problemas cuando el roll se acerca a los 90º, provocándose un giro del yaw inexistente (esto se debe a la forma de utilizar el filtro con el PI). Este problema no existe cuando se utilizan cuaterniones con el filtro complementario.

En ambos métodos se comprueba que existe cierta desviación del yaw cuando se utiliza el respectivo filtro, esto es debido a ligeras variaciones magnéticas que perturban las medidas del magnetómetro e inducen un giro en el yaw.

Teniendo en cuanta las razones anteriores se determina que el método utilizado será el de los cuaterniones, aunque cuando se ejecute el

programa mediante la comunicación con el puerto serie es posible cambiar el método durante la ejecución introduciendo un comando al programa. Esto ayudará a realizar una comparación entre ambos métodos.

CAPÍTULO 5: GPS

GPS son las siglas de "Global Positioning System" que hace referencia al sistema de posicionamiento global. Este sistema tiene la capacidad de poder conocer la posición exacta de un objeto (mediante un receptor GPS) en la Tierra con una gran precisión con respecto a otros sistemas similares (con un GPS diferencial se puede tener una precisión de centímetros).

Un GPS permite conocer al usuario las coordenadas de su posición, utilizando la latitud y la longitud, pero esto no es lo único que se puede obtener, como se explicará posteriormente.

El GPS fue creado por el Departamento de Defensa de los Estados Unidos, en principio la motivación para su desarrollo fue militar, pero posteriormente se desarrollaron sus aplicaciones civiles de las que actualmente cualquier teléfono móvil dispone. Cabe diferencias el funcionamiento del GPS para aplicaciones militares y civiles, su funcionamiento para aplicaciones militares es mucho más potente y preciso que el utilizado para aplicaciones civiles, ambos utilizan frecuencias distintas para evitar interferencias entre ellos.

Antes del año 2000 los Estados Unidos llevaban a cabo el llamado SA (Selected Avaibility), que aplicaba un error a propósito en las medidas dadas por el GPS para evitar que los usuarios pudieran utilizar plenamente la potencia de su sistema. Este error era de ±100 metros, lo que dificultaba la utilización del GPS para ciertas aplicaciones. En el año 2000 Estados Unidos decidió eliminar el SA ante la posibilidad de que otros sistemas de posicionamiento pudieran superar en popularidad al GPS.

Además el GPS tiene un método para que, a menos que se tenga permiso de los Estados Unidos, no permite su funcionamiento para objetos los cuales se detecte que se desplazan a más de una determinada velocidad, evitando así que pueda ser usado para aplicaciones militares no autorizadas por ellos (misiles guiados).

Actualmente su mayor aplicación es la de posicionamiento de vehículos, con un receptor GPS localizado en un vehículo se puede conocer la posición de este en la red de carreteras, utilizando complementariamente un sistema de mapas inteligente se puede usar como método de guiado de rutas. En la figura 5.1 se muestras un ejemplo de un monitor GPS para un vehículo.

Figura 5.1: Monitor GPS para un vehículo

Las funciones prácticas con las que se puede utilizar un receptor GPS son muy amplias, a continuación se citan algunas de ellas:

- Navegación por tierra, mar y aire.
- Cálculo y optimización de rutas.
- Sistemas de gestión de trenes, flotas de autobuses, camiones, etc…
- Localización. Sirven para conocer la situación de vehículos robados, animales, personas, etc…
- Robots móviles
- Fotografía acompañada de coordenadas gráficas.
- Topografía y geodesia.
- Guiado de misiles.

En este tipo de proyecto se puede utilizar el GPS para cumplir 2 funciones, la primera es para conocer la posición global del vehículo, y poder fusionarlo con los datos obtenidos por la IMU para corregirla. La segunda es para obtener la dirección hacia la que se desplaza el vehículo, para ayudar a obtener la orientación, aunque como se explicará posteriormente esto no se llevará a cabo debido a la baja velocidad que tiene el vehículo de este proyecto (cuanta menor es la velocidad a la que se utiliza menor es la precisión), utilizándose en su lugar el magnetómetro de la IMU.

5.1 FUNCIONAMIENTO

Para su funcionamiento es necesaria una red de al menos 24 satélites que orbitan a la Tierra, los cuales están programados para que todos los puntos del planeta tengan la posibilidad de conectarse con al menos 4 de ellos. Este receptor intercambia señales con los satélites, en función del

tiempo que tarda cada señal en llegar y rebotar en el satélite se calcula con gran precisión la distancia entre receptor y satélite, estas distancias nos permiten calcular la posición relativa del receptor con respecto a los satélites. Como además la posición de los satélites con respecto a la Tierra es conocida en todo momento, es posible calcular la posición del global receptor en la Tierra.

Es necesaria la conexión con 4 satélites ya que es el mínimo necesario para conocer la posición exacta de un receptor. Si se tiene la conexión con un satélite, se puede conocer la distancia a este, por lo tanto existen un número infinito de posibles posiciones repartidos en una esfera con centro en el satélite, mediante un segundo satélite se reducen las posibles posiciones a las que forman la intersección entre 2 esferas, es decir, un circulo, un 3 satélite reduciría las posibilidades a 2 puntos, y el cuarto satélite las dejaría en una única posible posición.

Cada uno de los satélites están equipados con alta tecnología para su funcionamiento, entre sus componentes cabe destacar el transmisor de radiofrecuencia (utiliza una frecuencia de 1575,42 MHz para las transmisiones comerciales y una frecuencia de 1227,60 para las militares), un sistema computacional para el procesado de la información, y un reloj atómico de muy alta precisión (con un error aproximado de 1 segundo cada 30000 años).

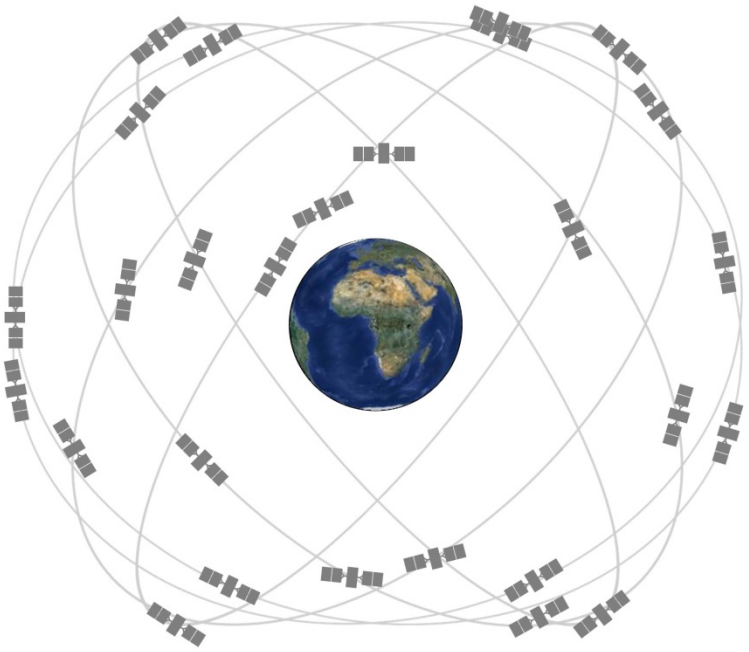


Figura 5.2: Representación de las orbitas de los satélites GPS

Además de los satélites en órbita, también son necesarias 5 estaciones repartidas estratégicamente por el mundo (estas se encuentran en:

Hawai, Diego García, isla de Ascensión, atolón de Kwajalein y Colorado Spring), que se utilizan para rastrear y controlar los satélites y sus comunicaciones

Un GPS también es capaz de calcular la velocidad y la dirección a la que se desplaza el receptor, para ello lo que hace es medir la distancia que se desplaza en un determinado tiempo (generalmente un segundo, pero se puede reducir a 0,2s), obviamente este dato será de nula utilidad cuando el receptor no se mueva o lo haga muy lentamente, ya que el error de posicionamiento afectará mucho al cálculo de la velocidad, por eso se suele utilizar en vehículos que viajan a velocidades considerables.

Para que la precisión del posicionamiento sea alta, es de vital importancia la exactitud de los relojes atómicos de los que disponen todos los satélites de la red. También es importante que el número de satélites con los que conecta el receptor sea alto y que estén en posiciones idóneas para triangular mejor la posición. Si todo esto se cumple de forma aceptable el GPS nos proporcionará un posicionamiento con una precisión menor de 2’5 metros.

Existen también los llamados GPS diferenciales (DGPS), los cuales tienen una precisión mayor que los GPS normales, llegando a tener precisiones de centímetros, ya que tienen la capacidad de corregir el error que hay en las señales. Para ello utilizan como referencia un receptor GPS fijo ubicado en una posición cercana, del cual se conoce perfectamente su posición en la tierra, y se calcula su posición según GPS, calculando la diferencia entre ambos posicionamientos se obtiene el error existente, que se presupone igual al error que existirá en el receptor que se quiere calcular su posición.

En la figura 5.3 se muestra el equipamiento necesario para la utilización de un DGPS.

Figura 5.3: Estaciones DGPS

La mayoría de los receptores GPS utilizan dos formatos para mostrar los datos obtenidos, la llamada interfaz NMEA, y en binario. La NMEA consiste en una cadena de caracteres, separados por comas, de forma que una persona puede interpretar directamente los datos, existen varios métodos en función de los datos de salida requeridos. El formato binario puede transmitir mayor información que el anterior, pero para la interpretación por parte de una persona hace falta convertir posteriormente esta información. Una de las variables que por ejemplo el formato binario es capaz de transmitir y que el NMEA no hace es la velocidad del receptor en el eje Z de la tierra.

También existen dos sistemas distintos de coordenadas que puede utilizar un GPS, el primero de ellos y más típico es el que utiliza la longitud, latitud, altitud, velocidad y la dirección del receptor sobre el terreno (la dirección del movimiento está dada por el ángulo que difieren la dirección y el norte).

El otro sistema es conocido como ECEF (Earth-Centered, Earth-Fixed), que nos da las coordenadas X, Y y Z de la posición y la velocidad con respecto a las mismas coordenadas del centro de la Tierra.

En la figura 5.4 se muestra un mapa de la Tierra con los valores de latitud y longitud según el cuadrante.

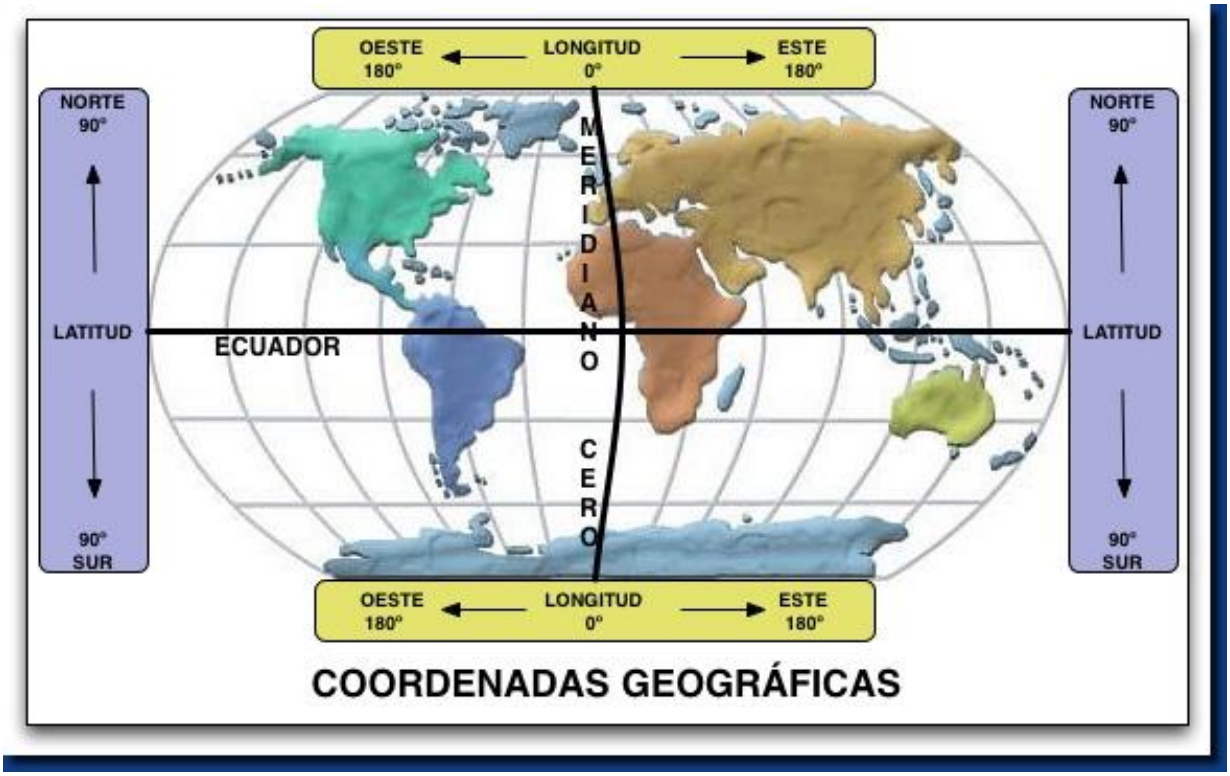


Figura 5.4: Sistema de coordenadas geográficas

La latitud es el ángulo de un punto de la Tierra con respecto el ecuador, tiene un rango de 90º Sur a 90º Norte, también se suele expresar utilizando un rango de -90º a +90º, si es negativo implica que son coordenadas del hemisferio sur, y si es positivo son coordenadas del hemisferio norte. De esta forma es más fácil operar con las coordenadas matemáticamente. A los puntos con la misma latitud se les llama paralelos, y su longitud depende de la latitud.

La longitud es el ángulo en cambio es el ángulo de un punto de la Tierra con respecto el meridiano de Greenwich, y a diferencia de la latitud, tiene un rango de -180º (oeste) a 180º (este). Los puntos con una misma longitud se denominan meridianos y todos miden lo mismo.

Para presentar la latitud y la longitud se pueden utilizar también unidades de minutos y segundos para los decimales de los grados.

Se da la circunstancia de que un grado sexagesimal de la longitud no equivale siempre a la misma distancia, debido a que los paralelos no miden lo mismo, pero sin embargo un grado sexagesimal de la latitud siempre tiene la misma equivalencia en kilómetros, un meridiano tiene una distancia de 40007,161 Km, si esta distancia la dividimos entre los 360º grados posibles, obtenemos que 1º corresponde con 111,13 Km, que, aunque no es exacto, es una muy buena aproximación. En el caso de

la longitud, la distancia de 1º depende totalmente de la latitud a la que esta medida. Por ejemplo en el ecuador 1º de longitud equivale a unos 111,3 Km, mientras que a una latitud de 45º, 1º de longitud supondría una distancia de 78,84 Km, como se puede ver en la siguiente tabla.

Latitud (º)	ΔLongitud (Km)
0	111,32
15	107,55
30	96,48
45	78,84
60	55,80
75	28,90
90	0

Tabla 5.1: Equivalencia en Km de una variación de 1º de longitud en función de la latitud

Para definir totalmente un punto es necesario un tercer parámetro, este será la altitud, aunque la precisión del GPS es menor que con la latitud y la longitud. La altura esta medida en metros con respecto el nivel del mar.

5.2 MÓDULO LONET SIM808

El modulo que se utilizará en este proyecto será la placa de Lonet que tiene un chip Sim808, el cual tiene la posibilidad de ser utilizado como GPS y como GSM/GPRS, aunque en este caso solo se usará el GPS.

En la figura 5.5 se puede ver una fotográfica del módulo Lonet con el chip SIM808.

Figura 5.5: Modulo Lonet SIM808

Para su correcto funcionamiento es imprescindible utilizar una antena GPS y una batería externa de entre 3.4V y 4.4V, que alimente la placa, además del Arduino para comunicarse y recopilar la información.

En cuanto a sus características técnicas (que están descritas más detalladamente en su Datasheet en los anexos), destacan que es un módulo de bajo consumo, con una precisión de unos 2.5m de error, tiene 66 canales de adquisición y 22 de seguimiento y una sensibilidad de seguimiento de -165 dBm.

En la figura 5.6 se muestra un esquema del módulo con los puertos de los que dispone, indicando la posición de los pines, las antenas, o los Leds.

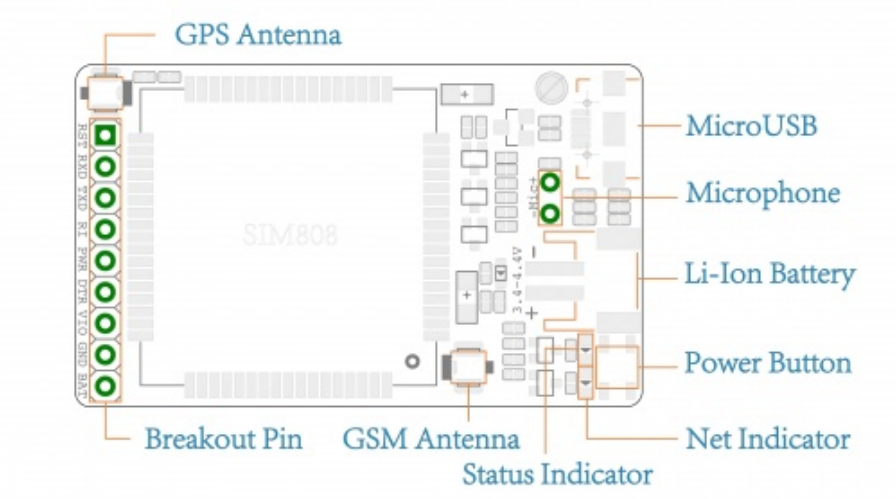


Figura 5.6: Esquema de Lonet SIM808

Para la comunicación con el Arduino se utiliza un puerto serie RX y TX, para enviar y recibir información, además necesita alimentarse mediante dos hilos más para la tierra y los 3.3V de nivel de tensión del Arduino Due.

Se dispone de 3 antenas receptoras GPS, con las cuales se realizarán pruebas para elegir la que mejor funciona según los resultados obtenidos.

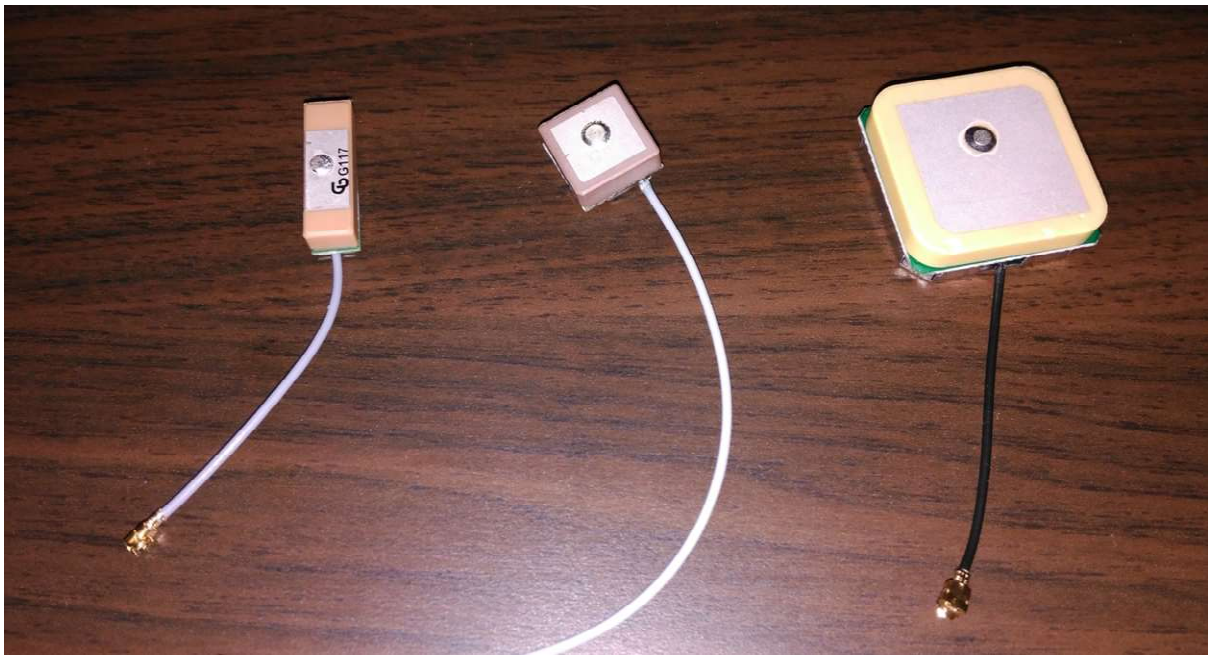


Figura 5.7: Antenas GPS

Como no se tienen datos de las antenas, en este proyecto se denominaran como A1, A2 y A3, en el orden que aparecen en la figura 5.7.

5.2.1 Comandos AT

La comunicación por software con el modulo se realizará mediante los llamados comandos AT, de tal manera que desde Arduino se envía por comunicación serie, un comando AT al módulo y este responde en función del comando enviado.

Los comandos AT generalmente siempre devuelven una cadena de caracteres con finalizando con un "OK" si el comando se ha ejecutado correctamente, o "ERROR" si ha habido un error en su ejecución.

En la tabla 5.2 se resumen los comandos más importantes para utilizar con el GPS.

COMANDO	DESCRIPCION
AT+CGPSPWR	Se utiliza para encender y apagar el modulo GPS internamente, AT+CGPSPWR=1 -> Encender. AT+CPGSPWR=0 -> Apagar.
AT+CGPSRST	Necesario para resetear el modulo para buscar satélites, existen 3 tipos de reset: AT+CGPSRST=0 -> Cold Start AT+CGPSRST=1 -> Hot Start AT+CGPSRST=2 -> Warm Start
AT+CGPSSTATUS?	Devuelve el estado del GPS, existen 4 posibilidades: "Location Unknown" -> GPS no está funcionando "Location Not Fix" -> No ha conectado con los satélites necesarios. "Location 2D Fix" -> Puede dar localización 2D "Location 3D Fix" -> Puede dar localización 3D
AT+CGPSINF	Devuelve una línea con la información requerida, tiene 8 modos que devuelven diferente información en función del formato NMEA elegido, y que se describirán a continuación.
AT+CGPSOUT	Devuelve una línea a cada segundo, es el comando necesario para obtener las coordenadas constantemente, tiene 8 modos, aunque el primero de ellos es de apagado.

Tabla 5.2: Principales comandos AT de GPS

5.2.2 Formatos de salida de datos

A continuación se describen los diferentes formatos de salida de los datos disponibles, en función del modo utilizado en el comando AT correspondiente.

- Modo 0: Para CGPSOUT este modo es de detención, pero para el CGPSINF se devuelve lo siguiente (este no es un formato NMEA):

<modo>,<latitud>,<longitud>,<altitud>,<tiempo UTC>,<Tiempo de conexión>,<Nº satélites>,<velocidad>,<dirección>

- Modo 1: GGA (Global Positioning System Fixed Data):

<tiempo UTC>, <latitud>, <N/S>, <longitud>, <E/W>, <conexión>, <Nº satélites>, <HDOP>, <altitud>, <unidad>, <separacióngeoide>, <unidad>, <DGPS>, <ID referencia DGPS>, <Checksum>, <terminación>

- Modo 2: GLL (Geographic Position – Latitude/Longitude)

<latitud>, <N/S>, <longitud>, <E/W>, <tiempo UTC>, <estado>, <modo>, <checksum>, <terminación>

- Modo 3: GSA (GNSS DOP and Active Satellites):

<modo 1 (automatico/manual)>, <modo 2 (conexión)>, <satélite 1>, <satélite 2>, … , <PDOP>, <HDOP>, <VDOP>, <checksum>, <terminación>

- Modo 4: GSV (GNSS Satellites in View):

<Nº mensajes>,<mensaje 1>,<Nº satélites>, <ID satélite 1>,<elevación>,<azimuth>,<SNR>, <ID satelite 2>, … , <checksum>,<terminación>

- Modo 5: RMC (Recommended Minimium Specific GNSS Data):

<tiempo UTC>,<estado>,<latitud>, <N/S>,<longitud>,<E/W>, <velocidad>, <dirección>,<fecha>,<variación magnética>,<E/W>,<modo>, <checksum>,<terminación>

- Modo 6: VTG (Couser Over Ground and Ground Speed):

<direccion>, <referencia>, <direccion>, <referencia>, <velocidad>, <unidades>, <velocidad>, <unidades>, <modo>, <checksum>, <terminación>

- Modo 7: ZDA (Time and Date):

<tiempo UTD>,<día>,<mes>, <año>, <zona horaria local>, <00>,<checksum>,<terminación>

Como se puede apreciar se pueden obtener una gran variedad de datos utilizando este GPS, en función del objetivo del proyecto se utilizará un formato de datos u otro.

5.3 PRUEBAS GPS

Para comprobar la fiabilidad del GPS y su correcto funcionamiento se realizan 2 pruebas que darán información para determinar el peso que se podrá dar al GPS en la fusión sensorial.

Para calcular las distancias que obtiene el GPS habrá que hacer una conversión de los datos en coordenadas geográficas a distancias lineales en metros. Para los datos de latitud que se obtienen con el GPS, se determina que la equivalencia grados sexagesimales – Km es la siguiente:

1º de la latitud = 111,10 Km

1º de longitud = 80,94 Km

5.3.1 Precisión en un punto estático

En esta prueba se realizarán varias medidas de las coordenadas GPS para tener una idea de la precisión que puede tener, con esta prueba se determinará que antena se utilizará para realizar el proyecto.

Para realizar la prueba se recogen datos durante aproximadamente 1 minuto utilizando el comando AT+CGPSOUT=2, del cual se puede obtener una línea de datos a cada segundo, entre esos datos se encuentran la latitud y la longitud obtenidas. El receptor del GPS no se moverá en ningún momento durante la prueba, por lo que teóricamente los datos obtenidos deberán ser siempre los mismos, esto no será así debido al error de precisión del GPS.

Mediante Arduino IDE se obtendrán los datos y posteriormente se almacenarán en un archivo de texto ".txt", que después se importarán en un programa de Matlab para su procesamiento. En el anexo de códigos se puede ver el archivo ".m" de Matlab.

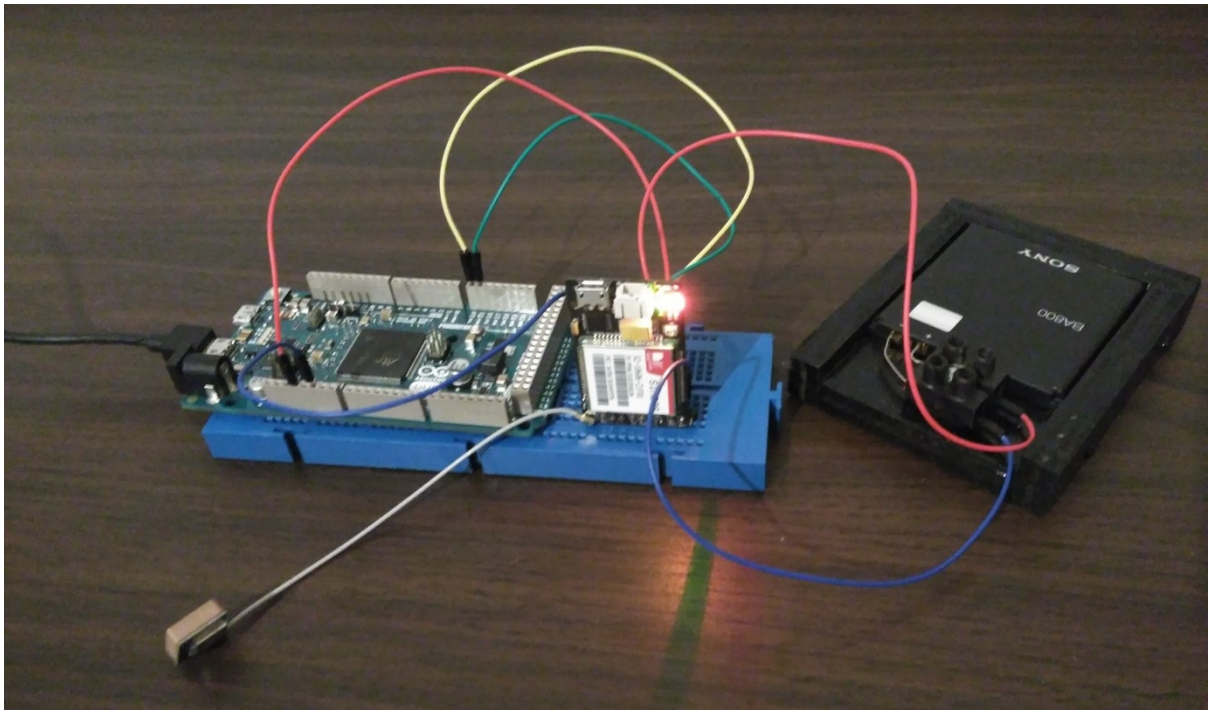


Figura 5.8: GPS en funcionamiento

En la figura 5.8 se muestran los dispositivos utilizados para la realización de esta prueba.

Los datos obtenidos por Arduino son del tipo como el siguiente:

```
$GPGGA,174724.000,4327.9720,N,00348.7999,W,1,5,1.49,81.9,M,51.5,M,,*7A
```

Tras ejecutar el programa en Matlab con los datos obtenidos por el GPS se consigue la gráfica de la figura 5.9:

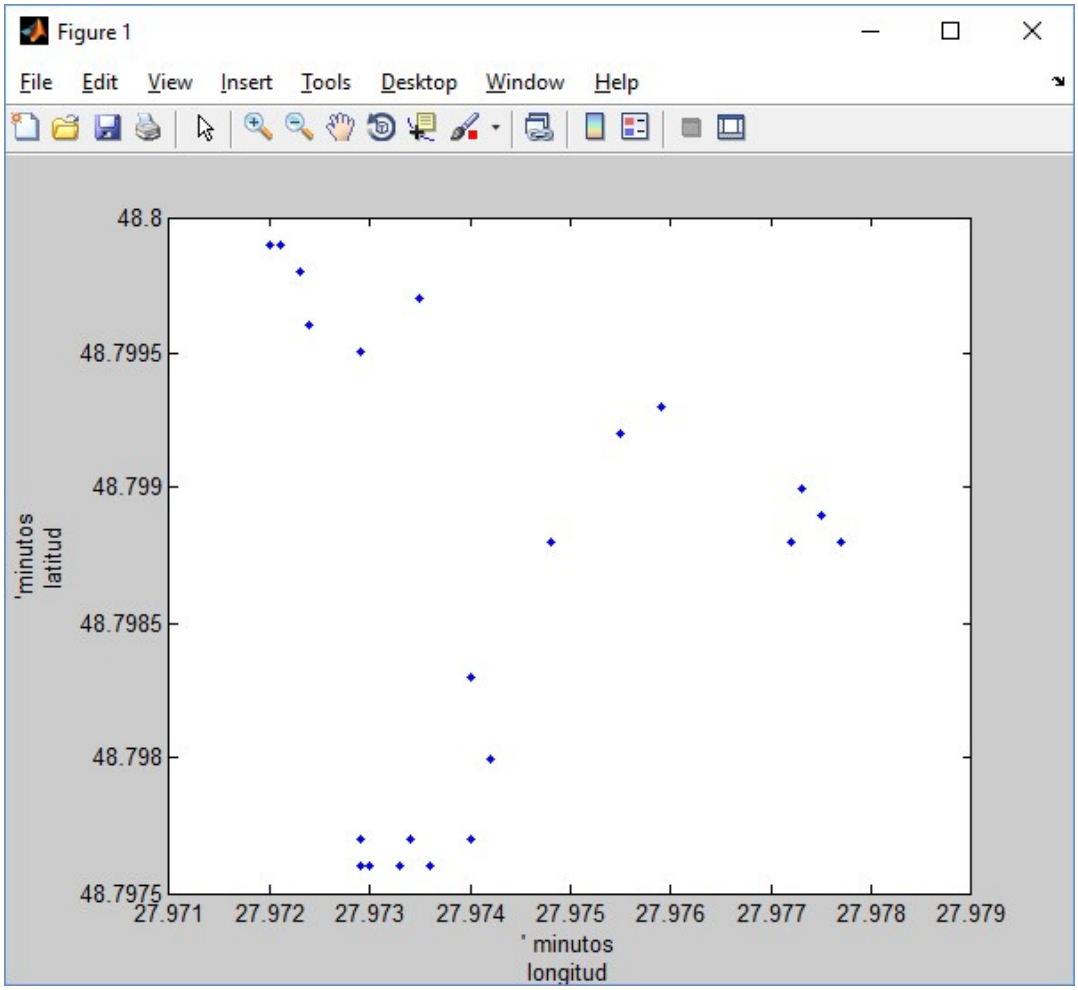


Figura 5.9: Dispersión de datos de latitud y longitud de un punto estático

También se calcula el error en metros tanto de la latitud como de la longitud, para ello se utiliza el valor máximo y mínimo obtenido para cada coordenada, y se convierte el resultado a metros para que la referencia sea mejor. Haciendo esto se obtiene que los errores para las diferentes antenas son:

ANTENA	Error Latitud	Error Longitud
A1	±5.46 m	±0.54 m
A2	±5.27 m	±1.55 m
A3	±20.55 m	±8.90 m

Tabla 5.3: Resultados de las diferencias antenas

Llama la atención que la antena con mayores errores sea la de mayor tamaño, que además es la que mayor tiempo de conexión tiene (49s), por lo tanto la antena A3 queda descartada.

También salta a la vista la diferencia entre los errores de la latitud y de la longitud, que en parte se debe a que la escala de la longitud es menor que el de la latitud. Además también se debe tener en cuenta que el error en la latitud es mayor que el que se indica en la información técnica del módulo GPS, aunque sigue estando en unos límites razonables para lo que se pretende.

Entre las antenas A1 y A2 se utilizará la segunda debido a que el error de la latitud es menor, y en este caso es el error más importante, además la antena A2 también tiene menor tiempo de conexión que la A1, por lo que es otro factor a su favor.

5.3.2 Recorrido con un vehículo

En esta prueba se realiza un recorrido con el Arduino conectado y recibiendo las coordenadas GPS, estas coordenadas que se han ido obteniendo a lo largo del recorrido se guardan posteriormente en un archivo de texto para procesarlas con Matlab.

Para realizar esta prueba se ha utilizado la antena A2 la cual se decidió anteriormente que sería la utilizada. Y se ha realizado un recorrido cerrado de entre 3 y 4 kilómetros, utilizando el comando AT+CGPSOUT=2, que da las coordenadas de latitud y longitud a cada segundo, como se hizo en la prueba anterior.

La muestra de resultados obtenidos tiene un tamaño de 413 datos de coordenadas, y mediante un programa de Matlab llamado "Recorrido.m" se consigue graficar los resultados, obteniendo la gráfica de la figura 5.9.

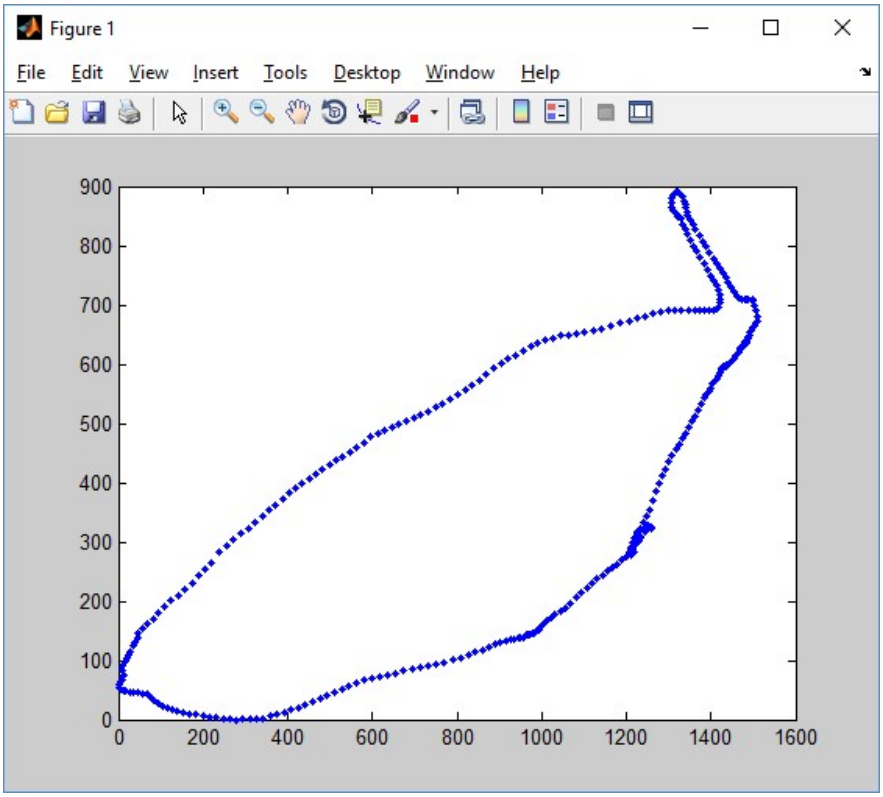


Figura 5.10: Resultados obtenidos en la prueba del recorrido

En la gráfica se ha establecido como origen de coordenadas el menor de los datos de latitud y el menor de los datos de longitud, y se ha escalado para eliminar la diferencia proporcional entre latitud-longitud, mostrando los ejes en metros.

Para evaluar la precisión de estos resultados se ha utilizado el programa "Google Earth" para superponer la gráfica obtenida en Matlab con el mapa real del recorrido de una imagen por satélite.

Para escalar la imagen en Google Earth se necesitan las coordenadas de 3 puntos significativos del recorrido, para ello obtenemos en Matlab las coordenadas de los 3 puntos elegidos y se crean estos puntos en el Google Earth, seguidamente se inserta la imagen de la gráfica y se modifica su tamaño y orientación manualmente para que los 3 puntos coincidan.

Hay que tener en cuenta las distintas formas de expresar las coordenadas, ya que por ejemplo el comando AT+CGPSOUT=2 da las coordenadas de latitud de la forma: ggmm.mmmm (donde g son grados y m son minutos) mientras que en Matlab se han utilizado únicamente grados con decimales, y para introducir coordenadas en Google Earth hace falta hacerlo utilizando grados, minutos y segundos.

En la tabla 5.4 se encuentran los datos de latitud y longitud para los 3 puntos de referencia:

Punto	Latitud (º)	Longitud (º)
PA	43.455488	-3.848391
PB	43.447950	-3.864665
PC	43.448786	-3.852530

Tabla 5.4: Coordenadas de referencia del recorrido

Una vez realizado lo anterior se obtiene la figura 5.11.

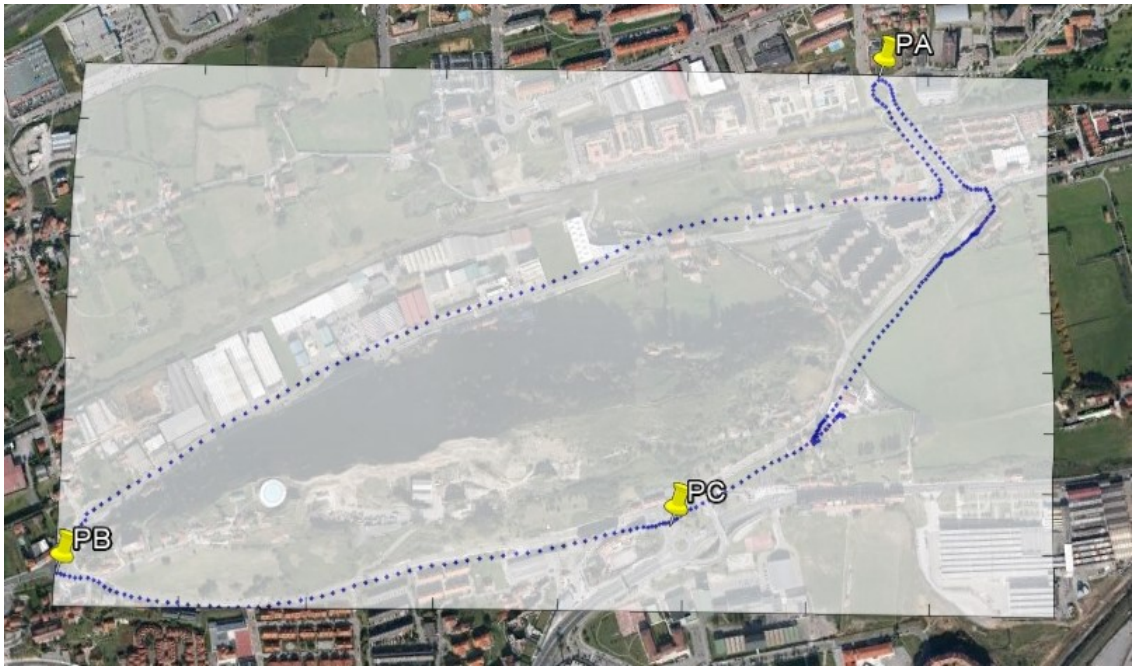


Figura 5.11: Superposición del recorrido en Google Earth

Se puede apreciar que los resultados son en su mayoría buenos, pero existen 2 zonas donde la diferencia entre las coordenadas obtenidas y las coordenadas reales de la carretera difieren en unos metros.

5.4 FILTRO KALMAN

Debido al error existente en las mediciones del GPS, se ha implementado un filtro para obtener las coordenadas filtradas, para ello se ha elegido el filtro Kalman.

Este filtro pretende conseguir que el error en cada medición del receptor GPS afecte lo menos posible a las coordenadas que el ordenador de a bordo del vehículo utilice para el control. Esto contribuirá a conseguir mejores resultados.

El GPS que se utiliza en los coches para la navegación no utiliza este sistema de filtrado, ya que no es necesario aumentar la precisión, por lo

tanto el sistema GPS de este proyecto es teóricamente mejor que un sistema GPS que utilizan los vehículos (aunque el objetivo es distinto).

El filtro Kalman es un algoritmo recursivo, que debe su nombre a su creador, Rudolf E. Kalman, que en 1960 lo utilizo para estimar un estado oculto de un sistema dinámico lineal, y que funciona aun estando el sistema sometido a un ruido blanco aditivo. El mayor avance que supuso fue la actualización sistemática de la ganancia K, cuando se conocen las varianzas de los ruidos que afectan al sistema.

El filtro Kalman puede funcionar a tiempo real utilizando únicamente las entradas, el estado anterior, y la matriz de incertidumbre.

Sus aplicaciones más comunes se encuentran en la guía, navegación y control de vehículos, en el procesamiento de señales, y en la econometría.

Su aparición en el mundo tecnológico supuso un gran avance en el ámbito de las naves espaciales, ya aumento la precisión y la eficiencia en la navegación de estas. También ha sido de gran utilidad en el avance de la navegación y el seguimiento de cualquier tipo de vehículo.

Un filtro Kalman tiene 2 pasos esenciales en su funcionamiento, un primer paso de predicción o estimación del estado futuro, y un segundo paso de corrección de esa estimación.

En la figura 5.12 se puede ver el diagrama sencillo del funcionamiento de un filtro Kalman.

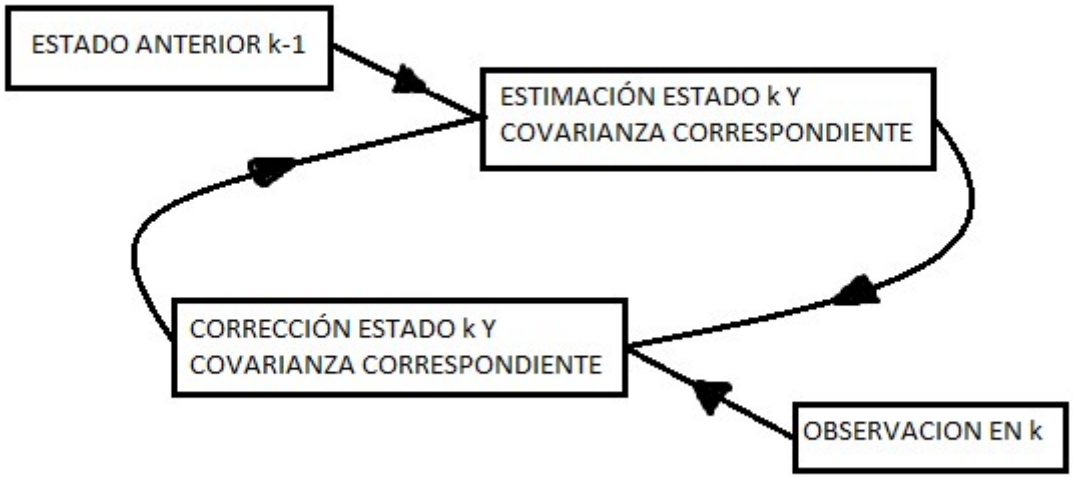


Figura 5.12: Diagrama filtro Kalman

Las ecuaciones en las que se basa un filtro Kalman para tiempo discreto (como es este caso) son las siguientes:

$$x_k = A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + w_{k-1} \quad (5.1)$$

$$z_k = H_k x_k + v_k \quad (5.2)$$

Dónde:

-k representa el instante actual, mientras que k-1 es el instante anterior.

-x_k es el vector de estados que utiliza el filtro Kalman.

-z_k es el vector de entradas al sistema.

-w_k es el ruido blanco de valor promedio nulo y varianza R_k.

-v_k es el ruido blanco de valor promedio nulo y varianza Q_k.

-A_k es la matriz que relaciona el estado el estado predicho y el estado anterior.

-H_k es la matriz que relaciona las entradas del sistema y el vector de estados en el supuesto de que no existiera ruido.

Mientras que las ecuaciones que se utilizan para la predicción y la corrección recursivas son las siguientes:

- Estimación

$$\hat{x}_k = A_k x_{k-1} \quad (5.3)$$

$$\hat{P}_k = A_k P_{k-1} A_k^T + Q_k \quad (5.4)$$

- Corrección

$$K_k = \frac{\hat{P}_k H_k^T}{H_k \hat{P}_k H_k^T + R_k} \quad (5.5)$$

$$x_k = \hat{x}_k + K_k(z_k - H_k \hat{x}_k) \quad (5.6)$$

$$P_k = (I - K_k H_k)\hat{P}_k \quad (5.7)$$

Donde aparecen nuevas variables las cuales se pueden definir así:

-$\hat{x}_k$ es el estado estimado.

-P_k es la matriz de covarianza del error, y $\hat{P}_k$ la matriz de covarianza estimada del error.

-K_k es la ganancia del Kalman, la cual es calculada en cada iteración.

-I es una matriz identidad.

Con estas ecuaciones se ha implementado en Arduino el filtro Kalman, de forma que el programa obtiene las coordenadas geográficas a partir del GPS, y estas pasan al filtro que devuelve un vector de estados con las coordenadas filtradas.

Existe la posibilidad de enviar también al filtro la velocidad obtenida a partir del GPS, ya que 2 de los estados utilizados en el filtro son las velocidades en las 2 coordenadas, aunque esto no se ha realizado debido a la poca fiabilidad que tiene la velocidad devuelta por el GPS.

Al igual que se hizo anteriormente se ha realizado una prueba del GPS realizando un recorrido en un vehículo y recogiendo datos en Arduino, con estos datos se introducen posteriormente en Matlab y se representan en una gráfica para después superponerla en un mapa mediante Google Earth.

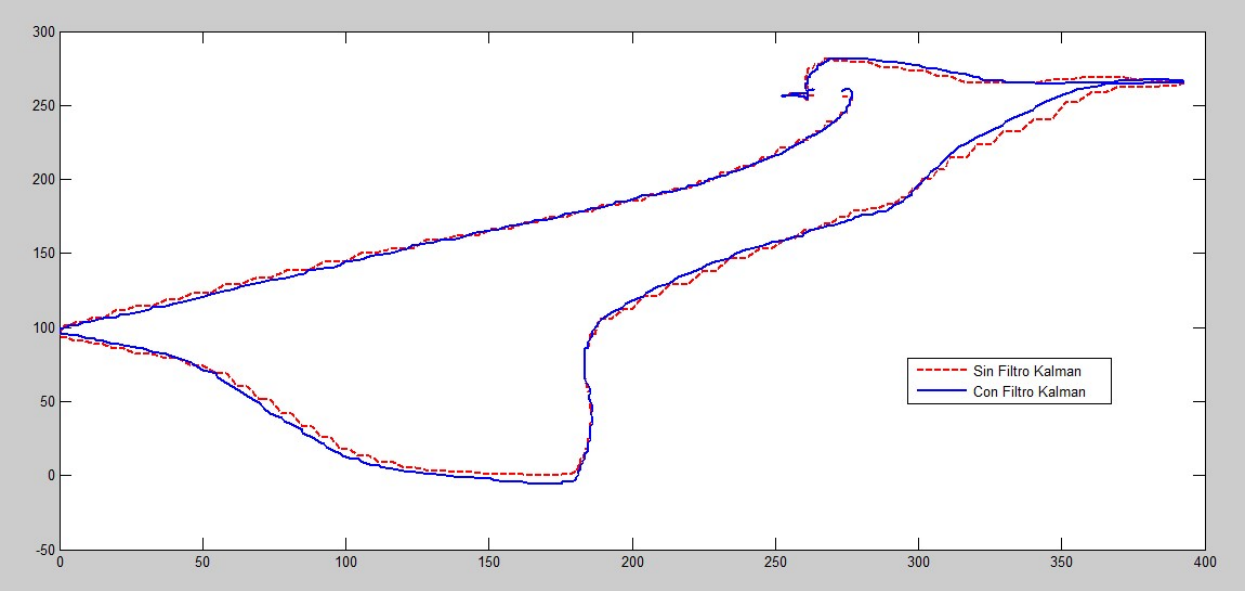


Figura 5.13: Superposición de resultados con y sin filtro Kalman

En la figura 5.13 se muestran los resultados obtenidos tanto con filtro Kalman como sin filtro, se puede apreciar cierta diferencia entre ambas líneas. Como resultado positivo se puede ver que mientras que la línea sin filtro zigzaguea (siendo evidente que el coche no lo hacía y por lo tanto es debido a un error en las mediciones del GPS), la señal filtrada no lo hace, siendo bastante más suave y más real respecto al recorrido del coche.

También se puede apreciar un aspecto negativo del filtro, en la parte superior derecha de la figura se puede apreciar que el filtro Kalman provoca un pequeño error, debido principalmente a la inexactitud de las coordenadas GPS recibidas.

Otro error bastante visible es que el punto de comienzo y el punto final del recorrido según el GPS no coinciden, siendo que en realidad sí que debieran coincidir, esto se debe a errores de las coordenadas que da el GPS.

En la figura 5.14 se muestran los resultados superpuestos con Google Earth.

Figura 5.14: Superposición de los resultados con y sin filtro en Google Earth

En esta imagen se puede ver la inexactitud que puede tener el GPS, que más o menos se encuentra dentro del límite aceptable para el correcto funcionamiento del sistema de control.

CAPÍTULO 6: PROCESSING

Processing es un lenguaje de programación de código abierto (open source) con un entorno propio de desarrollo integrado. Su punto fuerte es gran capacidad para la representación visual, que puede ayudar para métodos de enseñanza y aprendizaje, motivando a los estudiantes de programación.

El lenguaje de programación que utiliza está basado en Java, aunque se ha simplificado y orientado a la programación gráfica.

Sus ventajas principales son su facilidad de uso, y también su potencia, que permite la realización de aplicaciones muy complejas. Además también se puede utilizar complementariamente con Arduino, creando aplicaciones a partir de datos que provienen de este.

Processing fue creado en 2001 por Casey Reas y por Benjamin Fry, como un proyecto para el MIT media lab con el objetivo de ser utilizado en la enseñanza con artes gráficos, posteriormente, cuando el proyecto avanzó, se unió Daniel Shiffman para crear la fundación Processing.

Figura 6.1: Logo de Processing

El entorno de programación utiliza dos formas de representar los resultados, una consola para mostrar resultados escritos, tal y como hace la mayor parte de los entornos de programación, y otra pantalla donde se pueden representar imágenes en 2D o 3D y que es la singularidad de Processing.

6.1 PROGRAMACIÓN EN PROCESSING

Como se ha comentado anteriormente, Processing dispone de un entorno de desarrollo integrado (IDE, de sus siglas en inglés Integrated Development Environment) el cual es conocido por PDE (Processing Development Environment).

Los programas de Processing tienen extensión .pde, la cual es únicamente utilizada para Processing.

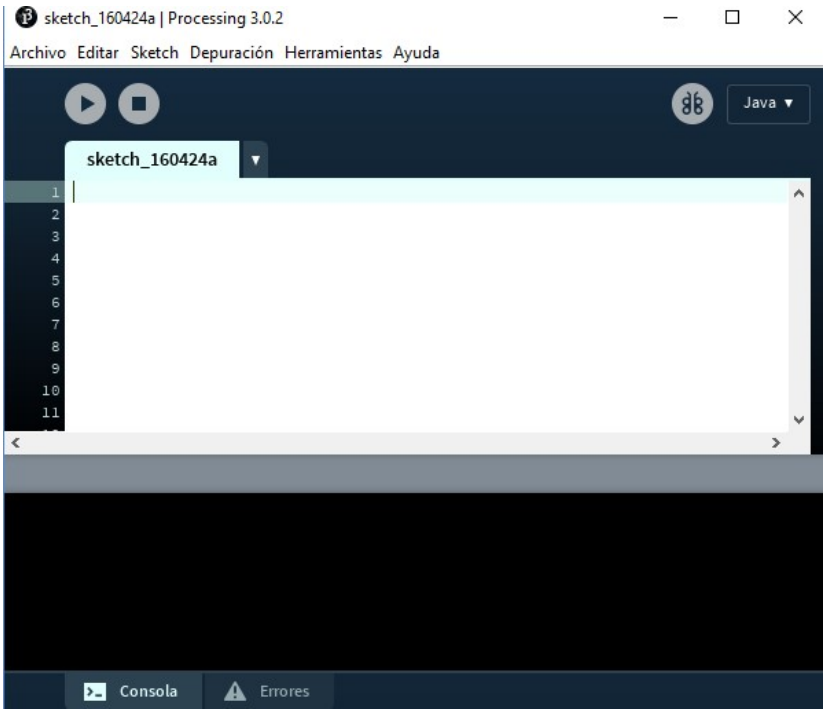


Figura 6.2: Pantalla de inicio del PDE

Como se puede ver en la figura 6.2, Processing dispone de una zona donde se debe escribir el código del programa, y una consola en la parte inferior donde se muestran los resultados que da la aplicación o los errores de compilación o que surgen durante el desarrollo del programa.

También dispone de 2 botones principales, uno para ejecutar ("Run") y otro para detener el programa ("Stop"), cuando una aplicación de Processing se ejecuta se abre una ventana donde se representan gráficamente los resultados, el tamaño de esta ventana se debe elegir mediante programación. Al detener el programa con el botón de "stop" se cierra dicha ventana.

El lenguaje de programación que se debe utilizar es muy similar a java y a C++, las expresiones y funciones básicas tienen prácticamente la misma forma. La programación es más sencilla que en otros entornos ya que se nos permite usar una sintaxis "más relajada".

Al igual que en Arduino, Processing tiene 2 funciones principales que realizan una funcionalidad muy similar de las de Arduino, sin embargo, a

diferencia de Arduino, no es necesario la presencia de estas funciones para tener un programa valido. Estas funciones son la Setup y Draw.

La función Setup tiene exactamente la misma funcionalidad que la función Setup de Arduino, se ejecuta únicamente al principio de la aplicación, y sirve para inicializar el programa y otras funciones o variables si fuera necesario.

En esta función es típico inicializar el tamaño de la ventana grafica utilizando el comando "Size".

La función Draw también es muy similar a la función Loop de Arduino, pues ambas se ejecutan repetitivamente tras cada iteración. La diferencia principal entre ambas es que la función Draw está especialmente creada para el diseño gráfico de la ventana de la aplicación en Processing, en un programa con gráficos animados. Esta función podría no aparecer en un programa de Processing, ya que el dibujo puede ser estático y no necesitar que se realicen infinitas iteraciones.

Además de estas funciones principales, Processing también tiene unas funciones especiales para su entorno, que pueden ser de utilidad para su uso en cualquier programa, por ejemplo la función "Keypressed", que se ejecuta únicamente cuando estando la ventana grafica abierta, se pulsa una tecla, además se puede elegir que tecla es la que tiene que estar pulsada para que se ejecute dicha función.

También dispone de otras funciones específicas para dibujar diferentes formas, tales como círculos, elipses o cubos, de forma sencilla. Por ejemplo, con el siguiente código podemos crear la imagen de la figura 6.3:

```
size(150,100);
quad(61,60, 94,60, 99,83, 81,90);
rect(10,10,60,60);
ellipse(90,30,60,60);
triangle(12,50, 120,15, 125,60);
```

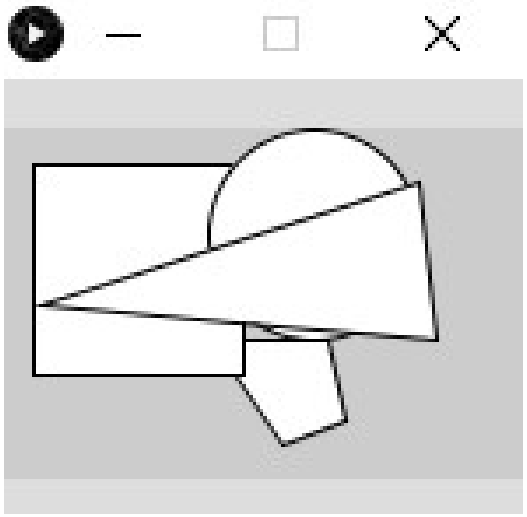


Figura 6.3: Representación de figuras geométricas en Processing

Otra forma de representar formas geométricas es la combinación de los comandos "beginShape" y "endShape", que funcionan tanto en 2D como en 3D, para ello es necesario utilizar también el comando "vertex", este comando crea los vértices de las figuras geométricas en los puntos deseados.

6.2 REPRESENTACIÓN DE LA ORIENTACIÓN DE LA IMU EN PROCESSING

Processing será útil en este proyecto para representar de forma visual los resultados obtenidos en Arduino, en este caso se representara la orientación de la IMU en el espacio a partir de los ángulos de Euler que se pasarán continuamente desde Arduino mediante el puerto serie (USB).

Evidentemente la representación tendrá lugar en 3 dimensiones, por lo que en la función "setup" se inicializará una ventana con capacidad para representar el 3D. En esta función setup también se realizaran otras funciones como cargar una fotografía de la IMU GY-80 e inicializar la comunicación en serie para conectar con Arduino.

El resto de funciones se aplicaran en la función "draw", lo primero será recibir en cada iteración nuevos datos del Arduino, en este caso los valores de los ángulos pitch, roll y yaw. Posteriormente se dibuja una caja con el comando box de unas dimensiones más o menos proporcionales a las de la imu, y se pega una fotografía de la GY-80 en la cara superior de la caja.

El truco para mostrar la orientación no será girar el prisma, sino que este permanecerá quieto y en la misma posición, siendo el punto de visión del prisma lo que se moverá, por lo que simulará que el objeto está girando.

Dado que el ángulo yaw que Arduino nos da es global (como una brújula) respecto la tierra (como una brújula), su visión en la pantalla del ordenador debiera depender de la orientación de la propia pantalla, pero esto es preferible que no tenga que ser así, por lo tanto se ha incluido una función que crea un offset del yaw para inicializarlo cuando se desee, y por lo tanto alinear la posición actual de la IMU con la posición representada en Processing.

Para visualizar y comparar el diferente comportamiento de los 2 métodos utilizados para calcular los ángulos de Euler, se ha hecho también que al pulsar las teclas "c" o "m" cambien a los métodos de cuaterniones o matriz DCM respectivamente.

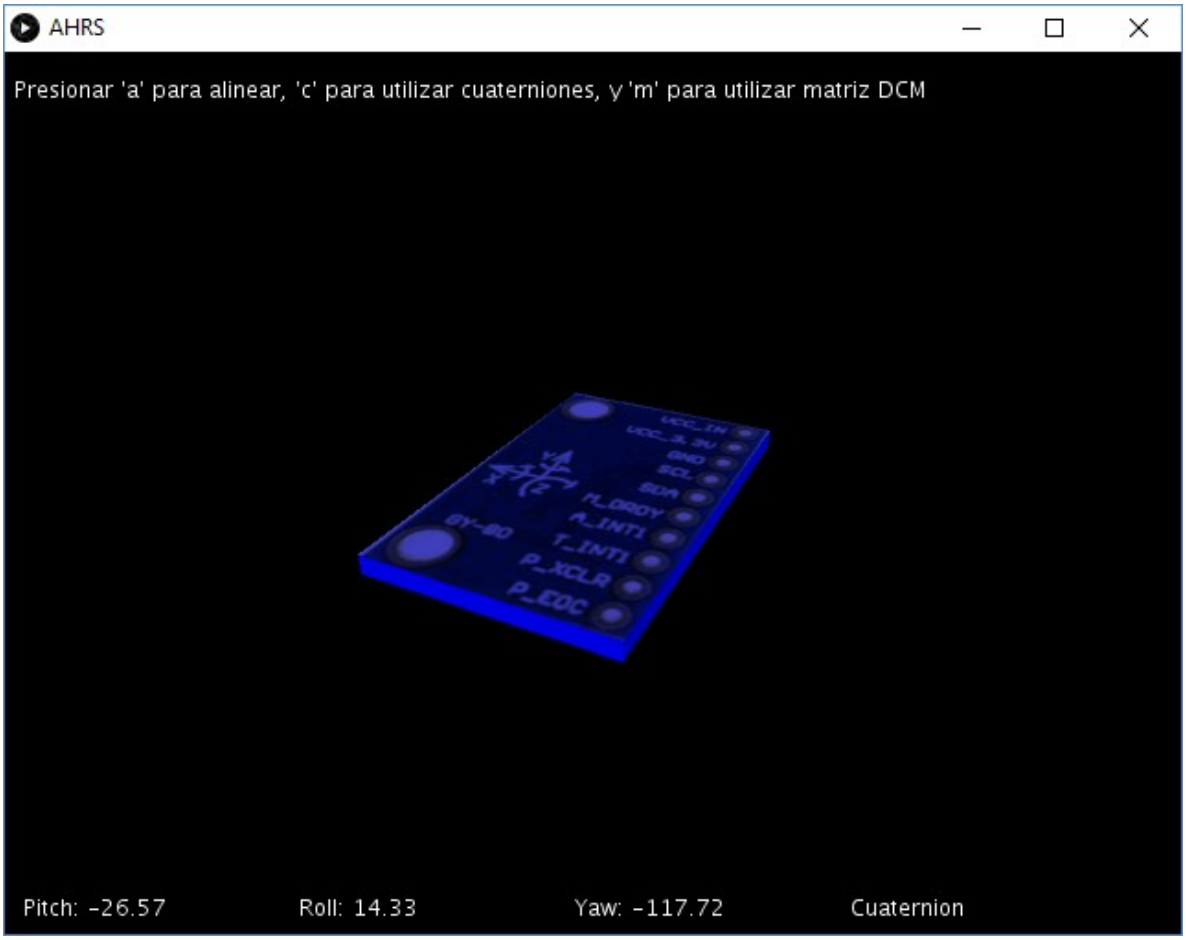


Figura 6.4: Ventana gráfica de Processing representando la orientación de la IMU

En la figura 6.4 se puede ver como resulta la representación en Processing, además de representar la IMU GY80 también se incluyen los valores de los ángulos de Euler, así como el método utilizado en ese momento.

El código escrito para la realización de este programa se encuentra en el anexo de códigos correspondiente, donde se podrá ver más detalladamente lo que hace el programa.

Se ha realizado un video para poder ver el programa en funcionamiento, dicho video está alojado en la siguiente dirección:

https://www.youtube.com/watch?v=PxecUCjxWwI

CAPÍTULO 7: CONCLUSIONES

Al finalizar el proyecto se ha conseguido la obtención y transmisión de datos muy importantes para el control de un vehículo autónomo mediante dispositivos de bajo costo. El precio aproximado de todos los dispositivos utilizados en este proyecto se puede ver en la tabla 7.1.

Elemento	Precio aproximado
Arduino Due	40€
IMU GY80	20€
GPS Lonet SIM808	40€
Batería 3.7V	10€

Tabla 7.1: Precio aproximado de los dispositivos utilizados

Aunque lo realizado en este proyecto solo supone una parte de un proyecto mucho mayor, es una pieza indispensable para la consecución de un vehículo autónomo, y además podría ser aplicado en otros proyectos como por ejemplo el control de un dron.

7.1 ANÁLISIS DE LO RESULTADOS OBTENIDOS

Una vez realizados todas las pruebas y escrito todo el código de programación se ha unido tanto el programa de cálculo de los ángulos de Euler como el de obtención de coordenadas GPS.

Ambas funciones se ejecutan de forma distinta en cuanto a la gestión del tiempo se refiere, el GPS devuelve las coordenadas cada un cierto tiempo más o menos constante, que es de aproximadamente de 400 ms. Mientras que los datos de los ángulos de Euler son devueltos cada vez que se ejecuta el loop, y este tiempo depende bastante del tipo de comunicación utilizado (esta entre 11 y 30 ms).

En el caso de que se utilice la comunicación serie a 19200 baudios, se reciben unos 25 valores de los ángulos de Euler por cada valor de las coordenadas GPS, mientras que si se utiliza la comunicación UDP se recibirán unos 37 valores de los ángulos por cada uno del GPS.

En la figura 7.1 se muestran los resultados que se obtienen ejecutando el programa utilizando la comunicación por USB.

Figura 7.1: Resultados obtenidos mediante comunicación serie

Esta diferencia de tiempo entre lo que se tarda en recibir un dato u otro se puede reducir, pero se ha decidido utilizar así para aprovechar al máximo el Arduino y sus posibilidades, manteniendo al mínimo el periodo de muestreo en el cálculo de los ángulos de Euler para disminuir el error cometido en las integraciones.

En cuanto a la calidad y exactitud de los datos obtenidos es aceptablemente buena, en el caso de los ángulos de Euler es difícilmente mejorable, tal vez el uso de un filtro Kalman en lugar de un filtro complementario conseguiría mejores resultados a la hora del filtrado, pero los cuaterniones funcionan de una forma bastante correcta y el filtro complementario mejora los resultados.

El GPS sin embargo se podría conseguir mejores resultados en cuanto a precisión y tiempo de espera para la fijación de satélites utilizando un módulo GPS de mayor calidad. Aunque con el módulo utilizado se puede llevar a cabo este proyecto con aceptables resultados ya que tampoco se necesita una precisión mucho mayor.

7.2 CONTINUACIÓN DE LA LINEA DE INVESTIGACIÓN

Para continuar este línea de investigación del departamento de control de la Universidad de Cantabria que pretende conseguir un coche autónomo, uno de los pasos futuros que se podrían dar sería utilizar los datos obtenidos de la IMU para conseguir un rastreo 3D del movimiento de la IMU y del coche.

Para ello, una forma sería utilizar los datos de los acelerómetros para obtener la posición mediante una doble integración, aunque esto da resultados muy negativos debido a los errores que se acumulan. Mediante un doble filtro paso alto (uno para la velocidad y otro para la posición) se pueden mejorar mucho estos resultados, aunque aún este avance seguirían teniendo un error importante.

También sería posible utilizar los datos de los acelerómetros para realizar un filtrado Kalman junto a las coordenadas GPS, con el mismo filtro Kalman que se ha utilizado para este, añadiendo más estados y más por lo tanto dando mayor complejidad al filtro.

Otra forma de obtener esta información sería utilizar sensores de medida de por ejemplo la velocidad de las ruedas, para calcular cuánto se ha desplazado el vehículo, y mediante los datos recibidos por la IMU se puede deducir la dirección de ese movimiento.

Si se consigue realizar ese cálculo del movimiento del vehículo se podrían filtrar los resultados obtenidos con los datos recibidos del GPS, obteniendo así mejores y más fiables resultados.

También se deben conseguir la medición y transmisión de datos del resto de sensores que debe utilizar el vehículo como cámaras, ultrasonidos, etc...

Además se deberían implementar los actuadores necesarios para tomar el control del vehículo, tales como un servomotor que controle los giros del volante, una bomba hidráulica que controle el circuito de frenos, u un controlador para el motor eléctrico del que dispone el coche.

CAPÍTULO 8: BIBLIOGRAFÍA

[1] Antonio Barrientos, Luis Felipe Peñin, Carlos Balanguer, Rafael Aracil. "Fundamentos de Robotica". 2ª edición, McGraw Hill 2007.

[2] Ignacio Angulo Martínez, Mikel Etxebarria Isuskiza, Jose Mª Angulo Usategui. "GPS, compás. sónar, Rfid, control de motores e internet". Creaciones Copyright 2009.

[3] William Premerlani, Paul Bizard. "Direction Cosine Matriz IMU: Theory".

[4] Starlino. "DCM Tutorial – An Introduction to Orientation Kinematics". [http://www.starlino.com/dcm_tutorial.html].

[5] Jose Luis Ganoza Quinta, "Propiedades algebraricas e implicaciones geométricas de los cuaterniones". Universidad de Cantabria.

[6] Sebastian O.H. Madgwick, Andrew J.L. Harrison, Ravi Vaidyanathan. "Estimation of IMU and MARG orientation using a gradient descent algorithm".

[7] Sebastian O.H. Madgwick. "An efficient orientation filter for inertial and inertial/magnetic sensor arrays" April 2010.

[8] http://www.wired.com/2012/02/autonomous-vehicle-history/

[9] https://www.arduino.cc/

[10] https://es.wikipedia.org/wiki/Filtro_de_Kalman

[11] https://processing.org/

ANEXO A: CÓDIGOS GENERADOS

A.1 ARDUINO

A.1.1 Librería GY-80

Esta librería incluye la programación necesaria para utilizar todos los sensores de la IMU GY-80, y la programación para calcular mediante matriz DCM o mediante cuaterniones los 3 ángulos de Euler.

Para la utilización de esta librería hace falta llamarla desde un archivo ".ino" de Arduino, para ello hace falta introducir la siguiente línea:

GY80(int UDP, int METODO)

Donde UDP es un parámetro para elegir la comunicación de salida de datos, si UDP es 0 se utilizará la comunicación serie, mientras que si se inicializa UDP con 1, la comunicación será mediante cable Ethernet.

METODO establece el método utilizado para calcular los ángulos de Euler, si se escoge el valor 0, el programa realiza un filtro complementario utilizando cuaterniones, y si se escoge 1, se usaran las matrices DCM junto a un regulador PI. En el caso de que se utilice la comunicación serie es posible cambiar entre ambos métodos durante la ejecución del programa.

Para la inicialización de la IMU es necesario emplear la función "Inicializar" en el "setup" de Arduino, esta función necesita un parámetro de entrada, si este parámetro se configura a 0, no devolverá los ángulos de Euler a la salida, si en cambio se configura a 1, al finalizar su ejecución imprimirá los 3 ángulos de Euler mediante a comunicación escogida anteriormente.

Lo que esta función hará será iniciar la comunicación I2C con los sensores y configurar los registros necesarios para establecer el funcionamiento deseado. Para cambiar esa configuración será necesario modificar los valores de los registros en el archivo "GY80.h".

La inicialización del barómetro ha quedado comentada debido a que no será utilizada.

Por ultimo inicializa también el método de cálculo de los ángulos de Euler escogido cuando se llamó a la librería, para ello será necesario realizar la primera lectura de las mediciones de los sensores, esto se consigue utilizando la función "getSensor", función la cual actualiza en las variables respectivas el valor de las 3 medidas del acelerómetro, el giróscopo y el magnetómetro.

Para el cálculo en cada iteración de los ángulos de Euler habrá que incluir en el "loop" del programa la función "getEuler", la cual también necesita la introducción de un parámetro para mostrar los ángulos de Euler a la salida, de la misma forma que lo hacia la función "Inicializar".

Esta función lo único que hace es llamar a la función del método de cálculo que se va a utilizar en esa iteración, e inicializa el otro método en

el caso que mediante la utilización de un comando por comunicación serie se decida cambiar el método de cálculo.

Además de estas funciones principales la librería utiliza muchas funciones complementarias para conseguir funcionar, entre ellas se pueden encontrar algunas para la comunicación I2C, que se utilizan para leer o escribir en los registros de los sensores, o funciones para operaciones matriciales.

A continuación se encuentra un listado de las funciones públicas, es decir, que pueden ser utilizadas desde el programa de Arduino. Las funciones privadas en cambio solo podrán ser utilizadas desde el interior de la librería.

Funciones públicas:

- **Inicializar(int ON)** Utilizada en el "setup" para inicializar todos los sensores y las comunicaciones necesarias. Utiliza un parámetro para habilitar o deshabilitar la salida de datos en su ejecución.
- **getAcce()** Lee los valores medidos por el acelerómetro y calibra con los datos de offset y ganancia.
- **getGiro()** Lee los valores medidos por el giróscopo y calibra con los datos de offset.
- **getMagn()** Lee los valores medidos por el magnetómetro y calibra con los datos de offset y ganancia.
- **getPres()** Lee los valores medidos por el barómetro y calcula presión y temperatura.
- **getSensor()** Llama conjuntamente a las lecturas de todos los sensores de la IMU.
- **PrintSensors()** Imprime todo las lecturas de los sensores mediante el método de comunicación escogido previamente.
- **getEuler(int ON)** Se debe utilizar en el "setup" para obtener los valores de los ángulos de Euler, se debe introducir un parámetro para elegir si imprime los resultados obtenidos o no.
- **getPitchRoll()** Función que obtiene los valores del pitch y del roll a partir de los datos de acelerómetro.
- **getYaw()** Obtiene el valor del yaw a partir de las lecturas del magnetómetro y del pitch y el roll.
- **Inic_Matriz(int ON)** Realiza el cálculo inicial necesario para obtener la primera matriz DCM.
- **MatrizDCM(int ON)** Realiza el cálculo de los ángulos de Euler mediante el método de las matrices DCM, también se incluye la normalización y el filtro.
- **Inic_Cuaternion(int ON)** Inicializa el cálculo de los cuaterniones creando el cuaternion inicial.
- **Cuaternion(int ON)** Realiza el cálculo de los ángulos de Euler mediante el método de los cuaterniones, incluyendo el filtro complementario.
- **CalcAcc()** Realiza el cálculo del rastreo 3D mediante los datos de los acelerómetros, pero se obtienen malos resultados.

Variables públicas:

- **ACC[3]** Array de 3 componentes donde se almacenan los 3 últimos valores obtenidos de las 3 medidas de aceleración.
- **GIR[3]** Array de 3 componentes donde se almacenan los 3 últimos valores obtenidos de las 3 medidas de velocidad de giro.
- **MAG[3]** Array de 3 componentes donde se almacenan los 3 últimos valores obtenidos de las 3 medidas de campo magnético.
- **Dt** Periodo de muestreo de la última iteración en segundos.
- **pitch** Último valor obtenido del pitch.
- **roll** Último valor obtenido del roll.
- **yaw** Último valor obtenido del yaw.
- **Matriz_DCM[3][3]** Array de 9 componentes divididos en 3 filas y 3 columnas que almacena los valores de la matriz DCM.
- **Q[4]** Array de 4 componentes que almacena los valores del cuaternion principal que representa la orientación actual.
- **acc[3]** Array con los valores de aceleración dinámica en las 3 direcciones.
- **vel[3]** Array con los valores de velocidad en las 3 direcciones.
- **pos[3]** Array con los valores de posición en las 3 direcciones.

"GY80.h"

```
/*
GY80.cpp - Librería para la imu GY-80. Realizada por Daniel López Montes.
*/

//LIBRERIAS
#include "Arduino.h"
#include <Wire.h>
#include <SPI.h>
//#include <Ethernet.h>
//#include <EthernetUdp.h>

//DIRECCIONES I2C SENSORES
#define ADXL345                         (0x53)
#define L3G4200D                        (0x69)
#define HMC5883L                        (0x1E)
#define BMP085                          (0x77)

//REGISTROS ADXL345
#define ACC_REG_BW_RATE                 (0x2C)
#define ACC_REG_POWER_CTL               (0x2D)
#define ACC_REG_DATA_FORMAT             (0x31)
#define ACC_REG_DATAX0                  (0x32)
#define ACC_REG_DATAX1                  (0x33)
#define ACC_REG_DATAY0                  (0x34)
#define ACC_REG_DATAY1                  (0x35)
#define ACC_REG_DATAZ0                  (0x36)
#define ACC_REG_DATAZ1                  (0x37)

#define ACC_BW_RATE                     (0b1001) //Frecuencia 50Hz
#define ACC_POWER_CTL                   (0x08)
#define ACC_DATA_FORMAT                 (0x01) //Rango +- 4g
```

```
//REGISTROS L3G4200D
#define GIR_REG_1                (0x20)
#define GIR_REG_2                (0x21)
#define GIR_REG_3                (0x22)
#define GIR_REG_4                (0x23)
#define GIR_REG_5                (0x24)
#define GIR_REG_DATAX0           (0x28)
#define GIR_REG_DATAX1           (0x29)
#define GIR_REG_DATAY0           (0x2A)
#define GIR_REG_DATAY1           (0x2B)
#define GIR_REG_DATAZ0           (0x2C)
#define GIR_REG_DATAZ1           (0x2D)

#define GIR_1                    (0b00001111)
#define GIR_2                    (0b00000000)
#define GIR_3                    (0b00001000)
#define GIR_4                    (0b00110000)
#define GIR_5                    (0b00000000)

//REGISTROS HMC5883L
#define MAG_REG_CONFIG_A         (0x00)
#define MAG_REG_CONFIG_B         (0x01)
#define MAG_REG_MODE             (0x02)
#define MAG_REG_DATAX0           (0x03)
#define MAG_REG_DATAX1           (0x04)
#define MAG_REG_DATAY0           (0x07)
#define MAG_REG_DATAY1           (0x08)
#define MAG_REG_DATAZ0           (0x05)
#define MAG_REG_DATAZ1           (0x06)

#define MAG_CONFIG_A             (0x10)
#define MAG_CONFIG_B             (0xE0)
#define MAG_MODE                 (0x00)

//REGISTROS BMP085
#define BMP085_CAL_AC1           (0xAA)
#define BMP085_CAL_AC2           (0xAC)
#define BMP085_CAL_AC3           (0xAE)
#define BMP085_CAL_AC4           (0xB0)
#define BMP085_CAL_AC5           (0xB2)
#define BMP085_CAL_AC6           (0xB4)
#define BMP085_CAL_B1            (0xB6)
#define BMP085_CAL_B2            (0xB8)
#define BMP085_CAL_MB            (0xBA)
#define BMP085_CAL_MC            (0xBC)
#define BMP085_CAL_MD            (0xBE)

#define BMP085_CONTROL           (0xF4)
#define BMP085_TEMPDATA          (0xF6)
#define BMP085_PRESSUREDATA              (0xF6)
#define BMP085_READTEMPCMD               (0x2E)
#define BMP085_READPRESSURECMD           (0x34)

//CONSTANTES
#define CONSTANTE_ADXL345                (0.0766)          // 9,81(m/s^2/g)*(1/128)(g/LSB) -> m/s^2
```

```
#define CONSTANTE_L3G4200D          (0.07)    // 70(mdeg/s)/LSB *(1/1000) (deg/mdeg) -> deg/s
#define CONSTANTE_HMC5883L          (0.000271)        // (1/440*16) G/LSB -> G +

//DATOS CALIBRACION
#define ACCEL_X_MIN                             ((float) -10.2)
#define ACCEL_X_MAX                             ((float) 9.18)
#define ACCEL_Y_MIN                             ((float) -10)
#define ACCEL_Y_MAX                             ((float) 9.5)
#define ACCEL_Z_MIN                             ((float) -8.8)
#define ACCEL_Z_MAX                             ((float) 10.25)
#define GRAVEDAD                                ((float) 9.81)
#define ACCEL_X_OFFSET                          ((ACCEL_X_MIN + ACCEL_X_MAX) / 2)
#define ACCEL_Y_OFFSET                          ((ACCEL_Y_MIN + ACCEL_Y_MAX) / 2)
#define ACCEL_Z_OFFSET                          ((ACCEL_Z_MIN + ACCEL_Z_MAX) / 2)
#define ACCEL_X_SCALE                           (GRAVEDAD / (ACCEL_X_MAX - ACCEL_X_OFFSET))
#define ACCEL_Y_SCALE                           (GRAVEDAD / (ACCEL_Y_MAX - ACCEL_Y_OFFSET))
#define ACCEL_Z_SCALE                           (GRAVEDAD / (ACCEL_Z_MAX - ACCEL_Z_OFFSET))

#define OffsetGiroX                             (1.66)
#define OffsetGiroY                             (0.65)
#define OffsetGiroZ                             (-0.54)

#define OffsetMagnX                             (0.8325)
#define OffsetMagnY                             (-0.4860)
#define OffsetMagnZ                             (-2.22)
#define EscalaMagnX                             (0.0910)
#define EscalaMagnY                             (0.0928)
#define EscalaMagnZ                             (0.1186)

//Ganancias control PI para matriz DCM
#define Kp_ROLLPITCH                            (0.02)
#define Ki_ROLLPITCH                            (0.00002)
#define Kp_YAW                                  (0.8)
#define Ki_YAW                                  (0.00002)


class GY80
{
        public: GY80(int UDP, int METODO); // CONSTRUCTOR
                //IMU
                void    Inicializar(int ON);
                void    getSensor(void);
                void    getAcce(void);
                void    getGiro(void);
                void    getMagn(void);
                void    getPres(void);
                void    PrintSensors(void);
                float   ACC[3], GIR[3], MAG[3];
                float   Dt;

                //Angulos de Navegacion
                void    getEuler(int ON);
                void    getPitchRoll(void);
                void    getYaw(void);
                float   pitch, roll, yaw;
```

```
                //MATRIZ DCM
                void    Inic_Matriz(int ON);
                void    MatrizDCM(int ON);
                float   Matriz_DCM[3][3];

                //Cuaterniones
                float   Q[4];
                void    Inic_Cuaternion(int ON);
                void    Cuaternion(int ON);

                //CALCULAR ACELERACION
                void    CalcAcc(void);
                float   acc[3],vel[3],pos[3];

        private:
                float   ToDeg(float rad);
                float   ToRad(float deg);
                float   pi = 3.141592;
                void    writeTo(byte DireccionSensor, byte address, byte val);
                int     readReg8(int deviceAddress, byte address);
                int     readReg16(int deviceAddress, byte address);
                int     readRegInv16(int deviceAddress, byte address);

                void    Normalizar(void);
                void    CorreccionDrift(void);
                void    Vector_Suma(float v1[3], float v2[3]);
                float   Producto_Escalar(float v1[3], float v2[3]);
                void    Producto_Vectorial(float v1[3], float v2[3]);
                void    Mult_Matriz3x3(float Mat1[3][3], float Mat2[3][3]);

                long    tiempo;
                float   Matriz_Rot[3][3];
                float   vectorOut[3];
                float   W_P[3], W_I[3], W[3];
                float   Q_Giro[4], Q_Fil[4];
                float   theta, mag, ang, sq;
                float   Temperatura;
                long    Presion;
                int     _UDP, _Metodo, _read;
                int     BMP085_AC1,  BMP085_AC2,  BMP085_AC3,  BMP085_AC4,  BMP085_AC5,
BMP085_AC6, BMP085_B1, BMP085_B2, BMP085_MB, BMP085_MC, BMP085_MD;
                int     PortPc = 8000;
                float   magx,magy;
                void    FiltroCuat(void);
                float   ACC2[3], acct[3];
                float   _2q0mx, _2q0my, _2q0mz, _2q1mx, _2bx, _2bz, _4bx, _4bz, _2q0, _2q1, _2q2, _2q3,
_2q0q2, _2q2q3, q0q0, q0q1, q0q2, q0q3, q1q1, q1q2, q1q3, q2q2, q2q3, q3q3;

};
```

"GY80.cpp"

```
/*
GY80.cpp - Librería para la IMU GY-80. Realizada por Daniel López Montes.
*/


#include "Arduino.h"
#include <Wire.h>
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
#include "GY80.h"
EthernetUDP Udp;
IPAddress IPPc(192, 168, 1, 2);

GY80::GY80(int UDP, int METODO)
{
	_UDP = UDP;
	_Metodo = METODO;

}

void GY80::Inicializar(int ON)
{
	delay(25);
	Wire.begin();

	if (_UDP!=0){
		//Inicializar comunicacion UDP
		byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
		IPAddress IPArduino(192, 168, 1, 177);
		unsigned int PortArduino = 4000;
		Ethernet.begin(mac,IPArduino);
		Udp.begin(PortArduino);
		delay(500);
	}
	if (_UDP!=1) {
		Serial.begin(19200);
	}

	//Acelerometro
	writeTo(ADXL345, ACC_REG_BW_RATE, ACC_BW_RATE);
	writeTo(ADXL345, ACC_REG_POWER_CTL, ACC_POWER_CTL);
	writeTo(ADXL345, ACC_REG_DATA_FORMAT, ACC_DATA_FORMAT);

	//Giroscopo
	writeTo(L3G4200D, GIR_REG_1, GIR_1);
	writeTo(L3G4200D, GIR_REG_2, GIR_2);
	writeTo(L3G4200D, GIR_REG_3, GIR_3);
	writeTo(L3G4200D, GIR_REG_4, GIR_4);
	writeTo(L3G4200D, GIR_REG_5, GIR_5);
	delay(25);

	//Magnetometro
```

```
	writeTo(HMC5883L, MAG_REG_CONFIG_A, MAG_CONFIG_A);
	writeTo(HMC5883L, MAG_REG_CONFIG_B, MAG_CONFIG_B);
	writeTo(HMC5883L, MAG_REG_MODE, MAG_MODE);

	delay(25);

	//Barometro
	/*
	BMP085_AC1 = (short)(readRegInv16(BMP085, BMP085_CAL_AC1)); //Serial.print(BMP085_AC1);
Serial.print(" ; ");
	BMP085_AC2 = (short)(readRegInv16(BMP085, BMP085_CAL_AC2)); //Serial.print(BMP085_AC2);
Serial.print(" ; ");
	BMP085_AC3 = (short)(readRegInv16(BMP085, BMP085_CAL_AC3)); //Serial.print(BMP085_AC3);
Serial.print(" ; ");
	BMP085_AC4 = readRegInv16(BMP085, BMP085_CAL_AC4); //Serial.print(BMP085_AC4);
Serial.print(" ; ");
	BMP085_AC5 = readRegInv16(BMP085, BMP085_CAL_AC5); //Serial.print(BMP085_AC5);
Serial.print(" ; ");
	BMP085_AC6 = readRegInv16(BMP085, BMP085_CAL_AC6); //Serial.print(BMP085_AC6);
Serial.print(" ; ");
	BMP085_B1 = (short)(readRegInv16(BMP085, BMP085_CAL_B1)); //Serial.print(BMP085_B1);
Serial.print(" ; ");
	BMP085_B2 = (short)(readRegInv16(BMP085, BMP085_CAL_B2)); //Serial.print(BMP085_B2);
Serial.print(" ; ");
	BMP085_MB = (short)(readRegInv16(BMP085, BMP085_CAL_MB)); //Serial.print(BMP085_MB);
Serial.print(" ; ");
	BMP085_MC = (short)(readRegInv16(BMP085, BMP085_CAL_MC)); //Serial.print(BMP085_MC);
Serial.print(" ; ");
	BMP085_MD = (short)(readRegInv16(BMP085, BMP085_CAL_MD)); //Serial.print(BMP085_MD);
Serial.print(" ; ");
	delay(50); */

	//Inicializar matriz de calculo de los ángulos.
	tiempo=millis();
	if (_Metodo == 0) {
		Inic_Cuaternion(ON);
	}
	else {
	Inic_Matriz(ON);
	}

	delay(50);
}

void GY80::getSensor(void)
{
	getAcce();
	getGiro();
	getMagn();
	//getPres();
}

void GY80::getAcce(void)
{

	short x = readReg16(ADXL345, ACC_REG_DATAX0);
```

```
	short y = readReg16(ADXL345, ACC_REG_DATAY0);
	short z = readReg16(ADXL345, ACC_REG_DATAZ0);

	ACC[1]=(x*CONSTANTE_ADXL345-ACCEL_Y_OFFSET)*ACCEL_Y_SCALE;
	ACC[0]=(y*CONSTANTE_ADXL345-ACCEL_X_OFFSET)*ACCEL_X_SCALE;
	ACC[2]=-((z*CONSTANTE_ADXL345+ACCEL_Z_OFFSET)*ACCEL_Z_SCALE);
	//Aceleracion sin Calibracion
	/*
	ACC[1]=(x*CONSTANTE_ADXL345);
	ACC[0]=(y*CONSTANTE_ADXL345);
	ACC[2]=-(z*CONSTANTE_ADXL345); //*/
}

void GY80::getGiro(void)
{

	byte GIR_DATA1 = readReg8(L3G4200D, GIR_REG_DATAX1);
	byte GIR_DATA0 = readReg8(L3G4200D, GIR_REG_DATAX0);
	short x = ((GIR_DATA1 << 8) | GIR_DATA0);
	byte GIR_DATA3 = readReg8(L3G4200D, GIR_REG_DATAY1);
	byte GIR_DATA2 = readReg8(L3G4200D, GIR_REG_DATAY0);
	short y = ((GIR_DATA3 << 8) | GIR_DATA2);
	byte GIR_DATA5 = readReg8(L3G4200D, GIR_REG_DATAZ1);
	byte GIR_DATA4 = readReg8(L3G4200D, GIR_REG_DATAZ0);
	short z = ((GIR_DATA5 << 8) | GIR_DATA4);

	GIR[1]=(x*CONSTANTE_L3G4200D)-OffsetGiroX;
	GIR[0]=(y*CONSTANTE_L3G4200D)-OffsetGiroY;
	GIR[2]=-((z*CONSTANTE_L3G4200D)-OffsetGiroZ);

	//Serial.print(GIR[0]);Serial.print("  ");Serial.print(GIR[1]);Serial.print("  ");Serial.println(GIR[2]);
}

void GY80::getMagn(void)
{

	short x = readReg16(HMC5883L, MAG_REG_DATAX0);
	short y = readReg16(HMC5883L, MAG_REG_DATAY0);
	short z = readReg16(HMC5883L, MAG_REG_DATAZ0);

	MAG[1]= (x*CONSTANTE_HMC5883L-OffsetMagnX)*EscalaMagnX;
	MAG[0]= (y*CONSTANTE_HMC5883L-OffsetMagnY)*EscalaMagnY;
	MAG[2]= -((z*CONSTANTE_HMC5883L-OffsetMagnZ)*EscalaMagnZ);

}

void GY80::getPres(void)
{
	//Temperatura
	writeTo(BMP085, BMP085_CONTROL, BMP085_READTEMPCMD); delay(5);
	long UT = readRegInv16(BMP085, BMP085_TEMPDATA);

	long X1 = (UT-BMP085_AC6) * BMP085_AC5 >> 15;
	long X2 = (BMP085_MC << 11) / (X1 + BMP085_MD);
	Temperatura=((X1 + X2 + 8) >> 4) /10; // Temperatura en ºC
```

```
	//Presión
	int oss = 0; //MODO STANDARD
	writeTo(BMP085, BMP085_CONTROL, BMP085_READPRESSURECMD + (oss << 6)); delay(8);
	long UP = readRegInv16(BMP085, BMP085_PRESSUREDATA); UP <<= 8; UP |= readReg8(BMP085, BMP085_PRESSUREDATA+2); UP >>= (8 - oss);

	long B6 = X1 + X2 - 4000;
	X1 = (BMP085_B2 * (B6 * B6 >> 12)) >> 11;
	X2 = (BMP085_AC2 * B6) >> 11;
	long X3 = X1 + X2;
	long B3 = (((BMP085_AC1 * 4 + X3) << oss) + 2) / 4;
	X1 = (BMP085_AC3 * B6) >> 13;
	X2 = (BMP085_B1 * (B6 * B6 >> 12)) >> 16;
	X3 = ((X1 + X2) + 2) >> 2;
	unsigned long B4 = (BMP085_AC4 * (unsigned long)(X3 + 32768)) >> 15;
	unsigned long B7 = ((unsigned long)UP - B3) * (50000 >> oss);
	unsigned long p;

	if (B7 < 0x80000000) {
		p = (B7 * 2)/B4;
	} else {
		p = (B7 / B4) * 2;
	}

	X1 = (((p >> 8) * (p >> 8)) * 3038) >> 16;
	X2 = (-7357 * (long)p) >> 16;
	Presion = p + (X1 + X2 + 3791); // >>4???

}

void GY80::PrintSensors(void)
{
	getSensor();
	if (_UDP != 0){
		Udp.beginPacket(IPPc, PortPc);
		Udp.print(ACC[0]);Udp.print(";");
		Udp.print(ACC[1]);Udp.print(";");
		Udp.print(ACC[2]);Udp.print(";");
		Udp.print(GIR[0]);Udp.print(";");
		Udp.print(GIR[1]);Udp.print(";");
		Udp.print(GIR[2]);Udp.print(";");
		Udp.print(MAG[0]);Udp.print(";");
		Udp.print(MAG[1]);Udp.print(";");
		Udp.print(MAG[2]);Udp.print(";");
		Udp.endPacket();
	}
	if (_UDP != 1) {
		Serial.print(ACC[0],3);Serial.print(";");
		Serial.print(ACC[1],3);Serial.print(";");
		Serial.print(ACC[2],3);Serial.print(";");
		Serial.print(GIR[0],3);Serial.print(";");
		Serial.print(GIR[1],3);Serial.print(";");
		Serial.print(GIR[2],3);Serial.print(";");
		Serial.print(MAG[0],3);Serial.print(";");
		Serial.print(MAG[1],3);Serial.print(";");
		Serial.print(MAG[2],3);Serial.print(";");
```

```
                //Serial.print(Temperatura,1);Serial.print(";");
                //Serial.print(Presion); Serial.print(";");
                Serial.println();
        }
}

void GY80::getEuler(int ON)
{
        if (_UDP != 1) { //Solo si se utiliza el serial
                if (Serial.available()){
                        while(Serial.available()){
                                _read = (Serial.read());
                                if (_read == 67) { //Introducir C
                                _Metodo = 0;
                                Inic_Cuaternion(ON);
                                }
                                if (_read == 77) { //Introducir M
                                _Metodo = 1;
                                Inic_Matriz(ON);
                                }
                        }
                }
        }
        if (_Metodo == 0) {
                Cuaternion(ON);
        }
        else if (_Metodo == 1) {
                MatrizDCM(ON);
        }
        else {
        Serial.println("Metodo no correcto");
        delay(2000);
        }
}

void GY80::getPitchRoll(void)
{
        pitch = atan2(-ACC[0], (sqrt( ACC[1] * ACC[1] + ACC[2] * ACC[2])));
        roll= atan2(ACC[1], ACC[2]);
}

void GY80::getYaw(void)
{
        magx = MAG[0]*cos(pitch) + MAG[1]*sin(roll)*sin(pitch) + MAG[2]*cos(roll)*sin(pitch);
        magy = - MAG[1]*cos(roll) + MAG[2]*sin(roll);
        yaw = -atan2(-magy, magx);
}

void GY80::Inic_Matriz(int ON)
{
        getSensor();
        getPitchRoll();getYaw();

        if (ON == 1) {
                if (_UDP != 0) {
                        Udp.beginPacket(IPPc, PortPc);
```

```
                        Udp.print(ToDeg(pitch));Udp.print(";");
                        Udp.print(ToDeg(roll));Udp.print(";");
                        Udp.print(ToDeg(yaw));Udp.print(";");
                        Udp.endPacket();
                }
                if (_UDP != 1)  {
                        Serial.print(ToDeg(pitch)); Serial.print(";");
                        Serial.print(ToDeg(roll)); Serial.print(";");
                        Serial.print(ToDeg(yaw)); Serial.println(";1;");
                }
        }

        Matriz_DCM[0][0] = cos(pitch) * cos(yaw);
        Matriz_DCM[0][1] = cos(yaw) * sin(roll) * sin(pitch) - cos(roll) * sin(yaw);
        Matriz_DCM[0][2] = sin(roll) * sin(yaw) + cos(roll) * cos(yaw) * sin(pitch);
        Matriz_DCM[1][0] = cos(pitch) * sin(yaw);
        Matriz_DCM[1][1] = cos(roll) * cos(yaw) + sin(roll) * sin(pitch) * sin(yaw);
        Matriz_DCM[1][2] = cos(roll) * sin(pitch) * sin(yaw) - cos(yaw) * sin(roll);
        Matriz_DCM[2][0] = -sin(pitch);
        Matriz_DCM[2][1] = cos(pitch) * sin(roll);
        Matriz_DCM[2][2] = cos(roll) * cos(pitch);


}

void GY80::MatrizDCM(int ON)
{
        getSensor();
        //Calculo Dt entre cada actualizacion (en milisegundos);
        Dt = float(millis()- tiempo)/1000;
        tiempo = millis();

        for (int c=0; c<3; c++) {W[c]=ToRad(GIR[c])+W_P[c]+W_I[c];}
        //Matriz de Rotacion con Giroscopos y correccion del Drift.
        Matriz_Rot[0][0]=1;
        Matriz_Rot[0][1]=-Dt*W[2];//-z
        Matriz_Rot[0][2]=Dt*W[1];//y
        Matriz_Rot[1][0]=Dt*W[2];//z
        Matriz_Rot[1][1]=1;
        Matriz_Rot[1][2]=-Dt*W[0];
        Matriz_Rot[2][0]=-Dt*W[1];
        Matriz_Rot[2][1]=Dt*W[0];
        Matriz_Rot[2][2]=1;

        Mult_Matriz3x3( Matriz_DCM, Matriz_Rot);

        Normalizar();
        CorreccionDrift();


        pitch = -asin(Matriz_DCM[2][0]);
        roll = atan2(Matriz_DCM[2][1],Matriz_DCM[2][2]);
        yaw = atan2(Matriz_DCM[1][0],Matriz_DCM[0][0]);
        if (ON == 1) {
                if (_UDP != 0) {
                Udp.beginPacket(IPPc, PortPc);
```

```
                    Udp.print(ToDeg(pitch));Udp.print(";");
                    Udp.print(ToDeg(roll));Udp.print(";");
                    Udp.print(ToDeg(yaw));Udp.print(";");
                    Udp.endPacket();
                    }
                    if (_UDP != 1)  {
                    Serial.print(ToDeg(pitch)); Serial.print(";");
                    Serial.print(ToDeg(roll)); Serial.print(";");
                    Serial.print(ToDeg(yaw)); Serial.println(";1;");
                    }
          }
}

void GY80::Normalizar()
{

          float error = -0.5 * Producto_Escalar(&Matriz_DCM[0][0], &Matriz_DCM[1][0]);
          float x_est[3]; for(int c=0; c<3; c++) {x_est[c]=Matriz_DCM[1][c]* error;}
          float y_est[3]; for(int c=0; c<3; c++) {y_est[c]=Matriz_DCM[0][c]* error;}
          Vector_Suma(&Matriz_DCM[0][0], x_est);
          Matriz_DCM[0][0] = vectorOut[0]; Matriz_DCM[0][1] = vectorOut[1]; Matriz_DCM[0][2] =
vectorOut[2];
          Vector_Suma(&Matriz_DCM[1][0], y_est);
          Matriz_DCM[1][0] = vectorOut[0]; Matriz_DCM[1][1] = vectorOut[1]; Matriz_DCM[1][2] =
vectorOut[2];
          Producto_Vectorial( &Matriz_DCM[0][0], &Matriz_DCM[1][0]);
          Matriz_DCM[2][0] = vectorOut[0]; Matriz_DCM[2][1] = vectorOut[1]; Matriz_DCM[2][2] =
vectorOut[2];
          for(int c=0; c<3; c++) {
                    Matriz_DCM[0][c] = 0.5 * ( 3 - Producto_Escalar(&Matriz_DCM[0][0], &Matriz_DCM[0][0])) *
Matriz_DCM[0][c];
                    Matriz_DCM[1][c] = 0.5 * ( 3 - Producto_Escalar(&Matriz_DCM[1][0], &Matriz_DCM[1][0])) *
Matriz_DCM[1][c];
                    Matriz_DCM[2][c] = 0.5 * ( 3 - Producto_Escalar(&Matriz_DCM[2][0], &Matriz_DCM[2][0])) *
Matriz_DCM[2][c];
          }

}

void GY80::CorreccionDrift()
{

          float ErrorAccel = sqrt(ACC[0]*ACC[0]+ACC[1]*ACC[1]+ACC[2]*ACC[2])/9.81;
          ErrorAccel = constrain(1 - 2*abs(1 - ErrorAccel),0,1);
          Producto_Vectorial(&ACC[0], &Matriz_DCM[2][0]);
          for (int c=0; c<3; c++) {W_P[c]=vectorOut[c]*Kp_ROLLPITCH*ErrorAccel;}
          for (int c=0; c<3; c++) {W_I[c]=W_I[c]+vectorOut[c]*Ki_ROLLPITCH*ErrorAccel;}

          getYaw();

          magx = cos(yaw);
          magy = sin(yaw);
          float ErrorYaw=(Matriz_DCM[0][0]*magy) - (Matriz_DCM[1][0]*magx);
          float VecErrorYaw[3];
          for (int c=0; c<3; c++) {VecErrorYaw[c]=Matriz_DCM[2][c]*ErrorYaw;}
          for (int c=0; c<3; c++) {W_P[c]=W_P[c]+VecErrorYaw[c]*Kp_YAW;}
```

```
        for (int c=0; c<3; c++) {W_I[c]=W_I[c]+VecErrorYaw[c]*Ki_YAW;}
}

void GY80::Inic_Cuaternion(int ON)
{
        getSensor();

        getPitchRoll();getYaw();

        Q[0]=cos(roll/2)*cos(pitch/2)*cos(yaw/2)+sin(roll/2)*sin(pitch/2)*sin(yaw/2);
  Q[1]=sin(roll/2)*cos(pitch/2)*cos(yaw/2)-cos(roll/2)*sin(pitch/2)*sin(yaw/2);
  Q[2]=cos(roll/2)*sin(pitch/2)*cos(yaw/2)+sin(roll/2)*cos(pitch/2)*sin(yaw/2);
  Q[3]=cos(roll/2)*cos(pitch/2)*sin(yaw/2)-sin(roll/2)*sin(pitch/2)*cos(yaw/2);
        mag = sqrtf(Q[0]*Q[0]+Q[1]*Q[1]+Q[2]*Q[2]+Q[3]*Q[3]);
        for (int c=0; c<4 ; c++) { Q[c]=Q[c]/mag; }

        if (ON == 1) {
                if (_UDP != 0) {
                Udp.beginPacket(IPPc, PortPc);
                Udp.print(ToDeg(pitch));Udp.print(";");
                Udp.print(ToDeg(roll));Udp.print(";");
                Udp.print(ToDeg(yaw));Udp.print(";");
                Udp.endPacket();
                }
                if (_UDP != 1)  {
                Serial.print(ToDeg(pitch)); Serial.print(";");
                Serial.print(ToDeg(roll)); Serial.print(";");
                Serial.print(ToDeg(yaw)); Serial.println(";0;");
                }
        }
}

void GY80::Cuaternion(int ON)
{
        getSensor();

        //Calculo Dt entre cada actualizacion (en milisegundos);
        Dt = float(millis()- tiempo)/1000;
        tiempo = millis();

        float gx = ToRad(GIR[0]); float gy = ToRad(GIR[1]); float gz= ToRad(GIR[2]);
        Q_Giro[0] = 0.5f * (-Q[1] * gx - Q[2] * gy - Q[3] * gz);
        Q_Giro[1] = 0.5f * (Q[0] * gx + Q[2] * gz - Q[3] * gy);
        Q_Giro[2] = 0.5f * (Q[0] * gy - Q[1] * gz + Q[3] * gx);
        Q_Giro[3] = 0.5f * (Q[0] * gz + Q[1] * gy - Q[2] * gx);

        FiltroCuat();

        Q[0] += Q_Giro[0] * Dt;
        Q[1] += Q_Giro[1] * Dt;
        Q[2] += Q_Giro[2] * Dt;
        Q[3] += Q_Giro[3] * Dt;
        mag = sqrtf(Q[0]*Q[0]+Q[1]*Q[1]+Q[2]*Q[2]+Q[3]*Q[3]);
        for (int c=0; c<4 ; c++) { Q[c]=Q[c]/mag; }

        roll=atan2(2*Q[2]*Q[3]+2*Q[0]*Q[1],1-2*Q[1]*Q[1]-2*Q[2]*Q[2]);
  pitch=asin(-2*Q[1]*Q[3]+2*Q[0]*Q[2]);
```

```
  yaw=atan2(2*Q[1]*Q[2]+2*Q[0]*Q[3],1-2*(Q[2]*Q[2]+Q[3]*Q[3]));

        if (ON == 1) {
                if (_UDP != 0) {
                Udp.beginPacket(IPPc, PortPc);
                Udp.print(ToDeg(pitch));Udp.print(";");
                Udp.print(ToDeg(roll));Udp.print(";");
                Udp.print(ToDeg(yaw));Udp.print(";");
                Udp.endPacket();
                }
                if (_UDP != 1)  {
                Serial.print(ToDeg(pitch)); Serial.print(";");
                Serial.print(ToDeg(roll)); Serial.print(";");
                Serial.print(ToDeg(yaw)); Serial.println(";0;");
                }
        }
}

void GY80::FiltroCuat()
{
                //Normalizacion de los datos del acelerometro y del magnetometro.
                mag = sqrtf(ACC[0]*ACC[0]+ACC[1]*ACC[1]+ACC[2]*ACC[2]);
                for (int c=0; c<3 ; c++) { ACC2[c]=ACC[c]/mag; }
                mag = sqrtf(MAG[0]*MAG[0]+MAG[1]*MAG[1]+MAG[2]*MAG[2]);
                for (int c=0; c<3 ; c++) { MAG[c]=MAG[c]/mag; }

                //Operaciones previas.
                _2q0mx = 2.0f * Q[0] * MAG[0];
                _2q0my = 2.0f * Q[0] * MAG[1];
                _2q0mz = 2.0f * Q[0] * MAG[2];
                _2q1mx = 2.0f * Q[1] * MAG[0];
                _2q0 = 2.0f * Q[0];
                _2q1 = 2.0f * Q[1];
                _2q2 = 2.0f * Q[2];
                _2q3 = 2.0f * Q[3];
                _2q0q2 = 2.0f * Q[0] * Q[2];
                _2q2q3 = 2.0f * Q[2] * Q[3];
                q0q0 = Q[0]*Q[0];
                q0q1 = Q[0]*Q[1];
                q0q2 = Q[0]*Q[2];
                q0q3 = Q[0]*Q[3];
                q1q1 = Q[1]*Q[1];
                q1q2 = Q[1]*Q[2];
                q1q3 = Q[1]*Q[3];
                q2q2 = Q[2]*Q[2];
                q2q3 = Q[2]*Q[3];
                q3q3 = Q[3]*Q[3];

                //Referencia campo magnetico mediante magnetometros
                magx = MAG[0] * q0q0 - _2q0my * Q[3] + _2q0mz * Q[2] + MAG[0] * q1q1 + _2q1 * MAG[1] *
Q[2] + _2q1 * MAG[2] * Q[3] - MAG[0] * q2q2 - MAG[0] * q3q3;
                magy = _2q0mx * Q[3] + MAG[1] * q0q0 - _2q0mz * Q[1] + _2q1mx * Q[2] - MAG[1] * q1q1 +
MAG[1] * q2q2 + _2q2 * MAG[2] * Q[3] - MAG[1] * q3q3;
                _2bx = sqrt(magx * magx + magy * magy);
                _2bz = -_2q0mx * Q[2] + _2q0my * Q[1] + MAG[2] * q0q0 + _2q1mx * Q[3] - MAG[2] * q1q1 +
_2q2 * MAG[1] * Q[3] - MAG[2] * q2q2 + MAG[2] * q3q3;
```

```
                    _4bx = 2.0f * _2bx;
                    _4bz = 2.0f * _2bz;

                    // Calculo del cuaternion para el filtro
                    Q_Fil[0] = -_2q2 * (2.0f * q1q3 - _2q0q2 - ACC2[0]) + _2q1 * (2.0f * q0q1 + _2q2q3 - ACC2[1]) - _2bz * Q[2] * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - MAG[0]) + (-_2bx * Q[3] + _2bz * Q[1]) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - MAG[1]) + _2bx * Q[2] * (_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - MAG[2]);
                    Q_Fil[1] = _2q3 * (2.0f * q1q3 - _2q0q2 - ACC2[0]) + _2q0 * (2.0f * q0q1 + _2q2q3 - ACC2[1]) - 4.0f * Q[1] * (1 - 2.0f * q1q1 - 2.0f * q2q2 - ACC2[2]) + _2bz * Q[3] * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - MAG[0]) + (_2bx * Q[2] + _2bz * Q[0]) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - MAG[1]) + (_2bx * Q[3] - _4bz * Q[1]) * (_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - MAG[2]);
                    Q_Fil[2] = -_2q0 * (2.0f * q1q3 - _2q0q2 - ACC2[0]) + _2q3 * (2.0f * q0q1 + _2q2q3 - ACC2[1]) - 4.0f * Q[2] * (1 - 2.0f * q1q1 - 2.0f * q2q2 - ACC2[2]) + (-_4bx * Q[2] - _2bz * Q[0]) * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - MAG[0]) + (_2bx * Q[1] + _2bz * Q[3]) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - MAG[1]) + (_2bx * Q[0] - _4bz * Q[2]) * (_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - MAG[2]);
                    Q_Fil[3] = _2q1 * (2.0f * q1q3 - _2q0q2 - ACC2[0]) + _2q2 * (2.0f * q0q1 + _2q2q3 - ACC2[1]) + (-_4bx * Q[3] + _2bz * Q[1]) * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - MAG[0]) + (-_2bx * Q[0] + _2bz * Q[2]) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - MAG[1]) + _2bx * Q[1] * (_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - MAG[2]);
                    mag = sqrtf(Q_Fil[0]*Q_Fil[0]+Q_Fil[1]*Q_Fil[1]+Q_Fil[2]*Q_Fil[2]+Q_Fil[3]*Q_Fil[3]);
                    for (int c=0; c<4 ; c++) { Q_Fil[c]=Q_Fil[c]/mag; }

                    // Aplicacion del filtro al cuaternion de giros
                    for (int c=0; c<4 ; c++) { Q_Giro[c] = 0.98f * Q_Giro[c] - 0.02f * Q_Fil[c];}
}

void GY80::Vector_Suma(float v1[3], float v2[3])
{
  for(int c=0; c<3; c++)
  {
    vectorOut[c]=v1[c]+v2[c];
  }
}

float GY80::Producto_Escalar(float v1[3], float v2[3])
{
        float resultado = 0;
        for(int c=0; c<3; c++)
          {
                resultado += v1[c]* v2[c];
          }
        return resultado;
}

void GY80::Producto_Vectorial(float v1[3], float v2[3])
{
  vectorOut[0]= (v1[1]*v2[2]) - (v1[2]*v2[1]);
  vectorOut[1]= (v1[2]*v2[0]) - (v1[0]*v2[2]);
  vectorOut[2]= (v1[0]*v2[1]) - (v1[1]*v2[0]);
}

void GY80::Mult_Matriz3x3(float Mat1[3][3], float Mat2[3][3])
{
        float Temp[3];
        for(int x=0; x<3; x++)
        {
```

```
                    for(int y=0; y<3; y++)
                    {
                            for(int w=0; w<3; w++)
                            {
                            Temp[w]=Mat1[x][w]*Mat2[w][y];
                            }
                    Matriz_DCM[x][y]=Temp[0]+Temp[1]+Temp[2];
                    }
            }
}

void GY80::CalcAcc(void) {
            acct[0]=-GRAVEDAD*sin(pitch);
            acct[1]=GRAVEDAD*sin(roll);
            acct[2]=GRAVEDAD*cos(pitch)*cos(roll);
            for (int n=0; n<3; n++) {acc[n]=ACC[n]-acct[n];}
            //Serial.print(acc[0],5);Serial.print(";");Serial.print(acc[1],5);Serial.print(";");Serial.print(acc[2],5);Serial
.println(";");
            for (int n=0; n<3; n++) {vel[n]=vel[n]+acc[n]*Dt;}
            //Serial.print("Velocidades:
");Serial.print(vel[0],4);Serial.print(";");Serial.print(vel[1],4);Serial.print(";");Serial.print(vel[2],4);Serial.println(";
");
            for (int n=0; n<3; n++) {pos[n]=pos[n]+vel[n]*Dt+0.5f*acc[n]*Dt*Dt;}
            //Serial.print("Movimiento:
");Serial.print(pos[0],4);Serial.print(";");Serial.print(pos[1],4);Serial.print(";");Serial.print(pos[2],4);Serial.println(
";");
}

void GY80::writeTo(byte DireccionSensor, byte address, byte val) {
            Wire.beginTransmission(DireccionSensor);
            Wire.write(address);
            Wire.write(val);
            Wire.endTransmission();
}

int GY80::readReg8(int deviceAddress, byte address){

   int v;
   Wire.beginTransmission(deviceAddress);
   Wire.write(address);
   Wire.endTransmission();

   Wire.requestFrom(deviceAddress, 1);

   v = Wire.read();
   return v;
}

int GY80::readReg16(int deviceAddress, byte address){
            int a;
            int b;
            int v;
            Wire.beginTransmission(deviceAddress);
            Wire.write(address);
            Wire.endTransmission();

            Wire.requestFrom(deviceAddress, 2);
```

```
	a = Wire.read();
	b = Wire.read();
	v= (( b << 8) | a);

	return v;
}

int GY80::readRegInv16(int deviceAddress, byte address){
	int a;
	int b;
	int v;
	Wire.beginTransmission(deviceAddress);
	Wire.write(address);
	Wire.endTransmission();

	Wire.requestFrom(deviceAddress, 2);
	a = Wire.read();
	b = Wire.read();
	v= (( a << 8) | b);

	return v;
}

float GY80::ToDeg(float rad) {
	float deg;
	deg = 57.295 * rad;
	return deg;
}

float GY80::ToRad(float deg) {
	float rad;
	rad = deg / 57.295;
	return rad;
}
```

A.1.2 Calibración sensores GY-80

Programa de Arduino para realizar los pasos para poder calibrar el acelerómetro, el giróscopo y el magnetómetro, tiene 3 funciones, una para cada sensor, que deberán ser llamadas para calibrar el sensor deseado.

"Calibracion.ino"

```
/*
Calibracion.ino - Proyecto fin de Master Universidad de Cantabria, "Fusión sensorial para a estimación del estado de un vehículo autónomo".
Realizado por Daniel López Montes y dirigido por Luciano Alonso Renteria. 2016.
*/

#include <Wire.h>
#include <GY80.h>
float AccelMinX=0;
float AccelMinY=0;
float AccelMinZ=0;
```

```
float AccelMaxX=0;
float AccelMaxY=0;
float AccelMaxZ=0;
float OffsetGir[3];

GY80 IMU=GY80(0,0);

void setup() {
        Serial.begin(19200);
        IMU.Inicializar(0);
        //CalibrarMagnetometro();
        GananciaAcc();

  OffsetGir[0] = 1.66;
  OffsetGir[1] = 0.65;
  OffsetGir[2] = -0.54;
}


void loop() {
   OffsetGiro();
//        IMU.getSensor();
//
Serial.print(IMU.ACC[0],3);Serial.print(";");Serial.print(IMU.ACC[1],3);Serial.print(";");Serial.print(IMU.ACC[2],3)
;Serial.println(";");
//
Serial.print(IMU.GIR[0],3);Serial.print(";");Serial.print(IMU.GIR[1],3);Serial.print(";");Serial.print(IMU.GIR[2],3);S
erial.println(";");
//  delay(150);
}
void GananciaAcc(){
        //Funcion para medir medidas maximas de los acelerometros  con la IMU en posición estatica. Debe
ser llamada en la funcion Setup.
        Serial.println("Calibracion de los acelerometros:");
        Serial.println("Para salir introducir '2', para realizar una medida introducir '1'");
        int OK = 0;
        while (OK != 50) {
                OK = 0;
                if (Serial.available()){
                        while(Serial.available()){
                                OK = (Serial.read());
                                if (OK == 49) {
                                        for (int c=0; c<10; c++) {
                                                IMU.getAcce();
                                                if (IMU.ACC[0] < AccelMinX) AccelMinX = IMU.ACC[0];
                                                if (IMU.ACC[0] > AccelMaxX) AccelMaxX = IMU.ACC[0];

                                                if (IMU.ACC[1] < AccelMinY) AccelMinY = IMU.ACC[1];
                                                if (IMU.ACC[1] > AccelMaxY) AccelMaxY = IMU.ACC[1];

                                                if (IMU.ACC[2] < AccelMinZ) AccelMinZ = IMU.ACC[2];
                                                if (IMU.ACC[2] > AccelMaxZ) AccelMaxZ = IMU.ACC[2];
                                        }
                                        Serial.print("Aceleraciones minimas (X,Y,Z): ");
Serial.print(AccelMinX,3);Serial.print("                                \t");Serial.print(AccelMinY,3);Serial.print("
\t");Serial.print(AccelMinZ,3);Serial.println("; \t");
```

```
                                        Serial.print("Aceleraciones maximas (X,Y,Z): ");
Serial.print(AccelMaxX,3);Serial.print("; \t");Serial.print(AccelMaxY,3);Serial.print("; \t");Serial.print(AccelMaxZ,3);Serial.println("; \t");
                                        Serial.println("Para salir introducir '2', para realizar una medida introducir '1'");
                                    }
                                }
                            }
                        }
}

void OffsetGiro() {
        //Funcion que recoge datos de los giroscopos con la imu en posicion estatica y hace la media para calcular el offset.
        //Esta funcion debe se llamada desde el Loop
        Serial.println("Calculo Offset Giroscopos:");
        Serial.println("Mantener totalmente quieta la IMU. Escribir '1' cuando este preparado.");
        int OK = 0;
        while ( OK != 49) {
                if (Serial.available()){
                        while(Serial.available()){
                        OK = (Serial.read());
                        }
                }
}
        int MuestrasOffsetGiro=2000;
        float OffsetGiroProv[3];
  OffsetGiroProv[0]= 0; OffsetGiroProv[1]= 0;OffsetGiroProv[2]= 0;
        float OffsetGiro[3];
        for (int c=0; c<MuestrasOffsetGiro; c++){
                IMU.getGiro();
                for (int h=0; h<3; h++){
                        OffsetGiroProv[h] += (IMU.GIR[h]+OffsetGir[h]);
                }
        }
        for (int h=0; h<3; h++){
                OffsetGiro[h] = OffsetGiroProv[h] / MuestrasOffsetGiro;
        }
        Serial.print("Offset Eje X: ");Serial.print(OffsetGiro[0],4); Serial.print("\t Offset Eje Y: ");Serial.print(OffsetGiro[1],4); Serial.print("\t Offset Eje Z: ");Serial.println(OffsetGiro[2],4);
}

void CalibrarMagnetometro() {
        //Funcion para obtener datos de los magnetometros y realizar la calibracion posteriormente mediante Matlab.
        Serial.println("Calibracion de los magnetometros:");
        Serial.println("Para salir introducir '2'");
        int OK = 0;
        while (OK != 50) {
                OK = 0;
                if (Serial.available()){
                        while(Serial.available()){
                                OK = (Serial.read());
                        }
                }
   IMU.getMagn();
```

```
   Serial.println(IMU.MAG[0],3);
   Serial.println(IMU.MAG[1],3);
   Serial.println(IMU.MAG[2],3);
   delay(100);
          }
}

void MostrarDatosGiro() {
 Serial.print(IMU.GIR[0],2);Serial.print("; \t"); Serial.print(IMU.GIR[0]+OffsetGir[0],2);Serial.print("; \t");
 Serial.print(IMU.GIR[1],2);Serial.print("; \t");Serial.print(IMU.GIR[1]+OffsetGir[1],2);Serial.print("; \t");
 Serial.print(IMU.GIR[2],2);Serial.print("; \t");Serial.print(IMU.GIR[2]+OffsetGir[2],2);Serial.println();
}
```

A.1.3 Librería GPSSIM808

Para la realización todas las operaciones relativas a la utilización del GPS, se ha creado la librería GPSSIM808, la cual servirá para la obtención de los datos útiles para este proyecto.

Para utilizar esta librería hay que utilizar la siguiente línea en el archivo .ino de arduino:

GPSSIM808(int UDP, int MODO)

UDP es el parámetro que hay que utilizar para elegir la forma de salida de datos, siendo "0" si se utiliza la comunicación USB, o "1" si se utiliza la comunicación UDP. Si en cambio el parámetro UDP no es ninguno de los anteriores, la salida de datos será tanto por USB como por Ethernet, esto será útil para la realización de pruebas.

MODO en cambio es otro parámetro que se utiliza elegir si se utiliza el filtro Kalman o no. En el caso de que sea "0" no se utilizara el filtro Kalman, y los datos de salida serán directamente los que da el GPS. Si MODO es "1" se obtendrán únicamente lo datos de la salida del filtro Kalman, y en el caso de que sea otro número se obtendrán los datos tanto sin filtro como con filtro, diferenciados ya que la cadena de caracteres si se utiliza el filtro finalizan en "1;", mientras que la cadena de caracteres sin filtro termina con "0;".

Existen 2 funciones principales, la primera de ellas para inicializar el GPS, y que deberá ser llamada en la función Setup del programa de Arduino en la que se vaya a utilizar.

Esta función realizará y comprobará el éxito en su realización de encender el GPS, resetear el GPS y buscar satélites para conectar, las primeras 2 tareas serán rápidamente realizadas, pero la última puede tomar un tiempo considerable, dependiendo de las condiciones en las que se está utilizando el GPS, el resto de funciones no se llevarán a cabo hasta que el

GPS haya conseguido iniciar una conexión con los satélites. Es por este motivo que la función de inicialización del GPS deberá ser la primera a llamar en el programa de Arduino.

Una vez se consigue la fijación de los satélites el programa saldrá de la función de inicialización y podrá iniciar la segunda función principal de la librería. Esta función es la de salida de datos del GPS, la cual recogerá datos cada un cierto tiempo y comunicará por el puerto escogido (Ethernet o USB).

Ya que la obtención de la información del GPS es bastante lenta comparada con por ejemplo la obtención de datos de los sensores de la IMU, no se deben utilizar delays, y se utilizan temporizadores, de modo que las diferentes fases para la obtención de una línea de datos del GPS solo se llevarán a cabo cuando el temporizador lo permita.

Esto provocará que en el caso de utilizar la salida de datos de GPS y de la IMU al mismo tiempo, se envíen con mayor frecuencia los de la IMU que los del GPS.

Los datos recogidos por la librería son los siguientes:

-Latitud (En grados).

-Longitud (En grados).

-Velocidad (En Km/h).

-Dirección (En grados, de 0 a 360).

Para ello se utiliza el comando AT+CGPSINF=32, que es el único disponible que proporciona todos estos datos, y tras alguna transformación de la unidades se obtienen los valores de los anteriores parámetros.

Debido a que la salida de datos del GPS es mediante una cadena de caracteres, en ciertas ocasiones esta cadena no es tal y como se espera, por lo tanto los valores leídos por Arduino podrían ser erróneos, en este caso la librería devolvería un línea de error en el caso de que se utilice la salida de datos por USB.

Posteriormente los resultados recogidos se almacenan en sus respectivas variables como valores numéricos, y pasados al filtro Kalman, el filtro realiza las operaciones matriciales necesarias (para las cuales se han creado también varias funciones utilizadas para la multiplicación, suma y resta de matrices).

Una vez realizadas las operaciones del filtro, se almacenan las variables y se envían utilizando el método de comunicación elegido anteriormente.

Funciones públicas:

- **Inicializar()** Es la utilizada en el "setup" para inicializar el GPS, encendiéndolo, reseteándolo y buscando satélites.
- **GPSOUT()** Debe ser utilizada en el "loop" y es la que recibe los datos del GPS en cada iteración y se los pasa al fitlro Kalman si fuera necesario.

Variables públicas:

- **Lat, Long, Vel, Dir, VelNorte, VelOeste** Variables utilizadas para almacenar la información obtenida del GPS.
- **LatK, LongK, VelNorteK, VelOesteK** Variables que almacenan la información obtenida del filtro Kalman.

"GPSSIM808.h"

```
/*
GPSSIM808.cpp - Librería para el GPS con la sim808. Realizada por Daniel López Montes.
*/

//LIBRERIAS
#include "Arduino.h"
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>

class GPSSIM808
{
public: GPSSIM808(int UDP, int MODO); // CONSTRUCTOR
        void    Inicializar(void);
        void    GPSOUT(void);
        float   Lat, Long, Vel, Dir, VelNorte, VelOeste;
        float   LatK, LongK, VelNorteK, VelOesteK;
private:
        int     _UDP, _MODO;
        int     _PWRON = 0, _RESETOK[3], _3DFIX = 0, _NUEVODATO[2], _DATOOK = 0;
        int     a, b, c, d, e, f, dato;
        int     cont = 0;
        char    Char0 = {'0'}, Char1 = {'1'}, Char3 = {'3'}, Char4 = {'4'}, CharD = {'D'};
        char    CharO = {'O'}, CharK = {'K'}, CharPunto = {'.'};
        char    A[1], B[1], C[1], D[1], E[1], F[1];
        int             Latitud[10], Longitud[10], Velocidad[10], Direccion[10];
        void    PWRON(void), RESETHOT(void), RESETWARM(), COORDENADAS(),ALMACENAR(char read),
KALMAN();
        long    tiempo;
        int     nLat, nLong, nVel, nDir;
        int     CONT=-1;
```

```
        int     PortPc = 8000;

        //Filtro KALMAN
        long    tiempo2, tiempo3;
        float   Dt;
        float   pi = 3.1415926;
        float   GPS_data1[3], prevGPS_data[3], GPS_data[3] = {0,0,0};
        int             N=1;
        float   cosLat;
        float   EstadoX[4][1] = {{0},{0},{0},{0}}, EstimacionProxEstadoX[4][1] = {{0},{0},{0},{0}};
        float   MatrizATrasp[4][4] = {{0, 0, 0, 0},{0, 0, 0, 0},{0, 0, 0, 0},{0, 0, 0, 0}};
        float   CovarianzaErrorP[4][4] = {{0.001, 0, 0, 0},{0, 0.001, 0, 0},{0, 0, 0.02, 0},{0, 0, 0, 0.02}};
        float   EstimacionCovarianzaErrorP[4][4] = {{0, 0, 0, 0},{0, 0, 0, 0},{0, 0, 0, 0},{0, 0, 0, 0}};
        float   MatrizIntermedia1[4][4] = {{0, 0, 0, 0},{0, 0, 0, 0},{0, 0, 0, 0},{0, 0, 0, 0},};
        float   MatrizIntermedia2[4][4] = {{0, 0, 0, 0},{0, 0, 0, 0},{0, 0, 0, 0},{0, 0, 0, 0},};
        float   MatrizIntermedia3[2][4] = {{0, 0, 0, 0},{0, 0, 0, 0}}, MatrizIntermedia4[2][2] = {{0, 0},{0, 0}};
        float   MatrizIntermedia5[2][2] = {{0, 0},{0, 0}},MatrizIntermedia6[4][2] = {{0, 0},{0, 0},{0, 0},{0, 0}};
        float   MatrizIntermedia7[2][1] = {{0},{0}}, MatrizIntermedia8[2][1] = {{0},{0}};
        float   MatrizIntermedia9[4][1] = {{0},{0},{0},{0}};
        float   MatrizI[4][4] = {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
        float   MatrizH[2][4] = {{1,0,0,0},{0,1,0,0}}, MatrizHT[4][2] = {{1, 0},{0, 1},{0, 0},{0, 0},};
        float   GananciaKalman[4][2] = {{0, 0},{0, 0},{0, 0},{0, 0}};
        void    Mult(float* A, float* B, int m, int p, int n, float* C);
        void    Trasp(float* A, int m, int n, float* C);
        void    Sum(float* A, float* B, int m, int n, float* C);
        void    Rest(float* A, float* B, int m, int n, float* C);
        void    Print(float* A, int m, int n, String label);

};
```

"GPSSIM808.cpp"

```
/*
GPSSIM808.cpp - Librería para el GPS de la SIM808. Realizada por Daniel López Montes.
*/


#include "Arduino.h"
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
#include "GPSSIM808.h"
EthernetUDP Udp2;
IPAddress IPPc2(192, 168, 1, 2);

GPSSIM808::GPSSIM808(int UDP, int MODO)
{
        _UDP = UDP;
        _MODO = MODO;
}

void GPSSIM808::Inicializar()
{
        if (_UDP!=1 ) {
                Serial.begin(19200);
                Serial.println("INICIALIZANDO");
```

```
        }
        Serial3.begin(19200);
        delay(500);

        while (_PWRON == 0) {
        a=0;
        Serial3.write("AT+CGPSPWR?\n");delay(100);
         if (Serial3.available()) { a=0;  while (Serial3.available()){ A[a] = (Serial3.read()); a++; }} for (a=23; a<26;
a++) {if ((A[a] == Char1) and (A[a-1] == ' ')) {_PWRON=1;break;} else { delay(200);PWRON();}}}
        if (_UDP!=1) {Serial.println("MODULO GPS ON");}
        _RESETOK[0]=0;_RESETOK[1]=0;_RESETOK[2]=0;
        while (_RESETOK[0] == 0) {
                c=0; delay(200);
                Serial3.write("AT+CGPSRST=0\n"); delay(100);
                if (Serial3.available()) { c=0;  while (Serial3.available()){ C[c] = (Serial3.read()); c++; }} for
(c=13; c<16; c++) {if ((C[c] == CharO) and (C[c+1] == CharK) ) { _RESETOK[0]=1;break;}}}
        if (_UDP!=1) {Serial.println("COLD RESET MODULO GPS OK");}
        delay(2000);
        if (_UDP!=1) {Serial.println("INICIANDO FIJACION DE SATELITES");}
        //_3DFIX = 1; //SIN FIJACION DE SATELITES!!!!!
        tiempo=millis();
        while (_3DFIX == 0) {
                d=0; delay(2000); cont++;
                if (cont == 10) {RESETHOT();} if (cont == 35) {RESETWARM();}
                Serial3.write("AT+CGPSSTATUS?\n"); delay(100);
                if (Serial3.available()) { d=0; while (Serial3.available()) { D[d] = (Serial3.read()); d++; }} for (int
p=(d-15); p<(d-12); p++) {if ((D[p] == Char3) and (D[p+1] == CharD)) {_3DFIX=1;}} if
(_UDP!=1){Serial.print(".");}}
        if (_UDP!=1) {Serial.println();Serial.println("LOCALIZACION 3D FIJADA");}

        if (_UDP!=0){
                //Inicializar comunicacion UDP
                byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
                IPAddress IPArduino(192, 168, 1, 177);
                unsigned int PortArduino = 4000;
                Ethernet.begin(mac,IPArduino);
                Udp2.begin(PortArduino);
                delay(250);
        }

        tiempo=millis(); tiempo3 = millis();
        _NUEVODATO[0]=1;_NUEVODATO[1]=1;
        for (int n=0; n<10; n++) {Latitud[n] = 0;}Serial.println();
        for (int n=0; n<10; n++) {Longitud[n] = 0;}Serial.println();
        for (int n=0; n<10; n++) {Velocidad[n] = 0;}Serial.println();
        for (int n=0; n<10; n++) {Direccion[n] = 0;}Serial.println();
        }


void GPSSIM808::GPSOUT()
{
        if (_NUEVODATO[0] == 1) {Serial3.write("AT+CGPSINF=32\n"); _NUEVODATO[0] = 0;tiempo=millis();}
        if ((millis() >= (tiempo+100)) and (_NUEVODATO[1] == 1)) {if (Serial3.available()) {dato=0; while
(Serial3.available()){  ALMACENAR(Serial3.read());dato++;}}  _NUEVODATO[1]=0;COORDENADAS();tiempo =
millis();}
        if ((millis()) >= (tiempo+175)) {tiempo=millis(); _NUEVODATO[0] = 1; _NUEVODATO[1] = 1;}
```

```
}

void GPSSIM808::ALMACENAR(char read)
{
        //Serial.write(read);
        if (read == ',') {CONT++;}
        if ((CONT == 2) and (read != '.') and (read != ',')) {Latitud[nLat-1] = read - 48;}    if (CONT == 2) {nLat++;}
        if ((CONT == 4) and (read != '.') and (read != ',')) {Longitud[nLong-1] = read - 48;}          if (CONT == 4)
{nLong++;}
        if ((CONT == 6) and (read != '.') and (read != ',')) {Velocidad[nVel-1] = read - 48;}          if (CONT == 6)
{nVel++;}
        if ((CONT == 7) and (read != '.') and (read != ',')) {Direccion[nDir-1] = read - 48;} if (CONT == 7) {nDir++;}

}

void GPSSIM808::COORDENADAS()
{
        //Serial.print(nLat);Serial.print(";");Serial.print(nLong);Serial.print(";");Serial.print(nVel);Serial.print(";"
);Serial.print(nDir);Serial.println(";");
        // for (int n=0; n<10; n++) {Serial.print(Latitud[n]);}Serial.println();
        // for (int n=0; n<10; n++) {Serial.print(Longitud[n]);}Serial.println();
        // for (int n=0; n<10; n++) {Serial.print(Velocidad[n]);}Serial.println();
        // for (int n=0; n<10; n++) {Serial.print(Direccion[n]);}Serial.println();
        _DATOOK = 1;
        for (int n=0; n < 10; n++) {if ((Latitud[n] < 0) or (Latitud[n] > 9) or (Longitud[n] < 0) or (Longitud[n] > 9)
or (Velocidad[n] < 0) or (Velocidad[n] > 9) or (Direccion[n] < 0) or (Direccion[n] > 9)) {_DATOOK=0;}}
        if (_DATOOK == 1) {

        Lat=Latitud[0]*10+Latitud[1]+Latitud[2]/6.0f+Latitud[3]/60.0f+Latitud[5]/600.0f+Latitud[6]/6000.0f+L
atitud[7]/60000.0f+Latitud[8]/600000.0f;

        Long=Longitud[0]*100+Longitud[1]*10+Longitud[2]+Longitud[3]/6.0f+Longitud[4]/60.0f+Longitud[6]/
600.0f+Longitud[7]/6000.0f+Longitud[8]/60000.0f+Longitud[9]/600000.0f;
                if (nVel == 6) {Vel = (Velocidad[0]+0.1f*
Velocidad[2]+0.01f*Velocidad[3]+0.001f*Velocidad[4])*0.5144f;}
                else if (nVel == 7) {Vel = (10*Velocidad[0]+Velocidad[1]+0.1f*
Velocidad[3]+0.01f*Velocidad[4]+0.001f*Velocidad[5])*0.5144f;}
                if (nDir == 5) {Dir=Direccion[0]+0.1f*Direccion[2]+0.01f*Direccion[3];}
                else if (nDir == 6) {Dir=10*Direccion[0]+Direccion[1]+0.1f*Direccion[3]+0.01f*Direccion[4];}
                else if (nDir == 7)
{Dir=100*Direccion[0]+10*Direccion[1]+Direccion[2]+0.1f*Direccion[4]+0.01f*Direccion[5];}
                if (Dir >= 360) {Dir = Dir - 360;}
                VelNorte = Vel * cos(Dir * pi /180); VelOeste = -Vel * sin(Dir * pi / 180);
                if (_MODO !=0) {KALMAN();}
                if (_MODO != 1) {
                        if (_UDP != 0) {
                                Udp2.beginPacket(IPPc2, PortPc);
                                Udp2.print(Lat);Udp2.print(";");
                                Udp2.print(Long);Udp2.print(";");
                                Udp2.print(VelNorte);Udp2.print(";");
                                Udp2.print(VelOeste);Udp2.print(";");
                                Udp2.endPacket();
                        }
                        if (_UDP !=1) {
                                Serial.print(Lat,6); Serial.print(";");
```

```
                                        Serial.print(Long,6); Serial.print(";");
                                        Serial.print(VelNorte,3);Serial.print(";");
                                        Serial.print(VelOeste,3); Serial.println(";0;");
                                }
                        }
                }
                else if (_UDP != 1) {Serial.println("DATOS NO OK");}
                CONT=-1; nLat=0; nLong=0; nVel=0; nDir=0;
}

void GPSSIM808::KALMAN()
{
                tiempo2 = millis();
                prevGPS_data[0] = GPS_data[0];
                prevGPS_data[1] = GPS_data[1];
                prevGPS_data[2] = GPS_data[2];
                GPS_data[0] = Lat;
                GPS_data[1] = Long;
                GPS_data[2] = tiempo2;

                if (N==1) {
                        GPS_data1[0] = GPS_data[0];
                        GPS_data1[1] = GPS_data[1];
                        GPS_data1[2] = GPS_data[2];
                         cosLat = cos(GPS_data1[0]*pi/180);
                        prevGPS_data[2] = tiempo3;
                        N++;
                }

                Dt = ((float) (GPS_data[2] - prevGPS_data[2]))/ 1000;
                float MatrizA[4][4] = { {1, 0, Dt, 0},{0, 1,  0, Dt},{0, 0, 1, 0},{0, 0, 0, 1}};
                float MatrizCovarianzaQ[4][4] = {{0.01, 0, 0, 0},{0, 0.01, 0, 0},{0, 0, 0.001, 0},{0, 0, 0, 0.001}};
                float MatrizCovarianzaR[2][2] = {{2.21017383364137, 3.51637078682249},{3.51637078682249,
13.8032720993553}};


                Mult((float*) MatrizA, (float*) EstadoX, 4, 4, 1, (float*) EstimacionProxEstadoX);
                Trasp((float*) MatrizA, 4 ,4, (float *) MatrizATrasp);
                Mult((float*) MatrizA, (float*) CovarianzaErrorP, 4,4,4, (float*) MatrizIntermedia1);
                Mult((float*) MatrizIntermedia1, (float*) MatrizATrasp, 4,4,4, (float*) MatrizIntermedia2);
                Sum((float*) MatrizIntermedia2, (float*) MatrizCovarianzaQ, 4, 4, (float*)
EstimacionCovarianzaErrorP);

                Mult((float*) MatrizH, (float*) EstimacionCovarianzaErrorP, 2,4,4, (float*) MatrizIntermedia3);
                Mult((float*) MatrizIntermedia3, (float*) MatrizHT, 2,4,2, (float*) MatrizIntermedia4);
                Sum((float*) MatrizIntermedia4, (float*) MatrizCovarianzaR, 2, 2, (float*) MatrizIntermedia5);

                Mult((float*) EstimacionCovarianzaErrorP, (float*) MatrizHT, 4,4,2, (float*) MatrizIntermedia6);
                Mult((float*) MatrizIntermedia6, (float*) MatrizIntermedia5, 4,2,2, (float*) GananciaKalman);

                float ZkT[2][1] = {{(float) (GPS_data[0]-GPS_data1[0])*111100},{(GPS_data[1]-
GPS_data1[1])*111100*cosLat}};

                Mult((float*) MatrizH, (float*) EstimacionProxEstadoX, 2,4,1, (float*) MatrizIntermedia7);
                Rest((float*) ZkT, (float*) MatrizIntermedia7, 2, 1, (float*) MatrizIntermedia8);
                Mult((float*) GananciaKalman, (float*) MatrizIntermedia8, 4,2,1, (float*) MatrizIntermedia9);
```

```
        Sum((float*) EstimacionProxEstadoX, (float*) MatrizIntermedia9, 4,1, (float*) EstadoX);
        Mult((float*) GananciaKalman, (float*) MatrizH, 4,2,4, (float*) MatrizIntermedia1);
        Rest((float*) MatrizI, (float*) MatrizIntermedia1, 4,4, (float*) MatrizIntermedia6);
        Mult((float*) MatrizIntermedia6, (float*) EstimacionCovarianzaErrorP, 4,4,4, (float*) CovarianzaErrorP);

        LatK = GPS_data1[0] + (EstadoX[0][0] / 111100); LongK = GPS_data1[1] + (EstadoX[1][0] / (111100 * cosLat));
        VelNorteK = EstadoX[2][0]; VelOesteK = EstadoX[3][0];

        if (_UDP != 0) {
                Udp2.beginPacket(IPPc2, PortPc);
                Udp2.print(LatK);Udp2.print(";");
                Udp2.print(LongK);Udp2.print(";");
                Udp2.print(VelNorteK);Udp2.print(";");
                Udp2.print(VelOesteK);Udp2.print(";");
                Udp2.endPacket();
        }
        if (_UDP != 1)  {
                Serial.print(LatK,6); Serial.print(";");
                Serial.print(LongK,6); Serial.print(";");
                Serial.print(VelNorteK,3);Serial.print(";");
                Serial.print(VelOesteK,3); Serial.println(";0;");
        }

}

void GPSSIM808::Mult(float* A, float* B, int m, int p, int n, float* C)
{
  // A = (m x p); B = (p x n); C = A*B (m x n)
  int i, j, k;
  for (i=0;i<m;i++)
    for(j=0;j<n;j++)
    {
      C[n*i+j]=0;
      for (k=0;k<p;k++)
        C[n*i+j]= C[n*i+j]+A[p*i+k]*B[n*k+j];
    }
}

void GPSSIM808::Trasp(float* A, int m, int n, float* C)
{
  // A =(m x n); C = A' (n x m)
  int i, j;
  for (i=0;i<m;i++)
    for(j=0;j<n;j++)
      C[m*j+i]=A[n*i+j];
}

void GPSSIM808::Sum(float* A, float* B, int m, int n, float* C)
{
  // A = (m x n); B = (m x n); C = A+B (m x n)
  int i, j;
  for (i=0;i<m;i++)
    for(j=0;j<n;j++)
      C[n*i+j]=A[n*i+j]+B[n*i+j];
```

```
}

void GPSSIM808::Rest(float* A, float* B, int m, int n, float* C)
{
  // A = (m x n); B = (m x n); C = A-B (m x n)
  int i, j;
  for (i=0;i<m;i++)
    for(j=0;j<n;j++)
      C[n*i+j]=A[n*i+j]-B[n*i+j];
}

void GPSSIM808::Print(float* A, int m, int n, String label){
  // A = input matrix (m x n)
  int i,j;
  Serial.println();
  Serial.println(label);
  for (i=0; i<m; i++){
    for (j=0;j<n;j++){
      Serial.print(A[n*i+j],4);
      Serial.print("\t");
    }
    Serial.println();
  }
}

void GPSSIM808::PWRON()
{
        Serial3.write("AT+CGPSPWR=1\n");delay(100);
        if (Serial3.available()) { b=0;  while (Serial3.available()){ B[b] = (Serial3.read()); b++; }} if (B[b-3] ==
Chark ) {_PWRON=1;}
        //Serial.println("PWRON");
        delay(200);
}

void GPSSIM808::RESETHOT()
{
        while (_RESETOK[1]==0){
        Serial3.write("AT+CGPSRST=1\n");delay(100);
        if (Serial3.available()) { e=0;  while (Serial3.available()){ E[e] = (Serial3.read()); e++; }} if ((E[e-4] ==
CharO) and (E[e-3] == CharK) ) { _RESETOK[1]=1; if (_UDP!=1) {Serial.print("HOT RESET");}}}
}

void GPSSIM808::RESETWARM()
{
        while (_RESETOK[2]==0){
        Serial3.write("AT+CGPSRST=2\n");delay(100);
        if (Serial3.available()) { f=0;  while (Serial3.available()){ F[f] = (Serial3.read()); f++; }} if ((F[f-4] ==
CharO) and (F[f-3] == CharK) ) { _RESETOK[2]=1; if (_UDP!=1) {Serial.print("WARM RESET");}}}
}
```

A.1.4 Programa principal

Este archivo ".ino" ejecutable en el Arduino IDE es el que utilizará las librerías tanto de la IMU como del GPS, obteniendo los resultados de

ambas librerías, para ello se incluyen las librerías y se ejecutan sus funciones principales.

Otro detalle que incluye es la utilización de un LED de la placa Arduino para saber si el programa se encuentra ejecutando el setup (y posiblemente buscando la fijación de satélites), o si está ejecutando el loop.

En este archivo es donde se decidirá el método de comunicación utilizado para la salida de los datos, bien por comunicación serie o bien por comunicación UDP.

"PFM.ino"

```
/*
PFM.ino - Proyecto fin de Master Universidad de Cantabria, "Fusión sensorial para a estimación del estado de un vehículo autónomo".
Realizado por Daniel López Montes y dirigido por Luciano Alonso Renteria. 2016.

*/


#include <GY80.h>
#include <GPSSIM808.h>

//GPSSIM808(UDP,MODO); Si UDP=0 -> Salida de datos por USB, UDP=1 -> Salida de datos por Ethernet.
//Si MODO=0 -> Datos sin filtro Kalman, Si MODO=1-> Datos con filtro Kalman,
GPSSIM808 GPS=GPSSIM808(1,1);

//GY80(UDP, METODO); Si UDP=0 -> Salida de datos por USB, UDP=1 -> Salida de datos por Ethernet
//Si Metodo=0 -> Filtro Complementario con Cuaterniones, Si Metodo=1 -> PI con Matriz DCM.
GY80 IMU=GY80(1,0);

void setup() {
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
  GPS.Inicializar();
 IMU.Inicializar(1);
  digitalWrite(13, LOW);

}

void loop() {

  IMU.getEuler(1);
  GPS.GPSOUT();
}
```

A.2 PROCESSING

A.2.1 Representación ángulos de Euler.

"AHRS.pde"

Programa de Processing para mostrar un prisma que simula la IMU GY-80 para ver su orientación en función de los 3 ángulos de Euler que se obtienen en una cadena de caracteres por puerto serie.

```
import processing.serial.*;

int end = 10;
String serial;
Serial port;

float pitch;
float roll;
float yaw;
float yawOffset;
char metodo;
PImage IMU;

void setup() {

  size(640, 480, OPENGL);
  smooth();
  noStroke();
  textureMode(NORMAL);
  IMU = loadImage("imu.jpg");

  port = new Serial(this, Serial.list()[0], 19200);
  port.clear();
  serial = port.readStringUntil(end);
  serial = null;
}

void draw() {
 background(0);
 lights();
 ReadData();

 pushMatrix();
 translate(width/2, height/2, -350);
 drawBoard();
 popMatrix();

 fill(255);

 textAlign(LEFT);

 text("Presionar 'a' para alinear, 'c' para utilizar cuaterniones, y 'm' para utilizar matriz DCM", 5, 25);


  pushMatrix();
  translate(10, height - 10);
  textAlign(LEFT);
```

```
  text("Yaw: " + yaw, 300, 0);
  text("Pitch: " + pitch, 0, 0);
  text("Roll: " + roll, 150, 0);
    if (metodo == 'M') {text("Matriz DCM",450,0);}
    else if (metodo == 'C') {text("Cuaternion",450,0);}
    else {text("ERROR",450,0);}
  popMatrix();

}

void ReadData () {
    while (port.available() > 0) {
    serial = port.readStringUntil(end);
  }
    if (serial != null) {

      String[] num = split(serial, ';');

      pitch= float(num[0]);
      roll= float(num[1]);
      yaw= float(num[2]);

      if (int(num[3]) == 1) {metodo = 'M';}
      else if (int(num[3]) == 0) {metodo = 'C';}
    }
}

void drawBoard(){
 rotateY(radians(yaw - yawOffset));
 rotateZ(-radians(pitch+90));
 rotateY(radians(roll+90));

 fill(0, 0, 255);
 box(250, 400, 20);
 ImagenIMU(IMU);
}

void ImagenIMU(PImage IMU){
  beginShape(QUADS);
  texture(IMU);
  vertex(-125, -200,  10.5, 0, 0);
  vertex( 125, -200,  10.5, 1, 0);
  vertex( 125,  200,  10.5, 1, 1);
  vertex(-125,  200,  10.5, 0, 1);
  endShape();
}
void keyPressed() {
  switch (key) {
    case 'a':
      yawOffset = yaw;break;
    case 'm' :
      port.write('M');break;
    case 'c':
      port.write('C');break;
  }
}
```

A.3 MATLAB

A.3.1 Cálculo del error estatico del GPS

Esta pequeña función de Matlab permite obtener la máxima diferencia entre los datos obtenidos por el GPS, los cuales se deben almacenar en un archivo ".txt".

"ErrorEstatico.m"

```
clear; close all; clc;format long;B=zeros(2,1);
 Datos=importdata('ErrorEstaticoA2.txt');
 b=cell2mat(strfind(Datos,','));
 for n=1:size(b,1)
    A(n,1)=str2num(Datos{n}((b(n,2)+1):(b(n,3)-1)))/100;
    A(n,2)=str2num(Datos{n}((b(n,4)+1):(b(n,5)-1)))/100;
 end
 plot((A(:,1)-floor(A(:,1)))*100,(A(:,2)-floor(A(:,2)))*100,'.')
 for n=1:size(A,1)
    for k=1:2
    A(n,k)=floor(A(n,k))+(A(n,k)-floor(A(n,k)))*100/60;
    B(k)=B(k)+A(n,k);
    end
 end
 B(2)=-B(2);
 Coordenadas=B/size(A,1)
 DiffLat=max(A(:,1))-min(A(:,1));
 DiffLong=max(A(:,2))-min(A(:,2));
 ErrorLat=(DiffLat/2)*111100
 ErrorLong=(DiffLong/2)*80940
```

A.3.2 Procesamiento de datos de un recorrido

Mediante esta función de matlab se recogen los datos de latitud y longitud guardados en un archivo de texto y se realiza una gráfica con la latitud en metros en el eje X, y la longitud en metros en el eje Y.

"Recorrido.m"

```
clear; close all; clc;format long;
 Datos=importdata('Recorrido.txt');
 b=cell2mat(strfind(Datos,','));
 %Recoger datos del archivo
 for n=1:size(b,1)
    A(n,1)=str2num(Datos{n}((b(n,2)+1):(b(n,3)-1)))/100;
    A(n,2)=str2num(Datos{n}((b(n,4)+1):(b(n,5)-1)))/100;
 end
 %Pasar a grados con decimales
 for n=1:size(A,1)
    for k=1:2
    A(n,k)=floor(A(n,k))+(A(n,k)-floor(A(n,k)))*10/6;
```

```
    end
 end
 A(:,2)=-A(:,2);
 plot(A(:,2),A(:,1),'.')
 %Estableer punto base y distancias en metros.
 BaseLat=min(A(:,1));
 BaseLong=min(A(:,2));
 for n=1:size(A,1)
    B(n,1)=(A(n,1)-BaseLat)*111100;
    B(n,2)=(A(n,2)-BaseLong)*80940;
 end
 B
plot(B(:,2),B(:,1),'.')
```

A.3.3 Procesamiento de datos de un recorrido con y sin filtro Kalman

Similar al código anterior, aunque en esta ocasión está preparado para recoger los datos ya procesados por la librería del GPS, y además utiliza una cadena de caracteres para los datos sin filtro Kalman, y otra para los datos con filtro Kalman. Al final superpone las gráficas creadas de ambos para observar la comparación.

"RecorridoK.m"

```
clear; close all; clc;format long;
Datos=importdata('RecorridoK.txt');
A=[];AK=[];
%Recoger datos del archivo
for n=1:2:size(Datos,1)
AK=[AK;Datos(n,1),Datos(n,2)];
A=[A;Datos(n+1,1),Datos(n,2)];
end
A(:,2)=-A(:,2); AK(:,2)=-AK(:,2);
%plot(A(:,2),A(:,1),'r')
%hold on
%plot (AK(:,2),AK(:,1),'b');

%Estableer punto base y distancias en metros.
BaseLat=min(A(:,1));
BaseLong=min(A(:,2));
for n=1:size(A,1)
B(n,1)=(A(n,1)-BaseLat)*111100;
B(n,2)=(A(n,2)-BaseLong)*80940;
BK(n,1)=(AK(n,1)-BaseLat)*111100;
BK(n,2)=(AK(n,2)-BaseLong)*80940;
end
plot(B(:,2),B(:,1),'r')
hold on
plot(BK(:,2),BK(:,1),'b')
```

ANEXO B: DATASHEETS

En las próximas páginas se encuentran las hojas de datos de los sensores presentes en la IMU GY-80, y que han sido descritos en el apartado 3.3.

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

Data Sheet

ADXL345

FEATURES

Ultralow power: as low as 23 µA in measurement mode and 0.1 µA in standby mode at V_S = 2.5 V (typical)
Power consumption scales automatically with bandwidth
User-selectable resolution
Fixed 10-bit resolution
Full resolution, where resolution increases with *g* range, up to 13-bit resolution at ±16 *g* (maintaining 4 mg/LSB scale factor in all *g* ranges)
Embedded memory management system with FIFO technology minimizes host processor load
Single tap/double tap detection
Activity/inactivity monitoring
Free-fall detection
Supply voltage range: 2.0 V to 3.6 V
I/O voltage range: 1.7 V to V_S
SPI (3- and 4-wire) and I^2C digital interfaces
Flexible interrupt modes mappable to either interrupt pin
Measurement ranges selectable via serial command
Bandwidth selectable via serial command
Wide temperature range (−40°C to +85°C)
10,000 *g* shock survival
Pb free/RoHS compliant
Small and thin: 3 mm × 5 mm × 1 mm LGA package

APPLICATIONS

Handsets
Medical instrumentation
Gaming and pointing devices
Industrial instrumentation
Personal navigation devices
Hard disk drive (HDD) protection

GENERAL DESCRIPTION

The ADXL345 is a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at 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 I^2C 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 (3.9 mg/LSB) enables measurement of inclination changes less than 1.0°.

Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion by comparing the acceleration on any axis with user-set thresholds. Tap sensing detects single and double taps in any direction. Free-fall sensing detects if the device is falling. These functions can be mapped individually to either of two interrupt output pins. An integrated memory management system with a 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor activity and lower overall system power consumption.

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

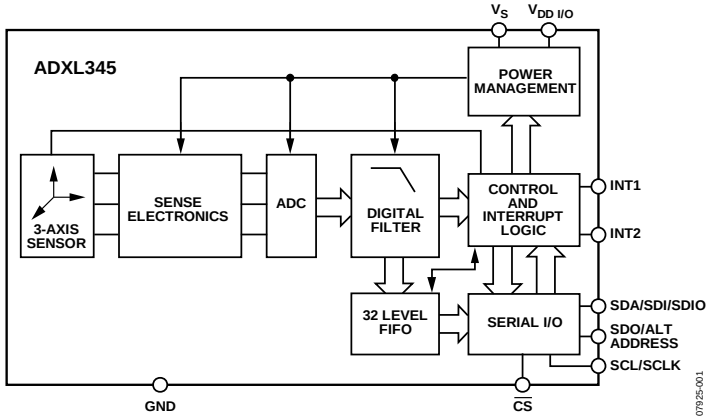


Figure 1.

Rev. E Document Feedback

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

TABLE OF CONTENTS

REVISION HISTORY

6/15—Rev. D to Rev. E

Changes to Features Section and General Description Section 1
Change to Figure 36 15
Change to FIFO Section 21

2/13—Rev. C to Rev. D

Changes to Figure 13, Figure 14, and Figure 15 9
Change to Table 15 22

5/11—Rev. B to Rev. C

Added Preventing Bus Traffic Errors Section 15
Changes to Figure 37, Figure 38, Figure 39 16
Changes to Table 12 19
Changes to Using Self-Test Section 31
Changes to Axes of Acceleration Sensitivity Section 35

11/10—Rev. A to Rev. B

Change to 0 *g* Offset vs. Temperature for Z-Axis Parameter, Table 1 4
Changes to Figure 10 to Figure 15 9
Changes to Ordering Guide 37

4/10—Rev. 0 to Rev. A

Changes to Features Section and General Description Section 1
Changes to Specifications Section 3
Changes to Table 2 and Table 3 5
Added Package Information Section, Figure 2, and Table 4; Renumbered Sequentially 5
Changes to Pin 12 Description, Table 5 6
Added Typical Performance Characteristics Section 7
Changes to Theory of Operation Section and Power Sequencing Section 12
Changes to Powers Savings Section, Table 7, Table 8, Auto Sleep Mode Section, and Standby Mode Section 13
Changes to SPI Section 14
Changes to Figure 36 to Figure 38 15
Changes to Table 9 and Table 10 16
Changes to I^2C Section and Table 11 17
Changes to Table 12 18
Changes to Interrupts Section, Activity Section, Inactivity Section, and FREE_FALL Section 19
Added Table 13 19
Changes to FIFO Section 20
Changes to Self-Test Section and Table 15 to Table 18 21
Added Figures 42 and Table 14 21
Changes to Table 19 22
Changes to Register 0x1D—THRESH_TAP (Read/Write) Section, Register 0x1E, Register 0x1F, Register 0x20—OFSX, OFSY, OSXZ (Read/Write) Section, Register 0x21—DUR (Read/Write) Section, Register 0x22—Latent (Read/Write) Section, and Register 0x23—Window (Read/Write) Section ... 23
Changes to ACT_X Enable Bits and INACT_X Enable Bit Section, Register 0x28—THRESH_FF (Read/Write) Section, Register 0x29—TIME_FF (Read/Write) Section, Asleep Bit Section, and AUTO_SLEEP Bit Section 24
Changes to Sleep Bit Section 25
Changes to Power Supply Decoupling Section, Mechanical Considerations for Mounting Section, and Tap Detection Section 27
Changes to Threshold Section 28
Changes to Sleep Mode vs. Low Power Mode Section 29
Added Offset Calibration Section 29
Changes to Using Self-Test Section 30
Added Data Formatting of Upper Data Rates Section, Figure 48, and Figure 49 31
Added Noise Performance Section, Figure 50 to Figure 52, and Operation at Voltages Other Than 2.5 V Section 32
Added Offset Performance at Lowest Data Rates Section and Figure 53 to Figure 55 33

6/09—Revision 0: Initial Version

SPECIFICATIONS

T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V, acceleration = 0 *g*, C_S = 10 µF tantalum, $C_{I/O}$ = 0.1 µF, output data rate (ODR) = 800 Hz, unless otherwise noted. All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.

Table 1.

Parameter	Test Conditions	Min	Typ[1]	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 resolution		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}	All *g*-ranges, full resolution	230	256	282	LSB/*g*
	±2 *g*, 10-bit resolution	230	256	282	LSB/*g*
	±4 *g*, 10-bit resolution	115	128	141	LSB/*g*
	±8 *g*, 10-bit resolution	57	64	71	LSB/*g*
	±16 *g*, 10-bit resolution	29	32	35	LSB/*g*
Sensitivity Deviation from Ideal	All *g*-ranges		±1.0		%
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	All *g*-ranges, full resolution	3.5	3.9	4.3	mg/LSB
	±2 *g*, 10-bit resolution	3.5	3.9	4.3	mg/LSB
	±4 *g*, 10-bit resolution	7.1	7.8	8.7	mg/LSB
	±8 *g*, 10-bit resolution	14.1	15.6	17.5	mg/LSB
	±16 *g*, 10-bit resolution	28.6	31.2	34.5	mg/LSB
Sensitivity Change Due to Temperature			±0.01		%/°C
0 *g* OFFSET	Each axis				
0 *g* Output for X_{OUT}, Y_{OUT}		−150	0	+150	m*g*
0 *g* Output for Z_{OUT}		−250	0	+250	m*g*
0 *g* Output Deviation from Ideal, X_{OUT}, Y_{OUT}			±35		m*g*
0 *g* Output Deviation from Ideal, Z_{OUT}			±40		m*g*
0 *g* Offset vs. Temperature for X-, Y-Axes			±0.4		mg/°C
0 *g* Offset vs. Temperature for Z-Axis			±1.2		mg/°C
NOISE					
X-, Y-Axes	ODR = 100 Hz for ±2 *g*, 10-bit resolution or all *g*-ranges, full resolution		0.75		LSB rms
Z-Axis	ODR = 100 Hz for ±2 *g*, 10-bit resolution or all *g*-ranges, full resolution		1.1		LSB rms
OUTPUT DATA RATE AND BANDWIDTH	User selectable				
Output Data Rate (ODR)[3, 4, 5]		0.1		3200	Hz
SELF-TEST[6]					
Output Change in X-Axis		0.20		2.10	*g*
Output Change in Y-Axis		−2.10		−0.20	*g*
Output Change in Z-Axis		0.30		3.40	*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	ODR ≥ 100 Hz		140		µA
	ODR < 10 Hz		30		µA
Standby Mode Leakage Current			0.1		µA
Turn-On and Wake-Up Time[7]	ODR = 3200 Hz		1.4		ms

Parameter	Test Conditions	Min	Typ[1]	Max	Unit
TEMPERATURE					
Operating Temperature Range		−40		+85	°C
WEIGHT					
Device Weight			30		m*g*

[1] The typical specifications shown are for at least 68% of the population of parts and are based on the worst case of mean ±1 σ, except for 0 *g* output and sensitivity, which represents the target value. For 0 *g* offset and sensitivity, the deviation from the ideal describes the worst case of mean ±1 σ.
[2] Cross-axis sensitivity is defined as coupling between any two axes.
[3] Bandwidth is the −3 dB frequency and is half the output data rate, bandwidth = ODR/2.
[4] The output format for the 3200 Hz and 1600 Hz ODRs is different than the output format for the remaining ODRs. This difference is described in the Data Formatting of Upper Data Rates section.
[5] Output data rates below 6.25 Hz exhibit additional offset shift with increased temperature, depending on selected output data rate. Refer to the Offset Performance at Lowest Data Rates section for details.
[6] Self-test change is defined as the output (*g*) when the SELF_TEST bit = 1 (in the DATA_FORMAT register, Address 0x31) minus the output (*g*) when the SELF_TEST bit = 0. Due to device filtering, the output reaches its final value after 4 × τ when enabling or disabling self-test, where τ = 1/(data rate). The part must be in normal power operation (LOW_POWER bit = 0 in the BW_RATE register, Address 0x2C) for self-test to operate correctly.
[7] Turn-on and wake-up times are determined by the user-defined bandwidth. At a 100 Hz data rate, the turn-on and wake-up times are each approximately 11.1 ms. For other data rates, the turn-on and wake-up times are each approximately τ + 1.1 in milliseconds, where τ = 1/(data rate).

ABSOLUTE MAXIMUM RATINGS

Table 2.

Parameter	Rating
Acceleration	
Any Axis, Unpowered	10,000 *g*
Any Axis, Powered	10,000 *g*
V_S	−0.3 V to +3.9 V
$V_{DD\ I/O}$	−0.3 V to +3.9 V
Digital Pins	−0.3 V to $V_{DD\ I/O}$ + 0.3 V or 3.9 V, whichever is less
All Other Pins	−0.3 V to +3.9 V
Output Short-Circuit Duration (Any Pin to Ground)	Indefinite
Temperature Range	
Powered	−40°C to +105°C
Storage	−40°C to +105°C

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

THERMAL RESISTANCE

Table 3. Package Characteristics

Package Type	θ_{JA}	θ_{JC}	Device Weight
14-Terminal LGA	150°C/W	85°C/W	30 mg

PACKAGE INFORMATION

The information in Figure 2 and Table 4 provide details about the package branding for the ADXL345. For a complete listing of product availability, see the Ordering Guide section.

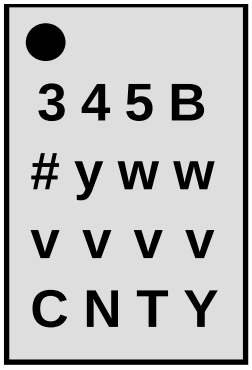


Figure 2. Product Information on Package (Top View)

Table 4. Package Branding Information

Branding Key	Field Description
345B	Part identifier for ADXL345
#	RoHS-compliant designation
yww	Date code
vvvv	Factory lot code
CNTY	Country of origin

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry, damage may occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

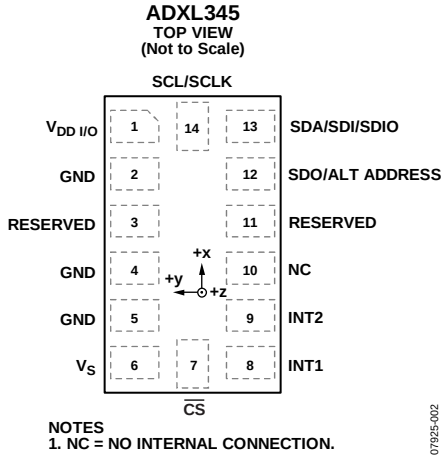


Figure 3. Pin Configuration (Top View)

Table 5. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	$V_{DD\ I/O}$	Digital Interface Supply Voltage.
2	GND	This pin must be connected to ground.
3	RESERVED	Reserved. This pin must be connected to V_S or left open.
4	GND	This pin must be connected to ground.
5	GND	This pin 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	NC	Not Internally Connected.
11	RESERVED	Reserved. This pin must be connected to ground or left open.
12	SDO/ALT ADDRESS	Serial Data Output (SPI 4-Wire)/Alternate I^2C Address Select (I^2C).
13	SDA/SDI/SDIO	Serial Data (I^2C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire).
14	SCL/SCLK	Serial Communications Clock. SCL is the clock for I^2C, and SCLK is the clock for SPI.

TYPICAL PERFORMANCE CHARACTERISTICS

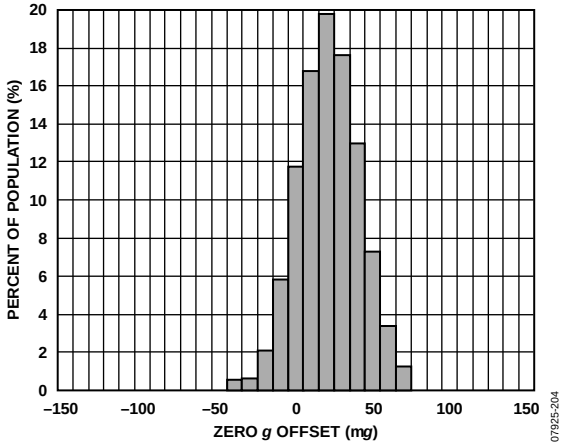


Figure 4. X-Axis Zero g Offset at 25°C, V_S = 2.5 V

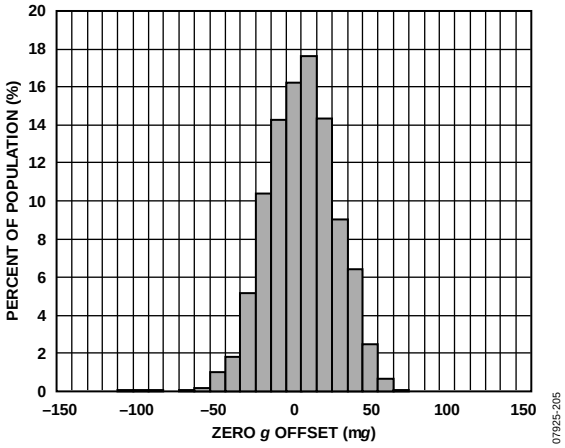


Figure 5. Y-Axis Zero g Offset at 25°C, V_S = 2.5 V

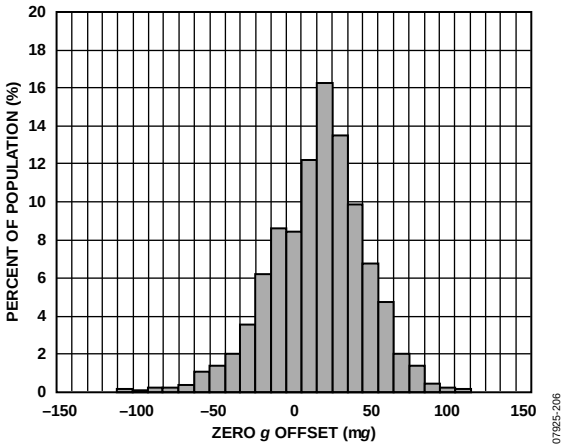


Figure 6. Z-Axis Zero g Offset at 25°C, V_S = 2.5 V

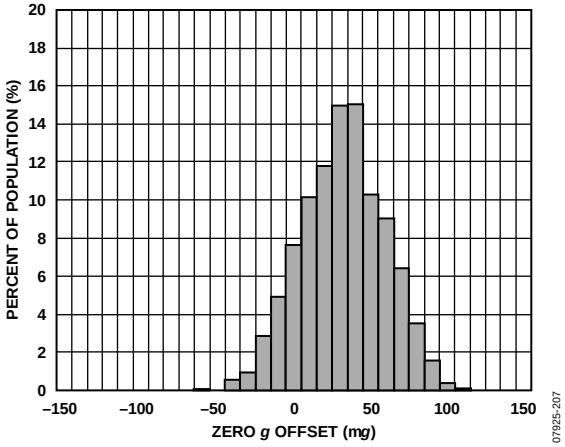


Figure 7. X-Axis Zero g Offset at 25°C, V_S = 3.3 V

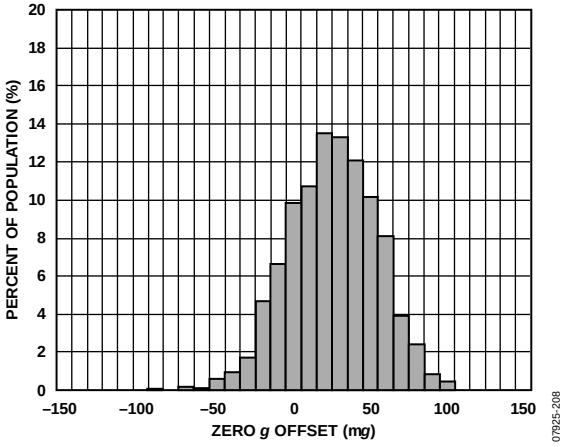


Figure 8. Y-Axis Zero g Offset at 25°C, V_S = 3.3 V

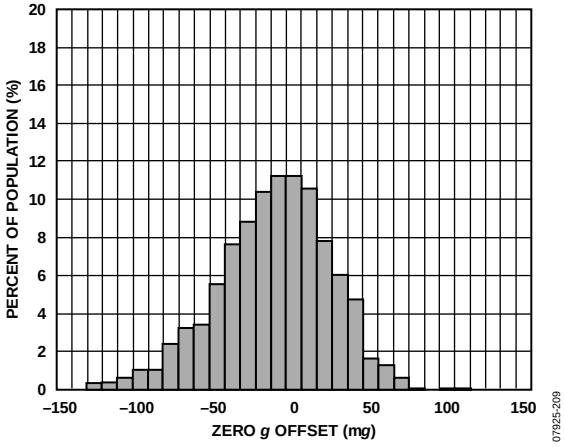


Figure 9. Z-Axis Zero g Offset at 25°C, V_S = 3.3 V

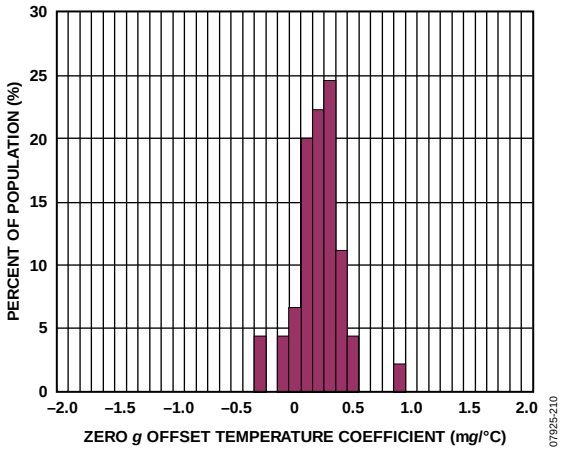


Figure 10. X-Axis Zero g Offset Temperature Coefficient, V_S = 2.5 V

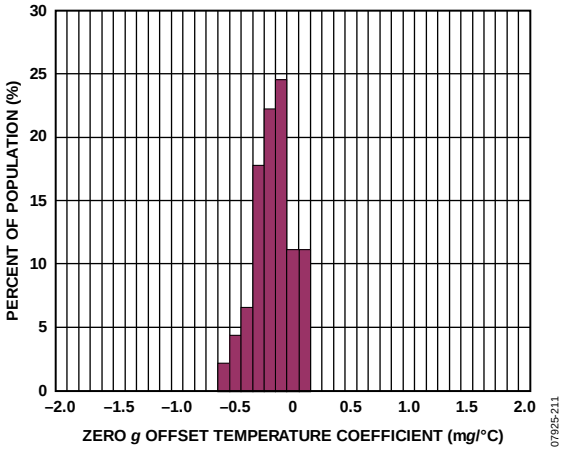


Figure 11. Y-Axis Zero g Offset Temperature Coefficient, V_S = 2.5 V

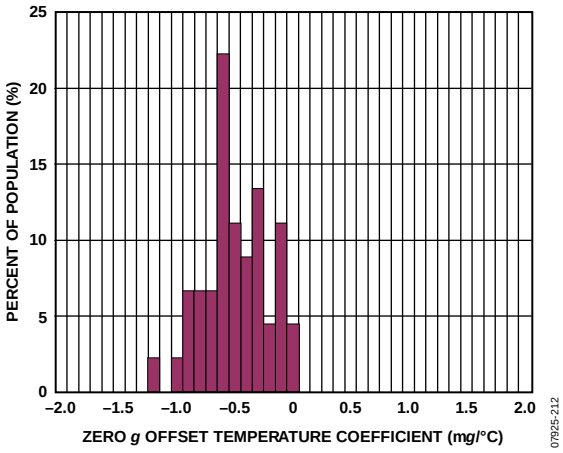


Figure 12. Z-Axis Zero g Offset Temperature Coefficient, V_S = 2.5 V

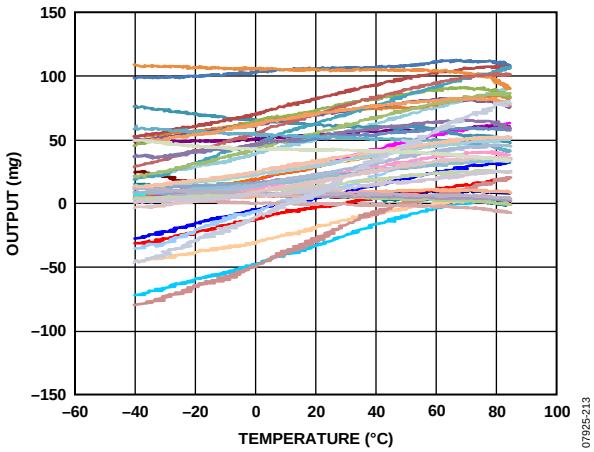


Figure 13. X-Axis Zero g Offset vs. Temperature—45 Parts Soldered to PCB, V_S = 2.5 V

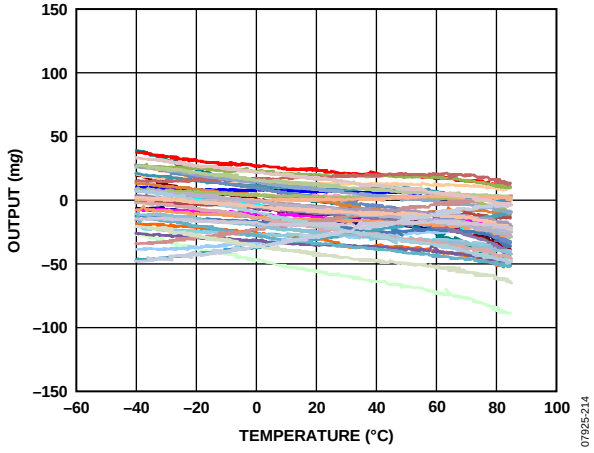


Figure 14. Y-Axis Zero g Offset vs. Temperature—45 Parts Soldered to PCB, V_S = 2.5 V

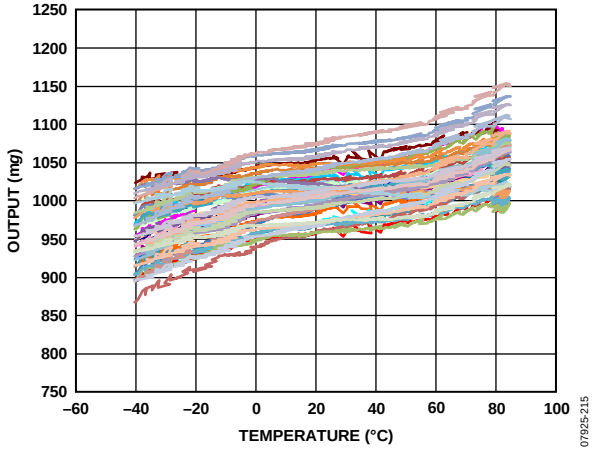


Figure 15. Z-Axis One g Offset vs. Temperature—45 Parts Soldered to PCB, V_S = 2.5 V

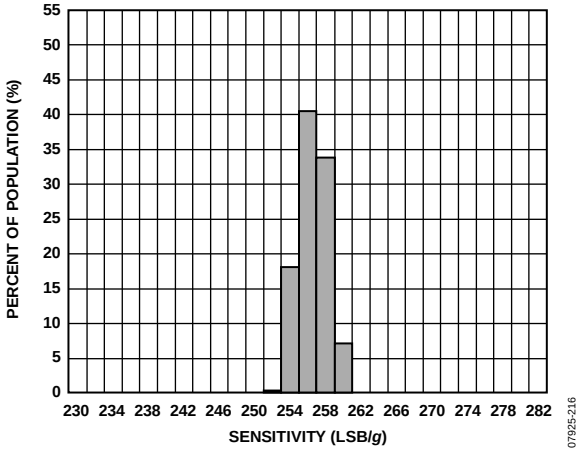


Figure 16. X-Axis Sensitivity at 25°C, V_S = 2.5 V, Full Resolution

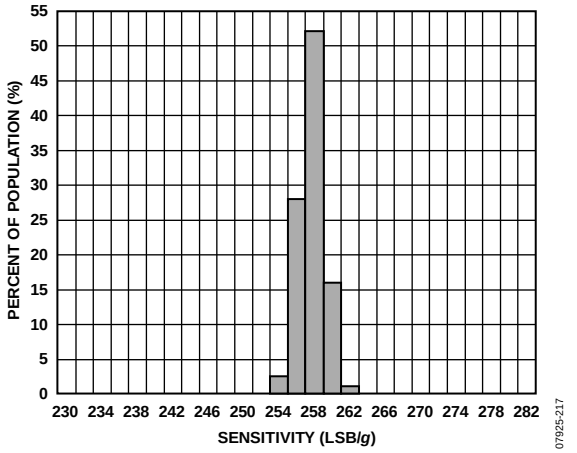


Figure 17. Y-Axis Sensitivity at 25°C, V_S = 2.5 V, Full Resolution

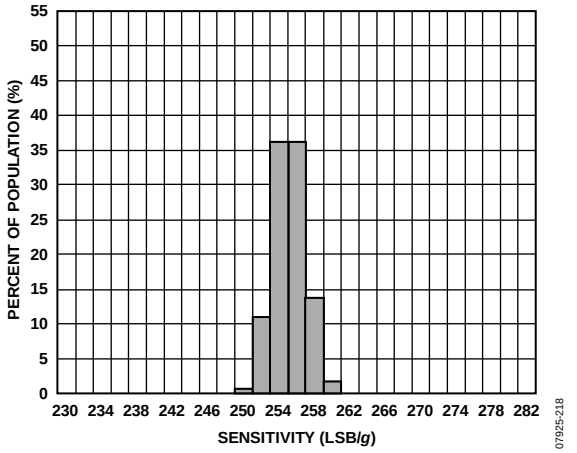


Figure 18. Z-Axis Sensitivity at 25°C, V_S = 2.5 V, Full Resolution

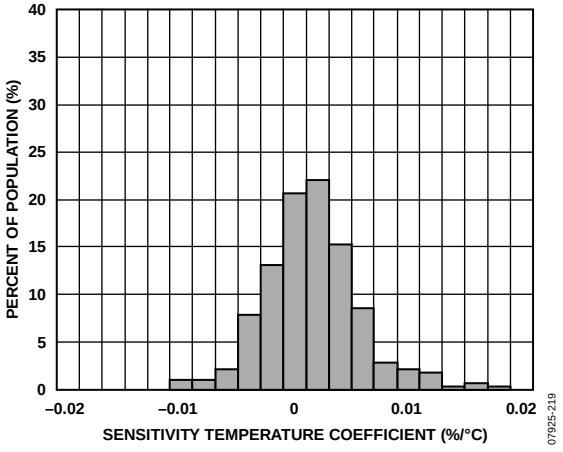


Figure 19. X-Axis Sensitivity Temperature Coefficient, V_S = 2.5 V

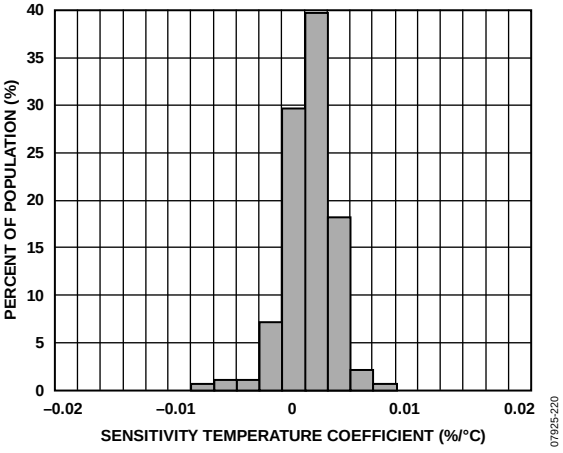


Figure 20. Y-Axis Sensitivity Temperature Coefficient, V_S = 2.5 V

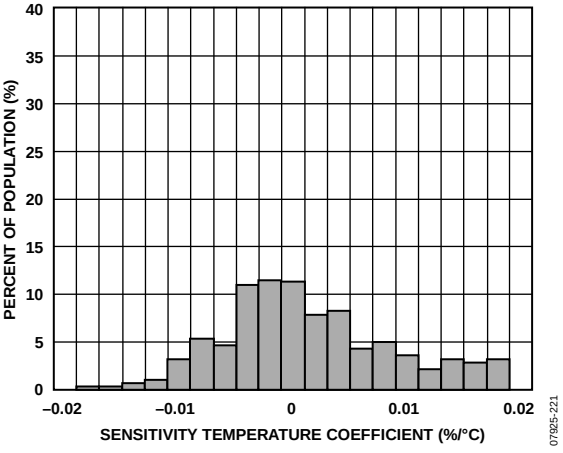


Figure 21. Z-Axis Sensitivity Temperature Coefficient, V_S = 2.5 V

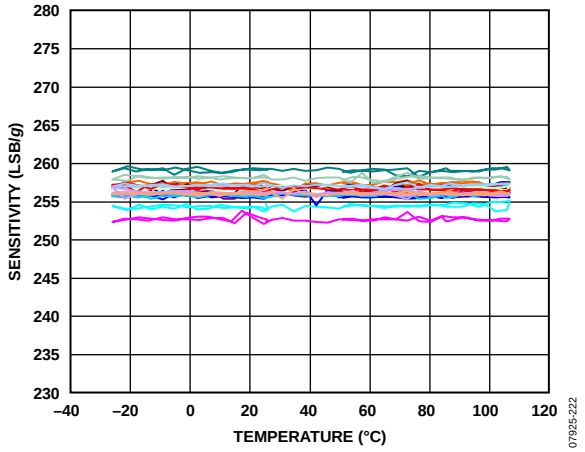


Figure 22. X-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 2.5 V, Full Resolution

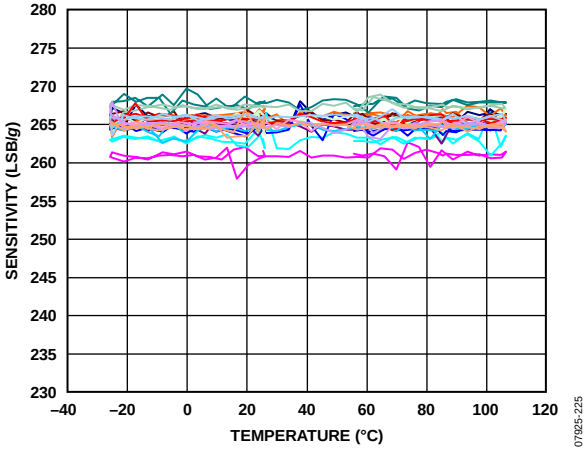


Figure 25. X-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 3.3 V, Full Resolution

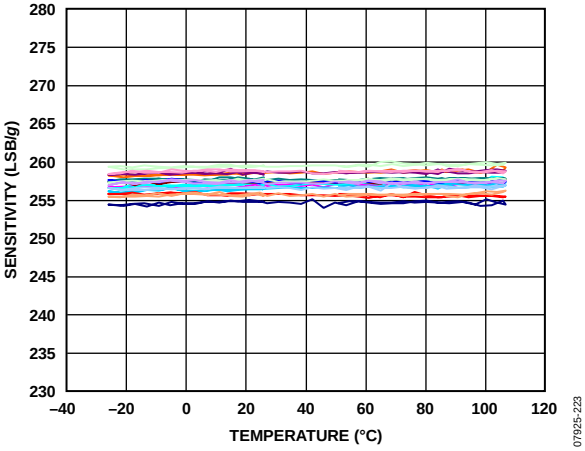


Figure 23. Y-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 2.5 V, Full Resolution

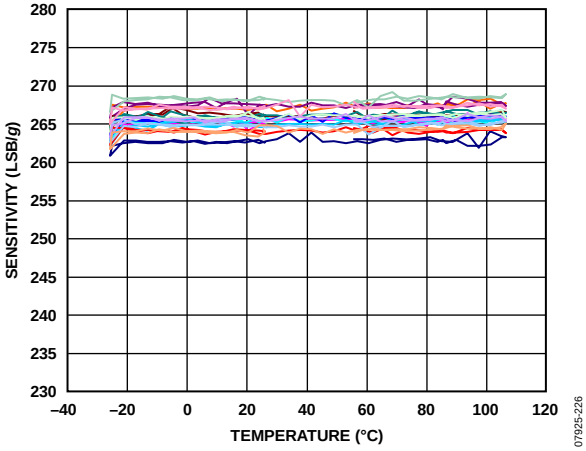


Figure 26. Y-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 3.3 V, Full Resolution

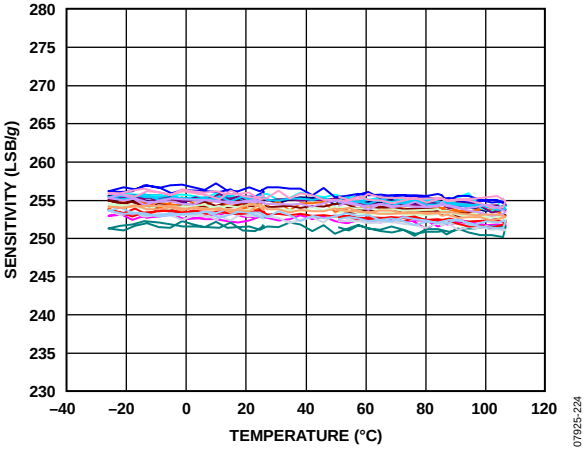


Figure 24. Z-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 2.5 V, Full Resolution

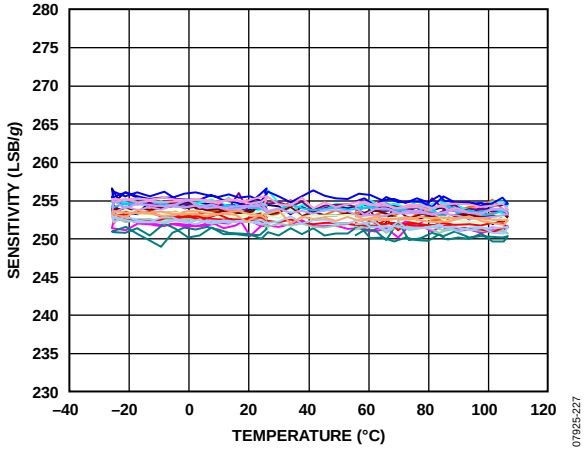


Figure 27. Z-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 3.3 V, Full Resolution

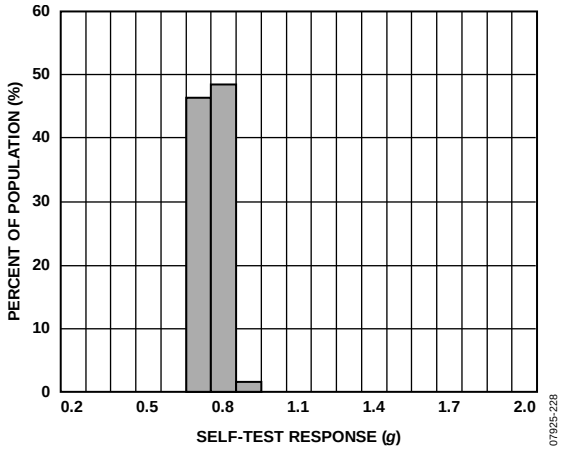


Figure 28. X-Axis Self-Test Response at 25°C, V_S = 2.5 V

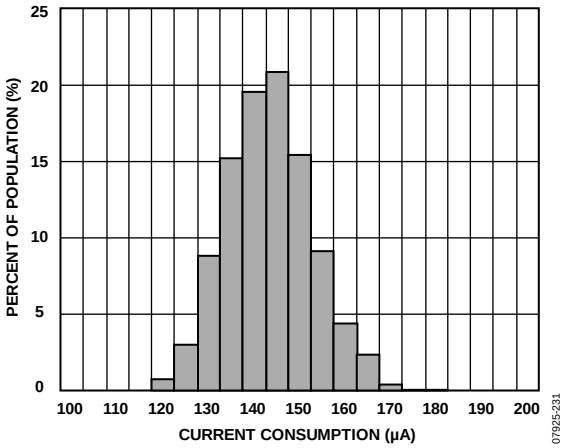


Figure 31. Current Consumption at 25°C, 100 Hz Output Data Rate, V_S = 2.5 V

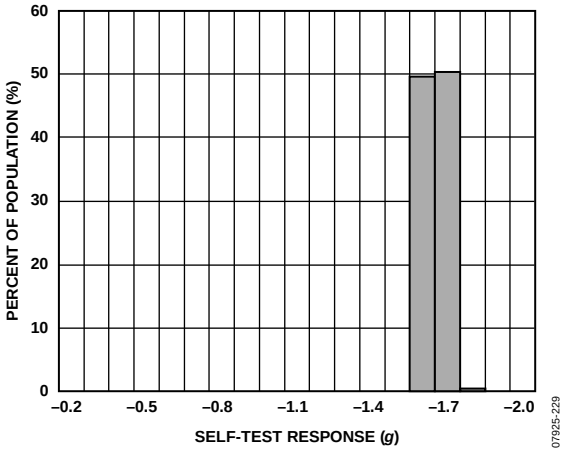


Figure 29. Y-Axis Self-Test Response at 25°C, V_S = 2.5 V

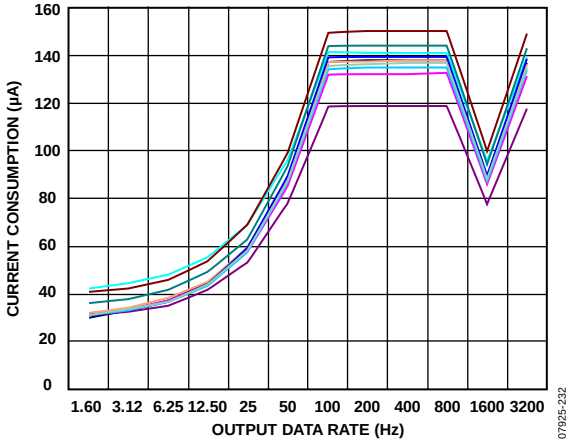


Figure 32. Current Consumption vs. Output Data Rate at 25°C—10 Parts, V_S = 2.5 V

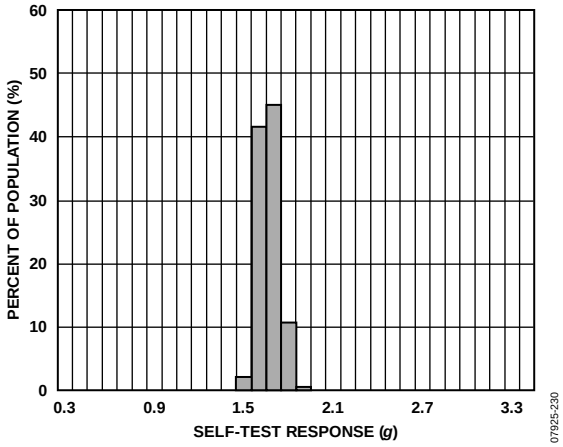


Figure 30. Z-Axis Self-Test Response at 25°C, V_S = 2.5 V

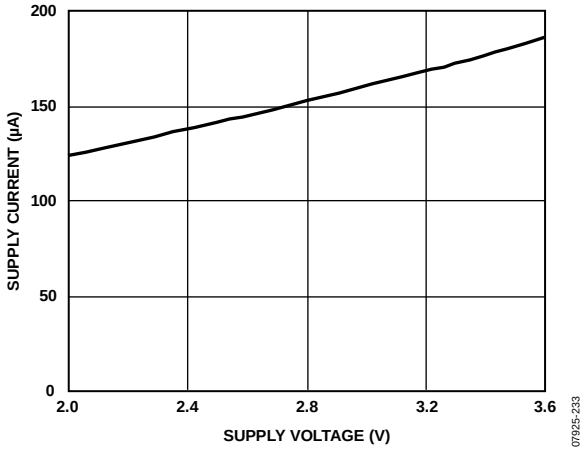


Figure 33. Supply Current vs. Supply Voltage, V_S at 25°C

THEORY OF OPERATION

The ADXL345 is a complete 3-axis acceleration measurement system with a selectable measurement range of ±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, that allows the device 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 forces due to applied acceleration.

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 proof mass 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 can be applied to V_S or $V_{DD\ I/O}$ in any sequence without damaging the ADXL345. All possible power-on modes are summarized in Table 6. 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 communication bus. For single-supply operation, $V_{DD\ I/O}$ can be the same as the main supply, V_S. In a dual-supply application, however, $V_{DD\ I/O}$ can differ from V_S to accommodate the desired interface voltage, as long as V_S is greater than or equal to $V_{DD\ I/O}$.

After V_S is applied, the device enters standby mode, where power consumption is minimized and the device waits for $V_{DD\ I/O}$ to be applied and for the command to enter measurement mode to be received. (This command can be initiated by setting the measure bit (Bit D3) in the POWER_CTL register (Address 0x2D).) In addition, while the device is in standby mode, any register can be written to or read from to configure the part. It is recommended to configure the device in standby mode and then to enable measurement mode. Clearing the measure bit returns the device to the standby mode.

Table 6. Power Sequencing

Condition	V_S	$V_{DD\ I/O}$	Description
Power Off	Off	Off	The device is completely off, but there is a potential for a communication bus conflict.
Bus Disabled	On	Off	The device is on in standby mode, but communication is unavailable and creates a conflict on the communication bus. The duration of this state should be minimized during power-up to prevent a conflict.
Bus Enabled	Off	On	No functions are available, but the device does not create a conflict on the communication 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. After the device is instructed to enter measurement mode, all sensor functions are available.

POWER SAVINGS

Power Modes

The ADXL345 automatically modulates its power consumption in proportion to its output data rate, as outlined in Table 7. 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 Hz to 400 Hz data rate range at the expense of slightly greater noise. To enter low power mode, set the LOW_POWER bit (Bit 4) in the BW_RATE register (Address 0x2C). Table 8 shows the current consumption in low power mode for cases where there is an advantage to using low power mode. Use of low power mode for a data rate not shown in Table 8 does not provide any advantage over the same data rate in normal power mode. Therefore, it is recommended that only data rates shown in Table 8 be used in low power mode. The current consumption values shown in Table 7 and Table 8 are for a V_S of 2.5 V.

Table 7. Typical Current Consumption vs. Data Rate (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (μA)
3200	1600	1111	140
1600	800	1110	90
800	400	1101	140
400	200	1100	140
200	100	1011	140
100	50	1010	140
50	25	1001	90
25	12.5	1000	60
12.5	6.25	0111	50
6.25	3.13	0110	45
3.13	1.56	0101	40
1.56	0.78	0100	34
0.78	0.39	0011	23
0.39	0.20	0010	23
0.20	0.10	0001	23
0.10	0.05	0000	23

Table 8. Typical Current Consumption vs. Data Rate, Low Power Mode (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (μA)
400	200	1100	90
200	100	1011	60
100	50	1010	50
50	25	1001	45
25	12.5	1000	40
12.5	6.25	0111	34

Auto Sleep Mode

Additional power can be saved if the ADXL345 automatically switches to sleep mode during periods of inactivity. To enable this feature, set the THRESH_INACT register (Address 0x25) and the TIME_INACT register (Address 0x26) each to a value that signifies inactivity (the appropriate value depends on the application), and then set the AUTO_SLEEP bit (Bit D4) and the link bit (Bit D5) in the POWER_CTL register (Address 0x2D). Current consumption at the sub-12.5 Hz data rates that are used in this mode is typically 23 μA for a V_S of 2.5 V.

Standby Mode

For even lower power operation, standby mode can be used. In standby mode, current consumption is reduced to 0.1 μA (typical). In this mode, no measurements are made. Enter standby mode by clearing the measure bit (Bit D3) in the POWER_CTL register (Address 0x2D). Placing the device into standby mode preserves the contents of 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 $V_{DD\ I/O}$. The $\overline{CS}$ pin should always be tied high to $V_{DD\ I/O}$ or be driven by an external controller because there is no default mode if the $\overline{CS}$ pin is left unconnected. Therefore, not taking these precautions may result in an inability to communicate with the part. 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- or 4-wire configuration is possible, as shown in the connection diagrams in Figure 34 and Figure 35. Clearing the SPI bit (Bit D6) in the DATA_FORMAT register (Address 0x31) selects 4-wire mode, whereas setting the SPI bit selects 3-wire mode. The maximum SPI clock speed is 5 MHz with 100 pF maximum loading, and the timing scheme follows clock polarity (CPOL) = 1 and clock phase (CPHA) = 1. If power is applied to the ADXL345 before the clock polarity and phase of the host processor are configured, the $\overline{CS}$ pin should be brought high before changing the clock polarity and phase. When using 3-wire SPI, it is recommended that the SDO pin be either pulled up to $V_{DD\ I/O}$ or pulled down to GND via a 10 kΩ resistor.

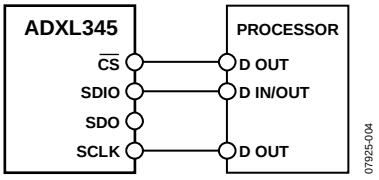


Figure 34. 3-Wire SPI Connection Diagram

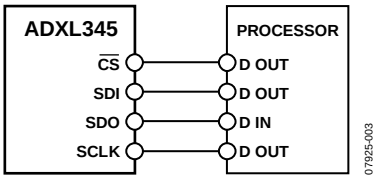


Figure 35. 4-Wire SPI Connection Diagram

$\overline{CS}$ is the serial port enable line and is controlled by the SPI master. This line must go low at the start of a transmission and high at the end of a transmission, as shown in Figure 37. SCLK is the serial port clock and is supplied by the SPI master. SCLK should idle high during a period of no transmission. SDI and SDO are the serial data input and output, respectively. Data is updated on the falling edge of SCLK and should be sampled on the rising edge of SCLK.

To read or write multiple bytes in a single transmission, the multiple-byte bit, located after the R/$\overline{W}$ bit in the first byte transfer (MB in Figure 37 to Figure 39), must be set. After the register addressing and the first byte of data, each subsequent set of clock pulses (eight clock pulses) causes the ADXL345 to point to the next register for a read or write. This shifting continues until the clock pulses cease and $\overline{CS}$ is deasserted. To perform reads or writes on different, nonsequential registers, $\overline{CS}$ must be deasserted between transmissions and the new register must be addressed separately.

The timing diagram for 3-wire SPI reads or writes is shown in Figure 39. The 4-wire equivalents for SPI writes and reads are shown in Figure 37 and Figure 38, respectively. For correct operation of the part, the logic thresholds and timing parameters in Table 9 and Table 10 must be met at all times.

Use of the 3200 Hz and 1600 Hz output data rates is only recommended with SPI communication rates greater than or equal to 2 MHz. The 800 Hz output data rate is recommended only for communication speeds greater than or equal to 400 kHz, and the remaining data rates scale proportionally. For example, the minimum recommended communication speed for a 200 Hz output data rate is 100 kHz. Operation at an output data rate above the recommended maximum may result in undesirable effects on the acceleration data, including missing samples or additional noise.

Preventing Bus Traffic Errors

The ADXL346 $\overline{CS}$ pin is used both for initiating SPI transactions, and for enabling I²C mode. When the ADXL346 is used on a SPI bus with multiple devices, its $\overline{CS}$ pin is held high while the master communicates with the other devices. There may be conditions where a SPI command transmitted to another device looks like a valid I²C command. In this case, the ADXL346 would interpret this as an attempt to communicate in I²C mode, and could interfere with other bus traffic. Unless bus traffic can be adequately controlled to assure such a condition never occurs, it is recommended to add a logic gate in front of the SDI pin as shown in Figure 36. This OR gate will hold the SDA line high when $\overline{CS}$ is high to prevent SPI bus traffic at the ADXL346 from appearing as an I²C start command.

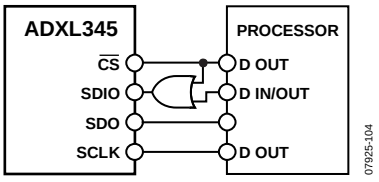


Figure 36. Recommended SPI Connection Diagram when Using Multiple SPI Devices on a Single Bus

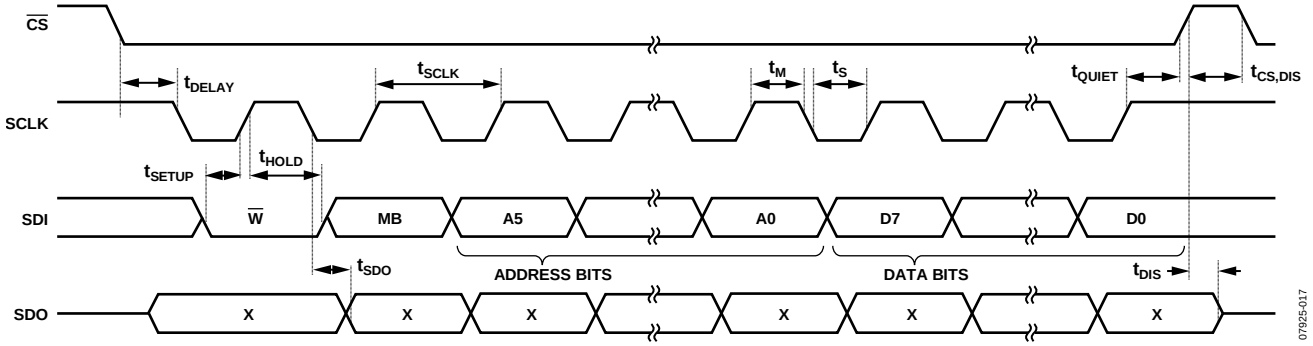


Figure 37. SPI 4-Wire Write

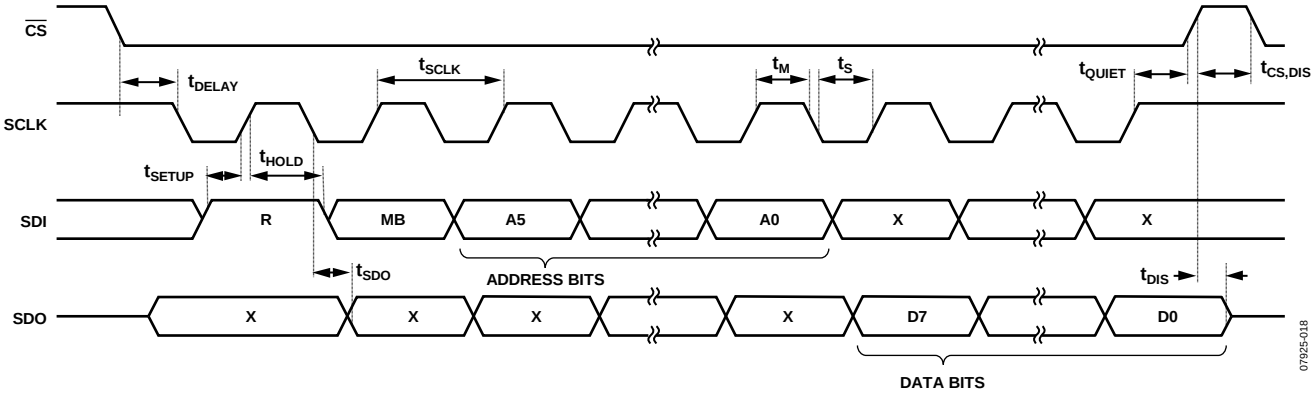
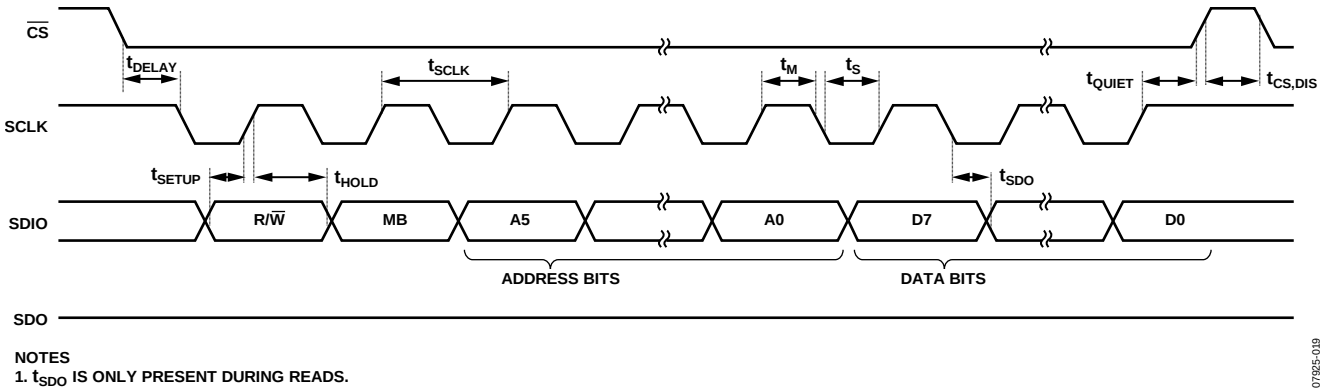


Figure 38. SPI 4-Wire Read



NOTES
1. t_{SDO} IS ONLY PRESENT DURING READS.

Figure 39. SPI 3-Wire Read/Write

Table 9. SPI Digital Input/Output

Parameter	Test Conditions	Limit[1] Min	Limit[1] Max	Unit
Digital Input				
Low Level Input Voltage (V_{IL})			$0.3 \times V_{DD\ I/O}$	V
High Level Input Voltage (V_{IH})		$0.7 \times V_{DD\ I/O}$		V
Low Level Input Current (I_{IL})	$V_{IN} = V_{DD\ I/O}$		0.1	μA
High Level Input Current (I_{IH})	$V_{IN} = 0$ V	−0.1		μA
Digital Output				
Low Level Output Voltage (V_{OL})	$I_{OL} = 10$ mA		$0.2 \times V_{DD\ I/O}$	V
High Level Output Voltage (V_{OH})	$I_{OH} = -4$ mA	$0.8 \times V_{DD\ I/O}$		V
Low Level Output Current (I_{OL})	$V_{OL} = V_{OL,\ max}$	10		mA
High Level Output Current (I_{OH})	$V_{OH} = V_{OH,\ min}$		−4	mA
Pin Capacitance	$f_{IN} = 1$ MHz, $V_{IN} = 2.5$ V		8	pF

[1] Limits based on characterization results, not production tested.

Table 10. SPI Timing (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)[1]

Parameter	Limit[2, 3] Min	Limit[2, 3] Max	Unit	Description
f_{SCLK}		5	MHz	SPI clock frequency
t_{SCLK}	200		ns	1/(SPI clock frequency) mark-space ratio for the SCLK input is 40/60 to 60/40
t_{DELAY}	5		ns	$\overline{CS}$ falling edge to SCLK falling edge
t_{QUIET}	5		ns	SCLK rising edge to $\overline{CS}$ rising edge
t_{DIS}		10	ns	$\overline{CS}$ rising edge to SDO disabled
$t_{CS,DIS}$	150		ns	$\overline{CS}$ deassertion between SPI communications
t_S	$0.3 \times t_{SCLK}$		ns	SCLK low pulse width (space)
t_M	$0.3 \times t_{SCLK}$		ns	SCLK high pulse width (mark)
t_{SETUP}	5		ns	SDI valid before SCLK rising edge
t_{HOLD}	5		ns	SDI valid after SCLK rising edge
t_{SDO}		40	ns	SCLK falling edge to SDO/SDIO output transition
t_R[4]		20	ns	SDO/SDIO output high to output low transition
t_F[4]		20	ns	SDO/SDIO output low to output high transition

[1] The $\overline{CS}$, SCLK, SDI, and SDO pins are not internally pulled up or down; they must be driven for proper operation.
[2] Limits based on characterization results, characterized with f_{SCLK} = 5 MHz and bus load capacitance of 100 pF; not production tested.
[3] The timing values are measured corresponding to the input thresholds (V_{IL} and V_{IH}) given in Table 9.
[4] Output rise and fall times measured with capacitive load of 150 pF.

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 40. The ADXL345 conforms to the *UM10204 I²C-Bus Specification and User Manual*, Rev. 03—19 June 2007, available from NXP Semiconductors. It supports standard (100 kHz) and fast (400 kHz) data transfer modes if the bus parameters given in Table 11 and Table 12 are met. Single- or multiple-byte reads/writes are supported, as shown in Figure 41. With the ALT ADDRESS pin high, the 7-bit I²C address for the device is 0x1D, followed by the R/$\overline{W}$ bit. This translates to 0x3A for a write and 0x3B for a read. An alternate I²C address of 0x53 (followed by the R/$\overline{W}$ bit) can be chosen by grounding the ALT ADDRESS pin (Pin 12). This translates to 0xA6 for a write and 0xA7 for a read.

There are no internal pull-up or pull-down resistors for any unused pins; therefore, there is no known state or default state for the $\overline{CS}$ or ALT ADDRESS pin if left floating or unconnected. It is required that the $\overline{CS}$ pin be connected to $V_{DD\ I/O}$ and that the ALT ADDRESS pin be connected to either $V_{DD\ I/O}$ or GND when using I²C.

Due to communication speed limitations, the maximum output data rate when using 400 kHz I²C is 800 Hz and scales linearly with a change in the I²C communication speed. For example, using I²C at 100 kHz would limit the maximum ODR to 200 Hz. Operation at an output data rate above the recommended maxi-mum may result in undesirable effect on the acceleration data, including missing samples or additional noise.

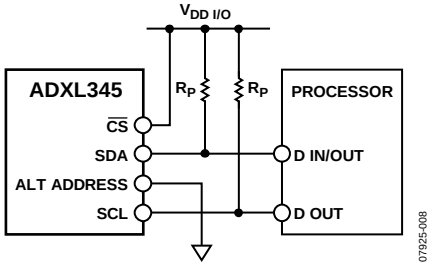


Figure 40. I²C Connection Diagram (Address 0x53)

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. External pull-up resistors, R_P, are necessary for proper I²C operation. Refer to the *UM10204 I²C-Bus Specification and User Manual*, Rev. 03—19 June 2007, when selecting pull-up resistor values to ensure proper operation.

Table 11. I²C Digital Input/Output

Parameter	Test Conditions	Limit[1] Min	Limit[1] Max	Unit
Digital Input				
Low Level Input Voltage (V_{IL})			$0.3 \times V_{DD\ I/O}$	V
High Level Input Voltage (V_{IH})		$0.7 \times V_{DD\ I/O}$		V
Low Level Input Current (I_{IL})	$V_{IN} = V_{DD\ I/O}$		0.1	μA
High Level Input Current (I_{IH})	$V_{IN} = 0$ V	−0.1		μA
Digital Output				
Low Level Output Voltage (V_{OL})	$V_{DD\ I/O} < 2$ V, $I_{OL} = 3$ mA		$0.2 \times V_{DD\ I/O}$	V
	$V_{DD\ I/O} \geq 2$ V, $I_{OL} = 3$ mA		400	mV
Low Level Output Current (I_{OL})	$V_{OL} = V_{OL,\ max}$	3		mA
Pin Capacitance	$f_{IN} = 1$ MHz, $V_{IN} = 2.5$ V		8	pF

[1] Limits based on characterization results; not production tested.

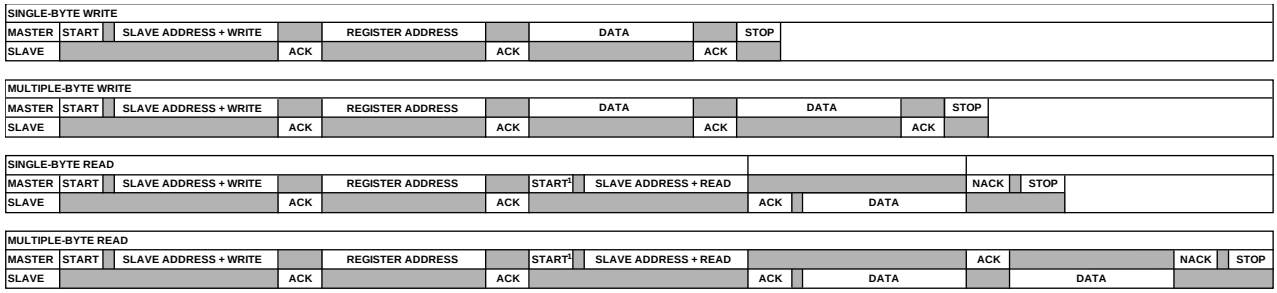


Figure 41. I²C Device Addressing

Table 12. I²C Timing (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

	Limit[1, 2]			
Parameter	**Min**	**Max**	**Unit**	**Description**
f_{SCL}		400	kHz	SCL clock frequency
t_1	2.5		µs	SCL cycle time
t_2	0.6		µs	t_{HIGH}, SCL high time
t_3	1.3		µs	t_{LOW}, SCL low time
t_4	0.6		µs	$t_{HD, STA}$, start/repeated start condition hold time
t_5	100		ns	$t_{SU, DAT}$, data setup time
t_6[3, 4, 5, 6]	0	0.9	µs	$t_{HD, DAT}$, data hold time
t_7	0.6		µs	$t_{SU, STA}$, setup time for repeated start
t_8	0.6		µs	$t_{SU, STO}$, stop condition setup time
t_9	1.3		µs	t_{BUF}, bus-free time between a stop condition and a start condition
t_{10}		300	ns	t_R, rise time of both SCL and SDA when receiving
	0		ns	t_R, rise time of both SCL and SDA when receiving or transmitting
t_{11}		300	ns	t_F, fall time of SDA when receiving
		250	ns	t_F, fall time of both SCL and SDA when transmitting
C_b		400	pF	Capacitive load for each bus line

[1] Limits based on characterization results, with f_{SCL} = 400 kHz and a 3 mA sink current; not production tested.
[2] All values referred to the V_{IH} and the V_{IL} levels given in Table 11.
[3] t_6 is the data hold time that is measured from the falling edge of SCL. It applies to data in transmission and acknowledge.
[4] A transmitting device must internally provide an output hold time of at least 300 ns for the SDA signal (with respect to $V_{IH(min)}$ of the SCL signal) to bridge the undefined region of the falling edge of SCL.
[5] The maximum t_6 value must be met only if the device does not stretch the low period (t_3) of the SCL signal.
[6] The maximum value for t_6 is a function of the clock low time (t_3), the clock rise time (t_{10}), and the minimum data setup time ($t_{5(min)}$). This value is calculated as $t_{6(max)} = t_3 - t_{10} - t_{5(min)}$.

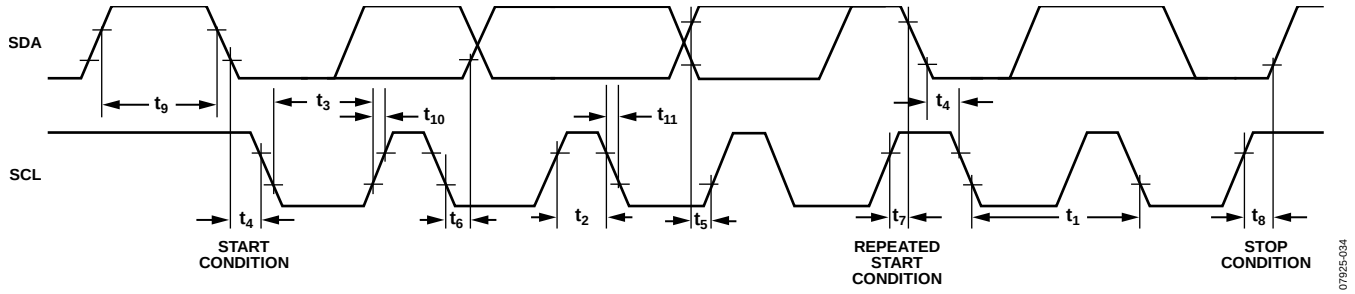


Figure 42. I²C Timing Diagram

INTERRUPTS

The ADXL345 provides two output pins for driving interrupts: INT1 and INT2. Both interrupt pins are push-pull, low impedance pins with output specifications shown in Table 13. The default configuration of the interrupt pins is active high. This can be changed to active low by setting the INT_INVERT bit in the DATA_FORMAT (Address 0x31) register. 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 (Address 0x2E) and are mapped to either the INT1 pin or the INT2 pin based on the contents of the INT_MAP register (Address 0x2F). When initially configuring the interrupt pins, it is recommended that the functions and interrupt mapping be done before enabling the interrupts. When changing the configuration of an interrupt, it is recommended that the interrupt be disabled first, by clearing the bit corresponding to that function in the INT_ENABLE register, and then the function be reconfigured before enabling the interrupt again. Configuration of the functions while the interrupts are disabled helps to prevent the accidental generation of an interrupt before desired.

The interrupt functions are latched and cleared by either reading the data registers (Address 0x32 to Address 0x37) until the interrupt condition is no longer valid for the data-related interrupts or by reading the INT_SOURCE register (Address 0x30) for the remaining interrupts. This section describes the interrupts that can be set in the INT_ENABLE register and monitored in the INT_SOURCE register.

DATA_READY

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

SINGLE_TAP

The SINGLE_TAP bit is set when a single acceleration event that is greater than the value in the THRESH_TAP register (Address 0x1D) occurs for less time than is specified in the DUR register (Address 0x21).

DOUBLE_TAP

The DOUBLE_TAP bit is set when two acceleration events that are greater than the value in the THRESH_TAP register (Address 0x1D) occur for less time than is specified in the DUR register (Address 0x21), with the second tap starting after the time specified by the latent register (Address 0x22) but within the time specified in the window register (Address 0x23). See the Tap Detection section for more details.

Activity

The activity bit is set when acceleration greater than the value stored in the THRESH_ACT register (Address 0x24) is experienced on any participating axis, set by the ACT_INACT_CTL register (Address 0x27).

Inactivity

The inactivity bit is set when acceleration of less than the value stored in the THRESH_INACT register (Address 0x25) is experienced for more time than is specified in the TIME_INACT register (Address 0x26) on all participating axes, as set by the ACT_INACT_CTL register (Address 0x27). The maximum value for TIME_INACT is 255 sec.

FREE_FALL

The FREE_FALL bit is set when acceleration of less than the value stored in the THRESH_FF register (Address 0x28) is experienced for more time than is specified in the TIME_FF register (Address 0x29) on all axes (logical AND). The FREE_FALL interrupt differs from the inactivity interrupt as follows: all axes always participate and are logically AND'ed, the timer period is much smaller (1.28 sec maximum), and the mode of operation is always dc-coupled.

Watermark

The watermark bit is set when the number of samples in FIFO equals the value stored in the samples bits (Register FIFO_CTL, Address 0x38). The watermark bit is cleared automatically when FIFO is read, and the content returns to a value below the value stored in the samples bits.

Table 13. Interrupt Pin Digital Output

Parameter	Test Conditions	Limit[1] Min	Limit[1] Max	Unit
Digital Output				
Low Level Output Voltage (V_{OL})	I_{OL} = 300 µA		0.2 × $V_{DD\ I/O}$	V
High Level Output Voltage (V_{OH})	I_{OH} = −150 µA	0.8 × $V_{DD\ I/O}$		V
Low Level Output Current (I_{OL})	V_{OL} = $V_{OL, max}$	300		µA
High Level Output Current (I_{OH})	V_{OH} = $V_{OH, min}$		−150	µA
Pin Capacitance	f_{IN} = 1 MHz, V_{IN} = 2.5 V		8	pF
Rise/Fall Time				
Rise Time (t_R)[2]	C_{LOAD} = 150 pF		210	ns
Fall Time (t_F)[3]	C_{LOAD} = 150 pF		150	ns

[1] Limits based on characterization results, not production tested.
[2] Rise time is measured as the transition time from $V_{OL, max}$ to $V_{OH, min}$ of the interrupt pin.
[3] Fall time is measured as the transition time from $V_{OH, min}$ to $V_{OL, max}$ of the interrupt pin.

Overrun

The overrun bit is set when new data replaces unread data. The precise operation of the overrun function depends on the FIFO mode. In bypass mode, the overrun bit is set when new data replaces unread data in the DATAX, DATAY, and DATAZ registers (Address 0x32 to Address 0x37). In all other modes, the overrun bit is set when FIFO is filled. The overrun bit is automatically cleared when the contents of FIFO are read.

FIFO

The ADXL345 contains technology for an embedded memory management system with 32-level FIFO that can be used to minimize host processor burden. This buffer has four modes: bypass, FIFO, stream, and trigger (see FIFO Modes). Each mode is selected by the settings of the FIFO_MODE bits (Bits[D7:D6]) in the FIFO_CTL register (Address 0x38).

Bypass Mode

In bypass mode, FIFO is not operational and, therefore, remains empty.

FIFO Mode

In FIFO mode, data from measurements of the x-, y-, and z-axes are stored in FIFO. When the number of samples in FIFO equals the level specified in the samples bits of the FIFO_CTL register (Address 0x38), the watermark interrupt is set. FIFO continues accumulating samples until it is full (32 samples from measurements of the x-, y-, and z-axes) and then stops collecting data. After FIFO stops collecting data, the device continues to operate; therefore, features such as tap detection can be used after FIFO is full. The watermark interrupt continues to occur until the number of samples in FIFO is less than the value stored in the samples bits of the FIFO_CTL register.

Stream Mode

In stream mode, data from measurements of the x-, y-, and z-axes are stored in FIFO. When the number of samples in FIFO equals the level specified in the samples bits of the FIFO_CTL register (Address 0x38), the watermark interrupt is set. FIFO continues accumulating samples and holds the latest 32 samples from measurements of the x-, y-, and z-axes, discarding older data as new data arrives. The watermark interrupt continues occurring until the number of samples in FIFO is less than the value stored in the samples bits of the FIFO_CTL register.

Trigger Mode

In trigger mode, FIFO accumulates samples, holding the latest 32 samples from measurements of the x-, y-, and z-axes. After a trigger event occurs and an interrupt is sent to the INT1 or INT2 pin (determined by the trigger bit in the FIFO_CTL register), FIFO keeps the last n samples (where n is the value specified by the samples bits in the FIFO_CTL register) and then operates in FIFO mode, collecting new samples only when FIFO is not full. A delay of at least 5 µs should be present between the trigger event occurring and the start of reading data from the FIFO to allow the FIFO to discard and retain the necessary samples. Additional trigger events cannot be recognized until the trigger mode is reset. To reset the trigger mode, set the device to bypass mode and then set the device back to trigger mode. Note that the FIFO data should be read first because placing the device into bypass mode clears FIFO.

Retrieving Data from FIFO

The FIFO data is read through the DATAX, DATAY, and DATAZ registers (Address 0x32 to Address 0x37). When the FIFO is in FIFO, stream, or trigger mode, reads to the DATAX, DATAY, and DATAZ registers read data stored in the FIFO. Each time data is read from the FIFO, the oldest x-, y-, and z-axes data are placed into the DATAX, DATAY and DATAZ registers.

If a single-byte read operation is performed, the remaining bytes of data for the current FIFO sample are lost. Therefore, all axes of interest should be read in a burst (or multiple-byte) read operation. To ensure that the FIFO has completely popped (that is, that new data has completely moved into the DATAX, DATAY, and DATAZ registers), there must be at least 5 µs between the end of reading the data registers and the start of a new read of the FIFO or a read of the FIFO_STATUS register (Address 0x39). The end of reading a data register is signified by the transition from Register 0x37 to Register 0x38 or by the $\overline{CS}$ pin going high.

For SPI operation at 1.6 MHz or less, the register addressing portion of the transmission is a sufficient delay to ensure that the FIFO has completely popped. For SPI operation greater than 1.6 MHz, it is necessary to deassert the $\overline{CS}$ pin to ensure a total delay of 5 µs; otherwise, the delay is not sufficient. The total delay necessary for 5 MHz operation is at most 3.4 µs. This is not a concern when using I^2C mode because 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 its mechanical and electronic systems simultaneously. When the self-test function is enabled (via the SELF_TEST bit in the DATA_FORMAT register, Address 0x31), 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. This effect is shown in Figure 43. The scale factors shown in Table 14 can be used to adjust the expected self-test output limits for different supply voltages, V_S. The self-test feature of the ADXL345 also exhibits a bimodal behavior. However, the limits shown in Table 1 and Table 15 to Table 18 are valid for both potential self-test values due to bimodality. Use of the self-test feature at data rates less than 100 Hz or at 1600 Hz may yield values outside these limits. Therefore, the part must be in normal power operation (LOW_POWER bit = 0 in BW_RATE register, Address 0x2C) and be placed into a data rate of 100 Hz through 800 Hz or 3200 Hz for the self-test function to operate correctly.

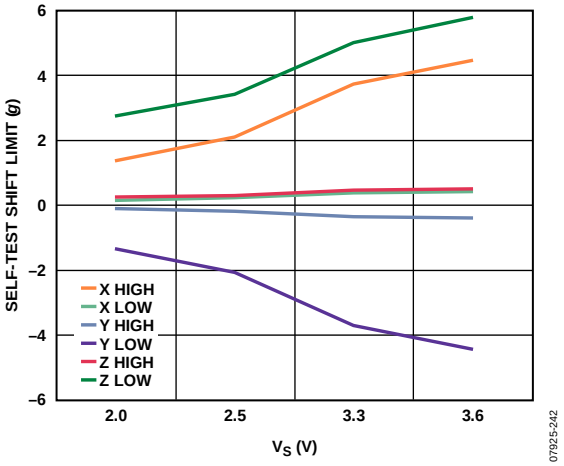


Figure 43. Self-Test Output Change Limits vs. Supply Voltage

Table 14. Self-Test Output Scale Factors for Different Supply Voltages, V_S

Supply Voltage, V_S (V)	X-Axis, Y-Axis	Z-Axis
2.00	0.64	0.8
2.50	1.00	1.00
3.30	1.77	1.47
3.60	2.11	1.69

Table 15. Self-Test Output in LSB for ±2 *g*, 10-Bit or Full Resolution (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Axis	Min	Max	Unit
X	50	540	LSB
Y	−540	−50	LSB
Z	75	875	LSB

Table 16. Self-Test Output in LSB for ±4 *g*, 10-Bit Resolution (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Axis	Min	Max	Unit
X	25	270	LSB
Y	−270	−25	LSB
Z	38	438	LSB

Table 17. Self-Test Output in LSB for ±8 *g*, 10-Bit Resolution (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Axis	Min	Max	Unit
X	12	135	LSB
Y	−135	−12	LSB
Z	19	219	LSB

Table 18. Self-Test Output in LSB for ±16 *g*, 10-Bit Resolution (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Axis	Min	Max	Unit
X	6	67	LSB
Y	−67	−6	LSB
Z	10	110	LSB

REGISTER MAP

Table 19.

Address					
Hex	**Dec**	**Name**	**Type**	**Reset Value**	**Description**
0x00	0	DEVID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/$\overline{W}$	00000000	Tap threshold
0x1E	30	OFSX	R/$\overline{W}$	00000000	X-axis offset
0x1F	31	OFSY	R/$\overline{W}$	00000000	Y-axis offset
0x20	32	OFSZ	R/$\overline{W}$	00000000	Z-axis offset
0x21	33	DUR	R/$\overline{W}$	00000000	Tap duration
0x22	34	Latent	R/$\overline{W}$	00000000	Tap latency
0x23	35	Window	R/$\overline{W}$	00000000	Tap window
0x24	36	THRESH_ACT	R/$\overline{W}$	00000000	Activity threshold
0x25	37	THRESH_INACT	R/$\overline{W}$	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/$\overline{W}$	00000000	Inactivity time
0x27	39	ACT_INACT_CTL	R/$\overline{W}$	00000000	Axis enable control for activity and inactivity detection
0x28	40	THRESH_FF	R/$\overline{W}$	00000000	Free-fall threshold
0x29	41	TIME_FF	R/$\overline{W}$	00000000	Free-fall time
0x2A	42	TAP_AXES	R/$\overline{W}$	00000000	Axis control for single tap/double tap
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of single tap/double tap
0x2C	44	BW_RATE	R/$\overline{W}$	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/$\overline{W}$	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/$\overline{W}$	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/$\overline{W}$	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupts
0x31	49	DATA_FORMAT	R/$\overline{W}$	00000000	Data format control
0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1
0x38	56	FIFO_CTL	R/$\overline{W}$	00000000	FIFO control
0x39	57	FIFO_STATUS	R	00000000	FIFO status

REGISTER DEFINITIONS

Register 0x00—DEVID (Read Only)

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

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

Register 0x1D—THRESH_TAP (Read/Write)

The THRESH_TAP register is eight bits and holds the threshold value for tap interrupts. The data format is unsigned, therefore, the magnitude of the tap event is compared with the value in THRESH_TAP for normal tap detection. The scale factor is 62.5 mg/LSB (that is, 0xFF = 16 *g*). A value of 0 may result in undesirable behavior if single tap/double tap interrupts are enabled.

Register 0x1E, Register 0x1F, Register 0x20—OFSX, OFSY, OFSZ (Read/Write)

The OFSX, OFSY, and OFSZ registers are each eight bits and offer user-set offset adjustments in twos complement format with a scale factor of 15.6 mg/LSB (that is, 0x7F = 2 *g*). The value stored in the offset registers is automatically added to the acceleration data, and the resulting value is stored in the output data registers. For additional information regarding offset calibration and the use of the offset registers, refer to the Offset Calibration section.

Register 0x21—DUR (Read/Write)

The DUR register is eight bits and contains 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 value of 0 disables the single tap/double tap functions.

Register 0x22—Latent (Read/Write)

The latent register is eight bits and contains an unsigned time value representing the wait time from the detection of a tap event to the start of the time window (defined by the window register) during which a possible second tap event can be detected. The scale factor is 1.25 ms/LSB. A value of 0 disables the double tap function.

Register 0x23—Window (Read/Write)

The window register is eight bits and contains an unsigned time value representing the amount of time after the expiration of the latency time (determined by the latent register) during which a second valid tap can begin. The scale factor is 1.25 ms/LSB. A value of 0 disables the double tap function.

Register 0x24—THRESH_ACT (Read/Write)

The THRESH_ACT register is eight bits and holds the threshold value for detecting activity. The data format is unsigned, so the magnitude of the activity event is compared with the value in the THRESH_ACT register. The scale factor is 62.5 mg/LSB. A value of 0 may result in undesirable behavior if the activity interrupt is enabled.

Register 0x25—THRESH_INACT (Read/Write)

The THRESH_INACT register is eight bits and holds the threshold value for detecting inactivity. The data format is unsigned, so the magnitude of the inactivity event is compared with the value in the THRESH_INACT register. The scale factor is 62.5 mg/LSB. A value of 0 may result in undesirable behavior if the inactivity interrupt is enabled.

Register 0x26—TIME_INACT (Read/Write)

The TIME_INACT register is eight bits and contains an unsigned time value representing the amount of time that acceleration must be less than the value in the THRESH_INACT register for inactivity to be declared. The scale factor is 1 sec/LSB. Unlike the other interrupt functions, which use unfiltered data (see the Threshold section), the inactivity function uses filtered output data. At least one output sample must be generated for the inactivity interrupt to be triggered. This results in the function appearing unresponsive if the TIME_INACT register is set to a value less than the time constant of the output data rate. A value of 0 results in an interrupt when the output data is less than the value in the THRESH_INACT register.

Register 0x27—ACT_INACT_CTL (Read/Write)

D7	D6	D5	D4
ACT ac/dc	ACT_X enable	ACT_Y enable	ACT_Z enable
D3	**D2**	**D1**	**D0**
INACT ac/dc	INACT_X enable	INACT_Y enable	INACT_Z enable

ACT AC/DC and INACT AC/DC Bits

A setting of 0 selects dc-coupled operation, and a setting of 1 enables ac-coupled operation. In dc-coupled operation, the current acceleration magnitude is compared directly with THRESH_ACT and THRESH_INACT 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 the THRESH_ACT value, the device triggers an activity interrupt.

Similarly, in ac-coupled operation for inactivity detection, a reference value is used for comparison and is updated whenever the device exceeds the inactivity threshold. After 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 less than the value in THRESH_INACT for the time in TIME_INACT, the device is considered inactive and the inactivity interrupt is triggered.

ACT_x Enable Bits and INACT_x Enable Bits

A setting of 1 enables x-, y-, or z-axis participation in detecting activity or inactivity. A setting of 0 excludes the selected axis from participation. If all axes are excluded, the function is disabled. For activity detection, all participating axes are logically OR'ed, causing the activity function to trigger when any of the participating axes exceeds the threshold. For inactivity detection, all participating axes are logically AND'ed, causing the inactivity function to trigger only if all participating axes are below the threshold for the specified time.

Register 0x28—THRESH_FF (Read/Write)

The THRESH_FF register is eight bits and holds the threshold value, in unsigned format, for free-fall detection. The acceleration on all axes is compared with the value in THRESH_FF to determine if a free-fall event occurred. The scale factor is 62.5 mg/LSB. Note that a value of 0 mg may result in undesirable behavior if the free-fall interrupt is enabled. Values between 300 mg and 600 mg (0x05 to 0x09) are recommended.

Register 0x29—TIME_FF (Read/Write)

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

Register 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

Suppress Bit

Setting the suppress bit suppresses double tap detection if acceleration greater than the value in THRESH_TAP is present between taps. See the Tap Detection section for more details.

TAP_x Enable Bits

A setting of 1 in the TAP_X enable, TAP_Y enable, or TAP_Z enable bit enables x-, y-, or z-axis participation in tap detection. A setting of 0 excludes the selected axis from participation in tap detection.

Register 0x2B—ACT_TAP_STATUS (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
0	ACT_X source	ACT_Y source	ACT_Z source	Asleep	TAP_X source	TAP_Y source	TAP_Z source

ACT_x Source and TAP_x Source Bits

These bits indicate the first axis involved in a tap or activity event. A setting of 1 corresponds to involvement in the event, and a setting of 0 corresponds to no involvement. When new data is available, these bits are not cleared but are overwritten by the new data. The ACT_TAP_STATUS register should be read before clearing the interrupt. Disabling an axis from participation clears the corresponding source bit when the next activity or single tap/double tap event occurs.

Asleep Bit

A setting of 1 in the asleep bit indicates that the part is asleep, and a setting of 0 indicates that the part is not asleep. This bit toggles only if the device is configured for auto sleep. See the AUTO_SLEEP Bit section for more information on autosleep mode.

Register 0x2C—BW_RATE (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	LOW_POWER	Rate			

LOW_POWER Bit

A setting of 0 in the LOW_POWER bit selects normal operation, and a setting of 1 selects reduced power operation, which has somewhat higher noise (see the Power Modes section for details).

Rate Bits

These bits select the device bandwidth and output data rate (see Table 7 and Table 8 for details). The default value is 0x0A, which translates to a 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 results in samples being discarded.

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

Link Bit

A setting of 1 in the link bit with both the activity and inactivity functions enabled delays the start of the activity function until inactivity is detected. After activity is detected, inactivity detection begins, preventing the detection of activity. This bit serially links the activity and inactivity functions. When this bit is set to 0, the inactivity and activity functions are concurrent. Additional information can be found in the Link Mode section.

When clearing the link bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the link bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

AUTO_SLEEP Bit

If the link bit is set, a setting of 1 in the AUTO_SLEEP bit enables the auto-sleep functionality. In this mode, the ADXL345 automatically switches to sleep mode if the inactivity function is enabled and inactivity is detected (that is, when acceleration is below the THRESH_INACT value for at least the time indicated by TIME_INACT). If activity is also enabled, the ADXL345 automatically wakes up from sleep after detecting activity and returns to operation at the output data rate set in the BW_RATE register. A setting of 0 in the AUTO_SLEEP bit disables automatic switching to sleep mode. See the description of the Sleep Bit in this section for more information on sleep mode.

If the link bit is not set, the AUTO_SLEEP feature is disabled and setting the AUTO_SLEEP bit does not have an impact on device operation. Refer to the Link Bit section or the Link Mode section for more information on utilization of the link feature.

When clearing the AUTO_SLEEP bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the AUTO_SLEEP bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

Measure Bit

A setting of 0 in the measure bit places the part into standby mode, and a setting of 1 places the part into measurement mode. The ADXL345 powers up in standby mode with minimum power consumption.

Sleep Bit

A setting of 0 in the sleep bit puts the part into the normal mode of operation, and a setting of 1 places the part into sleep mode. Sleep mode suppresses DATA_READY, stops transmission of data to FIFO, and switches the sampling rate to one specified by the wakeup bits. In sleep mode, only the activity function can be used. When the DATA_READY interrupt is suppressed, the output data registers (Register 0x32 to Register 0x37) are still updated at the sampling rate set by the wakeup bits (D1:D0).

When clearing the sleep bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the sleep bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

Wakeup Bits

These bits control the frequency of readings in sleep mode as described in Table 20.

Table 20. Frequency of Readings in Sleep Mode

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

Register 0x2E—INT_ENABLE (Read/Write)

D7 DATA_READY	**D6** SINGLE_TAP	**D5** DOUBLE_TAP	**D4** Activity
D3 Inactivity	**D2** FREE_FALL	**D1** Watermark	**D0** Overrun

Setting bits in this register to a value of 1 enables their respective functions to generate interrupts, whereas a value of 0 prevents the functions from generating interrupts. The DATA_READY, watermark, and overrun bits enable only the interrupt output; the functions are always enabled. It is recommended that interrupts be configured before enabling their outputs.

Register 0x2F—INT_MAP (R/$\overline{W}$)

D7 DATA_READY	**D6** SINGLE_TAP	**D5** DOUBLE_TAP	**D4** Activity
D3 Inactivity	**D2** FREE_FALL	**D1** Watermark	**D0** Overrun

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

Register 0x30—INT_SOURCE (Read Only)

D7 DATA_READY	**D6** SINGLE_TAP	**D5** DOUBLE_TAP	**D4** Activity
D3 Inactivity	**D2** FREE_FALL	**D1** Watermark	**D0** Overrun

Bits set to 1 in this register indicate that their respective functions have triggered an event, whereas a value of 0 indicates that the corresponding event has not occurred. The DATA_READY, watermark, and overrun bits are always set if the corresponding events occur, regardless of the INT_ENABLE register settings, and are cleared by reading data from the DATAX, DATAY, and DATAZ registers. The DATA_READY and watermark bits may require multiple reads, as indicated in the FIFO mode descriptions in the FIFO section. Other bits, and the corresponding interrupts, are cleared by reading the INT_SOURCE register.

Register 0x31—DATA_FORMAT (Read/Write)

D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

The DATA_FORMAT register controls the presentation of data to Register 0x32 through Register 0x37. All data, except that for the ±16 *g* range, must be clipped to avoid rollover.

SELF_TEST Bit

A setting of 1 in the SELF_TEST bit applies a self-test force to the sensor, causing a shift in the output data. A value of 0 disables the self-test force.

SPI Bit

A value of 1 in the SPI bit sets the device to 3-wire SPI mode, and a value of 0 sets the device to 4-wire SPI mode.

INT_INVERT Bit

A value of 0 in the INT_INVERT bit sets the interrupts to active high, and a value of 1 sets the interrupts to active low.

FULL_RES Bit

When this bit is set to a value of 1, the device is in full resolution mode, where the output resolution increases with the *g* range set by the range bits to maintain a 4 mg/LSB scale factor. When the FULL_RES bit is set to 0, the device is in 10-bit mode, and the range bits determine the maximum *g* range and scale factor.

Justify Bit

A setting of 1 in the justify bit selects left-justified (MSB) mode, and a setting of 0 selects right-justified mode with sign extension.

Range Bits

These bits set the *g* range as described in Table 21.

Table 21. *g* Range Setting

Setting		
D1	**D0**	***g* Range**
0	0	±2 *g*
0	1	±4 *g*
1	0	±8 *g*
1	1	±16 *g*

Register 0x32 to Register 0x37—DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0, DATAZ1 (Read Only)

These six bytes (Register 0x32 to Register 0x37) are eight bits each and hold the output data for each axis. Register 0x32 and Register 0x33 hold the output data for the x-axis, Register 0x34 and Register 0x35 hold the output data for the y-axis, and Register 0x36 and Register 0x37 hold the output data for the z-axis. The output data is twos complement, with DATAx0 as the least significant byte and DATAx1 as the most significant byte, where x represent X, Y, or Z. The DATA_FORMAT register (Address 0x31) controls the format of the data. It is recommended that a multiple-byte read of all registers be performed to prevent a change in data between reads of sequential registers.

Register 0x38—FIFO_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
FIFO_MODE		Trigger	Samples				

FIFO_MODE Bits

These bits set the FIFO mode, as described in Table 22.

Table 22. FIFO Modes

Setting			
D7	**D6**	**Mode**	**Function**
0	0	Bypass	FIFO is bypassed.
0	1	FIFO	FIFO collects up to 32 values and then stops collecting data, collecting new data only when FIFO is not full.
1	0	Stream	FIFO holds the last 32 data values. When FIFO is full, the oldest data is overwritten with newer data.
1	1	Trigger	When triggered by the trigger bit, FIFO holds the last data samples before the trigger event and then continues to collect data until full. New data is collected only when FIFO is not full.

Trigger Bit

A value of 0 in the trigger bit links the trigger event of trigger mode to INT1, and a value of 1 links the trigger event to INT2.

Samples Bits

The function of these bits depends on the FIFO mode selected (see Table 23). Entering a value of 0 in the samples bits immediately sets the watermark status bit in the INT_SOURCE register, regardless of which FIFO mode is selected. Undesirable operation may occur if a value of 0 is used for the samples bits when trigger mode is used.

Table 23. Samples Bits Functions

FIFO Mode	Samples Bits Function
Bypass	None.
FIFO	Specifies how many FIFO entries are needed to trigger a watermark interrupt.
Stream	Specifies how many FIFO entries are needed to trigger a watermark interrupt.
Trigger	Specifies how many FIFO samples are retained in the FIFO buffer before a trigger event.

Register 0x39—FIFO_STATUS (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
FIFO_TRIG	0	Entries					

FIFO_TRIG Bit

A 1 in the FIFO_TRIG bit corresponds to a trigger event occurring, and a 0 means that a FIFO trigger event has not occurred.

Entries Bits

These bits report how many data values are stored in FIFO. Access to collect the data from FIFO is provided through the DATAX, DATAY, and DATAZ registers. FIFO reads must be done in burst or multiple-byte mode because each FIFO level is cleared after any read (single- or multiple-byte) of FIFO. FIFO stores a maximum of 32 entries, which equates to a maximum of 33 entries available at any given time because an additional entry is available at the output filter of the device.

APPLICATIONS INFORMATION

POWER SUPPLY DECOUPLING

A 1 µF tantalum capacitor (C_S) at V_S and a 0.1 µF ceramic capacitor ($C_{I/O}$) at $V_{DD\ I/O}$ placed close to the ADXL345 supply pins is recommended to adequately decouple the accelerometer from noise on the power supply. If additional decoupling is necessary, a resistor or ferrite bead, no larger than 100 Ω, in series with V_S may be helpful. Additionally, increasing the bypass capacitance on V_S to a 10 µF tantalum capacitor in parallel with a 0.1 µF ceramic capacitor may also improve noise.

Care should be taken to ensure that the connection from the ADXL345 ground to the power supply ground has low impedance because noise transmitted through ground has an effect similar to noise transmitted through V_S. It is recommended that V_S and $V_{DD\ I/O}$ be separate supplies to minimize digital clocking noise on the V_S supply. If this is not possible, additional filtering of the supplies, as previously mentioned, may be necessary.

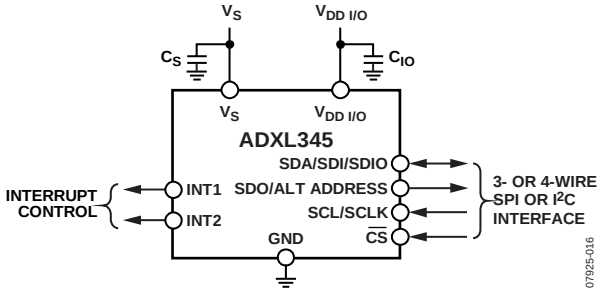


Figure 44. Application Diagram

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, as shown in Figure 45, may result in large, apparent measurement errors due to undampened PCB vibration. Locating the accelerometer near a hard mounting point ensures that any PCB vibration at the accelerometer is above the accelerometer's mechanical sensor resonant frequency and, therefore, effectively invisible to the accelerometer. Multiple mounting points, close to the sensor, and/or a thicker PCB also help to reduce the effect of system resonance on the performance of the sensor.

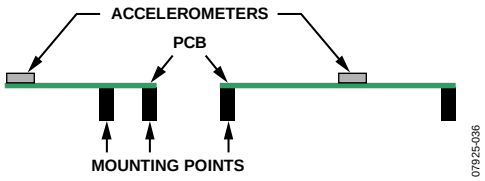


Figure 45. Incorrectly Placed Accelerometers

TAP DETECTION

The tap interrupt function is capable of detecting either single or double taps. The following parameters are shown in Figure 46 for a valid single and valid double tap event:

- The tap detection threshold is defined by the THRESH_TAP register (Address 0x1D).
- The maximum tap duration time is defined by the DUR register (Address 0x21).
- The tap latency time is defined by the latent register (Address 0x22) and is the waiting period from the end of the first tap until the start of the time window, when a second tap can be detected, which is determined by the value in the window register (Address 0x23).
- The interval after the latency time (set by the latent register) is defined by the window register. Although a second tap must begin after the latency time has expired, it need not finish before the end of the time defined by the window register.

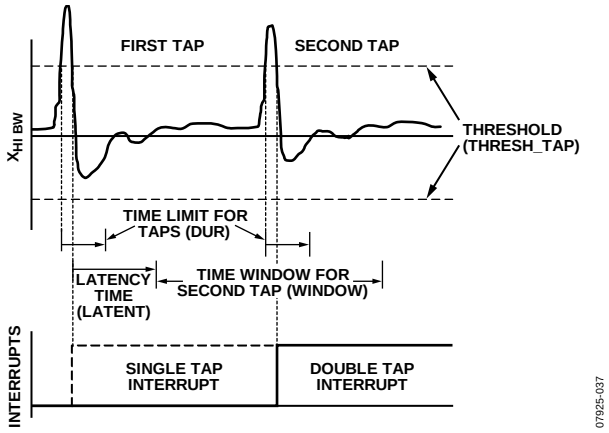


Figure 46. Tap Interrupt Function with Valid Single and Double Taps

If only the single tap function is in use, the single tap interrupt is triggered when the acceleration goes below the threshold, as long as DUR has not been exceeded. If both single and double tap functions are in use, the single tap interrupt is triggered when 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 (Address 0x2A) is set, any acceleration spike above the threshold during the latency time (set by the latent register) invalidates the double tap detection, as shown in Figure 47.

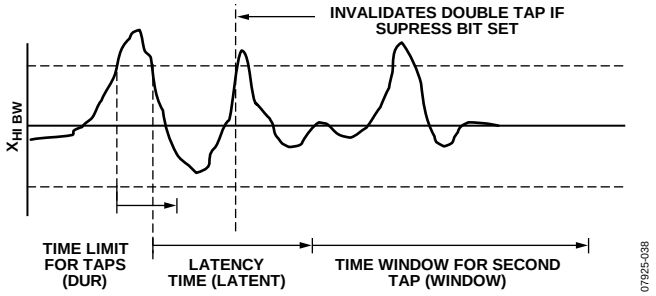


Figure 47. Double Tap Event Invalid Due to High g Event When the Suppress Bit Is Set

A double tap event can also be invalidated if acceleration above the threshold is detected at the start of the time window for the second tap (set by the window register). This results in an invalid double tap at the start of this window, as shown in Figure 48. Additionally, a double tap event can be invalidated if an acceleration exceeds the time limit for taps (set by the DUR register), resulting in an invalid double tap at the end of the DUR time limit for the second tap event, also shown in Figure 48.

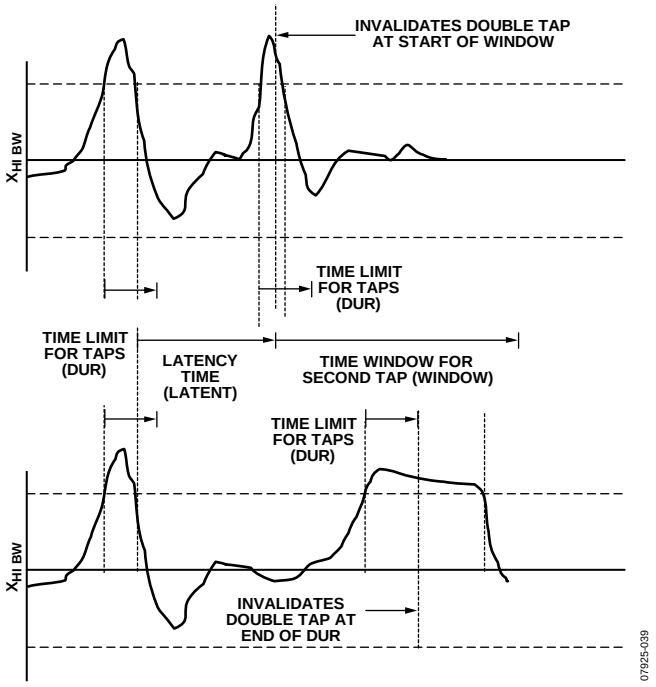


Figure 48. Tap Interrupt Function with Invalid Double Taps

Single taps, double taps, or both can be detected by setting the respective bits in the INT_ENABLE register (Address 0x2E). Control over participation of each of the three axes in single tap/double tap detection is exerted by setting the appropriate bits in the TAP_AXES register (Address 0x2A). For the double tap function to operate, both the latent and window registers must be set to a nonzero value.

Every mechanical system has somewhat different single tap/double tap responses based on the mechanical characteristics of the system. Therefore, some experimentation with values for the DUR, latent, window, and THRESH_TAP registers is required. In general, a good starting point is to set the DUR register to a value greater than 0x10 (10 ms), the latent register to a value greater than 0x10 (20 ms), the window register to a value greater than 0x40 (80 ms), and the THRESH_TAP register to a value greater than 0x30 (3 *g*). Setting a very low value in the latent, window, or THRESH_TAP register may result in an unpredictable response due to the accelerometer picking up echoes of the tap inputs.

After a tap interrupt has been received, the first axis to exceed the THRESH_TAP level is reported in the ACT_TAP_STATUS register (Address 0x2B). This register is never cleared but is overwritten with new data.

THRESHOLD

The lower output data rates are achieved by decimating a common sampling frequency inside the device. The activity, free-fall, and single tap/double tap detection functions without improved tap enabled are performed using undecimated data. Because the bandwidth of the output data varies with the data rate and is lower than the bandwidth of the undecimated data, the high frequency and high *g* data that is used to determine activity, free-fall, and single tap/double tap events may not be present if the output of the accelerometer is examined. This may result in functions triggering when acceleration data does not appear to meet the conditions set by the user for the corresponding function.

LINK MODE

The function of the link bit is to reduce the number of activity interrupts that the processor must service by setting the device to look for activity only after inactivity. For proper operation of this feature, the processor must still respond to the activity and inactivity interrupts by reading the INT_SOURCE register (Address 0x30) and, therefore, clearing the interrupts. If an activity interrupt is not cleared, the part cannot go into autosleep mode. The asleep bit in the ACT_TAP_STATUS register (Address 0x2B) indicates if the part is asleep.

SLEEP MODE VS. LOW POWER MODE

In applications where a low data rate and low power consumption is desired (at the expense of noise performance), it is recommended that low power mode be used. The use of low power mode preserves the functionality of the DATA_READY interrupt and the FIFO for postprocessing of the acceleration data. Sleep mode, while offering a low data rate and power consumption, is not intended for data acquisition.

However, when sleep mode is used in conjunction with the AUTO_SLEEP mode and the link mode, the part can automatically switch to a low power, low sampling rate mode when inactivity is detected. To prevent the generation of redundant inactivity interrupts, the inactivity interrupt is automatically disabled and activity is enabled. When the ADXL345 is in sleep mode, the host processor can also be placed into sleep mode or low power mode to save significant system power. When activity is detected, the accelerometer automatically switches back to the original data rate of the application and provides an activity interrupt that can be used to wake up the host processor. Similar to when inactivity occurs, detection of activity events is disabled and inactivity is enabled.

OFFSET CALIBRATION

Accelerometers are mechanical structures containing elements that are free to move. These moving parts can be very sensitive to mechanical stresses, much more so than solid-state electronics. The 0 *g* bias or offset is an important accelerometer metric because it defines the baseline for measuring acceleration. Additional stresses can be applied during assembly of a system containing an accelerometer. These stresses can come from, but are not limited to, component soldering, board stress during mounting, and application of any compounds on or over the component. If calibration is deemed necessary, it is recommended that calibration be performed after system assembly to compensate for these effects.

A simple method of calibration is to measure the offset while assuming that the sensitivity of the ADXL345 is as specified in Table 1. The offset can then be automatically accounted for by using the built-in offset registers. This results in the data acquired from the DATA registers already compensating for any offset.

In a no-turn or single-point calibration scheme, the part is oriented such that one axis, typically the z-axis, is in the 1 *g* field of gravity and the remaining axes, typically the x- and y-axis, are in a 0 *g* field. The output is then measured by taking the average of a series of samples. The number of samples averaged is a choice of the system designer, but a recommended starting point is 0.1 sec worth of data for data rates of 100 Hz or greater. This corresponds to 10 samples at the 100 Hz data rate. For data rates less than 100 Hz, it is recommended that at least 10 samples be averaged together. These values are stored as X_{0g}, Y_{0g}, and Z_{+1g} for the 0 *g* measurements on the x- and y-axis and the 1 *g* measurement on the z-axis, respectively.

The values measured for X_{0g} and Y_{0g} correspond to the x- and y-axis offset, and compensation is done by subtracting those values from the output of the accelerometer to obtain the actual acceleration:

$$X_{ACTUAL} = X_{MEAS} - X_{0g}$$

$$Y_{ACTUAL} = Y_{MEAS} - Y_{0g}$$

Because the z-axis measurement was done in a +1 *g* field, a no-turn or single-point calibration scheme assumes an ideal sensitivity, S_Z for the z-axis. This is subtracted from Z_{+1g} to attain the z-axis offset, which is then subtracted from future measured values to obtain the actual value:

$$Z_{0g} = Z_{+1g} - S_Z$$

$$Z_{ACTUAL} = Z_{MEAS} - Z_{0g}$$

The ADXL345 can automatically compensate the output for offset by using the offset registers (Register 0x1E, Register 0x1F, and Register 0x20). These registers contain an 8-bit, twos complement value that is automatically added to all measured acceleration values, and the result is then placed into the DATA registers. Because the value placed in an offset register is additive, a negative value is placed into the register to eliminate a positive offset and vice versa for a negative offset. The register has a scale factor of 15.6 mg/LSB and is independent of the selected *g*-range.

As an example, assume that the ADXL345 is placed into full-resolution mode with a sensitivity of typically 256 LSB/*g*. The part is oriented such that the z-axis is in the field of gravity and x-, y-, and z-axis outputs are measured as +10 LSB, −13 LSB, and +9 LSB, respectively. Using the previous equations, X_{0g} is +10 LSB, Y_{0g} is −13 LSB, and Z_{0g} is +9 LSB. Each LSB of output in full-resolution is 3.9 mg or one-quarter of an LSB of the offset register. Because the offset register is additive, the 0 *g* values are negated and rounded to the nearest LSB of the offset register:

$$X_{OFFSET} = -Round(10/4) = -3 \text{ LSB}$$

$$Y_{OFFSET} = -Round(-13/4) = 3 \text{ LSB}$$

$$Z_{OFFSET} = -Round(9/4) = -2 \text{ LSB}$$

These values are programmed into the OFSX, OFSY, and OFXZ registers, respectively, as 0xFD, 0x03 and 0xFE. As with all registers in the ADXL345, the offset registers do not retain the value written into them when power is removed from the part. Power-cycling the ADXL345 returns the offset registers to their default value of 0x00.

Because the no-turn or single-point calibration method assumes an ideal sensitivity in the z-axis, any error in the sensitivity results in offset error. For instance, if the actual sensitivity was 250 LSB/*g* in the previous example, the offset would be 15 LSB, not 9 LSB. To help minimize this error, an additional measurement point can be used with the z-axis in a 0 *g* field and the 0 *g* measurement can be used in the Z_{ACTUAL} equation.

USING SELF-TEST

The self-test change is defined as the difference between the acceleration output of an axis with self-test enabled and the acceleration output of the same axis with self-test disabled (see Endnote 4 of Table 1). This definition assumes that the sensor does not move between these two measurements, because if the sensor moves, a non–self-test related shift corrupts the test.

Proper configuration of the ADXL345 is also necessary for an accurate self-test measurement. The part should be set with a data rate of 100 Hz through 800 Hz, or 3200 Hz. This is done by ensuring that a value of 0x0A through 0x0D, or 0x0F is written into the rate bits (Bit D3 through Bit D0) in the BW_RATE register (Address 0x2C). The part also must be placed into normal power operation by ensuring the LOW_POWER bit in the BW_RATE register is cleared (LOW_POWER bit = 0) for accurate self-test measurements. It is recommended that the part be set to full-resolution, 16 *g* mode to ensure that there is sufficient dynamic range for the entire self-test shift. This is done by setting Bit D3 of the DATA_FORMAT register (Address 0x31) and writing a value of 0x03 to the range bits (Bit D1 and Bit D0) of the DATA_FORMAT register (Address 0x31). This results in a high dynamic range for measurement and a 3.9 m*g*/LSB scale factor.

After the part is configured for accurate self-test measurement, several samples of x-, y-, and z-axis acceleration data should be retrieved from the sensor and averaged together. The number of samples averaged is a choice of the system designer, but a recommended starting point is 0.1 sec worth of data for data rates of 100 Hz or greater. This corresponds to 10 samples at the 100 Hz data rate. For data rates less than 100 Hz, it is recommended that at least 10 samples be averaged together. The averaged values should be stored and labeled appropriately as the self-test disabled data, that is, X_{ST_OFF}, Y_{ST_OFF}, and Z_{ST_OFF}.

Next, self-test should be enabled by setting Bit D7 (SELF_TEST) of the DATA_FORMAT register (Address 0x31). The output needs some time (about four samples) to settle after enabling self-test. After allowing the output to settle, several samples of the x-, y-, and z-axis acceleration data should be taken again and averaged. It is recommended that the same number of samples be taken for this average as was previously taken. These averaged values should again be stored and labeled appropriately as the value with self-test enabled, that is, X_{ST_ON}, Y_{ST_ON}, and Z_{ST_ON}. Self-test can then be disabled by clearing Bit D7 (SELF_TEST) of the DATA_FORMAT register (Address 0x31).

With the stored values for self-test enabled and disabled, the self-test change is as follows:

$$X_{ST} = X_{ST\_ON} - X_{ST\_OFF}$$

$$Y_{ST} = Y_{ST\_ON} - Y_{ST\_OFF}$$

$$Z_{ST} = Z_{ST\_ON} - Z_{ST\_OFF}$$

Because the measured output for each axis is expressed in LSBs, X_{ST}, Y_{ST}, and Z_{ST} are also expressed in LSBs. These values can be converted to *g*'s of acceleration by multiplying each value by the 3.9 m*g*/LSB scale factor, if configured for full-resolution mode. Additionally, Table 15 through Table 18 correspond to the self-test range converted to LSBs and can be compared with the measured self-test change when operating at a V_S of 2.5 V. For other voltages, the minimum and maximum self-test output values should be adjusted based on (multiplied by) the scale factors shown in Table 14. If the part was placed into ±2 *g*, 10-bit or full-resolution mode, the values listed in Table 15 should be used. Although the fixed 10-bit mode or a range other than 16 *g* can be used, a different set of values, as indicated in Table 16 through Table 18, would need to be used. Using a range below 8 *g* may result in insufficient dynamic range and should be considered when selecting the range of operation for measuring self-test.

If the self-test change is within the valid range, the test is considered successful. Generally, a part is considered to pass if the minimum magnitude of change is achieved. However, a part that changes by more than the maximum magnitude is not necessarily a failure.

Another effective method for using the self-test to verify accelerometer functionality is to toggle the self test at a certain rate and then perform an FFT on the output. The FFT should have a corresponding tone at the frequency the self-test was toggled. Using an FFT like this removes the dependency of the test on supply voltage and on self-test magnitude, which can vary within a rather wide range.

DATA FORMATTING OF UPPER DATA RATES

Formatting of output data at the 3200 Hz and 1600 Hz output data rates changes depending on the mode of operation (full-resolution or fixed 10-bit) and the selected output range.

When using the 3200 Hz or 1600 Hz output data rates in full-resolution or ±2 g, 10-bit operation, the LSB of the output data-word is always 0. When data is right justified, this corresponds to Bit D0 of the DATAx0 register, as shown in Figure 49. When data is left justified and the part is operating in ±2 g, 10-bit mode, the LSB of the output data-word is Bit D6 of the DATAx0 register. In full-resolution operation when data is left justified, the location of the LSB changes according to the selected output range.

For a range of ±2 g, the LSB is Bit D6 of the DATAx0 register; for ±4 g, Bit D5 of the DATAx0 register; for ±8 g, Bit D4 of the DATAx0 register; and for ±16 g, Bit D3 of the DATAx0 register. This is shown in Figure 50.

The use of 3200 Hz and 1600 Hz output data rates for fixed 10-bit operation in the ±4 g, ±8 g, and ±16 g output ranges provides an LSB that is valid and that changes according to the applied acceleration. Therefore, in these modes of operation, Bit D0 is not always 0 when output data is right justified and Bit D6 is not always 0 when output data is left justified. Operation at any data rate of 800 Hz or lower also provides a valid LSB in all ranges and modes that changes according to the applied acceleration.

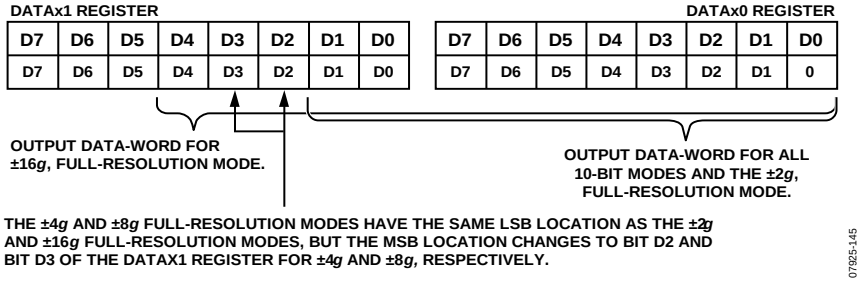


Figure 49. Data Formatting of Full-Resolution and ±2 g, 10-Bit Modes of Operation When Output Data Is Right Justified

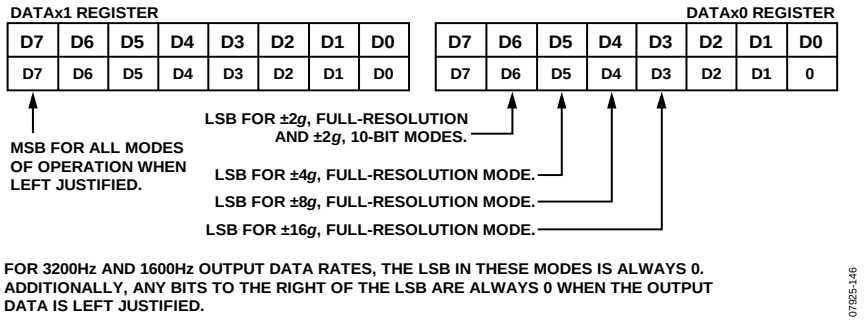


Figure 50. Data Formatting of Full-Resolution and ±2 g, 10-Bit Modes of Operation When Output Data Is Left Justified

NOISE PERFORMANCE

The specification of noise shown in Table 1 corresponds to the typical noise performance of the ADXL345 in normal power operation with an output data rate of 100 Hz (LOW_POWER bit (D4) = 0, rate bits (D3:D0) = 0xA in the BW_RATE register, Address 0x2C). For normal power operation at data rates below 100 Hz, the noise of the ADXL345 is equivalent to the noise at 100 Hz ODR in LSBs. For data rates greater than 100 Hz, the noise increases roughly by a factor of √2 per doubling of the data rate. For example, at 400 Hz ODR, the noise on the x- and y-axes is typically less than 1.5 LSB rms, and the noise on the z-axis is typically less than 2.2 LSB rms.

For low power operation (LOW_POWER bit (D4) = 1 in the BW_RATE register, Address 0x2C), the noise of the ADXL345 is constant for all valid data rates shown in Table 8. This value is typically less than 1.8 LSB rms for the x- and y-axes and typically less than 2.6LSB rms for the z-axis.

The trend of noise performance for both normal power and low power modes of operation of the ADXL345 is shown in Figure 51.

Figure 52 shows the typical Allan deviation for the ADXL345. The 1/f corner of the device, as shown in this figure, is very low, allowing absolute resolution of approximately 100 µg (assuming that there is sufficient integration time). Figure 52 also shows that the noise density is 290 µg/√Hz for the x-axis and y-axis and 430 µg/√Hz for the z-axis.

Figure 53 shows the typical noise performance trend of the ADXL345 over supply voltage. The performance is normalized to the tested and specified supply voltage, V_S = 2.5 V. In general, noise decreases as supply voltage is increased. It should be noted, as shown in Figure 51, that the noise on the z-axis is typically higher than on the x-axis and y-axis; therefore, while they change roughly the same in percentage over supply voltage, the magnitude of change on the z-axis is greater than the magnitude of change on the x-axis and y-axis.

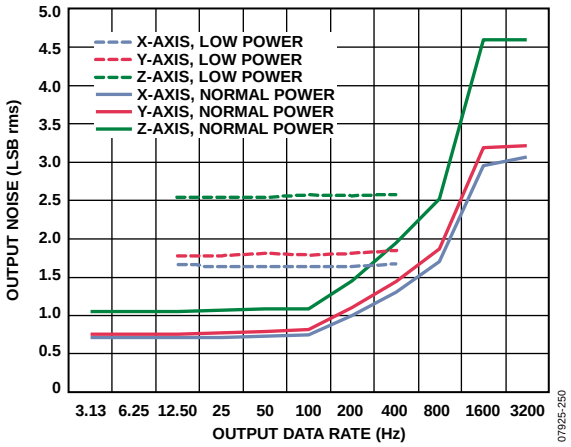


Figure 51. Noise vs. Output Data Rate for Normal and Low Power Modes, Full-Resolution (256 LSB/g)

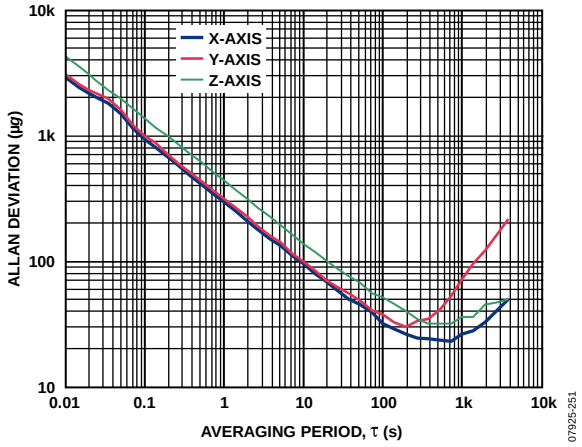


Figure 52. Root Allan Deviation

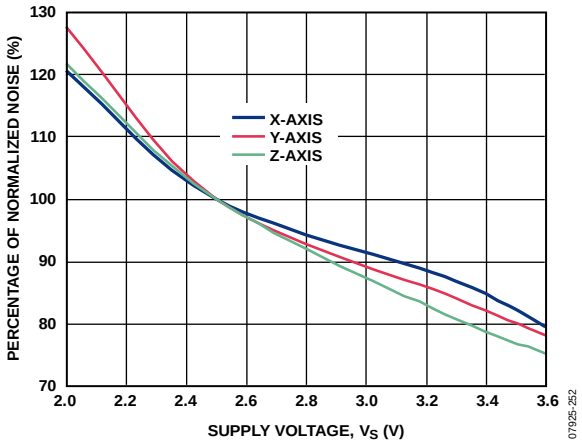


Figure 53. Normalized Noise vs. Supply Voltage, V_S

OPERATION AT VOLTAGES OTHER THAN 2.5 V

The ADXL345 is tested and specified at a supply voltage of V_S = 2.5 V; however, it can be powered with V_S as high as 3.6 V or as low as 2.0 V. Some performance parameters change as the supply voltage changes: offset, sensitivity, noise, self-test, and supply current.

Due to slight changes in the electrostatic forces as supply voltage is varied, the offset and sensitivity change slightly. When operating at a supply voltage of V_S = 3.3 V, the x- and y-axis offset is typically 25 mg higher than at Vs = 2.5 V operation. The z-axis is typically 20 mg lower when operating at a supply voltage of 3.3 V than when operating at V_S = 2.5 V. Sensitivity on the x- and y-axes typically shifts from a nominal 256 LSB/*g* (full-resolution or ±2 *g*, 10-bit operation) at V_S = 2.5 V operation to 265 LSB/*g* when operating with a supply voltage of 3.3 V. The z-axis sensitivity is unaffected by a change in supply voltage and is the same at V_S = 3.3 V operation as it is at V_S = 2.5 V operation. Simple linear interpolation can be used to determine typical shifts in offset and sensitivity at other supply voltages.

Changes in noise performance, self-test response, and supply current are discussed elsewhere throughout the data sheet. For noise performance, the Noise Performance section should be reviewed. The Using Self-Test section discusses both the operation of self-test over voltage, a square relationship with supply voltage, as well as the conversion of the self-test response in *g*'s to LSBs. Finally, Figure 33 shows the impact of supply voltage on typical current consumption at a 100 Hz output data rate, with all other output data rates following the same trend.

OFFSET PERFORMANCE AT LOWEST DATA RATES

The ADXL345 offers a large number of output data rates and bandwidths, designed for a large range of applications. However, at the lowest data rates, described as those data rates below 6.25 Hz, the offset performance over temperature can vary significantly from the remaining data rates. Figure 54, Figure 55, and Figure 56 show the typical offset performance of the ADXL345 over temperature for the data rates of 6.25 Hz and lower. All plots are normalized to the offset at 100 Hz output data rate; therefore, a nonzero value corresponds to additional offset shift due to temperature for that data rate.

When using the lowest data rates, it is recommended that the operating temperature range of the device be limited to provide minimal offset shift across the operating temperature range. Due to variability between parts, it is also recommended that calibration over temperature be performed if any data rates below 6.25 Hz are in use.

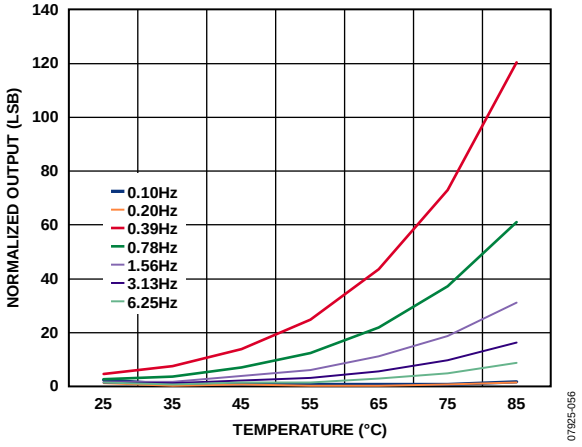


Figure 54. Typical X-Axis Output vs. Temperature at Lower Data Rates, Normalized to 100 Hz Output Data Rate, V_S = 2.5 V

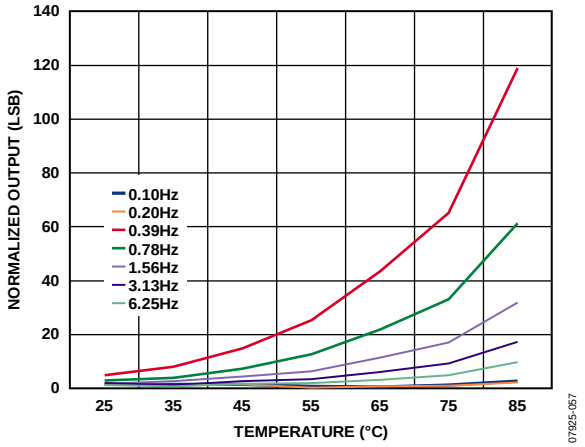


Figure 55. Typical Y-Axis Output vs. Temperature at Lower Data Rates, Normalized to 100 Hz Output Data Rate, V_S = 2.5 V

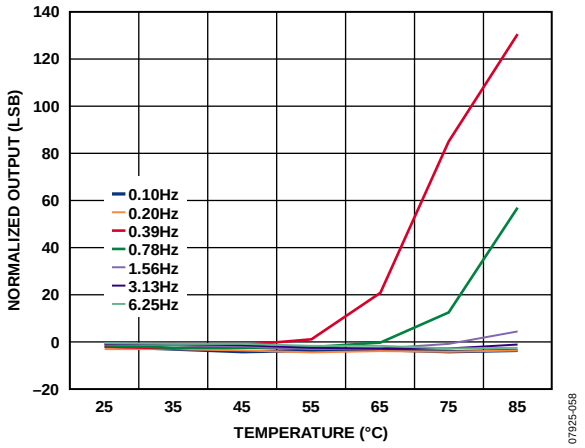


Figure 56. Typical Z-Axis Output vs. Temperature at Lower Data Rates, Normalized to 100 Hz Output Data Rate, V_S = 2.5 V

AXES OF ACCELERATION SENSITIVITY

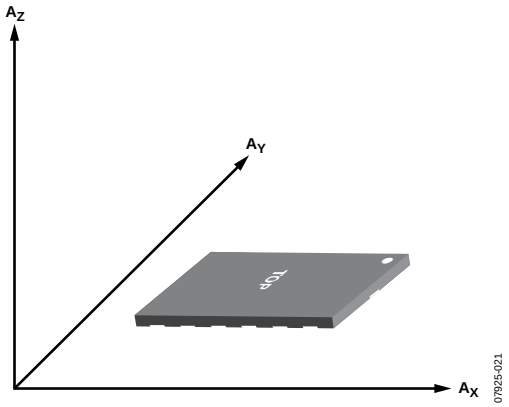


Figure 57. Axes of Acceleration Sensitivity (Corresponding Output Voltage Increases When Accelerated Along the Sensitive Axis)

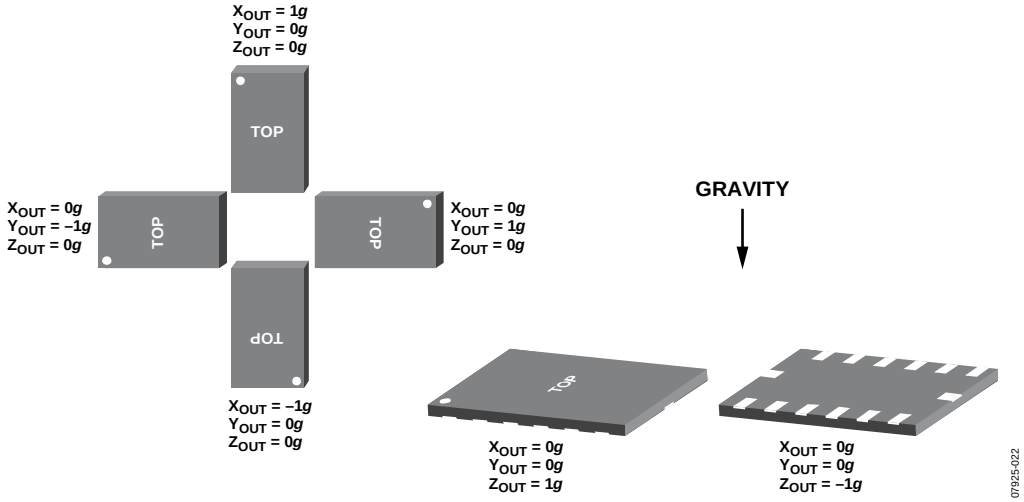


Figure 58. Output Response vs. Orientation to Gravity

LAYOUT AND DESIGN RECOMMENDATIONS

Figure 59 shows the recommended printed wiring board land pattern. Figure 60and Table 24 provide details about the recommended soldering profile.

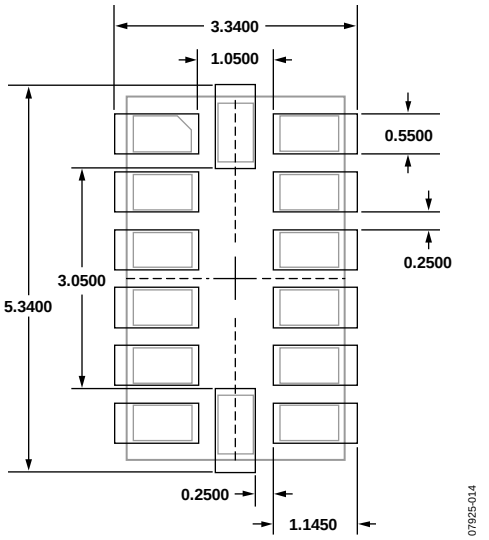


Figure 59. Recommended Printed Wiring Board Land Pattern (Dimensions shown in millimeters)

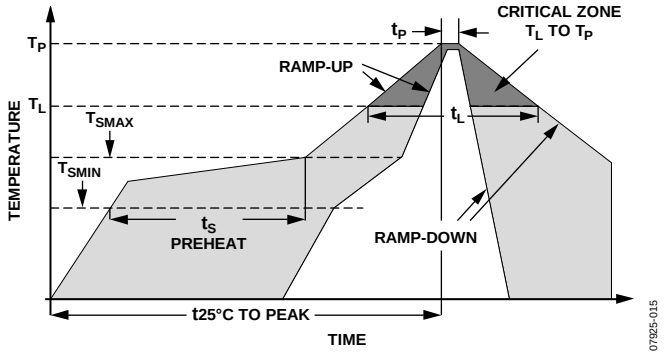


Figure 60. Recommended Soldering Profile

Table 24. Recommended Soldering Profile[1, 2]

	Condition	
Profile Feature	**Sn63/Pb37**	**Pb-Free**
Average Ramp Rate from Liquid Temperature (T_L) to Peak Temperature (T_P)	3°C/sec maximum	3°C/sec maximum
Preheat		
Minimum Temperature (T_{SMIN})	100°C	150°C
Maximum Temperature (T_{SMAX})	150°C	200°C
Time from 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 maximum	3°C/sec maximum
Liquid Temperature (T_L)	183°C	217°C
Time Maintained Above T_L (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 of Actual T_P − 5°C (t_P)	10 sec to 30 sec	20 sec to 40 sec
Ramp-Down Rate	6°C/sec maximum	6°C/sec maximum
Time 25°C to Peak Temperature	6 minutes maximum	8 minutes maximum

[1] Based on JEDEC Standard J-STD-020D.1.
[2] For best results, the soldering profile should be in accordance with the recommendations of the manufacturer of the solder paste used.

OUTLINE DIMENSIONS

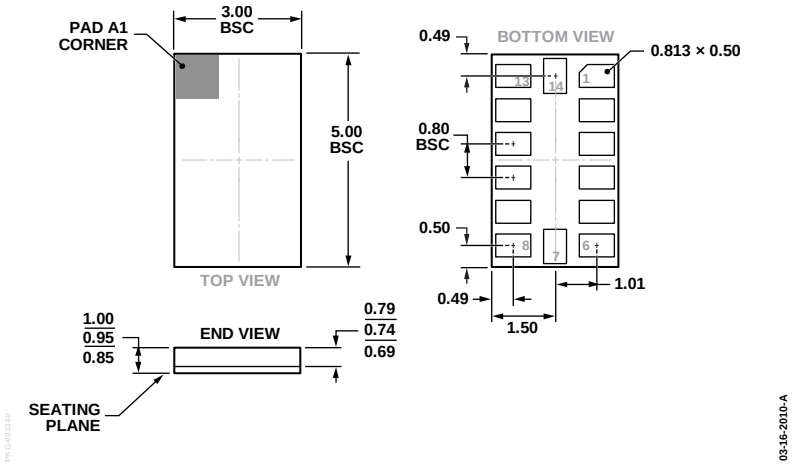


Figure 61. 14-Terminal Land Grid Array [LGA]
(CC-14-1)
Solder Terminations Finish Is Au over Ni
Dimensions shown in millimeters

ORDERING GUIDE

Model[1]	Measurement Range (*g*)	Specified Voltage (V)	Temperature Range	Package Description	Package Option
ADXL345BCCZ	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14-Terminal Land Grid Array [LGA]	CC-14-1
ADXL345BCCZ-RL	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14-Terminal Land Grid Array [LGA]	CC-14-1
ADXL345BCCZ-RL7	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14-Terminal Land Grid Array [LGA]	CC-14-1
EVAL-ADXL345Z				Evaluation Board	
EVAL-ADXL345Z-DB				Evaluation Board	
EVAL-ADXL345Z-M				Analog Devices Inertial Sensor Evaluation System, Includes ADXL345 Satellite	
EVAL-ADXL345Z-S				ADXL345 Satellite, Standalone	

[1] Z = RoHS Compliant Part.

NOTES

NOTES

NOTES

I²C refers to a communications protocol originally developed by Philips Semiconductors (now NXP Semiconductors).

D07925-0-6/15(E)

L3G4200D

MEMS motion sensor: ultra-stable three-axis digital output gyroscope

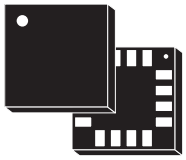
Preliminary data

Features

- Three selectable full scales (250/500/2000 dps)
- I^2C/SPI digital output interface
- 16 bit-rate value data output
- 8-bit temperature data output
- Two digital output lines (interrupt and data ready)
- Integrated low- and high-pass filters with user-selectable bandwidth
- Ultra-stable over temperature and time
- Wide supply voltage: 2.4 V to 3.6 V
- Low voltage-compatible IOs (1.8 V)
- Embedded power-down and sleep mode
- Embedded temperature sensor
- Embedded FIFO
- High shock survivability
- Extended operating temperature range (-40 °C to +85 °C)
- ECOPACK® RoHS and “Green” compliant

Applications

- Gaming and virtual reality input devices
- Motion control with MMI (man-machine interface)
- GPS navigation systems
- Appliances and robotics



LGA-16 (4x4x1.1 mm)

Description

The L3G4200D is a low-power three-axis angular rate sensor able to provide unprecedented stablility of zero rate level and sensitivity over temperature and time. It includes a sensing element and an IC interface capable of providing the measured angular rate to the external world through a digital interface (I^2C/SPI).

The sensing element is manufactured using a dedicated micro-machining process developed by STMicroelectronics to produce inertial sensors and actuators on silicon wafers.

The IC interface is manufactured using a CMOS process that allows a high level of integration to design a dedicated circuit which is trimmed to better match the sensing element characteristics.

The L3G4200D has a full scale of ±250/±500/ ±2000 dps and is capable of measuring rates with a user-selectable bandwidth.

The L3G4200D is available in a plastic land grid array (LGA) package and can operate within a temperature range of -40 °C to +85 °C.

Table 1. Device summary

Order code	Temperature range (°C)	Package	Packing
L3G4200D	-40 to +85	LGA-16 (4x4x1.1 mm)	Tray
L3G4200DTR	-40 to +85	LGA-16 (4x4x1.1 mm)	Tape and reel

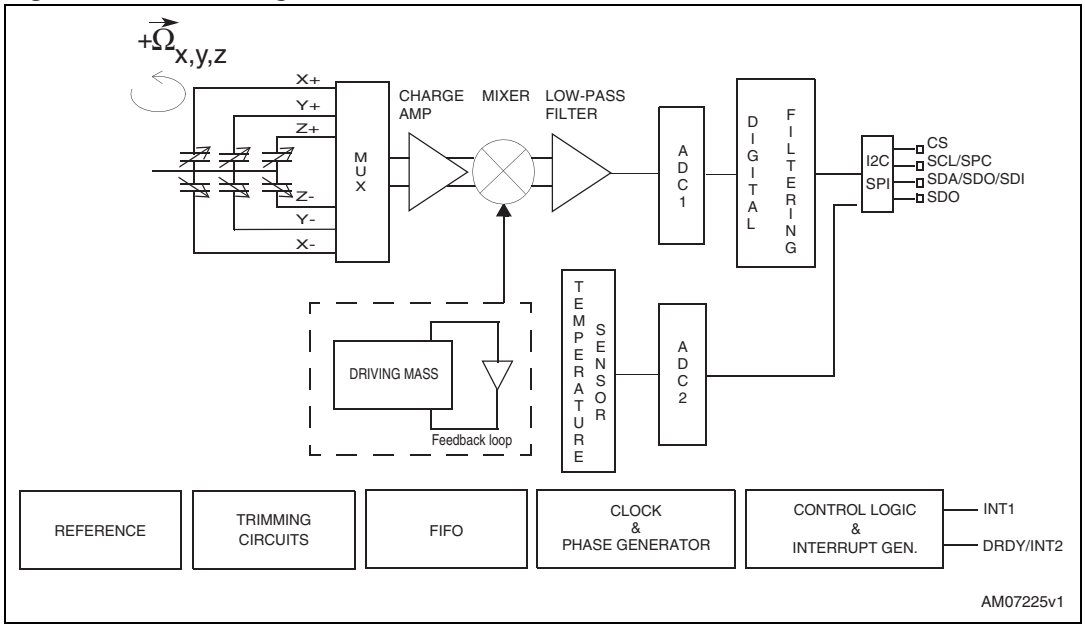
Contents

List of tables

List of figures

1 Block diagram and pin description

Figure 1. Block diagram



The vibration of the structure is maintained by drive circuitry in a feedback loop. The sensing signal is filtered and appears as a digital signal at the output.

1.1 Pin description

Figure 2. Pin connection

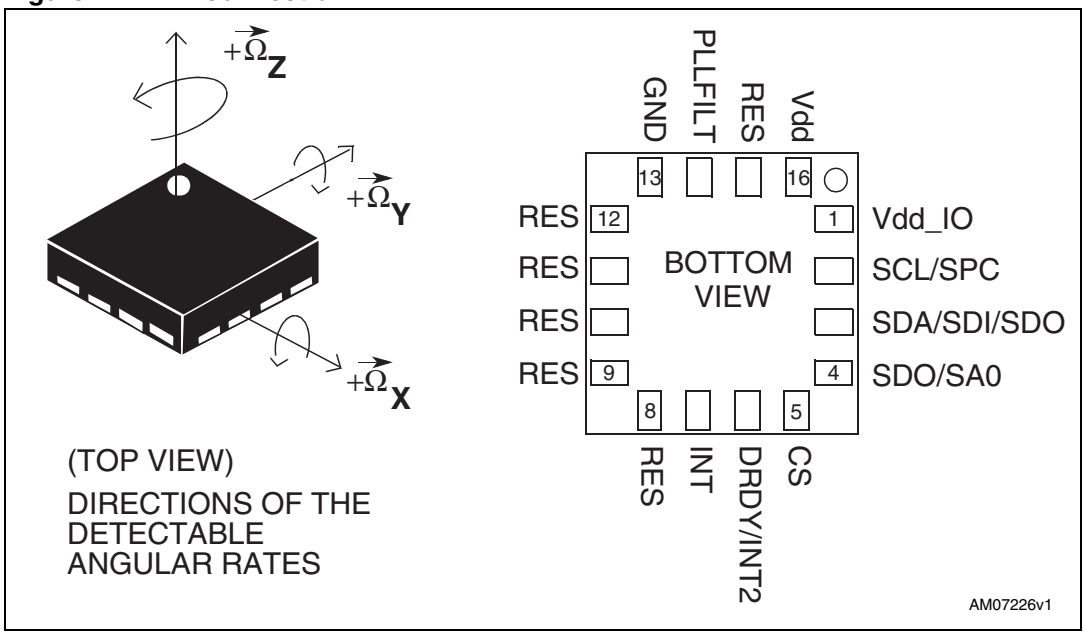
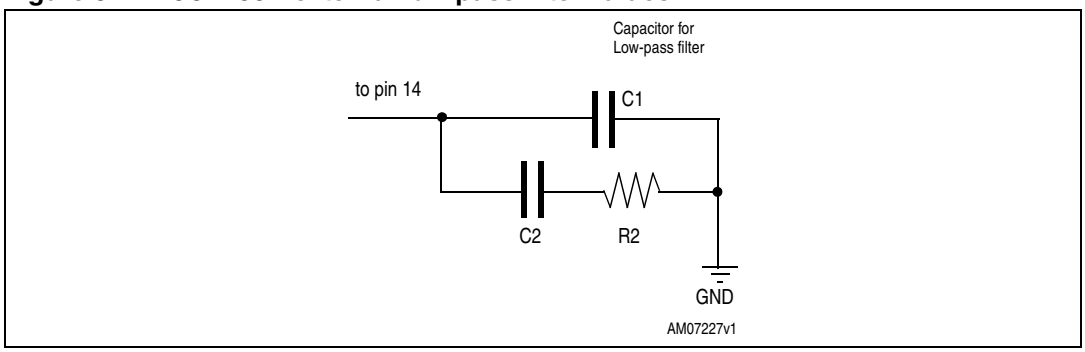


Table 2. Pin description

Pin#	Name	Function
1	Vdd_IO	Power supply for I/O pins
2	SCL SPC	I^2C serial clock (SCL) SPI serial port clock (SPC)
3	SDA SDI SDO	I^2C serial data (SDA) SPI serial data input (SDI) 3-wire interface serial data output (SDO)
4	SDO SA0	SPI serial data output (SDO) I^2C least significant bit of the device address (SA0)
5	CS	SPI enable I^2C/SPI mode selection (1:SPI idle mode / I^2C communication enabled; 0: SPI communication mode / I^2C disabled)
6	DRDY/INT2	Data ready/FIFO interrupt
7	INT1	Programmable interrupt
8	Reserved	Connect to GND
9	Reserved	Connect to GND
10	Reserved	Connect to GND
11	Reserved	Connect to GND
12	Reserved	Connect to GND
13	GND	0 V supply
14	PLLFILT	Phase-locked loop filter (see *Figure 3*)
15	Reserved	Connect to Vdd
16	Vdd	Power supply

Figure 3. L3G4200D external low-pass filter values (a)



a. Pin 14 PLLFILT maximum voltage level is equal to Vdd.

Table 3. Filter values

Parameter	Typical value
C1	10 nF
C2	470 nF
R2	10 kΩ

2 Mechanical and electrical characteristics

2.1 Mechanical characteristics

Table 4. Mechanical characteristics @ Vdd = 3.0 V, T = 25 °C, unless otherwise noted(1)

Symbol	Parameter	Test condition	Min.	Typ.(2)	Max.	Unit
FS	Measurement range	User-selectable		±250		dps
				±500		
				±2000		
So	Sensitivity	FS = 250 dps		8.75		mdps/digit
		FS = 500 dps		17.50		
		FS = 2000 dps		70		
SoDr	Sensitivity change vs. temperature	From -40 °C to +85 °C		±2		%
DVoff	Digital zero-rate level	FS = 250 dps		±10		dps
		FS = 500 dps		±15		
		FS = 2000 dps		±75		
OffDr	Zero-rate level change vs. temperature(3)	FS = 250 dps		±0.03		dps/°C
		FS = 2000 dps		±0.04		dps/°C
NL	Non linearity(4)	Best fit straight line		0.2		% FS
DST	Self-test output change	FS = 250 dps		130		dps
		FS = 500 dps		200		
		FS = 2000 dps		530		
Rn	Rate noise density	BW = 50 Hz		0.03		dps/ sqrt(Hz)
ODR	Digital output data rate			100/200/ 400/800		Hz
Top	Operating temperature range		-40		+85	°C

1. The product is factory calibrated at 3.0 V. The operational power supply range is specified in *Table 5*.
2. Typical specifications are not guaranteed.
3. Min/max values have been estimated based on the measurements of the current gyros in production.
4. Guaranteed by design.

2.2 Electrical characteristics

Table 5. Electrical characteristics @ Vdd =3.0 V, T=25 °C, unless otherwise noted(1)

Symbol	Parameter	Test condition	Min.	Typ.(2)	Max.	Unit
Vdd	Supply voltage		2.4	3.0	3.6	V
Vdd_IO	I/O pins supply voltage(3)		1.71		Vdd+0.1	V
Idd	Supply current			6.1		mA
IddSL	Supply current in sleep mode(4)	Selectable by digital interface		1.5		mA
IddPdn	Supply current in power-down mode	Selectable by digital interface		5		µA
Top	Operating temperature range		-40		+85	°C

1. The product is factory calibrated at 3.0 V.
2. Typical specifications are not guaranteed.
3. It is possible to remove Vdd maintaining Vdd_IO without blocking the communication busses, in this condition the measurement chain is powered off.
4. Sleep mode introduces a faster turn-on time compared to power-down mode.

2.3 Temperature sensor characteristics

Table 6. Temp. sensor characteristics @ Vdd =3.0 V, T=25 °C, unless otherwise noted(1)

Symbol	Parameter	Test condition	Min.	Typ.(2)	Max.	Unit
TSDr	Temperature sensor output change vs. temperature			-1		°C/digit
TODR	Temperature refresh rate			1		Hz
Top	Operating temperature range		-40		+85	°C

1. The product is factory calibrated at 3.0 V.
2. Typical specifications are not guaranteed.

2.4 Communication interface characteristics

2.4.1 SPI - serial peripheral interface

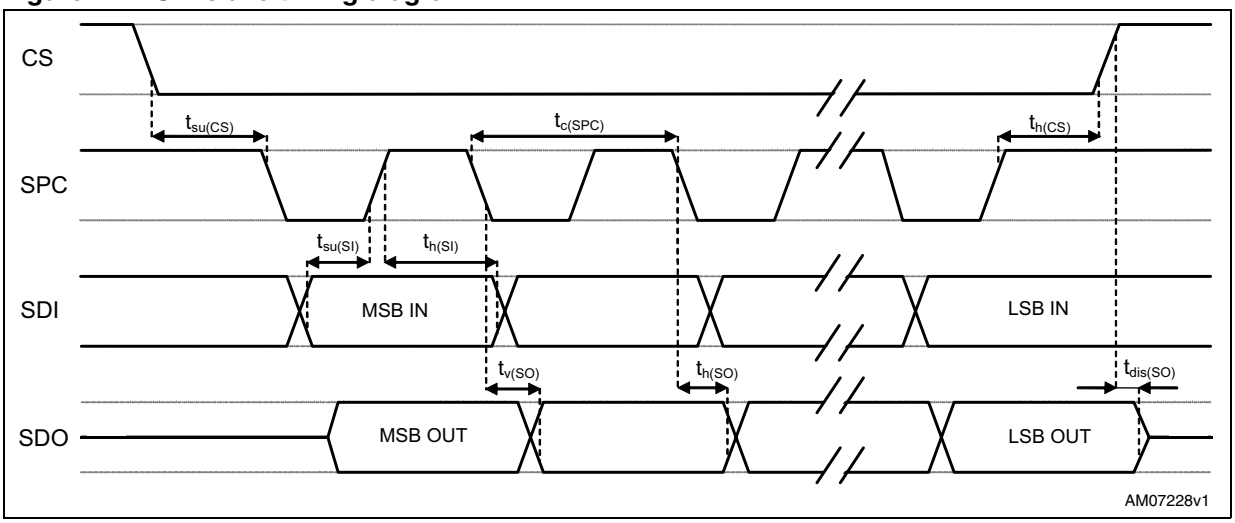
Subject to general operating conditions for Vdd and Top.

Table 7. SPI slave timing values

Symbol	Parameter	Value(1)		Unit
		Min.	Max.	
tc(SPC)	SPI clock cycle	100		ns
fc(SPC)	SPI clock frequency		10	MHz
tsu(CS)	CS setup time	5		ns
th(CS)	CS hold time	8		
tsu(SI)	SDI input setup time	5		
th(SI)	SDI input hold time	15		
tv(SO)	SDO valid output time		50	
th(SO)	SDO output hold time	6		
tdis(SO)	SDO output disable time		50	

1. Values are guaranteed at 10 MHz clock frequency for SPI with both 4 and 3 wires, based on characterization results; not tested in production.

Figure 4. SPI slave timing diagram(b)



b. Measurement points are done at 0.2·Vdd_IO and 0.8·Vdd_IO, for both input and output ports.

2.4.2 I²C - inter IC control interface

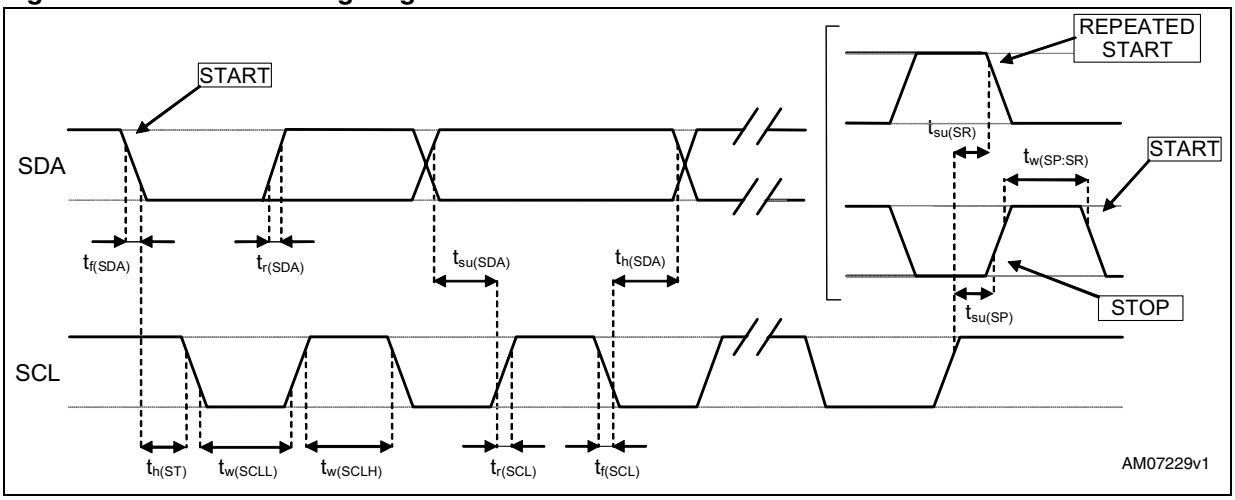
Subject to general operating conditions for Vdd and Top.

Table 8. I²C slave timing values

Symbol	Parameter	I²C standard mode(1)		I²C fast mode (1)		Unit
		Min	Max	Min	Max	
$f_{(SCL)}$	SCL clock frequency	0	100	0	400	kHz
$t_{w(SCLL)}$	SCL clock low time	4.7		1.3		µs
$t_{w(SCLH)}$	SCL clock high time	4.0		0.6		
$t_{su(SDA)}$	SDA setup time	250		100		ns
$t_{h(SDA)}$	SDA data hold time	0	3.45	0	0.9	µs
$t_{r(SDA)}$ $t_{r(SCL)}$	SDA and SCL rise time		1000	$20 + 0.1C_b$ (2)	300	ns
$t_{f(SDA)}$ $t_{f(SCL)}$	SDA and SCL fall time		300	$20 + 0.1C_b$ (2)	300	
$t_{h(ST)}$	START condition hold time	4		0.6		µs
$t_{su(SR)}$	Repeated START condition setup time	4.7		0.6		
$t_{su(SP)}$	STOP condition setup time	4		0.6		
$t_{w(SP:SR)}$	Bus free time between STOP and START condition	4.7		1.3		

1. Data based on standard I²C protocol requirement; not tested in production.
2. Cb = total capacitance of one bus line, in pF.

Figure 5. I²C slave timing diagram (c)



c. Measurement points are done at 0.2·Vdd_IO and 0.8·Vdd_IO, for both ports.

2.5 Absolute maximum ratings

Any stress above that listed as “Absolute maximum ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device under these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

Table 9. Absolute maximum ratings

Symbol	Ratings	Maximum value	Unit
Vdd	Supply voltage	-0.3 to 4.8	V
T_{STG}	Storage temperature range	-40 to +125	°C
Sg	Acceleration *g* for 0.1 ms	10,000	*g*
ESD	Electrostatic discharge protection	2 (HBM)	kV

This is a mechanical shock sensitive device, improper handling can cause permanent damage to the part

This is an ESD sensitive device, improper handling can cause permanent damage to the part

2.6 Terminology

2.6.1 Sensitivity

An angular rate gyroscope is a device that produces a positive-going digital output for counterclockwise rotation around the sensitive axis considered. Sensitivity describes the gain of the sensor and can be determined by applying a defined angular velocity to it. This value changes very little over temperature and time.

2.6.2 Zero-rate level

Zero-rate level describes the actual output signal if there is no angular rate present. The zero-rate level of precise MEMS sensors is, to some extent, a result of stress to the sensor and, therefore, the zero-rate level can slightly change after mounting the sensor onto a printed circuit board or after exposing it to extensive mechanical stress. This value changes very little over temperature and time.

2.6.3 Stability over temperature and time

Thanks to the unique single driving mass approach and optimized design, ST gyroscopes are able to guarantee a perfect match of the MEMS mechanical mass and the ASIC interface, and deliver unprecedented levels of stability over temperature and time.

With Zero rate level and sensitivity performances, up to ten times better than equivalent products now available on the market, L3G4200D allows the user to avoid any further compensation and calibration during production for faster time to market, easy application implementation, higher performances and cost saving.

2.7 Soldering information

The LGA package is compliant with the ECOPACK®, RoHS and “Green” standard.
It is qualified for soldering heat resistance according to JEDEC J-STD-020.

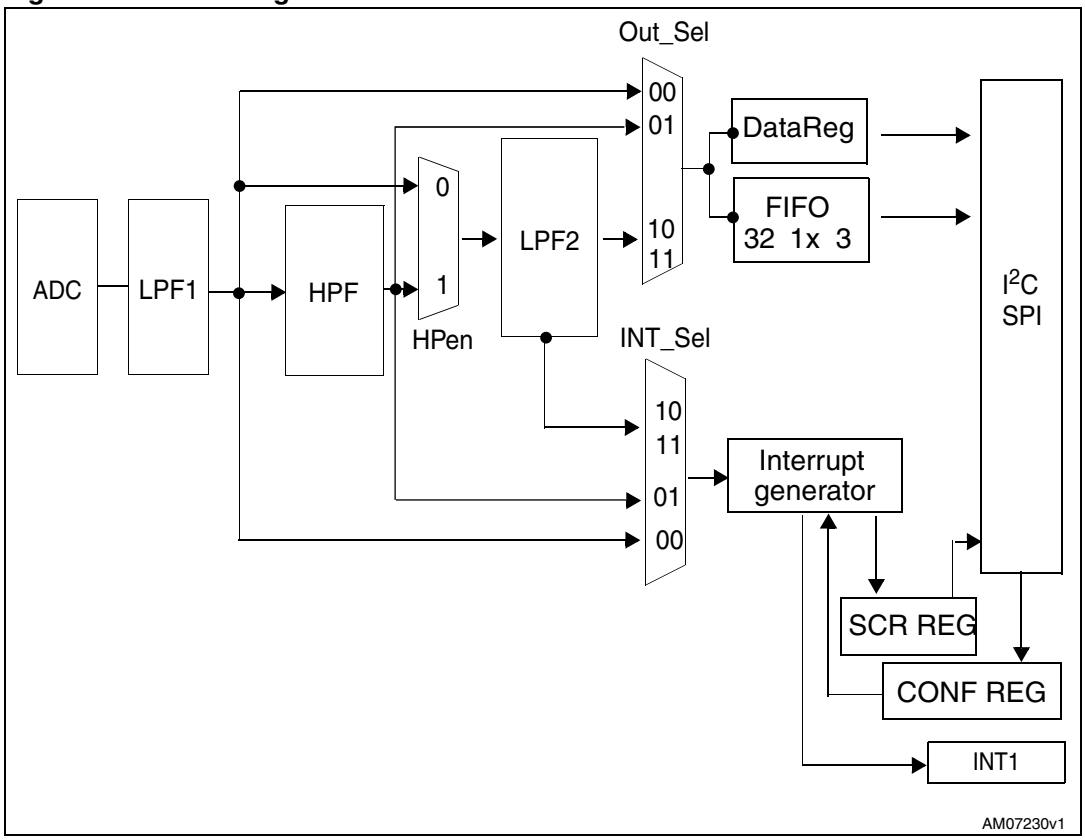
Leave “pin 1 Indicator” unconnected during soldering.

Land pattern and soldering recommendations are available at www.st.com/.

3 Main digital blocks

3.1 Block diagram

Figure 6. Block diagram



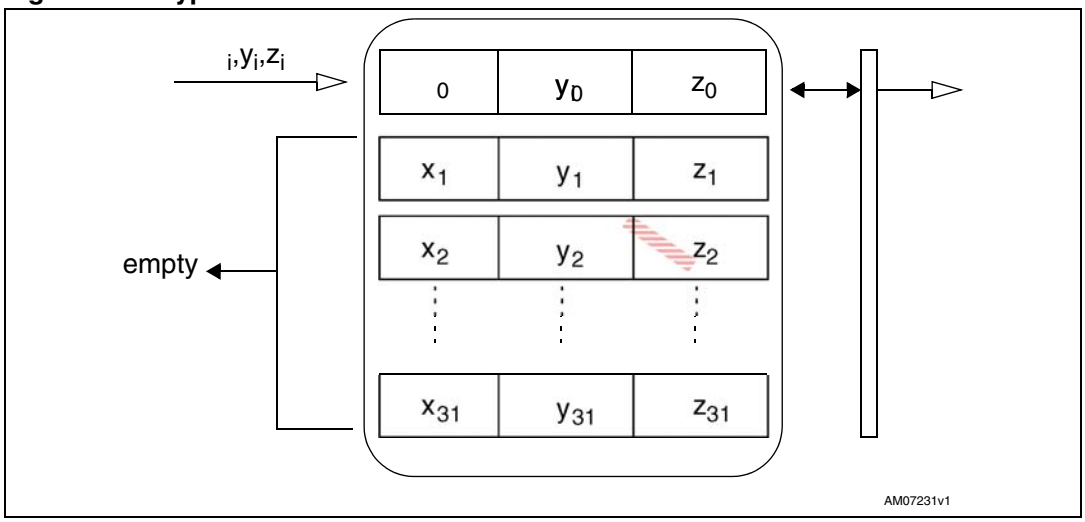
3.2 FIFO

The L3G4200D embeds a 32-slot, 16-bit data FIFO for each of the three output channels: yaw, pitch, and roll. This allows consistent power saving for the system, as the host processor does not need to continuously poll data from the sensor. Instead, it can wake up only when needed and burst the significant data out from the FIFO. This buffer can work in five different modes. Each mode is selected by the FIFO_MODE bits in the FIFO_CTRL_REG. Programmable watermark level, FIFO_empty or FIFO_Full events can be enabled to generate dedicated interrupts on the DRDY/INT2 pin (configured through CTRL_REG3), and event detection information is available in FIFO_SRC_REG. The watermark level can be configured to WTM4:0 in FIFO_CTRL_REG.

3.2.1 Bypass mode

In bypass mode, the FIFO is not operational and for this reason it remains empty. As illustrated in *Figure 7*, only the first address is used for each channel. The remaining FIFO slots are empty. When new data is available, the old data is overwritten.

Figure 7. Bypass mode

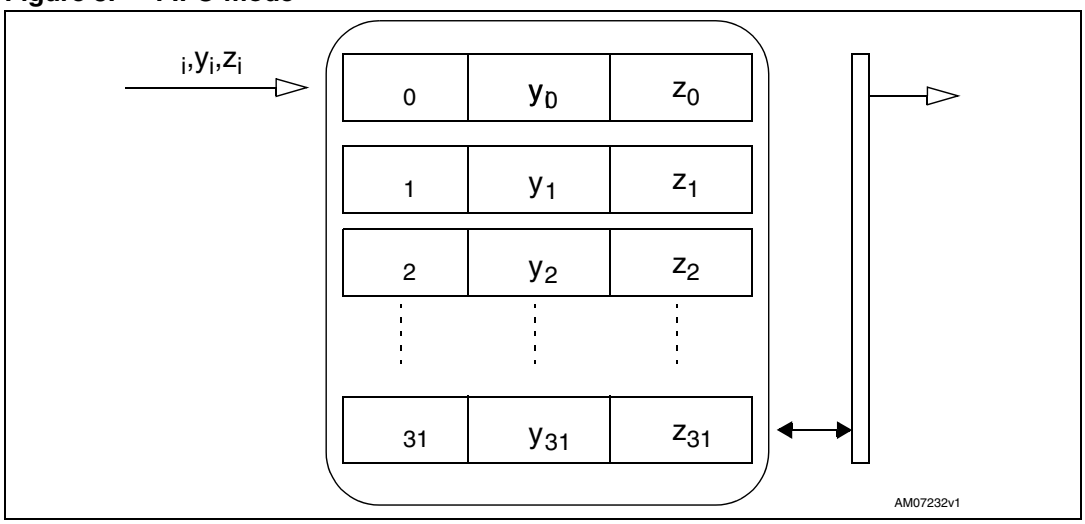


3.2.2 FIFO mode

In FIFO mode, data from the yaw, pitch, and roll channels are stored in the FIFO. A watermark interrupt can be enabled (I2_WMK bit in CTRL_REG3), which is triggered when the FIFO is filled to the level specified in the WTM 4:0 bits of FIFO_CTRL_REG. The FIFO continues filling until it is full (32 slots of 16-bit data for yaw, pitch, and roll). When full, the FIFO stops collecting data from the input channels. To restart data collection, it is necessary to write FIFO_CTRL_REG back to bypass mode.

FIFO mode is represented in *Figure 8*.

Figure 8. FIFO mode



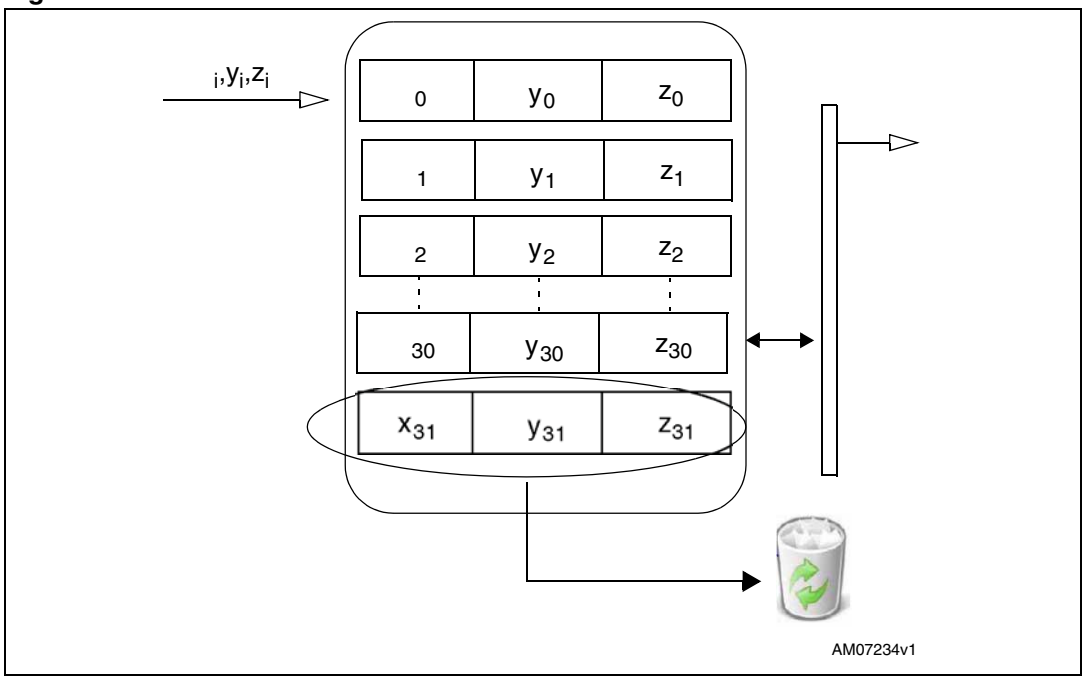
3.2.3 Stream mode

In stream mode, data from yaw, pitch, and roll measurements are stored in the FIFO. A watermark interrupt can be enabled and set as in FIFO mode. The FIFO continues filling until full (32 slots of 16-bit data for yaw, pitch, and roll). When full, the FIFO discards the

older data as the new data arrives. Programmable watermark level events can be enabled to generate dedicated interrupts on the DRDY/INT2 pin (configured through CTRL_REG3).

Stream mode is represented in *Figure 9*.

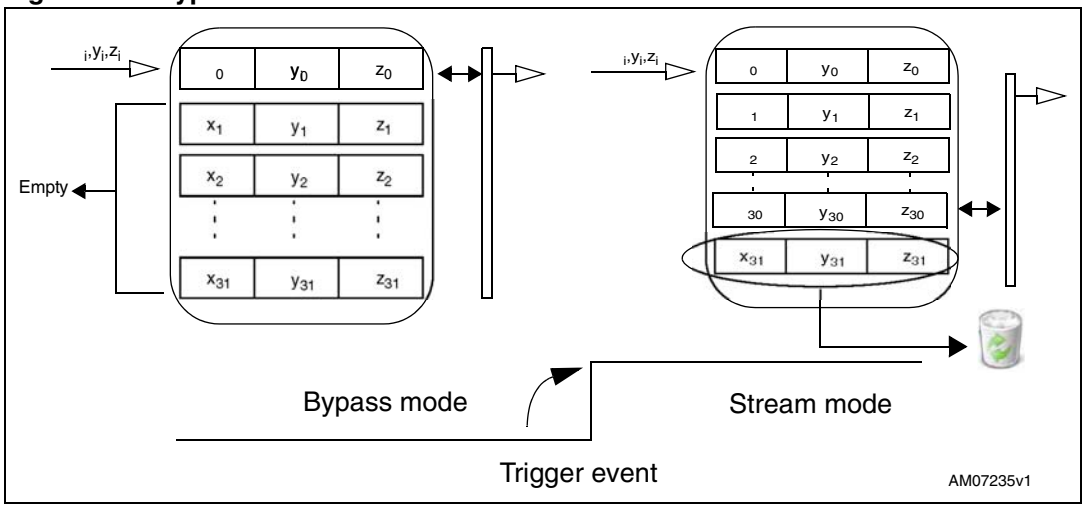
Figure 9. Stream mode



3.2.4 Bypass-to-stream mode

In bypass-to-stream mode, the FIFO starts operating in bypass mode, and once a trigger event occurs (related to INT1_CFG register events), the FIFO starts operating in stream mode (see *Figure 10*).

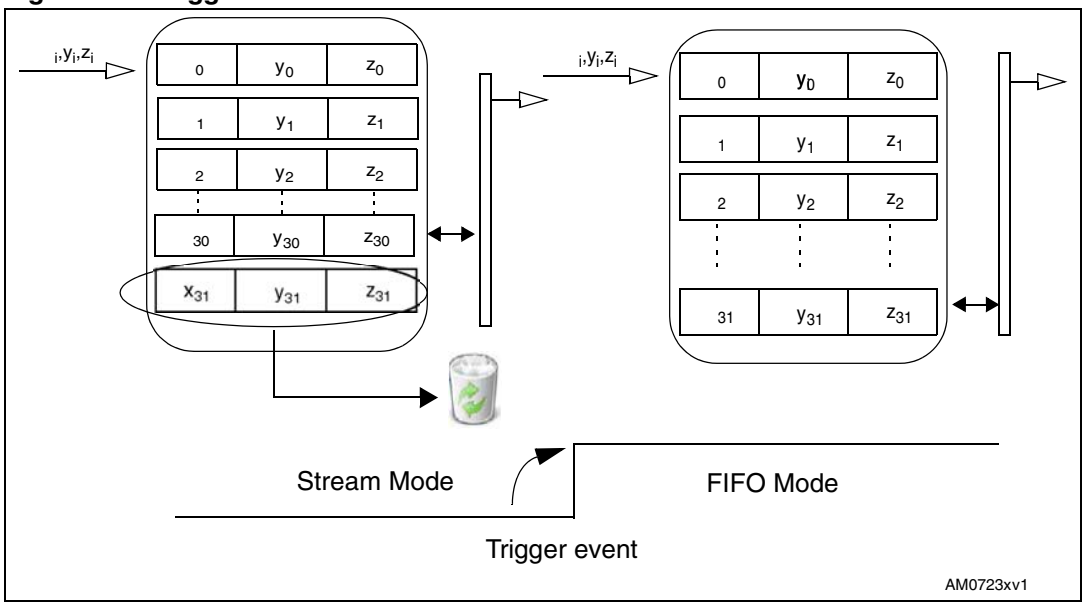
Figure 10. Bypass-to-stream mode



3.2.5 Stream-to-FIFO mode

In stream-to-FIFO mode, data from yaw, pitch, and roll measurements are stored in the FIFO. A watermark interrupt can be enabled on pin DRDY/INT2, setting the I2_WTM bit in CTRL_REG3, which is triggered when the FIFO is filled to the level specified in the WTM4:0 bits of FIFO_CTRL_REG. The FIFO continues filling until full (32 slots of 16-bit data for yaw, pitch, and roll). When full, the FIFO discards the older data as the new data arrives. Once a trigger event occurs (related to INT1_CFG register events), the FIFO starts operating in FIFO mode (see *Figure 11*).

Figure 11. Trigger stream mode

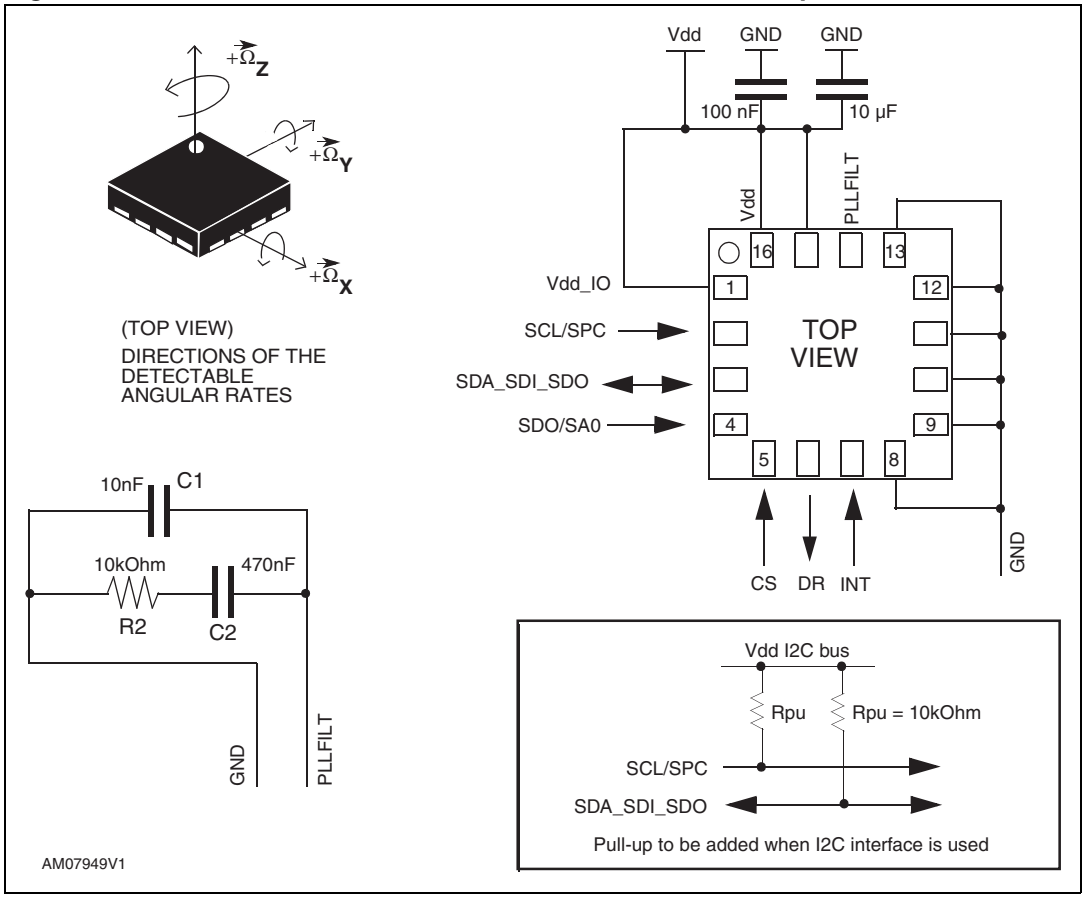


3.2.6 Retrieve data from FIFO

FIFO data is read through the OUT_X, OUT_Y and OUT_Z registers. When the FIFO is in stream, trigger or FIFO mode, a read operation to the OUT_X, OUT_Y or OUT_Z registers provides the data stored in the FIFO. Each time data is read from the FIFO, the oldest pitch, roll, and yaw data are placed in the OUT_X, OUT_Y and OUT_Z registers and both single read and read_burst (X,Y & Z with auto-incremental address) operations can be used. In read_burst mode, when data included in OUT_Z_H is read, the system again starts to read information from addr OUT_X_L.

4 Application hints

Figure 12. L3G4200D electrical connections and external component values



Power supply decoupling capacitors (100 nF ceramic or polyester +10 µF) should be placed as near as possible to the device (common design practice).

If Vdd and Vdd_IO are not connected together, power supply decoupling capacitors (100 nF and 10 µF between Vdd and common ground, 100 nF between Vdd_IO and common ground) should be placed as near as possible to the device (common design practice).

The L3G4200D IC includes a PLL (phase locked loop) circuit to synchronize driving and sensing interfaces. Capacitors and resistors must be added at the **PLLFILT** pin (as shown in *Figure 12*) to implement a second-order low-pass filter. *Table 10* summarizes the PLL low-pass filter component values.

Table 10. PLL low-pass filter component values

Component	Value
C1	10 nF ± 10 %
C2	470 nF ± 10 %
R2	10 kΩ ± 10 %

5 Digital interfaces

The registers embedded in the L3G4200D may be accessed through both the I^2C and SPI serial interfaces. The latter may be software-configured to operate either in 3-wire or 4-wire interface mode.

The serial interfaces are mapped onto the same pins. To select/exploit the I^2C interface, the CS line must be tied high (i.e., connected to Vdd_IO).

Table 11. Serial interface pin description

Pin name	Pin description
CS	SPI enable I^2C/SPI mode selection (1:SPI idle mode / I^2C communication enabled; 0: SPI communication mode / I^2C disabled)
SCL/SPC	I^2C serial clock (SCL) SPI serial port clock (SPC)
SDA/SDI/SDO	I^2C serial data (SDA) SPI serial data input (SDI) 3-wire interface serial data output (SDO)
SDO	SPI serial data output (SDO) I^2C least significant bit of the device address

5.1 I^2C serial interface

The L3G4200D I^2C is a bus slave. The I^2C is employed to write data to registers whose content can also be read back.

The relevant I^2C terminology is given in the table below.

Table 12. I^2C terminology

Term	Description
Transmitter	The device which sends data to the bus
Receiver	The device which receives data from the bus
Master	The device which initiates a transfer, generates clock signals and terminates a transfer
Slave	The device addressed by the master

There are two signals associated with the I^2C bus: the serial clock line (SCL) and the serial data line (SDA). The latter is a bidirectional line used for sending and receiving the data to/from the interface. Both lines must be connected to Vdd_IO through an external pull-up resistor. When the bus is free both the lines are high.

The I^2C interface is compliant with fast mode (400 kHz) I^2C standards as well as with normal mode.

5.1.1 I²C operation

The transaction on the bus is started through a START (ST) signal. A START condition is defined as a HIGH to LOW transition on the data line while the SCL line is held HIGH. After this has been transmitted by the master, the bus is considered busy. The next byte of data transmitted after the start condition contains the address of the slave in the first 7 bits and the eighth bit tells whether the master is receiving data from the slave or transmitting data to the slave. When an address is sent, each device in the system compares the first 7 bits after a start condition with its address. If they match, the device considers itself addressed by the master.

The slave address (SAD) associated with the L3G4200D is 110100xb. The SDO pin can be used to modify the least significant bit (LSb) of the device address. If the SDO pin is connected to the voltage supply, LSb is '1' (address 1101001b). Otherwise, if the SDO pin is connected to ground, the LSb value is '0' (address 1101000b). This solution permits the connection and addressing of two different gyroscopes to the same I²C bus.

Data transfer with acknowledge is mandatory. The transmitter must release the SDA line during the acknowledge pulse. The receiver must then pull the data line LOW so that it remains stable low during the HIGH period of the acknowledge clock pulse. A receiver which has been addressed is obliged to generate an acknowledge after each byte of data received.

The I²C embedded in the L3G4200D behaves like a slave device, and the following protocol must be adhered to. After the START (ST) condition, a slave address is sent. Once a slave acknowledge (SAK) has been returned, an 8-bit sub-address is transmitted. The 7 LSb represent the actual register address while the MSB enables address auto-increment. If the MSb of the SUB field is 1, the SUB (register address) is automatically incremented to allow multiple data read/write.

The slave address is completed with a read/write bit. If the bit is '1' (read), a REPEATED START (SR) condition must be issued after the two sub-address bytes; if the bit is '0' (write) the master transmits to the slave with the direction unchanged. *Table 13* describes how the SAD+read/write bit pattern is composed, listing all the possible configurations.

Table 13. SAD+read/write patterns

Command	SAD[6:1]	SAD[0] = SDO	R/W	SAD+R/W
Read	110100	0	1	11010001 (D1h)
Write	110100	0	0	11010000 (D0h)
Read	110100	1	1	11010011 (D3h)
Write	110100	1	0	11010010 (D2h)

Table 14. Transfer when master is writing one byte to slave

Master	ST	SAD + W		SUB		DATA		SP
Slave			SAK		SAK		SAK	

Table 15. Transfer when master is writing multiple bytes to slave

Master	ST	SAD + W		SUB		DATA		DATA		SP
Slave			SAK		SAK		SAK		SAK	

Table 16. Transfer when master is receiving (reading) one byte of data from slave

Master	ST	SAD + W		SUB		SR	SAD + R			NMAK	SP
Slave			SAK		SAK			SAK	DATA		

Table 17. Transfer when master is receiving (reading) multiple bytes of data from slave

Master	ST	SAD+W		SUB		SR	SAD+R			MAK		MAK		NMAK	SP
Slave			SAK		SAK			SAK	DATA		DATA		DATA		

Data are transmitted in byte format (DATA). Each data transfer contains 8 bits. The number of bytes transferred per transfer is unlimited. Data is transferred with the most significant bit (MSb) first. If a receiver cannot receive another complete byte of data until it has performed some other function, it can hold the clock line SCL LOW to force the transmitter into a wait state. Data transfer only continues when the receiver is ready for another byte and releases the data line. If a slave receiver does not acknowledge the slave address (i.e., it is not able to receive because it is performing some real-time function) the data line must be left HIGH by the slave. The master can then abort the transfer. A LOW to HIGH transition on the SDA line while the SCL line is HIGH is defined as a STOP condition. Each data transfer must be terminated by the generation of a STOP (SP) condition.

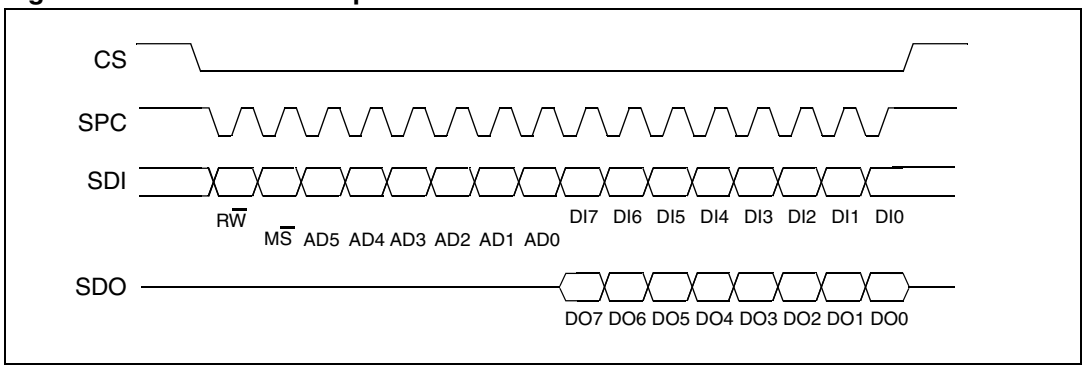
In order to read multiple bytes, it is necessary to assert the most significant bit of the sub-address field. In other words, SUB(7) must be equal to 1, while SUB(6-0) represents the address of the first register to be read.

In the presented communication format, MAK is “master acknowledge” and NMAK is “no master acknowledge”.

5.2 SPI bus interface

The SPI is a bus slave. The SPI allows writing and reading of the device registers. The serial interface interacts with the external world through 4 wires: **CS, SPC, SDI, and SDO**.

Figure 13. Read and write protocol



CS is the serial port enable and is controlled by the SPI master. It goes low at the start of the transmission and returns to high at the end. **SPC** is the serial port clock and is controlled by the SPI master. It is stopped high when **CS** is high (no transmission). **SDI** and **SDO** are, respectively, the serial port data input and output. These lines are driven at the falling edge of **SPC** and should be captured at the rising edge of **SPC**.

Both the read register and write register commands are completed in 16 clock pulses, or in multiples of 8 in case of multiple read/write bytes. Bit duration is the time between two falling edges of **SPC**. The first bit (bit 0) starts at the first falling edge of **SPC** after the falling edge of **CS** while the last bit (bit 15, bit 23, etc.) starts at the last falling edge of **SPC** just before the rising edge of **CS**.

Bit 0: R$\overline{W}$ bit. When 0, the data DI(7:0) is written to the device. When 1, the data DO(7:0) from the device is read. In the latter case, the chip drives **SDO** at the start of bit 8.

Bit 1: M$\overline{S}$ bit. When 0, the address remains unchanged in multiple read/write commands. When 1, the address is auto-incremented in multiple read/write commands.

Bit 2-7: address AD(5:0). This is the address field of the indexed register.

Bit 8-15: data DI(7:0) (write mode). This is the data that is written to the device (MSb first).

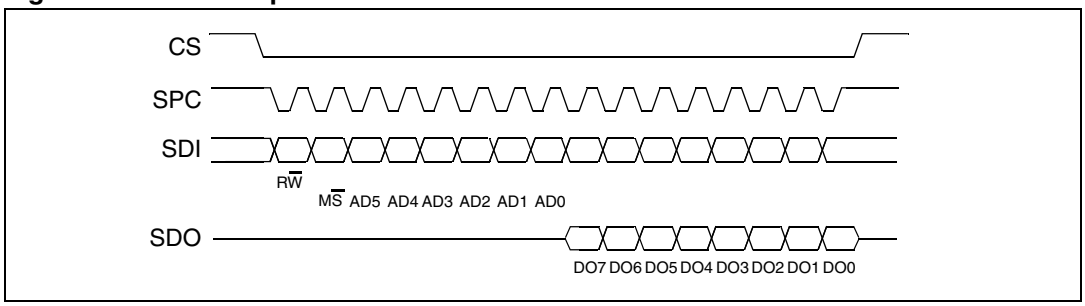
Bit 8-15: data DO(7:0) (read mode). This is the data that is read from the device (MSb first).

In multiple read/write commands, further blocks of 8 clock periods are added. When the M$\overline{S}$ bit is 0, the address used to read/write data remains the same for every block. When the M$\overline{S}$ bit is 1, the address used to read/write data is incremented at every block.

The function and the behavior of **SDI** and **SDO** remain unchanged.

5.2.1 SPI read

Figure 14. SPI read protocol



The SPI read command is performed with 16 clock pulses. A multiple byte read command is performed by adding blocks of 8 clock pulses to the previous one.

Bit 0: READ bit. The value is 1.

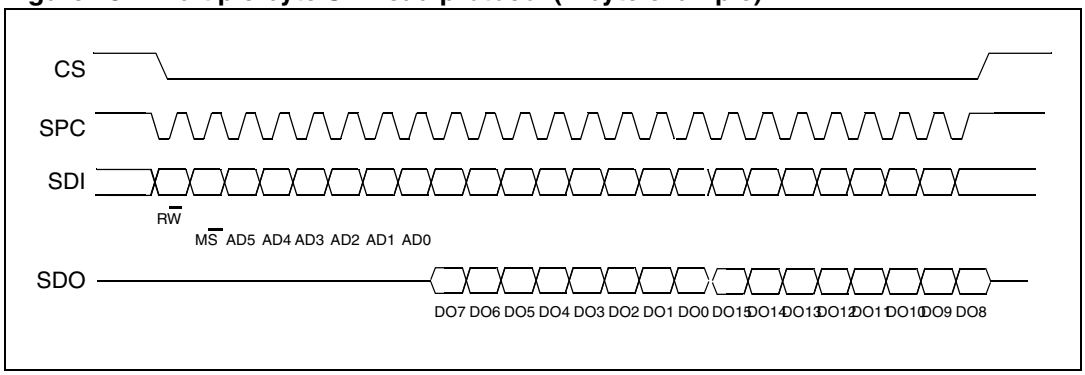
Bit 1: MS bit. When 0, do not increment address; when 1, increment address in multiple reading.

Bit 2-7: address AD(5:0). This is the address field of the indexed register.

Bit 8-15: data DO(7:0) (read mode). This is the data that is read from the device (MSb first).

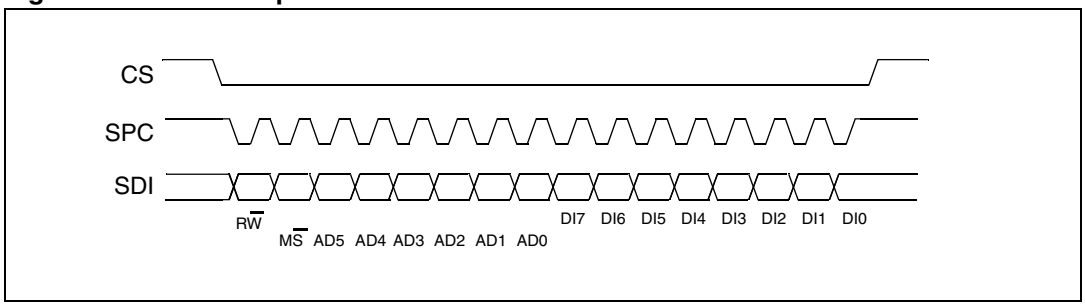
Bit 16-... : data DO(...-8). Further data in multiple byte reading.

Figure 15. Multiple byte SPI read protocol (2-byte example)



5.2.2 SPI write

Figure 16. SPI write protocol



The SPI write command is performed with 16 clock pulses. A multiple byte write command is performed by adding blocks of 8 clock pulses to the previous one.

Bit 0: WRITE bit. The value is 0.

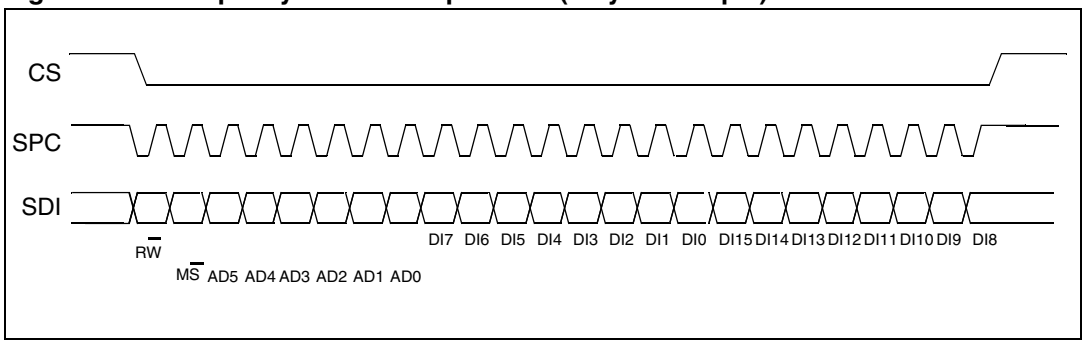
Bit 1: MS bit. When 0, do not increment address; when 1, increment address in multiple writing.

Bit 2 -7: address AD(5:0). This is the address field of the indexed register.

Bit 8-15: data DI(7:0) (write mode). This is the data that is written to the device (MSb first).

Bit 16-... : data DI(...-8). Further data in multiple byte writing.

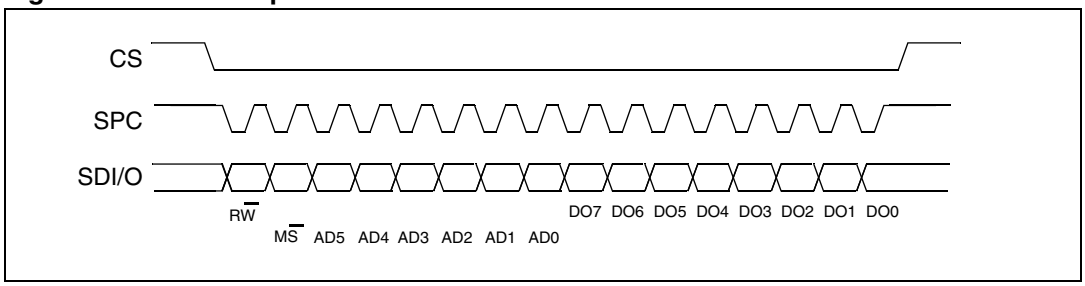
Figure 17. Multiple byte SPI write protocol (2-byte example)



5.2.3 SPI read in 3-wire mode

3-wire mode is entered by setting the SIM (SPI serial interface mode selection) bit to 1 in CTRL_REG2.

Figure 18. SPI read protocol in 3-wire mode



The SPI read command is performed with 16 clock pulses:

Bit 0: READ bit. The value is 1.

Bit 1: MS bit. When 0, do not increment address; when 1, increment address in multiple reading.

Bit 2-7: address AD(5:0). This is the address field of the indexed register.

Bit 8-15: data DO(7:0) (read mode). This is the data that is read from the device (MSb first).

The multiple read command is also available in 3-wire mode.

6 Output register mapping

The table given below provides a listing of the 8 bit registers embedded in the device and the related addresses:

Table 18. Register address map

Name	Type	Register address		Default	Comment
		Hex	Binary		
Reserved	-	00-0E	-	-	
WHO_AM_I	r	0F	000 1111	11010011	
Reserved	-	10-1F	-	-	
CTRL_REG1	rw	20	010 0000	00000111	
CTRL_REG2	rw	21	010 0001	00000000	
CTRL_REG3	rw	22	010 0010	00000000	
CTRL_REG4	rw	23	010 0011	00000000	
CTRL_REG5	rw	24	010 0100	00000000	
REFERENCE	rw	25	010 0101	00000000	
OUT_TEMP	r	26	010 0110	output	
STATUS_REG	r	27	010 0111	output	
OUT_X_L	r	28	010 1000	output	
OUT_X_H	r	29	010 1001	output	
OUT_Y_L	r	2A	010 1010	output	
OUT_Y_H	r	2B	010 1011	output	
OUT_Z_L	r	2C	010 1100	output	
OUT_Z_H	r	2D	010 1101	output	
FIFO_CTRL_REG	rw	2E	010 1110	00000000	
FIFO_SRC_REG	r	2F	010 1111	output	
INT1_CFG	rw	30	011 0000	00000000	
INT1_SRC	r	31	011 0001	output	
INT1_TSH_XH	rw	32	011 0010	00000000	
INT1_TSH_XL	rw	33	011 0011	00000000	
INT1_TSH_YH	rw	34	011 0100	00000000	
INT1_TSH_YL	rw	35	011 0101	00000000	
INT1_TSH_ZH	rw	36	011 0110	00000000	
INT1_TSH_ZL	rw	37	011 0111	00000000	
INT1_DURATION	rw	38	011 1000	00000000	

Registers marked as *Reserved* must not be changed. The writing to those registers may cause permanent damages to the device.

The content of the registers that are loaded at boot should not be changed. They contain the factory calibration values. Their content is automatically restored when the device is powered-up.

7 Register description

The device contains a set of registers which are used to control its behavior and to retrieve acceleration data. The registers address, made of 7 bits, is used to identify them and to write the data through serial interface.

7.1 WHO_AM_I (0Fh)

Table 19. WHO_AM_I register

1	1	0	1	0	0	1	1

Device identification register.

7.2 CTRL_REG1 (20h)

Table 20. CTRL_REG1 register

DR1	DR0	BW1	BW0	PD	Zen	Yen	Xen

Table 21. CTRL_REG1 description

DR1-DR0	Output Data Rate selection. Refer to *Table 22*
BW1-BW0	Bandwidth selection. Refer to *Table 22*
PD	Power down mode enable. Default value: 0 (0: power down mode, 1: normal mode or sleep mode)
Zen	Z axis enable. Default value: 1 (0: Z axis disabled; 1: Z axis enabled)
Yen	Y axis enable. Default value: 1 (0: Y axis disabled; 1: Y axis enabled)
Xen	X axis enable. Default value: 1 (0: X axis disabled; 1: X axis enabled)

DR<1:0> is used to set ODR selection. **BW <1:0>** is used to set Bandwidth selection.

In the following table are reported all frequency resulting in combination of DR / BW bits.

Table 22. DR and BW configuration setting

DR <1:0>	BW <1:0>	ODR [Hz]	Cut-Off
00	00	100	12.5
00	01	100	25
00	10	100	25
00	11	100	25

Table 22. DR and BW configuration setting (continued)

DR <1:0>	BW <1:0>	ODR [Hz]	Cut-Off
01	00	200	12.5
01	01	200	25
01	10	200	50
01	11	200	70
10	00	400	20
10	01	400	25
10	10	400	50
10	11	400	110
11	00	800	30
11	01	800	35
11	10	800	50
11	11	800	110

Combination of **PD, Zen, Yen, Xen** are used to set device in different modes (power down / normal / sleep mode) according with the following table.

Table 23. Power mode selection configuration

Mode	PD	Zen	Yen	Xen
Power down	0	-	-	-
Sleep	1	0	0	0
Normal	1	-	-	-

7.3 CTRL_REG2 (21h)

Table 24. CTRL_REG2 register

0(1)	0(1)	HPM1	HPM1	HPCF3	HPCF2	HPCF1	HPCF0

1. Value loaded at boot. This value must not be changed

Table 25. CTRL_REG2 description

HPM1-HPM0	High Pass filter Mode Selection. Default value: 00 Refer to *Table 26*
HPCF3-HPCF0	High Pass filter Cut Off frequency selection Refer to *Table 28*

Table 26. High pass filter mode configuration

HPM1	HPM0	High Pass filter Mode
0	0	Normal mode (reset reading HP_RESET_FILTER)
0	1	Reference signal for filtering
1	0	Normal mode
1	1	Autoreset on interrupt event

Table 27. High pass filter cut off frecuency configuration [Hz]

HPCF3	ODR= 100 Hz	ODR= 200 Hz	ODR= 400 Hz	ODR= 800 Hz
0000	8	15	30	56
0001	4	8	15	30
0010	2	4	8	15
0011	1	2	4	8
0100	0.5	1	2	4
0101	0.2	0.5	1	2
0110	0.1	0.2	0.5	1
0111	0.05	0.1	0.2	0.5
1000	0.02	0.05	0.1	0.2
1001	0.01	0.02	0.05	0.1

7.4 CTRL_REG3 (22h)

Table 28. CTRL_REG1 register

I1_Int1	I1_Boot	H_Lactive	PP_OD	I2_DRDY	I2_WTM	I2_ORun	I2_Empty

Table 29. CTRL_REG3 description

I1_Int1	Interrupt enable on INT1 pin. Default value 0. (0: Disable; 1: Enable)
I1_Boot	Boot status available on INT1. Default value 0. (0: Disable; 1: Enable)
H_Lactive	Interrupt active configuration on INT1. Default value 0. (0: High; 1:Low)
PP_OD	Push- Pull / Open drain. Default value: 0. (0: Push- Pull; 1: Open drain)
I2_DRDY	Date Ready on DRDY/INT2. Default value 0. (0: Disable; 1: Enable)
I2_WTM	FIFO Watermark interrupt on DRDY/INT2. Default value: 0. (0: Disable; 1: Enable)
I2_ORun	FIFO Overrun interrupt on DRDY/INT2 Default value: 0. (0: Disable; 1: Enable)
I2_Empty	FIFO Empty interrupt on DRDY/INT2. Default value: 0. (0: Disable; 1: Enable)

7.5 CTRL_REG4 (23h)

Table 30. CTRL_REG4 register

BDU	BLE	FS1	FS0	-	ST1	ST0	SIM

Table 31. CTRL_REG4 description

BDU	Block Data Update. Default value: 0 (0: continous update; 1: output registers not updated until MSB and LSB reading)
BLE	Big/Little Endian Data Selection. Default value 0. (0: Data LSB @ lower address; 1: Data MSB @ lower address)
FS1-FS0	Full Scale selection. Default value: 00 (00: 250 dps; 01: 500 dps; 10: 2000 dps; 11: 2000 dps)
ST1-ST0	Self Test Enable. Default value: 00 (00: Self Test Disabled; Other: See *Table*)
SIM	SPI Serial Interface Mode selection. Default value: 0 (0: 4-wire interface; 1: 3-wire interface).

Table 32. Self test mode configuration

ST1	ST0	Self test mode
0	0	Normal mode
0	1	Self test 0 (+)(1)
1	0	--
1	1	Self test 1 (-)(1)

1. DST sign (absolute value in *Table 4*)

7.6 CTRL_REG5 (24h)

Table 33. CTRL_REG5 register

BOOT	FIFO_EN	--	HPen	INT1_Sel1	INT1_Sel0	Out_Sel1	Out_Sel0

Table 34. CTRL_REG5 description

BOOT	Reboot memory content. Default value: 0 (0: normal mode; 1: reboot memory content)
FIFO_EN	FIFO enable. Default value: 0 (0: FIFO disable; 1: FIFO Enable)
HPen	High Pass filter Enable. Default value: 0 (0: HPF disabled; 1: HPF enabled. See *Figure 20*)
INT1_Sel1- INT1_Sel0	INT1 selection configuration. Default value: 0 (See *Figure 20*)
Out_Sel1- Out_Sel1	Out selection configuration. Default value: 0 (See *Figure 20*

Figure 19. INT1_Sel and Out_Sel configuration block diagram

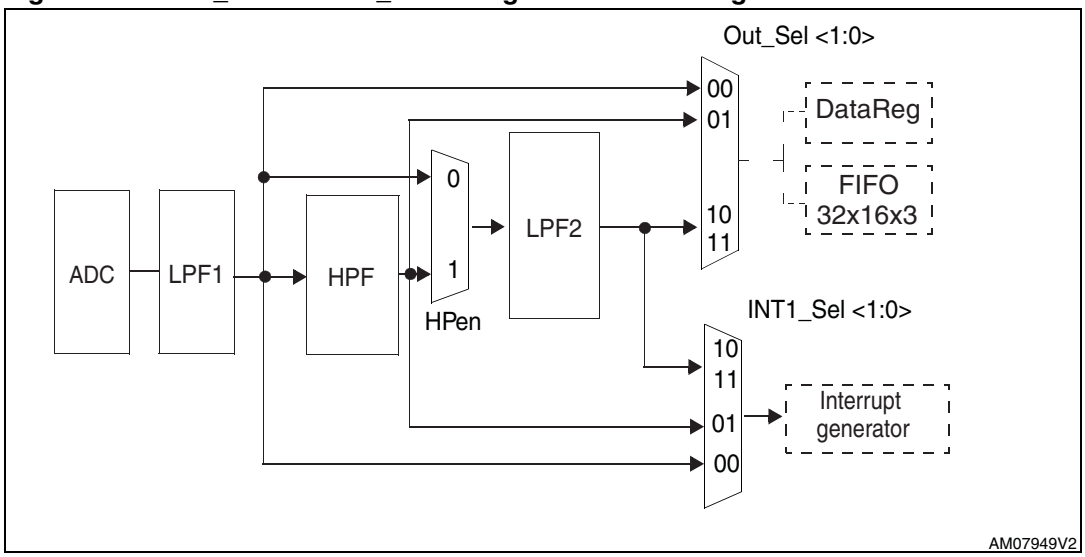


Table 35. Out_Sel configuration setting

Hpen	OUT_SEL1	OUT_SEL0	Description
x	0	0	Data in DataReg and FIFO are non-high-pass-filtered
x	0	1	Data in DataReg and FIFO are high-pass-filtered
0	1	x	Data in DataReg and FIFO are low-pass-filtered by LPF2
1	1	x	Data in DataReg and FIFO are high-pass and low-pass-filtered by LPF2

Table 36. INT_SEL configuration setting

Hpen	INT_SEL1	INT_SEL2	Description
x	0	0	Non-high-pass-filtered data are used for interrupt generation
x	0	1	High-pass-filtered data are used for interrupt generation
0	1	x	Low-pass-filtered data are used for interrupt generation
1	1	x	High-pass and low-pass-filtered data are used for interrupt generation

7.7 REFERENCE/DATACAPTURE (25h)

Table 37. REFERENCE register

Ref7	Ref6	Ref5	Ref4	Ref3	Ref2	Ref1	Ref0

Table 38. REFERENCE register description

Ref 7-Ref0	Reference value for Interrupt generation. Default value: 0

7.8 OUT_TEMP (26h)

Table 39. OUT_TEMP register

Temp7	Temp6	Temp5	Temp4	Temp3	Temp2	Temp1	Temp0

Table 40. OUT_TEMP register description

Temp7-Temp0	Temperature data.

7.9 STATUS_REG (27h)

Table 41. STATUS_REG register

ZYXOR	ZOR	YOR	XOR	ZYXDA	ZDA	YDA	XDA

Table 42. STATUS_REG description

ZYXOR	X, Y, Z -axis data overrun. Default value: 0 (0: no overrun has occurred; 1: new data has overwritten the previous one before it was read)
ZOR	Z axis data overrun. Default value: 0 (0: no overrun has occurred; 1: a new data for the Z-axis has overwritten the previous one)
YOR	Y axis data overrun. Default value: 0 (0: no overrun has occurred; 1: a new data for the Y-axis has overwritten the previous one)
XOR	X axis data overrun. Default value: 0 (0: no overrun has occurred; 1: a new data for the X-axis has overwritten the previous one)
ZYXDA	X, Y, Z -axis new data available. Default value: 0 (0: a new set of data is not yet available; 1: a new set of data is available)
ZDA	Z axis new data available. Default value: 0 (0: a new data for the Z-axis is not yet available; 1: a new data for the Z-axis is available)
YDA	Y axis new data available. Default value: 0 (0: a new data for the Y-axis is not yet available;1: a new data for the Y-axis is available)
XDA	X axis new data available. Default value: 0 (0: a new data for the X-axis is not yet available; 1: a new data for the X-axis is available)

7.10 OUT_X_L (28h), OUT_X_H (29h)

X-axis angular rate data. The value is expressed as two's complement.

7.11 OUT_Y_L (2Ah), OUT_Y_H (2Bh)

Y-axis angular rate data. The value is expressed as two's complement.

7.12 OUT_Z_L (2Ch), OUT_Z_H (2Dh)

Z-axis angular rate data. The value is expressed as two's complement.

7.13 FIFO_CTRL_REG (2Eh)

Table 43. REFERENCE register

FM2	FM1	FM0	WTM4	WTM3	WTM2	WTM1	WTM0

Table 44. REFERENCE register description

FM2-FM0	FIFO mode selection. Default value: 00 (see *Table*)
WTM4-WTM0	FIFO threshold. Watermark level setting

Table 45. FIFO mode configuration

FM2	FM1	FM0	FIFO mode
0	0	0	Bypass mode
0	0	1	FIFO mode
0	1	0	Stream mode
0	1	1	Stream-to-FIFO mode
1	0	0	Bypass-to-Stream mode

7.14 FIFO_SRC_REG (2Fh)

Table 46. FIFO_SRC register

WTM	OVRN	EMPTY	FSS4	FSS3	FSS2	FSS1	FSS0

Table 47. FIFO_SRC register description

WTM	Watermark status. (0: FIFO filling is lower than WTM level; 1: FIFO filling is equal or higher than WTM level)
OVRN	Overrun bit status. (0: FIFO is not completely filled; 1:FIFO is completely filled)

Table 47. FIFO_SRC register description (continued)

EMPTY	FIFO empty bit. (0: FIFO not empty; 1: FIFO empty)
FSS4-FSS1	FIFO stored data level

7.15 INT1_CFG (30h)

Table 48. INT1_CFG register

AND/OR	LIR	ZHIE	ZLIE	YHIE	YLIE	XHIE	XLIE

Table 49. INT1_CFG description

AND/OR	AND/OR combination of Interrupt events. Default value: 0 (0: OR combination of interrupt events 1: AND combination of interrupt events
LIR	Latch Interrupt Request. Default value: 0 (0: interrupt request not latched; 1: interrupt request latched) Cleared by reading INT1_SRC reg.
ZHIE	Enable interrupt generation on Z high event. Default value: 0 (0: disable interrupt request; 1: enable interrupt request on measured accel. value higher than preset threshold)
ZLIE	Enable interrupt generation on Z low event. Default value: 0 (0: disable interrupt request; 1: enable interrupt request on measured accel. value lower than preset threshold)
YHIE	Enable interrupt generation on Y high event. Default value: 0 (0: disable interrupt request; 1: enable interrupt request on measured accel. value higher than preset threshold)
YLIE	Enable interrupt generation on Y low event. Default value: 0 (0: disable interrupt request; 1: enable interrupt request on measured accel. value lower than preset threshold)
XHIE	Enable interrupt generation on X high event. Default value: 0 (0: disable interrupt request; 1: enable interrupt request on measured accel. value higher than preset threshold)
XLIE	Enable interrupt generation on X low event. Default value: 0 (0: disable interrupt request; 1: enable interrupt request on measured accel. value lower than preset threshold)

Configuration register for Interrupt source.

7.16 INT1_SRC (31h)

Table 50. INT1_SRC register

0	IA	ZH	ZL	YH	YL	XH	XL

Table 51. INT1_SRC description

IA	Interrupt active. Default value: 0 (0: no interrupt has been generated; 1: one or more interrupts have been generated)
ZH	Z high. Default value: 0 (0: no interrupt, 1: Z High event has occurred)
ZL	Z low. Default value: 0 (0: no interrupt; 1: Z Low event has occurred)
YH	Y high. Default value: 0 (0: no interrupt, 1: Y High event has occurred)
YL	Y low. Default value: 0 (0: no interrupt, 1: Y Low event has occurred)
XH	X high. Default value: 0 (0: no interrupt, 1: X High event has occurred)
XL	X low. Default value: 0 (0: no interrupt, 1: X Low event has occurred)

Interrupt source register. Read only register.

Reading at this address clears INT1_SRC IA bit (and eventually the interrupt signal on INT1 pin) and allows the refreshment of data in the INT1_SRC register if the latched option was chosen.

7.17 INT1_THS_XH (32h)

Table 52. INT1_THS_XH register

-	THSX14	THSX13	THSX12	THSX11	THSX10	THSX9	THSX8

Table 53. INT1_THS_XH description

THSX14 - THSX9	Interrupt threshold. Default value: 0000 0000

7.18 INT1_THS_XL (33h)

Table 54. INT1_THS_XL register

THSX7	THSX6	THSX5	THSX4	THSX3	THSX2	THSX1	THSX0

Table 55. INT1_THS_XL description

THSX7 - THSX0	Interrupt threshold. Default value: 0000 0000

7.19 INT1_THS_YH (34h)

Table 56. INT1_THS_YH register

-	THSY14	THSY13	THSY12	THSY11	THSY10	THSY9	THSY8

Table 57. INT1_THS_YH description

THSY14 - THSY9	Interrupt threshold. Default value: 0000 0000

7.20 INT1_THS_YL (35h)

Table 58. INT1_THS_YL register

THSR7	THSY6	THSY5	THSY4	THSY3	THSY2	THSY1	THSY0

Table 59. INT1_THS_YL description

THSY7 - THSY0	Interrupt threshold. Default value: 0000 0000

7.21 INT1_THS_ZH (36h)

Table 60. INT1_THS_ZH register

-	THSZ14	THSZ13	THSZ12	THSZ11	THSZ10	THSZ9	THSZ8

Table 61. INT1_THS_ZH description

THSZ14 - THSZ9	Interrupt threshold. Default value: 0000 0000

7.22 INT1_THS_ZL (37h)

Table 62. INT1_THS_ZL register

THSZ7	THSZ6	THSZ5	THSZ4	THSZ3	THSZ2	THSZ1	THSZ0

Table 63. INT1_THS_ZL description

THSZ7 - THSZ0	Interrupt threshold. Default value: 0000 0000

7.23 INT1_DURATION (38h)

Table 64. INT1_DURATION register

WAIT	D6	D5	D4	D3	D2	D1	D0

Table 65. INT1_DURATION description

WAIT	WAIT enable. Default value: 0 (0: disable; 1: enable)
D6 - D0	Duration value. Default value: 000 0000

D6 - D0 bits set the minimum duration of the Interrupt event to be recognized. Duration steps and maximum values depend on the ODR chosen.

WAIT bit has the following meaning:

Wait ='0': the interrupt falls immediately if signal crosses the selected threshold

Wait ='1': if signal crosses the selected threshold, the interrupt falls only after the duration has counted number of samples at the selected data rate, written into the duration counter register.

Figure 20. Wait disabled

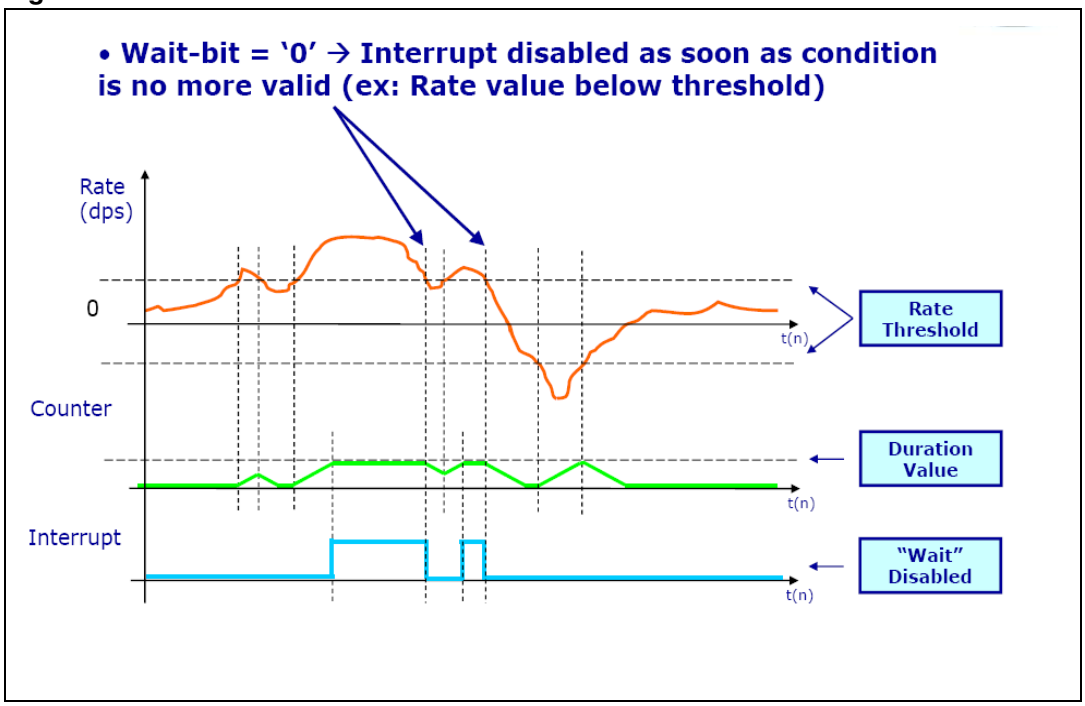
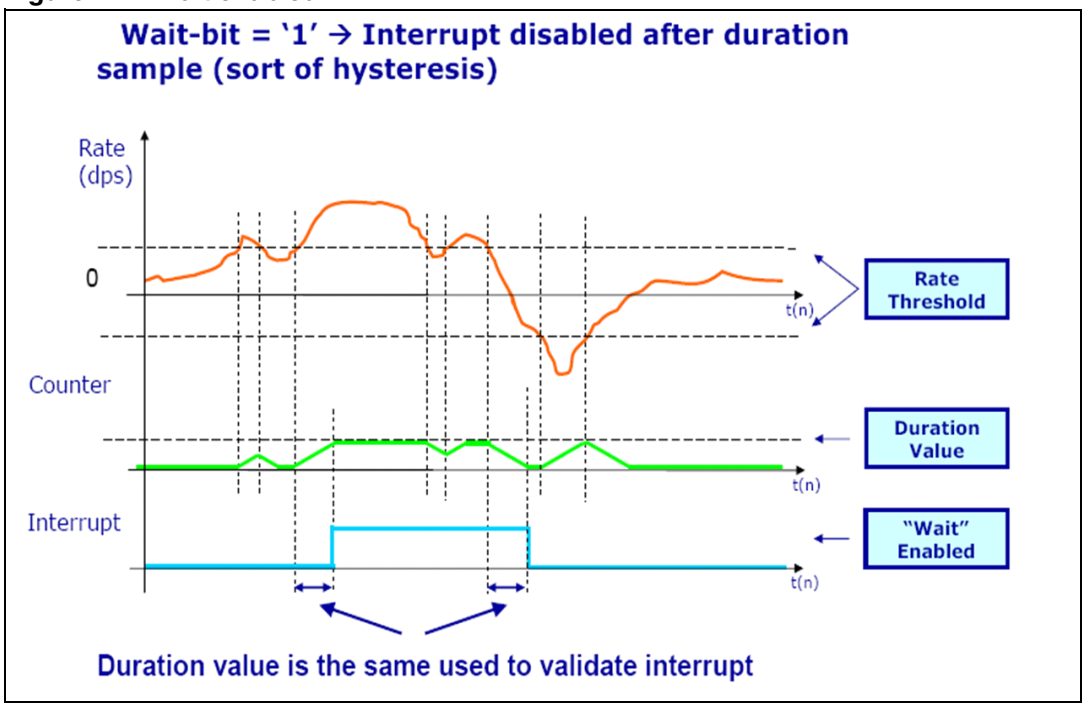


Figure 21. Wait enabled



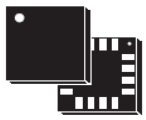
8 Package information

In order to meet environmental requirements, ST offers these devices in different grades of ECOPACK® packages, depending on their level of environmental compliance. ECOPACK® specifications, grade definitions and product status are available at www.st.com. ECOPACK is an ST trademark.

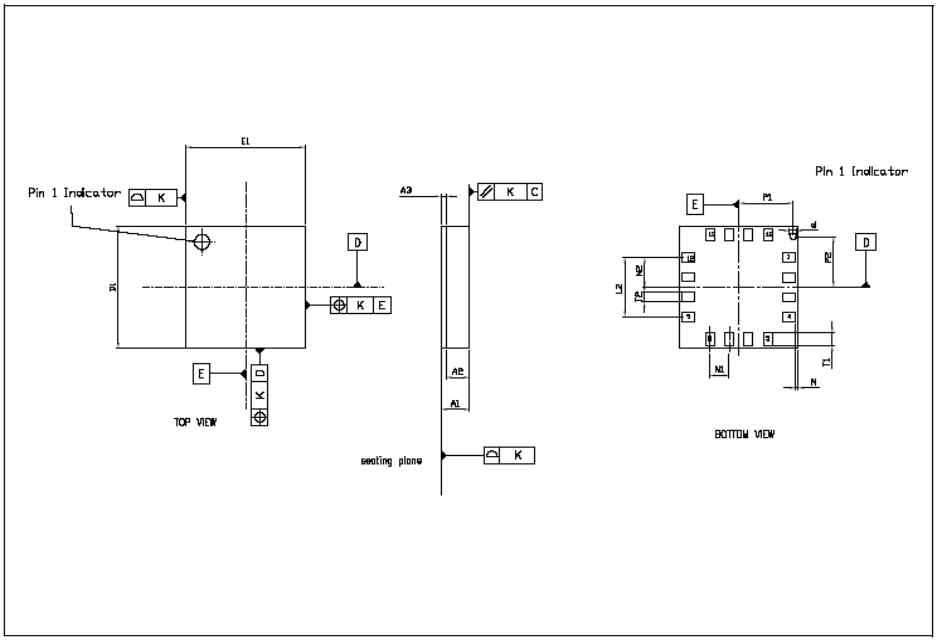
Figure 22. LGA-16: mechanical data and package dimensions

Dimensions			
Ref.	mm		
	Min.	Typ.	Max.
A1			1.100
A2		0.855	
A3		0.200	
d		0.300	
D1	3.850	4.000	4.150
E1	3.850	4.000	4.150
L2		1.950	
M		0.100	
N1		0.650	
N2		0.975	
P1		1.750	
P2		1.525	
T1		0.400	
T2		0.300	
k		0.050	

Outline and mechanical data



LGA 16L (4x4x1.1mm)
Land Grid Array Package



9 Revision history

Table 66. Document revision history

Date	Revision	Changes
01-Apr-2010	1	Initial release.
03-Sep-2010	2	Complete datasheet review.
22-Dec-2010	3	Inserted *Section 6: Output register mapping* and *Section 7: Register description*.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

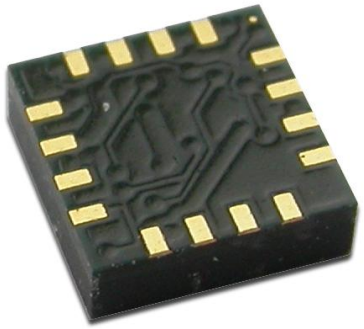
www.st.com

3-Axis Digital Compass IC HMC5883L

Honeywell

Advanced Information

The Honeywell HMC5883L is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as low-cost compassing and magnetometry. The HMC5883L includes our state-of-the-art, high-resolution HMC118X series magneto-resistive sensors plus an ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, and a 12-bit ADC that enables 1° to 2° compass heading accuracy. The I^2C serial bus allows for easy interface. The HMC5883L is a 3.0x3.0x0.9mm surface mount 16-pin leadless chip carrier (LCC). Applications for the HMC5883L include Mobile Phones, Netbooks, Consumer Electronics, Auto Navigation Systems, and Personal Navigation Devices.



The HMC5883L utilizes Honeywell's Anisotropic Magnetoresistive (AMR) technology that provides advantages over other magnetic sensor technologies. These anisotropic, directional sensors feature precision in-axis sensitivity and linearity. These sensors' solid-state construction with very low cross-axis sensitivity is designed to measure both the direction and the magnitude of Earth's magnetic fields, from milli-gauss to 8 gauss. Honeywell's Magnetic Sensors are among the most sensitive and reliable low-field sensors in the industry.

FEATURES	BENEFITS
3-Axis Magnetoresistive Sensors and ASIC in a 3.0x3.0x0.9mm LCC Surface Mount Package	Small Size for Highly Integrated Products. Just Add a Micro-Controller Interface, Plus Two External SMT Capacitors Designed for High Volume, Cost Sensitive OEM Designs Easy to Assemble & Compatible with High Speed SMT Assembly
12-Bit ADC Coupled with Low Noise AMR Sensors Achieves 2 milli-gauss Field Resolution in ±8 Gauss Fields	Enables 1° to 2° Degree Compass Heading Accuracy
Built-In Self Test	Enables Low-Cost Functionality Test after Assembly in Production
Low Voltage Operations (2.16 to 3.6V) and Low Power Consumption (100 µA)	Compatible for Battery Powered Applications
Built-In Strap Drive Circuits	Set/Reset and Offset Strap Drivers for Degaussing, Self Test, and Offset Compensation
I^2C Digital Interface	Popular Two-Wire Serial Data Interface for Consumer Electronics
Lead Free Package Construction	RoHS Compliance
Wide Magnetic Field Range (+/-8 Oe)	Sensors Can Be Used in Strong Magnetic Field Environments with a 1° to 2° Degree Compass Heading Accuracy
Software and Algorithm Support Available	Compassing Heading, Hard Iron, Soft Iron, and Auto Calibration Libraries Available
Fast 160 Hz Maximum Output Rate	Enables Pedestrian Navigation and LBS Applications

SPECIFICATIONS (* Tested at 25°C except stated otherwise.)

Characteristics	Conditions*	Min	Typ	Max	Units
Power Supply					
Supply Voltage	VDD Referenced to AGND VDDIO Referenced to DGND	2.16 1.71	2.5 1.8	3.6 VDD+0.1	Volts Volts
Average Current Draw	Idle Mode Measurement Mode (7.5 Hz ODR; No measurement average, MA1:MA0 = 00) VDD = 2.5V, VDDIO = 1.8V (Dual Supply) VDD = VDDIO = 2.5V (Single Supply)	- -	2 100	- -	µA µA
Performance					
Field Range	Full scale (FS)	-8		+8	gauss
Mag Dynamic Range	3-bit gain control	±1		±8	gauss
Sensitivity (Gain)	VDD=3.0V, GN=0 to 7, 12-bit ADC	230		1370	LSb/gauss
Digital Resolution	VDD=3.0V, GN=0 to 7, 1-LSb, 12-bit ADC	0.73		4.35	milli-gauss
Noise Floor (Field Resolution)	VDD=3.0V, GN=0, No measurement average, Standard Deviation 100 samples (See typical performance graphs below)		2		milli-gauss
Linearity	±2.0 gauss input range			0.1	±% FS
Hysteresis	±2.0 gauss input range		±25		ppm
Cross-Axis Sensitivity	Test Conditions: Cross field = 0.5 gauss, Happlied = ±3 gauss		±0.2%		%FS/gauss
Output Rate (ODR)	Continuous Measurment Mode Single Measurement Mode	0.75		75 160	Hz Hz
Measurement Period	From receiving command to data ready		6		ms
Turn-on Time	Ready for I2C commands Analog Circuit Ready for Measurements		200 50		µs ms
Gain Tolerance	All gain/dynamic range settings		±5		%
I^2C Address	8-bit read address 8-bit write address		0x3D 0x3C		hex hex
I^2C Rate	Controlled by I^2C Master			400	kHz
I^2C Hysteresis	Hysteresis of Schmitt trigger inputs on SCL and SDA - Fall (VDDIO=1.8V) Rise (VDDIO=1.8V)		0.2*VDDIO 0.8*VDDIO		Volts Volts
Self Test	X & Y Axes Z Axis		±1.16 ±1.08		gauss
	X & Y & Z Axes (GN=5) Positive Bias X & Y & Z Axes (GN=5) Negative Bias	243 -575		575 -243	LSb
Sensitivity Tempco	T_A = -40 to 125°C, Uncompensated Output		-0.3		%/°C
General					
ESD Voltage	Human Body Model (all pins) Charged Device Model (all pins)			2000 750	Volts
Operating Temperature	Ambient	-30		85	°C
Storage Temperature	Ambient, unbiased	-40		125	°C

Characteristics	Conditions*	Min	Typ	Max	Units
Reflow Classification	MSL 3, 260 °C Peak Temperature				
Package Size	Length and Width	2.85	3.00	3.15	mm
Package Height		0.8	0.9	1.0	mm
Package Weight			18		mg

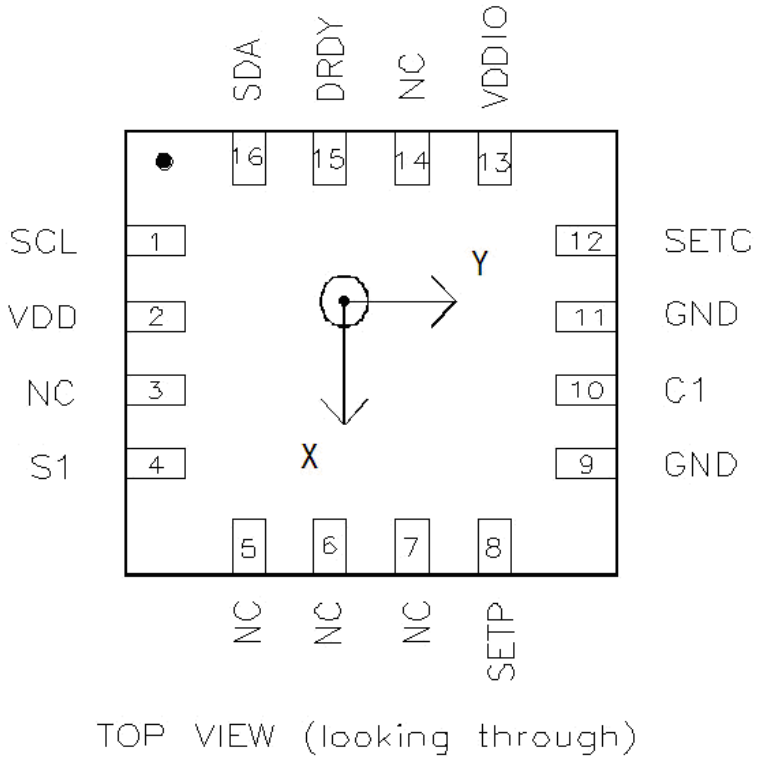
Absolute Maximum Ratings (* Tested at 25°C except stated otherwise.)

Characteristics	Min	Max	Units
Supply Voltage VDD	-0.3	4.8	Volts
Supply Voltage VDDIO	-0.3	4.8	Volts

PIN CONFIGURATIONS

Pin	Name	Description
1	SCL	Serial Clock – I^2C Master/Slave Clock
2	VDD	Power Supply (2.16V to 3.6V)
3	NC	Not to be Connected
4	S1	Tie to VDDIO
5	NC	Not to be Connected
6	NC	Not to be Connected
7	NC	Not to be Connected
8	SETP	Set/Reset Strap Positive – S/R Capacitor (C2) Connection
9	GND	Supply Ground
10	C1	Reservoir Capacitor (C1) Connection
11	GND	Supply Ground
12	SETC	S/R Capacitor (C2) Connection – Driver Side
13	VDDIO	IO Power Supply (1.71V to VDD)
14	NC	Not to be Connected
15	DRDY	Data Ready, Interrupt Pin. Internally pulled high. Optional connection. Low for 250 µsec when data is placed in the data output registers.
16	SDA	Serial Data – I^2C Master/Slave Data

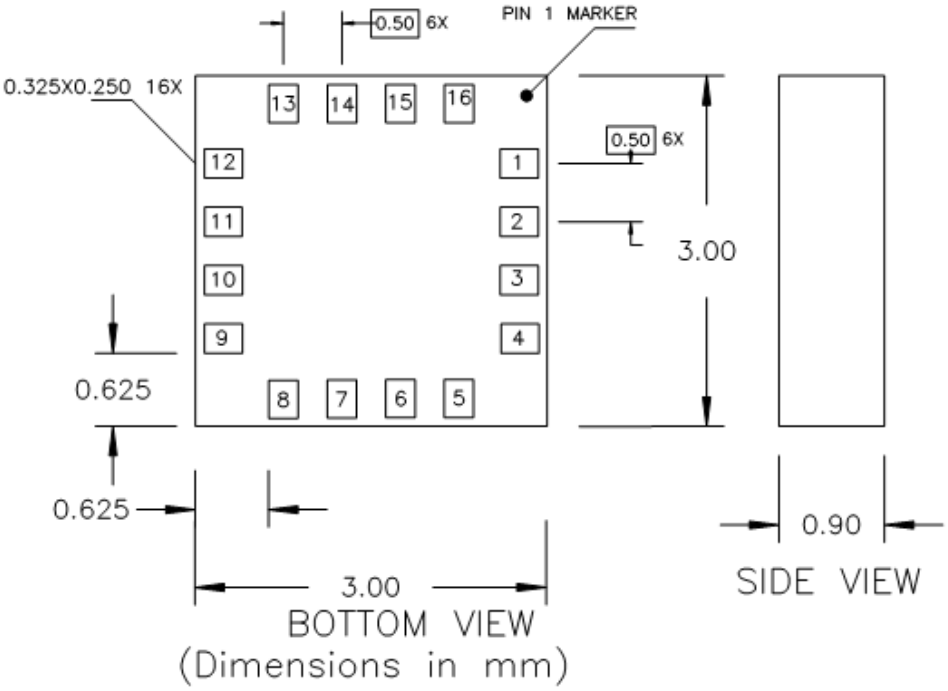
Table 1: Pin Configurations



Arrow indicates direction of magnetic field that generates a positive output reading in Normal Measurement configuration.

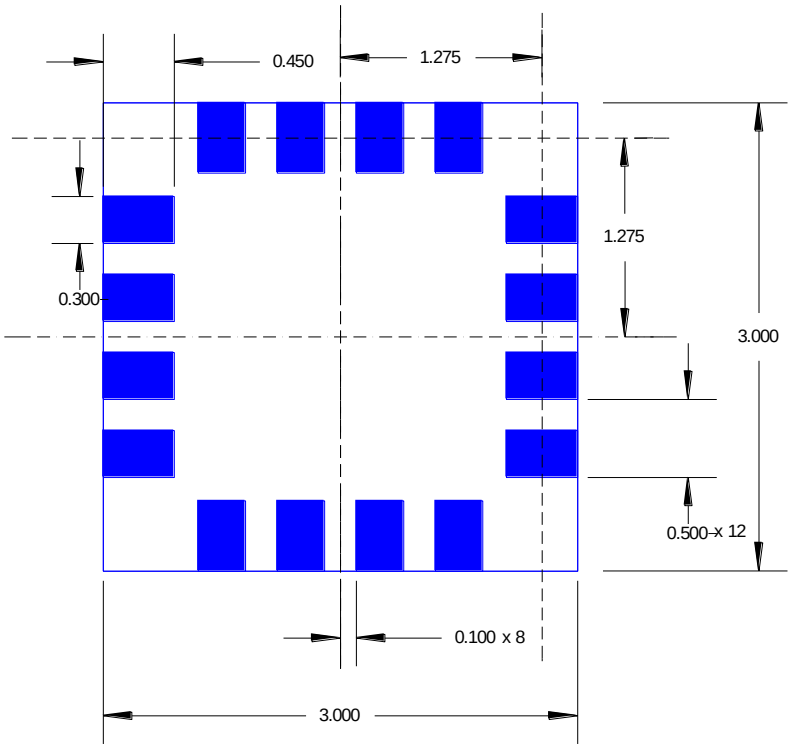
PACKAGE OUTLINES

PACKAGE DRAWING HMC5883L (16-PIN LPCC, dimensions in millimeters)



MOUNTING CONSIDERATIONS

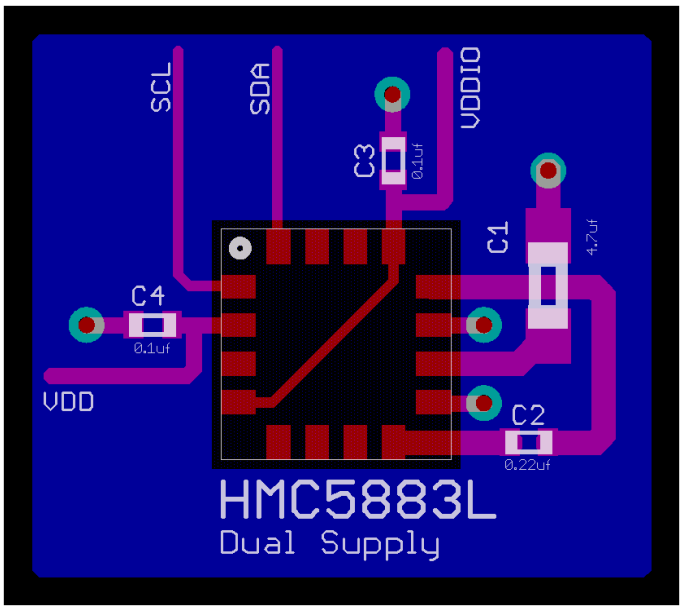
The following is the recommend printed circuit board (PCB) footprint for the HMC5883L.

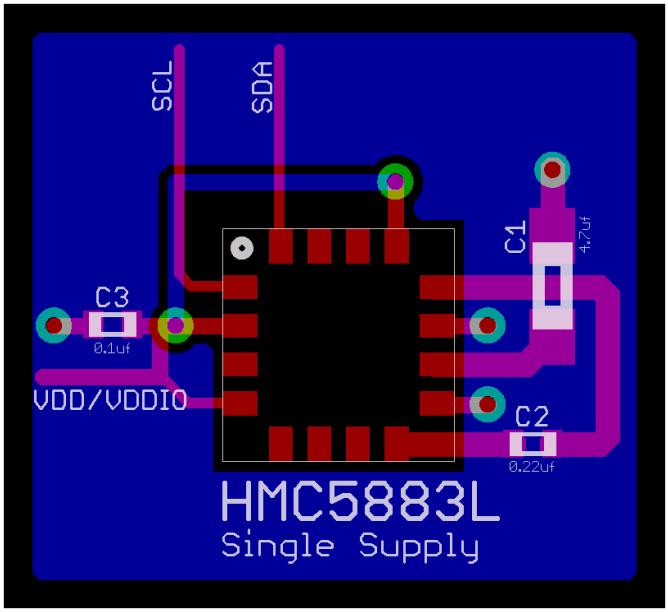


HMC5883 Land Pad Pattern
(All dimensions are in mm)

LAYOUT CONSIDERATIONS

Besides keeping all components that may contain ferrous materials (nickel, etc.) away from the sensor on both sides of the PCB, it is also recommended that there is no conducting copper under/near the sensor in any of the PCB layers. See recommended layout below. Notice that the one trace under the sensor in the dual supply mode is not expected to carry active current since it is for pin 4 pull-up to VDDIO. Power and ground planes are removed under the sensor to minimize possible source of magnetic noise. For best results, use non-ferrous materials for all exposed copper coding.





PCB Pad Definition and Traces

The HMC5883L is a fine pitch LCC package. Refer to previous figure for recommended PCB footprint for proper package centering. Size the traces between the HMC5883L and the external capacitors (C1 and C2) to handle the 1 ampere peak current pulses with low voltage drop on the traces.

Stencil Design and Solder Paste

A 4 mil stencil and 100% paste coverage is recommended for the electrical contact pads.

Reflow Assembly

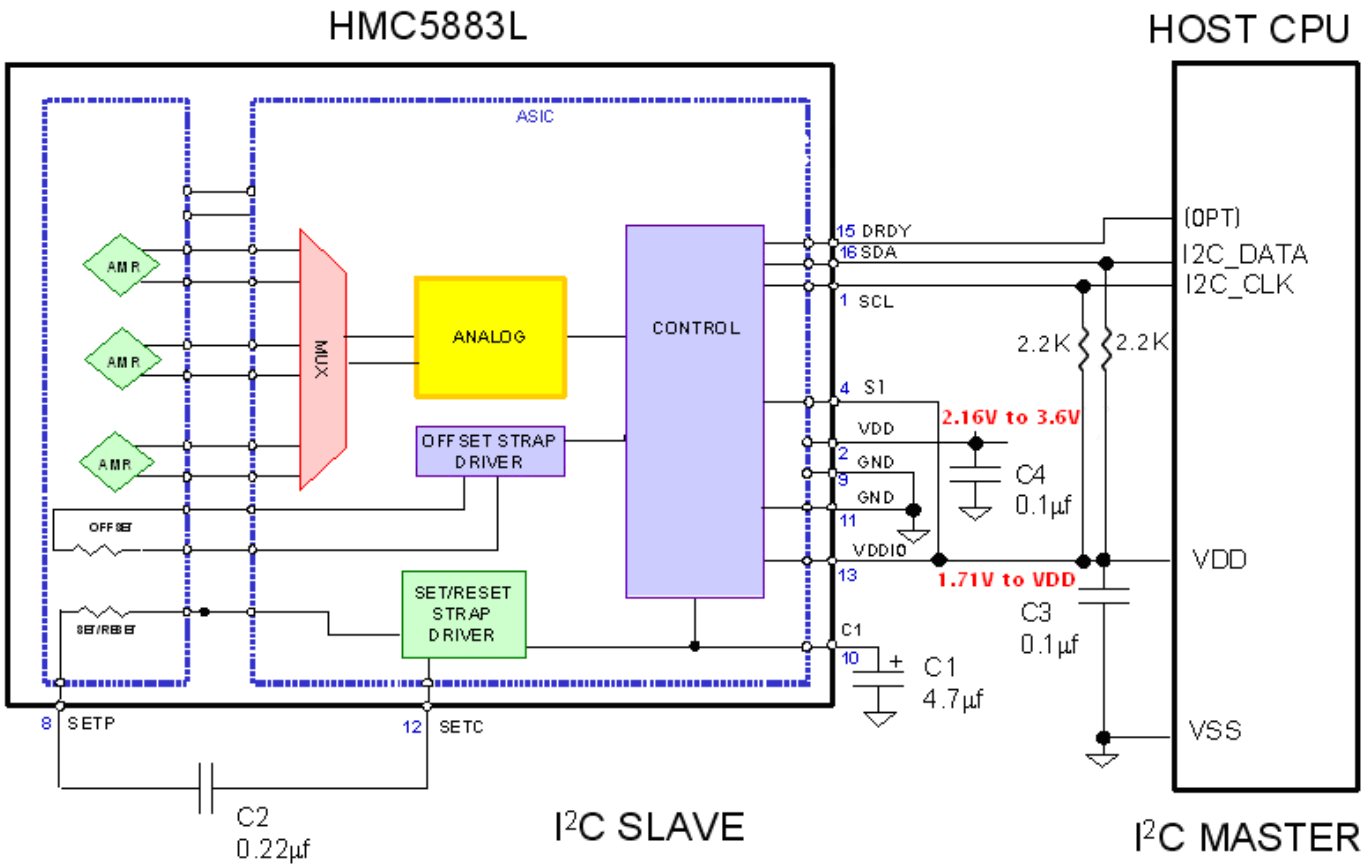
This device is classified as MSL 3 with 260°C peak reflow temperature. A baking process (125°C, 24 hrs) is required if device is not kept continuously in a dry (< 10% RH) environment before assembly. No special reflow profile is required for HMC5883L, which is compatible with lead eutectic and lead-free solder paste reflow profiles. Honeywell recommends adherence to solder paste manufacturer's guidelines. Hand soldering is not recommended. Built-in self test can be used to verify device functionalities after assembly.

External Capacitors

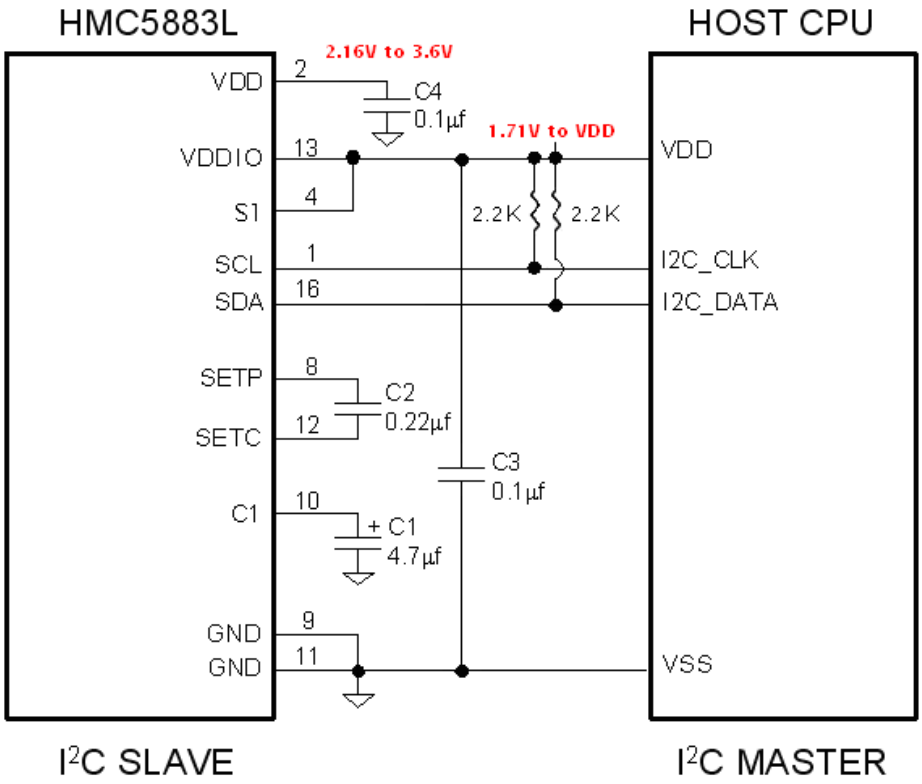
The two external capacitors should be ceramic type construction with low ESR characteristics. The exact ESR values are not critical but values less than 200 milli-ohms are recommended. Reservoir capacitor C1 is nominally 4.7 µF in capacitance, with the set/reset capacitor C2 nominally 0.22 µF in capacitance. Low ESR characteristics may not be in many small SMT ceramic capacitors (0402), so be prepared to up-size the capacitors to gain Low ESR characteristics.

INTERNAL SCHEMATIC DIAGRAM

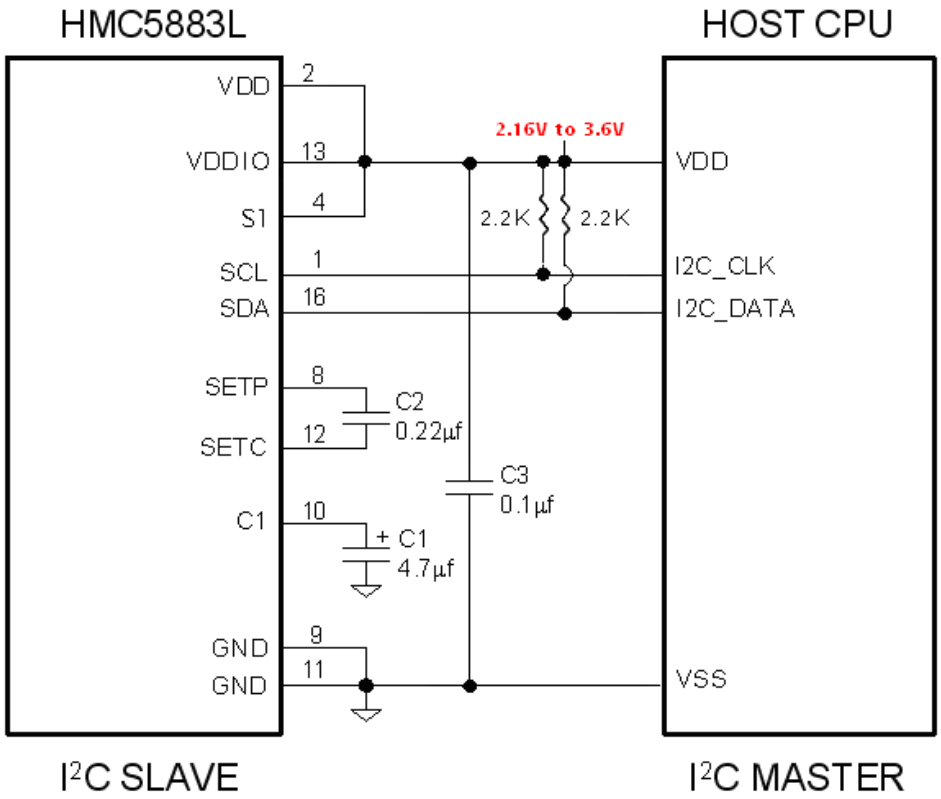
HMC5883L



DUAL SUPPLY REFERENCE DESIGN



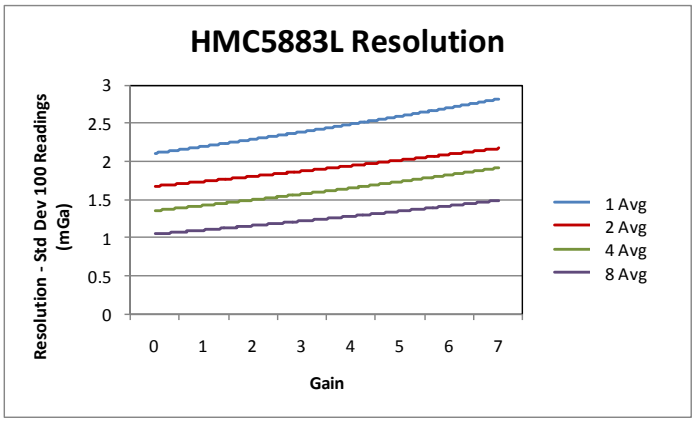
SINGLE SUPPLY REFERENCE DESIGN

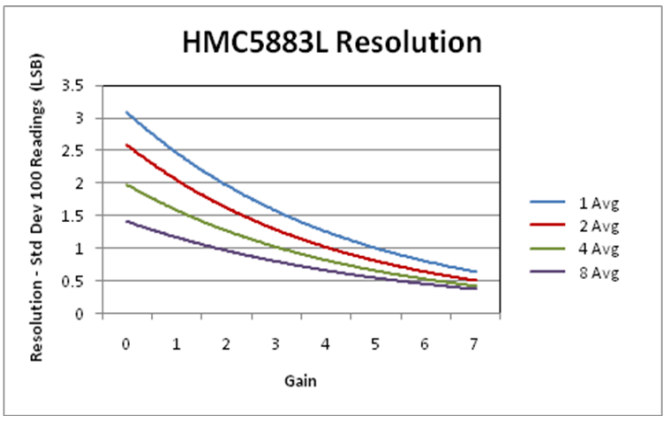


PERFORMANCE

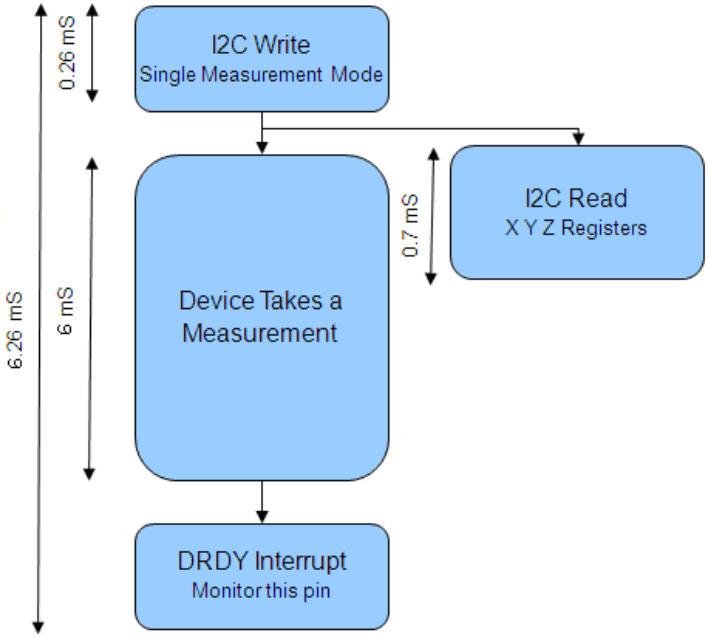
The following graph(s) highlight HMC5883L's performance.

Typical Noise Floor (Field Resolution)





Typical Measurement Period in Single-Measurement Mode



** Monitoring of the DRDY Interrupt pin is only required if maximum output rate is desired.*

BASIC DEVICE OPERATION

Anisotropic Magneto-Resistive Sensors

The Honeywell HMC5883L magnetoresistive sensor circuit is a trio of sensors and application specific support circuits to measure magnetic fields. With power supply applied, the sensor converts any incident magnetic field in the sensitive axis directions to a differential voltage output. The magnetoresistive sensors are made of a nickel-iron (Permalloy) thin-film and patterned as a resistive strip element. In the presence of a magnetic field, a change in the bridge resistive elements causes a corresponding change in voltage across the bridge outputs.

These resistive elements are aligned together to have a common sensitive axis (indicated by arrows in the pinout diagram) that will provide positive voltage change with magnetic fields increasing in the sensitive direction. Because the output is only proportional to the magnetic field component along its axis, additional sensor bridges are placed at orthogonal directions to permit accurate measurement of magnetic field in any orientation.

Self Test

To check the HMC5883L for proper operation, a self test feature in incorporated in which the sensor is internally excited with a nominal magnetic field (in either positive or negative bias configuration). This field is then measured and reported. This function is enabled and the polarity is set by bits MS[n] in the configuration register A. An internal current source generates DC current (about 10 mA) from the VDD supply. This DC current is applied to the offset straps of the magneto-resistive sensor, which creates an artificial magnetic field bias on the sensor. The difference of this measurement and the measurement of the ambient field will be put in the data output register for each of the three axes. By using this built-in function, the manufacturer can quickly verify the sensor's full functionality after the assembly without additional test setup. The self test results can also be used to estimate/compensate the sensor's sensitivity drift due to temperature.

For each "self test measurement", the ASIC:

1. Sends a "Set" pulse
2. Takes one measurement (M1)
3. Sends the (~10 mA) offset current to generate the (~1.1 Gauss) offset field and takes another measurement (M2)
4. Puts the difference of the two measurements in sensor's data output register:

   Output = [M2 – M1] (i.e. output = offset field only)

See SELF TEST OPERATION section later in this datasheet for additional details.

Power Management

This device has two different domains of power supply. The first one is VDD that is the power supply for internal operations and the second one is VDDIO that is dedicated to IO interface. It is possible to work with VDDIO equal to VDD; Single Supply mode, or with VDDIO lower than VDD allowing HMC5883L to be compatible with other devices on board.

I^2C Interface

Control of this device is carried out via the I^2C bus. This device will be connected to this bus as a slave device under the control of a master device, such as the processor.

This device is compliant with *I^2C-Bus Specification*, document number: 9398 393 40011. As an I^2C compatible device, this device has a 7-bit serial address and supports I^2C protocols. This device supports standard and fast modes, 100kHz and 400kHz, respectively, but does not support the high speed mode (Hs). External pull-up resistors are required to support these standard and fast speed modes.

Activities required by the master (register read and write) have priority over internal activities, such as the measurement. The purpose of this priority is to not keep the master waiting and the I^2C bus engaged for longer than necessary.

Internal Clock

The device has an internal clock for internal digital logic functions and timing management. This clock is not available to external usage.

H-Bridge for Set/Reset Strap Drive

The ASIC contains large switching FETs capable of delivering a large but brief pulse to the Set/Reset strap of the sensor. This strap is largely a resistive load. There is no need for an external Set/Reset circuit. The controlling of the Set/Reset function is done automatically by the ASIC for each measurement. One half of the difference from the measurements taken after a set pulse and after a reset pulse will be put in the data output register for each of the three axes. By doing so, the sensor's internal offset and its temperature dependence is removed/cancelled for all measurements. The set/reset pulses also effectively remove the past magnetic history (magnetism) in the sensor, if any.

For each "measurement", the ASIC:

1. Sends a "Set" pulse
2. Takes one measurement (Mset)
3. Sends a "Reset" pulse
4. Takes another measurement (Mreset)
5. Puts the following result in sensor's data output register:

Output = [Mset – Mreset] / 2

Charge Current Limit

The current that reservoir capacitor (C1) can draw when charging is limited for both single supply and dual supply configurations. This prevents drawing down the supply voltage (VDD).

MODES OF OPERATION

This device has several operating modes whose primary purpose is power management and is controlled by the Mode Register. This section describes these modes.

Continuous-Measurement Mode

During continuous-measurement mode, the device continuously makes measurements, at user selectable rate, and places measured data in data output registers. Data can be re-read from the data output registers if necessary; however, if the master does not ensure that the data register is accessed before the completion of the next measurement, the data output registers are updated with the new measurement. To conserve current between measurements, the device is placed in a state similar to idle mode, but the Mode Register is not changed to Idle Mode. That is, MD[n] bits are unchanged. Settings in the Configuration Register A affect the data output rate (bits DO[n]), the measurement configuration (bits MS[n]), when in continuous-measurement mode. All registers maintain values while in continuous-measurement mode. The I^2C bus is enabled for use by other devices on the network in while continuous-measurement mode.

Single-Measurement Mode

This is the default power-up mode. During single-measurement mode, the device makes a single measurement and places the measured data in data output registers. After the measurement is complete and output data registers are updated, the device is placed in idle mode, and the Mode Register *is* changed to idle mode by setting MD[n] bits. Settings in the configuration register affect the measurement configuration (bits MS[n])when in single-measurement mode. All registers maintain values while in single-measurement mode. The I^2C bus is enabled for use by other devices on the network while in single-measurement mode.

Idle Mode

During this mode the device is accessible through the I^2C bus, but major sources of power consumption are disabled, such as, but not limited to, the ADC, the amplifier, and the sensor bias current. All registers maintain values while in idle mode. The I^2C bus is enabled for use by other devices on the network while in idle mode.

REGISTERS

This device is controlled and configured via a number of on-chip registers, which are described in this section. In the following descriptions, *set* implies a logic 1, and *reset* or *clear* implies a logic 0, unless stated otherwise.

Register List
The table below lists the registers and their access. All address locations are 8 bits.

Address Location	Name	Access
00	Configuration Register A	Read/Write
01	Configuration Register B	Read/Write
02	Mode Register	Read/Write
03	Data Output X MSB Register	Read
04	Data Output X LSB Register	Read
05	Data Output Z MSB Register	Read
06	Data Output Z LSB Register	Read
07	Data Output Y MSB Register	Read
08	Data Output Y LSB Register	Read
09	Status Register	Read
10	Identification Register A	Read
11	Identification Register B	Read
12	Identification Register C	Read

Table2: Register List

Register Access

This section describes the process of reading from and writing to this device. The devices uses an address pointer to indicate which register location is to be read from or written to. These pointer locations are sent from the master to this slave device and succeed the 7-bit address (0x1E) plus 1 bit read/write identifier, i.e. 0x3D for read and 0x3C for write.

To minimize the communication between the master and this device, the address pointer updated automatically without master intervention. The register pointer will be incremented by 1 automatically after the current register has been read successfully.

The address pointer value itself cannot be read via the I^2C bus.
Any attempt to read an invalid address location returns 0's, and any write to an invalid address location or an undefined bit within a valid address location is ignored by this device.

To move the address pointer to a random register location, first issue a "write" to that register location with no data byte following the commend. For example, to move the address pointer to register 10, send 0x3C 0x0A.

Configuration Register A

The configuration register is used to configure the device for setting the data output rate and measurement configuration. CRA0 through CRA7 indicate bit locations, with *CRA* denoting the bits that are in the configuration register. CRA7 denotes the first bit of the data stream. The number in parenthesis indicates the default value of that bit.CRA default is 0x10.

CRA7	CRA6	CRA5	CRA4	CRA3	CRA2	CRA1	CRA0
(0)	MA1(0)	MA0(0)	DO2 (1)	DO1 (0)	DO0 (0)	MS1 (0)	MS0 (0)

Table 3: Configuration Register A

Location	Name	Description
CRA7	CRA7	Bit CRA7 is reserved for future function. Set to 0 when configuring CRA.
CRA6 to CRA5	MA1 to MA0	Select number of samples averaged (1 to 8) per measurement output. 00 = 1(Default); 01 = 2; 10 = 4; 11 = 8
CRA4 to CRA2	DO2 to DO0	Data Output Rate Bits. These bits set the rate at which data is written to all three data output registers.
CRA1 to CRA0	MS1 to MS0	Measurement Configuration Bits. These bits define the measurement flow of the device, specifically whether or not to incorporate an applied bias into the measurement.

Table 4: Configuration Register A Bit Designations

The Table below shows all selectable output rates in continuous measurement mode. All three channels shall be measured within a given output rate. Other output rates with maximum rate of 160 Hz can be achieved by monitoring DRDY interrupt pin in single measurement mode.

DO2	DO1	DO0	Typical Data Output Rate (Hz)
0	0	0	0.75
0	0	1	1.5
0	1	0	3
0	1	1	7.5
1	0	0	15 (Default)
1	0	1	30
1	1	0	75
1	1	1	Reserved

Table 5: Data Output Rates

MS1	MS0	Measurement Mode
0	0	Normal measurement configuration (Default). In normal measurement configuration the device follows normal measurement flow. The positive and negative pins of the resistive load are left floating and high impedance.
0	1	Positive bias configuration for X, Y, and Z axes. In this configuration, a positive current is forced across the resistive load for all three axes.
1	0	Negative bias configuration for X, Y and Z axes. In this configuration, a negative current is forced across the resistive load for all three axes..
1	1	This configuration is reserved.

Table 6: Measurement Modes

Configuration Register B

The configuration register B for setting the device gain. CRB0 through CRB7 indicate bit locations, with *CRB* denoting the bits that are in the configuration register. CRB7 denotes the first bit of the data stream. The number in parenthesis indicates the default value of that bit. CRB default is 0x20.

CRB7	CRB6	CRB5	CRB4	CRB3	CRB2	CRB1	CRB0
GN2 (0)	GN1 (0)	GN0 (1)	(0)	(0)	(0)	(0)	(0)

Table 7: Configuration B Register

Location	Name	Description
CRB7 to CRB5	GN2 to GN0	Gain Configuration Bits. These bits configure the gain for the device. The gain configuration is common for all channels.
CRB4 to CRB0	0	These bits must be cleared for correct operation.

Table 8: Configuration Register B Bit Designations

The table below shows nominal gain settings. Use the “Gain” column to convert counts to Gauss. The “Digital Resolution” column is the theoretical value in term of milli-Gauss per count (LSb) which is the inverse of the values in the “Gain” column. The effective resolution of the usable signal also depends on the noise floor of the system, i.e.

Effective Resolution = Max (Digital Resolution, Noise Floor)

Choose a lower gain value (higher GN#) when total field strength causes overflow in one of the data output registers (saturation). Note that the very first measurement after a gain change maintains the same gain as the previous setting. **The new gain setting is effective from the second measurement and on.**

GN2	GN1	GN0	Recommended Sensor Field Range	Gain (LSb/ Gauss)	Digital Resolution (mG/LSb)	Output Range
0	0	0	± 0.88 Ga	1370	0.73	0xF800–0x07FF (-2048–2047)
0	0	1	± 1.3 Ga	1090 (default)	0.92	0xF800–0x07FF (-2048–2047)
0	1	0	± 1.9 Ga	820	1.22	0xF800–0x07FF (-2048–2047)
0	1	1	± 2.5 Ga	660	1.52	0xF800–0x07FF (-2048–2047)
1	0	0	± 4.0 Ga	440	2.27	0xF800–0x07FF (-2048–2047)
1	0	1	± 4.7 Ga	390	2.56	0xF800–0x07FF (-2048–2047)
1	1	0	± 5.6 Ga	330	3.03	0xF800–0x07FF (-2048–2047)
1	1	1	± 8.1 Ga	230	4.35	0xF800–0x07FF (-2048–2047)

Table 9: Gain Settings

Mode Register

The mode register is an 8-bit register from which data can be read or to which data can be written. This register is used to select the operating mode of the device. MR0 through MR7 indicate bit locations, with *MR* denoting the bits that are in the mode register. MR7 denotes the first bit of the data stream. The number in parenthesis indicates the default value of that bit. Mode register default is 0x01.

MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
HS(0)	(0)	(0)	(0)	(0)	(0)	MD1 (0)	MD0 (1)

Table 10: Mode Register

Location	Name	Description
MR7 to MR2	HS	Set this pin to enable High Speed I2C, 3400kHz.
MR1 to MR0	MD1 to MD0	Mode Select Bits. These bits select the operation mode of this device.

Table 11: Mode Register Bit Designations

MD1	MD0	Operating Mode
0	0	Continuous-Measurement Mode. In continuous-measurement mode, the device continuously performs measurements and places the result in the data register. RDY goes high when new data is placed in all three registers. After a power-on or a write to the mode or configuration register, the first measurement set is available from all three data output registers after a period of $2/f_{DO}$ and subsequent measurements are available at a frequency of f_{DO}, where f_{DO} is the frequency of data output.
0	1	Single-Measurement Mode (Default). When single-measurement mode is selected, device performs a single measurement, sets RDY high and returned to idle mode. Mode register returns to idle mode bit values. The measurement remains in the data output register and RDY remains high until the data output register is read or another measurement is performed.
1	0	Idle Mode. Device is placed in idle mode.
1	1	Idle Mode. Device is placed in idle mode.

Table 12: Operating Modes

Data Output X Registers A and B

The data output X registers are two 8-bit registers, data output register A and data output register B. These registers store the measurement result from channel X. Data output X register A contains the MSB from the measurement result, and data output X register B contains the LSB from the measurement result. The value stored in these two registers is a 16-bit value in 2's complement form, whose range is 0xF800 to 0x07FF. DXRA0 through DXRA7 and DXRB0 through DXRB7 indicate bit locations, with *DXRA* and *DXRB* denoting the bits that are in the data output X registers. DXRA7 and DXRB7 denote the first bit of the data stream. The number in parenthesis indicates the default value of that bit.

In the event the ADC reading overflows or underflows for the given channel, or if there is a math overflow during the bias measurement, this data register will contain the value -4096. This register value will clear when after the next valid measurement is made.

DXRA7	**DXRA6**	**DXRA5**	**DXRA4**	**DXRA3**	**DXRA2**	**DXRA1**	**DXRA0**
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
DXRB7	**DXRB6**	**DXRB5**	**DXRB4**	**DXRB3**	**DXRB2**	**DXRB1**	**DXRB0**
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)

Table 13: Data Output X Registers A and B

Data Output Y Registers A and B

The data output Y registers are two 8-bit registers, data output register A and data output register B. These registers store the measurement result from channel Y. Data output Y register A contains the MSB from the measurement result, and data output Y register B contains the LSB from the measurement result. The value stored in these two registers is a 16-bit value in 2's complement form, whose range is 0xF800 to 0x07FF. DYRA0 through DYRA7 and DYRB0 through DYRB7 indicate bit locations, with *DYRA* and *DYRB* denoting the bits that are in the data output Y registers. DYRA7 and DYRB7 denote the first bit of the data stream. The number in parenthesis indicates the default value of that bit.

In the event the ADC reading overflows or underflows for the given channel, or if there is a math overflow during the bias measurement, this data register will contain the value -4096. This register value will clear when after the next valid measurement is made.

DYRA7	**DYRA6**	**DYRA5**	**DYRA4**	**DYRA3**	**DYRA2**	**DYRA1**	**DYRA0**
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
DYRB7	**DYRB6**	**DYRB5**	**DYRB4**	**DYRB3**	**DYRB2**	**DYRB1**	**DYRB0**
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)

Table 14: Data Output Y Registers A and B

Data Output Z Registers A and B

The data output Z registers are two 8-bit registers, data output register A and data output register B. These registers store the measurement result from channel Z. Data output Z register A contains the MSB from the measurement result, and data output Z register B contains the LSB from the measurement result. The value stored in these two registers is a 16-bit value in 2's complement form, whose range is 0xF800 to 0x07FF. DZRA0 through DZRA7 and DZRB0 through DZRB7 indicate bit locations, with *DZRA* and *DZRB* denoting the bits that are in the data output Z registers. DZRA7 and DZRB7 denote the first bit of the data stream. The number in parenthesis indicates the default value of that bit.

In the event the ADC reading overflows or underflows for the given channel, or if there is a math overflow during the bias measurement, this data register will contain the value -4096. This register value will clear when after the next valid measurement is made.

DZRA7	DZRA6	DZRA5	DZRA4	DZRA3	DZRA2	DZRA1	DZRA0
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
DZRB7	**DZRB6**	**DZRB5**	**DZRB4**	**DZRB3**	**DZRB2**	**DZRB1**	**DZRB0**
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)

Table 15: Data Output Z Registers A and B

Data Output Register Operation

When one or more of the output registers are read, new data cannot be placed in any of the output data registers until all six data output registers are read. This requirement also impacts DRDY and RDY, which cannot be cleared until new data is placed in all the output registers.

Status Register

The status register is an 8-bit read-only register. This register is used to indicate device status. SR0 through SR7 indicate bit locations, with *SR* denoting the bits that are in the status register. SR7 denotes the first bit of the data stream.

SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0
(0)	(0)	(0)	(0)	(0)	(0)	LOCK (0)	RDY(0)

Table 16: Status Register

Location	Name	Description
SR7 to SR2	0	These bits are reserved.
SR1	LOCK	Data output register lock. This bit is set when: 1.some but not all for of the six data output registers have been read, 2. Mode register has been read. When this bit is set, the six data output registers are locked and any new data will not be placed in these register until one of these conditions are met: 1.all six bytes have been read, 2. the mode register is changed, 3. the measurement configuration (CRA) is changed, 4. power is reset.
SR0	RDY	Ready Bit. Set when data is written to all six data registers. Cleared when device initiates a write to the data output registers and after one or more of the data output registers are written to. When RDY bit is clear it shall remain cleared for a 250 µs. DRDY pin can be used as an alternative to the status register for monitoring the device for measurement data.

Table 17: Status Register Bit Designations

Identification Register A

The identification register A is used to identify the device. IRA0 through IRA7 indicate bit locations, with *IRA* denoting the bits that are in the identification register A. IRA7 denotes the first bit of the data stream. The number in parenthesis indicates the default value of that bit.

The identification value for this device is stored in this register. This is a read-only register.
Register values. ASCII value *H*

IRA7	IRA6	IRA5	IRA4	IRA3	IRA2	IRA1	IRA0
0	1	0	0	1	0	0	0

Table 18: Identification Register A Default Values

Identification Register B

The identification register B is used to identify the device. IRB0 through IRB7 indicate bit locations, with *IRB* denoting the bits that are in the identification register A. IRB7 denotes the first bit of the data stream.

Register values. ASCII value *4*

IRB7	IRB6	IRB5	IRB4	IRB3	IRB2	IRB1	IRB0
0	0	1	1	0	1	0	0

Table 19: Identification Register B Default Values

Identification Register C

The identification register C is used to identify the device. IRC0 through IRC7 indicate bit locations, with *IRC* denoting the bits that are in the identification register A. IRC7 denotes the first bit of the data stream.

Register values. ASCII value *3*

IRC7	IRC6	IRC5	IRC4	IRC3	IRC2	IRC1	IRC0
0	0	1	1	0	0	1	1

Table 20: Identification Register C Default Values

I^{2}C COMMUNICATION PROTOCOL

The HMC5883L communicates via a two-wire I^{2}C bus system as a slave device. The HMC5883L uses a simple protocol with the interface protocol defined by the I^{2}C bus specification, and by this document. The data rate is at the standard-mode 100kbps or 400kbps rates as defined in the I^{2}C Bus Specifications. The bus bit format is an 8-bit Data/Address send and a 1-bit acknowledge bit. The format of the data bytes (payload) shall be case sensitive ASCII characters or binary data to the HMC5883L slave, and binary data returned. Negative binary values will be in two's complement form. The default (factory) HMC5883L 8-bit slave address is 0x3C for write operations, or 0x3D for read operations.

The HMC5883L Serial Clock (SCL) and Serial Data (SDA) lines require resistive pull-ups (Rp) between the master device (usually a host microprocessor) and the HMC5883L. Pull-up resistance values of about 2.2K to 10K ohms are recommended with a nominal VDDIO voltage. Other resistor values may be used as defined in the I^{2}C Bus Specifications that can be tied to VDDIO.

The SCL and SDA lines in this bus specification may be connected to multiple devices. The bus can be a single master to multiple slaves, or it can be a multiple master configuration. All data transfers are initiated by the master device, which is responsible for generating the clock signal, and the data transfers are 8 bit long. All devices are addressed by I^{2}C's unique 7-bit address. After each 8-bit transfer, the master device generates a 9th clock pulse, and releases the SDA line. The receiving device (addressed slave) will pull the SDA line low to acknowledge (ACK) the successful transfer or leave the SDA high to negative acknowledge (NACK).

Per the I^2C spec, all transitions in the SDA line must occur when SCL is low. This requirement leads to two unique conditions on the bus associated with the SDA transitions when SCL is high. Master device pulling the SDA line low while the SCL line is high indicates the Start (S) condition, and the Stop (P) condition is when the SDA line is pulled high while the SCL line is high. The I^2C protocol also allows for the Restart condition in which the master device issues a second start condition without issuing a stop.

All bus transactions begin with the master device issuing the start sequence followed by the slave address byte. The address byte contains the slave address; the upper 7 bits (bits7-1), and the Least Significant bit (LSb). The LSb of the address byte designates if the operation is a read (LSb=1) or a write (LSb=0). At the 9th clock pulse, the receiving slave device will issue the ACK (or NACK). Following these bus events, the master will send data bytes for a write operation, or the slave will clock out data with a read operation. All bus transactions are terminated with the master issuing a stop sequence.

I^2C bus control can be implemented with either hardware logic or in software. Typical hardware designs will release the SDA and SCL lines as appropriate to allow the slave device to manipulate these lines. In a software implementation, care must be taken to perform these tasks in code.

OPERATIONAL EXAMPLES

The HMC5883L has a fairly quick stabilization time from no voltage to stable and ready for data retrieval. The nominal 56 milli-seconds with the factory default single measurement mode means that the six bytes of magnetic data registers (DXRA, DXRB, DZRA, DZRB, DYRA, and DYRB) are filled with a valid first measurement.

To change the measurement mode to continuous measurement mode, after the power-up time send the three bytes:

0x3C 0x02 0x00

This writes the 00 into the second register or mode register to switch from single to continuous measurement mode setting. With the data rate at the factory default of 15Hz updates, a 67 milli-second typical delay should be allowed by the I^2C master before querying the HMC5883L data registers for new measurements. To clock out the new data, send:

0x3D, and clock out DXRA, DXRB, DZRA, DZRB, DYRA, and DYRB located in registers 3 through 8. The HMC5883L will automatically re-point back to register 3 for the next 0x3D query. All six data registers must be read properly before new data can be placed in any of these data registers.

Below is an example of a (power-on) initialization process for “continuous-measurement mode”:

1. Write CRA (00) – send **0x3C 0x00 0x70** (8-average, 15 Hz default, normal measurement)
2. Write CRB (01) – send **0x3C 0x01 0xA0** (Gain=5, or any other desired gain)
3. Write Mode (02) – send **0x3C 0x02 0x00** (Continuous-measurement mode)
4. Wait 6 ms or monitor status register or DRDY hardware interrupt pin
5. Loop
   Send **0x3D 0x06** (Read all 6 bytes. If gain is changed then this data set is using previous gain)
   Convert three 16-bit 2’s compliment hex values to decimal values and assign to X, Z, Y, respectively.
   Send **0x3C 0x03** (point to first data register 03)
   Wait about 67 ms (if 15 Hz rate) or monitor status register or DRDY hardware interrupt pin
   End_loop

Below is an example of a (power-on) initialization process for “single-measurement mode”:

1. Write CRA (00) – send **0x3C 0x00 0x70** (8-average, 15 Hz default or any other rate, normal measurement)
2. Write CRB (01) – send **0x3C 0x01 0xA0** (Gain=5, or any other desired gain)
3. For each measurement query:
   Write Mode (02) – send **0x3C 0x02 0x01** (Single-measurement mode)
   Wait 6 ms or monitor status register or DRDY hardware interrupt pin
   Send **0x3D 0x06** (Read all 6 bytes. If gain is changed then this data set is using previous gain)
   Convert three 16-bit 2’s compliment hex values to decimal values and assign to X, Z, Y, respectively.

SELF TEST OPERATION

To check the HMC5883L for proper operation, a self test feature in incorporated in which the sensor offset straps are excited to create a nominal field strength (bias field) to be measured. To implement self test, the least significant bits (MS1 and MS0) of configuration register A are changed from 00 to 01 (positive bias) or 10 (negetive bias).

Then, by placing the mode register into single or continuous-measurement mode, two data acquisition cycles will be made on each magnetic vector. The first acquisition will be a set pulse followed shortly by measurement data of the external field. The second acquisition will have the offset strap excited (about 10 mA) in the positive bias mode for X, Y, and Z axes to create about a 1.1 gauss self test field plus the external field. The first acquisition values will be subtracted from the second acquisition, and the net measurement will be placed into the data output registers.

Since self test adds ~1.1 Gauss additional field to the existing field strength, using a reduced gain setting prevents sensor from being saturated and data registers overflowed. For example, if the configuration register B is set to 0xA0 (Gain=5), values around +452 LSb (1.16 Ga * 390 LSb/Ga) will be placed in the X and Y data output registers and around +421 (1.08 Ga * 390 LSb/Ga) will be placed in Z data output register. To leave the self test mode, change MS1 and MS0 bit of the configuration register A back to 00 (Normal Measurement Mode). Acceptable limits of the self test values depend on the gain setting. Limits for Gain=5 is provided in the specification table.

Below is an example of a “positive self test” process using continuous-measurement mode:

1. Write CRA (00) – send **0x3C 0x00 0x71** (8-average, 15 Hz default, positive self test measurement)
2. Write CRB (01) – send **0x3C 0x01 0xA0** (Gain=5)
3. Write Mode (02) – send **0x3C 0x02 0x00** (Continuous-measurement mode)
4. Wait 6 ms or monitor status register or DRDY hardware interrupt pin
5. Loop
   Send **0x3D 0x06** (Read all 6 bytes. If gain is changed then this data set is using previous gain)
   Convert three 16-bit 2’s compliment hex values to decimal values and assign to X, Z, Y, respectively.
   Send **0x3C 0x03** (point to first data register 03)
   Wait about 67 ms (if 15 Hz rate) or monitor status register or DRDY hardware interrupt pin
   End_loop
6. Check limits –
   If all 3 axes (X, Y, and Z) are within reasonable limits (243 to 575 for Gain=5, adjust these limits basing on the gain setting used. See an example below.) Then
   All 3 axes pass positive self test
   Write CRA (00) – send **0x3C 0x00 0x70** (Exit self test mode and this procedure)
   Else
   If Gain<7
   Write CRB (01) – send **0x3C 0x01 0x_0** (Increase gain setting and retry, skip the next data set)
   Else
   At least one axis did not pass positive self test
   Write CRA (00) – send **0x3C 0x00 0x70** (Exit self test mode and this procedure)
   End If

Below is an example of how to adjust the “positive self” test limits basing on the gain setting:

1. If Gain = 6, self test limits are:
   Low Limit = 243 * 330/390 = 206
   High Limit = 575 * 330/390 = 487

2. If Gain = 7, self test limits are:
   Low Limit = 243 * 230/390 = 143
   High Limit = 575 * 230/390 = 339

SCALE FACTOR TEMPERATURE COMPENSATION

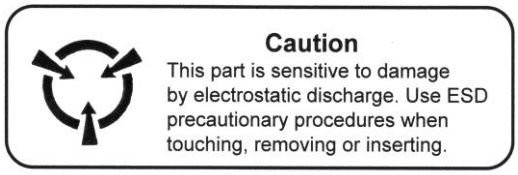
The built-in self test can also be used to periodically compensate the scaling errors due to temperature variations. A compensation factor can be found by comparing the self test outputs with the ones obtained at a known temperature. For example, if the self test output is 400 at room temperature and 300 at the current temperature then a compensation factor of (400/300) should be applied to all current magnetic readings. A temperature sensor is not required using this method.

Below is an example of a temperature compensation process using positive self test method:

1. If self test measurement at a temperature “when the last magnetic calibration was done”:
   X_STP = 400
   Y_STP = 410
   Z_STP = 420
2. If self test measurement at a different tmperature:
   X_STP = 300 (Lower than before)
   Y_STP = 310 (Lower than before)
   Z_STP = 320 (Lower than before)
   Then
   X_TempComp = 400/300
   Y_TempComp = 410/310
   Z_TempComp = 420/320
3. Applying to all new measurements:
   X = X * X_TempComp
   Y = Y * Y_TempComp
   Z = Z * Z_TempComp
   Now all 3 axes are temperature compensated, i.e. sensitivity is same as “when the last magnetic calibration was done”; therefore, the calibration coefficients can be applied without modification.
4. Repeat this process periodically or,for every Δt degrees of temperature change measured, if available.

ORDERING INFORMATION

Ordering Number	Product
HMC5883L-T	Cut Tape
HMC5883L-TR	Tape and Reel 4k pieces/reel



CAUTION: ESDS CAT. 1B

FIND OUT MORE

For more information on Honeywell’s Magnetic Sensors visit us online at www.magneticsensors.com or contact us at 1-800-323-8295 (763-954-2474 internationally).

Honeywell
12001 Highway 55
Plymouth, MN 55441
Tel: 800-323-8295
www.magneticsensors.com

Form # 900405 Rev E
February 2013

Honeywell

BMP085
Digital pressure sensor

Data sheet

Bosch Sensortec

BOSCH
Invented for life

BMP085 Data sheet	
Order code	0 273 300 144
Package type	LCC8
Data sheet revision	1.2
Release date	15 Oct 2009
Document number	BST-BMP085-DS000-05
Notes	The BMP085 digital pressure sensor is functionally compatible to the existing Bosch Sensortec SMD500 digital pressure sensor. Product photos and pictures are for illustration purposes only and may differ from the real product's appearance. Specifications are subject to change without notice.

BMP085 Digital pressure sensor

Key features

Pressure range:	300 ... 1100hPa (+9000m ... -500m above sea level)
Supply voltage:	1.8 ... 3.6V (V_{DDA}) 1.62V ... 3.6V (V_{DDD})
LCC8 package:	Robust, ceramic lead-less chip carrier (LCC) package Small footprint: 5.0mm x 5.0mm Super-flat: 1.2mm height
Low power:	5µA at 1 sample / sec. in standard mode
Low noise:	0.06hPa (0.5m) in ultra low power mode 0.03hPa (0.25m) ultra high resolution mode down to 0.1m (rms noise) possible

- Temperature measurement included
- I^2C interface
- Fully calibrated
- Pb-free, halogen-free and RoHS compliant,
- MSL 1

New features comparison

	BMP085	**SMD500**
Smaller package height	1.2mm	1.55mm
Faster conversion time (standard mode each)	7.5ms (max.)	34ms
Faster I^2C data transfer	max. 3.4MHz	max. 400kHz
Extended min. supply voltage	min. 1.8V	min. 2.2V
Lower stand-by current (typ.)	0.1µA	0.7µA
External clock	not necessary	necessary

Typical applications

- Enhancement of GPS navigation (dead-reckoning, slope detection, etc.)
- In- and out-door navigation
- Leisure and sports
- Weather forecast
- Vertical velocity indication (rise/sink speed)

BMB085 general description

The BMP085 is the fully pin- and function compatible successor of the SMD500, a new generation of high precision digital pressure sensors for consumer applications. The universal C-code SMD500/BMP085 ("BMP085_SMD500_API") is fully upward compatible to SMD500 and recognizes automatically the device ID. Customers already working with the SMD500 pressure sensor are invited to contact Bosch Sensortec as soon as they intend to switch-over to the BMP085 sensor for getting first-hand support.

The ultra-low power, low voltage electronics of the BMP085 is optimized for use in mobile phones, PDAs, GPS navigation devices and outdoor equipment. With a low altitude noise of merely 0.25m at fast conversion time, the BMP085 offers superior performance. The I^2C interface allows for easy system integration with a microcontroller.

The BMP085 is based on piezo-resistive technology for EMC robustness, high accuracy and linearity as well as long term stability.

Robert Bosch is the world market leader for pressure sensors in automotive applications. Based on the experience of over 200 million pressure sensors in the field, the BMP085 continues a new generation of micro-machined pressure sensors.

TABLE OF CONTENTS

1 Electrical characteristics

If not stated otherwise, the given values are maximum values over temperature/voltage range in the given operation mode.

Parameter	Symbol	Condition	Min	Typ	Max	Units
Operating temperature	T_A	operational	-40		+85	°C
		full accuracy	0		+65	
Supply voltage	V_{DD}	ripple max. 50mVpp	1.8	2.5	3.6	V
	V_{DDIO}		1.62	2.5	3.6	V
Supply current @ 1 sample / sec. 25°C	I_{DDLOW}	ultra low power mode		3		µA
	I_{DDSTD}	standard mode		5		µA
	I_{DDHR}	high resolution mode		7		µA
	I_{DDUHR}	ultra high res. mode		12		µA
Peak current	I_{peak}	during conversion		650	1000	µA
Standby current	I_{DDSBM}	at 25°C		0.1		µA
Serial data clock	f_{SCL}				3.4	MHz
Conversion time temperature	t_{C_temp}	standard mode		3	4.5	ms
Conversion time pressure	$t_{c\_p\_low}$	ultra low power mode		3	4.5	ms
	$t_{c\_p\_std}$	standard mode		5	7.5	ms
	$t_{c\_p\_hr}$	high resolution mode		9	13.5	ms
	$t_{c\_p\_luhr}$	ultra high res. mode		17	25.5	ms
Absolute accuracy pressure V_{DD} = 3.3V		700 . . . 1100 hPa 0 . . . +65 °C	-2.5	±1.0	+2.5	hPa
		300 . . . 700 hPa 0 . . . +65 °C	-3.0	±1.0	+3.0	hPa
		300 . . . 1100 hPa -20 . . . 0 °C	-4.0	±1.5	+4.0	hPa
Resolution of output data		pressure		0.01		hPa
		temperature		0.1		°C
Relative accuracy pressure		700 . . . 1100 hPa @ 25 °C		±0.2		hPa
V_{DD} = 3.3V		0 . . . 65 °C @ p const.		±0.5		hPa
Noise in pressure		see table on page 10				
Absolute accuracy temperature V_{DD} = 3.3V		@ 25 °C	-1.5	±0.5	+1.5	°C
		0 . . . +65 °C	-2.0	±1.0	+2.0	°C

Solder drifts		Minimum solder height 50µm		±1.0		hPa
Long term stability		12 months		±1.0		hPa

2 Absolute maximum ratings

Parameter	Condition	Min	Max	Units
Storage temperature		-40	+85	°C
Supply voltage	all pins	-0.3	+4.25	V
ESD rating	HBM, R = 1.5kΩ, C = 100pF		±2	kV
Overpressure			10,000	hPa

The BMP085 has to be handled as Electrostatic Sensitive Device (ESD).

3 Operation

3.1 General description

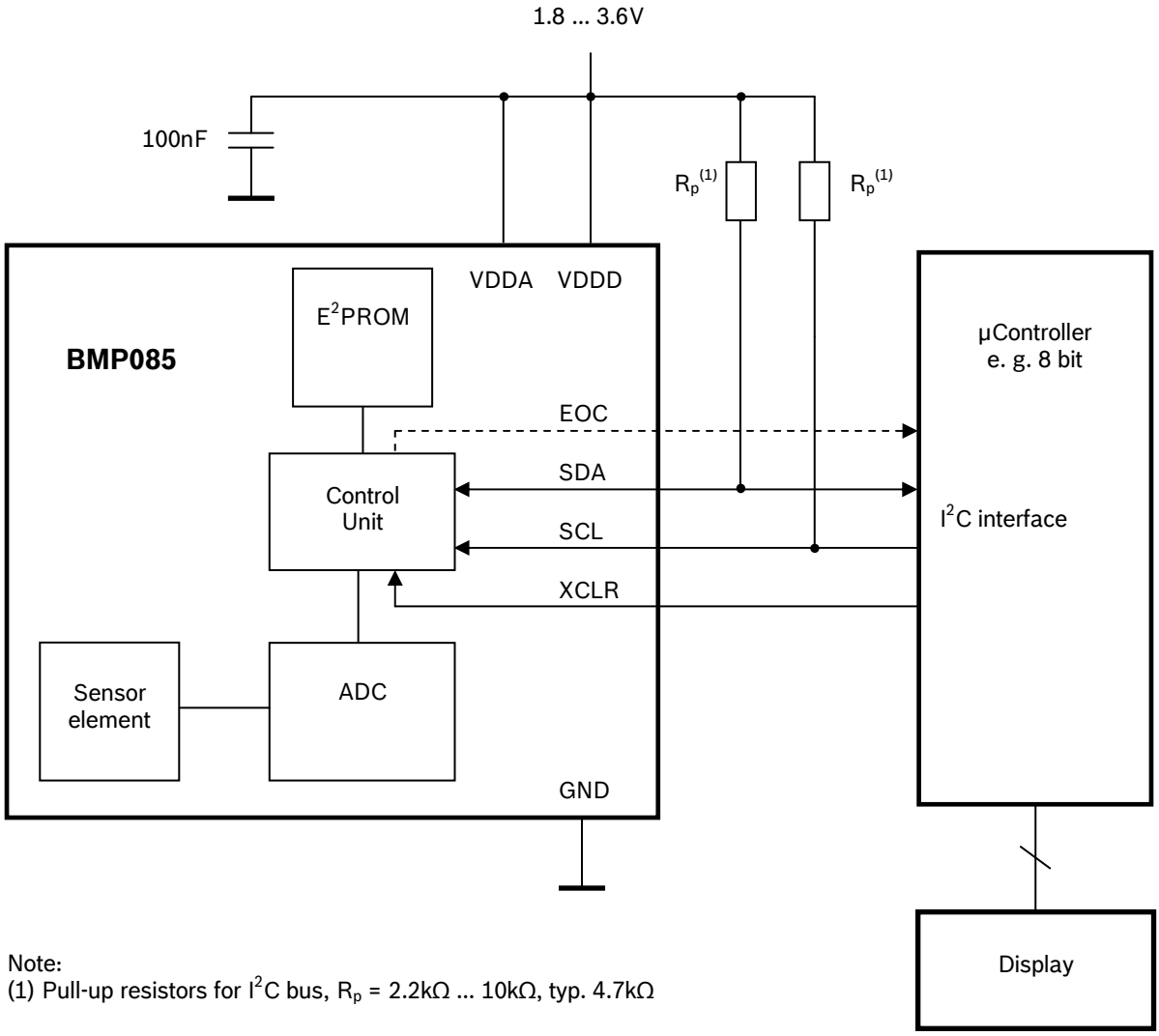
The BMP085 is designed to be connected directly to a microcontroller of a mobile device via the I^2C bus. The pressure and temperature data has to be compensated by the calibration data of the E^2PROM of the BMP085.

3.2 General function and application schematics

The BMP085 consists of a piezo-resistive sensor, an analog to digital converter and a control unit with E^2PROM and a serial I^2C interface. The BMP085 delivers the uncompensated value of pressure and temperature. The E^2PROM has stored 176 bit of individual calibration data. This is used to compensate offset, temperature dependence and other parameters of the sensor.

- UP = pressure data (16 to 19 bit)
- UT = temperature data (16 bit)

Typical application circuit:



Note:
(1) Pull-up resistors for I^2C bus, R_p = 2.2kΩ ... 10kΩ, typ. 4.7kΩ

Note:
The BMP085 can be supplied independently with different levels of V_{DDA} and V_{DDD}, which is not possible with the SMD500. In case of different voltage levels, V_{DDA} and V_{DDD} shall have a 100nF decoupling capacitor each.

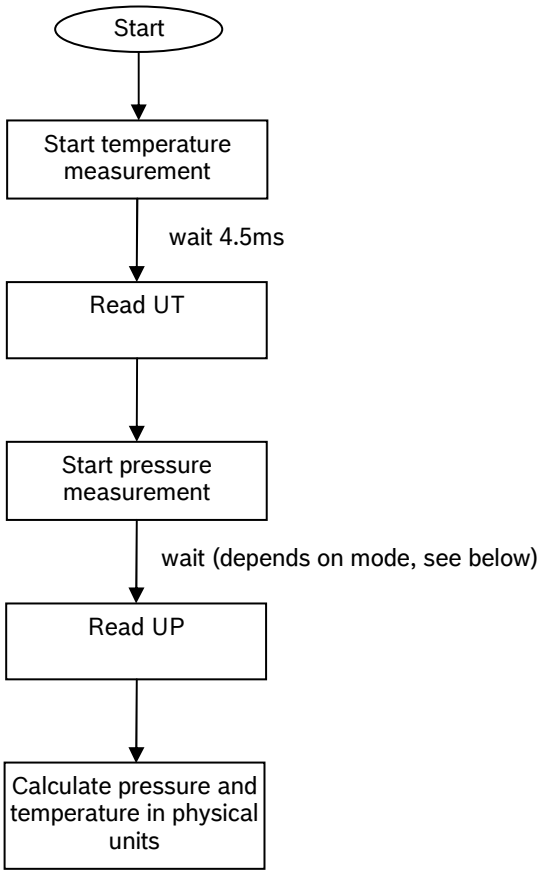
3.3 Measurement of pressure and temperature

For all calculations presented here an ANSI C code is available from Bosch Sensortec ("BMP085_SMD500_API").

The microcontroller sends a start sequence to start a pressure or temperature measurement. After converting time, the result value (UP or UT, respectively) can be read via the I^2C interface. For calculating temperature in °C and pressure in hPa, the calibration data has to be used. These constants can be read out from the BMP085 E^2PROM via the I^2C interface at software initialization.

The sampling rate can be increased up to 128 samples per second (standard mode) for dynamic measurement. In this case, it is sufficient to measure the temperature only once per second and to use this value for all pressure measurements during the same period.

By using different modes the optimum compromise between power consumption, speed and resolution can be selected, see below table.



Overview of BMP085 modes, selected by driver software via the variable *oversampling_setting*:

Mode	Parameter *oversampling_setting*	Internal number of samples	Conversion time pressure max. [ms]	Avg. current @ 1 sample/s typ. [µA]	RMS noise typ. [hPa]	RMS noise typ. [m]
ultra low power	0	1	4.5	3	0.06	0.5
standard	1	2	7.5	5	0.05	0.4
high resolution	2	4	13.5	7	0.04	0.3
ultra high resolution	3	8	25.5	12	0.03	0.25

The noise data is calculated as standard deviation of 10 data points. For further information on noise characteristics see the relevant application note "Noise in pressure sensor applications".

All modes can be performed at higher speeds, e.g. up to 128 times per second for standard mode, with the current consumption increasing proportionally to the sample rate. This way the noise can be decreased further by software averaging.

3.4 Calibration coefficients

The 176 bit E^2PROM is partitioned in 11 words of 16 bit each. These contain 11 calibration coefficients. Every sensor module has individual coefficients. Before the first calculation of temperature and pressure, the master reads out the E^2PROM data.
The data communication can be checked by checking that none of the words has the value 0 or 0xFFFF.

	BMP085 reg adr	
Parameter	**MSB**	**LSB**
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

3.5 Calculating pressure and temperature

The mode (ultra low power, standard, high, ultra high resolution) can be selected by the variable *oversampling_setting* (0, 1, 2, 3) in the C code.

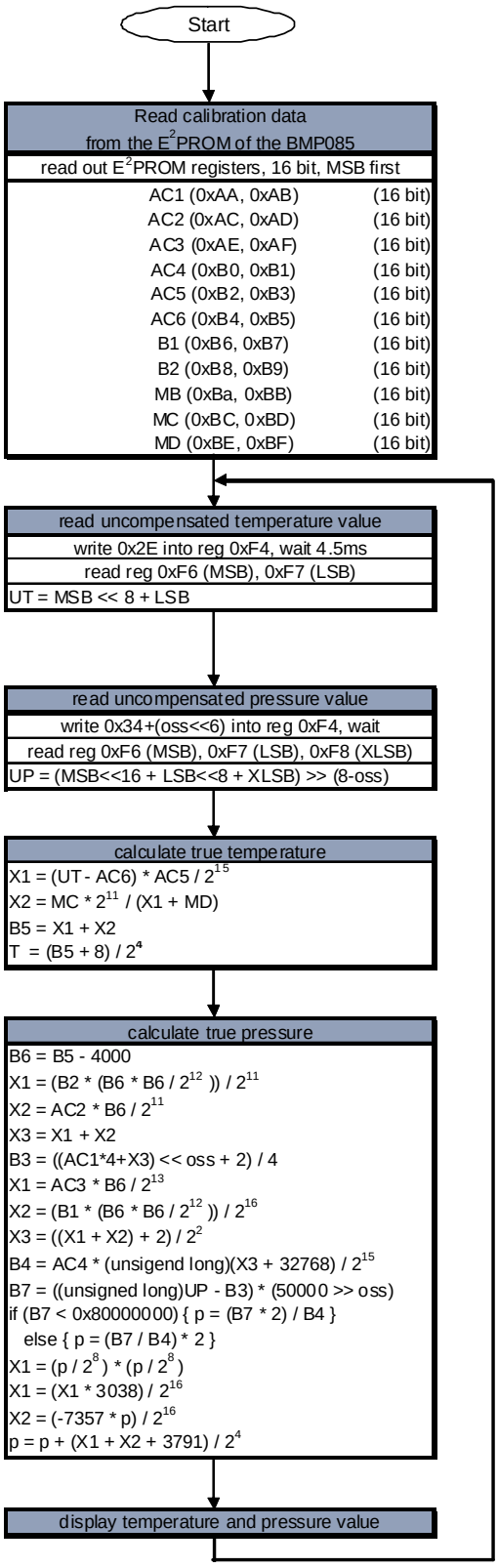
The universal code SMD500/BMP085 is fully upward compatible to SMD500 and recognizes automatically the device ID. Thus, the SMD500 can be replaced "on the fly" by the BMP085 without changing hardware or software.

Calculation of true temperature and pressure in steps of 1Pa (= 0.01hPa = 0.01mbar) and temperature in steps of 0.1°C.

The following figure shows the detailed algorithm for pressure and temperature measurement.

This algorithm is available to customers as reference C source code ("BMP085_SMD500_API") from Bosch Sensortec and via its sales and distribution partners. **Please contact your Bosch Sensortec representative for details.**

Calculation of pressure and temperature for BMP085



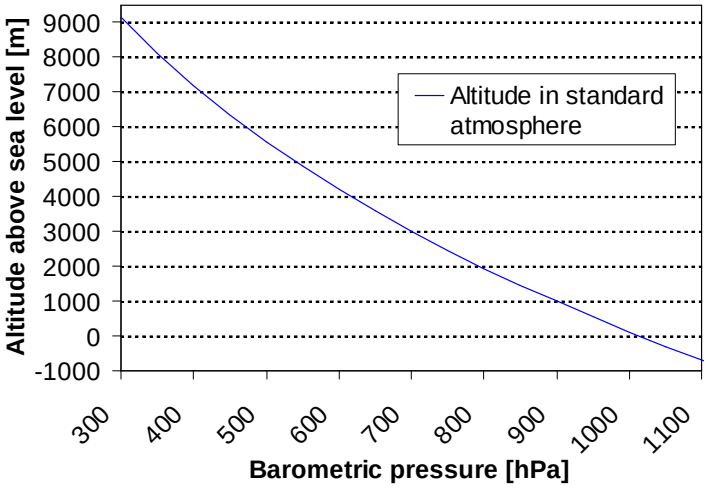
	example:	C code function:	type:
		bmp085_get_cal_param	
AC1 =	408		short
AC2 =	-72		short
AC3 =	-14383		short
AC4 =	32741		unsigned short
AC5 =	32757		unsigned short
AC6 =	23153		unsigned short
B1 =	6190		short
B2 =	4		short
MB =	-32768		short
MC =	-8711		short
MD =	2868		short
		bmp085_get_ut	
UT =	**27898**		long
oss =	0		
= *oversampling_setting*			short (0 .. 3)
(ultra low power mode)		bmp085_get_up	
UP =	**23843**		long
		bmp085_get_temperature	
X1 =	4743		long
X2 =	-2344		long
B5 =	2399		long
T =	**150**	***temp in 0.1℃***	long
		BMP085_calpressure	
B6 =	-1601		long
X1 =	1		long
X2 =	56		long
X3 =	57		long
B3 =	422		long
X1 =	2810		long
X2 =	59		long
X3 =	717		long
B4 =	33457		unsigned long
B7 =	1171050000		unsigned long
p =	70003		long
			long
X1 =	74529		long
X1 =	3454		long
X2 =	-7859		long
p =	**69964**	***press. in Pa***	long

3.6 Calculating absolute altitude

With the measured pressure p and the pressure at sea level p_0 e.g. 1013.25hPa, the altitude in meters can be calculated with the international barometric formula:

$$\text{altitude} = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

Thus, a pressure change of Δp = 1hPa corresponds to 8.43m at sea level



3.7 Calculating pressure at sea level

With the measured pressure p and the absolute altitude the pressure at sea level can be calculated:

$$p_0 = \frac{p}{\left(1 - \frac{\text{altitude}}{44330}\right)^{5.255}}$$

Thus, a difference in altitude of Δaltitude = 10m corresponds to 1.2hPa pressure change at sea level.

4 I²C Interface

- I²C is a digital two wire interface
- Clock frequencies up to 3.4Mbit/sec. (I²C standard, fast and high-speed mode supported)
- SCL and SDA needs a pull-up resistor, typ. 4.7kOhm to V_{DDD}
  (one resistor each for all the I²C bus)

The I²C bus is used to control the sensor, to read calibration data from the E²PROM and to read the measurement data when A/D conversion is finished. SDA (serial data) and SCL (serial clock) have open-drain outputs.

For detailed I²C-bus specification please refer to:
http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

The BMP085 has a master clear (XCLR) low-active input that is used to reset the BMP085 and initializes internal registers and counters. The device is automatically reset by power on reset (POR) circuitry. XCLR can be left floating if not used. The pad has an internal pull-up resistor of typ. 120kOhm.

4.1 I²C specification

Electrical parameters for the I²C interface:

Parameter	Symbol	Min.	Typ	Max.	Units
Clock input frequency	f_{SCL}			3.4	MHz
Input-low level	V_{IL}	0		0.2 * V_{DDD}	V
Input-high level	V_{IH}	0.8 * V_{DDD}		V_{DDD}	V
SDA and SCL pull-up resistor	$R_{pull-up}$	2.2		10	kOhm
SDA sink current @ V_{DDD} = 1.62V, V_{OL} = 0.3V	I_{SDA_sink}		9		mA
EOC sink current @ V_{DDD} = 1.62V, V_{OL} = 0.3V	I_{SDA_sink}		7.7		mA
EOC source current @ V_{DDD} = 1.62V, V_{OH} = 1.32V	I_{SDA_source}		1.5		mA
XCLR pulse length	t_{XCLR}	1			µs
Start-up time after power-up, before first communication	t_{Start}	10			ms

4.2 Device and register address

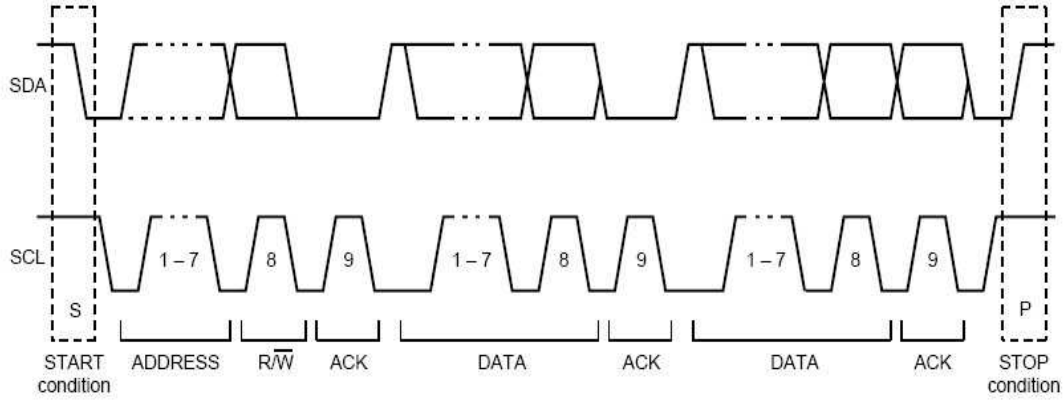
The BMP085 module address is shown below. The LSB of the device address distinguishes between read (1) and write (0) operation, corresponding to address 0xEF (read) and 0xEE (write).

A7	A6	A5	A4	A3	A2	A1	W/R
1	1	1	0	1	1	1	0/1

There is an easy way to connect two BMP085 to the same I^2C bus: You can use the XCLR input of BMP085 to set one BMP085 part silent while you communicate with the other BMP085 part via I^2C and vice versa. The signals can be provided by two digital outputs of the micro-controller, or one digital output and one inverter.

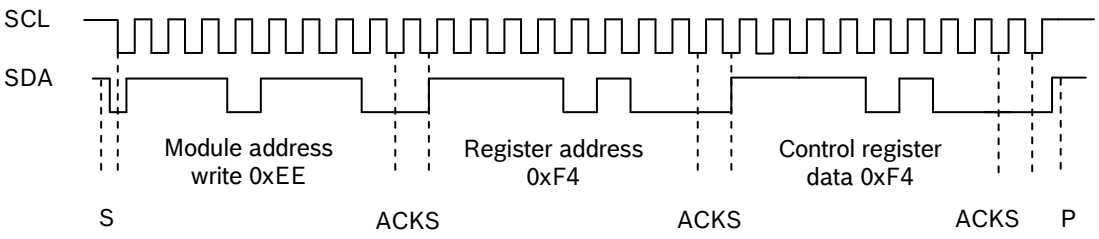
4.3 I^2C protocol

The I^2C interface protocol has special bus signal conditions. Start (S), stop (P) and binary data conditions are shown below. At start condition, SCL is high and SDA has a falling edge. Then the slave address is sent. After the 7 address bits, the direction control bit R/W selects the read or write operation. When a slave device recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle.
At stop condition, SCL is also high, but SDA has a rising edge. Data must be held stable at SDA when SCL is high. Data can change value at SDA only when SCL is low.



4.4 Start temperature and pressure measurement

The timing diagrams to start the measurement of the temperature value UT and pressure value UP are shown below. After start condition the master sends the device address write, the register address and the control register data. The BMP085 sends an acknowledgement (ACKS) every 8 data bits when data is received. The master sends a stop condition after the last ACKS.



Timing diagram for starting pressure measurement

Abbreviations:

S	Start
P	Stop
ACKS	Acknowledge by Slave
ACKM	Acknowledge by Master
NACKM	Not Acknowledge by Master

Control registers values for different internal *oversampling_setting* (osrs):

Measurement	Control register value (register address 0xF4)	Max. conversion time [ms]
Temperature	0x2E	4.5
Pressure (osrs = 0)	0x34	4.5
Pressure (osrs = 1)	0x74	7.5
Pressure (osrs = 2)	0xB4	13.5
Pressure (osrs = 3)	0xF4	25.5

Instead of waiting for the maximum conversion time, the output pin EOC (end of conversion) can be used to check if the conversion is finished (logic 1) or still running (logic 0). After the conversion is finished BMP085 switches automatically in standby mode.

4.5 Read A/D conversion result or E²PROM data

To read out the temperature data word UT (16 bit), the pressure data word UP (16 to 19 bit) and the E²PROM data proceed as follows:

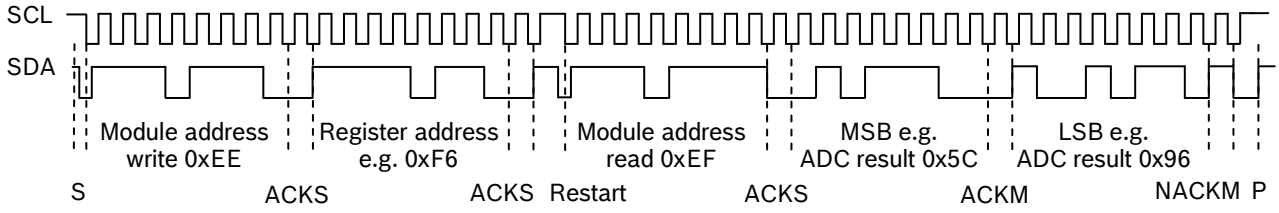
After the start condition the master sends the module address write command and register address. The register address selects the read register:

E²PROM data registers	0xAA to 0xBF
Temperature or pressure value UT or UP	0xF6 (MSB), 0xF7 (LSB), optionally 0xF8 (XLSB)

Then the master sends a restart condition followed by the module address read that will be acknowledged by the BMP085 (ACKS). The BMP085 sends first the 8 MSB, acknowledged by the master (ACKM), then the 8 LSB. The master sends a "not acknowledge" (NACKM) and finally a stop condition.

Optionally for ultra high resolution, the XLSB register with address 0xF8 can be read to extend the 16 bit word to up to 19 bits; refer to the application programming interface (API) software rev. 1.1 ("BMP085_SMD500_API", available from Bosch Sensortec).

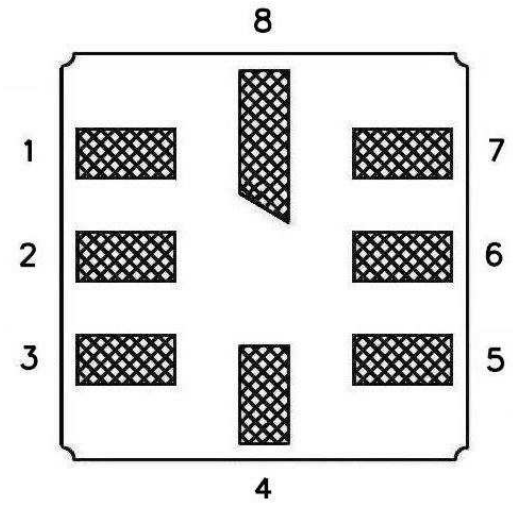
Timing diagram read 16 bit A/D conversion result:



5 Package

5.1 Pin configuration

Picture shows the device in top view. Device pins are shown here transparently only for orientation purposes.



Pin No.	Name	Function	Type
1	GND	Ground	Power
2	EOC	End of conversion	Digital output
3	V_{DDA}	Power supply	Power
4	V_{DDD}	Digital power supply	Power
5	NC	no internal connection	-
6	SCL	I^2C serial bus clock input	Digital input
7	SDA	I^2C serial bus data	Digital bi-directional
8	XCLR	master clear (low active) input	Digital input

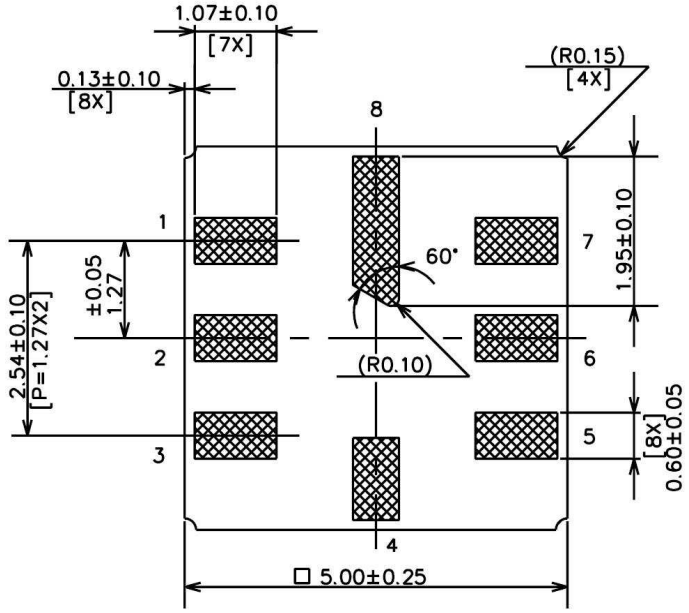
5.2 Outline dimensions

The sensor housing is a standard 8-pin lead-less chip carrier (LCC8) ceramic package. Its dimensions are 5.0mm x 5.0mm (±0.25mm) x 1.2mm (±0.12mm).
Package weight is approximately 0.09grams.

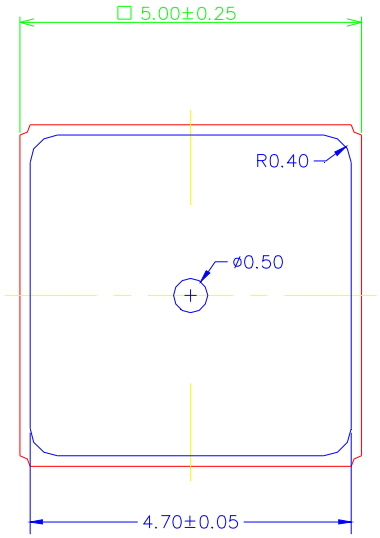
Note: All dimensions are in mm.

5.2.1 Top view

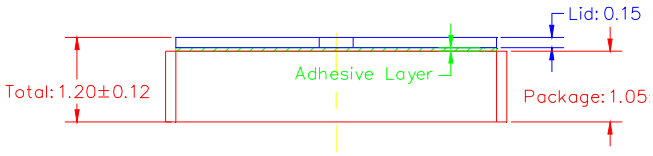
Device pins are shown here transparently only for orientation purposes.



5.2.2 Top view with lid

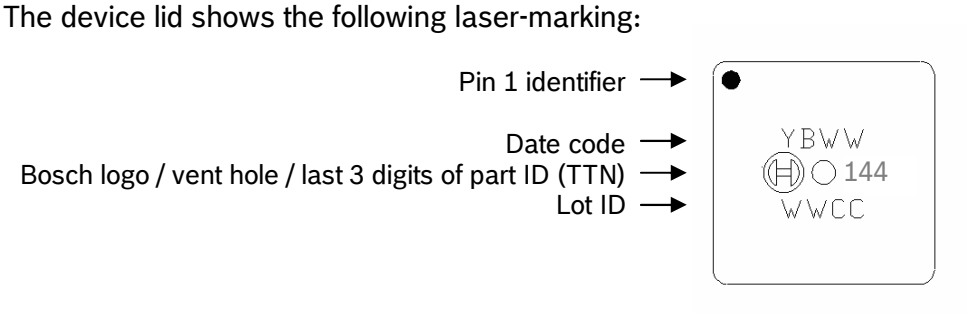


5.2.3 Side view with lid



5.3 Device marking

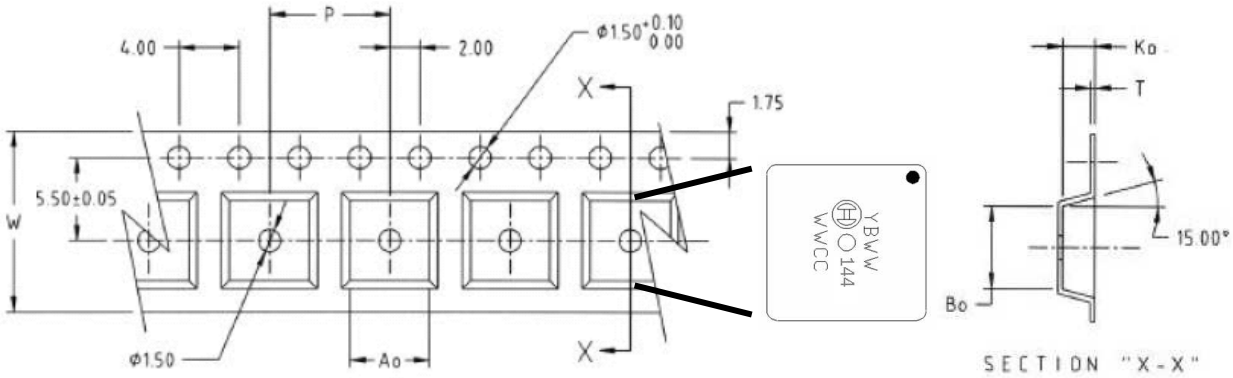
The device lid shows the following laser-marking:



The vent hole (diameter 0.5mm) is in the center of the lid, between the Bosch logo and the part ID code (144).

5.4 Tape on reel

Number of parts per reel: 3,000
Orientation of the parts inside the reel is according to EN60286-3.



Carrier tape material: Conductive polystyrene C 100, 10^3 – 10^6 Ohm/sq.

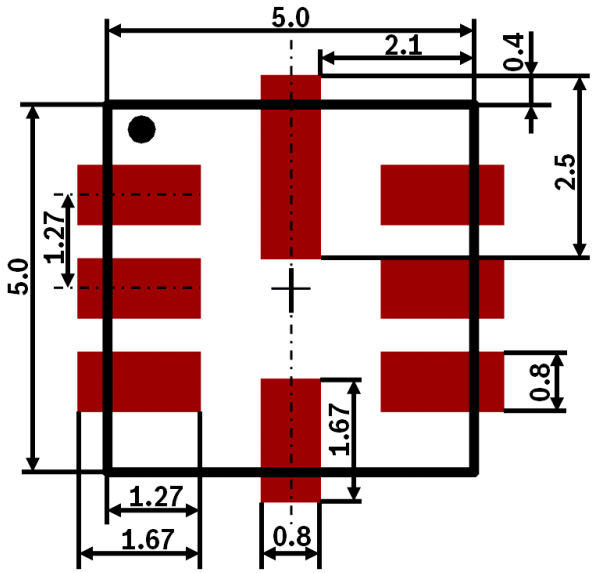
Dimensions are in mm:

A_0 = 5.30 ± 0.10
B_0 = 5.30 ± 0.10
K_0 = 2.10 ± 0.10
P = 8.00 ± 0.10
W = 12.00 +0.30 / -0.10
t = 0.30 ± 0.05

The bars between the cavities are embossed about 0.2mm into the direction of K_0. All other dimensions and tolerances follow the EIA 481 standard.

5.5 Printed circuit board (PCB) design

Recommended PCB design (top view):



5.6 Moisture sensitivity level and soldering

The BMP085 is classified MSL 1 (moisture sensitivity level) according to IPC/JEDEC standards J-STD-020D and J-STD-033A.

The device can be soldered Pb-free with a peak temperature of 260°C for 20 to 40 sec. The minimum height of the solder after reflow shall be at least 50µm. This is required for good mechanical decoupling between the sensor device and the printed circuit board (PCB).

The BMP085 devices have to be soldered within 6 months after shipment (shelf life). To ensure good solder-ability, the devices shall be stored at room temperature (20°C).

The soldering process can lead to an offset shift of typically 1hPa.

5.7 RoHS compliancy

The BMP085 sensor meets the requirements of the EC directive "Restriction of hazardous substances (RoHS)", please refer also to:

"Directive 2002/95/EC of the European Parliament and of the Council of 27 January 2003 on the restriction of the use of certain hazardous substances in electrical and electronic equipment".

The BMP085 sensor is also halogen-free. Please contact your Bosch Sensortec representative for the corresponding analysis report.

5.8 Mounting and assembly recommendations

Please read the following recommendations carefully:

- The clearance above the metal lid shall be 0.1mm at minimum.
- For the device housing appropriate venting needs to be provided in case the ambient pressure shall be measured. If waterproof packaging is needed, venting can be accomplished by a vent element with a membrane like Gore-Tex(TM).
- Liquids shall not come into direct contact with the device.
- During operation the sensor is sensitive to light, which can influence the accuracy of the measurement (photo-current of silicon). Therefore, the hole in the top lid shall not be exposed to direct light during operation.
- The BMP085 shall not the placed close the fast heating parts. In case of gradients > 3°C/sec. it is recommended to follow Bosch Sensortec application note ANP015, "Correction of errors induced by fast temperature changes". Please contact your Bosch Sensortec representative for details.
- For further details, please refer to the BMP085 handling, soldering & mounting instructions manual that is also available from Bosch Sensortec.

6 Legal disclaimer

6.1 Engineering samples

Engineering Samples are marked with an asterisk (*) or (e). Samples may vary from the valid technical specifications of the product series contained in this data sheet. They are therefore not intended or fit for resale to third parties or for use in end products. Their sole purpose is internal client testing. The testing of an engineering sample may in no way replace the testing of a product series. Bosch Sensortec assumes no liability for the use of engineering samples. The Purchaser shall indemnify Bosch Sensortec from all claims arising from the use of engineering samples.

6.2 Product use

Bosch Sensortec products are developed for the consumer goods industry. They may only be used within the parameters of this product data sheet. They are not fit for use in life-sustaining or security sensitive systems. Security sensitive systems are those for which a malfunction is expected to lead to bodily harm or significant property damage. In addition, they are not fit for use in products which interact with motor vehicle systems.

The resale and/or use of products are at the purchaser's own risk and his own responsibility. The examination of fitness for the intended use is the sole responsibility of the Purchaser.

The purchaser shall indemnify Bosch Sensortec from all third party claims arising from any product use not covered by the parameters of this product data sheet or not approved by Bosch Sensortec and reimburse Bosch Sensortec for all costs in connection with such claims.

The purchaser must monitor the market for the purchased products, particularly with regard to product safety, and inform Bosch Sensortec without delay of all security relevant incidents.

6.3 Application examples and hints

With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Bosch Sensortec hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights or copyrights of any third party. The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. They are provided for illustrative purposes only and no evaluation regarding infringement of intellectual property rights or copyrights or regarding functionality, performance or error has been made.

7 Document history and modifications

Rev. No	Chapter	Description of modifications/changes	Date
0.1		First edition for description of serial production material	
0.2		Noise data update, peak current added, several minor changes	19-Feb-2008
0.3	1	Update peak current, typo correction	21-Apr-2008
	1	Changed V_{DDA} to V_{DD} and V_{DDD} to V_{DDIO}	21-Apr-2008
	3.5	Updated flow diagram	21-Apr-2008
	4	New comment on floating XCLR	21-Apr-2008
	4.1	Added details on start-up time	21-Apr-2008
	5.2	Added package weight	21-Apr-2008
1.0	1	Absolute accuracy at -20°C added	01-July-2008
	2	Updated storage temperature	01-July-2008
	3.2	Added optional EOC to schematic	01-July-2008
	5.7	BMP085 is halogen-free	01-July-2008
1.1	3.3	Added comment on noise data calculation	06-April-2009
	3.5	Variable type of B7 is unsigned long	
	4	Added comment on low active master clear and 120kOhms pull-up resistor	
	4.2	Added solution to connect two devices BMP085 to the same I2C bus.	
	4.4	Added remark about automatic standby-mode	
	5.6	JEDEC J-STD-020D	
	5.5	Updated or recommended PCB-design picture	
	5.8	Explained that light sensitivity is only during operation	
1.2	3.5	Added comment on availability of the ref. source code	15-Oct-2009

Bosch Sensortec GmbH
Gerhard-Kindler-Strasse 8
72770 Reutlingen / Germany

contact@bosch-sensortec.com
www.bosch-sensortec.com

Modifications reserved | Printed in Germany
Specifications are subject to change without notice
Version_1.2_102009
Document number: BST-BMP085-DS000-05