UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Geodésica, Cartográfica y Topográfica

Adquisición de datos IMU en un sistema embebido

Trabajo Fin de Grado

Ingeniería en Geomática y Topografía

Autor: David Ferrer Calatayud

Tutor: Dr. Ángel Marqués Mateu

Julio, 2015

AGRADECIMIENTOS

Quiero agradecer desde estas líneas a mi tutor, el Dr. Àngel Marqués Mateu, toda la ayuda y atención que me ha ofrecido en estos meses. Este proyecto no hubiese sido real sin el material cedido y sin la paciencia que me ha brindado en mis incesantes preguntas.

Pero estas líneas no las estaría escribiendo si no fuese por el empeño que mis padres han puesto en mí. Jamás se rindieron ni dejaron de creer en mis posibilidades, ni siquiera cuando yo dejé de creer en mí. A ellos se los debo todo. Tampoco puedo olvidar a mis hermanas, siempre han sido un grandísimos apoyo para mí, han sido mis pequeños ángeles de la guarda.

Y muy especialmente a mi prometida, Marta, la cual hizo que me centrase de una vez por todas, que me hizo ver que en esta vida hay más cosas, que me daba las fuerzas en los peores momentos y que ante todo, siempre confió en mí y tuvo unas palabras reconfortantes y cariñosas para mí. Sin ella hoy no estaría aquí acabando mí carrera. Gracias de todo corazón.

RESUMEN

La finalidad del proyecto es la de diseñar y desarrollar un sistema emebebido compuesto por un microordenador y una IMU que permita obtener lecturas de movimientos inerciales.

Una IMU es un dispositivo formado por diferentes componentes como un acelerómetro, que medirá los cambios de la fuerza registrada producidos por el movimiento del dispositivo; un magnetómetro, que interactuará con el campo magnético terrestre y un giroscopio, registrará la variación de la posición de los ejes representados por la IMU.

Estos datos que serán proporcionados por la IMU, son de gran importancia para muchas disciplinas que están estrechamente relacionadas con el mundo de la topografía y la geomática, desde conocer la posición de la cámara de un avión cuando se realiza un vuelo fotogramétrico, hasta para la asistencia y control de los modernos y avanzados drones topográficos, civiles o militares y que se están convirtiendo en una herramienta muy común en nuestra sociedad.

Del mismo modo, el software desarrollado para la gestión del sensor IMU proporcionará una herramienta muy útil cuyo objetivo, no es otro que ayudar a la obtención, gestión y manipulación de los datos inerciales que una IMU es capaz de ofrecernos. Para su desarrollo se ha recurrido a uno de los lenguajes de programación más potentes y utilizados de los últimos tiempos, como es Python.

El sistema embebido se ha construido con uno de los mejores microordenadores del mercado, como es las Raspberry Pi B+ y para el sensor IMU, se eligió por un probado y fiable dispositivo de Polou, el MinIMU9 v2.

Palabras clave:

Python, embebido, Linux, IMU, Inercial, giroscopio, magnetómetro, acelerómetro

ABSTRACT

The aim of this project is to design and develop an embebed system composed of a microcomputer and an IMU which get data of inertial movements.

An IMU is a device which consists of different items such an accelerometer which will measure changes of the registered strength caused by the device movements; a magnetometer which will interact with the Earth magnetic field and a “giro” to register the changes in the axes position which are represented by the IMU.

The data given by the IMU are very important for many disciplines, which are strongly related to topography and geomatic world from knowing the positioning of a camera in a plane to assist and control the modern topographic drones. These are common tools in our society.

In the same way, the developed software to control the IMU sensor will provide us a very useful tool which target is to help in the getting, process and handling of the inertial data that an IMU can offer us. To develop it have been use one of the most powerful and used programming languages like the Python.

The system have been assembled with one of the best current microcomputers, the Raspberry Pi B+. To the IMU sensor was choosen the device of Polou, MinIMU9 v2, because of its realibility.

KeyWords:

Python, embebed, Linux, IMU, Inertial, gyroscope, magnetometer, accelerometer

Índice

1. INTRODUCCIÓN

La introducción del presente trabajo pretende poner en situación el proyecto realizado, describiendo así los componentes o utilidades empleadas en su desarrollo. Por tanto, en esta presentación se hará especial énfasis en cuatro aspectos básicos:

- Qué es una IMU[1] y cómo trabajan sus componentes
- Qué es Python y el porqué de su utilización
- Tkinter: Un diseñador de interfaces para Python: potente, pero con limitaciones
- Raspberry: De herramienta de aprendizaje a piedra angular del mundo geek

En la primera parte veremos con detalle que es una IMU, de que elementos está compuesta y qué relaciones matemáticas y fundamentos físicos subyacen en su funcionamiento. Aunque muchos de estos conceptos no se van utilizar debido a que el software con el que funciona la IMU ya está desarrollado, si parecía interesante conocerlo, de este modo sería más sencillo detectar posibles errores o fallos de funcionamiento en la IMU. Además, con unos conocimiento más avanzados en materia de programación y electrónica básica, se podría incluso realizar modificaciones en el propio dispositivo, añadiendo nuevos chips y componentes, que junto a la modificación del código fuente del software que gestiona y controla la IMU se podrían obtener una grandísima variedad de datos y productos derivados diferentes. Aunque en este proyecto no se abordará esta cuestión, si resulta un grandísimo punto de partida, pues la práctica adquirida en la manipulación del hardware del proyecto y la progresiva familiarización con el lenguaje de programación utilizado, lo harían posible.

En la segunda parte, se hará un repaso sobre que es Python[2] y por qué se ha utilizado. A estas alturas a nadie le resulta desconocido hablar de Python, y mucho menos en el ámbito de la geomática y la topografía. No obstante realizar una breve revisión sobre sus peculiaridades, sus diferentes versiones, su curiosa división en dos "familias" hace unos años y el futuro próximo que le espera a esta plataforma de desarrollo, es la forma más idónea para dar coherencia a todo el proceso realizado en este proyecto.

En la tercera parte veremos algunos detalles del diseñador de interfaces Tkinter[3]. Tal y como se ha hecho mención anteriormente, Tkinter es un binding de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python, que aunque lleva muchos años en el mundo de la programación, sigue siendo una de las herramientas más potentes en el campo de la creación de GUI en Python. Sin embargo, en este proyecto ha presentado algunas limitaciones que se irán viendo a lo largo del desarrollo del mismo, aunque a pesar de ello, se han ido solucionando de la forma más eficiente conocida.

Una vez dicho esto, cabe destacar que no se pretenderá entrar en detalle sobre qué opciones y capacidades tiene Tkinter, sino que tan solo se dará alguna pincelada para tener una visión más global y clara de lo que supone esta herramienta. En todo caso, si se desea profundizar en este campo, se dejará en las referencias todo el material utilizado para la realización del proyecto con posibilidad de consulta.

[1] IMU: Inertial Measurement Unit ó Unidad de Medición Inercial
[2] Python: Sitio oficial - https://www.python.org/
[3] Tkinter: Sitio oficial - https://wiki.python.org/moin/TkInter

Finalmente, se hará un pequeño repaso al miniordenador que nos ha permitido construir el sistema embebido, la Raspberry B+. Este pequeño ordenador ha sido la base del desarrollo de este proyecto y, aunque puede parecer sencillo y muy básico, la incorporación de los Pins GPIO hace de él una herramienta con un elevado potencial. Así, este miniordenador nació como un intento de acercar los ordenadores y la tecnología a niños, debido a su bajo coste. Sin embargo, consecuentemente a sus cualidades, se ha posicionado como un referente en el mundo del desarrollo y del mundo *geek*[4] generando a su alrededor una cultura bloguera de gran importancia, cultura en la que los desarrolladores proponen y comparten proyectos tomando como base una Raspberry en sus diferentes versiones.

[4] Geek: (del inglés geek) es un término que se utiliza para referirse a la persona fascinada por la tecnología y la informática.

1.1. Qué es una IMU y cómo trabajan sus componentes

El objeto de este proyecto, es el de proporcionar una herramienta que permita obtener datos de un sensor IMU que registra los movimientos inerciales de este. La intención es poder hacer accesibles unos datos, que para el ámbito de la topografía, la teledetección y la fotogrametría son de extraordinario valor y la mayoría de veces inexistentes. Dichos datos son los movimientos inerciales que se producen en la toma de datos que estas disciplinas necesitan para su estudio y desarrollo.

Una unidad de medición inercial o IMU (del inglés Inertial Measurement Unit) es un dispositivo electrónico cuyo objetivo es obtener mediciones de velocidad, rotación y fuerzas gravitacionales de un aparato de forma autónoma, usando una combinación de acelerómetros y giróscopos. Se utilizan como componentes fundamentales en los sistemas de navegación de barcos, aviones, helicópteros, transbordadores, satélites o cualquier móvil en que sean precisos estos datos sin posibilidad alguna de obtenerlos de forma externa. Los últimos estudios han permitido, junto la inclusión de una IMU, producir dispositivos GPS que no se ven afectados por la interferencia electromagnética.

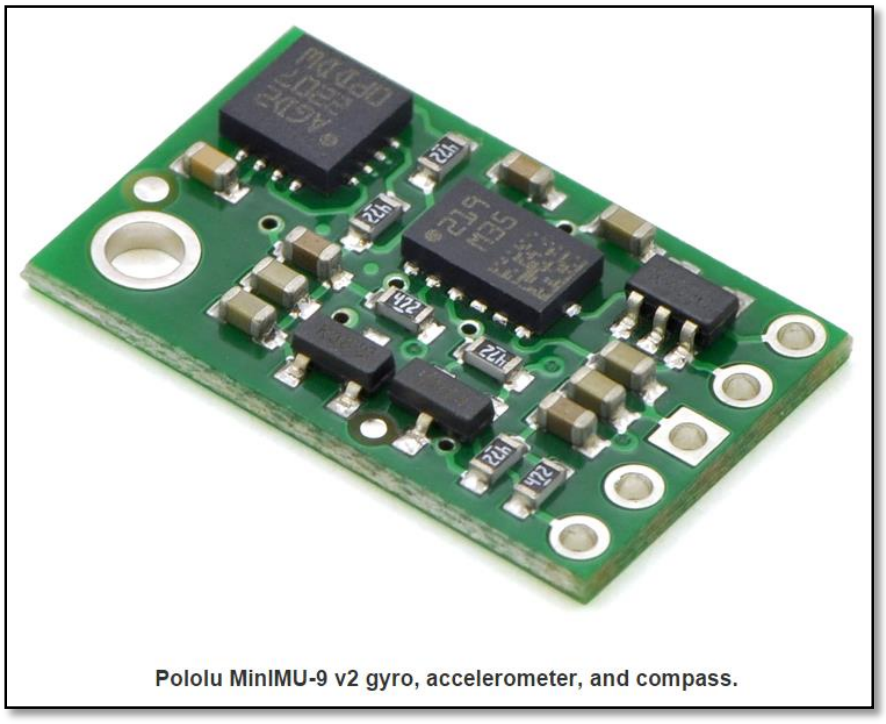
Una IMU suele estar compuesta por un conjunto de acelerómetros y giróscopos, que obtienen datos de uno o más ejes ortogonales (dependiendo de las especificaciones de la IMU), los cuales son controlados y administrados por un microprocesador, lo que se conoce como sistema embebido (en este proyecto el sistema embebido lo formará la Raspberry), para así realizar los cálculos necesarios para obtener las estimaciones de aceleración y velocidad de rotación requeridas o pivotación (Yaw, Pitch y Roll).

Como se ha comentado, la IMU es un componente esencial para la navegación para diferentes sistemas pero presenta especiales inconvenientes para su uso. Una de las grandes desventajas de usar una IMU para la navegación es que las mediciones que realiza son afectadas por un error acumulativo. Debido a que en la navegación por estima el sistema se guía continuamente agregando los cambios detectados a las posiciones previamente calculadas, cualquier error en la medición, se van acumulando de punto a punto. Esto genera una importante deriva o diferencia que aumenta siempre entre la posición real del sistema y la posición teórica calculada por el conjunto de la IMU. Es por ello, que para corregir o mitigar estos errores producidos por la IMU, los sistemas de navegación se apoyan en otros instrumentos como el GPS. Para usos más específicos, también se incluyen sensores de gravedad (para la corrección de la vertical local), sensores de velocidad externos (para compensar la deriva por velocidad), un sistema barométrico (para la corrección de la altitud y un compás magnético).

Pero la cuestión inevitable es: ¿Cómo funciona realmente una IMU? y ¿Qué relación existe entre todos sus componentes?
Para responder a esta cuestión es imprescindible entrar en campos que pueden resultar extraños como el de la electrónica, pues básicamente, lo que veremos será todo el proceso necesario para convertir los impulsos eléctricos (valores de ganancia) generados por los chips que conforman la IMU en valores que sean reconocibles.

Por tanto, se intentará explicar:

- Que es un acelerómetro
- Que es un giróscopo
- Como convertir las lecturas analógicas y digitales (ADC) obtenidas por el sensor, en medidas que tengan un significado físico (normalmente suele ser unidades g para el acelerómetro y deg/s para el giroscopio)
- Como combinar las lecturas de acelerómetro y giroscopio para obtener información de la posición de los ejes del dispositivo



Pololu MinIMU-9 v2 gyro, accelerometer, and compass.

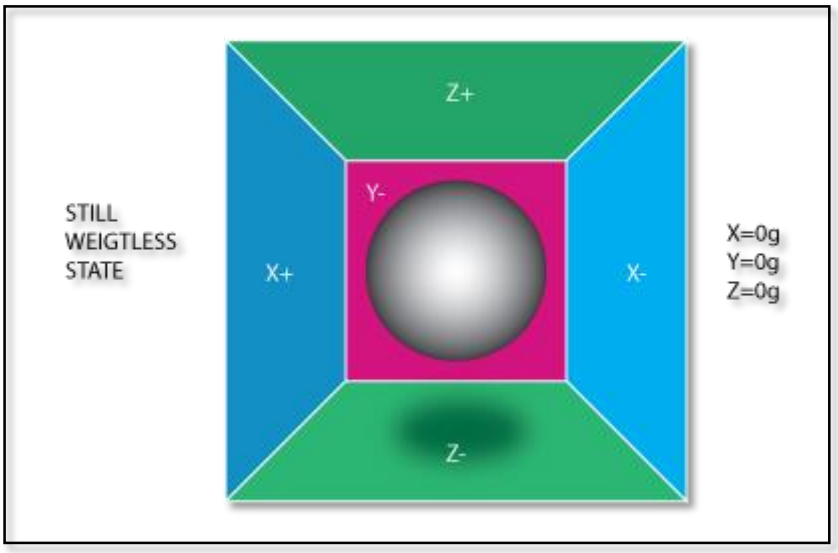
Con los conocimientos básicos acerca de las funciones básicas del Sen/Coseno/Tangente no deberíamos tener problemas para comprender las funciones que nos permitirán obtener los valores físicos que se quiere obtener. Se podría decir que para usar una IMU no es necesario utilizar matemáticas muy complejas ni tampoco se necesitará utilizar filtros complejos FIR[5], filtro IIR[6] y filtros como los de Kalman, Parks-McClellan.

El objetivo de buscar esa simplicidad en su ejecución, radica en que un sistema que es simple, es más fácil de controlar y supervisar. Sin embargo, se deberá recurrir a sistemas embebidos, pues los elementos que componen una IMU no tienen la capacidad ni los recursos para implementar algoritmos complejos que requieren cálculos matriciales.

[5] Intersil$_{TM}$ Complex Filtering: http://www.intersil.com/content/dam/Intersil/documents/an94/an9418.pdf
[6] Francisco J.García.Filtro IIR: http://www.ieesa.com/universidades/tesis01/capt3b.pdf

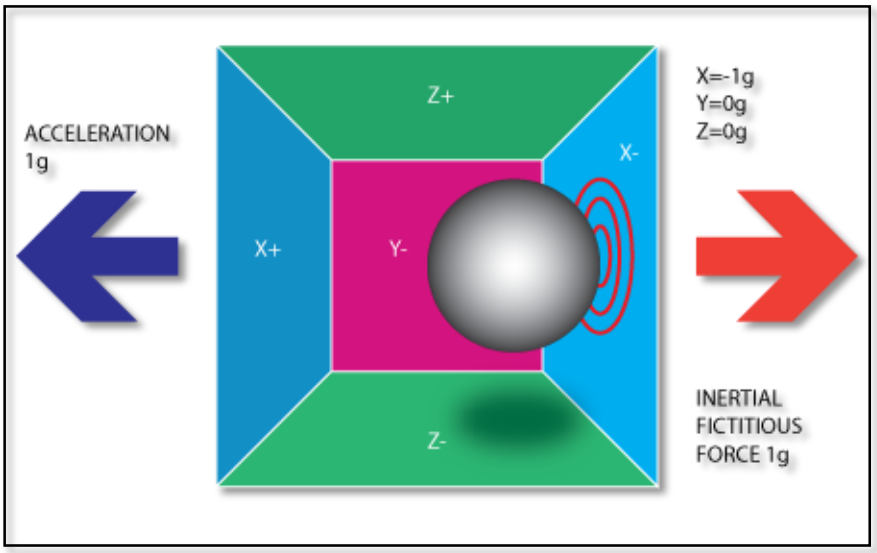
1.1.1. Acelerómetro

Cuando se piensa en un acelerómetro, es común imaginarse un cubo con una bola en su interior tal y como se ve en la siguiente imagen:



Si se considera que el cubo no se encuentra afectado por el campo de gravitación o para el caso, sin otros campos que puedan afectar a la posición de la bola, dicha bola flotaría en el interior del cubo. Así, se puede imaginar que el cubo se encuentra en el espacio exterior, lejos de cualquier cuerpo cósmico o, si el cubo se encontrase en el interior de la Estación Espacial Internacional orbitando alrededor de la Tierra en estado de ingravidez.

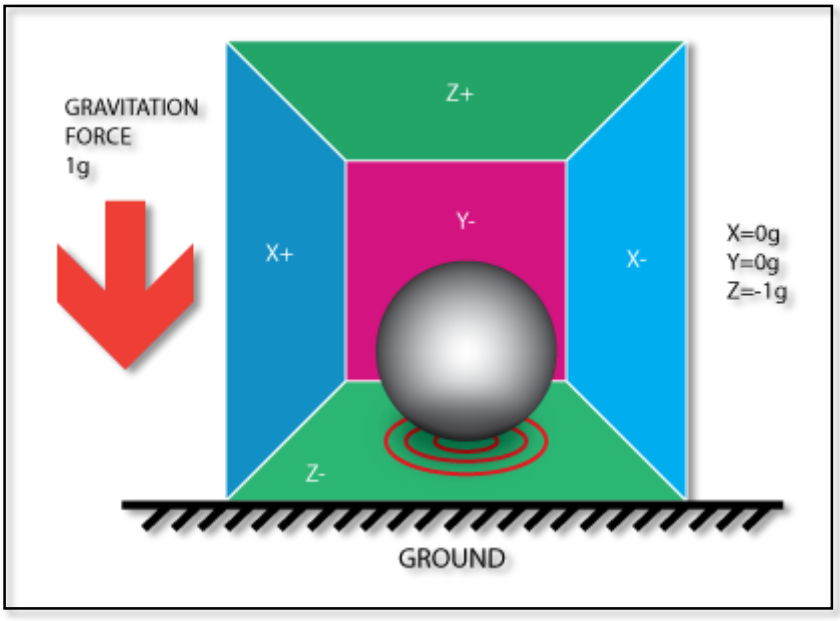
Vemos como a partir de la imagen superior podemos asignar a cada eje (X, Y, Z) un par de paredes del cubo. Se supone, que cada pared es sensible a la presión que la bola en su interior sea susceptible de aplicar. Si movemos de forma repentina el cubo hacia la izquierda (aceleración de $1g = 9'8\ m/s^2$), la bola golpeará la pared X-. Será en este momento cuando la bola aplicará una presión en la pared de 1g, de forma que se obtendrá un vector de salida de 1g en el eje X.



Nótese como el acelerómetro detectará una fuerza en la dirección opuesta (flecha roja) a la dirección del vector aceleración (flecha azul). Esta fuerza se suele llamar Fuerza Inercial o

Fuerza Ficticia. Por esta razón, un aspecto a tener en cuenta es que las lecturas del acelerómetro miden la aceleración indirectamente gracias a una fuerza que se aplica a una de las paredes del cubo que no hemos imaginado anteriormente. Esta fuerza está causada por la aceleración, pero como veremos más adelante no siempre es así.

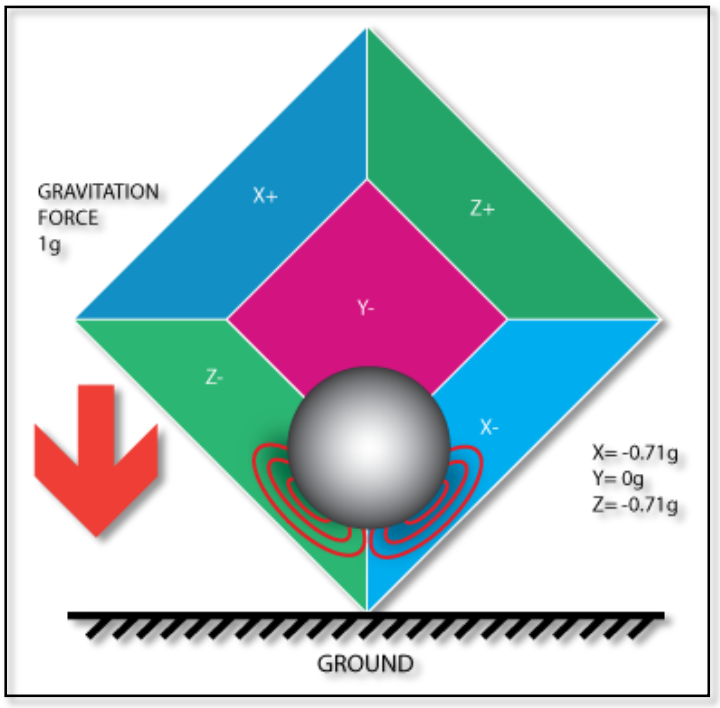
Si tomamos nuestro modelo anterior y lo ponemos en sobre la Tierra, la bola caerá sobre la pared Z-suelo y aplicará una fuerza de 1g sobre la pared, tal como se muestra en la imagen inferior:



En este caso, el cubo no se está moviendo, pero el acelerómetro está obteniendo lecturas de -1g en el eje Z. La presión que la bola ejerce sobre la pared del cubo es causada por la fuerza de la gravitación. En teoría, esto podría ser producido por otro tipo diferente de fuerza, por ejemplo, si la bola fuese metálica y colocásemos un imán a un lado del cubo, podríamos mover la bola hasta otra pared del cubo. Con este sencillo ejemplo, lo que se pretende es comprender, que en esencia el acelerómetro mide la fuerza aplicada, no la aceleración. Lo que sucede es que la aceleración causa una fuerza inercial que es medida por el acelerómetro.

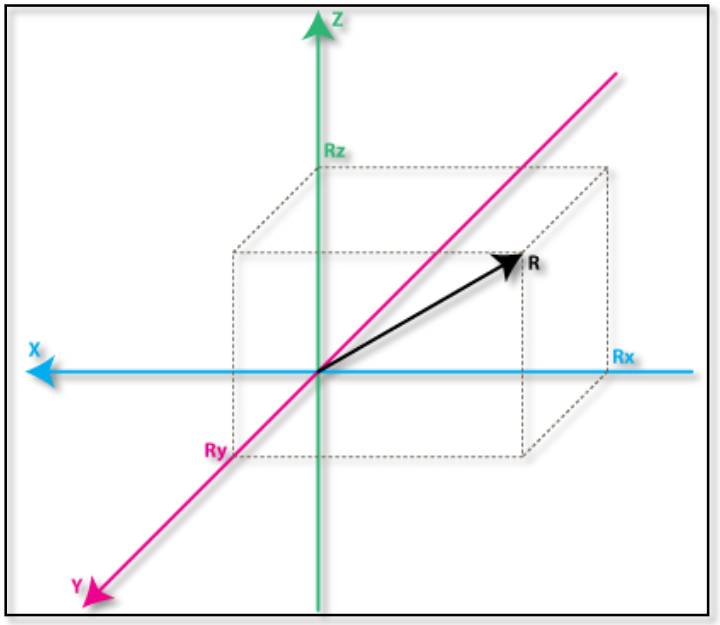
Hasta el momento, se ha analizado la salida del acelerómetro en un solo eje, y esto sería suficiente si se trabajase con un sensor de un solo eje, pero este no es nuestro caso. El valor real de los acelerómetros triaxiales se obtiene a partir de las fuerzas de inercia de los tres ejes.

De vuelta a nuestro modelo, giramos el cubo 45 grados a la derecha. La bola tocará en este momento dos paredes, Z- y X- como se ve en la siguiente imagen:



El valor de aceleración obtenido de 0’71 g no es arbitrario sino una aproximación de SQRT(0’5). Esto se hará más evidente en el momento en el que se introduzca el próximo modelo del acelerómetro. En el modelo previo se fijó la fuerza de gravitación y se giró el cubo imaginario.

En los dos últimos ejemplos analizados se estudia la fuerza registrada por el magnetómetro con dos posiciones diferentes del cubo imaginario, mientras que la fuerza aplicada se mantuvo siempre constante. Estos ejemplos han sido útiles para comprender como el acelerómetro interactúa con las fuerzas externas a él. No obstante, nos resultará más práctico para realizar los cálculos que necesitemos si fijamos el sistema de coordenadas a los ejes del acelerómetro e imaginamos que los vectores de la fuerza rotan alrededor de nuestro sistema de coordenadas.



Si nos fijamos en el modelo anterior, vemos que los tres ejes representados (X, Y, Z), son perpendiculares a las caras del anterior modelo utilizado (el cubo y la bola imaginarios). En el nuevo modelo se ve representado un vector R, que podría ser el vector de la fuerza medido por el acelerómetro (bien puede ser la fuerza gravitacional o una fuerza inercial o una combinación de ambas a la vez). Por tanto, Rx, Ry y Rz son proyecciones del vector R en los ejes X, Y Z. Esto se expresa en la siguiente relación:

$$R^2 = Rx^2 + Ry^2 + Rz^2 \qquad (1)$$

(*) Esta ecuación es básicamente el Teorema de Pitágoras en 3 dimensiones.

Después de toda la teoría introductoria vamos a centrarnos en los datos numéricos. Los valores obtenidos de Rx, Ry y Rz están linealmente relacionados con los valores obtenidos del acelerómetro y que posteriormente se pueden utilizar para los diversos cálculos que se consideren necesarios.

Antes de empezar a realizar estos cálculos se debe conocer de qué forma los acelerómetros entregan la información. La mayoría de los acelerómetros son digitales o analógicos. Los acelerómetros darán la información usando un protocolo de comunicación específico como I2C, SPI o UART (en este caso utilizaremos el puerto I2C) y tendrán como salida un nivel de tensión dentro de un rango predefinido que deberá convertirse en un valor digital usando un módulo ADC[7].

No se va a entrar en mucho detalle acerca de la transformación de los módulos ADC, en parte porque es un tema muy amplio y además porque existen grandes diferencias de una plataforma a la otra. Existen algunos microcontroladores que tienen el módulo ADC incorporado, aunque también existen microcontroladores que necesitan de componentes externos para realizar las conversiones ADC. Ahora bien, independientemente del módulo ADC utilizado se tendrá un valor de voltaje con un cierto rango. Por ejemplo un módulo ADC de 10 bits tendrá como salida un valor que se encontrará en el rango de 0 - 1023. Se ha de considerar que 1023 = 2^{10} -1. Un módulo ADC de 12 bits de salida será tendrá un valor en el rango de 0 - 4095, sabiendo que 4095 = 2^{12}-1.

Para una mejor comprensión de lo que se intenta explicar, pasamos a considerar un ejemplo sencillo. Supongamos que tenemos un módulo ADC de 10 bits el cual no da los siguientes valores para los tres canales del acelerómetro (ejes X, Y, Z):

$$ADC_{Rx} = 586$$
$$ADC_{Ry} = 630$$
$$ADC_{Rz} = 561$$

Cada módulo ADC tendrá una tensión de referencia, que en este caso asumiremos de 3’3V. Para convertir un valor de 10 bits ADC de voltaje seguimos la siguiente formula:

$$V_{Rx} = \frac{ADC_{Rx} \cdot V_{Ref}}{1023}$$

[7] ADC: Analogic Digital Converter o Convertidor Analógico Digital)

Aplicando esta fórmula a los tres canales tenemos:

$$V_{Rx} = \frac{586 \cdot 3'3V}{1023} \cong 1'89V$$
$$V_{Ry} = \frac{630 \cdot 3'3V}{1023} \cong 2'03V$$
$$V_{Rz} = \frac{561 \cdot 3'3V}{1023} \cong 1'81V$$

Cada acelerómetro tendrá un nivel de voltaje de 0g, el cual deberemos buscar en las especificaciones del acelerómetro para saber cuál es ese nivel de tensión correspondiente a 0g.

Supongamos entonces que el nivel de voltaje de 0g es 1'65 V. Se calcula entonces el cambio de voltaje del siguiente modo:

$$\delta V_{Rx} = 1'89V - 1'65V \cong 0'24V$$
$$\delta V_{Ry} = 2'03V - 1'65V \cong 0'38V$$
$$\delta V_{Rz} = 1'81V - 1'65V \cong 0'16V$$

Ahora las lecturas del acelerómetro están en voltios, aunque todavía no están unidades de gravedad (9,8 ms^2). Para hacer la conversión final aplicamos la sensibilidad del acelerómetro, normalmente expresada en mVg. Se establece que la sensibilidad es de 478.5mVg o lo que es lo mismo 0.4785Vg. Estos valores de sensibilidad también se pueden encontrar en las especificaciones del acelerómetro. Para obtener los valores de fuerza final expresados en g, utilizamos la siguiente fórmula:

$$R_x = \frac{\delta V_{Rz}}{Sensibilidad} \qquad (2)$$

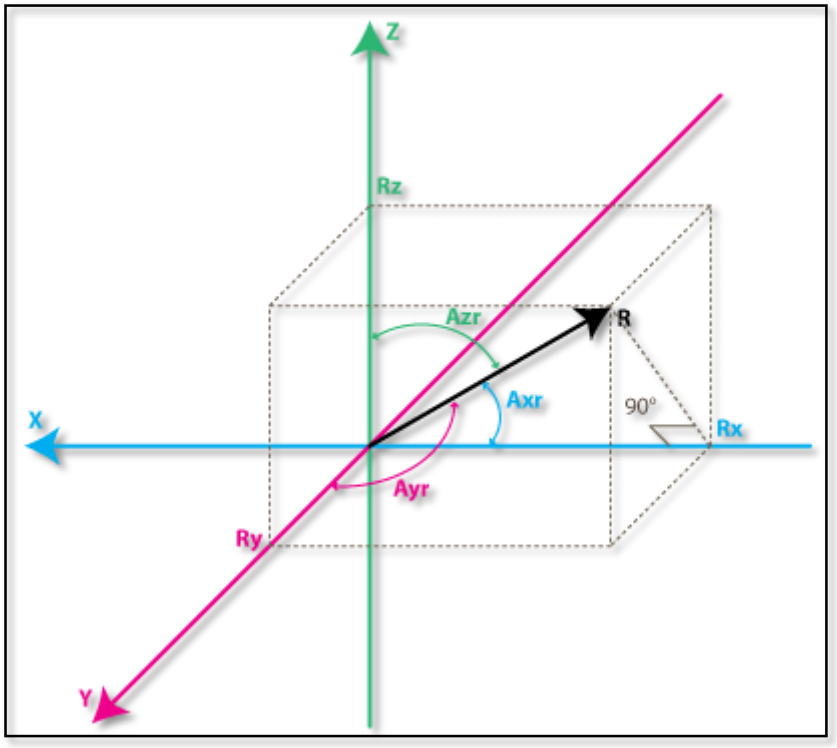
Quedando los tres canales del siguiente modo:

$$R_x = \frac{0'24V}{0'4785V_g} \approx 0'5g$$
$$R_y = \frac{0'38V}{0'4785V_g} \approx 0'79g$$
$$R_z = \frac{0'16V}{0'4785V_g} \approx 0'33g$$

Visto en detalle los pasos necesarios para convertir lecturas ADC en lecturas de fuerza expresadas en gravedad, ahora ya podríamos hacer todos los pasos en una sola ecuación que sería la siguiente:

$$R_x = \frac{\left(\frac{ADCR_x \cdot V_{Ref}}{1023 - V_{0g}}\right)}{Sensibilidad}$$

Una vez llegado a este punto ya se tienen los 3 componentes que definen el vector de fuerza inercial. Si el dispositivo no se encuentra afectado por otras fuerzas de gravitación, se puede asumir que esta es la dirección del vector de la fuerza de gravitación. Ahora se estaría en disposición de obtener la inclinación del dispositivo respecto al suelo del mismo, no sería difícil calcular el ángulo entre este vector de fuerza y el eje Z. Si fuese necesario, también se podría calcular la dirección al eje de inclinación, dividiendo el resultado en 2 componentes: la inclinación en el eje X y Y que puede calcularse como el ángulo entre el vector de gravitación y los ejes X/Y. El cálculo de estos ángulos es más sencillo de lo que en un principio pueda parecer, ahora que se ha calculado los valores de Rx, Ry y Rz. Si se observa el nuevo modelo de acelerómetro, se podrá hacer alguna anotación interesante:



Los ángulos que son de especial interés serían los que se encuentran entre los ejes X, Y, Z, junto con el vector de la fuerza, el vector R. Se definen estos ángulos como Axr, Ayr y Azr. Se puede ver el triángulo rectángulo formado por R y Rx que vendría definido por:

$$Cos(Axr) = \frac{R_x}{R}$$
$$Cos(Ayr) = \frac{R_y}{R}$$
$$Cos(Azr) = \frac{R_z}{R}$$

Se puede deducir de la ecuación 1 lo siguiente:

$$R = \sqrt{({R_x}^2 + {R_y}^2 + {R_z}^2}$$

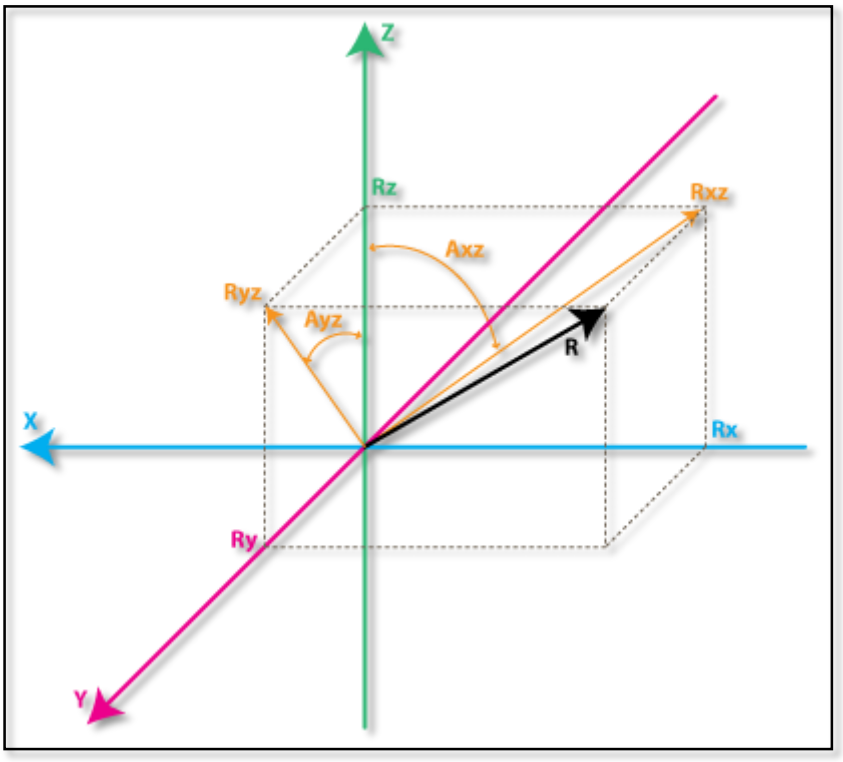
Por lo que ahora se pueden encontrar los ángulos mediante la ecuación ArcCos:
Podemos encontrar ahora nuestros ángulos mediante el uso de arccos () función (la función cos () inversa):

$$Axr = \cos^{-1}\left(\frac{R_x}{R}\right)$$
$$Ayr = \cos^{-1}\left(\frac{R_y}{R}\right)$$
$$Azr = \cos^{-1}\left(\frac{R_z}{R}\right)$$

Después de todo este proceso, pasando por los diversos modelos del acelerómetro, se ha llegado a estas fórmulas. Dependiendo de que aplicación se quiera desarrollar, se puede utilizar cualquier de las fórmulas intermedias aquí desarrolladas.

1.1.2. Giroscopio

En este caso, no se va a introducir un modelo como en el caso anterior de un cubo y una bola, sino que se pasará directamente a mostrar qué mediría un giroscopio.



Cada canal del giroscopio mide la rotación alrededor de uno de los ejes. Por ejemplo, medir la rotación alrededor de un giroscopio de dos ejes, el eje X y el eje Y. Para expresar estas rotaciones en los ejes, se deben definir antes algunos conceptos previos o anotaciones:

- R_{XZ}: es la proyección del vector de la fuerza inercial R sobre el plano XZ
- R_{yz}: es la proyección del vector de la fuerza inercial R sobre el plano YZ

Desde el triángulo rectángulo formado por R_{XZ} y $R_{Z,}$ si se usa el Teorema de Pitágoras se obtiene:

$$R_{xz}{}^2 = R_x{}^2 + R_z{}^2$$

Y del mismo modo:

$$R_{yz}{}^2 = R_y{}^2 + R_z{}^2$$

También debemos tener en cuenta que:

$$R^2 = R_{xz}{}^2 + R_y{}^2$$

Esta se deriva de la ecuación 1 anteriormente nombrada, o también puede ser deducida del triángulo rectángulo formado por R y R_{yz}.

$$R^2 = R_{yz}{}^2 + R_x{}^2$$

Llegado a este punto, se definirá el ángulo entre el eje Z y los vectores R_{xz} y R_{yz} como sigue:

- A_{xz}: es el ángulo medido entre el vector R_{xz} (proyección de R sobre plano XZ) y el eje Z.
- A_{yz}: es el ángulo medido entre el vector R_{yz} (proyección de R sobre plano YZ) y el eje Z.

Es en este punto cuando se va viendo que es lo que mide el giróscopo. El giróscopo mide la tasa de cambio de los ángulos definidos anteriormente, es decir, el giroscopio dará la salida de un valor que está linealmente relacionado con la tasa de cambio de estos ángulos. Para poder explicar esto, se debe asumir que se ha medido el ángulo de rotación alrededor del eje Y en el tiempo t0, y se definiría como $A_{xz}0$. Posteriormente se mide este ángulo en un tiempo t1 y lo llamaremos $A_{xz}1$. La tasa de cambio se mide como sigue:

$$RateA_{xz} = \frac{(A_{xz}1 - A_{xz}0)}{(t1 - t0)}$$

Si expresamos A_{xz} en grados y el tiempo en segundos, este valor se expresará en deg/s. Esto sería lo que el giroscopio mediría.
En la práctica un giroscopio (salvo que sea un giroscopio digital especial) raramente dará un valor expresado en unidades deg/s. Del mismo modo que en el acelerómetro se obtendrá un valor ADC que se deberá convertir en deg/s empleando una fórmula similar a la ecuación 2 que se ha definido anteriormente para el acelerómetro. Se podrá introducir el valor ADC a la fórmula de conversión de deg/s del giroscopio (se supone que se utiliza un módulo de 10 bits ADC, para 8 bits ADC se ha de reemplazar el valor 1023 por 255, para 12 bits ADC Reemplace 1023 por 4095).

$$RateA_{xz} = \frac{\frac{(ADCGyro_{xz} \cdot V_{Ref})}{(1023 - V_{zeroRate})}}{Sensibilidad} \qquad (3)$$

$$RateA_{yz} = \frac{\frac{(ADCGyro_{yz} \cdot V_{Ref})}{(1023 - V_{zeroRate})}}{Sensibilidad}$$

- ADC_{GyroXZ}, ADC_{GyroYZ}: se obtienen del módulo ADC empleado y que representan los canales que medirán la rotación de la proyección del vector R en XZ respectivamente en los planos YZ, que resulta ser el equivalente a la rotación producida en los ejes Y X respectivamente.
- V_{ref}: es la tensión de referencia del ADC si el voltaje utilizado es 3.3V
- $V_{zeroRate}$: es el Ratio-Cero del voltaje, en otras palabras, es el voltaje de salida del giroscopio cuando no está sujeto a cualquier rotación(está en reposo), por ejemplo, para AccGyro el voltaje sería de 1.23V (estos valores estándar se pueden encontrar en las especificaciones del giroscopio), no obstante, estas especificaciones pueden variar una vez que es giroscopio se ha manipulado, tal como soldarse en una placa, hecho que ha sido comprobado por diferentes giroscopios en diferentes situaciones, al apreciarse como el $V_{zeroRate}$ variaba. Este problema no será una preocupación en este proyecto, pues la IMU de Pololu viene con todos estos elementos integrados y se manipulan de la forma más cuidadosa tratando que estas situaciones no se produzcan.
- Sensibilidad: es la sensibilidad del giroscopio expresada en mV/ (deg/s) a menudo escrito como mV/deg/s. Básicamente esta medida nos dice cuántos mV aumentará la salida de la lectura del giroscopio, si aumenta la velocidad de rotación en 1 deg/s. La sensibilidad del AccGyro, por ejemplo, es 2mV/deg/s o 0.002V/deg/s.

Para terminar de comprender esta teoría expuesta, se expondrá un sencillo ejemplo. Se supone que se tiene un módulo ADC que devuelve los siguientes valores:

$$ADC_{GyroXZ} = 571$$
$$ADC_{GyroYZ} = 323$$

Utilizando las fórmulas anteriores y los parámetros de las especificaciones del AccGyro se obtiene:

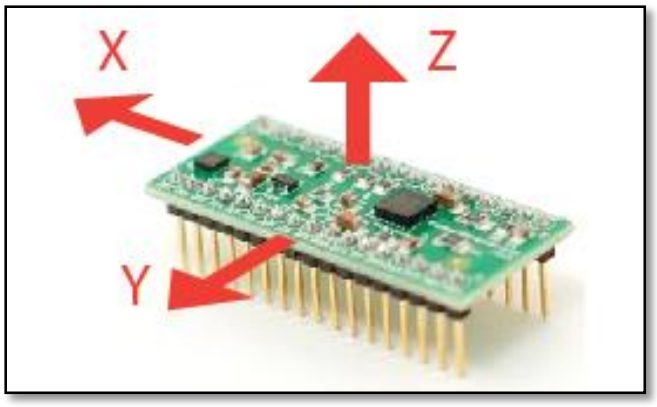
$$RateA_{xz} = \frac{\frac{(571 \cdot\ 3'3V\ )}{(1023 -\ 1'23V)}}{\frac{0'002V}{deg}/s} \cong 306\ deg/s$$

$$RateA_{xz} = \frac{\frac{(323 \cdot\ 3'3V\ )}{(1023 -\ 1'23V)}}{\frac{0'002V}{deg}/s} \cong -94\ deg/s$$

En otras palabras el dispositivo gira alrededor del eje Y (o en el plano XZ) con una velocidad de 306 deg/s y alrededor del eje X (o en el plano YZ) con una velocidad de degs-94. Hay que tener en cuenta que el signo negativo significa que el dispositivo está girando en la dirección opuesta al sentido positivo, que ha sido así decidido por convención. Una buena hoja de especificaciones de un giroscopio nos indicaría que dirección es considerada positiva, de lo contrario, habría que ir probando con el dispositivo y viendo que lecturas nos devolvía. Si se dispone de un multímetro, se puede calcular también esto, si se mantiene una velocidad de rotación constante durante al menos unos segundos, y observando la tensión durante esta rotación, se puede comparar posteriormente con el $V_{zeroRate}$. Si es mayor que el voltaje del Ratio-Cero, significa que la que la dirección de la rotación es positiva.

1.1.3. Combinación de datos de acelerómetro y giroscopio.

Si nuestro dispositivo no tuviese todos los elementos integrados (no es el caso de la IMU que se ha utilizado, que lleva todos los elementos integrados en un mismo dispositivo), el primer paso en el uso de un dispositivo IMU en el que debemos combinar un acelerómetro, un giroscopio y un magnetómetro (como la IMU de Pololu) es alinear sus sistemas de coordenadas. La forma más sencilla de hacerlo es eligiendo el sistema de coordenadas del acelerómetro como el sistema de coordenadas de referencia. En las especificaciones técnicas del acelerómetro se especificará la dirección de los ejes X, Y, Z en relación con el chip físico o el dispositivo. Por ejemplo, aquí están las direcciones de los ejes X, Y, Z en una IMU de ejemplo:



Los pasos a seguir son:

- Identificar la salida del giroscopio que corresponden al $RateA_{xz}$, $RateA_{yz}$
- Determinar si estas salidas deben invertirse debido a la posición física del giroscopio en relación con el acelerómetro. Un giroscopio tiene una salida marcada con X o Y, que corresponderá a cualquiera de los ejes en el sistema de coordenadas del acelerómetro. De modo que, se deberá realizar una toma de datos para determinar que la salida del giroscopio corresponde al valor de $RateA_{xz.}$
- Tomamos como punto de partida la situación del dispositivo en una posición horizontal, para medir el voltaje de salida correspondiente al valor zero-g.
- Se girará el dispositivo alrededor del eje Y, otra forma de decirlo, se gira el dispositivo en el plano XZ, de modo que, la salida de las lecturas del acelerómetro en los ejes X y Z variaran con el giro, mientras que el eje Y permanecerá constante.
- Mientras se gira el dispositivo a una velocidad constante la salida del giroscopio sufre cambios, mientras que las otras salidas del giroscopio deben permanecer constantes.
- La salida del giroscopio que fue cambiando durante la rotación alrededor del eje Y proporcionará el valor de entrada para el ADC_{GyroXZ}, de la cual se calculó el $RateA_{xz}$
- El paso final es garantizar que el sentido de giro corresponde con el sentido de giro de nuestro modelo. En algunos casos es posible que se tenga que invertir el valor de $RateA_{xz}$ debido a la posición física del giroscopio en relación con la posición del acelerómetro.
- Si se realiza nuevamente la prueba anterior, al girar el dispositivo alrededor del eje Y, se deberá controlar la salida del eje X del acelerómetro (ADC_{Rx} en nuestro modelo). Si el valor del ADC_{Rx} crece (en los primeros 90º de rotación desde la posición horizontal), entonces debería disminuir el valor de ADC_{GyroXZ}. Esto es debido a que se está monitorizando el vector de gravedad y cuando el dispositivo gira en una dirección el vector gira en sentido opuesto (en relación con el sistema de coordenadas del dispositivo). Por lo tanto, será necesario invertir el de lo contrario necesitará invertir el $RateA_{xz}$. Esto se consigue mediante la introducción de una factor en la ecuación 3, tal como mostramos a continuación:

$$RateA_{xz} = InversA_{xz} \cdot \frac{\left(ADCGyro_{xz} \cdot V_{Ref}\right)}{(1023 - V_{zeroRate})} \over Sensibilidad$$

Donde $InversA_{xz}$ tiene un valor de -1 o 1.

El mismo test puede hacerse para $RateA_{yz}$, haciendo girar el dispositivo alrededor del eje X, de modo que puede identificarse que salida de giroscopio corresponde a $RateA_{yz}$, y si en este caso es necesario que se invierta. Una vez conocido el valor para $InversA_{yz}$, se debe utilizar la siguiente fórmula para calcular el $RateA_{yz}$:

$$RateA_{yz} = InversA_{yz} \cdot \frac{\left(ADCGyro_{yz} \cdot V_{Ref}\right)}{(1023 - V_{zeroRate})} \over Sensibilidad$$

Si se hacen pruebas acerca del AccGyro, se deben obtener los siguientes resultados:

- el pin de salida para $RateA_{xz}$ es GX4 y $InversA_{xz}$ 1
- el pin de salida para $RateA_{yz}$ es GY4 y $InversA_{yz}$ 1

Llegado a este punto se debe considerar que la configuración de la IMU sea la correcta de modo que pueda calcular los valores correctos para Axr, Ayr, Azr y los valores $RateA_{xz}$, $RateA_{yz}$. Es importante analizar las relaciones entre estos valores, que resultan útiles para obtener la estimación más precisa de la inclinación del dispositivo con respecto al plano de tierra.

Un detalle interesante de la combinación de los diferentes elementos, es que, como se ha dicho anteriormente, el acelerómetro mide la fuerza inercial, y dicha fuerza puede ser causada por la gravitación, pero también puede ser causada por la aceleración o movimiento del dispositivo.

Como conclusión, incluso el acelerómetro estando en un estado relativamente estable, es todavía muy sensible a la vibración y el ruido mecánico en general. Esta es la razón principal por qué la mayoría sistemas IMU utilizan un giroscopio para suavizar cualquier errores de acelerómetro. Pero, ¿el giroscopio se encuentra libre de ruido?

Obviamente, el giroscopio no se encuentra libre de ruido porque mide la rotación, que es muy sensible a los movimientos mecánicos lineales, el tipo de ruido que sufre un acelerómetro, sin embargo los giroscopios tienen otros tipos de problemas como por ejemplo la deriva. Sin embargo, si se promedian los datos del acelerómetro y del giroscopio se puede obtener una estimación de la inclinación del dispositivo relativamente mejor que la que se obtendría utilizando los datos de los acelerómetros únicamente. Antes de desarrollar los algoritmos, se debe conocer algunos conceptos.

RACC: es el vector de fuerza inercial medida por el acelerómetro, que consta de los siguientes componentes (proyecciones en los ejes X, Y, Z):

$$R_xAcc = \frac{\left(ADCR_x \cdot V_{Ref}\right)}{(1023 - V_{zeroG})} \over Sensibilidad$$

$$R_yAcc = \frac{\left(ADCR_y \cdot V_{Ref}\right)}{(1023 - V_{zeroG})} \over Sensibilidad$$

$$R_zAcc = \frac{\left(ADCR_z \cdot V_{Ref}\right)}{(1023 - V_{zeroG})} \over Sensibilidad$$

Hasta ahora tenemos un conjunto de valores que podemos obtener exclusivamente de los valores de ADC de acelerómetro. A partir de ahora, utilizaremos la siguiente notación.

$$R_{Acc} = [R_x Acc, R_y Acc, R_z Acc]$$

Como estos componentes del R_{Acc} pueden obtenerse de los datos del acelerómetro, los podemos considerar como una entrada a nuestro algoritmo.
Hay que tener en cuenta que las medidas de la fuerza de gravitación R_{Acc}, será correcto asumir que la longitud de este vector definido es igual o cerca de 1 g.

$$|R_{Acc}| = \sqrt{(R_x Acc^2 + R_y Acc^2 + R_z Acc^2)}$$

Sin embargo debemos asegurarnos que tiene sentido actualizar este vector como sigue:

$$R_{Acc}(Normalized) = \left[\frac{R_x Acc}{|R_{Acc}|}, \frac{R_y Acc}{|R_{Acc}|}, \frac{R_z Acc}{|R_{Acc}|}\right]$$

Esto garantizará que la longitud del vector normalizado del R_{Acc} es siempre 1.
Lo siguiente sería introducir un nuevo vector que se llamará:

$$R_{est} = [R_x Est, R_y Est, R_z Est]$$

El vector, resulta ser la salida del algoritmo, estos son valores corregidos en base a datos obtenidos del giroscopio y basado en datos estimados anteriores.

El algoritmo realizará:

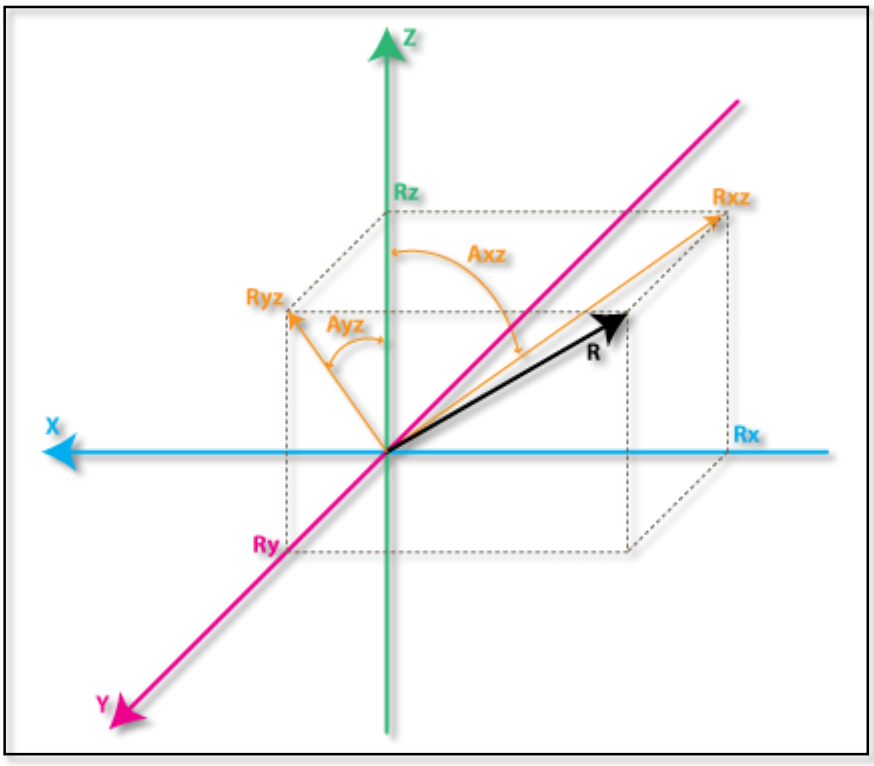
- El acelerómetro dará información acerca de la posición de Racc
- Se corregirá la información del acelerómetro con datos del giroscopio, así como con los últimos datos de reposo y salida de un nuevo vector R_{est}.
- Consideramos al R_{est} como una mejor opción en tanto en cuanto a la posición actual del dispositivo.

Posteriormente se deben hacer mediciones regulares a intervalos de tiempo iguales de T segundos y así obtener nuevas medidas que bien se pueden definir como R_{Acc} (1), R_{Acc} (2), R_{Acc} (3) y así sucesivamente. Así también se estimarán en intervalos de tiempo iguales el vector R_{est} (1), R_{est} (2), R_{est} (3) y así sucesivamente.
En este momento tenemos dos sets de valores:
R_{est} (n-1): Es la estimación anterior, con R_{est} (0) = R_{Acc} (0)
R_{Acc} (n): es la medición actual del acelerómetro

Antes de calcular R_{est} (n), se tiene que introducir un nuevo valor medido, que se puede obtener del giroscopio. Lo llamaremos R_{gyro}, y es también un vector de 3 componentes:

$$R_{gyro} = [R_x Gyro, R_y Gyro, R_z Gyro]$$

Se tiene que calcular este vector en un determinado instante. Empezaremos con R_xGyro.



Si observamos el modelo de giroscopio, podemos ver la relación del triángulo rectángulo formado por R_z y R_{xz} del cual podemos deducir que:

$$\tan(A_{xz}) = \frac{R_x}{R_z}$$

Por tanto:

$$A_{xz} = \tan^{-1}(R_x, R_z)$$

Si sabemos que R_xEst(n-1) y R_zEst(n-1) podemos deducir:

$$A_{xz}(n-1) = atan2\,(R_xEst(n-1), R_zEst(n-1))$$

Cabe recordar que el giroscopio mide la tasa de cambio del ángulo A_{xz}. Así podemos calcular el ángulo nuevo A_{xz}(n) como sigue:

$$A_{xz}(n) = A_{xz}(n-1) + \text{Rate}A_{xz}(\text{n}) \cdot \text{T}$$

Los valores RateA_{xz} pueden obtenerse a partir de las lecturas ADC del giroscopio. Una fórmula más exacta puede utilizar un índice de rotación promedio calculado como sigue:

$$RateA_{xz}A_{vg} = \frac{(RateA_{xz}(n) + RateA_{xz}(n-1))}{2}$$

$$A_{xz}(n) = A_{xz}(n-1) + RateA_{xz}A_{vg} \cdot T$$

Del mismo modo, también se puede encontrar como sigue:

$$A_{yz}(n) = A_{yz}(n-1) + RateA_{yz}A_{vg} \cdot T$$

Una vez obtenido $A_{xz}(n)$ y $A_{yz}(n)$, se deberá buscar la forma de deducir R_xGyro/R_yGyro a partir de la ecuación 1, de modo que la longitud del vector Rgyro se expresaría del siguiente modo:

$$|Rgyro| = \sqrt{(R_xGyro^2 + R_yGyro^2 + R_zGyro^2)}$$

Esto también se puede hacer debido a que se ha normalizado el vector R_{Acc}, por lo que suponemos que su longitud es 1 y no ha cambiado después de la rotación, así que se puede escribir:

$$|R_{Gyro}| = 1$$

Esto permite adoptar una notación temporal mucho más corta para los cálculos siguientes:

$$x = R_xGyro$$
$$y = R_yGyro$$
$$z = R_zGyro$$

Y utilizando la notación anterior se puede escribir:

$$x = \frac{x}{1} = \frac{x}{\sqrt{x^2 + y^2 + z^2}}$$

Podemos dividir ahora el numerador y denominador por $\sqrt{x^2 + y^2}$

$$x = \frac{\frac{x}{\sqrt{x^2 + y^2}}}{\frac{\sqrt{x^2 + y^2 + z^2}}{\sqrt{x^2 + y^2}}}$$

Debemos tener en cuenta que:

$\frac{x}{\sqrt{x^2+y^2}} = \sin(A_{xz})$, por tanto:

$$x = \frac{\sin(A_{xz})}{\sqrt{\frac{1 + y^2}{x^2 + z^2}}}$$

Ahora, multiplicaremos el numerador y el denominador de dentro de la raíz por z^2 quedando:

$$x = \frac{\sin(A_{xz})}{\sqrt{\frac{1 + y^2 \cdot z^2}{z^2 \cdot (x^2 + z^2)}}}$$

Nótese que:

$$\frac{y}{z} = \tan(A_{yz})^2 \qquad y \qquad \frac{z}{\sqrt{(x^2 + z^2)}} = \cos(A_{xz})$$

Por lo que finalmente:

$$x = \frac{\sin(A_{xz})}{\sqrt{\left(1 + \cos(A_{xz})^2 \cdot \tan(A_{yz})^2\right)}}$$

Llegado aquí, regresamos a la notación anterior:

$$R_x Gyro = \frac{\sin(A_{xz}(n))}{\sqrt{\left(1 + \cos(A_{xz}(n))^2 \cdot \tan(A_{yz}(n))^2\right)}}$$

Del mismo modo se tiene que:

$$R_y Gyro = \frac{\sin(A_{yz}(n))}{\sqrt{\left(1 + \cos(A_{yz}(n))^2 \cdot \tan(A_{xz}(n))^2\right)}}$$

Y finalmente:

$$R_z Gyro = Sign\,(R_z Gyro) \cdot \sqrt{(1 - R_z Gyro^2 - R_z Gyro^2)}$$

Donde:

$$Sign(R_z Gyro) = 1 \quad si \quad (R_z Gyro) \geq 0$$

O

$$Sign(R_z Gyro) = -1 \quad si \quad (R_z Gyro) < 0$$

Una forma sencilla de estimar que valor debe tomar es:

$$Sign(R_z Gyro) = Sign(R_z Est(n-1))$$

En la práctica hay que tener cuidado cuando R_zEst(n-1) está cerca de 0, pues se podría saltar una fase de giro en este caso y asignar: Rgyro = Rest(n-1).

Recapitulando, en este punto se tiene:
R_{Acc}: que corresponde a las lecturas de corriente de nuestro acelerómetro.
R_{gyro}: obtenida a partir de R_{est} (n-1) y las lecturas en el momento del giroscopio.

Ahora se deberá seleccionar que valores se utilizarán para obtener la estimación actualizada de $R_{est}(n)$. La respuesta es utilizar un promedio ponderado de lo anteriormente mencionado (R_{Acc} y R_{gyro})

$$R_{est}(n) = \frac{(R_{Acc} \cdot w1 + R_{gyro} \cdot w2)}{(w1 + w2)}$$

Esta fórmula se puede simplificar dividiendo numerador y denominador por la fracción de w1.

$$R_{est}(n) = \frac{\left(\frac{R_{Acc} \cdot w1}{w1} + \frac{R_{gyro} \cdot w2}{w1}\right)}{\left(\frac{w1}{w1} + \frac{w2}{w1}\right)}$$

Y posteriormente sustituir lo siguiente:

$$\frac{w2}{w1} = wGyro$$

Quedando entonces:

$$R_{est}(n) = \frac{(R_{Acc} + R_{gyro} \cdot w_{Gyro})}{(1 + w_{Gyro})}$$

En la formula anterior, wGyro nos dice cuánto giro se ha registrado con el giroscopio en comparación con el acelerómetro. Este valor puede ser elegido experimentalmente, aunque generalmente toma valores entre 5 y 20, que se considerarán buenos resultados.

La principal diferencia de este algoritmo desarrollado respecto al filtro de Kalman [8] es que los pesos se mantienen relativamente fijos, mientras que en el filtro de Kalman los pesos se actualizan permanentemente basado en el ruido medido de las lecturas del acelerómetro. El filtro de Kalman está enfocado en ofrecer los mejores resultados teóricos, pero, considerando que en una aplicación práctica también puede ofrecer buenos resultados. Se podría implementar un algoritmo que ajustase wGyro dependiendo de algunos factores del ruido que se mediría, pero los pesos fijos funcionan bien para la mayoría de las aplicaciones.

[8] El filtro de Kalman es un algoritmo desarrollado por Rudolf E. Kalman en 1960 que sirve para poder identificar el estado oculto (no medible) de un sistema dinámico lineal, al igual que el observador de Luenberger, pero sirve además cuando el sistema está sometido a ruido blanco aditivo.

Ahora, no resulta complicado conseguir los valores estimados actualizados como se verá a continuación:

$$R_xEst(n) = \frac{\left(R_xAcc + R_xGyro \cdot w_{Gyro}\right)}{\left(1 + w_{Gyro}\right)}$$

$$R_yEst(n) = \frac{\left(R_yAcc + R_yGyro \cdot w_{Gyro}\right)}{\left(1 + w_{Gyro}\right)}$$

$$R_zEst(n) = \frac{\left(R_zAcc + R_zGyro \cdot w_{Gyro}\right)}{\left(1 + w_{Gyro}\right)}$$

Ahora, se está en disposición de normalizar el vector como sigue:

$$R = \sqrt{R_xEst(n)^2 + R_yEst(n)^2 + R_zEst(n)^2}$$

$$R_xEst(n) = \frac{\left(R_xEst(n)\right)}{R}$$

$$R_yEst(n) = \frac{\left(R_yEst(n)\right)}{R}$$

$$R_zEst(n) = \frac{\left(R_zEst(n)\right)}{R}$$

Y con esto, repetiríamos el bucle una y otra vez.

Con toda esta explicación, se ha pretendido explicar el fundamento físico y matemático que subyace en la utilización de una IMU. Todo este desarrollo expuesto es en lo que se ha basado el autor y diseñador del software *minimu9-ahrs* que se utiliza en este proyecto para la toma de datos y gestión de los modos de ejecución de la misma. Es por ello, que aunque no sea algo que se vaya a utilizar explícitamente a lo largo del proyecto, sí que parecía coherente e interesante exponer todo el proceso deductivo que hay detrás del software utilizado y como trabaja el hardware existente.

En este punto, se está en disposición de presentar y comentar un poco, que IMU se va a emplear. Para este proyecto se ha utilizado el modelo de IMU MinIMU9[9] en su versión número 2 de Pololu.

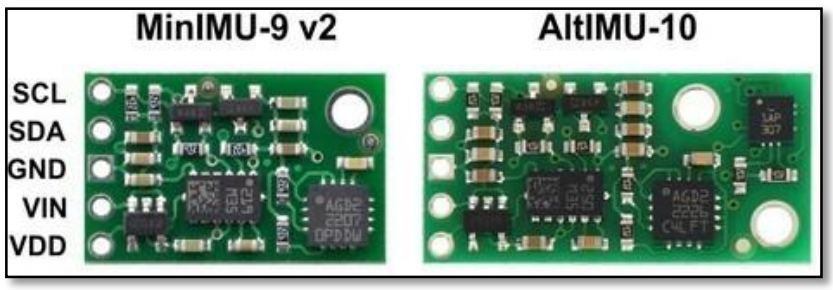
[9] MinIMU9 V2 : https://www.pololu.com/product/1268

El chip MinIMU-9 v2 de Pololu es una unidad de medición inercial (IMU) compuesto por un giroscopio de tres ejes (modelo L3GD20), un acelerómetro de 3 ejes (modelo LSM303DLHC) y un magnetómetro de 3 ejes en una pequeña placa de material cerámico de 0.8″ × 0.5″. Posee una interfaz I²C que permite el acceso a nueve rotaciones, aceleración y mediciones magnéticas de forma independiente (en otras palabras, tiene nueve grados de libertad) que pueden utilizarse para calcular la orientación absoluta del sensor. El módulo incluye un regulador de voltaje y un circuito de cambio de nivel que permite la operación de 2.5 a 5.5 V.

Dimensiones	
Tamaño	0.8" x 0.5" x .1"
Peso	0.7 g

Especificaciones generales	
Interfaz	I²C
Mínimo voltaje operativo	2.5 V
Máximo voltaje operativo	5.5 V
Ejes	pitch (x), roll (y), and yaw (z)
	±250, ±500, ±2000 º/s (gyro)
Rango de medida	±2, ±4, ±6, ±8, ±16 g (acelerómetro)
	±1.3, ±1.9,± 2.5, ±4.0, ±4.7, ±5.6 o ±8.1 gauss (magnetómetro)
Fuente de alimentación	10 mA

La IMU es capaz de operar con voltajes por debajo de 3’5 V, lo cual puede dificultar la comunicación con otros microcontroladores que funcionen a 5 V. El chip MinIMU-9 v2 aborda estas cuestiones mediante la incorporación de componentes electrónicos adicionales, como un regulador de voltaje y un circuito de cambio de nivel, manteniendo el tamaño tan compacto como sea posible.



El chip MinIMU-9 v2 es compatible mediante un pin con el AltIMU-10 (no empleado en este proyecto), que ofrece la misma funcionalidad, pero ampliando las capacidades del sensor, gracias al barómetro digital que puede utilizarse para obtener mediciones de presión y la altitud.

El AltIMU incluye un segundo orificio de montaje y es más largo que el chip MinIMU-9 v2 en 0.2″. Cualquier código escrito para el chip MinIMU-9 v2 también debería funcionar con el chip AltIMU-10

El giroscopio L3GD20 y el acelerómetro LSM303DLHC tienen muchas opciones configurables, incluyendo sensibilidades dinámicamente seleccionables para el giroscopio, acelerómetro y magnetómetro, así como una gran variedad de tipos de datos de salida para cada sensor. Con un algoritmo apropiado, un microcontrolador o un ordenador pueden utilizar los datos para calcular la orientación exacta del dispositivo. Por otro lado, el giroscopio puede utilizarse para obtener los giros que se producen en el recorrido con una gran precisión y en intervalos de tiempo muy pequeño, mientras que el acelerómetro y la brújula pueden ayudar a compensar la deriva de giro en el tiempo transcurrido en un marco de referencia absoluto. Los respectivos ejes de los dos chips se alinean en la placa para facilitar los cálculos cuando se fusionan los datos obtenidos de ambos chips.

Una vez expuesto todo el fundamento físico y matemático del funcionamiento de una IMU y detallado todas las especificaciones del sensor IMU que se va a emplear, llega el momento de hacerse la siguiente pregunta: ¿Cómo se gestiona el sensor y toda la información proporcionada?

Esta no era una cuestión sencilla de responder, y dado las limitaciones existentes en los conocimientos de programación se optó por un lenguaje sencillo, a la par que anárquico, lo que a veces resulta contraproducente. También has surgido otros problemas

1.2. Qué es Python y el porqué de su utilización

Python se remonta hacia finales de los 80 principio de los 90 y su implementación empezó a ser una realidad en diciembre de 1989, cuando en Guido Van Rossum decidió empezar el proyecto. Es un lenguaje con una sintaxis muy limpia y que favorece un código legible y que se enfoca en ser fácil de usar y aprender. En sus inicios el proyecto no trascendió como se esperaba. Van Rossum es por tanto el autor principal de Python y continúa ejerciendo un rol central decidiendo la dirección del lenguaje. El nombre "Python" viene dado por la afición de Van Rossum al grupo Monty Python.

Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad, problema que en ciertas ocasiones es latente en nuestro proyecto. Aunque la lentitud que se comenta, también viene deriva da las limitaciones de la Raspberry, la cual tiene un procesador muy modesto.

Un lenguaje interpretado o de script es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directamente un ordenador (un lenguaje compilado sería seria C+). L mayor ventaja que presenta un lenguaje compilado es que su ejecución es mucho más rápida, pero en cambio, los lenguajes interpretados son más flexibles y portables.
Python tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi interpretado. En Python (en Java también), el código fuente se traduce a un pseudocódigo máquina intermedio llamado bytecode. La primera vez que se ejecuta, genera archivos unos archivos .pyc o .pyo (bytecode optimizado), que son los que se ejecutarán en sucesivas ocasiones.

Python es un lenguaje multiplataforma, esto quiere decir que existen versiones disponibles de Python en muchos sistemas informáticos distintos (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.). Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él. Si no se utilizan utilizamos librerías específicas de cada plataforma el programa que confeccionemos se podrá ejecutar en todos estos sistemas sin grandes cambios.

Además, Python resulta un lenguaje muy interactivo, al disponer de un intérprete por línea de comandos en el que se pueden introducir sentencias. Cada sentencia se ejecuta y produce un resultado visible, que puede ayudarnos a entender mejor el lenguaje y probar los resultados de la ejecución de porciones de código rápidamente.

Otra característica esencial de Python es que es un leguaje orientado a objetos la cual ofrece en muchos casos una manera sencilla de crear programas con componentes reutilizables. La orientación a objetos es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y objetos en nuestro programa. Se busca establecer una relación entre los elementos del mundo real con los objetos de nuestro programa.

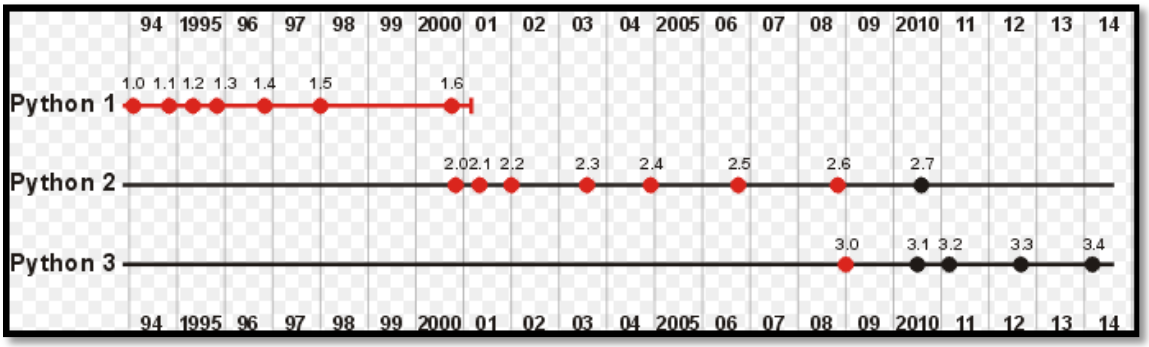
Otra de las características más interesantes de Python es que es un lenguaje con tipado dinámico, es decir, no es necesario declarar el tipo de dato que va a contener una determinada variable, sino que su tipo se determinará en tiempo de ejecución según el tipo del valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo.
De esto se deriva que en Python no se permita tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo previamente. Esta característica hace que se hable de Python como un lenguaje fuertemente tipado.

Otra ventaja que posee es la existencia de funciones incorporadas en el propio lenguaje, para el tratamiento de strings, números, archivos, etc. Además, existen muchas librerías que podemos importar en los programas para tratar temas específicos como la programación de ventanas o sistemas en red o cosas tan interesantes como crear archivos comprimidos en .zip. Esto es resultado de una comunidad de desarrolladores que se preocupan por ir añadiendo funciones y utilidades constantemente a Python.

Por último, destacar que Python tiene una sintaxis muy visual, gracias a una notación identada de obligado cumplimiento. En muchos lenguajes, para separar porciones de código, se utilizan elementos como las llaves o las palabras clave. Para separar las porciones de código en Python se debe tabular hacia dentro, colocando un margen al código que iría dentro de una función o un bucle. Esto ayuda a que todos los programadores adopten unas mismas notaciones y que los programas de cualquier persona tengan un aspecto muy similar.

Todas estas características hacen de Python un lenguaje muy asequible a la hora de empezar desde cero, que es la situación en la que se empezó el proyecto, con cero conocimientos ni de Python, ni de *Raspbian*[10], sistema operativo con el que trabaja la Raspberry.
Además, el utilizar Python nos permitía la posibilidad de desarrollar una interfaz gráfica amigable, con todas las opciones necesarias para la utilización del software *minimu9-ahrs* y de la gestión de sus datos. Para desarrollar esta interfaz, se ha utilizado Tkinter, una herramienta muy útil, pero que solo funciona a día de hoy con las versiones 2.X de Python, de ahí, que se decidiese usar para el proyecto la versión 2.7.10 de Python, la última de la versión 2.

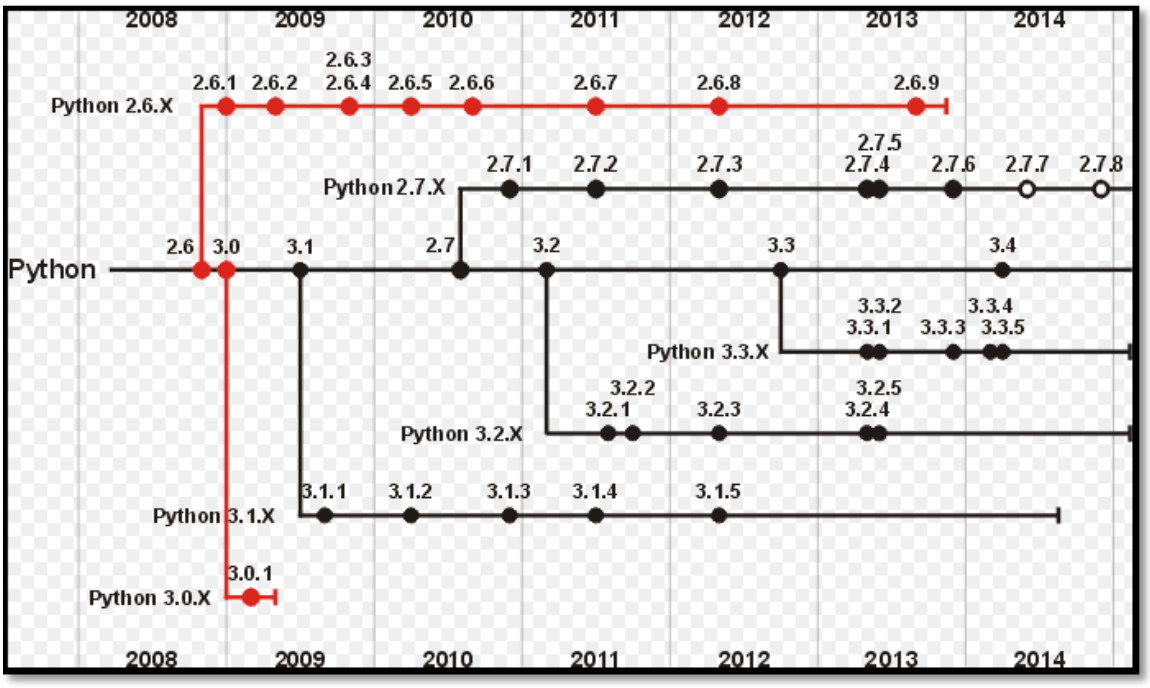
Se puede observar, en esta pequeña cronología las versiones mayores que han existido en la actualidad de Python.



NOTA: Las que están en color rojo se consideran obsoletas

En esta otra imagen, se puede apreciar, la división de Python en sus versiones 2.X y 3.X y todas las diferentes actualizaciones que se han ido realizando:

[10] Raspbian: SO basado en Debian, pero optimizado para la Raspberry. Debian es una distribución de Linux.


2008
2009
2010
2011
2012
2013
2014
Python 2.6.X
2.6.1
2.6.2
2.6.3
2.6.4
2.6.5
2.6.6
2.6.7
2.6.8
2.6.9
Python 2.7.X
2.7.1
2.7.2
2.7.3
2.7.4
2.7.5
2.7.6
2.7.7
2.7.8
Python
2.6
3.0
3.1
2.7
3.2
3.3
3.4
Python 3.3.X
3.3.1
3.3.2
3.3.3
3.3.4
3.3.5
Python 3.2.X
3.2.1
3.2.2
3.2.3
3.2.4
3.2.5
Python 3.1.X
3.1.1
3.1.2
3.1.3
3.1.4
3.1.5
Python 3.0.X
3.0.1
2008
2009
2010
2011
2012
2013
2014

1.3. Tkinter: Un diseñador de interfaces para Python

Tkinter es un *binding* de Python del toolkit *Tk GUI*. Es la interfaz estándar de Python para el toolkit *Tk GUI*, lo que lo hace un estándar Python *de facto*. Está incluido en el instalador por defecto de Windows y Mac OS y por supuesto Linux. El nombre de Tkinter viene de interfaz Tk, el cual fue escrito por Fredrik Lundh.

Tkinter consta de una serie de módulos. La interfaz de Tk es proporcionada por un módulo de extensión binaria llamado tkinter. Este módulo contiene la interfaz de bajo nivel Tk, y nunca debe ser utilizado directamente por los programadores de aplicaciones. Suele ser una biblioteca compartida (o DLL), pero podría en algunos casos ser estáticamente ligada con el intérprete de Python.

Las llamadas a Tkinter se traducen en comandos Tcl que alimentan al intérprete embebido, permitiendo mezclar Python y Tcl en una sola aplicación. La interfaz pública se proporciona a través de una serie de módulos de Python. El módulo de interfaz más importante es el propio módulo Tkinter. Para utilizar Tkinter, todo lo que necesitas hacer es importar el módulo Tkinter:

```
import Tkinter
```

O, más a menudo:

```
import Tkinter as Tk
```

El módulo Tkinter sólo exporta las clases llamadas widget y sus constantes asociadas.

Cuando se decidió optar por Tkinter para desarrollar el software del proyecto para gestionar la IMU, se preveyó que el script se ejecutase desde el *shell* de Python y se lanzasen todas las funcionalidades implementadas, pero no ha podido ser así. El problema es que Tkinter y la versión 2.X tienen limitaciones en cuanto a la ejecución desde el *shell*, de modo que para que funcione correctamente el script programado se debe de lanzar el programa desde la ventana de comandos del sistema operativo para que de este modo las funcionalidades implementadas funcionen correctamente.

1.4. Raspberry Pi: De herramienta de aprendizaje a piedra angular del mundo Geek

Es interesante el camino recorrido por el proyecto de Raspberry Pi, el cual fue ideado en 2006 pero no fue lanzado al mercado hasta febrero de 2012. Ha sido desarrollado en la Universidad de Cambridge y su misión es fomentar la enseñanza de las ciencias de la computación de los niños. De hecho en enero de 2013 Google donó más de 15000 unidades de Raspberry Pi para colegios en Reino Unido. En mayo de 2009, la Fundación Raspberry Pi [11]fue fundada en Caldecote, South Cambridgeshire, Reino Unido como asociación caritativa.
La Raspberry Pi, ha sido diseñada, no sólo con la intención de ser lo más barata posible y llegar al mayor número de usuarios, sino también pretendiendo facilitar el aprendizaje de la programación, su lenguaje, o incluso concebir pequeños proyectos de hardware, como el que nos ocupa en estas líneas.

Ahora bien, ¿qué especificaciones tiene este pequeño ordenador que lo hace tan interesante? En la actualidad existen diferentes versiones de este miniordenador, por lo que la mejor forma de ver sus especificaciones es con una pequeña tabla extraída de Wikipedia [12]y que ha sido contrastada con la página oficial de la Fundación Raspberry Pi.

	Modelo A	Modelo B	Modelo B+
SoC:[5]	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + puerto USB)[3]		
CPU:	ARM 1176JZF-S a 700 MHz (familia ARM11)[3]		
Juego de instrucciones:	RISC de 32 bits		
GPU:	Broadcom VideoCore IV,,[60] OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia),[58] 1080p30 H.264/MPEG-4 AVC[3]		
Memoria (SDRAM):	256 MiB (compartidos con la GPU)	512 MiB (compartidos con la GPU)[4] desde el 15 de octubre de 2012	
Puertos USB 2.0:[54]	1	2 (vía hub USB integrado)[53]	4
Entradas de vídeo:[61]	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la RPF		
Salidas de vídeo:[5]	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4),[62] Interfaz DSI para panel LCD[63] [64]		
Salidas de audio:[5]	Conector de 3.5 mm, HDMI		
Almacenamiento integrado:	SD / MMC / ranura para SDIO		MicroSD
Conectividad de red:[5]	Ninguna	10/100 Ethernet (RJ-45) vía hub USB[53]	
Periféricos de bajo nivel:	8 x GPIO, SPI, I²C, UART[60]		
Reloj en tiempo real:[5]	Ninguno		
Consumo energético:	500 mA, (2.5 W)[5]	700 mA, (3.5 W)	600 mA, (3.0 W)
Fuente de alimentación:[5]	5 V vía Micro USB o GPIO header		
Dimensiones:	85.60mm × 53.98mm[65] (3.370 × 2.125 inch)		
Sistemas operativos soportados:	GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux. RISC OS[2]		

[11] Fundación Raspberry Pi https://www.raspberrypi.org
[12] Wikipedia: https://es.wikipedia.org

1.4.1. Especificaciones técnicas

Actualmente existen diferentes de Raspberry Pi. Estos son: Modelo A, Modelo B y Modelo B +.

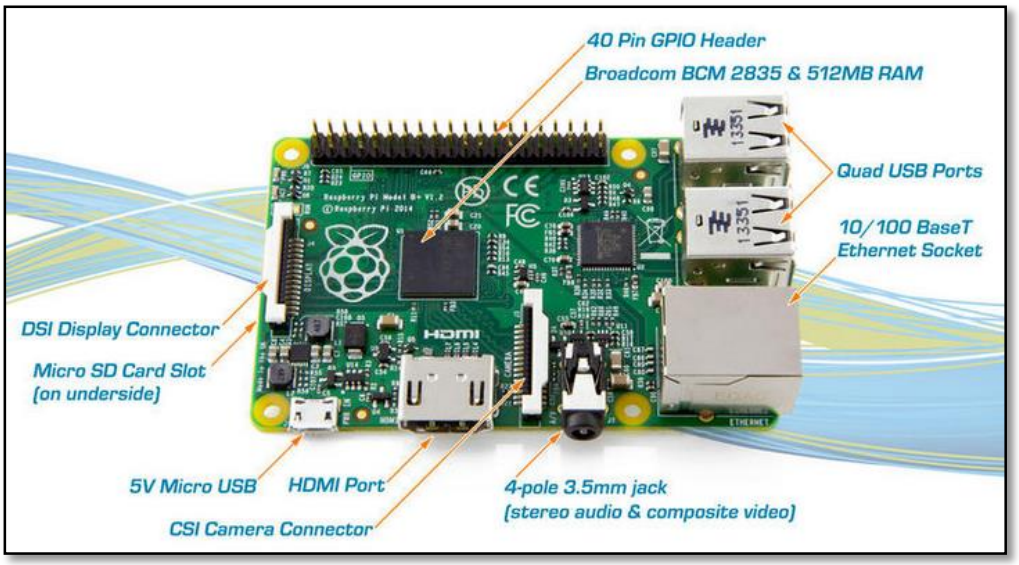
- **SoC**

Broadcom BCM2835 (cpu+gpu+dsp+sdram+puerto usb)

El BCM2835 es un procesador de aplicaciones multimedia de bajo coste, full HD. Optimizado para móviles avanzados y sistemas integrados que requieren altos niveles de rendimiento multimedia. Diseñado y optimizado para una buena eficiencia energética. El BCM2835 es un procesador system-on-chip (SoC), esto quiere decir que la mayor parte de los componentes del sistema, incluidos la CPU y la GPU junto con el audio y el hardware de comunicaciones, se encuentran integrados dentro del chip de la memoria de 512Mb en el centro de la placa.

Broadcom BCM2836

Este nuevo procesador conserva todas las características del BCM2835, pero reemplaza el único ARM11 700MHz con un quad-core complejo ARM Cortex -A7 900MHz: todo lo demás se mantiene igual, por lo que no hay una transición dolorosa o disminución de la estabilidad.



- **CPU**

ARM1176JZF a 700MHz (familia ARM11)

La CPU funciona a 700MHz y es capaz de soportar overclock a 1GHz. La CPU está basada en la versión 6 de la arquitectura ARM, la cual no es soportada por una gran cantidad de distribuciones Linux, por ejemplo Ubuntu, de ahí la importancia de iniciativas dentro del software libre como Raspbian, que es una derivación de *Debian* especialmente desarrollado y adaptado para la Raspberry.

- **GPU**

Broadcom VideoCore IV, OpenGL ES 2.0, MPEG-2 y VC-1

La GPU es capaz de mover contenidos con calidad Blu-ray. Dispone de un núcleo 3D con soporte para las librerías arriba mencionadas. Es capaz de decodificar 1080p30.

- **SDRAM**

Único módulo el cual funciona a 400MHz en su modo normal y alcanzando los 600MHz en su versión "TURBO".

- **Modelo A**

Este modelo dispone de 256Mb compartidos con la GPU.

- **Modelo B y B+**

Estos modelos disponen de 512Mb compartidos con la GPU.

- **Raspberry Pi 2 Modelo B**

Disponen de 1Gb de memoria RAM.

- **Almacenamiento**

La Raspberry Pi no dispone de un disco duro tradicional, sino que dispone de un lector para memorias SD, un sistema de almacenamiento integrado en estado sólido. El arranque del sistema se hará desde la propia tarjeta SD. Están disponibles Tarjetas SD con el sistema operativo precargado en la tienda oficial de la Raspberry Pi.

- **Puertos**

Salidas de audio

Conector de Jack de 3.5mm además del propio HDMI.

Salidas de video

El puerto HDMI de la Raspberry Pi B+ puede desplegar imágenes a la resolución de 1920x1080 Full HD. La salida DSI (Display Serial Interface) es utilizada en los monitores de pantalla plana de las tablets y los smartphones.

Tarjeta de red

Los modelos B y B+ poseen un conector de Ethernet (RJ-45) con una velocidad de 10/100Mbps. No dispone de Wifi pero es posible añadirlo utilizando un adaptador USB para red inalámbrica.

USB

Los puertos USB son 2.0. El modelo B+ posee 4 puertos.

Conector GPIO

Es un pin genérico en un chip, cuyo comportamiento (incluyendo si es un pin de entrada o salida) se puede controlar (programar) por el usuario en tiempo de ejecución. Esta característica es de las más interesantes que tiene la Raspberry y que es la que nos brinda la posibilidad de programar e interactuar de forma sencilla y practica con diferentes dispositivos de hardware.

Alimentación

La placa no dispone de un interruptor de encendido/apagado. La alimentación que necesita es de 5V, que podemos proveer con un conector microUSB estándar. El consumo de la placa es de 700mA, (3,5W).

2. PREPARACIÓN DE LA RASPBERRY MODEL B+

2.1. Instalación de utilidades Python y los puertos I2C

Para poder llevar a término este proyecto, primero se ha de preparar la Raspberry B+ para ello. Dado que la IMU de Pololu MinIMU-9 v2 trabajo con los puertos I2C de los GPIOs de la Raspberry, debemos de habilitar los puertos. La Raspberry tiene dos puertos I2C, el puerto 0 y el puerto 1. Por defecto, el modelo B+ de 512 MB de RAM, que es sobre el que se va a desarrollar el proyecto, trabaja sobre el puerto 1.

Esta tarea no resulta a priori ningún problema, aunque por cuestiones que son desconocidas, el servidor de *Debian*, encargada de desarrollar y mantener la distribución de Raspbian sobre la que estamos trabajando, ha suprimido algunos archivos necesarios para la preparación de la Raspberry, de modo que hemos tenido que buscar los paquetes de instalación, pasarlos al sistema de archivo de Raspbian, y desplegar los paquetes manualmente.

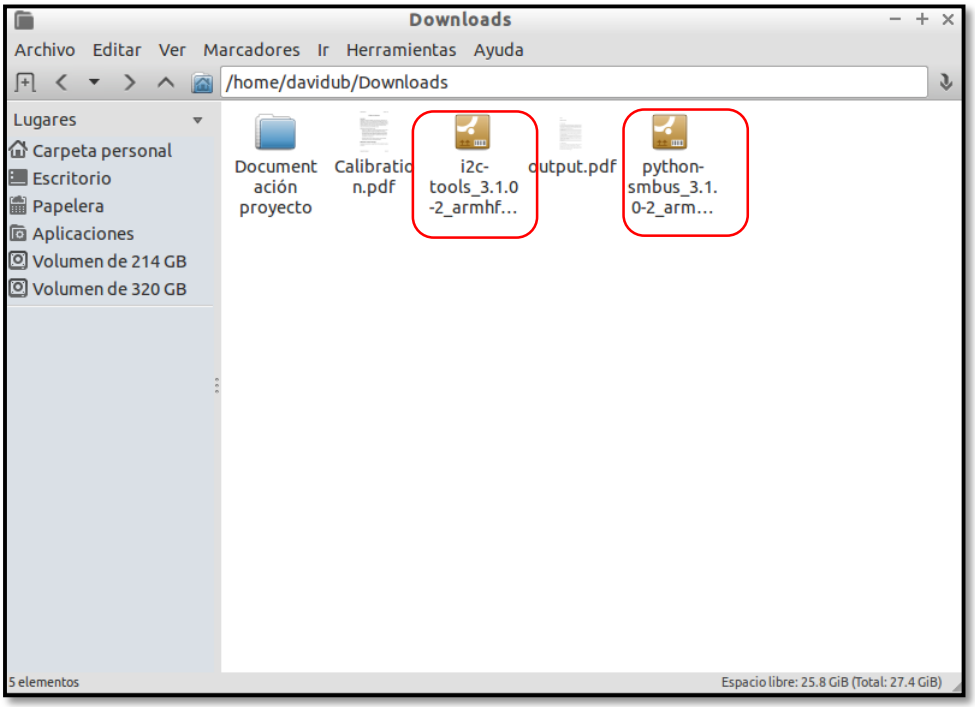
Lo primero que debemos instalar, será los siguientes paquetes.

- python-smbus_3.1.0-2_armhf.deb
- i2c-tools_3.1.0-2_armhf.deb

- En caso de que los servidores de *Debian* funcionasen bastaría con:

```
sudo apt-get install python-smbus
sudo apt-get install i2c-tools
```

- En nuestro caso, en el que no ha sido posible acceder al servidor de *Debian* hemos tenido que descargar primero los paquetes.



Teniendo una conexión SSH configurada para la comunicación entre el Host y la Raspberry, hemos descargado un gestor de transmisión de archivos. Este gestor se llama Filezilla y para instalarlo ejecutamos el comando:

```
sudo apt-get install filezilla
```

Para conectar la Raspberry, tenemos que poner:
IP de la Raspberry: 192.168.1.180
User: pi
Contraseña: raspberry
Puerto por defecto: 22

Una vez realizada la conexión, tendremos en la ventana izquierda el sistema de archivos del Host con el SO OSGeo Live 8 y a la derecha el sistema de archivos de la Raspberry con Raspbian.

Una vez con los paquetes en la Raspberry, ejecutamos los siguientes comandos tras viajar a la carpeta contenedora de los paquetes:

```
sudo dpkg –i python-smbus_3.1.0-2_armhf.deb
sudo dpkg –i i2c-tools_3.1.0-2_armhf.deb
```

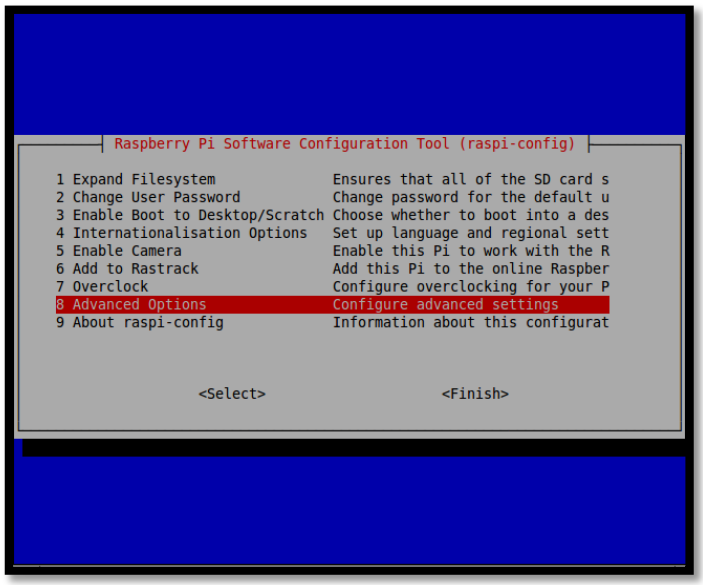
2.2. Configuración del puerto I2C de la Raspberry Model B+

2.2.1. Configuración del kernel: Installing Kernel Support (with Raspi-Config)

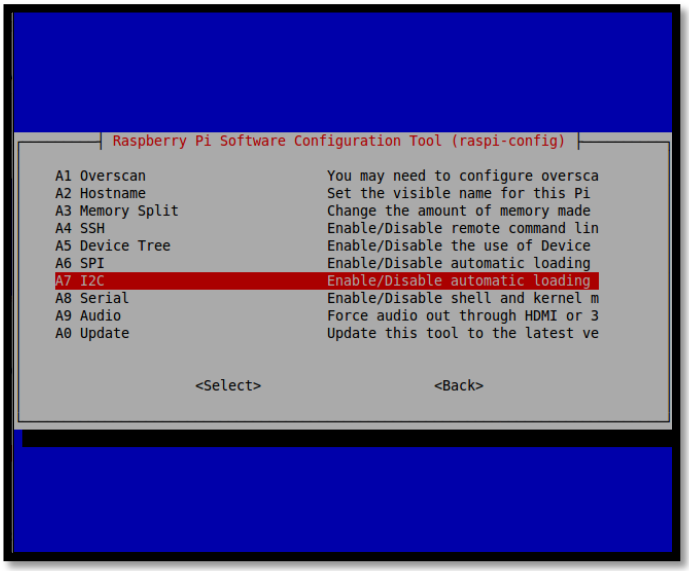
Para la configuración del kernel primero accedemos al menú de configuración de la Raspberry tecleando en el terminal el comando:

```
sudo raspi-config
```

Nos aparece el menú principal de configuración y vamos a *Advanced Options*



Nos desplazamos con las flechas del cursor hasta la opción *A7 I2C*



Se nos preguntará si queremos activar la opción la interfaz *ARM I2C*

Seleccionaremos que sí y una pantalla nos avisará de que se la interfaz ARM I2C se ha activado

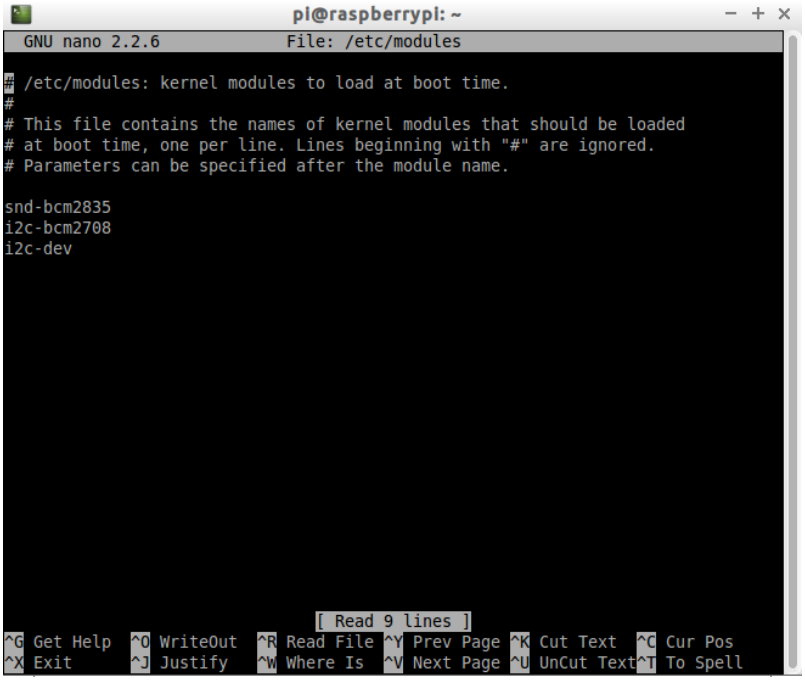
Una vez hecho todo este proceso debemos modificar el fichero modules de la Raspberry para acceder al puerto I2C. Para ello, tecleamos en el terminal:

```
sudo nano /etc/modules
```

En él, incluiremos estas dos líneas al final del archivo:

```
i2c-bcm2708
i2c-dev
```

Quedando el archivo finalmente del siguiente modo:

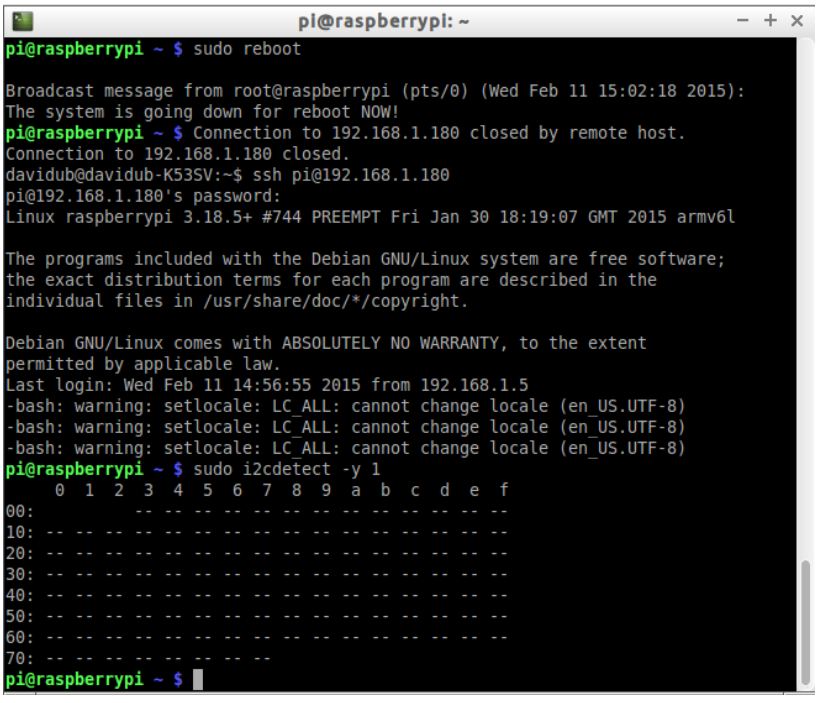


Finalmente, tenemos que reiniciar la Raspberry y testear si el puerto I2C funciona. Para ello, teclearemos el siguiente comando en el terminal de las Raspberry Pi.

```
sudo adduser pi i2c

sudo i2cdetect –y 1
```

Si funciona, debería aparecernos por pantalla una imagen similar a esta:



2.3. Instalación del software minimu9–ahrs

Para la instalación del software minimu9-ahrs buscamos el repositorio de enlaces para poder descargar todos los archivos necesarios desde el terminal.

http://www.davidegrayson.com/minimu9-ahrs/debian/

Para su instalación, navegamos en la página anterior, hasta encontrar el paquete que deseamos instalar, en este caso, queremos obtener el enlace del paquete minimu9-ahrs. Una vez aquí, obtenemos el enlace y ya nos vamos al terminal y tecleamos consola:

```
wget http://www.davidegrayson.com/minimu9-ahrs/debian/
minimu9-ahrs_2.0.0-1_armhf.deb

#Mensaje de confirmación de la carga del paquete de instalación

Grabando a: “minimu9-ahrs_2.0.0-1_armhf.deb”
```

El mensaje anterior nos indica que tenemos el paquete preparado para instalarlo. Seguiremos entonces tecleando en consola:

```
dpkg -i minimu9-ahrs_VERSION_armhf.deb

#Aparecera el siguiente mensaje de error:

dpkg: error al procesar minimu9-ahrs (--install):
 problemas de dependencias - se deja sin configurar
Procesando disparadores para man-db ...
Se encontraron errores al procesar:
 minimu9-ahrs
```

Para solucionar dicho problema ejecutaremos el siguiente comando en consola:

```
sudo apt-get -f install

#Mensaje de confirmación de instalación
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Corrigiendo dependencias... Listo
```

De este modo se corrigen las dependencias y se instalan nuevos paquetes de software necesarios. Los enumeramos a continuación:

Se instalarán los siguientes paquetes extras:

```
-libamd2.2.0 libboost-program-options1.49.0 libexpat1-dev libssl-dev
-libssl-doc libumfpack5.4.0 python-dev python-imaging python-scipy
-python2.7-dev
```

Paquetes sugeridos:

```
-python-imaging-doc python-imaging-dbg
```

Se instalarán los siguientes paquetes NUEVOS:

```
-libamd2.2.0 libboost-program-options1.49.0 libexpat1-dev libssl-dev
-libssl-doc libumfpack5.4.0 python-dev python-imaging python-scipy
-python2.7-dev
```

Hecho esto, queda todo el software necesario instalado y configurado en la Raspberry Pi B+ con el cual ya podremos leer el sensor de Pololu, la MinIMU-9 v2 y calibrarla correctamente. Aunque podremos leerla, queremos poder visualizar los ejes y los giros que realiza el sensor. El siguiente paso será instalar el paquete de software llamado ahrs-visualizer_1.0.1-1_armhf.deb

2.4. Instalación del software ahrs–visualizer

Para la instalación de este paquete teclearemos en terminal las siguientes líneas de comandos:

```
wget http://www.davidegrayson.com/minimu9-ahrs/debian/ahrs-
visualizer_1.0.1-1_armhf.deb

sudo dpkg -i ahrs-visualizer_1.0.1-1_armhf.deb

sudo apt-get -f install
```

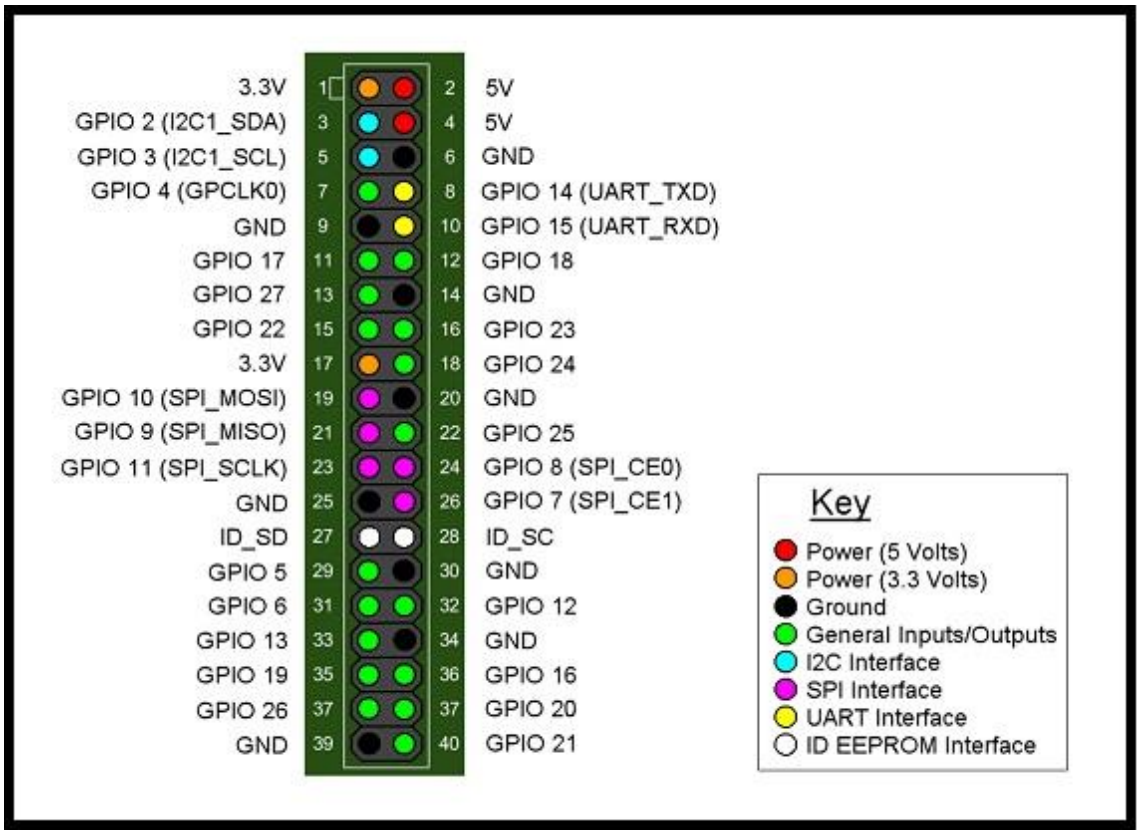
Una vez ejecutados estos comandos ya tendremos el visualizador instalado que utilizaremos posteriormente para poder ver en pantalla todos los movimientos de la MinIMU-9 v2.

3. CONEXIÓN, PUESTA EN MARCHA Y LECTURA DE LA IMU CON LA RASPBERRY PI B+

3.1. Conexión de la IMU con la Raspberry

Una de los motivos por los que se decidió optar por la Raspberry Pi B+ para la realización de este proyecto fue por que en su construcción se incluyó una opción de hardware de enorme potencialidad, como se muestra en este proyecto y es lo que se conoce como las conexiones GPIO[13] o GPIO PIN. Gracias a esta posibilidad de hardware, nos permite conectar multitud de sensores de diferenstes areas de conocimiento y uso. Si queremos hacernos una idea de la gran cantidad de aplicaciones posibles que tenemos con los GPIO PIN, deberiamos visitar las diferentes paginas web especialistas en sensores y aplicaciones para las Raspbery, como por ejemplo, la mas completa, adafruit[14].

Una vez sabemos porque se eligió, vamos a ver que significa y que usos tiene cada uno de los puertos GPIO que tiene nuestra Raspberry. Para ello, nos apoyaremos en la siguiente imagen.



Podemos observar, que la Raspberry tiene dos posibles voltajes de salida, 3.3 voltios y 5 voltios, esto significa, que dependiendo de las especificaciones del sensor que vayamos a utilizar deberemos alimentarlo desde un GPIO PIN en concreto. Auque en nuestro caso no ha sido necesario dado que el sensor MinIMU-V2 admite 3.3 voltios, habrá ocasiones en las que

[13] GPIO: *General Purpose Input/Output* (Entrada/Salida de Propósito General)
[14] Adafruit: https://www.adafruit.com/

tenderemos que ayudarnos de resistencias para no quemar o dañar el sensor debido a uin voltaje de entrada incorrecto.
Viendo los voltajes de salida, podemos sospechar del mismo modo, que voltajes de entrada puede soportar como máximo la Raspberry y que deberemos ser cuidadosos de estudiar para no echar a perder toda la Raspberry.

Otro de los GPIO PIN que podemos ver, son los llamados GPIO, que estos son los pins de conexión y comunicación estándadr que tiene la Raspberry, y que serán los que se utilizarán siempre, a nos ser que el sensor con el que queremos trabajar requiera del puerto I2C para comunicarse correctamente con la Raspberry.

Finalemente tenemos el grupo de pins GPIO mas específicos. Un grupo de pins que tienen unas características especiales que permiten, según que sensor, la explotación de ciertas funcionalidades específicas. Entre este grupo de conexiones, tenemos:

- I2C Interface[15]

I²C es un bus de comunicaciones en serie. Su nombre viene de Inter-Integrated Circuit (Inter-Circuitos Integrados). En la actualidad, la versión que se utiliza es la 2.1 que data del año 2000. La velocidad es de transmisión 100 kbit/s en el modo estándar, aunque puede alcanzar velocidades de 3.4 Mbit/s. Es un bus muy usado en la industria, principalmente para comunicar microcontroladores y sus periféricos en sistemas embebidos (Embedded Systems).

La principal característica de I²C es que utiliza dos líneas para transmitir la información: una para los datos y otra para la señal de reloj. También es necesaria una tercera línea, pero esta sólo es la referencia (masa).

Las líneas se llaman:
- SDA: datos (GPIO 2)
- SCL: reloj (GPIO 3)
- GND: tierra (GPIO 6)

Los dispositivos conectados al bus I²C tienen una dirección única para cada uno. También pueden ser maestros o esclavos. El dispositivo maestro inicia la transferencia de datos y además genera la señal de reloj, pero no es necesario que el maestro sea siempre el mismo dispositivo, esta característica se la pueden ir pasando los dispositivos que tengan esa capacidad. Esta característica hace que al bus I²C se le denomine bus multimaestro.

El código del kernel de Linux para el soporte I2C está separado en varias piezas lógicas:

- I2C chip driver (maneja uno de los chips conectados al bus I2C, tanto si se comporta como maestro o como esclavo)
- I2C bus driver
- I2C algorithm driver
- I2C core (la parte genérica del subsistema de I2C)

Este será el bus de comunicaciones que utilizaremos para conectar nuestra Raspberry con nuetro sensor. Mas adelante veremos las conexiones que hemos realizado con nuestro sensor.

[15] Especificaciones I2C: http://www.i2c-bus.org/

- SPI Interface[16]

El Bus SPI (Serial Peripheral Interface) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj (comunicación sincrónica).

- UART Interface[17]

UART, son las siglas en inglés de *Universal Asynchronous Receiver-Transmitter*, en español: Transmisor-Receptor Asíncrono Universal, es el dispositivo que controla los puertos y dispositivos serie. Se encuentra integrado en la placa base o en la tarjeta adaptadora del dispositivo.

El UART normalmente no genera directamente o recibe las señales externas entre los diferentes módulos del equipo. Usualmente se usan dispositivos de interfaz separados para convertir las señales de nivel lógico del UART hacia y desde los niveles de señalización externos.

- ID EEPROM Interface

Puerto que se añadió en las últimas Raspberry que salieron al mercado, las B+ y que tiene como principal funcionalidad la de potenciar, mediante la introducción de chips EEPROM, la memoria y velocidad de proceso de la Raspberry. Esta opción es algo de lo que todavia no existe mucha información ni fabricantes que construyan chips especificos para la Raspberry, si no que se adaptan chips existentes para este cometido, aunque inicialmente no era su uso previsto.

- <u>CONECTAMOS LA IMU</u>

Para la conexión del sensor MinIMU-V2 segiremos el esquema de los GPIO PIN que hemos visto anteriormente y nos fijaremos que conexiones tiene el sensor.

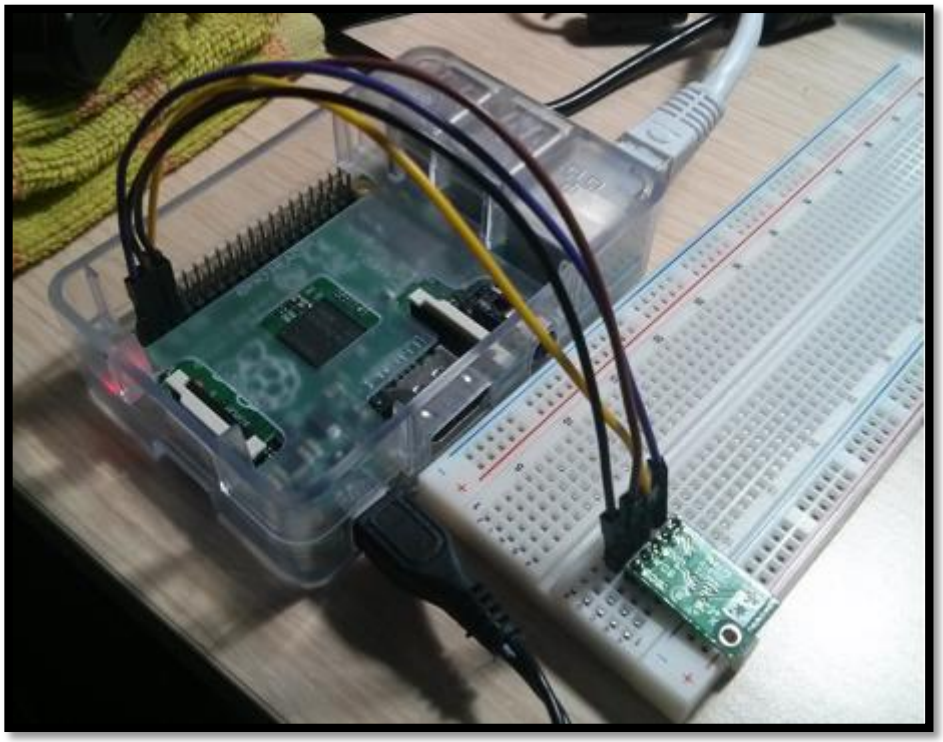
Vemos que en realidad, los pins que nos interesaran serán los que están marcados como SCL, SDA, GND y VDD.

[16] Especificaciones SPI: https://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/README.md
[17] Especificaciones UART: http://whatis.techtarget.com/definition/UART-Universal-Asynchronous-Receiver-Transmitter

Lo resumimos en el siguiente cuadro en el que nos indica la correspondencia sensor-pins que conectaremos con wire jumpers para hacer la conexión en una protoboard que utilizaremos:

Raspberry Pi pin	MinIMU-9 pin
GND	GND
3V3 Power	VDD
GPIO 0 (SDA)	SDA
GPIO 1 (SCL)	SCL

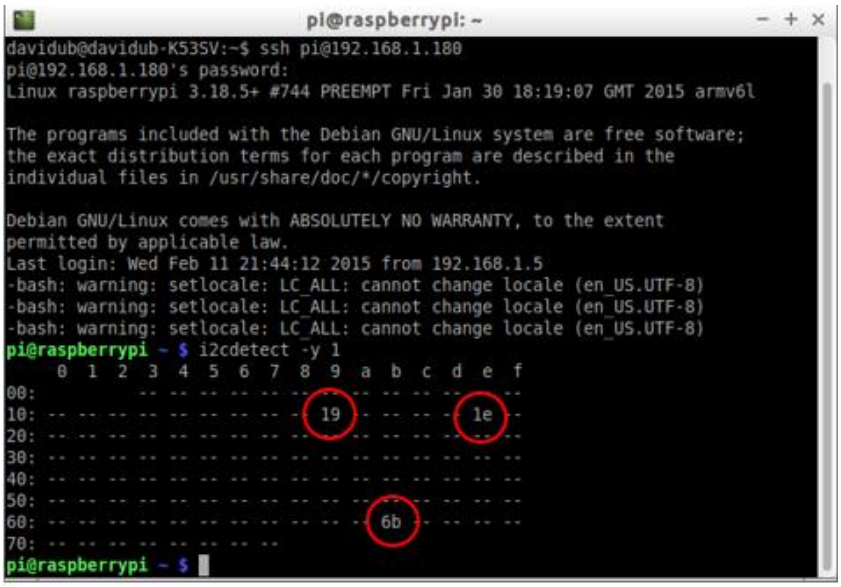
Finalmente quedan las conexiones de este modo:



Para comprobar que las conexiones que hemos realizado están correctas y que el sensor IMU funciona y es capaz de comunicarse con la Raspberry por el puerto 1, que es el que viene definido por defecto con el kernel de la Raspberry B+, tecleamos el siguiente comando:

```
sudo i2cdetect -y 1
```

Si todo es correcto, nos debería mostrar en pantalla algo parecido a esto:



Si por el contrario, no apareciese el resultado anterior podría ser, principalmente, por tres motivos:

1) El primero y más obvio, que las conexiones que se han especificado anteriormente, no se hubiesen hecho correctamente, por lo que lo primero sería revisar que todo se encuentra bien conectado y en buen estado.

2) El segundo, que por algún motivo (por ejemplo, que hubiésemos instalado una distribución de *Raspbian* por actualizar), que el puerto I2C que tiene la Raspberry por defecto sea el 0 y no el 1 como viene siendo habitual en la Raspberry B+, por lo que en ese caso deberíamos de teclear el comando:

```
sudo i2cdetect -y 0
```

3) El tercero, podría ser que después de todo el proceso descrito anteriormente para habilitar y configurar el puerto I2C, no haya sido suficiente y aún exista algún tipo de software que impida que los puertos I2C se lancen correctamente. Si es así, tecleamos el siguiente comando que instalará una librería que asegurará que nos dejará trabajar en directorio */dev*, lugar donde el software asociado a la IMU almacenará los archivos de escritura que se generen del funcionamiento del mismo.
   Y posteriormente, de nuevo tecleamos el comando anterior para testear que se reconoce la IMU.

```
sudo apt-get install libi2c-dev

sudo i2cdetect -y 1
```

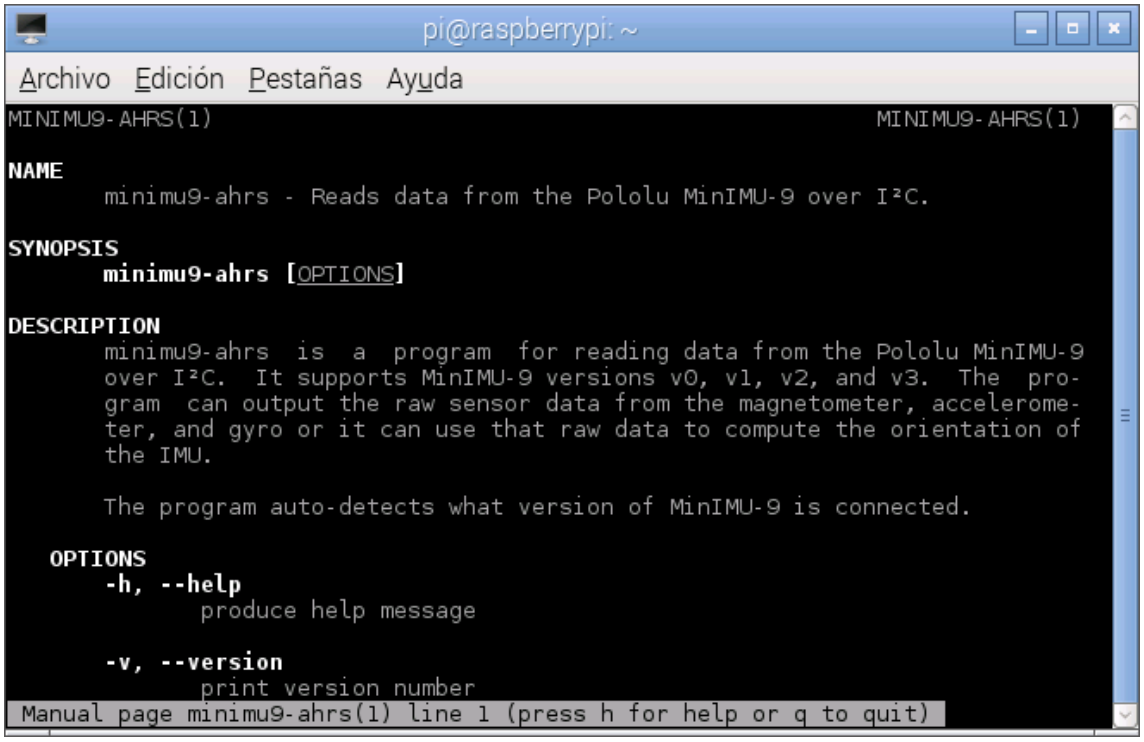
3.2. Puesta en marcha del sensor IMU

Una vez que está todo preparado y en funcionamiento, nos encontramos en disposición de poner en marcha la IMU, para ello, tecleamos el siguiente comando para asegurarnos de que instrucciones son las adecuadas.

```
man minimu9-ahrs
```

Este comando abre el manual del software *minimu9-ahrs* en el que podremos estudiar detalladamente todos los modos que tenemos de lectura de la IMU y las opciones que nos brinda.

Nos deberá aparecer en pantalla algo como esto:



De forma extendida, lo que aparece en la pantalla es el manual que mostramos ahora y que iremos explicando paso a paso según las opciones que vayamos seleccionando.

```
MINIMU9-AHRS(1)                                    MINIMU9-AHRS(1)
NAME
       minimu9-ahrs - Reads data from the Pololu MinIMU-9 over I²C.

SYNOPSIS
       minimu9-ahrs [OPTIONS]

DESCRIPTION
       minimu9-ahrs  is  a  program  for reading data from the Pololu MinIMU-9
       over I²C.  It supports MinIMU-9 versions v0, v1, v2, and v3.  The  pro-
       gram  can output the raw sensor data from the magnetometer, accelerome-
       ter, and gyro or it can use that raw data to compute the orientation of
       the IMU.

       The program auto-detects what version of MinIMU-9 is connected.

OPTIONS
       -h, --help
              produce help message

       -v, --version
              print version number

       --mode MODE
              Specify what algorithm to use.
              normal (default): Fuse compass and gyro.
              gyro-only:  Use only gyro (drifts).
              compass-only:  Use only compass (noisy).
              raw: Just print raw values from sensors.

       --output FORMAT
              Specify  how to output the orientation.  Has no effect if --mode
              raw is specified.
              matrix (default): Direction Cosine Matrix.
              quaternion: Quaternion.
              euler: Euler angles (yaw, pitch roll).
       -b, --i2c-bus
              I²C-bus the IMU is connected to (default: /dev/i2c-0)

COORDINATE SYSTEMS
       There are two coordinate systems in use.

       The ground coordinate system uses the following axes,  in  this  order:
       North, East, and Down.

       The  body  coordinate system represents the orientation of the IMU, and
       uses the following axes, as labeled on the IMU: X, Y, and Z.
```

```
   RAW OUTPUT FORMAT
       If --mode raw is specified, the format of the output will be nine inte-
       gers:

              MX MY MZ  AX AY AZ  GX GY GZ

       where  MX  MY  MZ is the raw reading from the magnetometer, AX AY AZ is
       the raw reading from the accelerometer, and GX GY GZ is the raw reading
       from the magnetometer.

ORIENTATION OUTPUT FORMATS
       Unless --mode raw is specified, the output format will be of the form:

              ORIENTATION AX AY AZ BX BY BZ

       where the format of ORIENTATION is determined by the --output argument,
       AX AY AZ is the scaled acceleration vector in units of 1 g, and  BX  BY
       BZ is the scaled magnetic field vector, whose magnitude should normally
       be close to 1.

   ORIENTATION AS A MATRIX
       The default output format is --output matrix.  In  this  mode,  a  3x3
       transformation  matrix  (also  known as the direction cosine matrix) is
       sent to the standard output.  The matrix is sent  in  row-major  order;
       the first three numbers on the line are row 0, etc.

       The matrix is defined as the matrix that transforms column vectors from
       the body coordinate system to the ground  coordinate  system.  Accord-
       ingly,  its  nine elements can be specified as dot products of the unit
       vectors representing the two coordinate systems:

              north·x  north·y  north·z
              east·x   east·y   east·z
              down·x   down·y   down·z

   ORIENTATION AS A QUATERNION
       If --output quaternion is specified, the orientation is formatted as  a
       unit  quaternion  (four  numbers).  It is a left quaternion that trans-
       forms vectors from the body coordinate system to the ground  coordinate
       system.
```

```
ORIENTATION AS EULER ANGLES
       If  --output  euler  is specified, the orintation is formatted as three
       Euler angles in the following order: pitch, yaw, and roll.   These  are
       the  standard  angles used to represent the orientation of an airplane.
       The angles are output in degrees.

       The Euler angles represent the amount of rotation needed in three  dis-
       tinct  steps to turn the ground coordinate system into the body coordi-
       nate system.

       Imagine you start with a coordinate system  that  is  equal  to  ground
       coordinate system: north, east, down.

       Now  rotate  the coordinate system about its third axis (still pointing
       down) by a certain angle that we will call the yaw.  When  viewed  from
       above, the yaw rotation goes in the clockwise direction.

       Now  rotate  the  coordinate  system about its second axis by a certain
       angle that we will call the pitch.  A positive pitch  makes  the  first
       axis  of  the  coordinate system go up.  A pitch of 90.0 would make the
       first axis point straight up, while a pitch of  -90.0  would  make  the
       first axis point straight down.

       Now rotate the coordinate system about is first axis by a certain angle
       that we will call the roll.

       The coordinate system you now have is equal to the body coordinate sys-
       tem (x, y, z).

FILES
       ~/.minimu9-ahrs-cal
              Calibration file for the magnetomer, needed unless --mode raw is
              specified.  This file should be a one-line file with 6  integers
              separated by spaces: minimum x, maximum x, minimum y, maximum y,
              minimum z, maximum z.  These numbers specify the linear  mapping
              from the raw ellipsoid to the unit sphere. For example, if "min-
              imum x" is -414, it means that a magnetometer reading of -414 on
              the X axis will get mapped to -1.0 when the readings are scaled.
              Run minimu9-ahrs-calibrate to generate this file.

AUTHOR
       David Grayson <davidegrayson@gmail.com>
       http://www.github.com/DavidEGrayson/

SEE ALSO
       minimu9-ahrs-calibrate(1), minimu9-ahrs-calibrator(1)
```

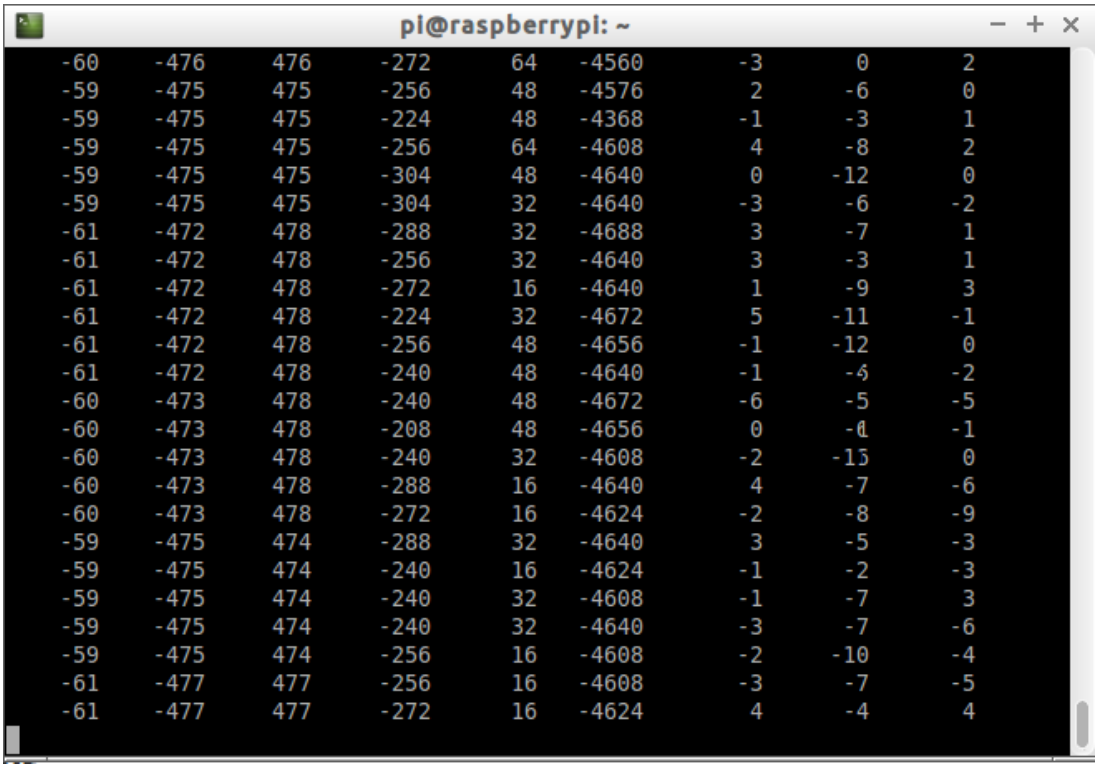
Para hacer una primera lectura (la IMU no está aún calibrada) vamos a hacer una lectura en crudo, es decir, los datos que nos dará serán valores sin mucho sentido, pero de este modo sabremos si está funcionando la IMU y todo el trabajo que se ha hecho ha sido correcto.
Como hemos visto en el manual del software *minimu9-ahrs* tenemos diferentes modos de ejecución, estos son:

- *normal (default): Fuse compass and gyro.*
  Nos muestra lecturas tanto del giróscopo como del magnetómetro.
- *gyro-only: Use only gyro (drifts).*
  Nos muestra solo las lecturas tanto del giróscopo.
- *compass-only: Use only compass (noisy).*
  Nos muestra solo las lecturas tanto del magnetómetro.
- *raw: Just print raw values from sensors.*
  Muestra los datos en crudo del sensor.

Este será el modo que utilizaremos para hacer nuestro primer inicio del sensor y la primera lectura del mismo.
También se debe especificar, como se ha dicho anteriormente, que puerto I2C utilizaremos. En el manual se nos indica que para determinar el puerto debemos seguir la sintaxis *-b, --i2c-bus* siendo por defecto *-b /dev/i2c-0.*
Por tanto, el comando que introduciremos en consola y ejecutaremos será el siguiente:

```
minimu9-ahrs --mode raw –b /dev/i2c-1
```

Y obtenemos la primera lectura de la IMU



Las nueve columnas de valores que nos da de salida son tres vectores. De derecha a izquierda, tenemos la lectura en crudo del magnetómetro, la lectura en crudo del acelerómetro y la lectura en crudo del giroscopio. Estos vectores son tres números enteros en orden X-Y-Z.

3.3. Calibración del sensor IMU

Una vez que estamos obteniendo datos de la IMU deberemos proceder a la calibración del sensor. El magnetómetro necesita ser calibrado para generar un mapeo del elipsoide de las lecturas que toma en modo *raw*, y las escale dentro de la esfera unidad. El software de calibración minimu9-ahrs asume que la forma de las lecturas en modo raw tienen una forma de elipsoide desplazado y deformado a lo largo de los ejes X, Y y Z. Es por eso que lo que se pretende con la calibración es ajustar este elipsoide a la esfera unidad, haciendo coincidir sus centros.
Cuando ejecutamos el software de calibración minimu9-ahrs-calibrate genera un archivo oculto llamado *~/.minimu9-ahrs-cal* en el tenemos una línea con seis numero enteros separados por espacios. Estos valores significan:

- X mínima
- X máxima
- Y mínima
- Y máxima
- Z mínima
- Z máxima

Estos enteros especifican los valores que alcanza el elipsoide dentro de la esfera unidad. Esto quiere decir, que si la X mínima tiene un valor de -317, es porque el magnetómetro ha leído un valor de -317 en el eje X del elipsoide, pero cuando el magnetómetro haya sido calibrado y los valores se hayan escalado, la lectura del magnetómetro será -1.

Para lanzar el software de calibración, deberemos escribir el siguiente comando:

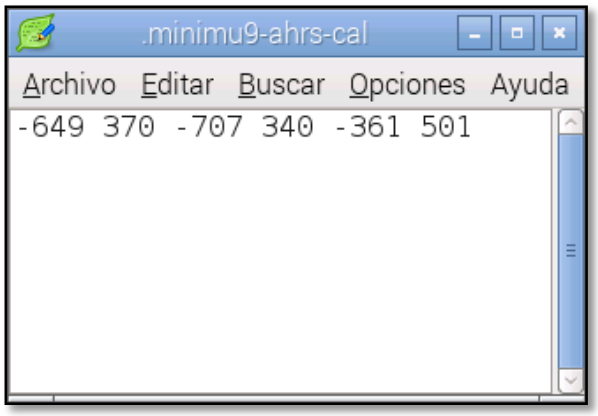
```
minimu9-ahrs-calibrate –b /dev/i2c-1
```

Inmediatamente el software nos mostrará un mensaje en el que nos pedirá que movamos el sensor:

```
pi@raspberrypi ~ $ minimu9-ahrs-calibrate -b /dev/i2c-1
To calibrate the magnetometer, start rotating the IMU through
as many different orientations as possible.
Reading data... done.
Optimizing calibration...
```

Al tiempo que movemos el sensor durante un par de minutos nos informa que el software está leyendo datos. Cuando finalice de leer datos pasara a optimizar la calibración. En este tiempo es aconsejable dejar la IMU junto con la Raspberry en un lugar lo más estable posible, de modo que no se le puedan dar golpes o pueda sufrir movimientos bruscos que pueda alterar la calibración. Una vez que haya finalizado el proceso, nos avisara que la calibración ha concluido mostrándonos *Optimizing calibration... done*.

Para ver que valores ha tomado el magnetómetro durante el proceso de calibración iremos a buscar el archivo *~/.minimu9-ahrs-cal* que se encuentra en el directorio */home/pi*.



A partir de ahora en adelante ya tenderemos el sensor calibrado, por lo que podremos ejecutar el resto de modos que nos permite el sensor. Es recomendable, aunque no necesario, que cuando queramos hacer lecturas del magnetómetro en trabajos posteriores repitamos el proceso de calibración para asegurar unos resultados de calidad.

Estos datos valores que nos devolverá el sensor deberemos de estudiarlos posteriormente para entender su significado y que son exactamente.

4. OBTENCIÓN, COMPRENSIÓN E INTERPRETACIÓN DE LOS DATOS IMU

Actualmente, ya podemos poner en marcha el sensor y empezar a generar archivos de las lecturas del sensor. Para ello, o bien podemos ejecutar los comandos por consola, como hemos visto en el documento *man* del programa, o bien a través del software de gestión de la IMU que en este proyecto se ha desarrollado. Pero para ello, parece lógico explicar que comandos podemos ejecutar por consola gracias al software minimu9-ahrs y que hace cada uno de estos comandos.

- Modos de ejecución:
  Con este parámetros que especificaremos poniendo en la línea de comando *–mode* especificaremos que algoritmo de gestión de los sensores de la IMU que querremos utilizar, y por tanto, que datos estaremos recibiendo.
  - *--mode normal:* Esta es la opción por defecto, es decir, si no especificáramos nada, este sería el modo en el que leería la IMU. En este modo, se combinan las lecturas del magnetómetro y del giroscopio.
  - *--mode gyro-only:* Con esta opción solo obtendremos lecturas del giroscopio[18], de modo que podremos medir los cambios de dirección del sensor.
  - *--mode compass-only:* En este modo, recibiremos lecturas solo del magnetómetro [19]o brújula, de forma que estaremos midiendo las variaciones de ángulo respecto al campo magnético terrestre.
  - *--mode raw:* Este modo de ejecución nos da las lecturas en crudo de todos los sensores. Aunque no se puede interpretar directamente, existen funciones de conversión para convertir estos datos en valores que se puedan manejar fácilmente.

    Si el modo de ejecución es *raw* (datos en crudo) el formato de salida serán nueve números enteros con el formato:

    MX MY MZ AX AY AZ GX GY GZ

    Donde:
    - MX MY MZ es un vector con los datos en crudo del magnetómetro
    - AX AY AZ es un vector con los datos en crudo del acelerómetro
    - GX GY GZ es un vector con las lecturas en crudo del giroscopio

  NOTA: En el modo *raw* no se puede especificar el formato de salida.

[18] Giroscopio: El giróscopo o giroscopio es un dispositivo mecánico que sirve para medir, mantener o cambiar la orientación en el espacio de algún aparato o vehículo

[19] Magnetómetro: Se llaman magnetómetros a los dispositivos que sirven para cuantificar en fuerza o dirección la señal magnética

- Formatos de salida:
  Estos formatos nos servirán para especificar como queremos obtener la orientación del sensor. Salvo en el modo *raw* todos los formatos de salida tienen la siguiente estructura:

  ORIENTATION AX AY AZ BX BY BZ

  Donde:
  - ORIENTATION está determinado por el formato de salida que especificamos y que llevará sus unidades asociadas.
  - AX AY AZ será el vector escalado de la aceleración en unidades de 1 g [20]
  - BX BY BZ será el vector escalado del campo magnético. Su magnitud se encontrará normalmente, próximo a 1.

Pasaremos a ver los diferentes formatos que tenemos definidos:

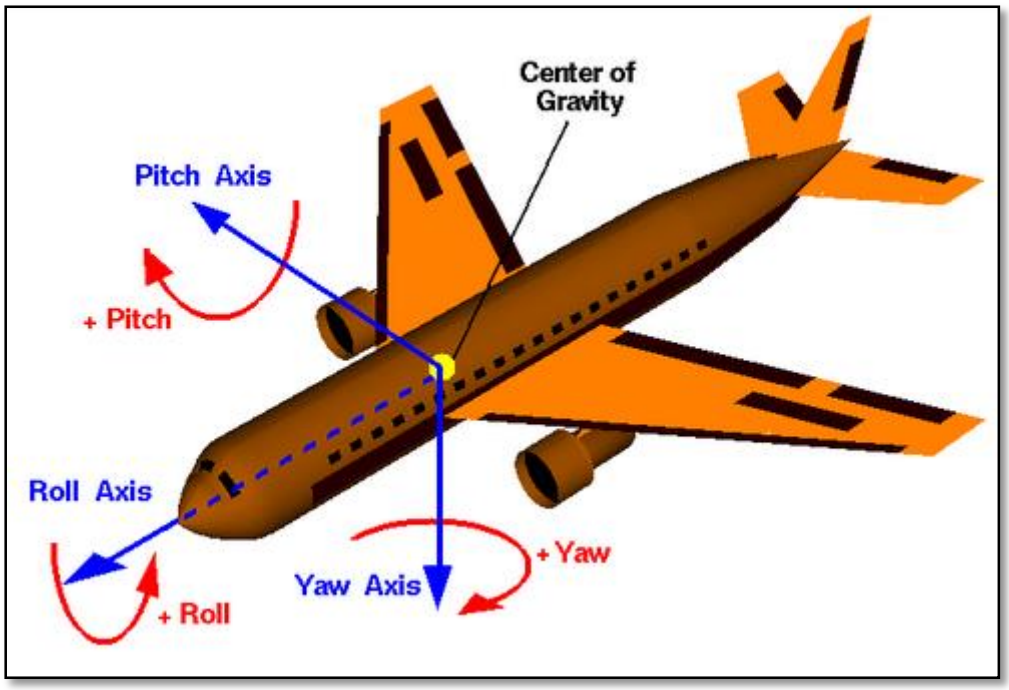
- --*output matrix:* Con este formato de salida, obtenemos una matriz de transformación 3x3 (también conocida como matriz coseno directora). La matriz está ordenada por filas en orden descendente (row-major order). Por tanto, los tres primero números de la línea se corresponden con la fila 0 y así consecutivamente.
  La matriz queda definida como una matriz de transformación de los vectores columna del sistema de coordenadas del sensor (Body Coordinate System), al sistema de coordenadas North-East-Down (Ground Coordinate System).
  Por tanto, estos nueve elementos quedan definidos como el producto escalar del vector representado en los dos sistemas de coordenadas:

North·x	North·y	North·z
East·x	East·y	East·z
Down·x	Down·y	Down·z

- --*output quaternion:* La orientación con este formato de salida da una matriz de transformación del sistema de coordenadas del sensor, al sistema de coordenadas North-East-Down.

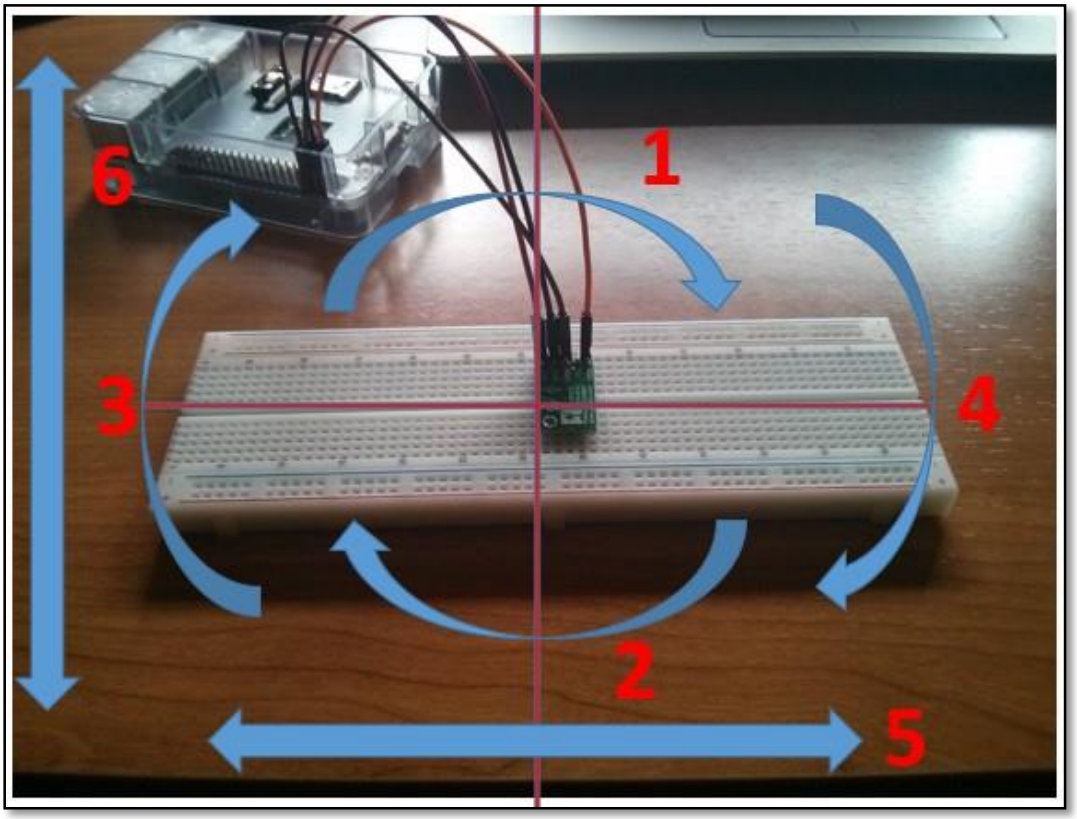
- --*output euler:* con este formato, la orientación viene definida como los tres ángulos de Euler en el orden, de izquierda a derecha, *pitch*, *yaw* y *roll*. Estos ángulos representarían los ángulos de orientación de una aeronave. Los ángulos de Euler vienen dados en grades sexagesimales.
  Los ángulos de Euler representan los diferentes giros necesarios en los tres ejes de rotación para del sistema de referencia North-East-Down al sistema de referencia de la IMU.

[20] 1 g: valor de la gravedad que corresponde a 9,8m/s^2



Una vez expuesto y especificado que hace cada una de las opciones que nos brinda el sensor minimu9-ahrs, vamos a proceder a insertar en consola cada uno de los comando, imprimir un listado, y ver si los datos que nos devuelven son correctos.

Para que las lecturas sean comparables, en cada ejecución hemos procurado registrar los mismos movimientos del sensor de modo que podamos ver también el comportamiento del sensor frente a los mismos movimiento en diferentes lecturas.



- Modo normal y formato de matriz coseno-directora

El comando que ejecutaremos para obtener estas lecturas será el siguiente:

```
minimu9-ahrs -b /dev/i2c-1 --mode normal --output matrix > normalmatrix.csv
```

Para una mejor comprensión vamos a desglosar el comando anterior:

- *--b /dev/i2c-1:* Con esta sentencia, especificamos el bus I2C con el que trabajará nuestra Raspberry.
- *--mode normal:* especificamos que el sensor trabajará tanto con magnetómetro como giroscopio.
- *--output matrix:* el formato de salida es la matriz coseno-directora.
- *> normalmatrix.csv*: generamos un fichero nuevo, con el nombre normalmatrix.csv donde se escribirán las lecturas.

1.000	-0.013	-0.000	0.013	1.000	-0.001	0.000	0.001	1.000	-0.117	-0.004	-1.113	-0.152	-0.077	0.794
1.000	-0.025	-0.000	0.025	1.000	-0.001	0.000	0.001	1.000	-0.121	0.008	-1.120	-0.152	-0.077	0.794
0.999	-0.038	-0.000	0.038	0.999	-0.002	0.000	0.002	1.000	-0.117	0.008	-1.105	-0.152	-0.077	0.794
0.999	-0.052	-0.000	0.052	0.999	-0.003	0.000	0.003	1.000	-0.117	0.000	-1.117	-0.152	-0.077	0.794
0.998	-0.066	-0.000	0.066	0.998	-0.003	0.001	0.003	1.000	-0.125	0.008	-1.124	-0.152	-0.077	0.794
0.997	-0.081	-0.001	0.081	0.997	-0.004	0.001	0.004	1.000	-0.125	0.004	-1.120	-0.156	-0.081	0.796
0.995	-0.095	-0.001	0.095	0.995	-0.005	0.001	0.005	1.000	-0.121	0.012	-1.117	-0.156	-0.081	0.796
0.994	-0.110	-0.001	0.110	0.994	-0.006	0.001	0.005	1.000	-0.121	0.012	-1.117	-0.156	-0.081	0.796
0.992	-0.127	-0.001	0.127	0.992	-0.007	0.002	0.006	1.000	-0.121	0.004	-1.120	-0.156	-0.081	0.796
0.990	-0.143	-0.001	0.143	0.990	-0.007	0.002	0.007	1.000	-0.121	0.008	-1.124	-0.156	-0.081	0.796
0.987	-0.160	-0.001	0.160	0.987	-0.008	0.002	0.008	1.000	-0.109	0.012	-1.120	-0.156	-0.081	0.796
0.984	-0.178	-0.001	0.178	0.984	-0.009	0.003	0.009	1.000	-0.113	0.008	-1.117	-0.156	-0.081	0.796
0.980	-0.197	-0.001	0.197	0.980	-0.010	0.003	0.010	1.000	-0.109	0.004	-1.109	-0.152	-0.079	0.791
0.977	-0.215	-0.001	0.215	0.976	-0.011	0.004	0.011	1.000	-0.113	0.016	-1.113	-0.152	-0.079	0.791
0.971	-0.237	-0.001	0.237	0.971	-0.012	0.004	0.012	1.000	-0.121	0.008	-1.113	-0.152	-0.079	0.791
0.966	-0.258	-0.001	0.258	0.966	-0.013	0.005	0.013	1.000	-0.117	0.008	-1.120	-0.152	-0.079	0.791
0.960	-0.279	-0.001	0.279	0.960	-0.014	0.005	0.014	1.000	-0.113	0.000	-1.128	-0.152	-0.079	0.791
0.954	-0.301	-0.001	0.301	0.954	-0.016	0.006	0.015	1.000	-0.117	0.004	-1.120	-0.152	-0.079	0.791
0.947	-0.322	-0.001	0.322	0.947	-0.017	0.006	0.016	1.000	-0.121	0.004	-1.117	-0.154	-0.074	0.794
0.939	-0.343	-0.001	0.343	0.939	-0.018	0.007	0.017	1.000	-0.121	0.008	-1.120	-0.154	-0.074	0.794
0.931	-0.365	-0.001	0.365	0.931	-0.019	0.008	0.018	1.000	-0.117	0.016	-1.109	-0.154	-0.074	0.794
0.923	-0.386	-0.000	0.386	0.922	-0.021	0.008	0.019	1.000	-0.187	0.008	-1.105	-0.154	-0.074	0.794
0.913	-0.409	-0.000	0.409	0.912	-0.023	0.009	0.021	1.000	-0.117	0.004	-1.117	-0.154	-0.074	0.794
0.902	-0.432	0.000	0.432	0.901	-0.024	0.010	0.022	1.000	-0.121	0.008	-1.120	-0.154	-0.074	0.794
0.890	-0.457	0.001	0.456	0.889	-0.025	0.011	0.023	1.000	-0.113	0.004	-1.128	-0.154	-0.074	0.794
0.876	-0.482	0.001	0.482	0.876	-0.026	0.012	0.024	1.000	-0.109	0.000	-1.128	-0.152	-0.074	0.789
0.862	-0.506	0.002	0.506	0.862	-0.028	0.012	0.025	1.000	-0.121	0.004	-1.113	-0.152	-0.074	0.789
0.847	-0.532	0.002	0.532	0.847	-0.029	0.014	0.026	1.000	-0.117	0.004	-1.120	-0.152	-0.074	0.789
0.829	-0.560	0.003	0.560	0.828	-0.031	0.015	0.027	1.000	-0.117	0.000	-1.117	-0.152	-0.074	0.789
0.811	-0.585	0.004	0.585	0.811	-0.033	0.016	0.029	0.999	-0.121	0.004	-1.113	-0.152	-0.074	0.789
0.792	-0.610	0.004	0.610	0.792	-0.034	0.017	0.030	0.999	-0.113	0.008	-1.113	-0.152	-0.074	0.789
0.771	-0.637	0.005	0.637	0.770	-0.036	0.019	0.031	0.999	-0.117	0.004	-1.109	-0.152	-0.074	0.789
0.749	-0.662	0.006	0.662	0.749	-0.037	0.020	0.032	0.999	-0.117	0.004	-1.113	-0.150	-0.074	0.787
0.725	-0.689	0.007	0.688	0.724	-0.039	0.021	0.033	0.999	-0.117	0.004	-1.120	-0.150	-0.074	0.787
0.701	-0.713	0.008	0.713	0.700	-0.040	0.023	0.034	0.999	-0.113	0.008	-1.124	-0.150	-0.074	0.787
0.675	-0.737	0.009	0.737	0.675	-0.041	0.025	0.035	0.999	-0.113	0.008	-1.124	-0.150	-0.074	0.787
0.648	-0.761	0.010	0.761	0.648	-0.043	0.026	0.036	0.999	-0.117	0.008	-1.113	-0.150	-0.074	0.787
0.620	-0.784	0.011	0.784	0.619	-0.045	0.028	0.037	0.999	-0.121	0.008	-1.124	-0.150	-0.074	0.787

- Modo normal y formato quaternion

El comando que ejecutaremos para obtener estas lecturas será el siguiente:

```
minimu9-ahrs -b /dev/i2c-1 --mode normal --output quaternion > normalquaternion.csv
```

Para una mejor comprensión vamos a desglosar el comando anterior:

- *--b /dev/i2c-1:* Con esta sentencia, especificamos el bus I2C con el que trabajará nuestra Raspberry.
- *--mode normal:* especificamos que el sensor trabajará tanto con magnetómetro como giroscopio.
- *--output quaternion:* el formato de salida es la matriz quaternaria.
- *> normalquaternion.csv*: generamos un fichero nuevo, con el nombre normalquaternion.csv donde se escribirán las lecturas.

1.000	0.000	0.000	0.000	-0.113	0.000	-1.124	-0.131	-0.129	0.798
1.000	0.001	-0.000	0.012	-0.121	0.000	-1.117	-0.131	-0.129	0.798
1.000	0.001	-0.000	0.022	-0.125	0.004	-1.117	-0.131	-0.129	0.798
0.999	0.002	-0.000	0.034	-0.113	0.004	-1.105	-0.131	-0.129	0.798
0.999	0.002	-0.001	0.045	-0.125	0.000	-1.113	-0.131	-0.129	0.798
0.998	0.003	-0.001	0.057	-0.121	0.000	-1.120	-0.131	-0.129	0.798
0.998	0.004	-0.001	0.070	-0.129	-0.004	-1.109	-0.125	-0.133	0.796
0.997	0.004	-0.001	0.082	-0.125	0.000	-1.117	-0.125	-0.133	0.796
0.995	0.005	-0.001	0.096	-0.113	0.000	-1.124	-0.125	-0.133	0.796
0.994	0.006	-0.002	0.111	-0.117	0.000	-1.128	-0.125	-0.133	0.796
0.992	0.007	-0.002	0.126	-0.121	0.000	-1.128	-0.125	-0.133	0.796
0.990	0.007	-0.002	0.140	-0.113	0.000	-1.113	-0.125	-0.133	0.796
0.988	0.008	-0.002	0.155	-0.117	0.004	-1.113	-0.125	-0.133	0.796
0.985	0.009	-0.003	0.170	-0.113	0.004	-1.109	-0.127	-0.133	0.798
0.983	0.010	-0.003	0.186	-0.105	0.000	-1.109	-0.127	-0.133	0.798
0.979	0.010	-0.003	0.201	-0.117	0.000	-1.120	-0.127	-0.133	0.798
0.976	0.011	-0.004	0.217	-0.125	0.004	-1.109	-0.127	-0.133	0.798
0.972	0.012	-0.004	0.233	-0.125	0.012	-1.120	-0.127	-0.133	0.798
0.968	0.013	-0.004	0.250	-0.121	0.008	-1.128	-0.127	-0.133	0.798
0.964	0.014	-0.005	0.266	-0.121	0.004	-1.120	-0.125	-0.129	0.798
0.959	0.015	-0.005	0.283	-0.121	0.004	-1.113	-0.125	-0.129	0.798
0.951	0.016	-0.005	0.309	-0.121	0.020	-1.109	-0.125	-0.129	0.798
0.945	0.017	-0.006	0.326	-0.133	0.012	-1.120	-0.125	-0.129	0.798
0.939	0.018	-0.006	0.344	-0.121	0.008	-1.120	-0.125	-0.129	0.798
0.932	0.019	-0.006	0.362	-0.125	0.004	-1.117	-0.125	-0.129	0.798
0.925	0.020	-0.007	0.379	-0.113	0.016	-1.109	-0.125	-0.129	0.798
0.917	0.021	-0.007	0.397	-0.117	0.008	-1.113	-0.127	-0.131	0.800
0.910	0.022	-0.007	0.415	-0.113	0.004	-1.113	-0.127	-0.131	0.800
0.901	0.023	-0.008	0.433	-0.117	0.004	-1.113	-0.127	-0.131	0.800
0.892	0.024	-0.008	0.451	-0.125	0.004	-1.105	-0.127	-0.131	0.800
0.883	0.025	-0.008	0.469	-0.121	0.004	-1.113	-0.127	-0.131	0.800
0.873	0.026	-0.008	0.487	-0.121	0.000	-1.124	-0.127	-0.131	0.800
0.863	0.027	-0.009	0.504	-0.121	-0.004	-1.117	-0.131	-0.138	0.800
0.853	0.028	-0.009	0.521	-0.113	-0.004	-1.113	-0.131	-0.138	0.800
0.842	0.028	-0.009	0.538	-0.117	0.004	-1.120	-0.131	-0.138	0.800
0.832	0.029	-0.009	0.555	-0.105	0.008	-1.117	-0.131	-0.138	0.800
0.820	0.030	-0.010	0.571	-0.117	0.008	-1.109	-0.131	-0.138	0.800
0.809	0.031	-0.010	0.587	-0.113	0.004	-1.117	-0.131	-0.138	0.800

- Modo normal y formato ángulos de Euler

El comando que ejecutaremos para obtener estas lecturas será el siguiente:

```
minimu9-ahrs -b /dev/i2c-1 --mode normal --output euler > normaleuler.csv
```

Para una mejor comprensión vamos a desglosar el comando anterior:

- *--b /dev/i2c-1:* Con esta sentencia, especificamos el bus I2C con el que trabajará nuestra Raspberry.
- *--mode normal:* especificamos que el sensor trabajará tanto con magnetómetro como giroscopio.
- *--output euler:* el formato de salida nos devuelve los ángulos de Euler.
- *> normaleuler.csv*: generamos un fichero nuevo, con el nombre normaleuler.csv donde se escribirán las lecturas.

0.000	-0.000	0.000	-0.113	0.004	-1.113	-0.134	-0.117	0.796
1.047	-0.015	0.051	-0.125	0.004	-1.113	-0.134	-0.117	0.796
2.126	-0.038	0.110	-0.125	0.004	-1.113	-0.134	-0.117	0.796
3.235	-0.062	0.176	-0.121	0.008	-1.113	-0.134	-0.117	0.796
4.418	-0.087	0.237	-0.113	0.004	-1.109	-0.131	-0.116	0.800
5.628	-0.109	0.296	-0.117	0.008	-1.109	-0.131	-0.116	0.800
6.917	-0.133	0.366	-0.113	0.000	-1.113	-0.131	-0.116	0.800
8.209	-0.163	0.430	-0.121	0.004	-1.105	-0.131	-0.116	0.800
9.549	-0.190	0.501	-0.113	0.008	-1.105	-0.131	-0.116	0.800
10.919	-0.225	0.563	-0.105	0.012	-1.097	-0.131	-0.116	0.800
12.678	-0.269	0.647	-0.113	0.008	-1.113	-0.134	-0.116	0.798
14.112	-0.322	0.709	-0.117	0.012	-1.120	-0.134	-0.116	0.798
15.607	-0.366	0.786	-0.121	0.008	-1.124	-0.134	-0.116	0.798
17.168	-0.419	0.863	-0.117	0.004	-1.109	-0.134	-0.116	0.798
18.735	-0.471	0.944	-0.129	0.008	-1.117	-0.134	-0.116	0.798
20.372	-0.517	1.014	-0.121	0.004	-1.120	-0.134	-0.116	0.798
22.060	-0.569	1.092	-0.121	0.004	-1.113	-0.134	-0.116	0.798
23.865	-0.630	1.166	-0.125	0.008	-1.120	-0.134	-0.117	0.794
25.642	-0.687	1.247	-0.129	0.004	-1.105	-0.134	-0.117	0.794
27.470	-0.754	1.326	-0.125	0.000	-1.105	-0.134	-0.117	0.794
29.369	-0.823	1.400	-0.121	-0.004	-1.101	-0.134	-0.117	0.794
31.284	-0.904	1.478	-0.121	0.004	-1.105	-0.134	-0.117	0.794
33.213	-0.985	1.552	-0.129	0.008	-1.120	-0.134	-0.117	0.794
35.205	-1.073	1.619	-0.125	0.004	-1.120	-0.134	-0.117	0.794
38.141	-1.215	1.719	-0.125	0.004	-1.117	-0.131	-0.116	0.794
40.223	-1.308	1.765	-0.117	0.008	-1.120	-0.131	-0.116	0.794
42.442	-1.411	1.820	-0.113	0.004	-1.113	-0.131	-0.116	0.794
44.606	-1.508	1.871	-0.113	0.004	-1.124	-0.131	-0.116	0.794
46.773	-1.618	1.930	-0.121	0.008	-1.113	-0.131	-0.116	0.794
48.964	-1.730	1.996	-0.129	0.004	-1.117	-0.131	-0.116	0.794
51.197	-1.844	2.049	-0.121	0.004	-1.113	-0.136	-0.121	0.789
53.449	-1.961	2.100	-0.125	0.004	-1.124	-0.136	-0.121	0.789
55.702	-2.086	2.154	-0.129	0.004	-1.128	-0.136	-0.121	0.789
57.976	-2.205	2.196	-0.121	0.004	-1.117	-0.136	-0.121	0.789
60.254	-2.335	2.217	-0.117	0.012	-1.113	-0.136	-0.121	0.789
62.543	-2.447	2.230	-0.109	0.008	-1.117	-0.136	-0.121	0.789
64.827	-2.581	2.245	-0.117	0.012	-1.105	-0.136	-0.123	0.789

- Modo gyro-only y formato de matriz coseno-directora

El comando que ejecutaremos para obtener estas lecturas será el siguiente:

```
minimu9-ahrs -b /dev/i2c-1 --mode gyro-only --output matrix > gyromatrix.csv
```

Para una mejor comprensión vamos a desglosar el comando anterior:

- *--b /dev/i2c-1:* Con esta sentencia, especificamos el bus I2C con el que trabajará nuestra Raspberry.
- *--mode gyro-only:* especificamos que el sensor trabajará solo con el giroscopio.
- *--output matrix:* el formato de salida es con la matriz coseno-directora.
- *> gyromatrix.csv*: generamos un fichero nuevo, con el nombre gyromatrix.csv donde se escribirán las lecturas.

1.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	1.000	-0.117	0.016	-1.113	-0.127	-0.144	0.794
1.000	0.000	0.000	-0.000	1.000	-0.000	-0.000	0.000	1.000	-0.113	0.008	-1.105	-0.127	-0.144	0.794
1.000	0.000	0.000	-0.000	1.000	-0.000	-0.000	0.000	1.000	-0.113	0.004	-1.105	-0.127	-0.131	0.787
1.000	0.000	0.000	-0.000	1.000	-0.000	-0.000	0.000	1.000	-0.113	0.008	-1.120	-0.127	-0.131	0.787
1.000	0.000	0.000	-0.000	1.000	-0.000	-0.000	0.000	1.000	-0.109	0.004	-1.113	-0.127	-0.131	0.787
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.117	0.004	-1.113	-0.129	-0.144	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.121	0.004	-1.109	-0.129	-0.144	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.121	0.008	-1.109	-0.129	-0.144	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.113	0.012	-1.113	-0.129	-0.144	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.109	0.008	-1.113	-0.129	-0.144	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.113	0.008	-1.109	-0.129	-0.144	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.113	0.008	-1.113	-0.129	-0.133	0.794
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.113	0.016	-1.120	-0.129	-0.133	0.794
1.000	-0.000	-0.001	0.000	1.000	-0.001	0.001	0.001	1.000	-0.113	0.016	-1.120	-0.129	-0.133	0.794
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.113	0.004	-1.117	-0.129	-0.133	0.794
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.121	0.004	-1.120	-0.129	-0.133	0.794
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.121	0.012	-1.120	-0.129	-0.133	0.794
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.121	0.008	-1.113	-0.129	-0.133	0.794
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.113	0.004	-1.117	-0.123	-0.142	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.117	0.008	-1.128	-0.123	-0.142	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.117	0.004	-1.124	-0.123	-0.142	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.125	0.008	-1.117	-0.123	-0.142	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.117	0.004	-1.101	-0.123	-0.142	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.121	0.004	-1.097	-0.123	-0.142	0.791
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.125	0.008	-1.105	-0.123	-0.142	0.791
1.000	-0.001	-0.001	0.001	1.000	-0.000	0.001	0.000	1.000	-0.121	0.004	-1.113	-0.121	-0.138	0.800
1.000	-0.001	-0.001	0.001	1.000	-0.001	0.001	0.001	1.000	-0.117	0.008	-1.117	-0.121	-0.138	0.800
1.000	-0.001	-0.001	0.001	1.000	-0.000	0.001	0.000	1.000	-0.121	0.008	-1.113	-0.121	-0.138	0.800
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.117	0.000	-1.117	-0.121	-0.138	0.800
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.102	0.008	-1.117	-0.121	-0.138	0.800
1.000	-0.001	-0.001	0.001	1.000	-0.000	0.001	0.000	1.000	-0.117	0.000	-1.113	-0.121	-0.138	0.800
1.000	-0.001	-0.001	0.001	1.000	-0.000	0.001	0.000	1.000	-0.113	0.000	-1.124	-0.125	-0.140	0.794
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.113	0.004	-1.117	-0.125	-0.140	0.794
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.105	0.008	-1.120	-0.125	-0.140	0.794
1.000	-0.000	-0.001	0.000	1.000	-0.000	0.001	0.000	1.000	-0.109	0.008	-1.109	-0.125	-0.140	0.794
1.000	-0.000	-0.002	0.000	1.000	-0.000	0.002	0.000	1.000	-0.121	0.008	-1.120	-0.125	-0.140	0.794
1.000	-0.000	-0.002	0.000	1.000	-0.000	0.002	0.000	1.000	-0.109	0.004	-1.117	-0.125	-0.140	0.794
1.000	-0.000	-0.002	0.000	1.000	-0.000	0.002	0.000	1.000	-0.121	0.008	-1.120	-0.125	-0.140	0.794

- Modo gyro-only y formato de matriz quaternaria

El comando que ejecutaremos para obtener estas lecturas será el siguiente:

```
minimu9-ahrs –b /dev/i2c-1 --mode gyro-only --output quaternion › gyroquaternion.csv
```

Para una mejor comprensión vamos a desglosar el comando anterior:

- *--b /dev/i2c-1:* Con esta sentencia, especificamos el bus I2C con el que trabajará nuestra Raspberry.
- *--mode gyro-only:* especificamos que el sensor trabajará solo con el giroscopio.
- *--output quaternion:* el formato de salida es la matriz quaternaria.
- *> gyroquaternion.csv*: generamos un fichero nuevo, con el nombre gyroquaternion.csv donde se escribirán las lecturas.

1.000 0.000 0.000 0.000 -0.125 0.008 -1.109 -0.125 -0.119 0.810
1.000 -0.000 0.000 0.000 -0.117 0.008 -1.120 -0.125 -0.119 0.810
1.000 -0.000 0.000 0.000 -0.117 0.008 -1.124 -0.125 -0.119 0.810
1.000 -0.000 0.000 -0.000 -0.113 0.004 -1.124 -0.125 -0.119 0.810
1.000 -0.000 0.000 -0.000 -0.117 0.004 -1.120 -0.125 -0.119 0.810
1.000 -0.000 0.000 -0.000 -0.121 0.004 -1.120 -0.125 -0.119 0.810
1.000 -0.000 0.000 -0.000 -0.125 0.004 -1.109 -0.121 -0.123 0.807
1.000 -0.000 0.000 -0.000 -0.117 0.004 -1.113 -0.121 -0.123 0.807
1.000 -0.000 0.000 -0.000 -0.125 0.004 -1.124 -0.121 -0.123 0.807
1.000 -0.000 -0.000 -0.000 -0.117 0.004 -1.124 -0.121 -0.123 0.807
1.000 -0.000 -0.000 -0.000 -0.113 0.008 -1.124 -0.121 -0.123 0.807
1.000 -0.000 0.000 -0.000 -0.121 0.004 -1.120 -0.121 -0.123 0.807
1.000 -0.000 -0.000 -0.000 -0.117 0.004 -1.113 -0.121 -0.123 0.810
1.000 -0.000 -0.000 -0.000 -0.117 0.004 -1.124 -0.121 -0.123 0.810
1.000 -0.000 -0.000 -0.000 -0.121 0.004 -1.117 -0.121 -0.123 0.810
1.000 -0.000 -0.000 -0.000 -0.125 0.008 -1.117 -0.121 -0.123 0.810
1.000 -0.000 0.000 -0.000 -0.125 0.004 -1.113 -0.121 -0.123 0.810
1.000 -0.000 0.000 -0.000 -0.121 0.004 -1.113 -0.121 -0.123 0.810
1.000 0.000 0.000 -0.000 -0.121 0.004 -1.113 -0.121 -0.123 0.810
1.000 0.000 -0.000 -0.000 -0.125 0.008 -1.117 -0.129 -0.125 0.807
1.000 0.000 -0.000 -0.000 -0.125 0.004 -1.120 -0.129 -0.125 0.807
1.000 -0.000 -0.000 -0.000 -0.129 0.004 -1.124 -0.129 -0.125 0.807
1.000 -0.000 -0.000 -0.000 -0.117 0.012 -1.128 -0.129 -0.125 0.807
1.000 -0.000 -0.000 -0.000 -0.121 0.012 -1.117 -0.129 -0.125 0.807
1.000 -0.000 -0.000 -0.000 -0.117 0.008 -1.120 -0.129 -0.125 0.807
1.000 -0.000 -0.000 -0.000 -0.117 0.000 -1.117 -0.129 -0.119 0.810
1.000 0.000 -0.000 -0.000 -0.121 0.004 -1.109 -0.129 -0.119 0.810
1.000 -0.000 -0.000 -0.000 -0.113 0.008 -1.120 -0.129 -0.119 0.810
1.000 -0.000 -0.000 -0.000 -0.121 0.008 -1.113 -0.129 -0.119 0.810
1.000 -0.000 -0.000 -0.000 -0.113 0.004 -1.109 -0.129 -0.119 0.810
1.000 0.000 0.000 -0.000 -0.113 0.000 -1.120 -0.129 -0.119 0.810
1.000 0.000 0.000 -0.000 -0.113 -0.004 -1.124 -0.129 -0.119 0.810
1.000 0.000 -0.000 -0.000 -0.113 0.008 -1.120 -0.121 -0.125 0.805
1.000 -0.000 -0.000 -0.000 -0.121 0.012 -1.120 -0.121 -0.125 0.805
1.000 -0.000 -0.000 -0.000 -0.121 0.008 -1.117 -0.121 -0.125 0.805
1.000 0.000 -0.000 -0.000 -0.125 0.000 -1.128 -0.121 -0.125 0.805
1.000 0.000 -0.000 -0.000 -0.125 0.004 -1.117 -0.121 -0.125 0.805
1.000 0.000 -0.000 -0.000 -0.121 0.004 -1.120 -0.121 -0.125 0.805
1.000 0.000 -0.000 -0.000 -0.129 0.008 -1.105 -0.121 -0.123 0.805

- Modo gyro-only y formato ángulos de Euler

El comando que ejecutaremos para obtener estas lecturas será el siguiente:

```
minimu9-ahrs –b /dev/i2c-1 --mode gyro-only --output euler > gyroeuler.csv
```

Para una mejor comprensión vamos a desglosar el comando anterior:

- *--b /dev/i2c-1:* Con esta sentencia, especificamos el bus I2C con el que trabajará nuestra Raspberry.
- *--mode gyro-only:* especificamos que el sensor trabajará solo con el giroscopio.
- *--output euler:* el formato de salida son los ángulos de Euler.
- *> gyroeuler.csv*: generamos un fichero nuevo, con el nombre gyroeuler.csv donde se escribirán las lecturas.

0.000	-0.000	0.000	-0.102	0.008	-1.109	-0.162	-0.060	0.784
-0.003	-0.001	-0.005	-0.105	0.008	-1.105	-0.162	-0.060	0.784
0.001	-0.006	-0.002	-0.117	0.004	-1.101	-0.162	-0.060	0.784
-0.006	-0.006	-0.006	-0.117	0.012	-1.113	-0.160	-0.051	0.787
-0.008	-0.010	-0.005	-0.121	0.012	-1.113	-0.160	-0.051	0.787
-0.012	-0.014	-0.004	-0.121	0.012	-1.113	-0.160	-0.051	0.787
-0.011	-0.017	-0.003	-0.117	0.012	-1.113	-0.160	-0.051	0.787
-0.010	-0.017	-0.002	-0.113	0.004	-1.109	-0.160	-0.051	0.787
-0.004	-0.019	-0.004	-0.121	0.008	-1.109	-0.160	-0.051	0.787
-0.000	-0.019	-0.008	-0.109	0.008	-1.120	-0.160	-0.051	0.787
0.008	-0.018	-0.004	-0.109	0.008	-1.113	-0.154	-0.051	0.782
0.009	-0.023	-0.014	-0.105	0.008	-1.113	-0.154	-0.051	0.782
0.012	-0.029	-0.020	-0.109	0.012	-1.120	-0.154	-0.051	0.782
0.016	-0.023	-0.011	-0.121	0.012	-1.113	-0.154	-0.051	0.782
0.021	-0.025	-0.013	-0.117	0.008	-1.109	-0.154	-0.051	0.782
0.021	-0.021	-0.006	-0.113	0.004	-1.113	-0.154	-0.051	0.782
0.026	-0.023	-0.013	-0.113	0.008	-1.113	-0.160	-0.054	0.784
0.024	-0.025	-0.011	-0.117	0.008	-1.101	-0.160	-0.054	0.784
0.027	-0.022	-0.006	-0.113	0.004	-1.097	-0.160	-0.054	0.784
0.034	-0.025	-0.002	-0.109	0.008	-1.113	-0.160	-0.054	0.784
0.036	-0.029	-0.001	-0.117	0.008	-1.113	-0.160	-0.054	0.784
0.032	-0.027	-0.002	-0.113	0.012	-1.117	-0.160	-0.054	0.784
0.033	-0.022	-0.002	-0.117	0.004	-1.120	-0.156	-0.051	0.791
0.028	-0.020	-0.003	-0.117	0.004	-1.120	-0.156	-0.051	0.791
0.029	-0.017	-0.012	-0.117	0.016	-1.117	-0.156	-0.051	0.791
0.021	-0.029	-0.017	-0.113	0.004	-1.117	-0.156	-0.051	0.791
0.014	-0.031	-0.016	-0.113	0.008	-1.113	-0.156	-0.051	0.791
0.008	-0.043	-0.015	-0.117	0.004	-1.109	-0.156	-0.051	0.791
0.012	-0.051	-0.010	-0.113	0.004	-1.113	-0.156	-0.051	0.791
0.011	-0.055	-0.004	-0.113	0.000	-1.101	-0.162	-0.049	0.789
0.008	-0.048	-0.006	-0.117	0.000	-1.117	-0.162	-0.049	0.789
0.007	-0.049	-0.008	-0.121	0.008	-1.117	-0.162	-0.049	0.789
0.008	-0.044	-0.018	-0.117	0.004	-1.113	-0.162	-0.049	0.789
0.009	-0.045	-0.014	-0.117	0.008	-1.101	-0.162	-0.049	0.789
0.002	-0.048	-0.013	-0.109	0.004	-1.105	-0.162	-0.049	0.789
0.002	-0.042	-0.016	-0.117	0.004	-1.120	-0.162	-0.049	0.789
-0.001	-0.044	-0.020	-0.117	0.008	-1.128	-0.160	-0.056	0.782
0.001	-0.040	-0.022	-0.109	0.012	-1.120	-0.160	-0.056	0.782
0.006	-0.048	-0.021	-0.105	0.012	-1.113	-0.160	-0.056	0.782

- Modo compass-only y formato de matriz coseno-directora

El comando que ejecutaremos para obtener estas lecturas será el siguiente:

```
minimu9-ahrs -b /dev/i2c-1 --mode compass-only --output matrix > compassmatrix.csv
```

Para una mejor comprensión vamos a desglosar el comando anterior:

- *--b /dev/i2c-1:* Con esta sentencia, especificamos el bus I2C con el que trabajará nuestra Raspberry.
- *--mode compass-only:* especificamos que el sensor trabajará solo con el magnetómetro.
- *--output matrix:* el formato de salida es con la matriz coseno-directora.
- *> compassmatrix.csv*: generamos un fichero nuevo, con el nombre compassmatrix.csv donde se escribirán las lecturas.

-0.939	-0.331	0.094	0.329	-0.944	-0.037	0.101	-0.003	0.995	-0.113	0.004	-1.117	-0.144	-0.081	0.798
-0.941	-0.324	0.100	0.322	-0.946	-0.038	0.107	-0.003	0.994	-0.121	0.004	-1.120	-0.144	-0.081	0.798
-0.949	-0.297	0.106	0.294	-0.955	-0.044	0.115	-0.010	0.993	-0.129	0.012	-1.117	-0.144	-0.081	0.798
-0.948	-0.300	0.103	0.297	-0.954	-0.043	0.112	-0.010	0.994	-0.125	0.012	-1.113	-0.144	-0.081	0.798
-0.940	-0.325	0.103	0.322	-0.946	-0.043	0.111	-0.007	0.994	-0.125	0.008	-1.117	-0.148	-0.087	0.807
-0.942	-0.322	0.095	0.319	-0.947	-0.043	0.104	-0.010	0.995	-0.117	0.012	-1.120	-0.148	-0.087	0.807
-0.944	-0.315	0.102	0.312	-0.949	-0.045	0.111	-0.010	0.994	-0.125	0.012	-1.117	-0.148	-0.087	0.807
-0.938	-0.331	0.107	0.328	-0.944	-0.041	0.115	-0.003	0.993	-0.129	0.004	-1.113	-0.148	-0.087	0.807
-0.941	-0.327	0.090	0.325	-0.945	-0.042	0.098	-0.011	0.995	-0.109	0.012	-1.105	-0.148	-0.087	0.807
-0.949	-0.303	0.092	0.300	-0.953	-0.048	0.102	-0.018	0.995	-0.113	0.020	-1.105	-0.148	-0.087	0.807
-0.950	-0.298	0.098	0.294	-0.954	-0.049	0.108	-0.017	0.994	-0.121	0.020	-1.113	-0.148	-0.087	0.807
-0.938	-0.330	0.103	0.327	-0.944	-0.043	0.112	-0.007	0.994	-0.125	0.008	-1.113	-0.144	-0.087	0.805
-0.935	-0.339	0.104	0.337	-0.941	-0.041	0.112	-0.003	0.994	-0.125	0.004	-1.109	-0.144	-0.087	0.805
-0.941	-0.323	0.099	0.320	-0.946	-0.045	0.108	-0.010	0.994	-0.121	0.012	-1.113	-0.144	-0.087	0.805
-0.938	-0.330	0.103	0.327	-0.944	-0.043	0.112	-0.007	0.994	-0.125	0.008	-1.113	-0.144	-0.087	0.805
-0.932	-0.346	0.108	0.344	-0.938	-0.040	0.115	-0.000	0.993	-0.129	0.000	-1.117	-0.144	-0.087	0.805
-0.930	-0.354	0.100	0.352	-0.935	-0.038	0.107	0.000	0.994	-0.121	0.000	-1.120	-0.144	-0.087	0.805
-0.930	-0.354	0.101	0.352	-0.935	-0.038	0.109	0.000	0.994	-0.121	0.000	-1.109	-0.144	-0.087	0.794
-0.931	-0.351	0.097	0.349	-0.936	-0.040	0.105	-0.003	0.994	-0.117	0.004	-1.113	-0.142	-0.087	0.794
-0.940	-0.325	0.101	0.322	-0.946	-0.046	0.111	-0.010	0.994	-0.125	0.012	-1.120	-0.142	-0.087	0.794
-0.944	-0.315	0.101	0.311	-0.949	-0.048	0.111	-0.014	0.994	-0.125	0.016	-1.117	-0.142	-0.087	0.794
-0.937	-0.335	0.102	0.332	-0.942	-0.044	0.111	-0.007	0.994	-0.125	0.008	-1.120	-0.142	-0.087	0.794
-0.934	-0.345	0.092	0.343	-0.938	-0.041	0.101	-0.007	0.995	-0.113	0.008	-1.117	-0.142	-0.087	0.794
-0.941	-0.324	0.099	0.321	-0.946	-0.045	0.108	-0.010	0.994	-0.121	0.012	-1.117	-0.144	-0.087	0.803
-0.944	-0.314	0.098	0.311	-0.949	-0.047	0.108	-0.014	0.994	-0.121	0.016	-1.117	-0.144	-0.087	0.803
-0.936	-0.337	0.096	0.335	-0.941	-0.042	0.104	-0.007	0.995	-0.117	0.008	-1.117	-0.144	-0.087	0.803
-0.937	-0.334	0.099	0.331	-0.943	-0.043	0.108	-0.007	0.994	-0.121	0.008	-1.117	-0.144	-0.087	0.803
-0.938	-0.331	0.102	0.328	-0.944	-0.043	0.111	-0.007	0.994	-0.125	0.008	-1.120	-0.144	-0.087	0.803
-0.934	-0.344	0.100	0.342	-0.939	-0.040	0.107	-0.003	0.994	-0.121	0.004	-1.120	-0.144	-0.087	0.803
-0.939	-0.331	0.092	0.328	-0.944	-0.043	0.101	-0.010	0.995	-0.113	0.012	-1.117	-0.144	-0.087	0.803
-0.934	-0.345	0.097	0.342	-0.939	-0.043	0.106	-0.007	0.994	-0.117	0.008	-1.101	-0.142	-0.089	0.805
-0.930	-0.355	0.098	0.353	-0.935	-0.041	0.106	-0.004	0.994	-0.117	0.004	-1.101	-0.142	-0.089	0.805
-0.928	-0.359	0.104	0.356	-0.933	-0.040	0.112	0.000	0.994	-0.125	0.000	-1.113	-0.142	-0.089	0.805
-0.931	-0.350	0.103	0.347	-0.937	-0.042	0.111	-0.003	0.994	-0.125	0.004	-1.120	-0.142	-0.089	0.805
-0.934	-0.344	0.098	0.342	-0.939	-0.043	0.107	-0.007	0.994	-0.121	0.008	-1.128	-0.142	-0.089	0.805
-0.927	-0.362	0.101	0.360	-0.932	-0.039	0.108	0.000	0.994	-0.121	0.000	-1.113	-0.142	-0.089	0.805
-0.941	-0.325	0.096	0.322	-0.946	-0.040	0.104	-0.007	0.995	-0.117	0.008	-1.120	-0.144	-0.083	0.796
-0.941	-0.324	0.097	0.322	-0.946	-0.040	0.105	-0.007	0.994	-0.117	0.008	-1.113	-0.144	-0.083	0.796

- Modo compass-only y formato de matriz quaternaria

El comando que ejecutaremos para obtener estas lecturas será el siguiente:

```
minimu9-ahrs -b /dev/i2c-1 --mode compass-only --output quaternion > compassquaternion.csv
```

Para una mejor comprensión vamos a desglosar el comando anterior:

- *--b /dev/i2c-1:* Con esta sentencia, especificamos el bus I2C con el que trabajará nuestra Raspberry.
- *--mode compass-only:* especificamos que el sensor trabajará solo con el magnetómetro.
- *--output quaternion:* el formato de salida es la matriz quaternaria.
- *> compassquaternion.csv*: generamos un fichero nuevo, con el nombre compassquaternion.csv donde se escribirán las lecturas.

-0.939	-0.331	0.094	0.329	-0.944	-0.037	0.101	-0.003	0.995	-0.113	0.004	-1.117	-0.144	-0.081	0.798
-0.941	-0.324	0.100	0.322	-0.946	-0.038	0.107	-0.003	0.994	-0.121	0.004	-1.120	-0.144	-0.081	0.798
-0.949	-0.297	0.106	0.294	-0.955	-0.044	0.115	-0.010	0.993	-0.129	0.012	-1.117	-0.144	-0.081	0.798
-0.948	-0.300	0.103	0.297	-0.954	-0.043	0.112	-0.010	0.994	-0.125	0.012	-1.113	-0.144	-0.081	0.798
-0.940	-0.325	0.103	0.322	-0.946	-0.043	0.111	-0.007	0.994	-0.125	0.008	-1.117	-0.148	-0.087	0.807
-0.942	-0.322	0.095	0.319	-0.947	-0.043	0.104	-0.010	0.995	-0.117	0.012	-1.120	-0.148	-0.087	0.807
-0.944	-0.315	0.102	0.312	-0.949	-0.045	0.111	-0.010	0.994	-0.125	0.012	-1.117	-0.148	-0.087	0.807
-0.938	-0.331	0.107	0.328	-0.944	-0.041	0.115	-0.003	0.993	-0.129	0.004	-1.113	-0.148	-0.087	0.807
-0.941	-0.327	0.090	0.325	-0.945	-0.042	0.098	-0.011	0.995	-0.109	0.012	-1.105	-0.148	-0.087	0.807
-0.949	-0.303	0.092	0.300	-0.953	-0.048	0.102	-0.018	0.995	-0.113	0.020	-1.105	-0.148	-0.087	0.807
-0.950	-0.298	0.098	0.294	-0.954	-0.049	0.108	-0.017	0.994	-0.121	0.020	-1.113	-0.148	-0.087	0.807
-0.938	-0.330	0.103	0.327	-0.944	-0.043	0.112	-0.007	0.994	-0.125	0.008	-1.113	-0.144	-0.087	0.805
-0.935	-0.339	0.104	0.337	-0.941	-0.041	0.112	-0.003	0.994	-0.125	0.004	-1.109	-0.144	-0.087	0.805
-0.941	-0.323	0.099	0.320	-0.946	-0.045	0.108	-0.010	0.994	-0.121	0.012	-1.113	-0.144	-0.087	0.805
-0.938	-0.330	0.103	0.327	-0.944	-0.043	0.112	-0.007	0.994	-0.125	0.008	-1.113	-0.144	-0.087	0.805
-0.932	-0.346	0.108	0.344	-0.938	-0.040	0.115	-0.000	0.993	-0.129	0.000	-1.117	-0.144	-0.087	0.805
-0.930	-0.354	0.100	0.352	-0.935	-0.038	0.107	0.000	0.994	-0.121	0.000	-1.120	-0.144	-0.087	0.805
-0.930	-0.354	0.101	0.352	-0.935	-0.038	0.109	0.000	0.994	-0.121	0.000	-1.109	-0.144	-0.087	0.794
-0.931	-0.351	0.097	0.349	-0.936	-0.040	0.105	-0.003	0.994	-0.117	0.004	-1.113	-0.142	-0.087	0.794
-0.940	-0.325	0.101	0.322	-0.946	-0.046	0.111	-0.010	0.994	-0.125	0.012	-1.120	-0.142	-0.087	0.794
-0.944	-0.315	0.101	0.311	-0.949	-0.048	0.111	-0.014	0.994	-0.125	0.016	-1.117	-0.142	-0.087	0.794
-0.937	-0.335	0.102	0.332	-0.942	-0.044	0.111	-0.007	0.994	-0.125	0.008	-1.120	-0.142	-0.087	0.794
-0.934	-0.345	0.092	0.343	-0.938	-0.041	0.101	-0.007	0.995	-0.113	0.008	-1.117	-0.142	-0.087	0.794
-0.941	-0.324	0.099	0.321	-0.946	-0.045	0.108	-0.010	0.994	-0.121	0.012	-1.117	-0.144	-0.087	0.803
-0.944	-0.314	0.098	0.311	-0.949	-0.047	0.108	-0.014	0.994	-0.121	0.016	-1.117	-0.144	-0.087	0.803
-0.936	-0.337	0.096	0.335	-0.941	-0.042	0.104	-0.007	0.995	-0.117	0.008	-1.117	-0.144	-0.087	0.803
-0.937	-0.334	0.099	0.331	-0.943	-0.043	0.108	-0.007	0.994	-0.121	0.008	-1.117	-0.144	-0.087	0.803
-0.938	-0.331	0.102	0.328	-0.944	-0.043	0.111	-0.007	0.994	-0.125	0.008	-1.120	-0.144	-0.087	0.803
-0.934	-0.344	0.100	0.342	-0.939	-0.040	0.107	-0.003	0.994	-0.121	0.004	-1.120	-0.144	-0.087	0.803
-0.939	-0.331	0.092	0.328	-0.944	-0.043	0.101	-0.010	0.995	-0.113	0.012	-1.117	-0.144	-0.087	0.803
-0.934	-0.345	0.097	0.342	-0.939	-0.043	0.106	-0.007	0.994	-0.117	0.008	-1.101	-0.142	-0.089	0.805
-0.930	-0.355	0.098	0.353	-0.935	-0.041	0.106	-0.004	0.994	-0.117	0.004	-1.101	-0.142	-0.089	0.805
-0.928	-0.359	0.104	0.356	-0.933	-0.040	0.112	0.000	0.994	-0.125	0.000	-1.113	-0.142	-0.089	0.805
-0.931	-0.350	0.103	0.347	-0.937	-0.042	0.111	-0.003	0.994	-0.125	0.004	-1.120	-0.142	-0.089	0.805
-0.934	-0.344	0.098	0.342	-0.939	-0.043	0.107	-0.007	0.994	-0.121	0.008	-1.128	-0.142	-0.089	0.805
-0.927	-0.362	0.101	0.360	-0.932	-0.039	0.108	0.000	0.994	-0.121	0.000	-1.113	-0.142	-0.089	0.805
-0.941	-0.325	0.096	0.322	-0.946	-0.040	0.104	-0.007	0.995	-0.117	0.008	-1.120	-0.144	-0.083	0.796
-0.941	-0.324	0.097	0.322	-0.946	-0.040	0.105	-0.007	0.994	-0.117	0.008	-1.113	-0.144	-0.083	0.796

- Modo compass-only y formato ángulos de Euler

El comando que ejecutaremos para obtener estas lecturas será el siguiente:

```
minimu9-ahrs –b /dev/i2c-1 --mode compass-only --output euler › compasseuler.csv
```

Para una mejor comprensión vamos a desglosar el comando anterior:

- *--b /dev/i2c-1:* Con esta sentencia, especificamos el bus I2C con el que trabajará nuestra Raspberry.
- *--mode compass-only:* especificamos que el sensor trabajará solo con el magnetómetro.
- *--output euler:* el formato de salida son los ángulos de Euler.
- *> compasseuler.csv*: generamos un fichero nuevo, con el nombre compasseuler.csv donde se escribirán las lecturas.

153.432	-6.251	-0.405	-0.121	0.008	-1.105	-0.140	-0.117	0.775
152.279	-5.830	-0.202	-0.113	0.004	-1.109	-0.140	-0.117	0.775
151.913	-5.967	0.000	-0.117	0.000	-1.120	-0.140	-0.117	0.775
153.137	-5.967	0.000	-0.117	0.000	-1.120	-0.140	-0.112	0.780
154.235	-6.362	-0.200	-0.125	0.004	-1.120	-0.140	-0.112	0.780
154.235	-6.362	-0.200	-0.125	0.004	-1.120	-0.140	-0.112	0.780
154.575	-6.186	-0.401	-0.121	0.008	-1.117	-0.140	-0.112	0.780
153.492	-5.810	-0.201	-0.113	0.004	-1.113	-0.140	-0.112	0.780
153.432	-5.770	-0.200	-0.113	0.004	-1.120	-0.140	-0.112	0.780
152.732	-5.592	-0.200	-0.109	0.004	-1.117	-0.136	-0.112	0.777
153.332	-6.009	-0.201	-0.117	0.004	-1.113	-0.136	-0.112	0.777
154.572	-6.051	-0.607	-0.117	0.012	-1.105	-0.136	-0.112	0.777
155.727	-6.473	-0.813	-0.125	0.016	-1.101	-0.136	-0.112	0.777
154.253	-6.251	-0.405	-0.121	0.008	-1.105	-0.136	-0.112	0.777
153.672	-6.251	-0.202	-0.121	0.004	-1.105	-0.136	-0.112	0.777
153.608	-6.208	-0.201	-0.121	0.004	-1.113	-0.136	-0.112	0.777
153.127	-5.947	-0.199	-0.117	0.004	-1.124	-0.138	-0.114	0.784
153.433	-6.165	-0.200	-0.121	0.004	-1.120	-0.138	-0.114	0.784
153.189	-5.988	-0.200	-0.117	0.004	-1.117	-0.138	-0.114	0.784
151.983	-5.553	0.000	-0.109	0.000	-1.124	-0.138	-0.114	0.784
151.192	-5.412	0.201	-0.105	-0.004	-1.113	-0.138	-0.114	0.784
153.220	-6.009	-0.201	-0.117	0.004	-1.113	-0.138	-0.114	0.784
155.003	-6.051	-0.405	-0.117	0.008	-1.105	-0.142	-0.110	0.780
154.453	-6.072	-0.203	-0.117	0.004	-1.101	-0.142	-0.110	0.780
155.229	-6.229	-0.403	-0.121	0.008	-1.109	-0.142	-0.110	0.780
155.743	-6.186	-0.601	-0.121	0.012	-1.117	-0.142	-0.110	0.780
154.847	-5.947	-0.398	-0.117	0.008	-1.124	-0.142	-0.110	0.780
154.790	-6.340	-0.199	-0.125	0.004	-1.124	-0.142	-0.110	0.780
154.980	-6.492	-0.198	-0.129	0.004	-1.132	-0.142	-0.110	0.780
154.274	-5.947	-0.199	-0.117	0.004	-1.124	-0.142	-0.110	0.780
154.594	-6.186	-0.200	-0.121	0.004	-1.117	-0.142	-0.110	0.780
154.303	-5.967	-0.200	-0.117	0.004	-1.120	-0.142	-0.110	0.780
154.066	-5.790	-0.200	-0.113	0.004	-1.117	-0.142	-0.110	0.780
153.515	-5.393	-0.200	-0.105	0.004	-1.117	-0.142	-0.110	0.780
154.363	-6.009	-0.201	-0.117	0.004	-1.113	-0.142	-0.110	0.780
153.515	-5.830	-0.202	-0.113	0.004	-1.109	-0.136	-0.110	0.780
153.572	-5.450	-0.405	-0.105	0.008	-1.105	-0.136	-0.110	0.780
153.260	-5.650	-0.202	-0.109	0.004	-1.105	-0.136	-0.110	0.780

- Modo raw

El comando que ejecutaremos para obtener estas lecturas será el siguiente:

```
minimu9-ahrs –b /dev/i2c-1 --mode raw > raw.csv
```

Para una mejor comprensión vamos a desglosar el comando anterior:

- *--b /dev/i2c-1:* Con esta sentencia, especificamos el bus I2C con el que trabajará nuestra Raspberry.
- *--mode raw:* el sensor dará lecturas de todos los sensores en crudo.
- *> raw.csv*: generamos un fichero nuevo, con el nombre raw .csv donde se escribirán las lecturas.

-203	-247	418	-464	48	-4608	4	-4	4
-203	-248	419	-512	32	-4624	5	-3	4
-203	-248	419	-480	32	-4544	-1	-2	1
-203	-248	419	-496	16	-4592	-4	-8	4
-203	-248	419	-480	16	-4592	-1	-6	-2
-203	-248	419	-480	32	-4592	-2	-7	1
-203	-248	419	-480	32	-4592	5	-2	5
-202	-250	419	-480	32	-4592	3	-6	-1
-202	-250	419	-528	16	-4544	1	-10	2
-202	-250	419	-512	32	-4592	4	-8	4
-202	-250	419	-496	16	-4576	-1	-14	3
-202	-250	419	-496	16	-4592	2	-5	-3
-204	-248	417	-480	48	-4608	3	-3	7
-204	-248	417	-464	32	-4608	-5	0	-1
-204	-248	417	-480	48	-4608	6	-7	-2
-204	-248	417	-512	32	-4608	-1	0	4
-204	-248	417	-496	16	-4624	-1	-7	7
-204	-248	417	-496	32	-4608	0	-5	-1
-204	-247	420	-480	32	-4640	1	-5	5
-204	-247	420	-512	0	-4608	-6	-1	4
-204	-247	420	-512	0	-4592	-1	-2	-3
-204	-247	420	-480	16	-4544	2	-4	3
-204	-247	420	-496	32	-4544	1	-5	4
-204	-247	420	-480	16	-4544	-1	-3	4
-203	-247	417	-512	16	-4560	-4	-4	6
-203	-247	417	-528	32	-4576	1	-4	3
-203	-247	417	-512	16	-4528	3	-8	3
-203	-247	417	-528	32	-4544	1	-5	8
-203	-247	417	-544	16	-4544	2	-6	3
-203	-247	417	-528	16	-4576	2	-5	-2
-204	-250	416	-496	16	-4576	1	-4	3
-204	-250	416	-528	32	-4576	-2	-6	5
-204	-250	416	-480	0	-4544	2	-8	3
-204	-250	416	-512	16	-4544	3	-4	6
-204	-250	416	-512	16	-4528	8	-5	0
-202	-249	415	-512	16	-4560	3	-4	3
-202	-249	415	-512	16	-4576	4	-8	1
-202	-249	415	-480	48	-4560	1	-8	0
-202	-249	415	-432	32	-4560	-3	-9	-1
-202	-249	415	-496	32	-4560	2	-9	-1

5. VISUALIZACIÓN 3D DE LA IMU CON EL SOFTAWARE ahrs-visualizer

Como se comentó anteriormente en el apartado en el que se detallaba la instalación del software *ahrs-visualizer*, el paquete de software que viene asociado con la IMU tiene la posibilidad de visualizar en pantalla el sensor IMU y sus tres ejes.

Para iniciar dicha posibilidad, debemos lanzar el siguiente comando por consola:

```
minimu9-ahrs | ahrs-visualizer
```

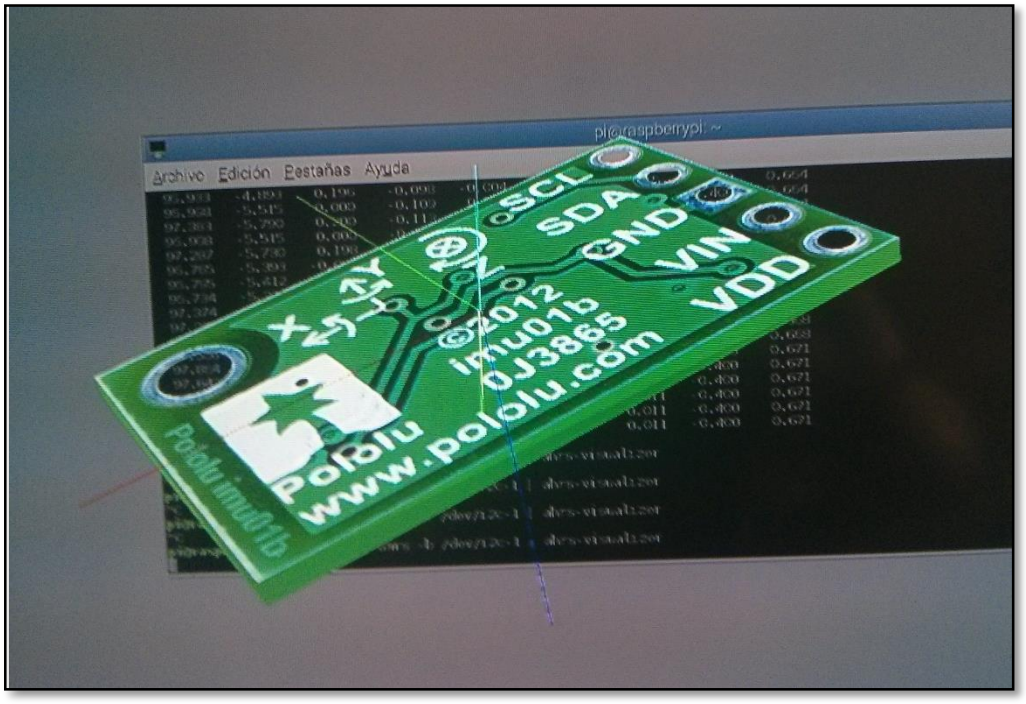
Si lanzamos este comando teniendo la Raspberry en su versión de 512 MB nos dará un error de ejecución que nos devolverá el siguiente mensaje:

```
Error: Could not open file i2c device: No such file or directory
```

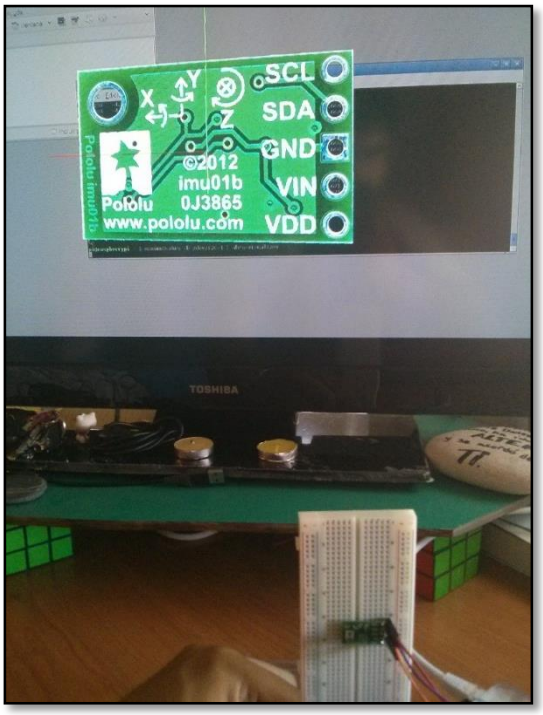
Esto es como ya hemos comentado anteriormente debido al kernel del modelo de la Raspberry, lo que nos obliga a especificar como siempre que el puerto con el que debe trabajar es el puerto 1, por ello el comando que deberemos lanzar será:

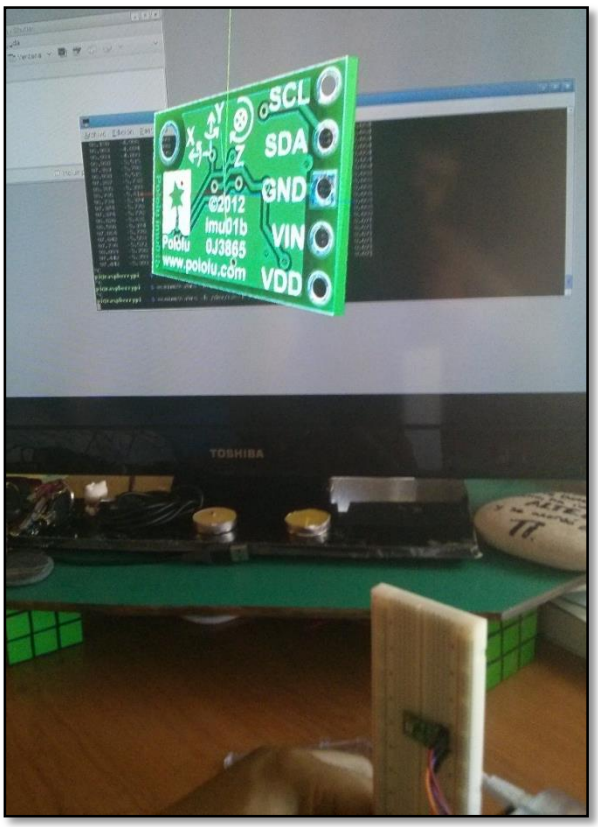
```
minimu9-ahrs -b /dev/i2c-1 | ahrs-visualizer
```

Con esto se iniciará el visualizador y podremos ver la IMU en pantalla. Un modelo 3D que viene proporcionado por el software ahrs-visualizer:



Como vemos en la siguiente secuencia de imágenes, todos los movimientos que aplicamos al sensor, se transmiten en pantalla:





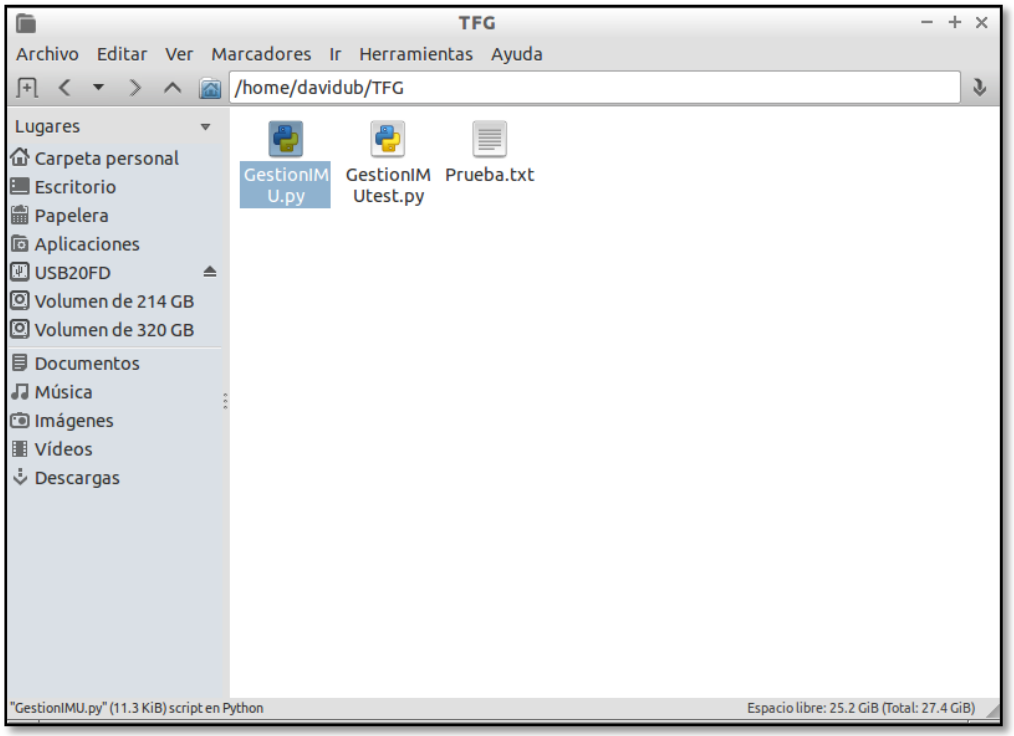
6. FUNCIONAMIENTO DEL SOFTWARE GestionIMU v.1

Como hemos comentado anteriormente, para este proyecto se ha confeccionado un software o script con Python para simplificar el manejo y gestión de la IMU con la que vamos a trabajar. Este software estaba concebido como una interfaz gráfica que mediante botones y diferentes *checks* se fuesen construyendo los comandos del sistema operativo que se lanzarían desde esta misma interfaz.

En este apartado vamos a intentar explicar cada una de las ventanas y opciones que tienen este programa, y para ello nos apoyaremos en el Anexo 1, en el que está todo el código escrito y dividido en imágenes para que sea, más sencillo hacer referencias a él.

6.1. Inicio del programa

Lo primero que haremos, será con el botón derecho del ratón desplegaremos el menú de opciones y seleccionamos IDLE Python 2.7.



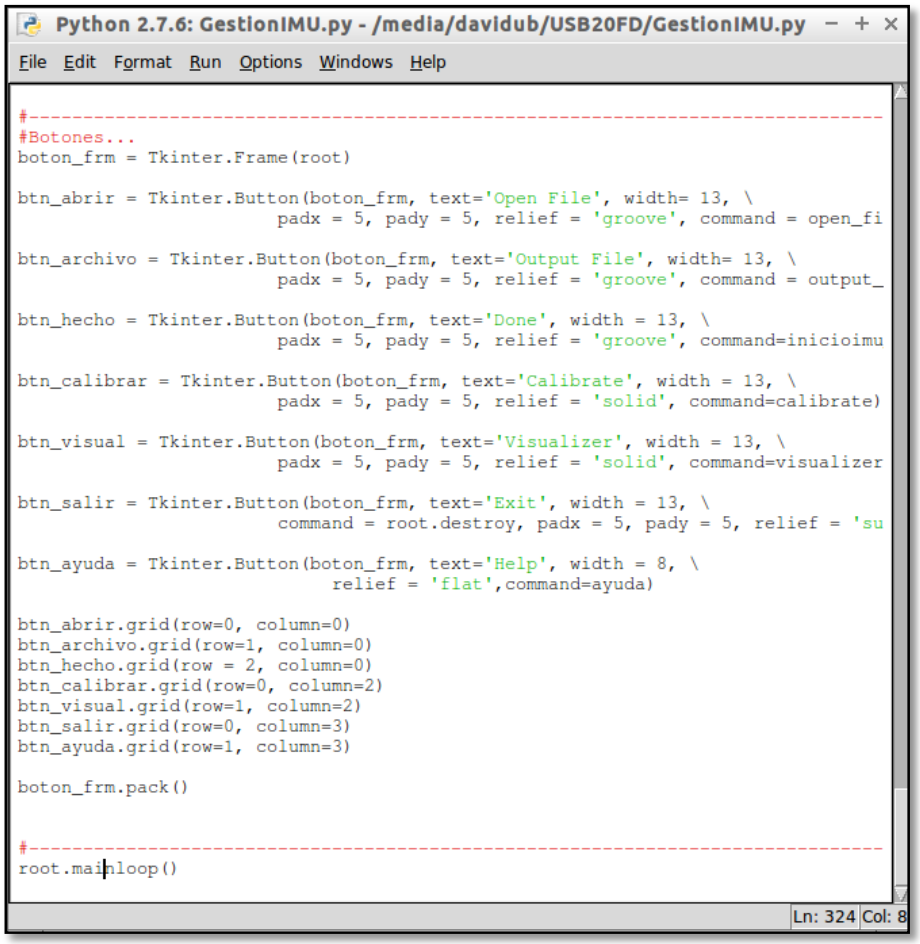
6.2. IDLE Python

Una vez abierto el IDLE con todo el código presionaremos F5, de este modo, primero se guardará el script y se ejecutará. Será entonces cuando nos aparecerá el interfaz gráfico programado. Aquí deberíamos hacer referencia a la Imagen 1 del Anexo I, para explicar, que para que nuestro script funcione correctamente, las dos primeras líneas que debemos poner son:

```
1)  #!/usr/bin/env python
2)  #-*- coding: utf-8 -*-
```

La primera línea comentada en Linux (nuestro caso Raspbian) se llama "Shebang". Debemos ponerla porque, como se ha dicho, este programa interactuara con el SO[21], y es esencial para indicar a la máquina con qué programa vamos a ejecutar dicho script. Python es un lenguaje de script, por tanto necesita de un programa que interprete sus líneas y muestre la salida.

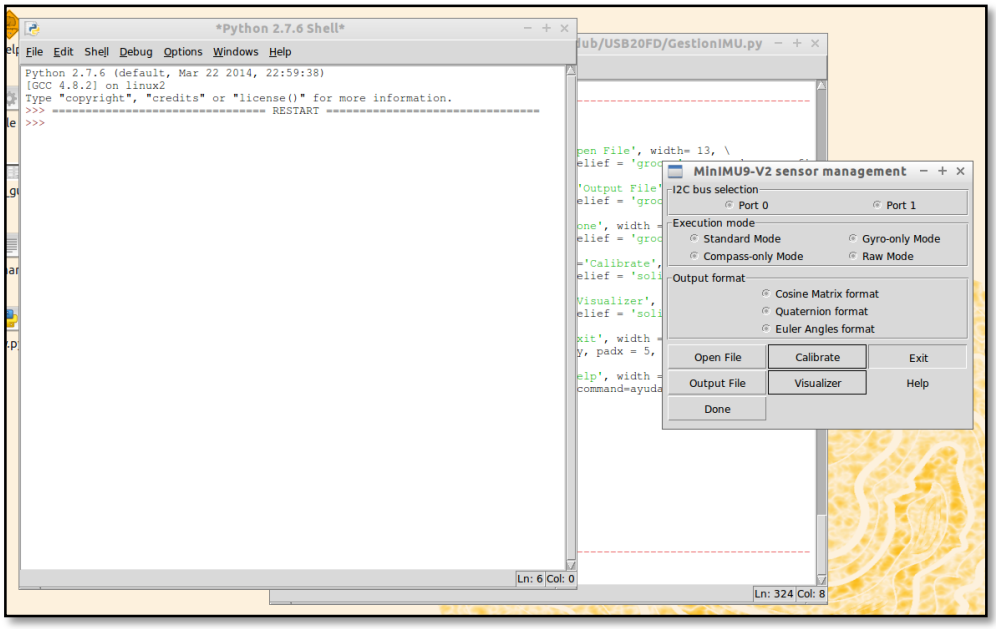
La segunda línea es la línea de codificación, que al contrario que la línea "Shebang" no es obligatoria. La función es la de establecer la codificación UTF-8 de modo que el compilador sea capaz de interpretar los caracteres especiales (ñ, á, ü, ò).



[21] SO: Sistema Operativo

6.3. Ejecución del script

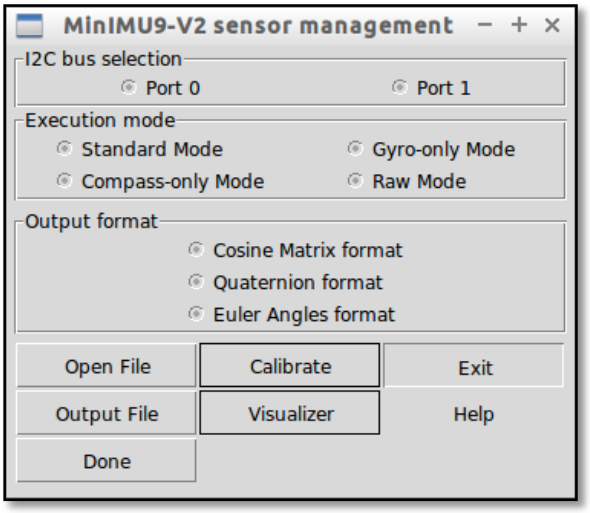
Una vez ejecutamos el script, se abrirá el Shell de Python y aparecerá la ventana grafica programada con Tkinter como se ve en la imagen inferior.



Al programar la ventana gráfica con Tkinter, se ha buscado obtener un resultado agradable a la vista y amigable en su utilización. Cada botón existente en la interfaz está programado para que esté asociado a una función determinada, de modo que cada botón presionado llama a una determinada función que iremos viendo poco a poco.

6.4. GUI principal GestionIMU v.1

El GUI principal se ha querido estructurar de modo que el usuario vaya seleccionando los parámetros necesarios para la construcción del comando necesario para el funcionamiento de la IMU. Como vemos, lo primero será seleccionar el puerto I2C con el que trabaja la Raspberry. Como se ha explicado anteriormente, dependiendo del modelo de la Raspberry, esta trabaja por defecto con un puerto determinado. En este proyecto se ha utilizado una Raspberry Pi B+, eso significa que deberemos seleccionar el puerto 1 en el primer *LabelFrame* de selección.



El segundo *LabelFrame* nos permite seleccionar en qué modo de ejecución se va a iniciar la IMU. Como hemos explicado anteriormente, si se seleccionase el modo *Standard Mode* obtendríamos lecturas del giroscopio y el acelerómetro simultáneamente. Si seleccionamos *Gyro-only Mode* solo se tendrá lecturas del giroscopio, que como sabemos registra los giros producidos por los ejes del sistema de referencia del dispositivo. El modo *Compass-only Mode* solo ofrece las lecturas del acelerómetro del sensor de la IMU. Para obtener los valores de ganancia que obtiene el sensor debemos ejecutar el modo *Raw Mode*. Este modo nos dará tres vectores diferentes referentes a las lecturas en crudo del al acelerómetro, el giroscopio y el magnetómetro.

El tercer *LabelFrame* nos permitirá elegir en que formato de salida queremos tener nuestras lecturas. Tendremos tres diferentes formato: *Cosine Matrix format* es una matrix de transformación coseno-directora en el que sus valores no tienen unidades propiamente dichas, sino que son los coeficientes de transformación entre el sistema de coordenadas del dispositivo y el sistema de coordenadas de suelo. *Quaternion format* da como resultado una matriz de transformación quaternaria cuyo objetivo es el mismo que de la matriz coseno-directora. Finalmente está el *Euler Angles format*, que nos dará las lecturas con los ángulos de Euler, lo que para nuestra especialidad es muy útil.

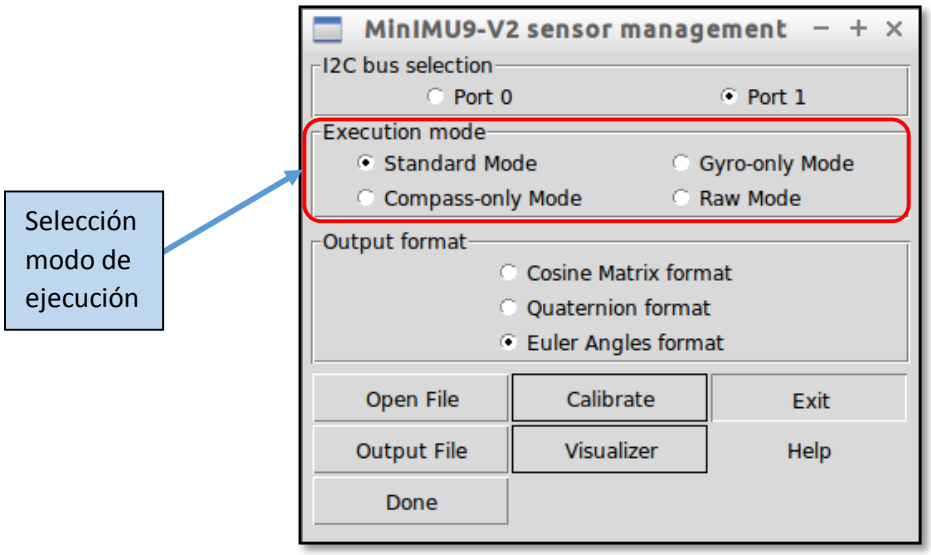
La programación de esta interfaz se puede ver detalladamente desde la Imagen 12 a la Imagen 19 del Anexo I. Se va a detallar que realiza cada una de estas partes del código programado.

En la Imagen 12 se ve como creamos el objeto que será la ventana principal de la interfaz y de la que colgaran el resto de *LabelFrame*. Del *LabelFrame* colgaran los *widgets* (Como los botones de selección). En el bloque 1 creamos el objeto al que llamamos *root* y además definimos el tamaño de dicho *root* con el método *root.maxsize* y *root.minsize.*

En el bloque 2 de la Imagen 12establecemos el tipo de variables que crearemos en los *widgets* que colgarán de los diferentes *LabelFrame*. Para finalizar con el método *root.title* establecemos el título que tendrá el objeto o la ventana de la interfaz programada.

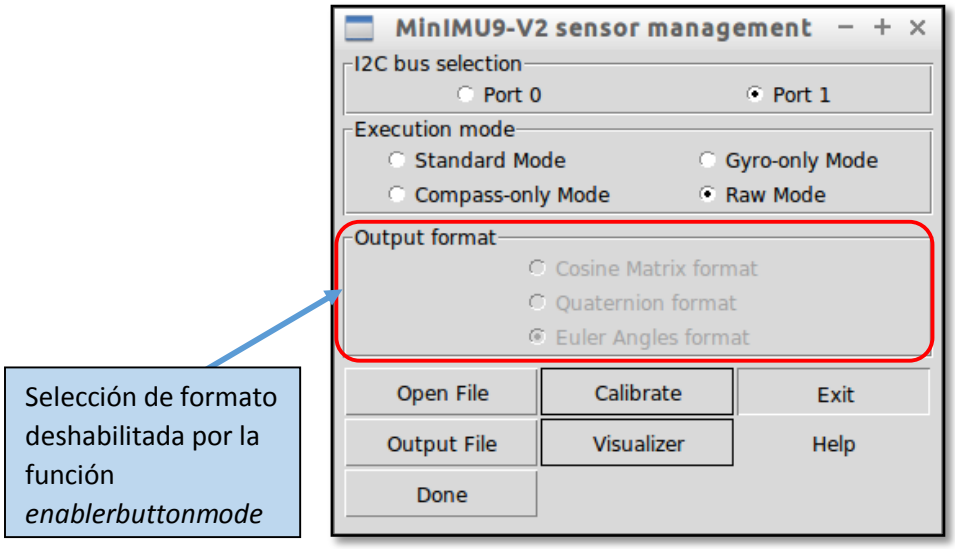
En la Imagen 13 del Anexo I se crea el *LabelFrame i2c_frm* que contendrá las opciones de selección del puerto I2C. Para ello se crean dos *widgets Radiobutton*. Entre las opciones que permite estos widgets, es definir una *variable* que llamaremos i2c y que será de tipo *String* como se ha definido anteriormente en la Imagen 12. Del mismo con la opción *value* se establecerá que valores tomarán las variables dependiendo del *widget Radiobutton* que se seleccione. Para finalizar este *LabelFrame*, lo empaquetamos con el método *.grid* con el cual organizaremos en filas y columnas los *widgets*.

En la Imagen 14 está el código referente al *LabelFrame* con el que elegiremos el modo de ejecución de la IMU. Igual que anteriormente, creamos un nuevo *LabelFrame* llamado *mod_frm*, del cual colgaran los cuatro *widgets Radiobutton*. De las opciones que se pueden elegir definimos una variable para todos los *widgets* que se llamará *mode*. Esta variable podrá tomar cuatro valores diferentes (uno por cada *widget*), gracias a la opción *value*. Los valores que podrá tomar la variable *mode* serán: *Standard Mode, Gyro-only mode, Compass-only mode y Raw mode*. Finalmente, se empaqueta con el método *.grid* y se organiza con filas y columnas. Entre las opciones de estos cuatro widgets tenemos *command*. Esta opción llama a una función cuando seleccionamos el *widget Radiobutton* que en este caso es *disablerbutonmode* y *enblerbuttonmode*.

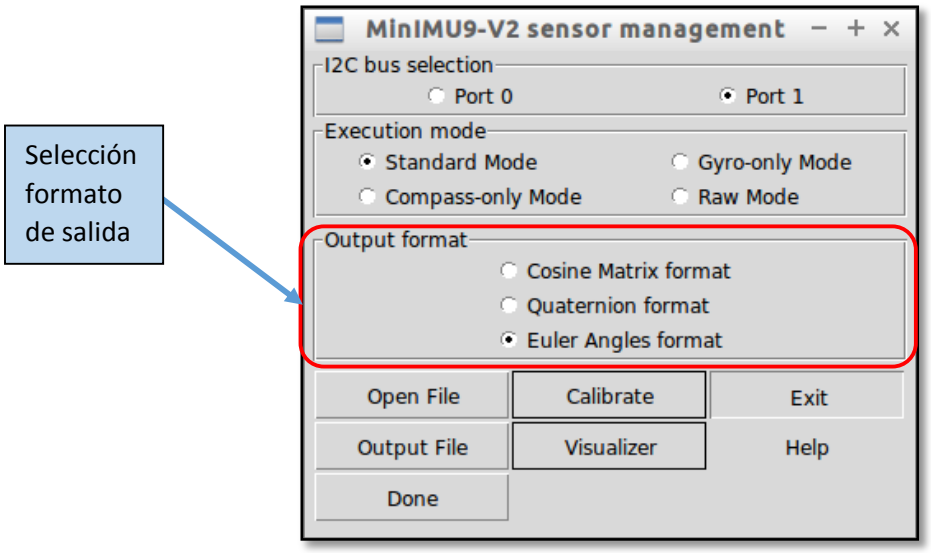


Podemos consultar la función *disablerbutonmode* en la Imagen 10 del Anexo I. Esta función se encarga de deshabilitar los *widgets* del *LabelFrame* de selección de formato cuando elegimos como modo de lectura el *mode raw*. Esto se implementa porque cuando se inicia en el software *minimu9-ahrs* (el creado para controlar la IMU) en *mode raw*, el formato de salida ya viene definido por defecto(tres vectores XYZ, uno para el magnetómetro, otro para el giroscopio y otro para el acelerómetro), por lo que es inútil que se pretenda seleccionar un formato cualquiera.

El código de la función *enablerbuttonmode* lo podemos ver en la Imagen 11 del Anexo I y cuya utilidad es habilitar de nuevo los *widgets* del *LabelFrame* de selección de formato cuando seleccionamos otro modo de ejecución que no sea el *mode raw*.



En la Imagen 15 del Anexo I se tiene el *LabelFrame formato_frm* para implementar el formato de salida de los datos capturados con la IMU. Del mismo modo que en anteriores ocasiones, gracias a las opciones que nos permiten los widgets, creamos una variable llamada *form* con la opción *variables* que será de tipo *String*. Aunque no se ha dicho anteriormente es importante que estas variables de tipo *String*, pues posteriormente se concatenarán todas las diferentes variables para construir el comando que se lanzará desde la consola de comandos. Con la opción *value* definimos los valores que estas variables pueden tomar y que en este caso serán: *matrix*, *quaternion* y *Euler*. Para finalizar, empaquetamos todo con el método *.grid*.

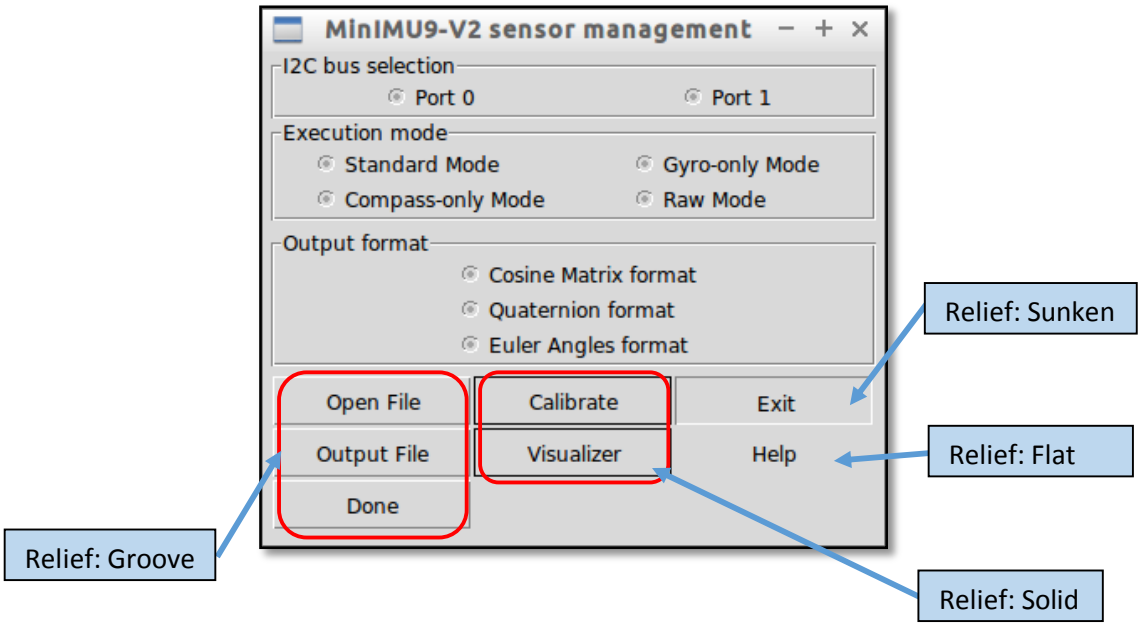


En la Imagen 16 del Anexo I se ve el código con el que empaquetamos todos los *LabelFrame*, pero en esta ocasión utilizamos el método *.pack*. Esta forma de hacerlo, en la que primero se empaqueta con el método *.grid* y posteriormente con el método *.pack* es una costumbre de programación en Python. Con el método *.pack* situamos los diferentes *LabelFrame* creados en la ventana padre (en este caso lo hemos llamado *root*) y lo organizará dentro de ella en el orden en el que aparece en el código. De las opciones que tenemos en el método .pack, se han modificado:

- Fill: indica que el widget tiene que llenar todo el espacio asignado. Al indicarle la opción *both* se le está diciendo que tiene que rellenar el espacio tanto vertical como horizontalmente.
- Expand: se le indicaría si se quiere expandir el espacio del widget. En este caso se le ha indicado que no.
- Padx, Pady: esta opción permite ajustar el espacio de los widgets respecto al margen. Esto es importante cuando queremos que el aspecto del interfaz sea lo más regular posible.

En la Imagen 17 del Anexo I se puede ver todo el código escrito para la creación de los botones del interfaz gráfico. Se puede ver como todos los botones están contenidos por *Frame botón_frm* el cual cuelga del padre *root*, como se especifica en la primera línea de código de dicha imagen. Todos los botones tienen las mismas opciones implementadas, las cuales vamos a ver a continuación:

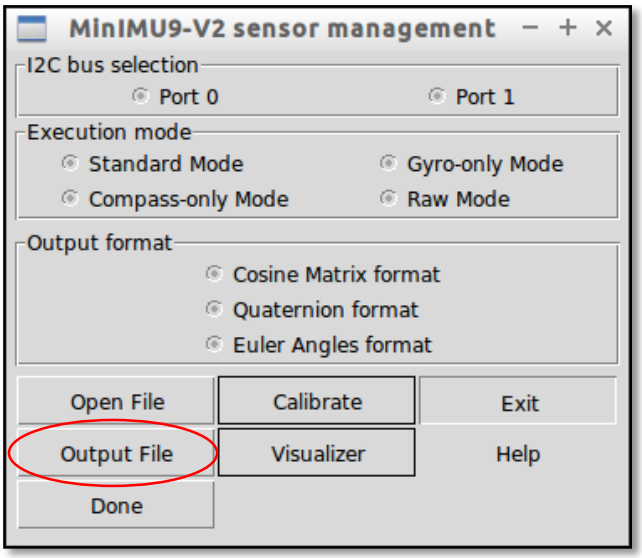
- Text: se indica que texto aparecerá en el botón y dependerá de que función realice cada botón. Por ejemplo, el botón para abrir un archivo se llamará *Open File*.
- Width: se especificará que ancho debe tener el botón. En este caso todos miden lo mismo (13 pixeles) salvo el botón de ayuda, que se le ha dado una anchura un poco menor (8 pixeles).
- Padx, Pady: Será el espacio que se asignará entre el texto del botón y el borde del mismo. En este caso para todos es el mismo, 5 pixeles, tanto en sentido horizontal como el vertical.
- Relief: Con esta opción se determina la decoración del borde del botón. Por defecto la opción es *Flat* (botón ayuda), aunque otras posibles opciones son *Sunken* (botón salir), *Raisedm*, *Solid* (botones calibrar y visual), *Groove* (botones archivo, abrir y hecho) y *Ridge*.
- Command: Con esta opción se está indicando al programa que comando o función debe llamar cuando se presiones el botón en cuestión. Las diferentes funciones que llaman los botones se verán posteriormente.



Para ver como empaquetamos todos los botones que hemos creado, nos remitimos a la Imagen 18 del Anexo I, en la que se puede observar como de nuevo, utilizamos dos métodos de empaquetamiento. El primer es con el método *.grid*, con el que se organiza toda la botonería dentro del *LabelFrame botón_frm*. Para gestionar la organización nos servimos de las filas y columnas para dar el aspecto final que queremos. Finalmente con el método *.pack* lo añadimos a la ventana padre root.

6.5. Apertura de archivos

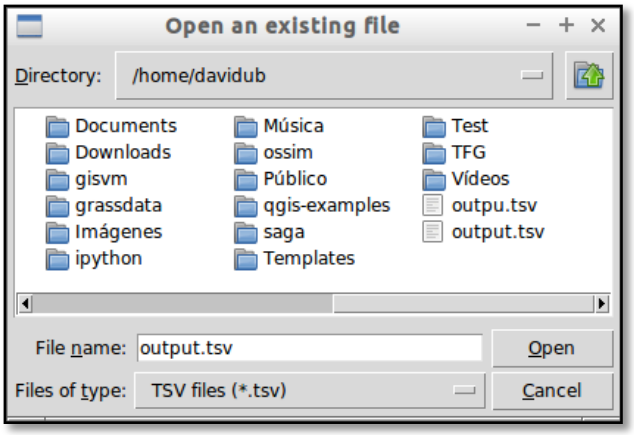
En la línea de hacer este software muy interactuable, se ha implementado la posibilidad de abrir diferentes archivos desde la propia interfaz. Para ver el desarrollo que ha implicado podemos ver la Imagen 2 del Anexo I.



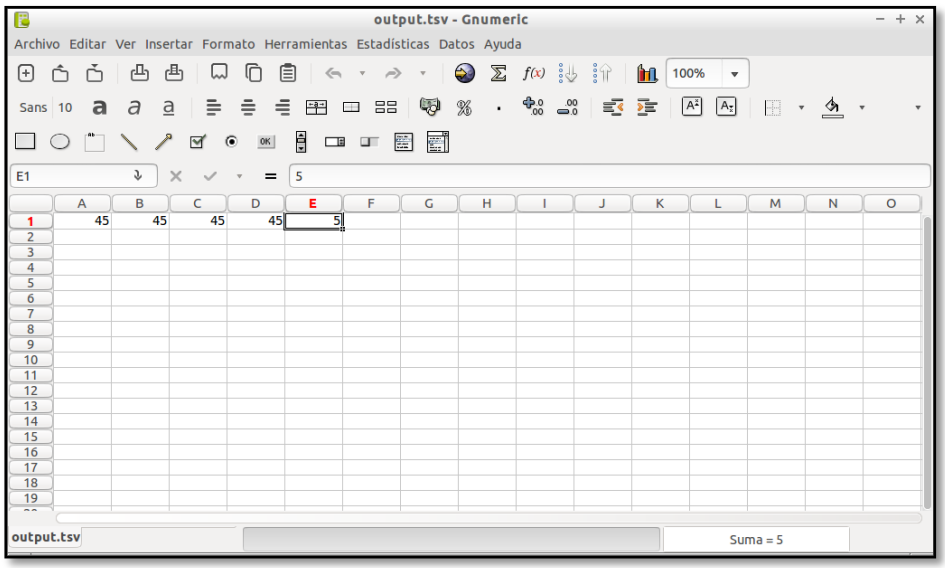
En la imagen 2 se ve la función implementada. En Python, las funciones se definen con la palaba clave *def*. Cuando presionamos el botón *Open File* se llama a la función *def open_file* que será la que nos abrirá el archivo seleccionado que elegiremos mediante la ventana de navegación.

En el bloque 1 de la Imagen 2 definimos las opciones con las que se desea que cuente la ventana de navegación. Definimos el título de la ventana, los formatos de los archivos, y el directorio donde se abrirá por defecto la ventana que cuando se ejecute en la *Raspberry* será en /home/pi. Las opciones que hemos utilizado para definir esto y que nos permite la función de Tkinter *askopenfilename* son:

- Filetypes: con esta opción se especifica el tipo de archivo que queremos que se pueda abrir. Para la cuestión que nos interesa, se ha decidido emplear únicamente dos formatos, tsv (un formato binario) y csv (formato delimitado por comas). Estos dos formatos que se pueden abrir desde el programa, serán los mismos en los que podremos exportar las lecturas que realicemos desde la IMU.
- Initialdir: gracias a esta opción de configuración, seleccionamos en que directorio de nuestro sistema de archivos, queremos que se abra la ventana cuando presionemos el botón *Open File.*
- Parent: indicaremos de que ventana padre colgará el pop-up para abrir archivos.
- Title: Definiremos, que título va a tener la ventana que se implementado.

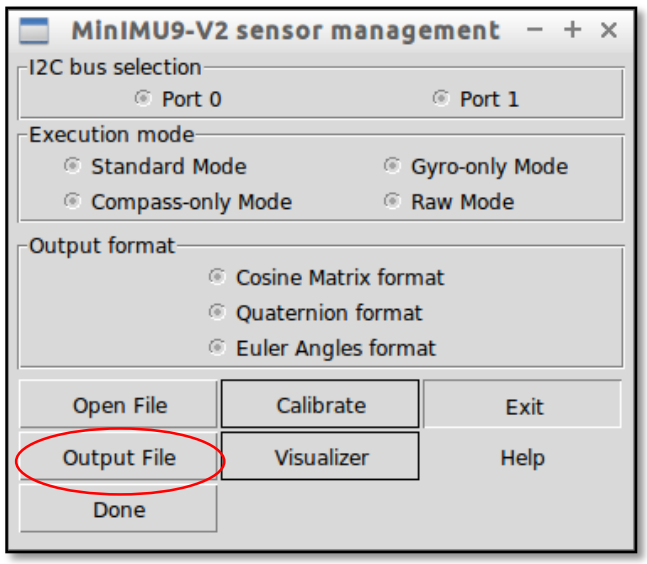


Aquí se ve un ejemplo de un archivo abierto con gnumeric, en el que se ve una serie de datos de prueba que se han tomado.

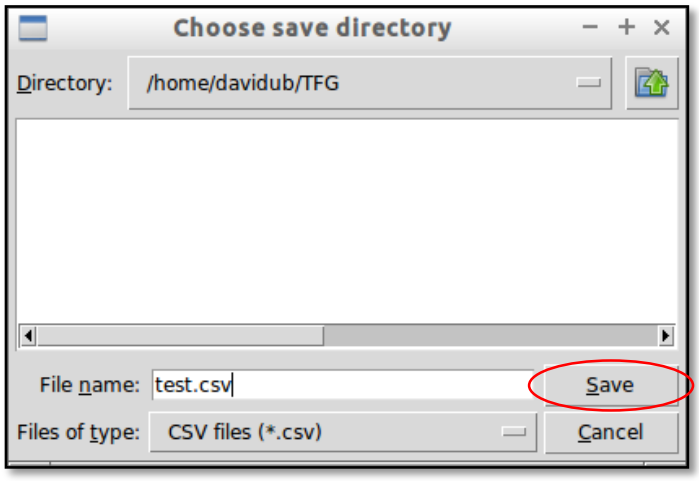


6.6. Lecturas de la IMU en un archivo de salida

Esta opción se inicia cuando presionamos el botón *Output File*, pero únicamente, si antes hemos seleccionado todas las opciones que el programa nos ofrece y que son necesarias para su ejecución, como el puerto I2C sobre el que va a trabajar, el modo de ejecución y el formato de salida de las lecturas realizadas. El código implementado para ello lo podemos ver en detalle en la Imagen 3 del Anexo I.

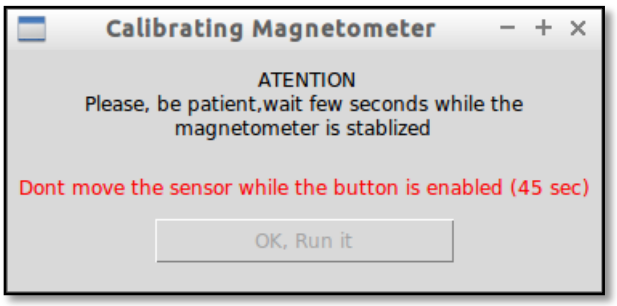


Las opciones de la ventana de dialogo de para seleccionar el directorio, nombre de archivo y formato de guardado, son las mismas que hemos expuesto anteriormente para la ventana de dialogo de la apertura de fichero y que podemos ver en el Bloque 1 de la Imagen 3.



Cuando presionemos el botón *Save* el programa tomará todos los parámetros que hemos seleccionado y lanzará un aviso que detallaremos posteriormente. Antes de iniciarse la lectura, como no se desea que el software pueda ser manipulado, deshabilitaremos todos los botones de la interfaz. Ello lo podemos ver en el Bloque 2 de la Imagen 3. Para deshabilitarlos llamamos al método *.configure* y modificaremos la opción *state* en *disabled*, para así dejar los botones deshabilitados y sin poder realizar ninguna función.

Volviendo al aviso que lanzamos cuando presionamos el botón *Save*, el código implementado para esta nueva ventana, lo tenemos en el Bloque 3 de la Imagen 3. En él se puede identificar todas las opciones que anteriormente se ha especificado, como el título de la ventana, el tamaño de la ventana, el tamaño de los *widgets* y el estado de los botones.



No obstante, si existen algunas novedades, como la opción de configuración del cursor. Lo que se ha querido con la opción *cursor* es que mientras el aviso esté activo y el sensor esté estabilizándose, el aspecto del cursor del ratón pase a un modo de carga.
Pero, ¿porque se decide lanzar este aviso?
Este aviso, básicamente viene a decir que antes de empezar las lecturas los sensores se deben estabilizar. No debemos olvidar que estos sensores son muy sensibles a los movimientos, por lo que para que las lecturas que se obtengan no estén alteradas desde el principio de la toma de datos, es recomendable dejar en reposo unos segundos (45 segundos) el sensor antes de registrar los datos. Esta funcionalidad se ha implementado con el método *.after* de Python, que resulta ser de una gran utilidad si se desea controlar el lanzamiento de ciertas funciones o utilidades dependiendo de la variable tiempo y del recorrido del propio script. Se puede ver, que pasado el tiempo especificado por el método *.after* se llama a la función *inicioimu_enablebutton* que vamos a ver seguidamente.

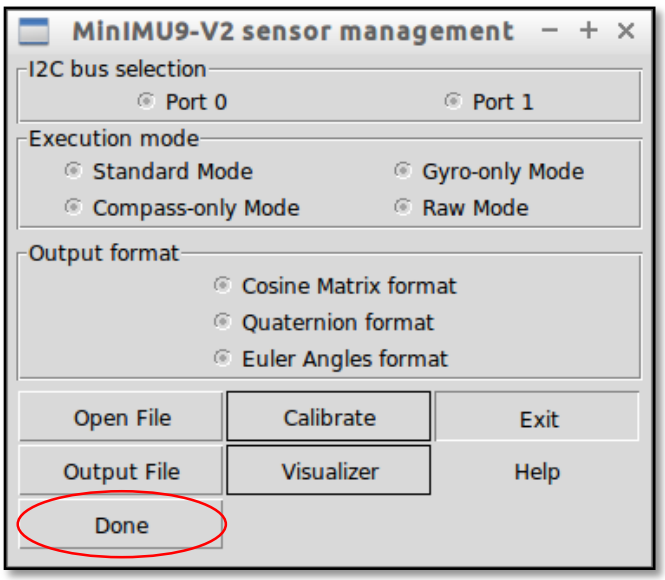
En la Imagen 5 del Anexo I vemos el código de la función *inicioimu_enablebutton*. Esta función tiene la tarea de habilitar de nuevo el botón del aviso anterior. Cuando este botón se habilite, se podrá interactuar con él, de modo que al presionarlo se cerrará el aviso y lanzará los comandos de arranque. Estos comandos resultan de la concatenación de las variables que hemos ido seleccionando en los diferentes widgets anteriores. Para lanzar el comando a la consola del sistema, utilizamos el módulo *os.system* de Linux. Se puede observar que esta función tiene un *if...else* y tiene una importante función. Como hemos dicho anteriormente, si seleccionamos el modo *raw*, los widgets de selección de formato se deshabilitan, porque no tenemos que elegir ningún formato de salida. Por ello, para evitar errores de ejecución, se concatenan dos comandos diferentes, uno específico para cuando lo ejecutamos en modo *raw*, y otro para el resto de modos, que sí que tienen la selección del formato de salida.

Una vez que se ha lanzado el comando, y se habilita el botón de la ventana de aviso, se llama a la función *enble_gui*, que es llamada con la opción *command* que nos permite el botón y que se puede ver en la Imagen 6 del Anexo I. Esta función tiene que habilitar de nuevo todos los botones de la interfaz gráfica, así como destruir el *popup* de aviso implementado.

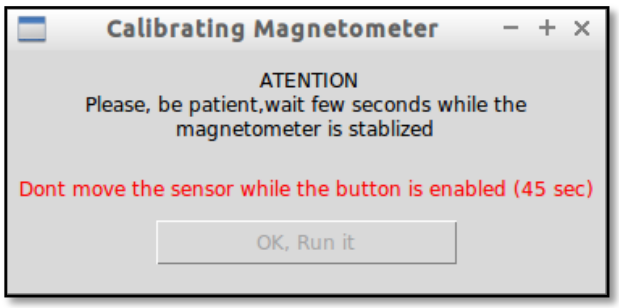
6.7. Lecturas de la IMU desde la línea de comandos

Como no siempre se querrá que la lectura de la IMU se obtenga en un archivo de salida, se he implementado la opción de que las lecturas de la IMU se puedan ver desde la línea de comandos. Esto resulta útil si solo queremos observar y estudiar el comportamiento del sensor en tiempo de ejecución ante diferentes situaciones que deseemos simular con el sensor.

Para ver con detalle este caso, se puede ver la Imagen 4 de Anexo I, en la que se observa, como en el casi anterior, deshabilitamos los botones de la interfaz. Ello queda reflejado en el Bloque I de dicha imagen.



Del mismo modo también se tiene un mensaje de aviso que nos alerta para que dejemos el sensor en estado de reposo durante 45 segundos antes de empezar la lectura y mostrarla por la ventana de comandos del sistema operativo. Esta implementación podemos verla en el Bloque II de la Imagen 4.

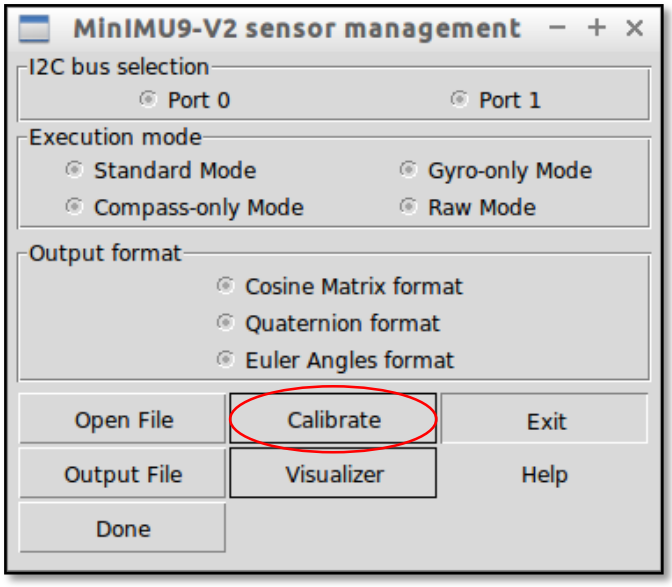


Esta ventana de aviso tiene exactamente la misma implementación que cuando se elige obtener las lecturas de la IMU en un archivo como se ha visto anteriormente.

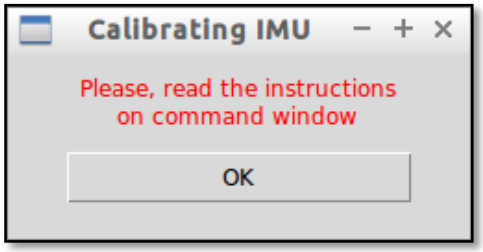
De hecho, esta funcionalidad implementada es exactamente la misma que la vista anteriormente, salvo por el hecho, de que no tenemos un archivo de salida, solo vemos las lecturas en la ventana de comandos.

6.8. Calibración de la IMU

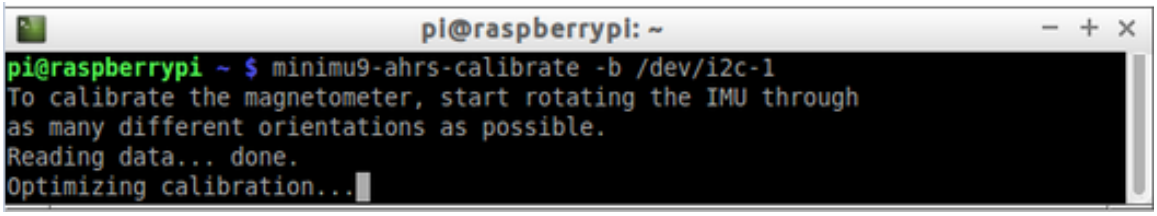
Desde la interfaz que se ha realizado, se ha añadido la posibilidad de calibrar la IMU. Esto resulta importante para obtener unas lecturas de calidad y que los cálculos que realicemos posteriormente con ellos no resulten erróneos ni distorsionados. El porqué de la calibración, ya se ha explicado anteriormente y na va a ser objeto de estudio en este apartado, pues se pretende analizar aquí el código que hay detrás de la funcionalidad implementada.



Si se observa la Imagen 7 del Anexo I, tenemos el código escrito. Se ve que de nuevo se ha querido confeccionar un pequeño popup que avise de las instrucciones que debemos seguir para la correcta calibración del sensor.



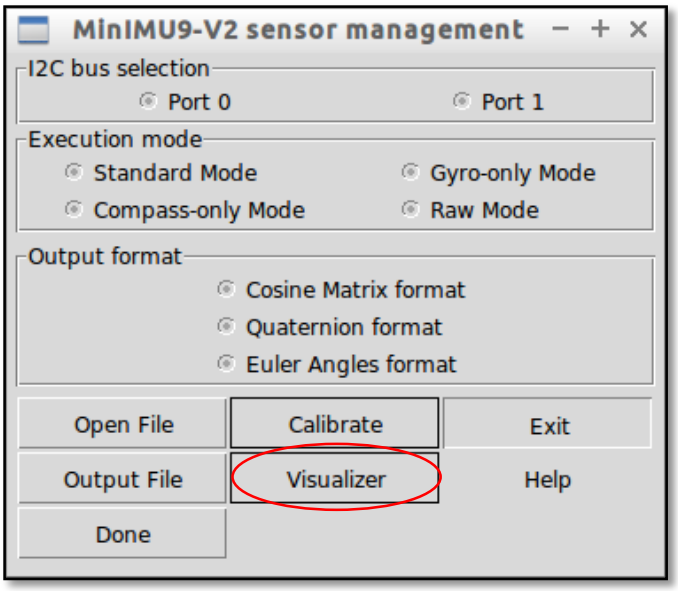
La tarea de esta función programada es la de mostrar el aviso, destruir la ventana cuando presionemos el botón OK, y posteriormente lanzar un comando de consola para que inicie la calibración. Llegado este punto debemos prestar atención a la ventana de comando del sistema operativo, que será la que mostrará las acciones que debemos realizar para asegurar la calibración del sensor.



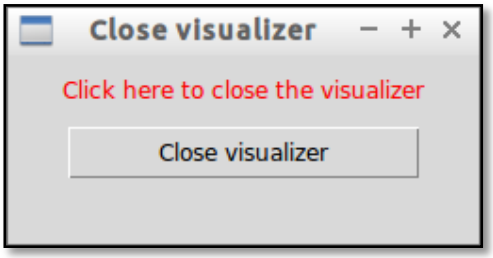
Como vemos desde la ventana de comandos nos ira diciendo el software minimu9-ahrs-calibrate que debemos hacer. Primero solicitará que el sensor se mueva en diferentes direcciones al tiempo que lee datos que se generan. Cuando aparezca *done* nos indicará que la lectura de datos ha concluido y deberemos dejar el sensor en una posición de reposo mientras optimiza la calibración. La ventaja de hacer esto desde una interfaz es la sencillez con la que podemos realizar tareas que a priori son más complejas.

6.9. Visualizador 3D de la IMUMinIMU9-v2

Una utilidad muy interesante y vistosa es la posibilidad de iniciar un visualizador 3D incluido en el software *minimu9-ahrs*. En este visualizador se puede ver en pantalla un modelo del sensor que se mueve del mismo modo que el sensor físico que está alojado en la *protoboard* que se ha utilizado para el montaje del sensor, y que según se cambia de posición se observa el mismo comportamiento del modelo.

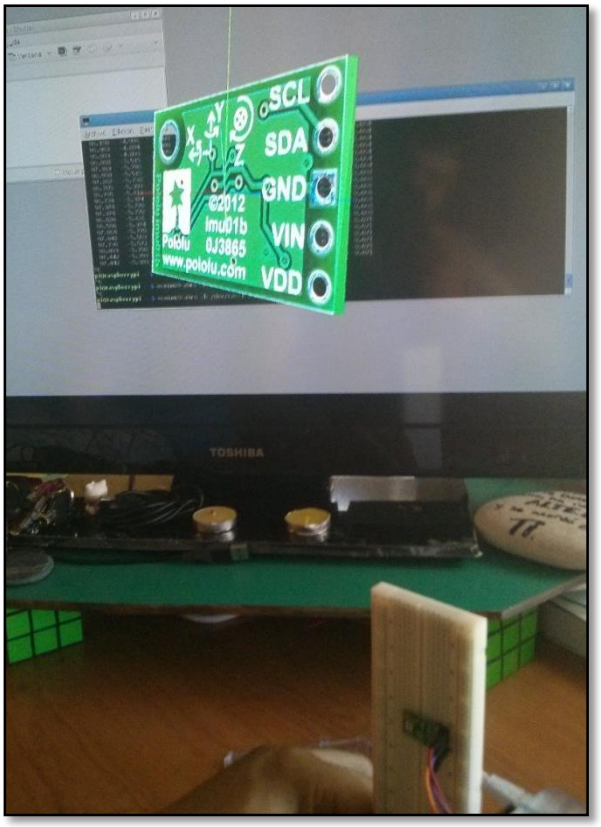


El código que se ha desarrollado para ellos se puede analizar en la Imagen 7 del Anexo I. Se ve cómo se construye el comando que es lanzado a la consola del sistema operativo con la concatenación de la selección del puerto I2C. Esto es muy importante, pues de lo contrario, nos saldrá un error que no nos permitirá el lanzamiento del visualizador. También se ha optado por la implementación de una ventana *toplevel* que nos permita cerrar el visualizador, pues al ejecutarlo se observó que el único modo de cerrar el visualiador era matando el proceso, por ello, en la búsqueda de hacer este software más sencillo se implementó la ventana que nos permite esa posibilidad. Cuando presionamos el botón *Close visualizer*, llamamos a una nueva función con la opción *command* llamada *closevisualizer*.



Esta función se puede analizar en la Imagen 8 del Anexo I en la que se observa cómo se lanza el comando con la utilidad *os.system* y se le dice que mate todos los procesos relacionados con el visualizador.

El resultado final será el que podemos observar en la imagen mostrada a continuación.



6.10. Manual de ayuda man

Por lo general, en la mayoría de software desarrollado para Linux tienen un incorporado un manual en el que se puede consultar todas las dudas que podamos tener en la utilización del software. Por suerte, el software que se ha utilizado en este proyecto (*minimu9-ahrs*) no ha sido una excepción. Por ello, se añadió un botón de ayuda para poder consultar información referente al software. El código que se implementó para esta tarea, podemos verlo en la Imagen 9 del Anexo I en la que se puede observar cómo se lanza el comando de sistema con la utilidad de Linux *os.system* como se ha hecho en anteriores ocasiones.

7. CONCLUSIÓN

En este proyecto ha quedado de manifiesto la potencialidad que nos permite el poder obtener datos inerciales y de posición del dispositivo con el que se está trabajando. Los sensores IMU de uso general como viene siendo el de Pololu abren la puerta a la investigación y desarrollo de proyectos particulares que hace cinco años no parecían posible. Esto también viene propiciado por la democratización de microordenadores como la Raspberry que permiten desarrollar aplicaciones y proyectos que antes estaban reservados a ámbitos muy específicos o restringidos debido principalmente al altísimo coste que implicaba la adquisición de microcontroladores profesionales. Otro factor muy importante, es la enorme comunidad de desarrolladores y amantes de la programación, que se preocupan por publicar todo lo que van desarrollando e investigando, además de la inestimable ayuda que significan los blogs especializados, en los que a los minutos se responde a cualquier duda planteada.

Con este proyecto, se plantea un interesantísimo punto de partida para la consecución de nuevos proyectos o ampliaciones de este. De hecho, el siguiente paso lógico que, a nivel personal mi curiosidad me pide a gritos, es la inclusión de un GPS, que además nos permita posicionar nuestras lecturas de la IMU. Teniendo estos dos elementos unidos, el límite lo pone la imaginación. De hecho, podría ser una gran solución al tan conocido y ansiado posicionamiento *indoor*. O podría ser la solución a los problemas de la navegación, con todos estos datos, se podrían diseñar sin problemas algoritmos que nos sirviesen correcciones a la navegación. El mundo de la fotogrametría aérea, o terrestre tiene en estos sensores sus mayores aliados. Quien no ha querido alguna vez haber podido acceder a los datos de la IMU del vuelo fotogramétrico para rectificar sus imágenes. Y así, se podría enumerar una cantidad de aplicaciones hasta uno cansarse. Pero como todo, este proyecto tiene importantes limitaciones, las cuales se han intentado solucionar del modo más eficiente posible con los recursos, tanto personales como materiales, de los que se disponían.

La principal limitación del proyecto viene de su propio planteamiento. Se ha diseñado un software escrito en Python que gestione el software de la IMU, que viene escrito en C++ y que además necesita interactuar con la línea de comando del sistema operativo Linux. Para cualquiera que tenga un mínimo conocimiento en informática ya sabe la envergadura del problema al que este proyecto se ha enfrentado. De modo que, se centraron los esfuerzos en encontrar el procedimiento que hiciese funcionar la interfaz gráfica como se deseaba. Y en este apartado, el resultado no ha podido ser más satisfactorio, pues aunque no se ha conseguido realizar como se había pensado desde un principio, la solución no ha resultado ser parche mal puesto. La idea inicial, era desde el IDLE de Python lanzar el script y que las funcionalidades implementadas se fuesen ejecutando. Esto no funcionó, y el motivo era muy sencillo. Nuestro script necesitaba interactuar con la ventana de comandos del sistema operativo, por lo que si queríamos que funcionase, debíamos llamar al script desde la línea de comandos. Como vemos, esto no suena como un problema serio, pero si planteo alguna dificultad, debiendo modificar secciones de código, o incluso, obligando a implementar código que en un principio no se tenía previsto.

Conociendo este problema, se ha de plantear la siguiente cuestión: ¿Qué se puede hacer para evitar que esto suceda?
Pues bien, la respuesta es sencilla. O bien, diseñar una interfaz gráfica con lenguaje C++ e introducir el código del software *minimu9-ahrs*, o bien, partiendo del software desarrollado en el proyecto (gestionIMU) reescribir todo el código del software *minimu9-ahrs* en Python. Otra opción intermedia sería incrustar las secciones del código C++ en el código de Python, pero esto plantearía una serie de problemas, que resultaría complicado que funcionase correctamente.
Con esto y todo, y tras mi experiencia personal, la solución que elegiría sería la de optar por programar una interfaz gráfica en C++. El porqué de esta decisión es muy sencillo. Es sabido, los

problemas que tiene Python en tanto en cuanto a la velocidad de ejecución, y aunque se podría pensar que este no es un problema que pueda afectarnos, se ha visto en las múltiples pruebas hechas con la IMU, que si se sufría de un ralentizamiento pronunciado de la velocidad de ejecución del software y de fluidez de funcionamiento de la Raspberry B+.

8. REFERENCIAS BIBLIOGRÁFICAS

- GONZÁLES DUQUE, R. (2012). Python para todos.
- KING A. D. (1998). "Inertial Navigation – Forty Years of Evolution", GEC REVIEW, VOL. 13, número 3, p. 1-6.
- Complex Filtering with the HSP43168 Dual FIR Filter. <http://www.intersil.com/content/dam/Intersil/documents/an94/an9418.pdf> [Consulta: de junio de 2015] 18
- ANGEL ÁLVAREZ, M. (2003) "Qué es Python" en desarrolladoresweb.com [Consulta:28 de mayo de 2015]
- UNIVERSIDAD TÉCNICA DE FEDERICO SANTA MARÍA. Interfaces gráficas. [Consulta: 13 de junio de 2015]
- SHIPMAN W. J. (2013). Tkinter 8.5 reference: aGUI for Python. <http://infohost.nmt.edu/tcc/help/pubs/tkinter/tkinter.pdf> [Consulta:22 de junio de 2015]
- RASPBERRY PI COMMUNITY. <https://www.raspberrypi.org/community/> [Consulta: 12 de junio de 2015]
- SMIRNOVA L. (2014) "Raspberry Pi y la democratización de la informática." en ibertronica.es [Consulta: 11 de junio de 2015]
- GitHub of DavidEGrayson <https://github.com/DavidEGrayson/minimu9-ahrs/wiki> [Consulta: 19 de junio de 2015]
- GRAYSON D. (2012) "Orientation sensing with the Raspberry Pi and MinIMU-9 v2" en David Grayson's blog 26 de noviembre <http://blog.davidegrayson.com/2012/11/orientation-sensing-with-raspberry-pi.html> [Consulta: 19 de junio 2013]
- Guide to interfacing a Gyro and Accelerometer with a Raspberry Pi. <http://ozzmaker.com/2013/04/29/guide-to-interfacing-a-gyro-and-accelerometer-with-a-raspberry-pi/> [Consulta: 14 de junio 2015]
- Python. <https://wiki.python.org/moin/TkInter> [Consulta: 23 de junio 2015]
- An Introduction to Tkinter (Work in Progress). <http://effbot.org/tkinterbook/> [Consulta: 23 de junio 2015]

Anexo I – Código fuente GestiónIMU

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import Tkinter
import tkFileDialog
import time
import os
```

Imagen 1: Importación de los módulos a utilizar

```
def open_file():
    options = {}
    options['filetypes'] = []
    options['filetypes'].append(('TSV files', '.tsv'))
    options['filetypes'].append(('CSV files', '.csv'))
    options['initialdir'] = '/home/pi'
    options['parent'] = root
    options['title'] = 'Open an existing file'

    openfile = tkFileDialog.askopenfilename(**options)

    if openfile:
        os.system('gnumeric ' + openfile)
        #Para version raspberry
        #os.system('soffice ' + openfile)
```

Bloque 1

Imagen 2: Función para abrir archivos

```
info_widget = """ ATENTION
Please, be patient,wait few seconds while the
magnetometer is stablized"""

#FUNCIONALIDAD CUANDO DESEAMOS GUARDAR LA LECTURA EN UN ARCHIVO
def output_file():
    options = {}
    options['defaultextension'] = '.csv'
    options['filetypes'] = []
    options['filetypes'].append(('CSV files', '.csv'))
    options['filetypes'].append(('TSV files', '.tsv'))
    options['title'] = 'Choose save directory'

#Seleccionamos la ruta donde guardaremos el archivo
    global outputfile
    outputfile = tkFileDialog.asksaveasfilename(**options)

#Deshabilitamos todos los botones
    btn_abrir.configure (state = 'disabled')
    btn_archivo.configure (state = 'disabled')
    btn_hecho.configure (state = 'disabled')
    btn_calibrar.configure (state = 'disabled')
    btn_visual.configure (state = 'disabled')
    btn_salir.configure (state = 'disabled')
    btn_ayuda.configure (state = 'disabled')
```

Bloque 1

Bloque 2

```
#Mostramos ventana de aviso
    global toplevel
    toplevel = Tkinter.Toplevel()
    toplevel.title('Calibrating Magnetometer')
    toplevel.minsize(350,150)
    toplevel.maxsize(350,150)
    toplevel.config(cursor= 'watch')
    label1 = Tkinter.Label(toplevel, justify = 'center',pady = 10\

                       , text=info_widget,width =40)
    label1.pack()

    label2 = Tkinter.Label(toplevel,fg = 'red', justify = 'center',pady = 10\
    , text="Dont move the sensor while the button is enabled (45 sec)", width =40)
    label2.pack()
    global buttonOK
    buttonOK = Tkinter.Button(toplevel, text= "OK, Run it" \
                            , width=20,state = 'disabled', command = enable_gui)
    buttonOK.pack()

    toplevel.after(45000,inicioimu_enablebutton)
```

Bloque 3

Imagen 3: Inicio de la IMU obteniendo un archivo de salida

```
#FUNCIONALIDAD CUANDO DESEAMOS VER LA LECTURA EN LA VENTANA DE COMANDOS
def inicioimu_disable():
#Deshabilitamos todos los botones
    btn_abrir.configure (state = 'disabled')
    btn_archivo.configure (state = 'disabled')
    btn_hecho.configure (state = 'disabled')
    btn_calibrar.configure (state = 'disabled')
    btn_visual.configure (state = 'disabled')
    btn_salir.configure (state = 'disabled')
    btn_ayuda.configure (state = 'disabled')
```

Bloque 1

```
#Mostramos ventana de aviso
    global toplevel
    toplevel = Tkinter.Toplevel()
    toplevel.title('Calibrating Magnetometer')
    toplevel.minsize(350,150)
    toplevel.maxsize(350,150)
    toplevel.config(cursor= 'watch')
    label1 = Tkinter.Label(toplevel, justify = 'center',pady = 10\
                           , text=info_widget,width =40)
    label1.pack()

    label2 = Tkinter.Label(toplevel,fg = 'red', justify = 'center',pady = 10\
    , text= "Dont move the sensor while the button is enabled (45 sec)" , width =40)
    label2.pack()
    global buttonOK
    buttonOK = Tkinter.Button(toplevel, text= "OK, Run it" \
                            , width=20,state = 'disabled', command = enable_gui)
    buttonOK.pack()

    toplevel.after(45000,inicioimu_enablebutton)
```

Bloque 2

Imagen 4: Inicio de la IMU sin obtener un archivo de salida

```
def inicioimu_enablebutton():

    buttonOK.configure(state = 'normal')
#Lanzamos comando de arranque
    if str(mode.get()) == 'raw':
        os.system( 'minimu9-ahrs -b /dev/'\
                    + str(i2c.get()) \
                    + ' -- mode ' + str(mode.get()) \
                    + ' > ' + outputfile)
    else:
        os.system( 'minimu9-ahrs -b /dev/'\
                    + str(i2c.get()) \
                    + ' -- mode ' + str(mode.get()) \
                    + ' -- output ' + str(form.get()) \
                    + ' > ' + outputfile)
```

Imagen 5: Función para habilitar el botón e iniciar lectura de la IMU

```
def enable_gui():
    btn_abrir.configure (state = 'normal')
    btn_archivo.configure (state = 'normal')
    btn_hecho.configure (state = 'normal')
    btn_calibrar.configure (state = 'normal')
    btn_visual.configure (state = 'normal')
    btn_salir.configure (state = 'normal')
    btn_ayuda.configure (state = 'normal')
    toplevel.destroy()
```

Imagen 6: Función para habilitar los botones del interfaz de gestión

```
def calibrate():

#GUI para seguir instrucciones calibracion de la IMU
    lbl_message = """Please, read the instructions
on command window"""
    topcalib = Tkinter.Toplevel()
    topcalib.title('Calibrating IMU')
    topcalib.minsize(250,100)
    topcalib.maxsize(250,100)
    topcalib.config(cursor= 'watch')
    labelcal = Tkinter.Label(topcalib,fg = 'red', justify = 'center',pady = 10\
                          , text=lbl_message, width =40)
    labelcal.pack()
    buttoncal = Tkinter.Button(topcalib, text= "OK" \
                            , width=20, command = topcalib.destroy)
    buttoncal.pack()

#Lanzamos comando de calibracion
    os.system( 'minimu9-ahrs-calibrate -b /dev/' + str(i2c.get()))
```

Imagen 7: Función para iniciar la calibración de la IMU

```
def visualizer():
#Lanzamos comando para el visualizador
    os.system( 'minimu9-ahrs -b /dev/' + str(i2c.get()) + ' | ahrs-visualizer')

#Ventana para cierre del visualizador
    global toplvlvis
    toplvlvis = Tkinter.Toplevel()
    toplvlvis.title('Close visualizer')
    toplvlvis.minsize(250,100)
    toplvlvis.maxsize(250,100)
    labelvis = Tkinter.Label(toplvlvis,fg = 'red', justify = 'center',pady = 10\
                          , text= "Click here to close the visualizer" , width =40)
    labelvis.pack()
    buttonclose = Tkinter.Button(toplvlvis, text= "Close visualizer" \
                            , width=20, command = closevisualizer)
    buttonclose.pack()
```

Imagen 7: Función para iniciar el visualizador 3D del sensor

```
#Lanzamos comando para matar el proceso del visualizador
def closevisualizer():
    os.system('killall ahrs-visualizer')
    toplvlvis.destroy()
```

Imagen 8: Función para matar el proceso del visualizador

```
#Lanzamos comando para abrir la ayuda
def ayuda():
    os.system('man minimu9-ahrs')
```

Imagen 9: Función para llamar al manual de la IMU

```
#Funcion para deshabilitar la eleccion de los formatos cuando elegimos el modo raw
def disablerbutmode():
    formato_Rbuton_matrix.configure(state = 'disabled')

    formato_Rbuton_quaternion.configure(state = 'disabled')

    formato_Rbuton_euler.configure(state = 'disabled')
```

Imagen 10: Función para deshabilitar los botones de selección del formato

```
#Funcion para habilitar la eleccion de los formatos cuando deseleccionamos el modo raw
def enablerbutmode():
    formato_Rbuton_matrix.configure(state = 'normal')

    formato_Rbuton_quaternion.configure(state = 'normal')

    formato_Rbuton_euler.configure(state = 'normal')
```

Imagen 11: Función para habilitar los botones de selección del formato

```
#********************** Creacion de la interfaz gráfica principal **********************#
#Creamos el GUI...
root = Tkinter.Tk()
root.minsize(371,295)
root.maxsize(371,295)

i2c = Tkinter.StringVar()
mode = Tkinter.StringVar()
form = Tkinter.StringVar()

#Titulo
root.title( "MinIMU9-V2 sensor management" )
```

Bloque 1

Bloque 2

Imagen 12: Inicialización del interfaz gráfico y declaración de variables

```
#-------------------------------------------------------------------------------------#
```

```
#Seleccion del bus I2C
i2c_frm = Tkinter.LabelFrame(root, text = "I2C bus selection" , width = '30' )

i2c_Rbuton0 = Tkinter.Radiobutton(i2c_frm, text= "Port 0" , width = 19,\
                        value= "i2c-0" , justify=Tkinter.CENTER, variable = i2c)

i2c_Rbuton1 = Tkinter.Radiobutton(i2c_frm, text= "Port 1" , width = 18,\
                        value= "i2c-1" , justify=Tkinter.CENTER, variable = i2c)
#Empaquetamos...
i2c_Rbuton0.grid(row = 0, column = 0)

i2c_Rbuton1.grid(row = 0, column = 1)
```

Imagen 13: Creación del Labelframe de selección del bus I2C

```
#-------------------------------------------------------------------------------------#
```

```
#Modo de ejecucion
modo_frm = Tkinter.LabelFrame(root, text= "Execution mode" )

modo_Rbuton_normal = Tkinter.Radiobutton(modo_frm, text= "Standard Mode       " ,\
                            width = 19, justify=Tkinter.LEFT, value= "normal" , \
                                    variable = mode, command = enablerbutmode)

modo_Rbuton_gyro = Tkinter.Radiobutton(modo_frm, text= "Gyro-only Mode" , \
                            width = 18, justify=Tkinter.LEFT, value= "gyro" ,\
                                    variable = mode, command = enablerbutmode)

modo_Rbuton_compass = Tkinter.Radiobutton(modo_frm, text= "Compass-only Mode" ,\
                            width = 19, justify=Tkinter.LEFT, value= "compass" ,\
                                    variable = mode, command = enablerbutmode)

modo_Rbuton_raw = Tkinter.Radiobutton(modo_frm, text= "Raw Mode        " ,\
                            width = 18, justify=Tkinter.LEFT, value= "raw" ,\
                                    variable = mode, command = disablerbutmode)

#Empaquetamos...
modo_Rbuton_normal.grid(row = 0, column = 0)

modo_Rbuton_gyro.grid(row = 0, column = 1)

modo_Rbuton_compass.grid(row = 1, column = 0)

modo_Rbuton_raw.grid(row = 1, column = 1)
```

Imagen 14: Creación del Labelframe de selección del modo de ejecución

```
#-------------------------------------------------------------------------------------#
```

```
#Formatos de salida...
formato_frm = Tkinter.LabelFrame(root, text= "Output format" )

formato_Rbuton_matrix = Tkinter.Radiobutton(formato_frm, text= "Cosine Matrix format" ,\
                    width = 41, justify=Tkinter.LEFT, value= "matrix" , variable = form)

formato_Rbuton_quaternion = Tkinter.Radiobutton(formato_frm, text= "Quaternion format   " ,\
                    width = 41, justify=Tkinter.LEFT, value= "quaternion" , variable = form)

formato_Rbuton_euler = Tkinter.Radiobutton(formato_frm, text= "Euler Angles format " ,\
                    width = 41, justify=Tkinter.LEFT, value= "euler" , variable = form)

#Empaquetamos...
formato_Rbuton_matrix.grid(row = 0, column = 0)

formato_Rbuton_quaternion.grid(row = 1, column = 0)

formato_Rbuton_euler.grid(row = 2, column = 0)
```

Imagen 15: Creación del Labelframe de selección del formato de salida

```
#-------------------------------------------------------------------------------------#
```

```
i2c_frm.pack(fill='both', expand='no', padx = 5)
modo_frm.pack(fill='both', expand='no', padx = 5)
formato_frm.pack(fill='both', expand='no', padx = 5,pady=5)
```

Imagen 16: Empaquetado de los Labelframe creados

```
#-------------------------------------------------------------------------------------#
```

```
#Botones...
boton_frm = Tkinter.Frame(root)

btn_abrir = Tkinter.Button(boton_frm, text='Open File', width= 13, \
                        padx = 5, pady = 5, relief = 'groove', command = open_file)

btn_archivo = Tkinter.Button(boton_frm, text='Output File', width= 13, \
                        padx = 5, pady = 5, relief = 'groove', command = output_file)

btn_hecho = Tkinter.Button(boton_frm, text='Done', width = 13, \
                        padx = 5, pady = 5, relief = 'groove', command=inicioimu_disable)

btn_calibrar = Tkinter.Button(boton_frm, text='Calibrate', width = 13, \
                        padx = 5, pady = 5, relief = 'solid', command=calibrate)

btn_visual = Tkinter.Button(boton_frm, text='Visualizer', width = 13, \
                        padx = 5, pady = 5, relief = 'solid', command=visualizer)

btn_salir = Tkinter.Button(boton_frm, text='Exit', width = 13, \
                        command = root.destroy, padx = 5, pady = 5, relief = 'sunken')

btn_ayuda = Tkinter.Button(boton_frm, text='Help', width = 8, \
                            relief = 'flat',command=ayuda)
```

Imagen 17: Creación de los botones de la interfaz

```
btn_abrir.grid(row=0, column=0)
btn_archivo.grid(row=1, column=0)
btn_hecho.grid(row = 2, column=0)
btn_calibrar.grid(row=0, column=2)
btn_visual.grid(row=1, column=2)
btn_salir.grid(row=0, column=3)
btn_ayuda.grid(row=1, column=3)

boton_frm.pack()
```

Imagen 18: Organización y empaquetado de los botones de la interfaz

```
#-------------------------------------------------------------------------------#
```

```
root.mainloop()
```

Imagen 19: Fin de la interfaz gráfica. Inicio del bucle del programa principal

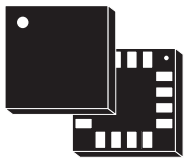
Anexo II - Especificaciones del giroscopio

life.augmented

L3GD20

MEMS motion sensor: three-axis digital output gyroscope

Datasheet - production data



LGA-16 (4x4x1 mm)

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
- 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

Description

The L3GD20 is a low-power three-axis angular rate sensor.

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 L3GD20 has a full scale of ±250/±500/ ±2000 dps and is capable of measuring rates with a user-selectable bandwidth.

The L3GD20 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
L3GD20	-40 to +85	LGA-16 (4x4x1 mm)	Tray
L3GD20TR	-40 to +85	LGA-16 (4x4x1 mm)	Tape and reel

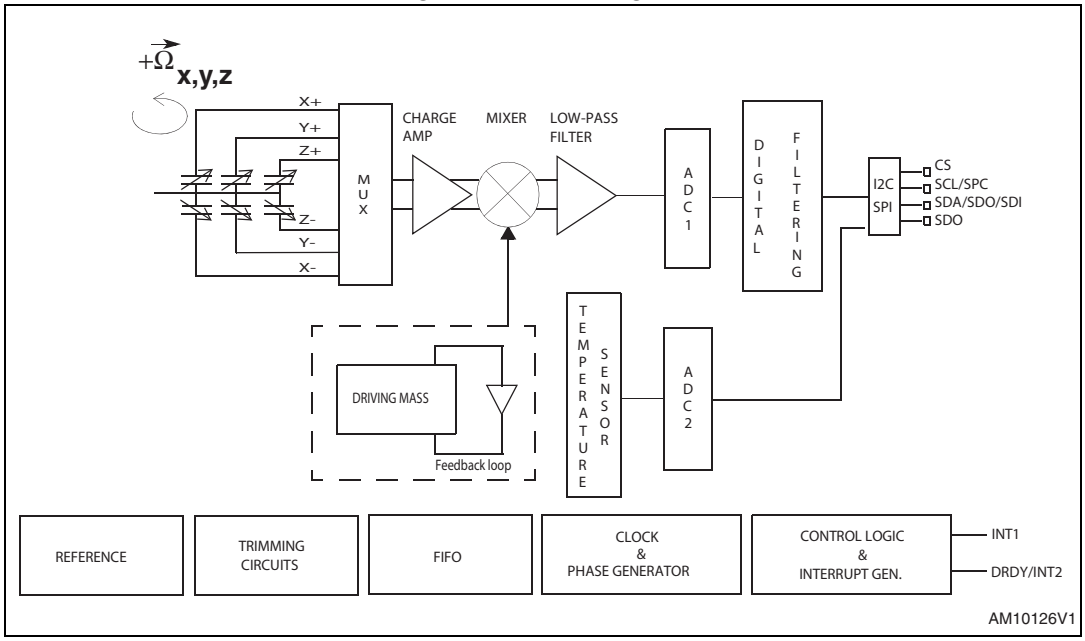
Contents

List of tables

List of figures

1 Block diagram and pin description

Figure 1. Block diagram



Note: 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

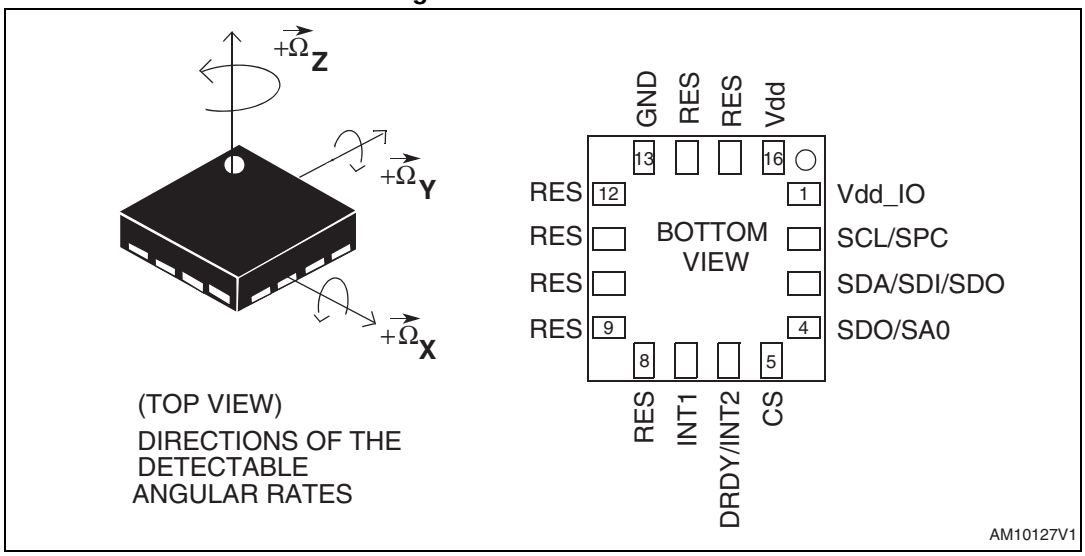


Table 2. Pin description

Table 3.

Pin#	Name	Function
1	Vdd_IO(1)	Power supply for I/O pins
2	SCL SPC	I²C serial clock (SCL) SPI serial port clock (SPC)
3	SDA SDI SDO	I²C serial data (SDA) SPI serial data input (SDI) 3-wire interface serial data output (SDO)
4	SDO SA0	SPI serial data output (SDO) I²C less significant bit of the device address (SA0)
5	CS	I²C/SPI mode selection (1: SPI idle mode / I²C communication enabled; 0: SPI communication mode / I²C disabled)
6	DRDY/INT2	Data ready/FIFO interrupt (Watermark/Overrun/Empty)
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	Reserved	Connect to GND with ceramic capacitor(2)
15	Reserved	Connect to Vdd
16	Vdd(3)	Power supply

1. 100 nF filter capacitor recommended.
2. 1 nF min value must be guaranteed under 11 V bias condition.
3. 100 nF plus 10 μF capacitors recommended.

2 Mechanical and electrical specifications

2.1 Mechanical characteristics

@ Vdd = 3.0 V, T = 25 °C unless otherwise noted.

Table 4. Mechanical characteristics(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	FS = 250 dps		±0.03		dps/°C
		FS = 2000 dps		±0.04		dps/°C
NL	Non linearity	Best fit straight line		0.2		% FS
Rn	Rate noise density			0.03		dps/(√Hz)
ODR	Digital output data rate			95/190/ 380/760		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.

2.2 Electrical characteristics

@ Vdd =3.0 V, T=25 °C unless otherwise noted.

Table 5. Electrical characteristics (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		2		mA
IddPdn	Supply current in power-down mode	Selectable by digital interface		5		µA
VIH	Digital high level input voltage		0.8*Vdd_IO			V
VIL	Digital low level input voltage				0.2*Vdd_IO	V
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 relative to power-down mode.

2.3 Temperature sensor characteristics

@ Vdd =3.0 V, T=25 °C unless otherwise noted.

Table 6. Electrical characteristics (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

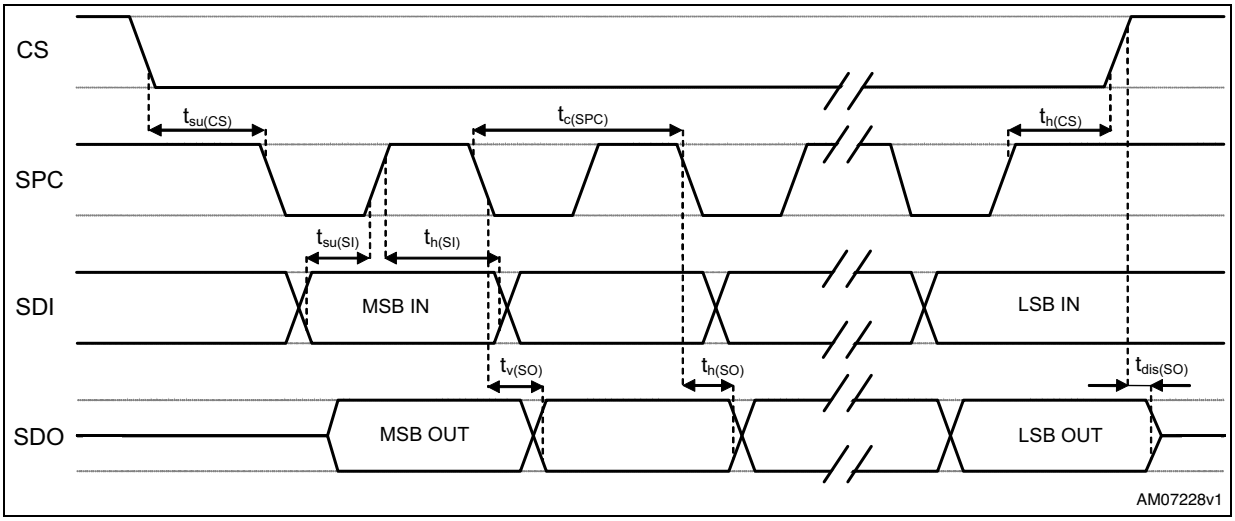
Subject to general operating conditions for Vdd and T_{op}.

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 a 10 MHz clock frequency for SPI with both 4 and 3 wires, based on characterization results; not tested in production.

Figure 3. SPI slave timing diagram (a)



a. Measurement points are at 0.2·Vdd_IO and 0.8·Vdd_IO, for both input and output port.

2.4.2 I²C - Inter IC control interface

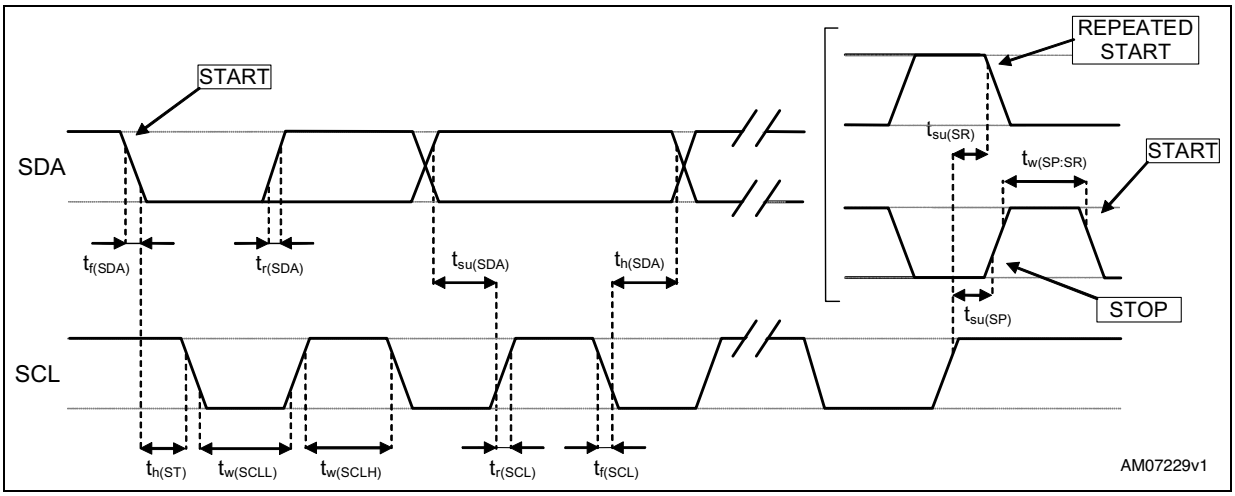
Subject to general operating conditions for Vdd and T_{op}.

Table 8. I²C slave timing values (TBC)

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 4. I²C slave timing diagram (b)



b. Measurement points are at 0.2·Vdd_IO and 0.8·Vdd_IO, for both ports.

2.5 Absolute maximum ratings

Stresses above those 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
		1.5 (CDM)	kV
		200 (MM)	V
Vin	Input voltage on any control pin (CS, SCL/SPC, SDA/SDI/SDO, SDO/SA0)	-0.3 to Vdd_IO +0.3	V

Note: Supply voltage on any pin should never exceed 4.8 V

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 counter-clockwise 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. Zero-rate level of precise MEMS sensors is, to some extent, a result of stress to the sensor and therefore 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.

2.7 Soldering information

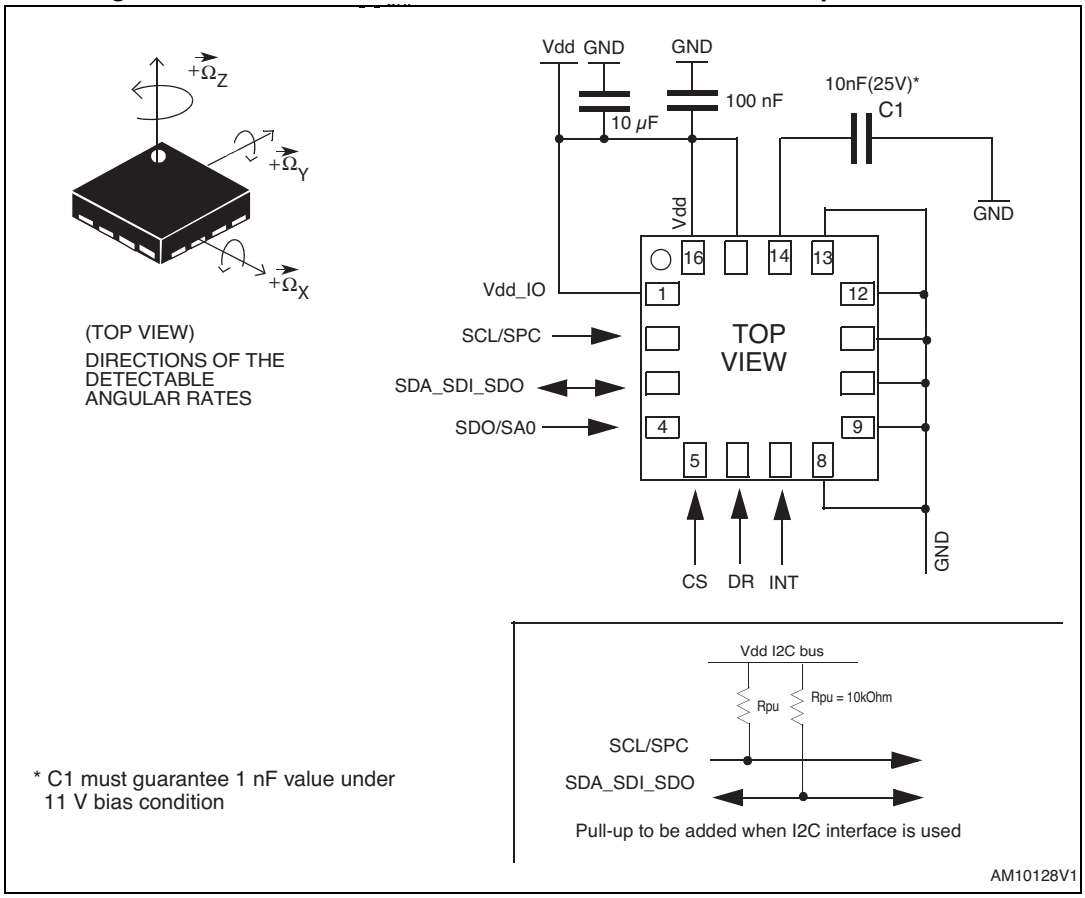
The LGA package is compliant with the ECOPACK®, RoHS and “Green” standard.
It is qualified for soldering heat resistance according to JEDEC J-STD-020.

Leave “Pin 1 Indicator” unconnected during soldering.

Land pattern and soldering recommendations are available at www.st.com/mems.

3 Application hints

Figure 5. L3GD20 electrical connections and external component values



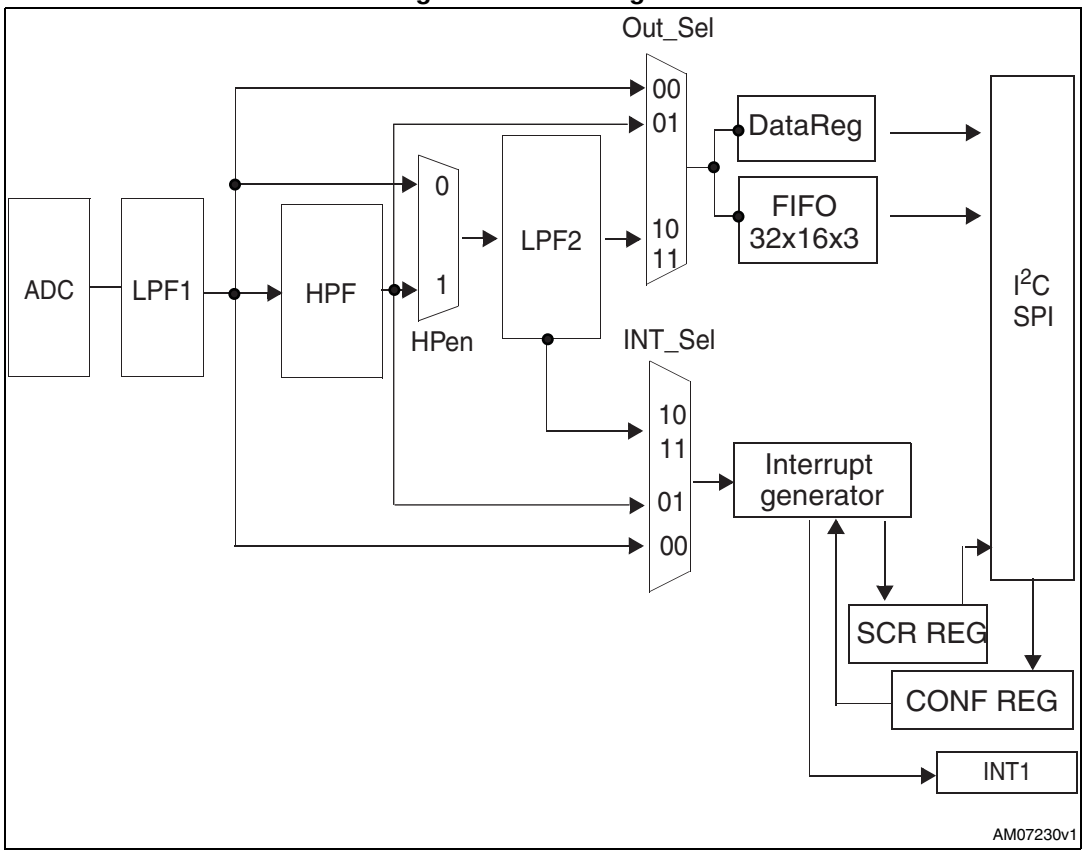
Power supply decoupling capacitors (100 nF + 10 μF) should be placed as near as possible to the device (common design practice).

If Vdd and Vdd_IO are not connected together, 100 nF and 10 μF decoupling capacitors must be placed between Vdd and common ground, and 100 nF between Vdd_IO and common ground. Capacitors should be placed as near as possible to the device (common design practice).

4 Digital main blocks

4.1 Block diagram

Figure 6. Block diagram



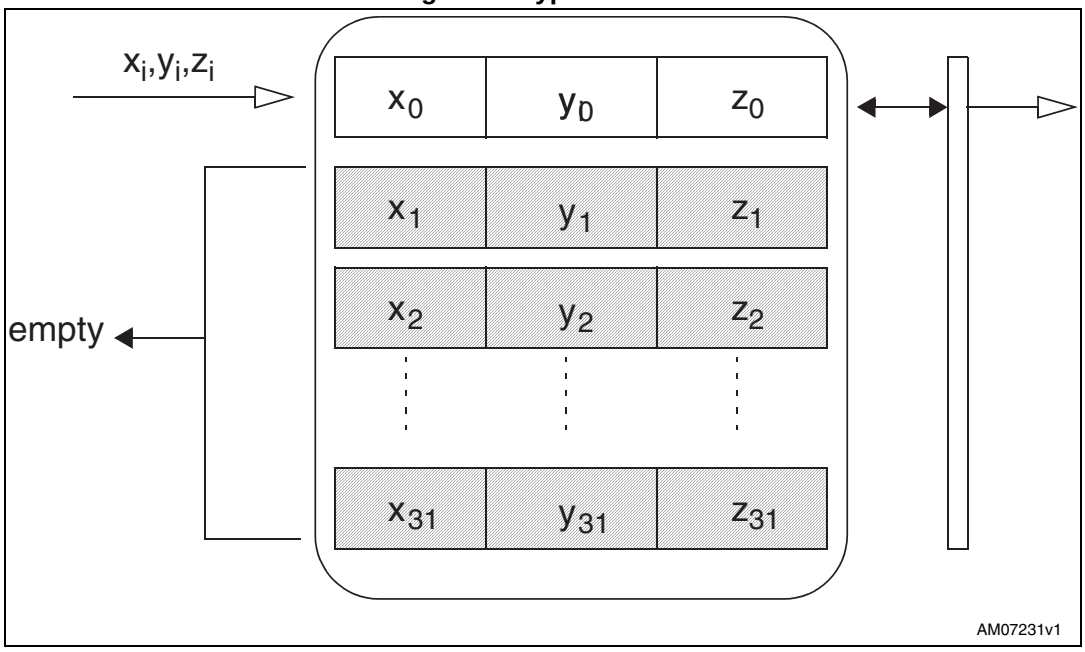
4.2 FIFO

The L3GD20 embeds 32 slots of 16-bit data FIFO for each of the three output channels: yaw, pitch and roll. This allows consistent power saving for the system, since the host processor does not need to continuously poll data from the sensor, but can wake up only when needed and burst the significant data out from the FIFO. This buffer can work accordingly in five different modes: Bypass mode, FIFO mode, Stream mode, Bypass-to-Stream mode and Stream-to-FIFO mode. Each mode is selected by the FIFO_MODE bits in the FIFO_CTRL_REG (2Eh). 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 (22h) and event detection information is available in FIFO_SRC_REG (2Fh). Watermark level can be configured to WTM4:0 in FIFO_CTRL_REG (2Eh).

4.2.1 Bypass mode

In Bypass mode, the FIFO is not operational and for this reason it remains empty. As described in *Figure 7* below, for each channel only the first address is used. The remaining FIFO slots are empty. When new data is available, the old data is overwritten.

Figure 7. Bypass mode

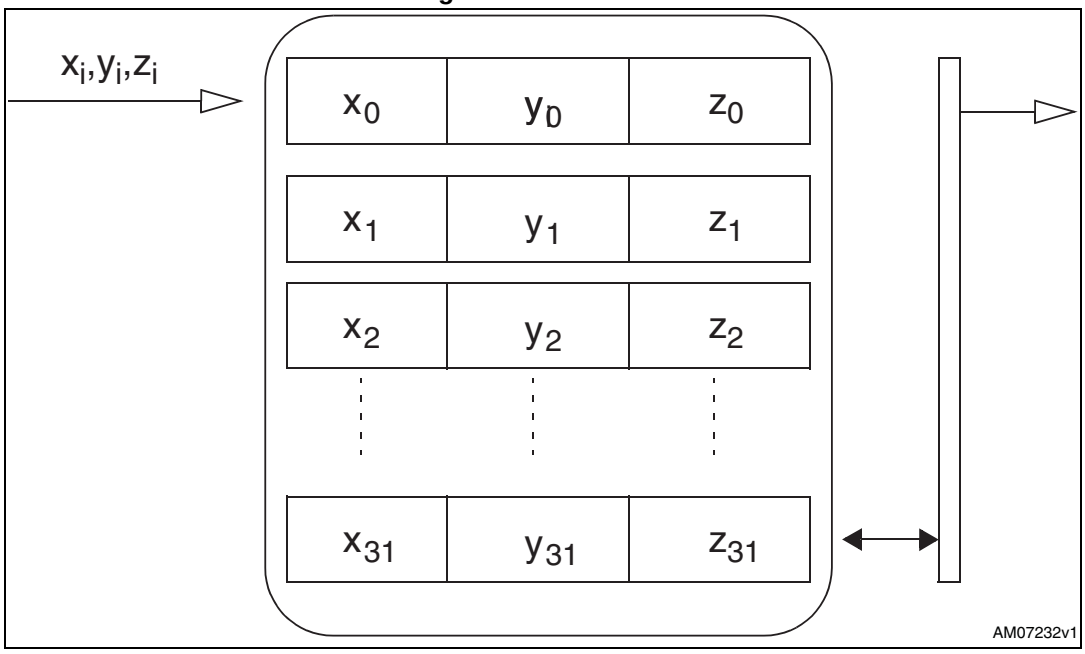


4.2.2 FIFO mode

In FIFO mode, data from the yaw, pitch and roll channels is stored in the FIFO. A watermark interrupt can be enabled (I2_WMK bit into CTRL_REG3 (22h)) in order to be raised when the FIFO is filled to the level specified in the WTM 4:0 bits of FIFO_CTRL_REG (2Eh). 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, the FIFO_CTRL_REG (2Eh) must be written back to Bypass mode.

FIFO mode is represented in *Figure 8: FIFO mode*.

Figure 8. FIFO mode

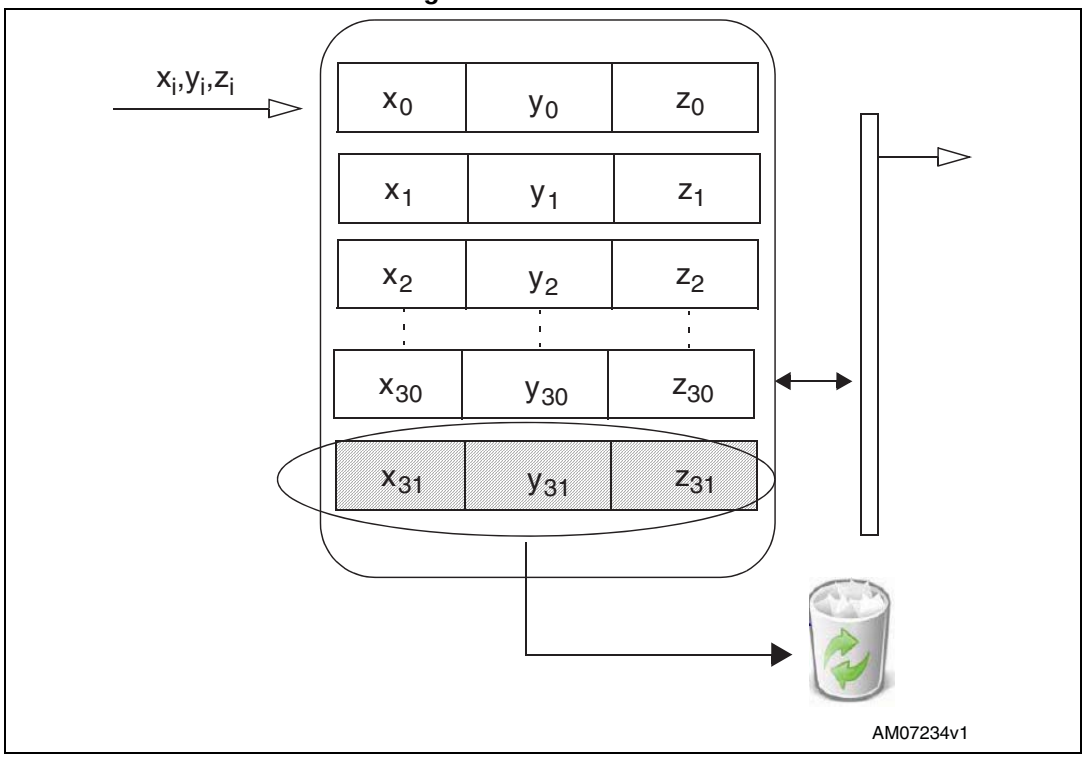


4.2.3 Stream mode

In Stream mode, data from yaw, pitch and roll measurement are stored in the FIFO. A watermark interrupt can be enabled and set as in the FIFO mode.The FIFO continues filling until it is 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 (22h).

Stream mode is represented in *Figure 9: Stream mode*.

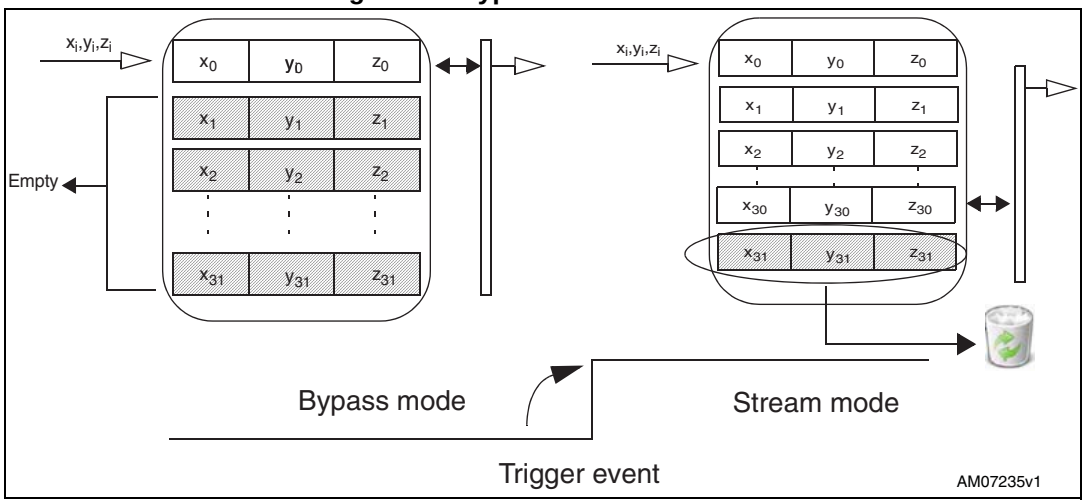
Figure 9. Stream mode



4.2.4 Bypass-to-stream mode

In Bypass-to-stream mode, the FIFO begins operating in Bypass mode and once a trigger event occurs (related to INT1_CFG (30h) register events), the FIFO starts operating in Stream mode. Refer to *Figure 10* below.

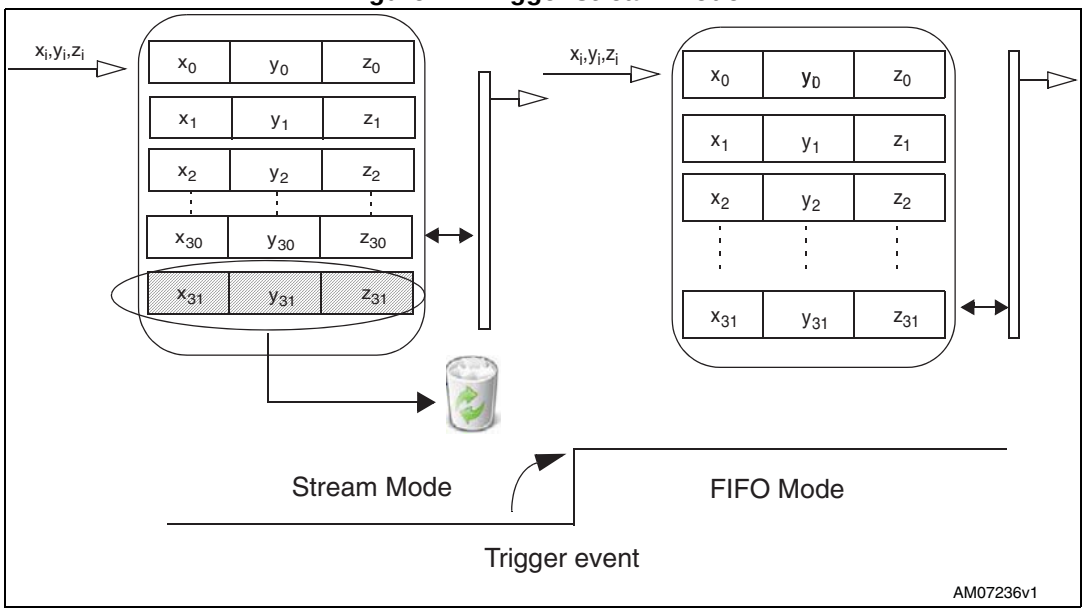
Figure 10. Bypass-to-stream mode



4.2.5 Stream-to-FIFO mode

In Stream-to-FIFO mode, data from yaw, pitch and roll measurement is stored in the FIFO. A watermark interrupt can be enabled on pin DRDY/INT2 by setting the I2_WTM bit in CTRL_REG3 (22h) in order to be raised when the FIFO is filled to the level specified in the WTM4:0 bits of FIFO_CTRL_REG (2Eh). The FIFO continues filling until it is 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 (30h) register events), the FIFO starts operating in FIFO mode. Refer to *Figure 11* below.

Figure 11. Trigger stream mode



4.2.6 Retrieve data from FIFO

FIFO data is read through OUT_X (Addr reg 28h,29h), OUT_Y (Addr reg 2Ah,2Bh) and OUT_Z (Addr reg 2Ch,2Dh). When the FIFO is in Stream, Trigger or FIFO mode, a read operation of 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 is placed in the OUT_X, OUT_Y and OUT_Z registers and both single read and read_burst (X,Y & Z with auto-incrementing address) operations can be used. When data included in OUT_Z_H (2Dh) is read, the system restarts to read information from addr OUT_X_L (28h).

5 Digital interfaces

The registers embedded in the L3GD20 may be accessed through both the I^2C and SPI serial interfaces. The latter may be SW-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 10. Serial interface pin description

Pin name	Pin description
CS	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 less significant bit of the device address

5.1 I^2C serial interface

The L3GD20 I^2C is a bus slave. The I^2C is employed to write data into registers whose content can also be read back.

The relevant I^2C terminology is given in the table below.

Table 11. 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 external pull-up resistors. When the bus is free, both 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 seven 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 L3GD20 is 110101xb. The **SDO** pin can be used to modify the less significant bit of the device address. If the SDO pin is connected to voltage supply, LSb is '1' (address 1101011b). Otherwise, if the SDO pin is connected to ground, the LSb value is '0' (address 1101010b). This solution allows to connect and address 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 obligated to generate an acknowledge after each byte of data received.

The I²C embedded in the L3GD20 behaves like a slave device and the following protocol must be adhered to. After the start condition (ST) 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) will be automatically incremented to allow multiple data read/write.

The slave address is completed with a Read/Write bit. If the bit was '1' (Read), a repeated START (SR) condition must be issued after the two sub-address bytes; if the bit is '0' (Write) the master will transmit to the slave with direction unchanged. *Table 12* explains how the SAD+Read/Write bit pattern is composed, listing all the possible configurations.

Table 12. SAD+read/write patterns

Command	SAD[6:1]	SAD[0] = SDO	R/W	SAD+R/W
Read	110101	0	1	11010101 (D5)
Write	110101	0	0	11010100 (D4)
Read	110101	1	1	11010111 (D7)
Write	110101	1	0	11010110 (D6)

Table 13. Transfer when master is writing one byte to slave

Master	ST	SAD + W		SUB		DATA		SP
Slave			SAK		SAK		SAK	

Table 14. Transfer when master is writing multiple bytes to slave

Master	ST	SAD + W		SUB		DATA		DATA		SP
Slave			SAK		SAK		SAK		SAK	

Table 15. 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 16. 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 is transmitted in byte format (DATA). Each data transfer contains 8 bits. The number of bytes sent 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.

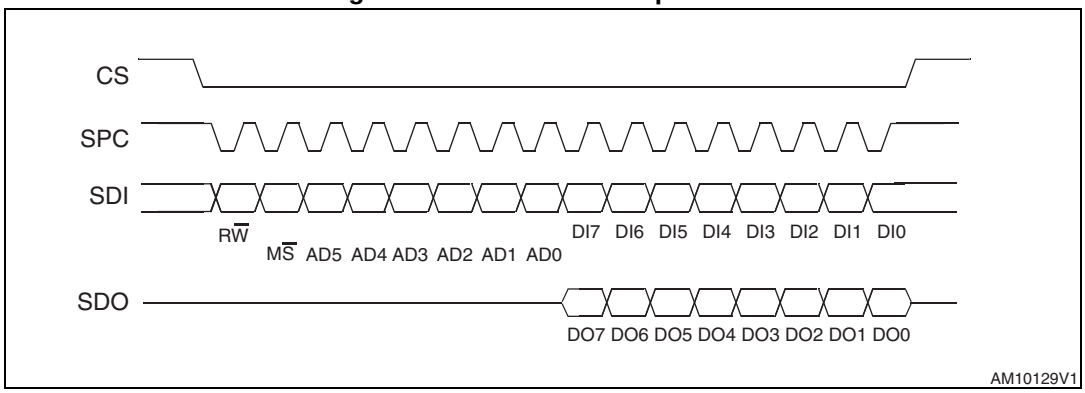
In the communication format presented, MAK is Master Acknowledge and NMAK is No Master Acknowledge.

5.2 SPI bus interface

The SPI is a bus slave. The SPI allows writing and reading the registers of the device.

The serial interface interacts with the outside world through 4 wires: **CS**, **SPC**, **SDI** and **SDO**.

Figure 12. 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 goes back high at the end. **SPC** is the Serial Port Clock and it 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. Those 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 bytes read/write. 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, ...) 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 will drive **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 will be 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 will be written to the device (MSb first).

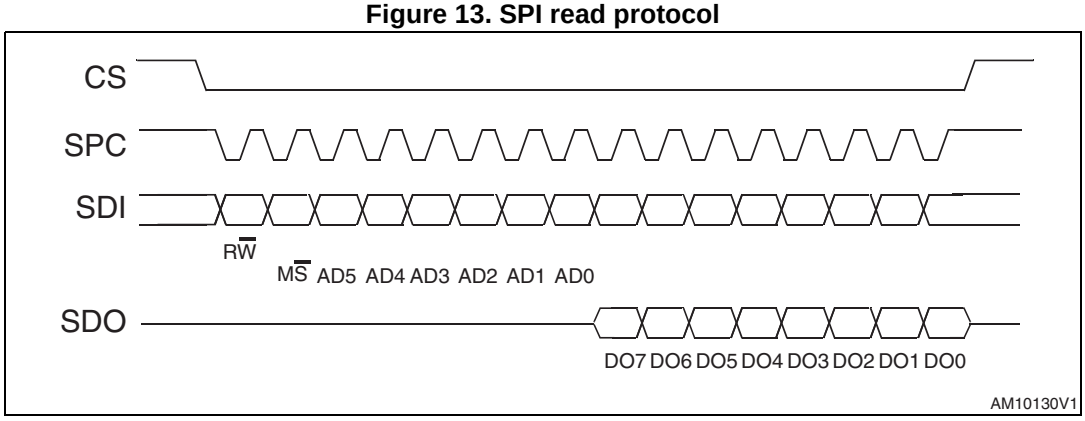
bit 8-15: data DO(7:0) (read mode). This is the data that will be read from the device (MSb first).

In multiple read/write commands, further blocks of 8 clock periods will be 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 13. SPI read protocol



The SPI read command is performed with 16 clock pulses. The 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.

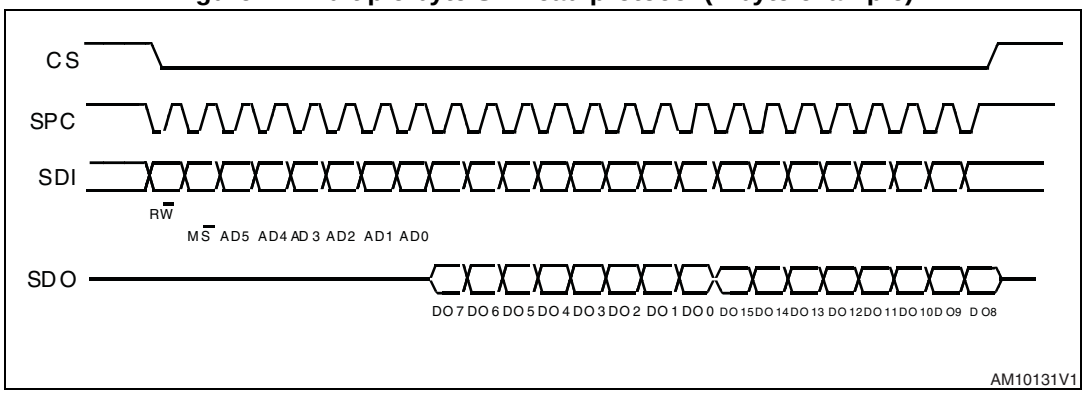
bit 1: MS bit. When 0 do not increment address; when 1 increment address in multiple reading.

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 will be read from the device (MSb first).

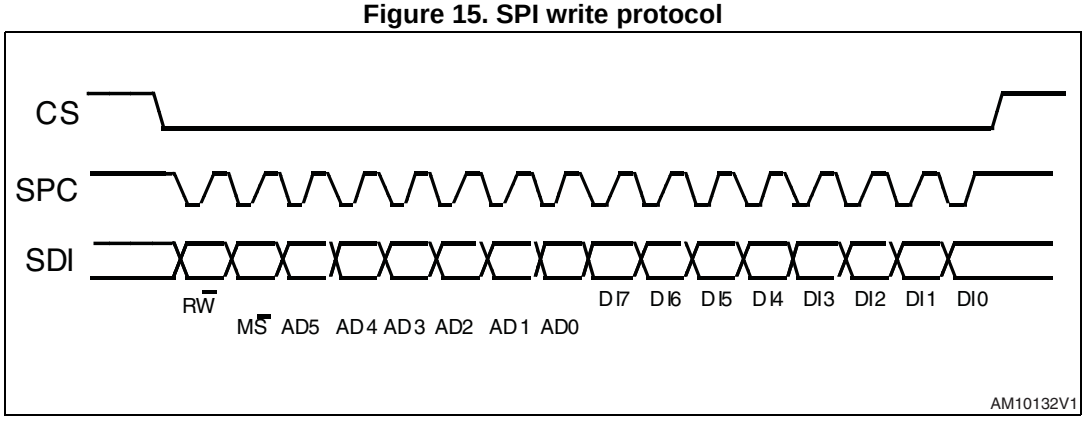
bit 16-... : data DO(...-8). Further data in multiple byte reading.

Figure 14. Multiple byte SPI read protocol (2-byte example)



5.2.2 SPI write

Figure 15. SPI write protocol



The SPI Write command is performed with 16 clock pulses. The 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.

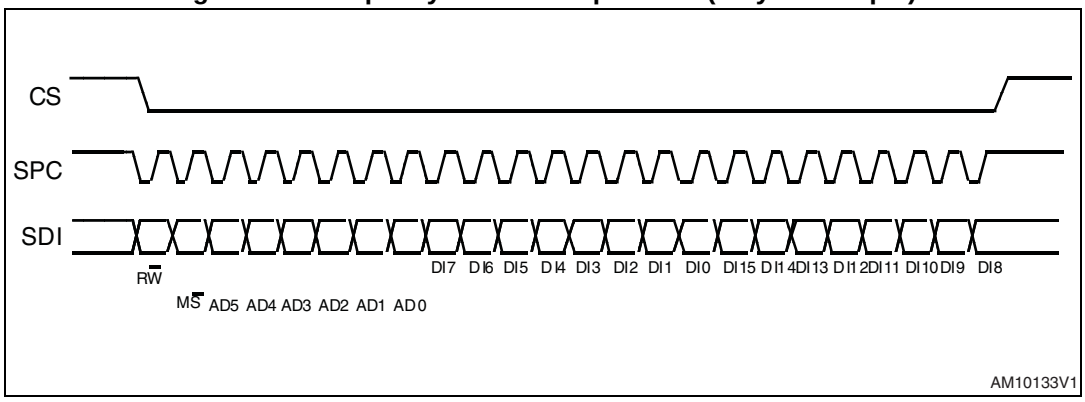
bit 1: $\overline{MS}$ bit. When 0, do not increment address; when 1, increment address in multiple writing.

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 will be written to the device (MSb first).

bit 16-... : data DI(...-8). Further data in multiple byte writing.

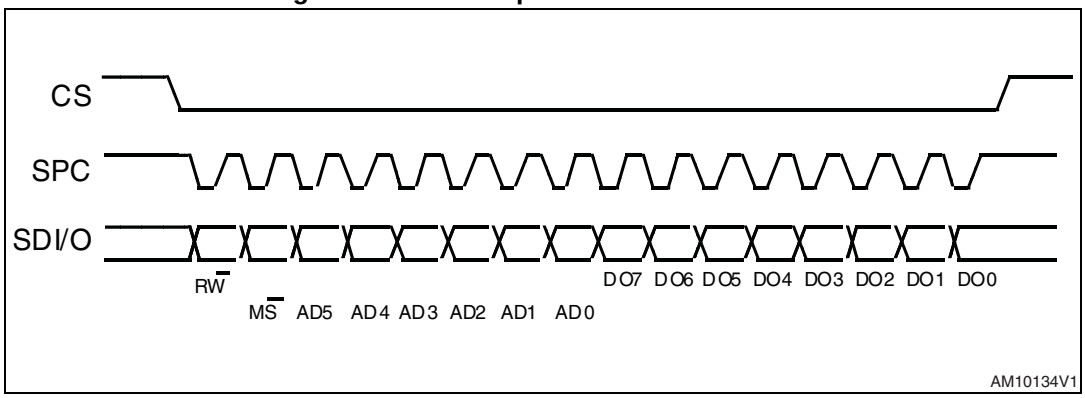
Figure 16. Multiple byte SPI write protocol (2-byte example)



5.2.3 SPI read in 3-wire mode

3-wire mode is entered by setting the bit SIM (SPI serial interface mode selection) to ‘1’ in CTRL_REG2.

Figure 17. 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: $M\overline{S}$ 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 will be read from the device (MSb first).

Multiple read command is also available in 3-wire mode.

LSM303DLHC

Ultra compact high performance e-compass 3D accelerometer and 3D magnetometer module

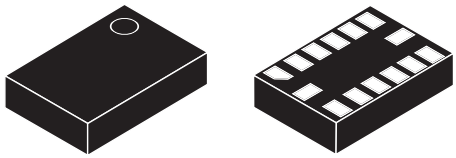
Preliminary data

Features

- 3 magnetic field channels and 3 acceleration channels
- From ±1.3 to ±8.1 gauss magnetic field full-scale
- ±2g/±4g/±8g/±16g selectable full-scale
- 16 bit data output
- I^2C serial interface
- Analog supply voltage 2.16 V to 3.6 V
- Power-down mode/ low-power mode
- 2 independent programmable interrupt generators for free-fall and motion detection
- Embedded temperature sensor
- Embedded FIFO
- 6D/4D orientation detection
- ECOPACK® RoHS and “Green” compliant

Applications

- Compensated compass
- Map rotation
- Position detection
- Motion-activated functions
- Free-fall detection
- Click/double click recognition
- Pedometer
- Intelligent power-saving for handheld devices
- Display orientation
- Gaming and virtual reality input devices
- Impact recognition and logging
- Vibration monitoring and compensation



LGA-14 (3x5x1mm)

Description

The LSM303DLHC is a system-in-package featuring a 3D digital linear acceleration sensor and a 3D digital magnetic sensor.

LSM303DLHC has linear acceleration full-scales of ±2*g* / ±4*g* / ±8*g* / ±16*g* and a magnetic field full-scale of ±1.3 / ±1.9 / ±2.5 / ±4.0 / ±4.7 / ±5.6 / ±8.1 gauss. All full-scales available are fully selectable by the user.

LSM303DLHC includes an I^2C serial bus interface that supports standard and fast mode 100 kHz and 400kHz. The system can be configured to generate interrupt signals by inertial wake-up/free-fall events as well as by the position of the device itself. Thresholds and timing of interrupt generators are programmable by the end user on the fly. Magnetic and accelerometer parts can be enabled or put into power-down mode separately.

The LSM303DLHC is available in a plastic land grid array package (LGA) and is guaranteed to operate over an extended temperature range from -40 °C to +85 °C.

Table 1. Device summary

Part number	Temperature range [°C]	Package	Packing
LSM303DLHC	-40 to +85	LGA-14	Tray
LSM303DLHCTR	-40 to +85	LGA-14	Tape and reel

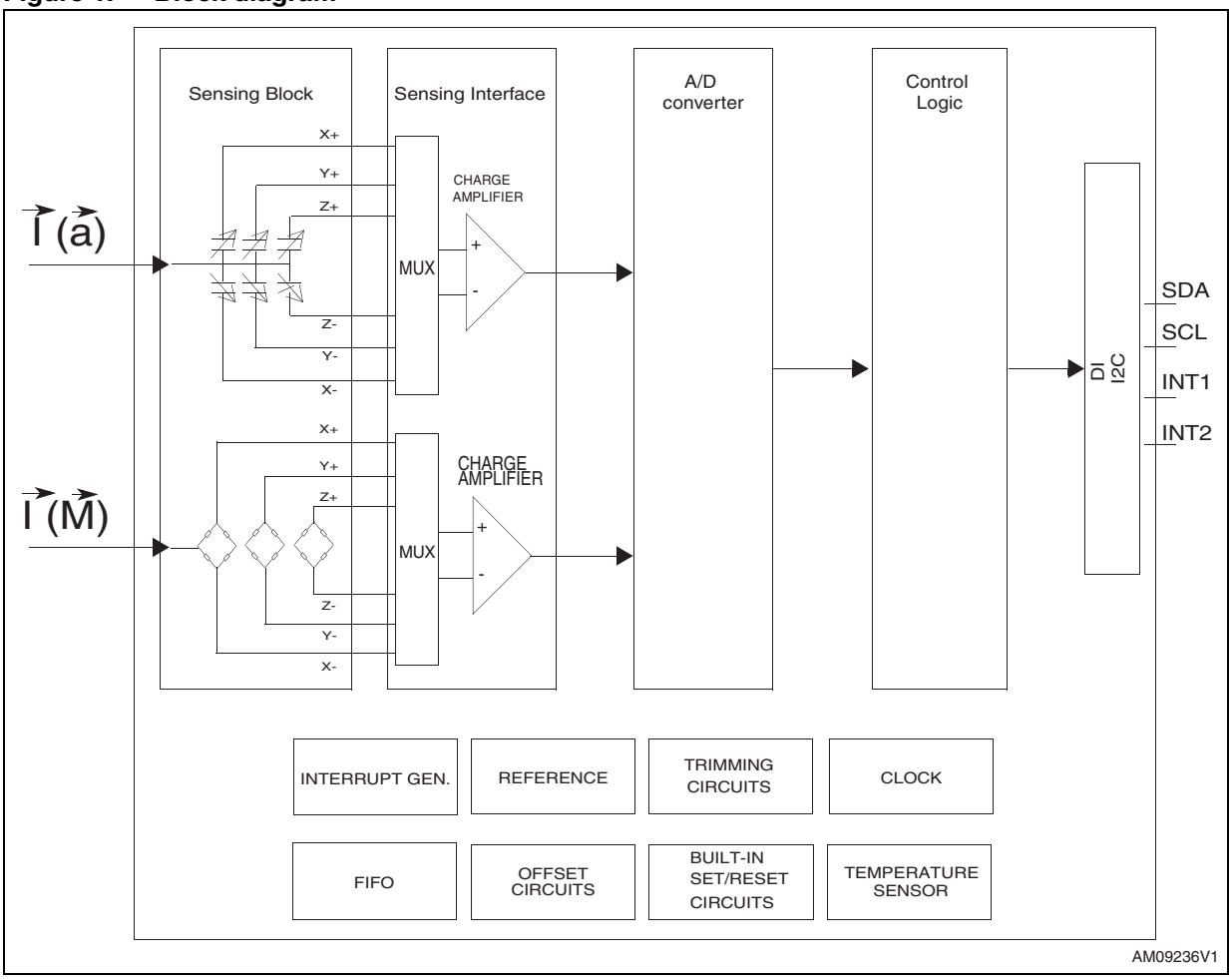
Contents

List of tables

1 Block diagram and pin description

1.1 Block diagram

Figure 1. Block diagram



1.2 Pin description

Figure 2. Pin connection

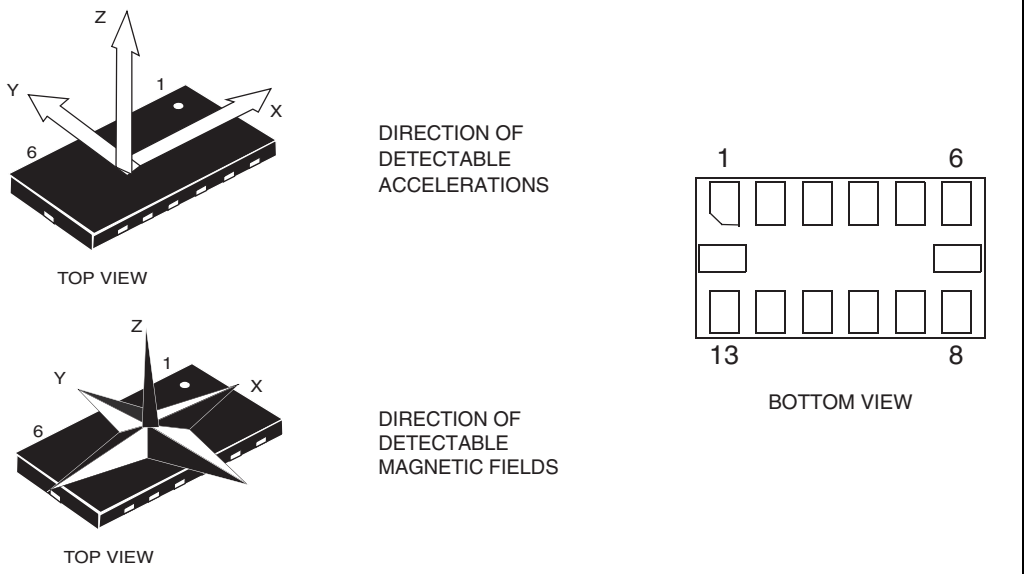


Table 2. Pin description

Pin#	Name	Function
1	Vdd_IO	Power supply for I/O pins
2	SCL	Signal interface I^2C serial clock (SCL)
3	SDA	Signal interface I^2C serial data (SDA)
4	INT2	Inertial Interrupt 2
5	INT1	Inertial Interrupt 1
6	C1	Reserved capacitor connection (C1)
7	GND	0 V supply
8	Reserved	Leave unconnected
9	DRDY	Data ready
10	Reserved	Connect to GND
11	Reserved	Connect to GND
12	SETP	S/R capacitor connection (C2)
13	SETC	S/R capacitor connection (C2)
14	Vdd	Power supply

2 Module specifications

2.1 Sensor characteristics

@ Vdd = 2.5 V, T = 25 °C unless otherwise noted[a].

Table 3. Sensor characteristics

Symbol	Parameter	Test conditions	Min.	Typ.[1]	Max.	Unit
LA_FS	Linear acceleration measurement range[2]	FS bit set to 00		±2		*g*
		FS bit set to 01		±4		
		FS bit set to 10		±8		
		FS bit set to 11		±16		
M_FS	Magnetic measurement range	GN bits set to 001		±1.3		gauss
		GN bits set to 010		±1.9		
		GN bits set to 011		±2.5		
		GN bits set to 100		±4.0		
		GN bits set to 101		±4.7		
		GN bits set to 110		±5.6		
		GN bits set to 111		±8.1		
LA_So	Linear acceleration sensitivity	FS bit set to 00		1		mg/LSB
		FS bit set to 01		2		
		FS bit set to 10		4		
		FS bit set to 11		12		
M_GN	Magnetic gain setting	GN bits set to 001 (X,Y)		1100		LSB/ gauss
		GN bits set to 001 (Z)		980		
		GN bits set to 010 (X,Y)		855		
		GN bits set to 010 (Z)		760		
		GN bits set to 011 (X,Y)		670		
		GN bits set to 011 (Z)		600		
		GN bits set to 100 (X,Y)		450		
		GN bits set to 100 (Z)		400		
		GN bits set to 101 (X,Y)		400		
		GN bits set to 101 (Z)		355		
		GN bits set to 110 (X,Y)		330		
		GN bits set to 110 (Z)		295		
		GN bits set to 111[2] (X,Y)		230		
		GN bits set to 111[2] (Z)		205		

a. The product is factory calibrated at 2.5 V. The operational power supply range is from 2.16 V to 3.6 V.

Table 3. Sensor characteristics (continued)

Symbol	Parameter	Test conditions	Min.	Typ.(1)	Max.	Unit
LA_TCSo	Linear acceleration sensitivity change vs. temperature	FS bit set to 00		±0.01		%/°C
LA_TyOff	Linear acceleration typical Zero-*g* level offset accuracy(3),(4)	FS bit set to 00		±60		m*g*
LA_TCOff	Linear acceleration Zero-*g* level change vs. temperature	Max. delta from 25 °C		±0.5		m*g*/°C
LA_An	Acceleration noise density	FS bit set to 00, normal mode(*Table 8.*), ODR bit set to 1001		220		ug/ sqrt(Hz)
M_R	Magnetic resolution			2		mgauss
M_CAS	Magnetic cross-axis sensitivity	Cross field =.0.5 gauss H applied = ±3 gauss		±1		%FS/ gauss
M_EF	Maximum exposed field	No permitting effect on zero reading			10000	gauss
M_DF	Disturbing field	Sensitivity starts to degrade. Use S/R pulse to restore sensitivity			20	gauss
Top	Operating temperature range		-40		+85	°C

1. Typical specifications are not guaranteed.
2. Verified by wafer level test and measurement of initial offset and sensitivity.
3. Typical Zero-*g* level offset value after MSL3 preconditioning.
4. Offset can be eliminated by enabling the built-in high pass filter.

2.2 Temperature sensor characteristics

@ Vdd = 2.5 V, T = 25 °C unless otherwise noted [b].

Table 4. Temperature sensor characteristics

Symbol	Parameter	Test condition	Min.	Typ.(1)	Max.	Unit
TSDr	Temperature sensor output change vs. temperature	-		8		LSB/°C(2)
TODR	Temperature refresh rate			ODR(3)		Hz
Top	Operating temperature range		-40		+85	°C

1. Typical specifications are not guaranteed.
2. 12-bit resolution.
3. For ODR configuration refer to *Table 72*.

b. The product is factory calibrated at 2.5 V.

2.3 Electrical characteristics

@ Vdd = 2.5 V, T = 25 °C unless otherwise noted.

Table 5. Electrical characteristics

Symbol	Parameter	Test conditions	Min.	Typ.(1)	Max.	Unit
Vdd	Supply voltage	-	2.16		3.6	V
Vdd_IO	Module power supply for I/O		1.71	1.8	Vdd+0.1	
Idd	Current consumption in normal mode(2)			110		µA
IddSL	Current consumption in sleep-mode(3)			1		µA
Top	Operating temperature range		-40		+85	°C

1. Typical specifications are not guaranteed.
2. Magnetic sensor setting ODR = 7.5 Hz, Accelerometer sensor ODR = 50 Hz.
3. Linear accelerometer in sleep-mode and magnetic sensor in power-down mode.

2.4 Communication interfaces characteristics

External pull-up resistors are required to support I²C standard and fast speed modes.

2.4.1 Sensor I²C - inter IC control interface

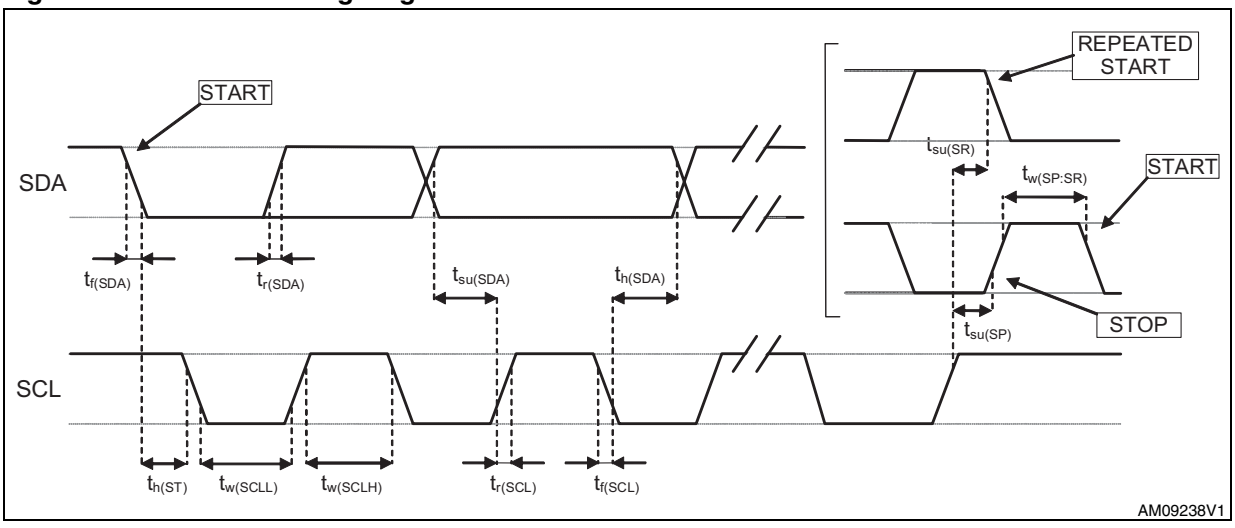
Subject to general operating conditions for Vdd and Top.

Table 6. 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	0.01	3.45	0.01	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 3. I²C slave timing diagram (c)



2.5 Absolute maximum ratings

Stresses above those 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 7. Absolute maximum ratings

Symbol	Ratings	Maximum value	Unit
Vdd	Supply voltage	-0.3 to 4.8	V
Vdd_IO	I/O pins supply voltage	-0.3 to 4.8	V
Vin	Input voltage on any control pin (SCL, SDA)	-0.3 to Vdd_IO +0.3	V
A_{POW}	Acceleration (any axis, powered, Vdd = 2.5 V)	3,000 for 0.5 ms	*g*
		10,000 for 0.1 ms	*g*
A_{UNP}	Acceleration (any axis, unpowered)	3,000 for 0.5 ms	*g*
		10,000 for 0.1 ms	*g*
T_{OP}	Operating temperature range	-40 to +85	°C
T_{STG}	Storage temperature range	-40 to +125	°C

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.

c. Measurement points are done at 0.2·Vdd_IO and 0.8·Vdd_IO, for both ports.

2.6 Terminology

2.6.1 Linear acceleration sensitivity

Linear acceleration sensitivity describes the gain of the accelerometer sensor and can be determined by applying 1 *g* acceleration to it. As the sensor can measure DC accelerations, this can be done easily by pointing the axis of interest towards the center of the Earth, noting the output value, rotating the sensor by 180 degrees (pointing to the sky) and noting the output value again. By doing so, ±1 *g* acceleration is applied to the sensor. Subtracting the larger output value from the smaller one, and dividing the result by 2, leads to the actual sensitivity of the sensor. This value changes very little over temperature and also very little over time. The sensitivity tolerance describes the range of sensitivities of a large population of sensors.

2.6.2 Zero-g level

Zero-*g* level offset (TyOff) describes the deviation of an actual output signal from the ideal output signal if no acceleration is present. A sensor in a steady-state on a horizontal surface measures 0 *g* in the X axis and 0 *g* in the Y axis whereas the Z axis measures 1 *g*. The output is ideally in the middle of the dynamic range of the sensor (content of OUT registers 00h, data expressed as 2's complement number). A deviation from the ideal value in this case is called Zero-*g* offset. Offset is, to some extent, a result of stress to the MEMS sensor and therefore the offset can slightly change after mounting the sensor onto a printed circuit board or exposing it to extensive mechanical stress. Offset changes little over temperature, see "Zero-*g* level change vs. temperature". The Zero-*g* level tolerance (TyOff) describes the standard deviation of the range of Zero-*g* levels of a population of sensors.

3 Functionality

The LSM303DLHC is a system-in-package featuring a 3D digital linear acceleration and 3D digital magnetic field detection sensor.

The system includes specific sensing elements and an IC interface capable of measuring both the linear acceleration and magnetic field applied on it and to provide a signal to the external world through an I^2C serial interface with separated digital output.

The sensing system is manufactured using specialized micromachining processes, while the IC interfaces are realized using a CMOS technology that allows to design a dedicated circuit which is trimmed to better match the sensing element characteristics.

The LSM303DLHC features two data-ready signals (RDY) which indicate when a new set of measured acceleration data and magnetic data are available, therefore simplifying data synchronization in the digital system that uses the device.

The LSM303DLHC may also be configured to generate a free-fall interrupt signal according to a programmed acceleration event along the enabled axes.

Linear acceleration operating mode

LSM303DLHC provides two different acceleration operating modes, respectively reported as “normal mode” and “low-power mode”. While normal mode guarantees high resolution, low-power mode reduces further the current consumption.

Table 8 summarizes how to select the operating mode.

Table 8. Accelerometer operating mode selection

Operating mode	CTRL_REG1[3] (LPen bit)	CTRL_REG4[3] (HR bit)	BW [Hz]	Turn-on time [ms]
Low-power mode	1	0	ODR/2	1
Normal mode	0	1	ODR/9	7/ODR

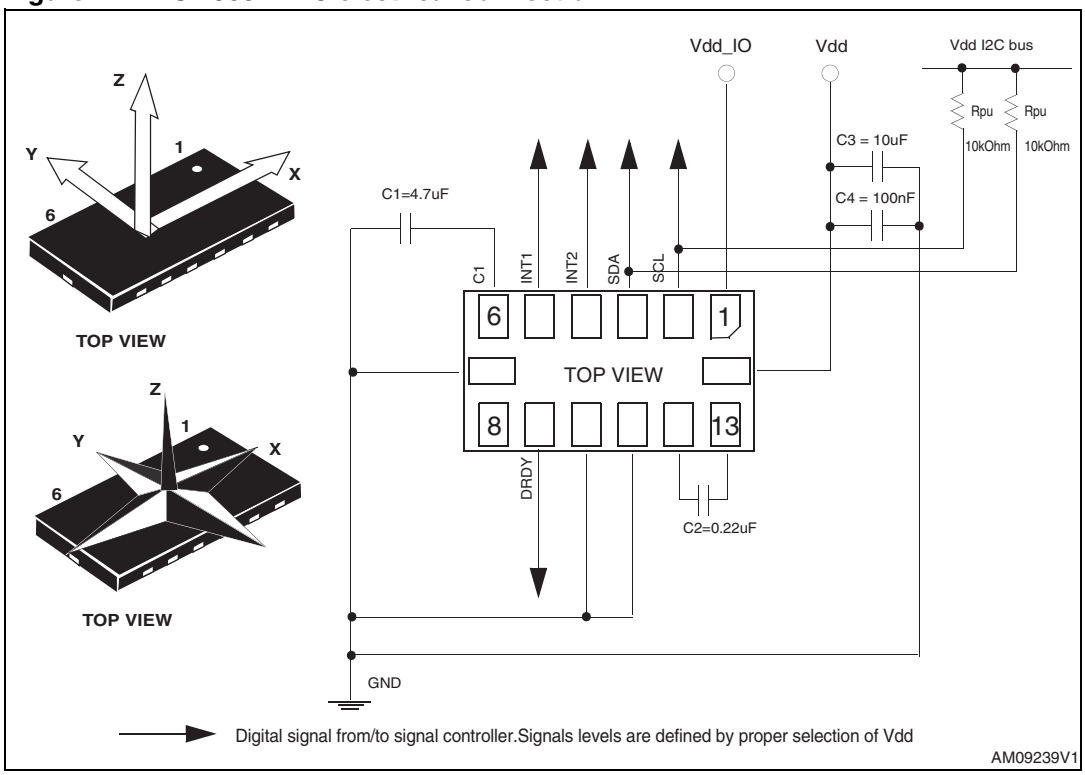
3.1 Factory calibration

The IC interface is factory calibrated for linear acceleration sensitivity (LA_So), and linear acceleration Zero-g level (LA_TyOff).

The trimming values are stored inside the device by a non-volatile memory. Any time the device is turned on, the trimming parameters are downloaded into the registers to be used during the normal operation. This allows the user to use the device without further calibration.

4 Application hints

Figure 4. LSM303DLHC electrical connection



4.1 capacitors

The C1 and C2 external capacitors should be low SR value ceramic type constructions (typ. suggested value 200 mOhm). Reservoir capacitor C1 is nominally 4.7 µF in capacitance, with the set/reset capacitor C2 nominally 0.22 µF in capacitance.

The device core is supplied through the Vdd line. Power supply decoupling capacitors (C4=100 nF ceramic, C3=10 µF Al) should be placed as near as possible to the supply pin of the device (common design practice). All the voltage and ground supplies must be present at the same time to have proper behavior of the IC (refer to *Figure 4*).

The functionality of the device and the measured acceleration/magnetic field data is selectable and accessible through the I^2C interface.

The functions, the threshold, and the timing of the two interrupt pins (INT 1 and INT 2) can be completely programmed by the user through the I^2C interface.

4.2 Pull-up resistors

Pull-up resistors (suggested value 10 kOhm) are placed on the two I^2C bus lines.

4.3 Digital interface power supply

This digital interface, dedicated to the linear acceleration and to the magnetic field signal, is capable of operating with a standard power supply (Vdd) or using a dedicated power supply (Vdd_IO).

4.4 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/mems.

4.5 High current wiring effects

High current in the wiring and printed circuit trace can be culprits in causing errors in magnetic field measurements for compassing.

Conductor generated magnetic fields add to the Earth’s magnetic field, causing errors in compass heading computation.

Keep currents higher than 10 mA a few millimeters further away from the sensor IC.

5 Digital interfaces

The registers embedded inside the LSM303DLHC are accessible through two separate I^2C serial interfaces, one for the accelerometer core and one for the magnetometer core.

Table 9. Serial interface pin description

PIN Name	PIN Description
SCL	I^2C serial clock (SCL)
SDA	I^2C serial data (SDA)

5.1 I^2C serial interface

The LSM303DLHC I^2C is a bus slave. The I^2C is employed to write the data into the registers whon also be read back.

The relevant I^2C terminology is given in the table below.

Table 10. Serial interface pin description

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.

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 bit 8 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 seven bits after a start condition with its address. If they match, the device considers itself addressed by the master.

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 inside the LSM303DLHC behaves like a slave device and the following protocol must be adhered to. After the start condition (ST) a slave address is sent, once a slave acknowledge (SAK) has been returned, an 8-bit sub-address (SUB) is transmitted; the 7 LSBs 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 increased to allow multiple data Read/Write.

Table 11. Transfer when master is writing one byte to slave

Master	ST	SAD + W		SUB		DATA		SP
Slave			SAK		SAK		SAK	

Table 12. Transfer when master is writing multiple bytes to slave:

Master	ST	SAD + W		SUB		DATA		DATA		SP
Slave			SAK		SAK		SAK		SAK	

Table 13. 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		

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 can't 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 doesn't 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.

5.1.2 Linear acceleration digital interface

For linear acceleration the default (factory) 7-bit slave address is 0011001b.

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 14* explains how the ead/write bit pattern is composed, listing all the possible configurations.

Table 14. SAD+Read/Write patterns

Command	SAD[7:1]	R/W	SAD+R/W
Read	0011001	1	00110011 (33h)
Write	0011001	0	00110010 (32h)

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.

Table 15. 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		

5.1.3 Magnetic field digital interface

For magnetic sensors the default (factory) 7-bit slave address is 0011110xb.

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 16* explains how the SAD is composed.

Table 16. SAD

Command	SAD[6:0]	R/W	SAD+R/W
Read	0011110	1	00111101 (3Dh)
Write	0011110	0	00111100 (3Ch)

Magnetic signal interface reading/writing

The interface 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 plus 1 bit Read/Write identifier.

To minimize the communication between the master and magnetic digital interface of LSM303DLHC, the address pointer updates automatically without master intervention.

This automatic address pointer update has two additional features. First, when address 12 or higher is accessed, the pointer updates to address 00, and secondly, when address 08 is reached, the pointer rolls back to address 03. Logically, the address pointer operation functions as shown below.

If (address pointer = 08) then the address pointer = 03

Or else, if (address pointer >= 12) then the address pointer = 0

Or else, (address pointer) = (address pointer) + 1

The address pointer value itself cannot be read via the I^2C bus.

Any attempt to read an invalid address location returns 0, and any write to an invalid address location, or an undefined bit within a valid address location, is ignored by this device.

6 Register mapping

Table 17 provides a listing of the 8-bit registers embedded in the device and the related addresses:

Table 17. Register address map

Name	Slave address	Type	Register address		Default	Comment
			Hex	Binary		
Reserved (do not modify)	*Table 14*		00 - 1F	--	--	Reserved
CTRL_REG1_A	*Table 14*	rw	20	010 0000	00000111	
CTRL_REG2_A	*Table 14*	rw	21	010 0001	00000000	
CTRL_REG3_A	*Table 14*	rw	22	010 0010	00000000	
CTRL_REG4_A	*Table 14*	rw	23	010 0011	00000000	
CTRL_REG5_A	*Table 14*	rw	24	010 0100	00000000	
CTRL_REG6_A	*Table 14*	rw	25	010 0101	00000000	
REFERENCE_A	*Table 14*	rw	26	010 0110	00000000	
STATUS_REG_A	*Table 14*	r	27	010 0111	00000000	
OUT_X_L_A	*Table 14*	r	28	010 1000	output	
OUT_X_H_A	*Table 14*	r	29	010 1001	output	
OUT_Y_L_A	*Table 14*	r	2A	010 1010	output	
OUT_Y_H_A	*Table 14*	r	2B	010 1011	output	
OUT_Z_L_A	*Table 14*	r	2C	010 1100	output	
OUT_Z_H_A	*Table 14*	r	2D	010 1101	output	
FIFO_CTRL_REG_A	*Table 14*	rw	2E	010 1110	00000000	
FIFO_SRC_REG_A	*Table 14*	r	2F	010 1111		
INT1_CFG_A	*Table 14*	rw	30	011 0000	00000000	
INT1_SOURCE_A	*Table 14*	r	31	011 0001	00000000	
INT1_THS_A	*Table 14*	rw	32	011 0010	00000000	
INT1_DURATION_A	*Table 14*	rw	33	011 0011	00000000	
INT2_CFG_A	*Table 14*	rw	34	011 0100	00000000	
INT2_SOURCE_A	*Table 14*	r	35	011 0101	00000000	
INT2_THS_A	*Table 14*	rw	36	011 0110	00000000	
INT2_DURATION_A	*Table 14*	rw	37	011 0111	00000000	
CLICK_CFG_A	*Table 14*	rw	38	011 1000	00000000	
CLICK_SRC_A	*Table 14*	rw	39	011 1001	00000000	
CLICK_THS_A	*Table 14*	rw	3A	011 1010	00000000	
TIME_LIMIT_A	*Table 14*	rw	3B	011 1011	00000000	

Table 17. Register address map (continued)

Name	Slave address	Type	Register address		Default	Comment
			Hex	Binary		
TIME_LATENCY_A	*Table 14*	rw	3C	011 1100	00000000	
TIME_WINDOW_A	*Table 14*	rw	3D	011 1101	00000000	
Reserved (do not modify)	*Table 14*		3E-3F	--	--	Reserved
CRA_REG_M	*Table 16*	rw	00	00000000	0001000	
CRB_REG_M	*Table 16*	rw	01	00000001	0010000	
MR_REG_M	*Table 16*	rw	02	00000010	00000011	
OUT_X_H_M	*Table 16*	r	03	00000011	output	
OUT_X_L_M	*Table 16*	r	04	00000100	output	
OUT_Z_H_M	*Table 16*	r	05	00000101	output	
OUT_Z_L_M	*Table 16*	r	06	00000110	output	
OUT_Y_H_M	*Table 16*	r	07	00000111	output	
OUT_Y_L_M	*Table 16*	r	08	00001000	output	
SR_REG_Mg	*Table 16*	r	09	00001001	00000000	
IRA_REG_M	*Table 16*	r	0A	00001010	01001000	
IRB_REG_M	*Table 16*	r	0B	00001011	00110100	
IRC_REG_M	*Table 16*	r	0C	00001100	00110011	
Reserved (do not modify)	*Table 16*		0D-30	--	--	Reserved
TEMP_OUT_H_M	*Table 16*		31	00000000	output	
TEMP_OUT_L_M	*Table 16*		32	00000000	output	
Reserved (do not modify)	*Table 16*		33-3A	--	--	Reserved

Registers marked as “reserved” must not be changed. The writing to these registers may cause permanent damage to the device.

The content of the registers that are loaded at boot should not be changed. They contain the factory calibrated values. Their content is automatically restored when the device is powered up.