Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación
Intensificación en Sistemas Electrónicos

Estudio y modelado de errores que afectan a una Unidad de Medidas Inerciales de bajo coste.

Autor: Francisco González Mañero
Tutor: Fernando Caballero Benítez

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Estudio y modelado de errores que afectan a una Unidad de Medidas Inerciales de bajo coste.

Autor:
Francisco González Mañero

Tutor:
Fernando Caballero Benítez
Profesor Contratado Doctor

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Fin de Grado: Estudio y modelado de errores que afectan a una Unidad de Medidas Inerciales de bajo coste.

Autor: Francisco González Mañero

Tutor: Fernando Caballero Benítez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

En primer lugar, quisiera dar las gracias a mi tutor, Fernando Caballero Benítez, por darme la posibilidad de realizar dicho proyecto y por atenderme siempre que necesité su ayuda de una forma cercana y amena, haciendo agradable y enriquecedor el desarrollo de este proyecto.

También agradecer a mi familia, que siempre me ha apoyado en todos los aspectos de mi vida, económica y moralmente, contribuyendo a formar la persona que hoy soy. A los profesores tanto de mi colegio como de universidad, porque todos me han enseñado algo y ellos han conseguido que llegue hasta esta etapa final de la carrera.

Gracias a mis amigos, tanto los de toda la vida, los incondicionales, como a los que he conocido en estos años de carrera, que de una forma u otra han formado parte de mi vida. Gracias también a Marta y a mis compañeros de clase, haciendo mención especial a Luis, Braza y sobre todo el tridente electrónico, con quien tanto tiempo he pasado y tantas asignaturas he compartido, disfrutando y sufriendo la carrera.

Se acaba así una etapa muy importante de mi vida, y sólo quiero agradecer a todo el que haya tenido algo que ver en esto por minúscula que haya sido su aportación.

Resumen

Este proyecto presenta un análisis y modelado de los errores que afectan a los sensores de una Unidad de Medidas Inerciales (IMU) de bajo coste utilizando Allan Variance. Se realizará también el diseño y desarrollo de un filtro de Kalman (KF) aplicado a dicha IMU. Dicho filtro tratará de estimar un estado que incluirá, entre otros, los valores correspondientes a los ángulos de Euler (pitch, roll, yaw) con el objetivo de obtener información acerca de la orientación del dispositivo. El trabajo está orientado a vehículos aéreos no tripulados (UAV), como drones.

Antes del desarrollo del proyecto, se proporcionará una pequeña introducción a conceptos importantes del trabajo, como los Sistemas de Navegación Inercial (INS), las IMU, los filtros de Kalman o el análisis Allan Variance.

Las IMU basadas en MEMS tienen un amplio rango de aplicación, principalmente debido a su bajo coste. Este bajo coste hace que sea necesario compensar los errores que puedan influir en su correcto funcionamiento. Se realizará un modelado de los errores más característicos de los giróscopos y acelerómetros que esta IMU incluye, centrándonos en la dependencia que la varianza de dichos errores tiene con la temperatura. Para obtener dicha dependencia se realizará un análisis Allan Variance, en el que identificaremos los diferentes tipos de errores que afectan a cada sensor.

Finalmente, obtendremos un filtro de Kalman que incorpora dichas compensaciones mediante los modelos anteriormente comentados. El filtro está programado en C++ y el análisis de Allan Variance en Matlab.

Abstract

This Project presents a noise analysis and modeling for inertial sensors in a low-cost IMU using Allan Variance. It contains the design and development of a Kalman filter applied to an IMU too. This filter will try to estimate a state that will include the values of Euler angles (pitch, roll, yaw) with the purpose of obtaining information about the attitude of the device. This work is focused on Unmanned Aerial Vehicles (UAV) such as drones.

Before the development of the proyect, a brief introduction to important concepts such as INS, IMU, KF or Allan Variance Analysis is presented.

MEMS based Inertial Measurement Units have a wide range of applications mainly due to their low cost. Because of this low cost, we have to compensate for errors that may affect the proper functioning. A model of most characteristics errors of gyros and accelerometer will be done, focusing on the dependence between temperature and variance. To obtain this dependence an Allan Variance analysis will be done, where we will identify the different kind of errors that affect each sensor.

Finally, we will obtain a KF with error compensation, programmed in C++ and an Allan Variance analysis made with Matlab.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

KF	Kalman Filter
IMU	Inertial Measurement Unit (Unidad de Medidas Inerciales)
UAV	Unmanned Aerial Vehicle (Vehículo no tripulado)
MEMS	Microelectromechanical Systems
INS	Inertial Navigation System (Sistema de Navegación Inercial)
GPS	Global Positioning System
EKF	Extended Kalman Filter
IoT	Internet of Things
DOF	Degrees Of Freedom
SPIRE	Space Inertial Reference Equipment
UKF	Unscented Kalman Filter
MOEMS	Micro-Opto-Electro-Mechanical Systems
FOG	Fiber Optic Gyroscope
RLG	Ring Laser Gyroscope
CVG	Coriolis Vibratory Gyroscope
NMR	Nuclear Magnetic Resonance
CCID	Charge Coupled Imaging Device
SAW	Surface Acoustic Wave
AMR	Anisotropic Magnetoresistance
PSD	Power Spectral Density
MPU	Motion Processing Unit
DMP	Digital Motion Processor
PCB	Printed Circuit Board
I2C	Inter-Inegrated Circuit
RGB	Red Green Blue (led colours)
SPI	Serial Peripheral Interface
RISC	Reduced Instruction Set Computer
ADC	Analogic to Digital Converter
LCC	Leadless Chip Carrier
IDE	Integrated Drive Electronics
FIFO	First In First Out
QFN	Quad Flat No-leads
SMD	Surface Mount Device
CMOS	Complementary Metal-Oxide-Semiconductor
SOI	Silicon On Insulator
ESOI	Engine Silicon On Insulator
DRIE	Deep Reactive Ion Etching
RIE	Reactive Ion Etching

QN	Quantization Noise
ARW	Angle Random Walk
VRW	Velocity Random Walk
BI	Bias Instability
RRW	Rate Random Walk
RR	Rate Ramp
GM	Gauss-Markov
LSB	Least Significant Bit
AWGN	Additive White Gaussian Noise

1. INTRODUCCIÓN

We're still in the first minutes of the first day of the Internet revolution.

- Scott Cook -

1.1 Motivación

Vivimos en un mundo en el que el número de dispositivos que nos rodean crece año tras año. No es de extrañar que un futuro no muy lejano no diferenciemos entre lo que es natural y lo que es artificial, ya que cada vez hay más avances en biotecnología y otros ámbitos de la tecnología en general. Una de las áreas que más han sonado en los medios en los últimos años es la relacionada con los UAV o vehículos no tripulados. Cada vez más empresas sustituyen sus empleados por robots, en este ámbito suena el nombre de Amazon y sus drones de reparto. No obstante, los sistemas inerciales de los que trata este documento son anteriores a los drones, vienen siendo utilizados prácticamente desde los 1930s. Pero cuando el transporte deja de estar tripulado y no existe la mano humana para corregir los errores que aún afectan a nuestros dispositivos, se necesita tener una mayor precisión de estas tecnologías.

En el ámbito de la navegación inercial, siempre ha sido un problema de gran importancia la propagación de los errores. Esto es así porque los sensores inerciales que utilizamos para obtener la posición son, principalmente, acelerómetros y giróscopos, que miden aceleración lineal y velocidad angular respectivamente. Estas medidas tienen que integrarse una o dos veces para conseguir obtener de este modo la posición, por lo que el error se propaga de manera exponencial a lo largo del tiempo. Imaginen que al comenzar a conducir un coche por una autovía recta gira un grado el volante, parece insignificante, pero a lo largo de varios metros comenzará a notar como su coche, a pesar de no haber movido el volante casi nada, está cercano al arcén y deberá reconducirlo. Esta propagación del error es lineal, mientras que en el caso de las IMU es exponencial. Es por ello que se ha dedicado mucho tiempo a la corrección de estos errores.

Una solución muy adecuada para este problema es el uso de filtros de Kalman, ya que el hecho de ser filtros predictivos y de tener un algoritmo recursivo hace que pueda, con el correcto diseño, mitigar estos errores con mayor o menor precisión. Existe una gran variedad de errores que afectan a estos sensores, pero podremos observar cómo la complejidad del algoritmo del filtro de Kalman, acompañado de los modelos correspondientes, los reduce enormemente.

1.2 Objetivo

Este Proyecto tiene como principal objetivo el de realizar la implementación de un filtro de Kalman sobre sensores MEMS de bajo coste, realizando un correcto modelado de sus errores.

Una IMU consta de varios sensores, principalmente acelerómetros y giróscopos, que proporcionan medidas utilizadas de forma conjunta para el cálculo de nuestra posición. Esta fusión de sensores puede llevarse a cabo a través del filtro de Kalman, que mediante una serie de matrices va a conseguir implementar la relación entre las diferentes medidas y a obtener los resultados necesarios.

Existen diferentes modelos de ruido para los sensores de la IMU, y durante este trabajo alternaremos modelos

ya desarrollados como pueda ser el de Gauss-Markov y modelos calculados como el de la temperatura.

1.3 Estado del arte

1.3.1 INS

La navegación inercial es un sistema de navegación autónomo en el que las mediciones proporcionadas por acelerómetros y giróscopos se utilizan para rastrear la orientación y posición de un objeto con respecto a una posición, velocidad y orientación iniciales conocidos [1].

Al principio, los sistemas de navegación solían constar únicamente de GPS, que mediante comunicación satélite, conseguían realizar el rastreo de la posición de un vehículo de forma eficiente. Sin embargo, el avance de las tecnologías hizo que no fuera suficiente con esto. La comunicación por GPS tiene gran potencial, pero también es muy sensible a diversos factores externos como puedan ser meteorológicos, geográficos o a errores en los mismos satélites. Con el objetivo de complementar, y en algunos casos sustituir a esta tecnología, surgen los sistemas de navegación inercial, que como mencionamos al comienzo del capítulo, son sistemas autónomos y no están tan afectados por factores externos.

El núcleo de un INS suele ser una IMU, compuesta por giróscopos y acelerómetros, que miden velocidad angular y aceleración lineal. La IMU además puede complementarse con brújulas digitales o magnetómetros, medidores del campo magnético. Normalmente estos módulos disponen también de un microcontrolador que recibe los datos de los sensores y los procesa para transmitirlos en forma de voltajes.

1.3.1.1 IMU

Podemos pensar que la incorporación de las IMU a la navegación es reciente, no obstante, su origen tiene lugar en los 1930s, en el sector de la aeronáutica [2]. El desarrollo de un arma estabilizada con un giróscopo por Dr. Charles Stark Draper en 1941 y su vuelo transcontinental en 1953 con el sistema inercial SPIRE (Figura 1-2) consiguiendo una precisión de 10 millas después de un vuelo de 13 horas, marcó el comienzo de la revolución en los campos de la navegación, guía y control [3].

Sin embargo, sus limitaciones tanto en tamaño, como en consumo y coste, hacían que sus aplicaciones estuviesen enfocadas a dispositivos grandes y pesados, inapropiadas para pequeños dispositivos o para productos destinados al consumo. Sin embargo, en los últimos años, han aparecido los MEMS, que amplían mucho más el rango de aplicaciones al que la IMU puede acceder debido a sus características: bajo coste, pequeño tamaño y bajo consumo. Empresas como X-Sens, Microstrain, STMicroelectronics o Ivensense están compitiendo en este sector intentando satisfacer las necesidades de los consumidores. Este cambio en el mercado está influido por la necesidad cada vez mayor de sensado. El auge del IoT[1], cuyos principales requisitos es un gran número de sensores, de pequeño tamaño y bajo consumo, concuerda con el surgimiento de los MEMS, ya que se adaptan perfectamente a dichas características [4].

En el comienzo, las IMU estaban compuestas únicamente de acelerómetros y de giróscopos, teniendo de esta forma 6 DOF[2], y mediante sus medidas de aceleración linear y velocidad angular estimaban la posición. Más adelante, se incorpora a la IMU un nuevo sensor, el magnetómetro, que ya utilizaremos en este trabajo. Si bien el magnetómetro no es el más preciso de los tres sensores, y además está influido fuertemente por ruidos externos, proporciona una medida del campo magnético que complementará a las medidas hechas por el giróscopo, obteniendo de esta manera 9 DOF.

Debido a todos estos avances, como comentamos anteriormente, el número de aplicaciones para estas IMU ha crecido con los años, y está cada vez más presente en el día a día gracias a los IoT. Algunos ejemplos de aplicaciones actuales son:

- Navegación. Las aplicaciones en navegación son extensas, como este mismo proyecto. Un ejemplo puede ser el expuesto por A. Zul Zafar y D. Hazry que utilizan una 6-DOF IMU para estabilizar un

[1] Internet of Things: es un concepto que se refiere a la interconexión digital de objetos cotidianos con internet. Fuente: Wikipedia.
[2] Degrees Of Freedom: Determina el número de parámetros independientes en un sistema. En nuestro caso, se refiere a DOF como el número de ejes en los cuales se mide una magnitud. Ejemplo: Giróscopo de 3 ejes + acelerómetro de 3 ejes = 6 DOF.

quadrotor [5].

- Robótica. Muchos robots, al igual que los vehículos o que el propio ser humano, necesitan información sobre aceleraciones y velocidades angulares para desplazarse, como por ejemplo el StarIETH, que tiene incorporada una IMU de X-Sens. Michael Bloesch utiliza el EKF para medir la posición de la pata de este robot [6].
- Rehabilitación médica. Este ámbito está íntimamente relacionado con el anterior, ya que gran parte de los usos de IMU en la medicina están orientadas a exoesqueletos, como es el caso de [7].
- Realidad Aumentada. Útil cuando realizas simuladores de guerra o de vuelo.
- Deportes. Se han ido incrementando las medidas de la tecnología en el deporte, para poder realizar un arbitraje más correcto o para obtener información útil para los deportistas, como la velocidad con la que se lanza un balón. En el béisbol, por ejemplo, se ha incorporado en las pelotas, y el tamaño y resistencia de los MEMS hace posible que sean aptas para ello. Véase figura 1-1.

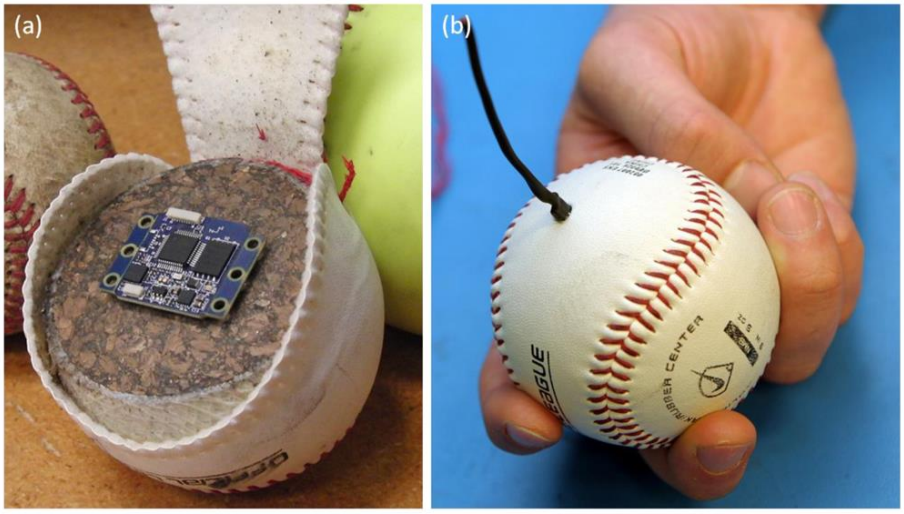


Figura 1-1. IMU embebida en una pelota de béisbol [8].

Figura 1-2. *SPIRE Inertial System* [9].

En el siguiente punto, incidiremos un poco en el giróscopo, ya que de los tres sensores que componen nuestra IMU, será en el que más nos centremos durante el proyecto.

1.3.1.1.1 Giróscopos

Desde su invención en 1852 por Jean Bernard Léon Foucault, Diversos tipos de giróscopos han ido surgiendo desde el giróscopo tradicional eminentemente mecánico (Figura 1-3) hasta el novedoso y aún futurista giróscopo cuántico (con capacidad para medir diferencias de minutos en cuanto a rotación angular se refiere), pasando por varios tipos, entre los que destacamos los MEMS, ya que han supuesto una revolución en la navegación.

Figura 1-3. Giróscopo tradicional.

La tecnología MEMS tiene su origen en los 1980s, y como hemos comentado anteriormente, es una tecnología rompedora, que provoca una gran revolución, lo que hace que tenga un valor estratégico y las grandes compañías tengan en cuenta su desarrollo. Desde la creación del primer giróscopo MEMS en 1988 en el CSDL Laboratory y Drapor Laboratory en Estados Unidos, su uso se ha incrementado enormemente, y el interés de grandes potencias como son el propio Estados Unidos, Europa, Japón o China (que venía estudiando la tecnología MEMS desde 1990), ha hecho [10]que sus empresas locales y el estado inviertan en esta nueva tecnología [10].

Existen grandes empresas de tecnología MEMS que están desarrollando giróscopos MEMS de tecnología punta, como por ejemplo [10]:

- STMicroelectronics, el mayor consumidor de MEMS del mundo, que recientemente desarrolló el giróscopo MEMS de alta calidad más pequeño del mundo, el L2G2IS, con un *package* de 2.3x2.3x0.7mm.
- ADI, que sacó al mercado la tercera generación de MEMS IMU ADIS16488A, con calidades muy altas en cuanto a parámetros de ruido se refiere. Se trata de una 10-DOF IMU completamente autónoma y con comunicación SPI.
- Sensonor, es el líder mundial en fabricación de giróscopos MEMS de alta calidad. Esta empresa sacó en 2014 el giróscopo de 3 ejes STIM210, que actualmente es el mejor del mundo, con un peso de 55 gramos y un Bias Instability de 0.5 º/h.

Las aplicaciones en las que estos giróscopos MEMS pueden ser utilizadas están divididas en tres grupos: aplicaciones de baja precisión, buenas para el consumidor donde prima el precio bajo y el tamaño pequeño (por ejemplo smartphones); aplicaciones de precisión media, generalmente utilizadas para fines industriales, donde se precisa un poco más de rango de temperatura que en las anteriores; y por último, aplicaciones de alta precisión, destinada generalmente al ámbito militar y aeroespacial [10].

En cuanto a la evolución futura de los giróscopos MEMS, es posible que sustituyan a los giróscopos de fibra

óptica en aplicaciones de nivel táctico (más adelante mostraremos qué es esto de nivel táctico en la tabla 5-1 y 5-2), y que la integración cada vez mayor, hará que en un futuro los acelerómetros, magnetómetros y giróscopos vayan integrados en un mismo módulo [10].

Figura 1-4. Giróscopo de Focault.

1.3.2 Filtro de Kalman

El filtro de Kalman fue por primera vez publicado en un paper en 1960 por Rudolf Emil Kálmán, y proponía una solución a problemas de filtrado lineal en tiempo discreto entre otros [11]. Desde su publicación, ha sido objeto de investigación por parte de muchos, de hecho, han surgido extensiones como el Extended Kalman Filter o el Unscented Kalman Filter. Las características del filtro hacen que sea una de las metodologías punteras en navegación, estando de este modo muy presente en casi todos los INS.

2. SISTEMAS DE NAVEGACIÓN INERCIAL

Computing is not about computers any more. It is about living.

- Nicholas Negroponte, founder and Chairman Emeritus of MIT's Media Lab -

2.1 Introducción

Como ya mencionamos anteriormente, los sistemas de navegación se componen principalmente de dos partes: GPS e INS.

La navegación por GPS permite obtener la posición de un objeto mediante comunicación satélite, aunque como vimos está sujeto a perturbaciones externas que limitan su precisión y fiabilidad. La otra parte de la navegación la compone los INS.

Al contrario que el GPS, los INS basan su funcionamiento en las medidas que sensores inerciales realizan, y no requiere de ninguna fuente de información externa. Son por ello sistemas autónomos, que no están tan limitados por perturbaciones externas, aunque sí están influidos por factores como la temperatura, de hecho, ese tipo de perturbaciones son las que hacen necesario que se realicen estudios como el llevado a cabo en este proyecto. El principal componente del INS es la IMU, que contiene los sensores inerciales. Dichos sensores son generalmente giróscopos, acelerómetros y, en algunos casos, magnetómetros.

Dependiendo de la disposición en la que se encuentre el INS respecto al dispositivo del que forma parte, (un móvil, un dron…) podemos clasificar los INS en dos tipos:

- Sistemas de plataforma estable o *gimbal systems.* En estos sistemas, el sistema inercial se encuentra en una plataforma que está aislada de toda rotación externa, es decir, que la plataforma está alineada con el sistema de referencia y no cambia su posición. Se encuentra en medio de un dispositivo con una mecánica similar a la del giróscopo que vimos en la figura 1-3, de forma que unos pequeños motores van moviendo el sistema para mantener el INS en la misma posición siempre [1].
- *Strapdown systems.* En este caso, el INS está fijo al dispositivo, y sus sensores inerciales miden las velocidades angulares y aceleración lineal teniendo en cuenta la posición actual del mismo (pitch, roll y yaw). Computacionalmente, este tipo de INS es más complejo que el anterior, aunque lo compensa con su menor tamaño y su mayor rigidez. Por ello, los *strapdown systems* son más utilizados que los *gimbal systems* [1].

2.2 La Unidad de Medidas Inerciales (IMU)

La Unidad de Medidas Inerciales o IMU, es el elemento más importante de un INS, ya que se compone de los sensores inerciales y, en la mayoría de los casos, de un pequeño microcontrolador que trata los datos de los sensores para acondicionarlos y sacarlos de la IMU en forma de voltaje. Los sensores inerciales que la componen son giróscopos, acelerómetros y en algunos casos magnetómetros.

2.2.1 Giróscopo

El giróscopo es un sensor que mide velocidades angulares en un eje concreto, las IMU suelen incorporar un giróscopo de tres ejes o tres giróscopos de un eje dispuestos ortogonalmente uno respecto al otro. Las medidas que realizan los giróscopos nos darán $\frac{medida\ angular}{unidad\ de\ tiempo}$, siendo en nuestro caso rad/s. En un sistema de navegación, los giros en los ejes cartesianos x, y, z corresponden a lo que llamamos roll, pitch, yaw, como podemos observar en la figura 2-1.

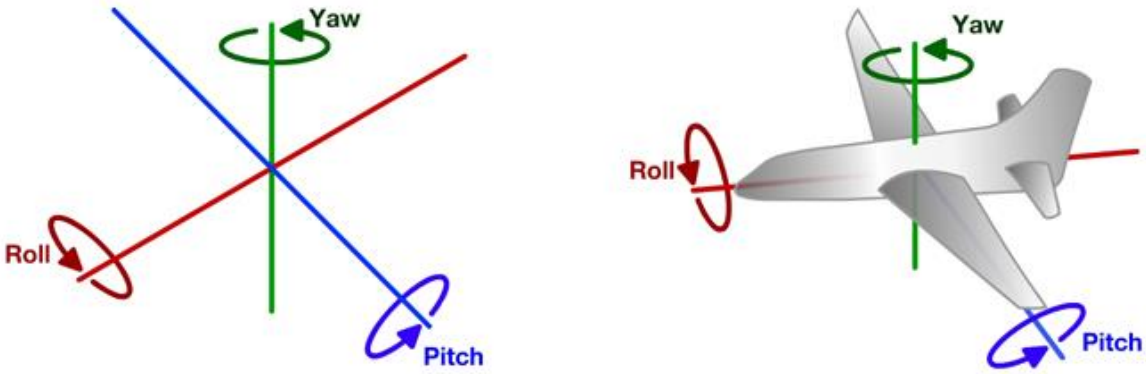


Figura 2-1. Roll, Pitch y Yaw

Vemos cómo se toma un avión como referencia, esto es así por el uso del giróscopo, muy extendido en navegación como hemos comentado anteriormente.

Éste es básicamente el concepto de giróscopo, un dispositivo que mide velocidades angulares, no obstante existen multitud de tipos de giróscopos y la medición de esta velocidad es diferente en cada caso. En el siguiente punto presentaremos algunos tipos de giróscopo [1], donde nos centraremos en el giróscopo MEMS vibratorio, que es el que la IMU que vamos a utilizar incluye. Algunas de estas categorías pueden ser mixtas, es decir, puede haber giróscopos ópticos que se fabriquen con tecnología MEMS. Una vía de investigación abierta es la de los MOEMS, que combina la tecnología MEMS con la óptica. Sustituye así los métodos capacitivos para medir pequeños desplazamientos, que limita la calidad del dispositivo, por métodos ópticos con mejores prestaciones [12].

2.2.1.1 Tipos de giróscopo

2.2.1.1.1 Mecánico

El giróscopo mecánico es el tradicional que ya mostrábamos en la figura 1-3, que consta de una estructura que le permite tener libertad de movimiento en los 3 ejes. Para obtener la medida se miden los ángulos entre los diferentes partes de la estructura. Este tipo de giróscopo ya está un poco obsoleto, ya que entre otros, tiene inconvenientes como que se compone de partes móviles que sufren un rozamiento entre ellas. Su tamaño además ha hecho que deje de ser práctico.

2.2.1.1.2 Óptico

Existen dos tipos fundamentales de giróscopo óptico: FOG y RLG. En el caso de los FOG, existe un cable de fibra óptica dispuesto de forma de aro, y a cada extremo del mismo se emite un rayo fino de luz. Cuando ambos rayos toman contacto, si ha habido desplazamiento angular, uno de los rayos recorre mayor camino que el otro. El resultado es una rayo combinado que tiene una intensidad directamente proporcional a la velocidad angular, a esto se le conoce como efecto Sagnac. En el caso de los RLG, el fundamento es el mismo, pero en lugar de utilizar fibra óptica, utiliza un sistema de espejos. Estos dispositivos tienen un nivel de precisión muy alto, de hecho, cuanto mayor sea, mayor precisión tendrá, lo que supone un problema cuando queremos reducir los dispositivos.

2.2.1.1.3 MEMS

La tecnología MEMS ha revolucionado la navegación, haciendo los dispositivos más pequeños y robustos. En

cuanto a los tipos de giróscopos MEMS, hay muchos, y una gran cantidad de combinaciones. En la figura 2-2 podemos observar algunas de las opciones a tener en cuenta a la hora de diseñar un giróscopo MEMS.

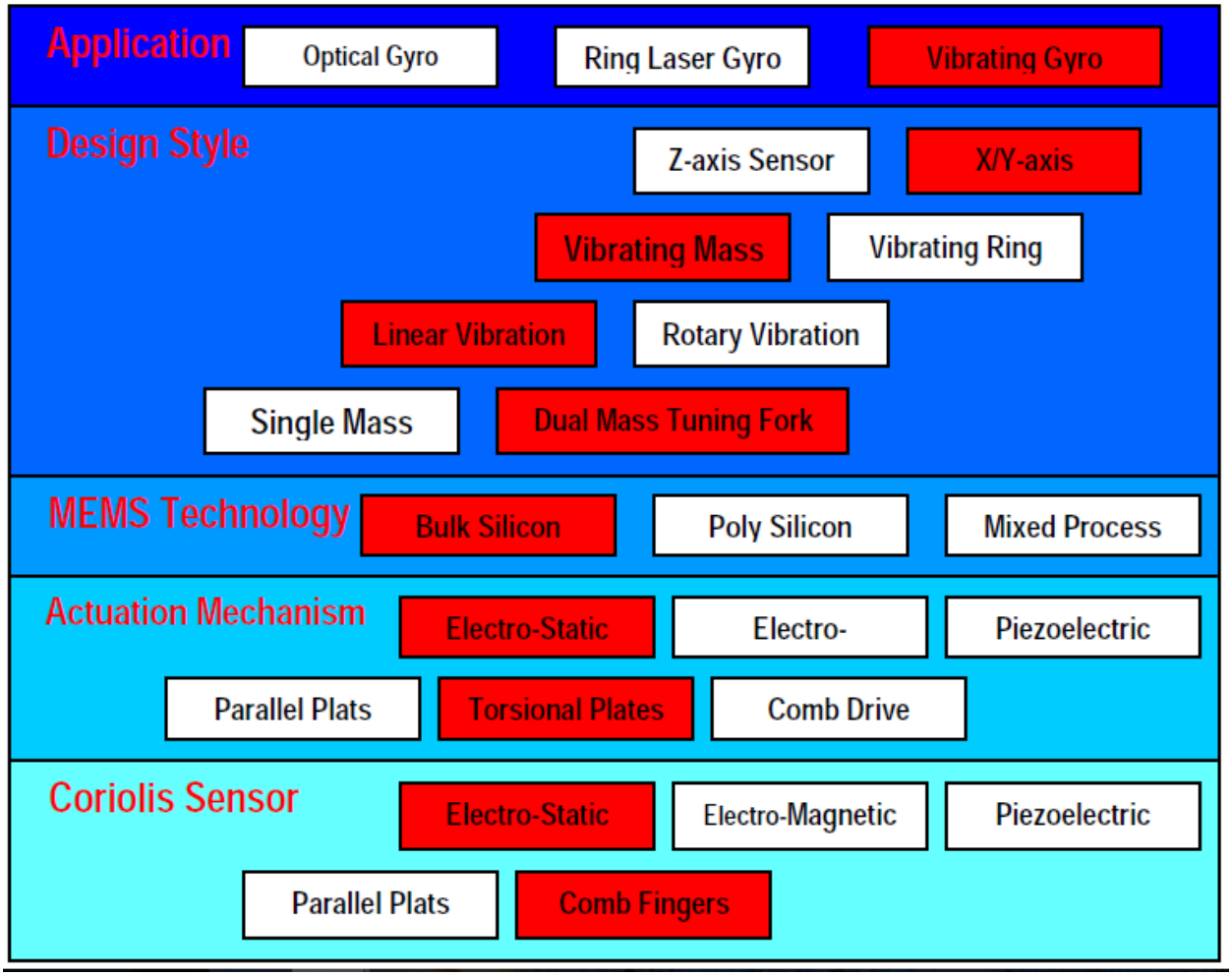


Figura 2-2. Consideraciones de diseño de un Giróscopo MEMS [13].

Nos centraremos en los vibratorios, ya que el giróscopo que utiliza nuestra IMU es vibratorio. No obstante, en el apartado de errores se incidirá más en los sensores de nuestra IMU. Los giróscopos MEMS vibratorios están compuestos por una o dos masas sujetas por una serie de partes relativamente flexibles que permiten que estas masas se muevan. El giróscopo está diseñado para vibrar en una dirección, y el giro sobre un eje hace que aparezca un componente vibratorio perpendicular a la vibración para la que está diseñado, el *drive mode*. Este segundo componente vibratorio aparece gracias al efecto Coriolis, fundamento del giróscopo vibratorio.

$$Fc = -2m(\omega \times v) \tag{2–1}$$

Donde Fc es el componente vibratorio que aparece, m la masa del giróscopo, ω la velocidad angular y v la velocidad a la que se mueve la masa. En la figura 2-3 podemos observar cómo sería un CVG[3] de una sola masa. También existe otro tipo de CVG, como son el Tuning Fork Gyroscope, que en lugar de tener una masa tiene dos que vibran en la misma dirección pero en sentidos opuestos. El Tuning Fork tiene dos modos de

[3] Coriolis Vibratory Gyroscope: Giróscopo Vibratorio.

funcionamiento, dependiendo de la dirección en la que las masas vibran, como explica la figura 2-4. Por último, otro tipo de CVG es el Vibrating-Wheel Gyroscope, que puede medir dos ejes con una única rueda, como por ejemplo vemos en la figura 2-5. Los giros en X y en Y hacen que exista un desplazamiento de la rueda y sea medido mediante condensadores.

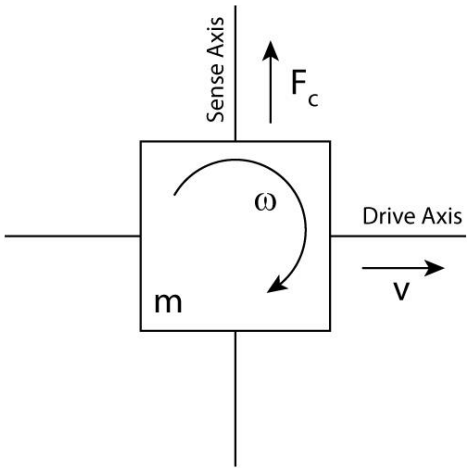


Figura 2-3. El efecto Coriolis explicado en un CVG de una sola masa.

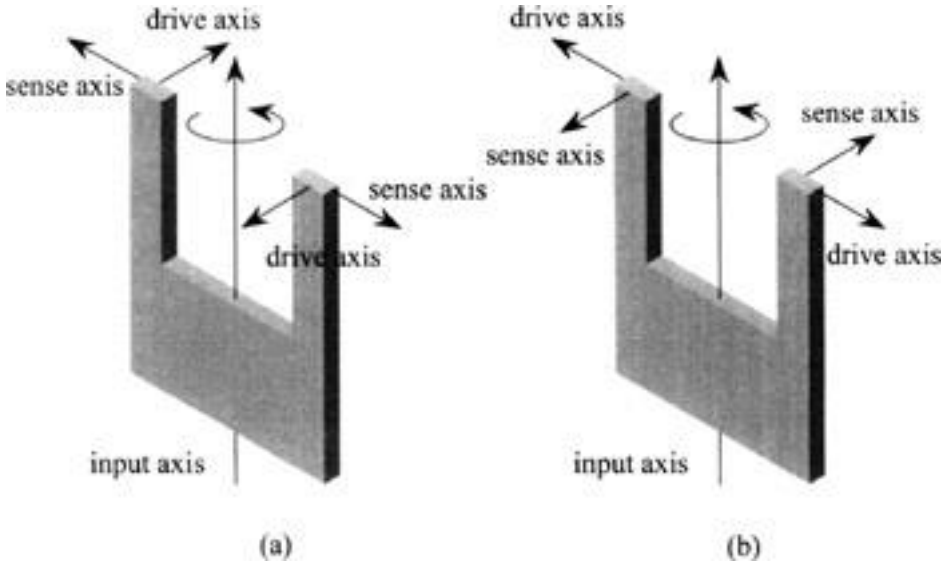


Figura 2-4. *Tuning Fork Gyroscope* [14]. (a) *out-of-plane driving* (b) *in-plane driving*.

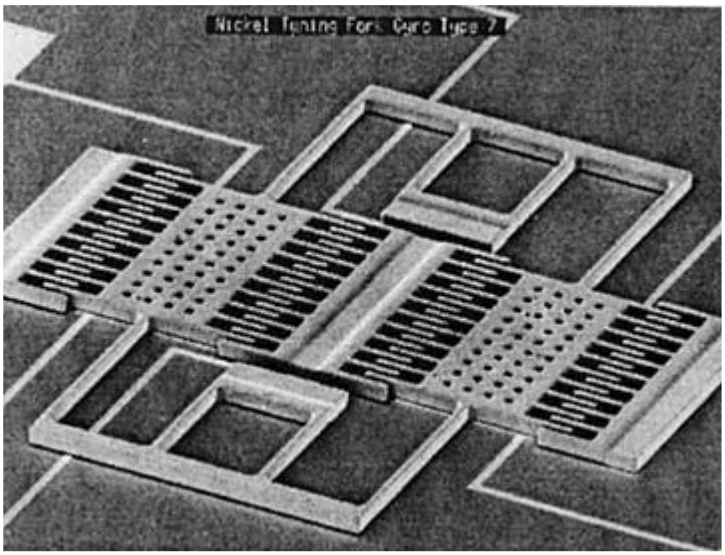


Figura 2-5. Primer prototipo de *Tuning Fork Gyro* del Draper Lab [13].

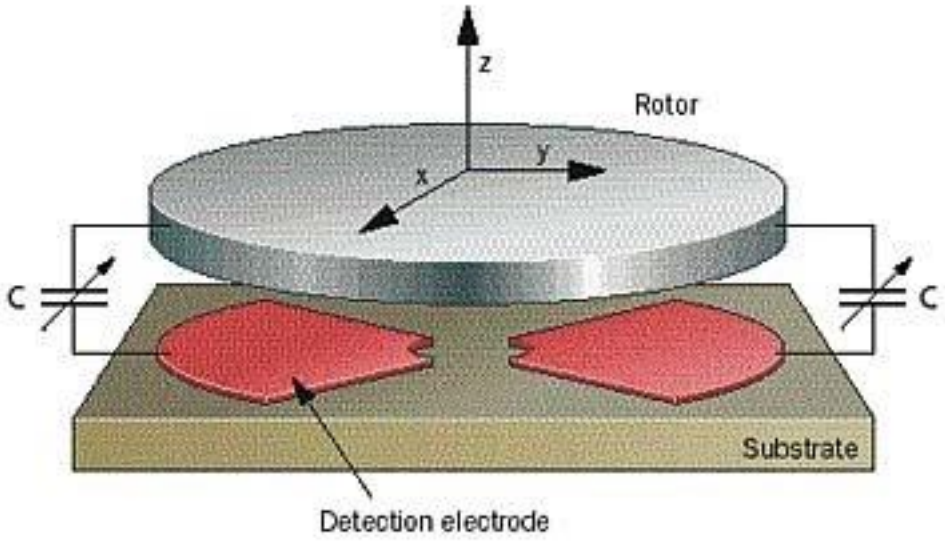


Figura 2-6. *Vibrating-Wheel Gyroscope* diseñado por Robert Bosch [13].

2.2.1.1.4 Cuántico

La tecnología va avanzando en el sentido de hacer todo más pequeño, tanto que dentro de unos años, la tecnología MEMS resultará demasiado grande para las aplicaciones que necesitaremos. Por eso, la tecnología cuántica hace su aparición para reducir los circuitos al nivel molecular, y no está exento de ello el giróscopo, que ya está empezando a desarrollarse en este campo. Richard Packard y sus colaboradores de la Universidad de California en Berkeley desarrollaron el primer giróscopo cuántico, que se basa en características de los superfluidos como el Helio-3 [15].

2.2.1.1.5 *Cryogenic nuclear magnetic resonance*

En los 1960s comenzaron las investigaciones sobre la Resonancia Magnética Nuclear (NMR) aplicadas al sensado de rotación angular. Una de las principales características de este giróscopo es que no tiene partes móviles, lo que lo hace muy robusto. Debido a su calidad y a su precisión a nivel atómico es potencialmente el sensor idóneo para la navegación inercial *strapdown.* Su funcionamiento se basa en la NMR, que es un efecto físico que trata de interacciones entre el núcleo de determinados elementos y el campo magnético externo [12].

2.2.2 Acelerómetro

El acelerómetro es el sensor encargado de medir aceleraciones lineales, y ha sido uno de los sensores que más éxito ha tenido a lo largo de los años. Al igual que los giróscopos, las IMU pueden incorporar un solo acelerómetro de tres ejes o tres acelerómetros de un eje ortogonalmente distribuidos. Las medidas que este sensor medirá serán $\frac{m}{s^2}$ siendo a veces coherente cambiar a g donde 1 g equivale a la fuerza de gravedad de la tierra.

2.2.2.1 Tipos de acelerómetro

2.2.2.1.1 Mecánico

Inicialmente, la idea del acelerómetro era de un dispositivo compuesto por muelles que sujetan una masa. La aceleración que afecta a esa masa hace que se desplace, y es este desplazamiento el que medimos para poder obtener la aceleración. Algunos tipos de acelerómetro mecánico son el *force-feedback pendulous accelerometer* o los acelerómetros de circuito abierto, como el de fibra óptica. Los acelerómetros de circuito abierto son menos estables y menos precisos que los de circuito cerrado, y están sujetos a muchas perturbaciones externas.

El funcionamiento del *force-feedback pendulous* se basa en que el movimiento del péndulo genera una alteración en un dispositivo pick-off[4]. En la figura 2-7 podemos ver el esquema de uno de estos acelerómetros. En el caso del acelerómetro de fibra óptica el funcionamiento es idéntico al anterior, con la diferencia de que por la forma del dispositivo pick-off en este caso y por las cualidades de la fibra óptica puede medir en dos ejes. El péndulo sería un cable de fibra y el dispositivo pick-off un panel foto-resistivo. El panel usado en la figura 2-8 es un *charge coupled imaging device* (CCID), adecuado para estos casos [12].

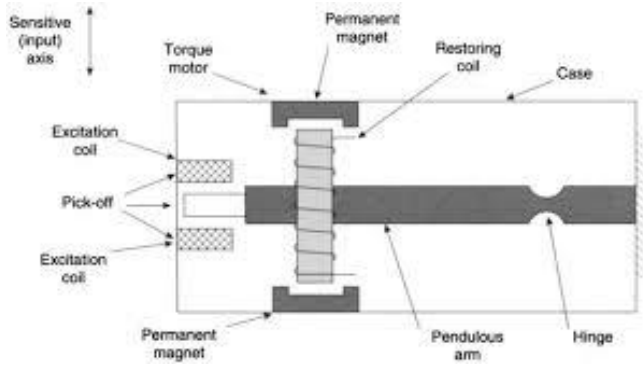


Figura 2-7. *Force-feedback pendulous accelerometer* [12].

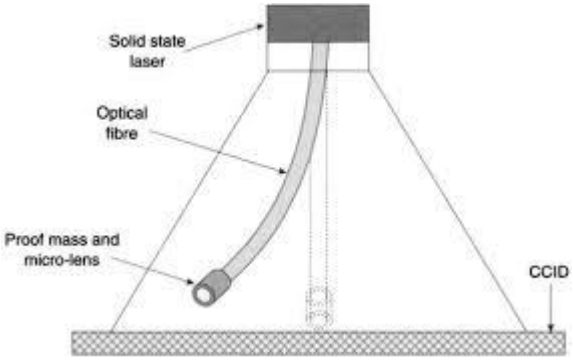


Figura 2-8. *Pendulous fibre optic accelerometer* [12].

[4] Dispositivo pick-off: dispositivo que emite o altera una salida eléctrica, óptica o neumática en respuesta a un movimiento.

2.2.2.1.2 *Solid-state* o de estado sólido

Los de estado sólido tienen gran repercusión en estos sensores, algunos de los tipos de acelerómetro más conocido forman parte de esta clase. Entre ellos se encuentran los *surface acoustic wave* (SAW), los de silicio, los de cuarzo y los de fibra óptica. Al igual que comentamos en los giróscopos, en los acelerómetros la clasificación de tipos es flexible, de ahí la aparición de la fibra óptica aquí. Pueden ser vibratorios de cuarzo, vibratorios de fibra óptica, fotoelásticos de fibra óptica… Un tipo característico es el SAW, que mostramos en la figura 2-9, que se basa en la variación de la frecuencia de una onda acústica superficial debido a la aceleración sufrida, que hace que se doble el dispositivo [12].

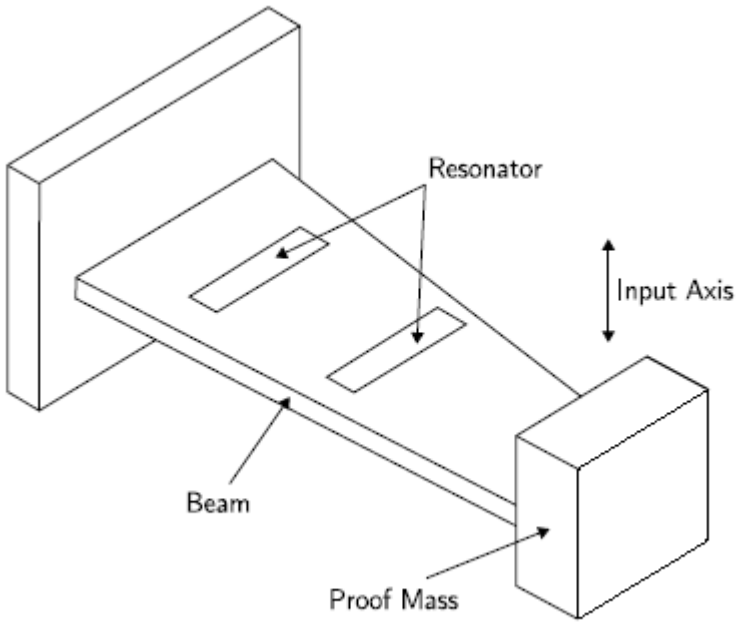


Figura 2-9. *SAW accelerometer* [12].

2.2.2.1.3 Angulares

No se ha invertido mucho de momento en este tipo de dispositivos, pero en lugar de medir aceleraciones lineales como el acelerómetro o velocidades angulares como el giróscopo, los acelerómetros angulares miden aceleraciones angulares, por ello nos darán $\frac{rad}{s^2}$ por ejemplo. Los más destacados son el *liquid rotor* y el *gas rotor*, que como sus propios nombres indican, se basan en líquidos y gases para realizar dichas medidas.

2.2.2.1.4 MEMS

Al igual que los giróscopos, en los acelerómetros ha ido creciendo el uso de tecnología MEMS, de hecho, algunos acelerómetros están hechos únicamente con silicio. Existen dos categorías en los acelerómetros MEMS, dependiendo de cómo se mide las aceleraciones: con desplazamiento de una masa o con el cambio de frecuencia de un elemento que vibra. La tecnología MEMS, como hemos visto, cumple los mismos principios que el resto de tipos de acelerómetros, siendo de este modo una versión reducida de los que hemos mencionado anteriormente. Los de tipo péndulo (desplazamiento de masa) tendrán una precisión menor que los que consten de una viga vibrante (cambio de frecuencia) [12].

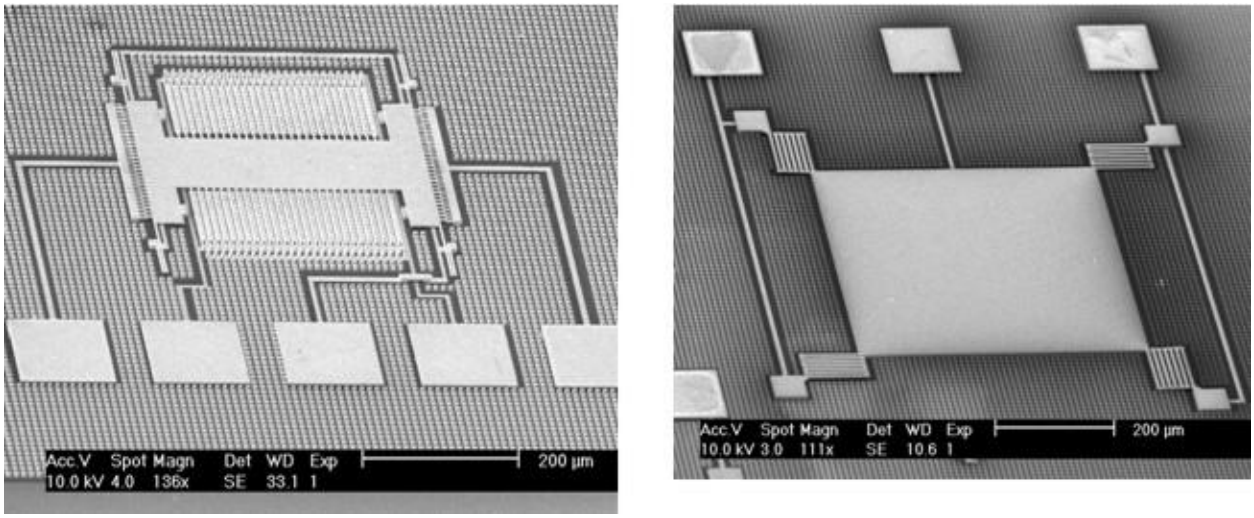


Figura 2-10. A la izquierda, acelerómetro *in-plane*; a la izquierda, acelerómetro *out-of-plane*. Ambos de tipo desplazamiento de masa [16].

2.2.3 Magnetómetro

El giróscopo y el acelerómetro son los componentes principales de una IMU, ya que son los sensores inerciales que habiualmente la componen. No obstante, con el paso de los años y el objetivo de mejorar la calidad de la IMU, los magnetómetros han ido incorporándose a la navegación inercial. Algunas IMU aún no lo incluyen, pero la mayoría si no lo incluyen, disponen de medios para añadir uno como periférico externo.

El magnetómetro, inventado por Carl Friedrich Gauss en 1833, puede ser de dos tipos: uno que mide la magnetización de un determinado objeto y otro que mide la dirección del campo magnético [17]. Los utilizados para la navegación son los segundos, usados como una simple brújula. Para aplicaciones en la superficie de la tierra, (para aplicaciones en el espacio no sería útil), mide el campo magnético de la misma, que como sabemos tiene unas líneas de campo que van de sur a norte magnéticos, por lo que simplemente señalará el norte. El magnetómetro es un dispositivo que está expuesto a muchas perturbaciones, más que los otros dos sensores, perturbaciones que comentaremos cuando estudiemos los errores que afectan a cada sensor, aunque este dispositivo no sea el objeto de nuestro estudio de errores tanto como el acelerómetro y sobre todo el giróscopo.

Existen multitud de tipos de magnetómetro, como por ejemplo los basados en tecnologías *Anisotropic Magnetoresistive* (AMR). El efecto AMR se basa en la variación de la resistencia eléctrica de un material dependiendo del ángulo que forma el campo magnético y una corriente eléctrica que circula por ese determinado material [18]. Honeywell, que goza de importancia en el sector de los magnetómetros, utiliza esta tecnología para fabricar algunos de sus dispositivos, como el HMC5883L [19].

3. FILTRO DE KALMAN

Let me say quite categorically that there is no such thing as a fuzzy concept... We do talk about fuzzy things but they are not scientific concepts. Some people in the past have discovered certain interesting things, formulated their findings in a non-fuzzy way, and therefore we have progressed in science.

- Rudolf E. Kálmán, 1972 -

3.1 Introducción

El ingeniero eléctrico, matemático e inventor de origen húngaro Rudolf Emil Kálmán publicó en 1960 un artículo que presentaba una solución recursiva a los problemas de filtrado lineal en tiempo discreto [20]. Surgió así el filtro que daba nombre a su creador, el filtro de Kalman, objeto de investigación en diversos campos, sobre todo en el de la navegación. Representado en el espacio de estados[5] [21], este filtro sirve para identificar un estado oculto, lo que llamaremos el vector observación. Es similar al observador de Luenberger, con la diferencia de que en el observador de Luenberger la ganancia K es fija y calculada previamente, mientras que en el filtro de Kalman la ganancia se va calculando de forma óptima en cada iteración [22]. Otra característica muy importante de este filtro es que debido a su algoritmo recursivo, no necesita mucha memoria para funcionar, por lo que puede ser implementado en dispositivos pequeños con pocas prestaciones, como una IMU de bajo coste.

3.2 El sistema

Se trata de un filtro predictivo, que tiene en cuenta las medidas pasadas y las actuales para realizar una estimación que posteriormente corrige. Además de predecir el estado oculto, sirve para sistemas sometidos a un ruido blanco gaussiano[6], o a ruidos que se puedan modelar como tal. En este trabajo se presupone que el lector tiene unos conocimientos básicos de estadística. Describimos un sistema con las siguientes ecuaciones:

Para el caso continuo:

$$\frac{d}{dt}x(t) = A(t)x(t) + B(t)u(t) + w(t) \tag{3–1}$$

[5] En ingeniería de control, una representación de espacios de estados es un modelo matemático de un sistema físico descrito mediante un conjunto de entradas, salidas y variables de estado relacionadas por ecuaciones diferenciales de primer orden que se combinan en una ecuación diferencial matricial de primer orden. Para prescindir del número de entradas, salidas y estados, las variables son expresadas como vectores y las ecuaciones algebraicas se escriben en forma matricial (esto último solo puede hacerse cuando el sistema dinámico es lineal e invariante en el tiempo).

[6] Ruido blanco gaussiano, en estadística, corresponde a un tipo de ruido cuya densidad espectral de potencia (PSD) es plana, es decir, en el dominio frecuencial su señal presenta la misma potencia para todas las frecuencias. En casos como el del ruido rosa o *flicker*, por ejemplo, la señal tiene más potencia a frecuencias más bajas, y va desapareciendo a medida que la frecuencia sube. Ambos desarrollados en el capítulo 6.

$$z(t) = H(t)x(t) + v(t) \tag{3–2}$$

Para el caso discreto:

$$x_k = A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + w_{k-1} \tag{3-3}$$

$$z_k = H_k x_k + v_k \tag{3-4}$$

Nos centraremos en el caso discreto, que es el más utilizado y el que vamos a utilizar en este proyecto. El subíndice *k* o *k-1* hace referencia a si esos valores se corresponden al instante k o al inmediatamente anterior. En estas ecuaciones, definimos sus componentes:

x es el vector de estado, el estado oculto del que hablábamos anteriormente.

u es la entrada de control del filtro.

z son las mediciones realizadas.

A es la matriz de transición de estados. Representa la relación entre el vector de estado y su valor anterior.

B es la matriz que relaciona el vector de estado con las entradas de control del filtro.

H es la matriz que indica la relación entre el vector de estados y las mediciones.

w y *v* son ruidos gaussianos de media cero[7] que afectan al sistema. *w* es el ruido asociado al proceso con matriz de covarianza *Q* y *v* es el ruido asociado a la medida, con matriz de covarianza *R*.

Para el caso continuo, las explicaciones para cada uno de los componentes son análogas al de su equivalente discreto. Definido este sistema en el que *x, u, z, w* y *v* son vectores y *A, B y H* son matrices, vamos a proceder al algoritmo del filtro. Para conocer un poco más a fondo el filtro puede recurrirse a una breve introducción en [11].

El objetivo de este filtro es, por tanto, estimar una serie de valores que componen el vector de estado que no pueden medirse directamente. Para conseguir esto, el algoritmo de filtro de Kalman tiene dos fases: la predicción y la corrección o actualización.

3.3 Algoritmo

En la fase de predicción, se realiza una estimación del vector de estado y la matriz de covarianza. La matriz de covarianza *P* es una matriz de *nxn* siendo *n* el número de elementos del vector de estado, que tiene en su diagonal las varianzas de dichos elementos y en el resto de posiciones la relación que estos elementos tienen unos con otros. El KF trata los elementos del vector de estado de forma probabilística, esto es, que considera que cada uno está distribuido como un proceso gaussiano donde tiene una media (su valor en el vector) y una varianza que hemos comentado.

En la fase de actualización, se realizan las medidas, y se obtiene un término al que haremos referencia con la letra *y,* que representa la diferencia entre las medidas realizadas y el valor que, partiendo de la estimación del vector de estado, debería ser. Con esas medidas se procede al cálculo de la ganancia de Kalman. Como comentamos anteriormente, el carácter dinámico de esta es lo que lo diferencia del observador de Luenberger. Para finalizar, se actualiza los valores del vector de estado y de la matriz de covarianza. El filtro tiene en cuenta la característica estadística de las medidas, proponiendo una actualización del vector de estado *x* y de la matriz

[7] Se especifica que tiene media cero porque los ruidos gaussianos tienen dos factores que los definen, la media y la varianza. La distribución normal o gaussiana

de covarianza *P* que es óptima si el sistema es, como es el caso, lineal gaussiano.

Para comprender el algoritmo recursivo del filtro, podemos observar la figura 3-1, donde se muestra un esquema. Las ecuaciones que vienen a continuación describen ambas fases. El superíndice – hace referencia a un valor estimado, por ejemplo, x_{k-1}^- haría referencia al valor de x estimado en el instante *k-1*.

- **Predicción**

$$x_k^- = Ax_{k-1} + Bu_{k-1} \tag{3-5}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{3-6}$$

- **Corrección o actualización**

$$y_k = z_k - Hx_k^- \tag{3-7}$$

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{3-8}$$

$$x_k = x_k^- + K_k y_k \tag{3-9}$$

$$P_k = (I - K_k H)P_k^- \tag{3-10}$$

Los valores de Q y R hacen referencia a las varianzas de los ruidos asociados al proceso y medidas respectivamente. Son matrices diagonales con las varianzas de estos elementos en su diagonal. Estos valores pueden variar con el tiempo, y tendrá que considerarse a la hora de calcular la matriz de covarianza, ya que pueden influir en ella. Los efectos que R y Q pueden tener respecto P son los siguientes:

- Q es la covarianza del ruido asociado al proceso y contribuye a la incertidumbre general del filtro. Cuando Q es mayor, el filtro es capaz de responder a grandes cambios en los datos más rápidamente.
- R es la covarianza del ruido asociado a la medida y alude a la cantidad de medida que utiliza el filtro. Si R es mayor, es decir, las varianzas de la medida son mayores, el filtro confía menos en las medidas, ya que son menos precisas. Si R es menor se tiene más en cuenta los valores de las medidas, que al tener menor varianza, son más precisas.

En principio las matrices H, B y A no varían, y la linealidad del filtro hace que su obtención sea simple y rápida. No obstante, como veremos en el siguiente apartado, el filtro tiene versiones donde trata sistemas no lineales.

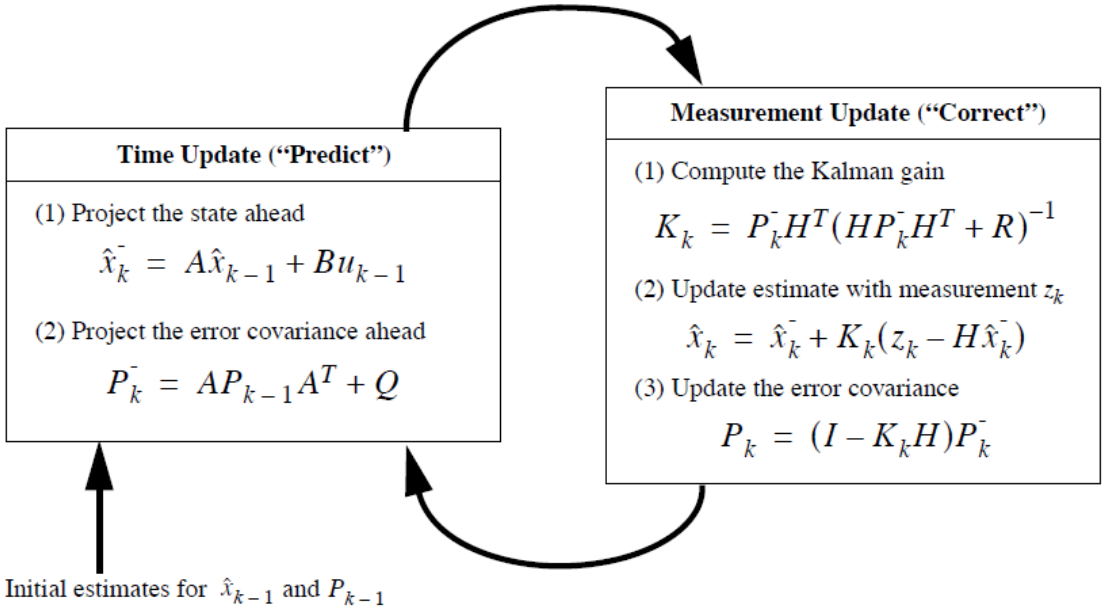


Figura 3-1. Algoritmo de filtro de Kalman. El vector de estado tiene un acento circunflejo (^) porque realmente es una estimación del vector de estado [11].

3.4 Filtro de Kalman Extendido (EKF)

A veces, nos encontramos con sistemas que no son lineales y el filtro dejaría de ser tan potente si un problema así hiciese imposible su uso. Las ecuaciones del EKF siguen la misma estructura que el KF, y el algoritmo es el mismo, pero debido al carácter no lineal del sistema, tendremos que linealizar algunas matrices calculando su matriz jacobiana. Para comenzar empezaremos definiendo el sistema con el que nos encontramos ahora.

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}) \tag{3-11}$$

$$z_k = h(x_k, v_k) \tag{3-12}$$

A diferencia de (3-3) y (3-4), en este sistema las relaciones entre sus elementos no son lineales, y se relacionan mediante las funciones *f* y *h* que muestran nuestras ecuaciones. Los vectores tienen el mismo significado, ya que este filtro no es más que una extensión del anterior. Evidentemente, no conocemos el valor de *w* y de *v,* pero suponemos que son gaussianos y aditivos, de ahí que en esa posición aparezca un 0 en las ecuaciones que veremos más adelante.

Las matrices A, *B* y *H* ya no son válidas, y habrá que linealizarlas calculando el jacobiano de las funciones que hemos definido en (3-11) y (3-12) respecto al vector que relacionaba a dichas matrices con el vector a la izquierda de la igualdad. Después de modificar las ecuaciones del filtro, obtenemos:

- **Predicción**

$$x_k^- = f(x_{k-1}, u_{k-1}, 0) \tag{3-13}$$

$$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \tag{3-14}$$

- **Corrección o actualización**

$$y_k = z_k - h(x_k^-, 0) \tag{3-15}$$

$$K_k = P_k^- H_k^T \left(H_k P_k^- H_k^T + V_k R_k V_k^T\right)^{-1} \tag{3-16}$$

$$x_k = x_k^- + K_k y_k \tag{3-17}$$

$$P_k = (I - K_k H_k) P_k^- \tag{3-18}$$

Existen algunas matrices como *V* o *W* que aparecen nuevas, y otras como A y H que incorporan subíndices. Esto es así porque al ser jacobianos tienen un valor diferente en cada iteración. Las matrices V y W surgen de suponer los ruidos gaussianos aditivos en el sistema no lineal, de forma que en este caso, *w* y *v* se distribuyen gaussianamente de la siguiente manera[8]:

$$w_k \sim N(0, WQ_kW^T) \tag{3-19}$$

$$v_k \sim N(0, VR_kV^T) \tag{3-20}$$

Donde las matrices W, V, H y A se definen[9]:

$$A = \frac{\partial f}{\partial x}(x_{k-1}, u_{k-1}, 0) \tag{3-21}$$

[8] Esta notación estadística significa que la variable w_k está distribuida como una variable normal o gaussiana con media 0 y varianza WQ_kW^T.

[9] La notación utilizada en las ecuacionces (3-21) hasta (3-24) alude al cálculo de una matriz jacobiana donde cada función de f, que es un vector de funciones, se deriva en función de x y se sustituyen los valores entre paréntesis.

$$W = \frac{\partial f}{\partial w}(x_{k-1}, u_{k-1}, 0) \quad (3\text{-}22)$$

$$H = \frac{\partial h}{\partial x}(x_k^-, 0) \quad (3\text{-}23)$$

$$V = \frac{\partial h}{\partial v}(x_k^-, 0) \quad (3\text{-}24)$$

De este modo, tenemos definido el EKF, cuyo algoritmo definimos en la figura 3-2. En ocasiones, la linealidad o no linealidad del sistema se encontrará en un término medio, de forma que al aplicar los jacobianos, encontraremos algunas relaciones lineales por ejemplo, obteniendo V o W igual a la matriz identidad, expresada en este capítulo como *I*. De este modo, si miramos las ecuaciones, podremos observar que si una de estas dos matrices es identidad, la podemos omitir sin problema. Las ecuaciones del EKF han sido sacadas de [11].

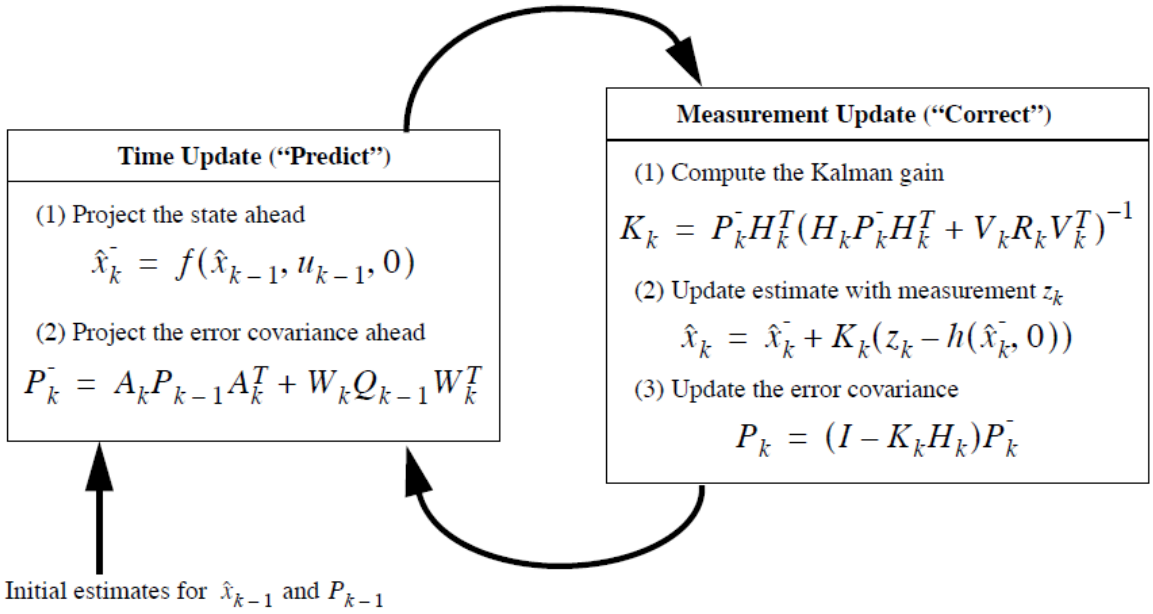


Figura 3-2. Algoritmo del filtro extendido de Kalman [11].

Para determinadas aplicaciones, el EKF se queda corto, ya que necesita que el sistema sea casi lineal. Además es de difícil implementación y de calibrar. Como solución a esto, existe una extensión del KF aún más exhaustiva que el EKF, y es el UKF [23], que es más preciso, más fácil de implementar y tiene un rango más alto.

4. DESCRIPCIÓN DEL SISTEMA

Technology is nothing. What's important is that you have a faith in people, that they're basically good and smart, and if you give them tools, they'll do wonderful things with them.

- Steve Jobs -

4.1 El Sistema

En este capítulo se procederá a presentar el sistema a utilizar, describiendo cada una de las tecnologías y dispositivos que han formado parte del proyecto. La base de todo el proyecto es la ArduIMU V3, más concretamente, la MPU-6000 que contiene, ya que en ella se integran el giróscopo y el acelerómetro, y es quien se comunica con el magnetómetro. Se puede ver el esquemático completo del ArduIMU v3 en [24]. No nos centraremos en la placa como tal, ya que el interés está en los sensores y en los errores que sufren, por lo que nos centraremos más en la MPU-6000 y en las características de los sensores. Conectaremos la placa al ordenador, donde utilizaremos Matlab y C++ para implementar el filtro de Kalman correspondiente y hacer el análisis de Allan Variance (ver anexo B). El filtro que utilizaremos será la versión extendida del KF, el EKF, debido a las no linealidades que aparecen en el sistema que vamos a tratar.

4.2 ArduIMU V3

Como hemos comentado, el componente más importante de todo nuestro sistema es el PCB[10], y el que usaremos en este caso será el ArduIMU v3. El ArduIMU v3 fue desarrollado por Jordi Muñoz, un desarrollador que pertenece a DIYDRONES, una comunidad enfocada en el desarrollo de UAVs. En noviembre de 2011, Jordi publicó un post en el que anunciaba dicha ArduIMU [25]. Presentaba la versión 3, mucho más pequeña y más eficiente en cuanto a espacio se refiere, además de incluir muchas más mejoras respecto a su antecesora, la versión 2. Entre las características más significativas de la ArduIMU v3 están:

- Incluye MPU-6000, con giróscopo de 3 ejes y acelerómetro de 3 ejes.
- Procesador Atmega328 funcionando a 16Mhz.
- Filtro de paso bajo configurable por usuario.
- Plataforma MotionApps™ soportada por Android, Linux y Windows.
- Sensor de temperatura de salida digital.
- Entrada digital en el pin FSYNC que soporta GPS y estabilizador de imagen[11] de video.

[10] Printed Circuit Board, en español Circuito Impreso o Placa de Circuito Impreso

[11] Estabilizador de imagen: dispositivo encargado de reducir las vibraciones en la captura de una imagen o video cuando se producen movimientos indeseados.

- Puerto I2C[12] a 3.3V.
- 6 pines analógicos.
- 3 leds de estado (RGB).
- Pines compatibles con la ArduIMU v2.
- Tamaño 1.5” x 1.0” (o 3.81 x 2.54 centímetros)[13].
- Corriente de alimentación para el chip completo en modo inactivo : 5µA
- Todos los componentes incluidos en la placa son SMD (dispositivo de montaje superficial).

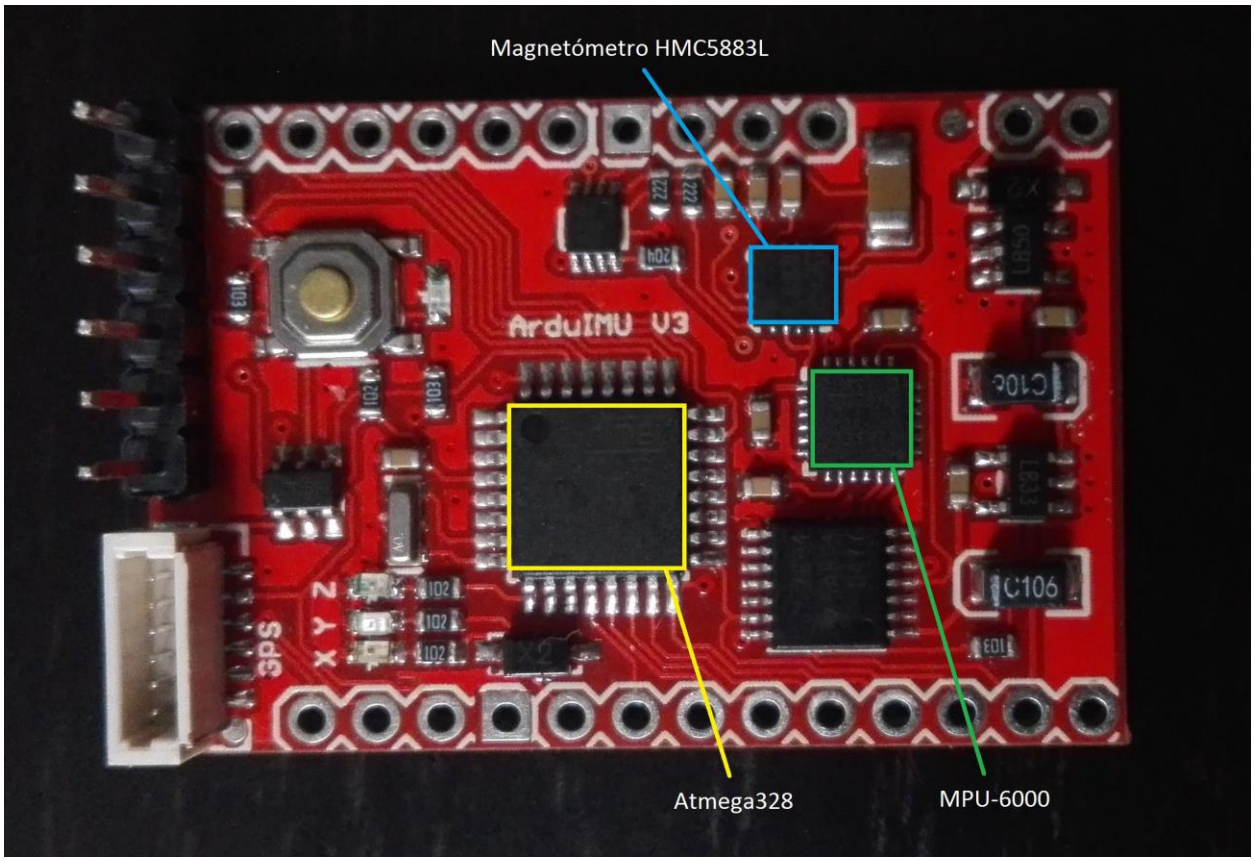


Figura 4-1. ArduIMU v3.

[12] I2C (Inter-Integrated Circuit), bus de datos serial basado en maestros y esclavos utilizado generalment en comunicación en el interior de la placa para comunicar periféricos y microcontroladores.
[13] El símbolo “representa pulgadas.

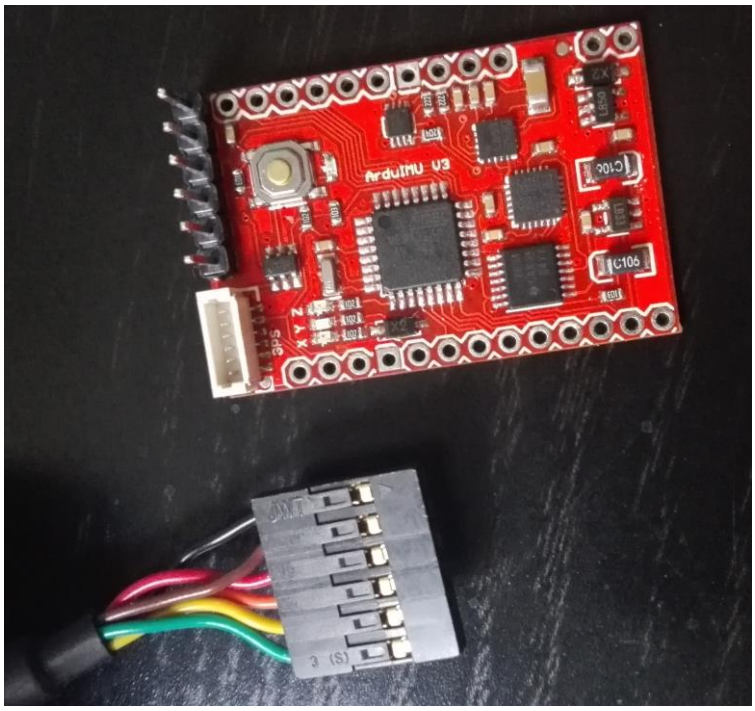


Figura 4-2. ArduIMU v3 con el cable correspondiente.

Nos centraremos ahora en los componentes que forman la ArduIMU.

4.2.1 Atmega328

El Atmega328 es un microcontrolador de 8 bits de bajo consumo desarrollado por Atmel. Este microcontrolador se encargará de comunicarse con la MPU-6000 y hará que toda la placa funcione. Podemos encontrar sus características en su datasheet [26]. En nuestra ArduIMU estaremos trabajando, como bien dijimos antes, a 16 Mhz, aunque los datos saldrán de la placa a una frecuencia mucho más baja, como veremos más adelante. Algunas de sus características son las siguientes.

- Voltaje entre 1.8 y 5.5 Voltios. No obstante, como estamos trabajando a 16 Mhz, el rango que utilizaremos será 4.5 – 5-5 Voltios.
- Rango de temperatura desde -40 a +85ºC.
- Interfaz SPI[14] maestro/esclavo. Como veremos, la MPU-6000 dispone, a diferencia de la MPU-6050, de puerto SPI, por lo que esta será una posible vía de comunicación.
- Arquitectura de instrucciones RISC.

4.2.2 Magnetómetro HMC5883L

El magnetómetro HMC5883L de 3 ejes es un dispositivo de montaje superficial diseñado para el sensado de campo magnético débil orientado a aplicaciones en las que sirve de brújula [19].

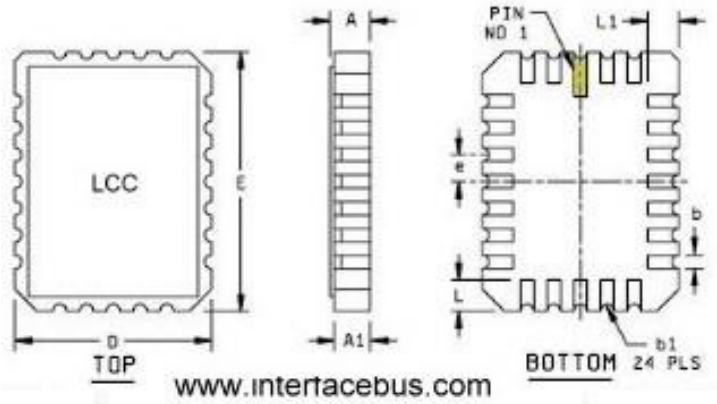
El HMC5883L dispone de muchas prestaciones, entra las que destacan la gran resolución que tiene la familia HMC118X (de mili-gauss a 8 gauss), compensación programable de las variaciones de temperatura que sufre el dispositivo y un convertidor Analógico-Digital (ADC) de 12 bits que permite que la medida tenga una calidad de 1 o 2 grados. Además de esto, soporta I2C y lo utilizará para comunicarse con la MPU, complementando al giróscopo y al acelerómetro. Podemos ver cómo se realizaría la interconexión entre magnetómetro y MPU-6000 en la figura 4-4. Honeywell utiliza la tecnología AMR para este dispositivo, como ya vimos en la descripción

[14] SPI (Serial Peripheral Interface), es un estándar de comunicaciones usado principalmente para la transferencia de información entre circuitos integrados y equipos electrónicos.

del magnetómetro como sensor en el capítulo 2. Estos sensores anisotrópicos tienen una gran precisión debido a su construcción de estado sólido (*solid-state*) y puede medir tanto la dirección como la intensidad del campo magnético de la Tierra, que será lo que midamos.

Este trio de sensores, cuando disponen de una corriente de alimentación, convierten el campo magnético en el eje en el que miden en un voltaje proporcional a la intensidad del mismo. Estos sensores están hechos de una capa muy fina de una aleación magnética formada por níquel y hierro llamada Permalloy, dispuesta en forma de puente entre dos puntos. Con la presencia de campo magnético, la resistividad del Permalloy cambia, cambiando así la diferencia de potencial entre los puntos a los extremos del puente. Una combinación de estos sensores, formarán el magnetómetro de 3 ejes.

En cuanto al dispositivo, físicamente hablando, es un módulo de montaje superficial con unas medidas de 3.0x3.0x0.9mm con 16 pines y un *package* LCC[15].



Dimensions				
	Inches		Millimeters	
Symbol	Min	Max	Min	Max
A	.062	.120	1.57	3.05
A1	.050	.066	1.27	1.68
b	.020	.030	0.50	0.76
b1	.008 R		0.20 R	
D	.292	.308	7.41	7.82
E	.392	.408	9.95	10.36
e	.045	.055	1.14	1.39
L	.040	.050	1.01	1.27
L1	.040	.050	1.01	1.27
N	24		24	

Figura 4-3. Dimensiones del *package* LCC [27].

[15] Leadless Chip Carrier, tipo de package con unas dimensiones determinadas que se pueden ver en la figura 4-3.

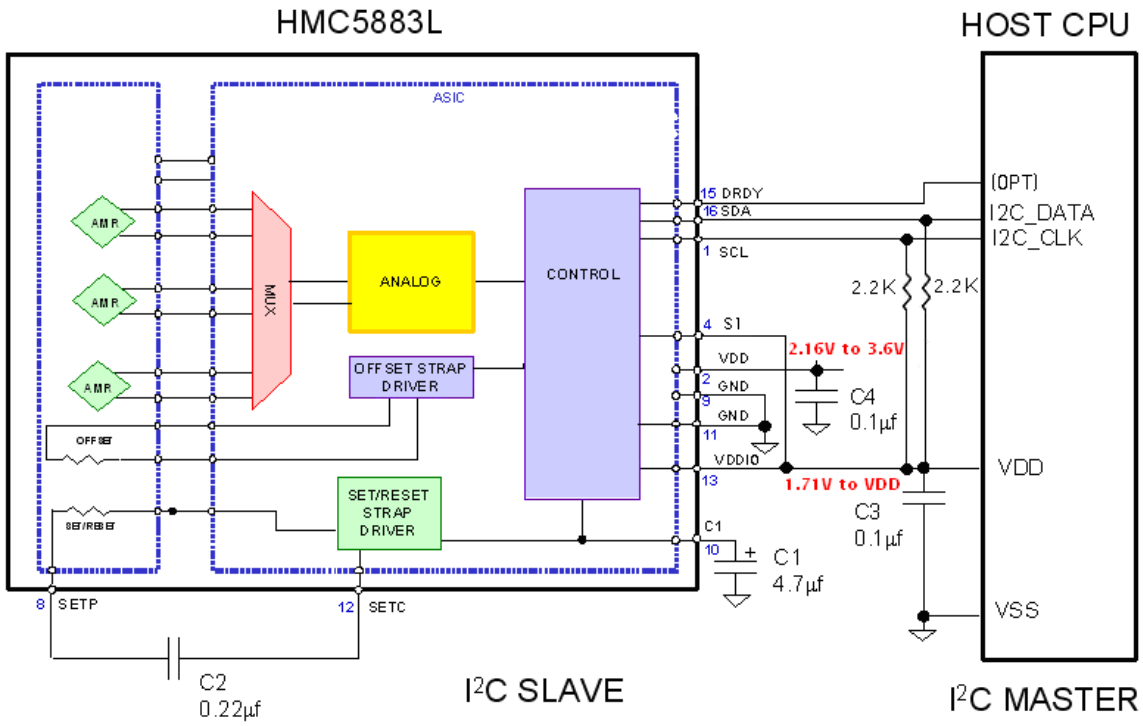


Figura 4-4. Conexión I2C entre el magnetómetro HMC5883L y la MPU [19].

Algunas otras características del magnetómetro son las siguientes.

- Operaciones de bajo voltaje (2.16 a 3.6V) y de baja potencia (100µA en modo medida).
- Rango de -8 a +8 gauss.
- Salida a 160 hz como máximo (más adelante veremos que esto es un problema).
- Rango de temperatura -30 a +85ºC.
- Compensación de los errores debidos a temperatura (ver página 20 de datasheet).

4.2.3 MPU-6000

Los dispositivos MPU son los primeros dispositivos que elimina la necesidad del *package* individual de sus sensores y reduce los errores de *misalignment* [28]. Estos dispositivos están compuestos por 3 giróscopos, 3 acelerómetros integrados en un dado con tecnología MEMS-CMOS [29] y una DMP que se encarga de procesar los datos. Además, esta DMP puede también procesar datos de un dispositivo externo como un magnetómetro, teniendo así una medición de 9 ejes, es decir, una 9-DOF IMU. La comunicación con este dispositivo externo se realiza a través de un puerto I2C, en el que, como vimos en la figura 4-4, el dispositivo externo es el esclavo y el maestro es la MPU [30], al revés que ocurre cuando la MPU se comunica con el Atmega328, donde la MPU sería el esclavo.

Figura 4-5. Esquema de interconexión de la MPU [31].

En nuestro sistema, el papel de IMU está desempeñado por la MPU-6000, de InvenSense, que pertenece a la familia MPU-60x0 junto con la MPU-6050. Las diferencias entre la MPU-6000 y la MPU-6050 son dos: la MPU-6000 dispone de un puerto SPI además del I2C que tiene la MPU-6050 y en cuanto al voltaje de alimentación, la MPU-6000 tiene un único pin VDD mientras que la MPU-6050 dispone del pin VDD para la parte analógica y de un pin VLOGIC para la parte digital. El diagrama de bloques completo de la MPU-6000 lo podemos ver en la figura 4-7.

La MPU dispone de una memoria FIFO que le permite tener un consumo bajo debido a que el procesador puede leer los datos de la FIFO a ráfagas y permite que la MPU entre en un estado de bajo consumo mientras recolecta más datos. Existen muchas opciones para el tema de bajo consumo en este dispositivo, en el datasheet [30] recogen todas las posibilidades, donde podemos ver que puede haber momentos en los que se necesite menos corriente porque sólo esté funcionando el giróscopo, o sólo el acelerómetro, o el DMP y uno de los sensores…

Las características fundamentales de la MPU-6000 son las siguientes:

- DMP con capacidad de procesamiento de algoritmos para 9 ejes.
- Bus auxiliar I2C a 400khz y bus SPI a 1Mhz, aunque es posible utilizarlo a 20Mhz.
- 6 ADCs para digitalizar las señales de los giróscopos y los acelerómetros.
- Memoria on-chip FIFO[16] de 1024 Bytes.
- Sensor de temperatura interno.
- *Package* QFN (Figura 4-6) de 4x4x0.9mm de los más pequeños del mercado, que además permite alto rendimiento, bajo nivel de ruido y bajo coste de *packaging*.
- Tolerancia a golpes de 10.000 g.
- Filtro paso-baja programable para los sensores.
- Alimentación 2.375-3.46V.

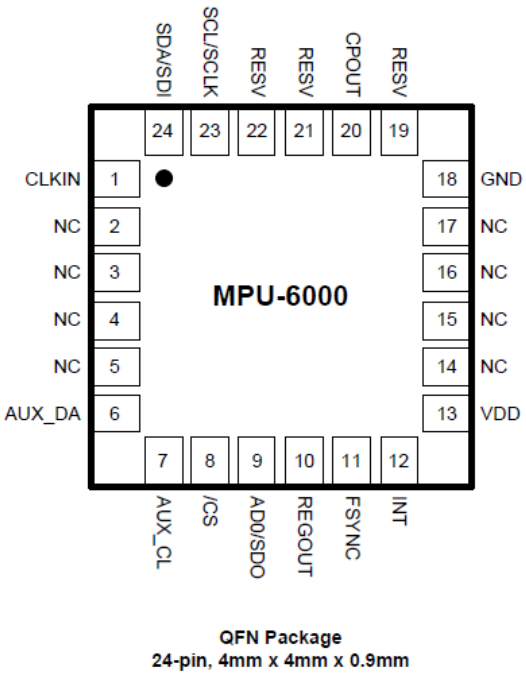


Figura 4-6. MPU-6000 *package* [30].

[16] FIFO (First In First Out), hace referencia a un tipo de memoria en la que sus datos está estructurados de manera que los primeros en entrar son los primeros en salir. Lo contrario sería LIFO (Last In First Out), donde los últimos datos en entrar en la memoria serían los primeros en salir.

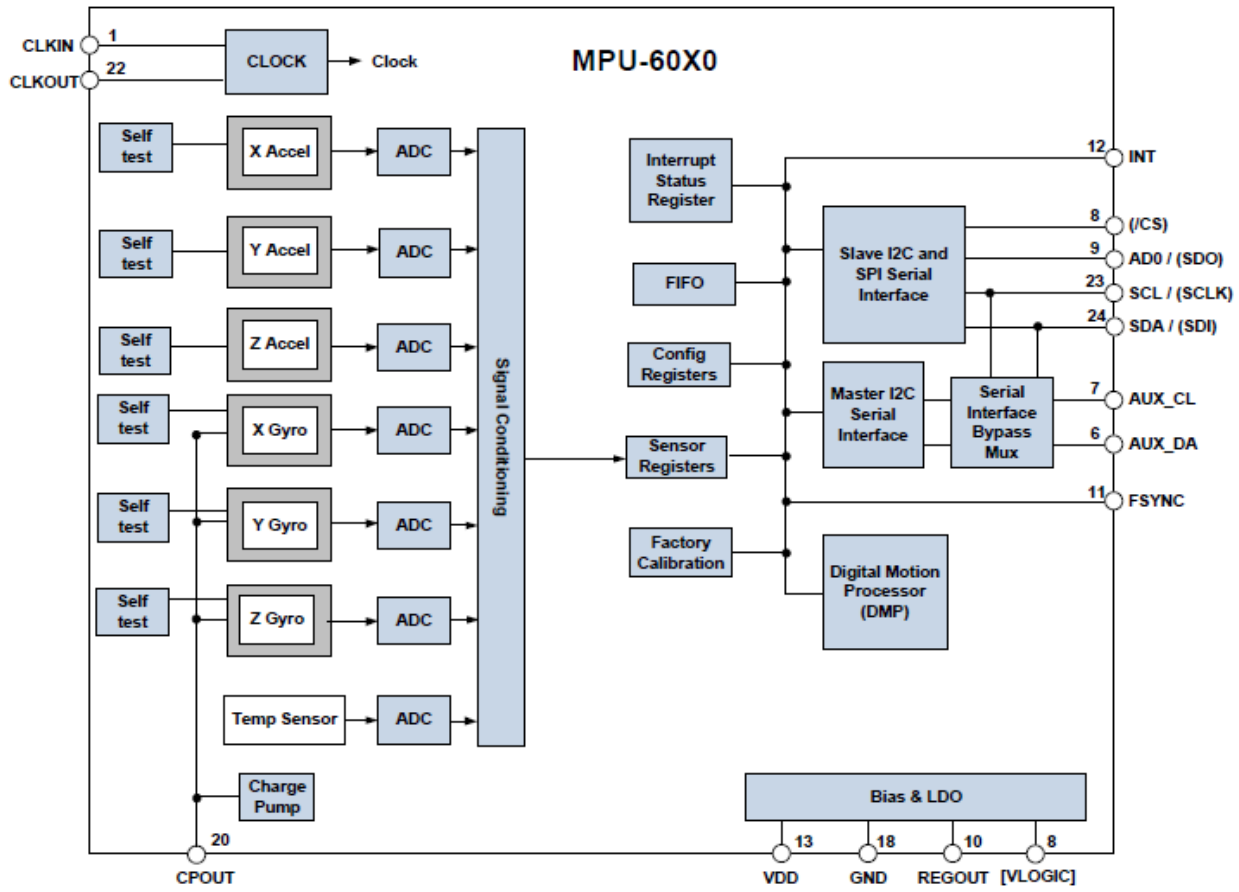


Figura 4-7. Diagrama de bloques de la MPU-6000 [30].

4.2.3.1 Giróscopo

La MPU-6000 incluye 3 giróscopos vibratorios que, basándose en la fórmula (2-1) del efecto Coriolis, utiliza condensadores para medir dicha fuerza. Este resultado capacitivo es acondicionado para obtener el voltaje que representará la velocidad angular [30]. Sin embargo, estas capacidades son muy pequeñas, ya que una variación de un grado por segundo generaría unos pocos aF, por ello, el tratamiento del ruido en este dispositivo es crítico. Este voltaje es convertido por los ADCs para conseguir las medidas que utilizaremos, utilizando una tasa de muestreo de 200hz.

En el tutorial sobre sensores inerciales MEMS de InvenSense [29] se define muy bien el giróscopo y el acelerómetro MEMS igual que las compensaciones que se tienen en cuenta. Es necesario implementar una serie de filtros sobre todo en el caso del giróscopo, que es de una complejidad mucho mayor que el acelerómetro mecánicamente hablando.

Otras características del giróscopo son las siguientes:

- Filtro de paso baja programable por el usuario.
- Rango programable a ±250, ±500, ±1000 y ±2000 º/s.
- Los ADCs permiten que el sampleo de las medidas del giróscopo se realice simultáneamente.
- Mejorada la variación del bias y de la sensibilidad con la temperatura, mejora la eficiencia de un modelo de compensación.
- Corriente de trabajo: 3.6mA.
- Corriente en modo *Standby*: 5µA.
- Calibrado para compensación del *Scale Factor*.

4.2.3.2 Acelerómetro

Los acelerómetros que contiene la MPU-6000 son 3 ortogonales que miden aceleraciones, y su funcionamiento es mucho más sencillo que el del giróscopo. Se trata de una masa sujetada por muelles que se desplaza con la aceleración, cuyo desplazamiento es medido por unos condensadores.

En [29] se explica detalladamente los procesos de medida y cómo realiza estos MEMS InvenSense, el propietario de dicha MPU. Se comenta también las fuentes de error que afectan a este dispositivo, y la serie de compensaciones que se realiza. Al igual que el giróscopo, existe una influencia de la temperatura en estos sensores, ya que, el bias que les afecta y su variación, depende de la temperatura. Se intenta que esta variación del bias con la temperatura sea lo mínima posible, de hecho, y esto es muy importante, **se intenta que la variación del bias con la temperatura sea lo más lineal posible, para que el modelo de compensación de la temperatura de primer orden que se aplique sea efectivo.** Como ya comentamos a lo largo del proyecto, realizaremos un modelo de compensación de la temperatura en el desarrollo del mismo.

Otras características [30]del acelerómetro son:

- Rango programable a ±2, ±4, ±8 y ±16 g.
- Los ADCs permiten el funcionamiento sin el uso de un multiplexor externo.
- Corriente de trabajo: 500μA.
- Diferentes corrientes para modo de bajo consumo: 10μA a 1.25hz, 20μA a 5hz, 60μA a 20hz, 110μA a 40hz.
- Interrupción por exceso de aceleración.
- Interrupciones programables por el usuario.

4.2.3.3 Fabricación MEMS-CMOS

Como bien se explica detalladamente en [29], InvenSense utiliza tecnología MEMS-CMOS para fabricar sus sensores, tecnología en la que mezcla los MEMS que ya hemos estado comentando y la tecnología que utilizan los transistores CMOS[17]. Con esta fusión InvenSense pretende conseguir mejorar aún las prestaciones de sus dispositivos de bajo coste, manteniendo su bajo coste y mejorando los productos. Una de las ventajas que presenta es que, según veremos en la fabricación, los dispositivos se encuentran en un hueco aislado herméticamente, por lo que reducen enormemente las perturbaciones externas. Cuando se une la oblea[18] de silicio que utilizan los MEMS y la de los CMOS, se crea el vacío. Además, si no se interconectasen directamente, la parte MEMS tendría que unirse a los CMOS mediante cableado, por lo que aumentarían las capacidades parásitas y el consumo del circuito. El hecho de sobreponer una oblea con otra hace que se utilice más espacio útil, y se ocupe menos área, de ahí que podamos utilizar el *package* QFN de 4x4x09mm tan pequeño.

En el proceso que viene detallado en [29], un proceso de 5 máscaras se define para la oblea de los MEMS.

1. Primero, se alinean las obleas de los MEMS y la de los CMOS para establecer las medidas con ayuda de unas máscaras. Se realiza después un hueco en la oblea de los MEMS (la llamaremos oblea 1).
2. Después, la oblea 1 se unirá a una oblea 2, utilizando un aislante entre ambas, formando así una oblea con tecnología *silicon-on-insulator*[19]. Obtenemos de esta forma un *engine silicon-on-insulator*, formado por ambas obleas y el aislante que las separa. La oblea 2 se escogerá del grosor del que queramos que sean nuestros dispositivos, ya que esta capa será la que les de forma.
3. Se realizan luego una serie de moldeos de esta capa, con técnicas como el *etching*[20], más concretamente el DRIE. Este proceso es un híbrido entre *Physical Sputtering* y *Plasma Etching*. En el primero, que es un proceso físico, se utiliza el bombardeo de iones para eliminación de material, mientras que en el

[17] La tecnología CMOS es una de las más utilizadas en circuitos integrados. Su "producto estrella" es el transistor CMOS, utilizado en todo lo que incumbe a la electrónica digital, como en puertas lógicas o en memorias.
[18] En microelectrónica, una oblea es una fina plancha de material semiconductor, como por ejemplo silicio, sobre la que se construyen microcircuitos con técnicas de dopado, grabado químico y deposición de materiales, con la ayuda, en ocasiones, de máscaras.
[19] SOI. Tecnología en la que se sustituye la oblea de silicio por un "sándwich" de capas semiconductor-aislante-semiconductor.
[20] Eliminación de parte del material de la oblea con el uso de bombardeo de iones, algún tipo de ácido u otros métodos.

segundo, que es un proceso químico, se utilizan ácidos o gases. Realmente, la fusión de ambos procesos sería el RIE, que combina ambos procesos, pero en este caso utilizaremos el DRIE, que consiste en realizar ataques RIE uno sobre otro, consiguiendo así más profundidad y paredes de la cavidad más rectas [32]. Sin embargo, este tipo de ataque puede crear irregularidades en la superficie en forma de pequeños escalones, como vemos en la figura 4-9 [32].

4. Para finalizar, la ya completa oblea MEMS se une a la CMOS mediante una unión eutéctica de aluminio y germanio (Al-Ge). Esta unión se hace a baja presión para conseguir el sellado al vacío. La oblea de CMOS se fabrica antes de hacer la unión y si es necesario, se hace un hueco para que la cavidad donde van a ir los dispositivos sea mayor (en este caso sería un proceso de 6 capas). Después de todo, se eliminan los materiales que sobren y el resultado es el que mostramos en la figura 4-8.

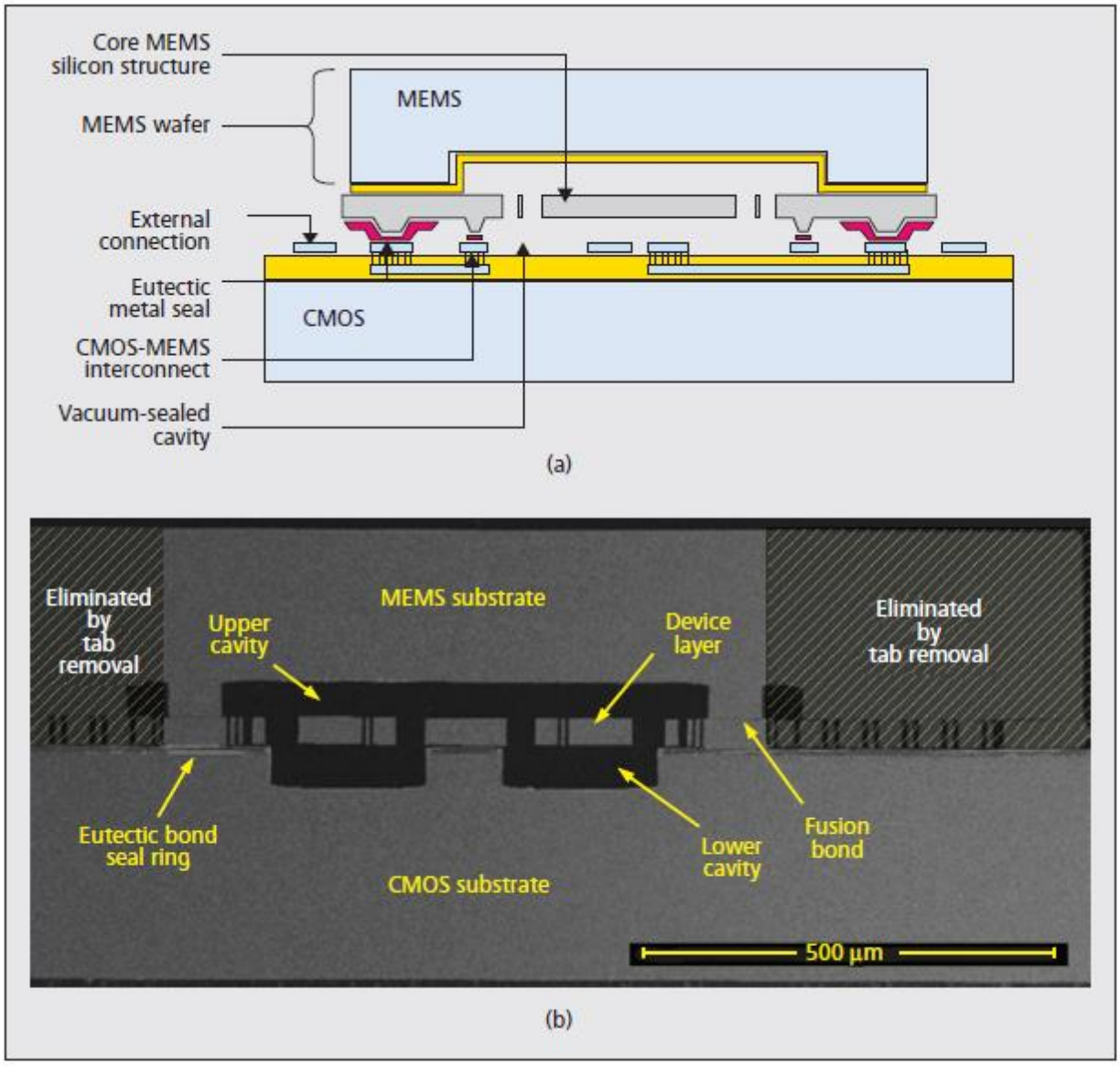


Figura 4-8. Sensor fabricado con tecnología MEMS-CMOS [29]. a) capas del proceso de fabricación y resultado. b) corte vertical de un dispositivo real.

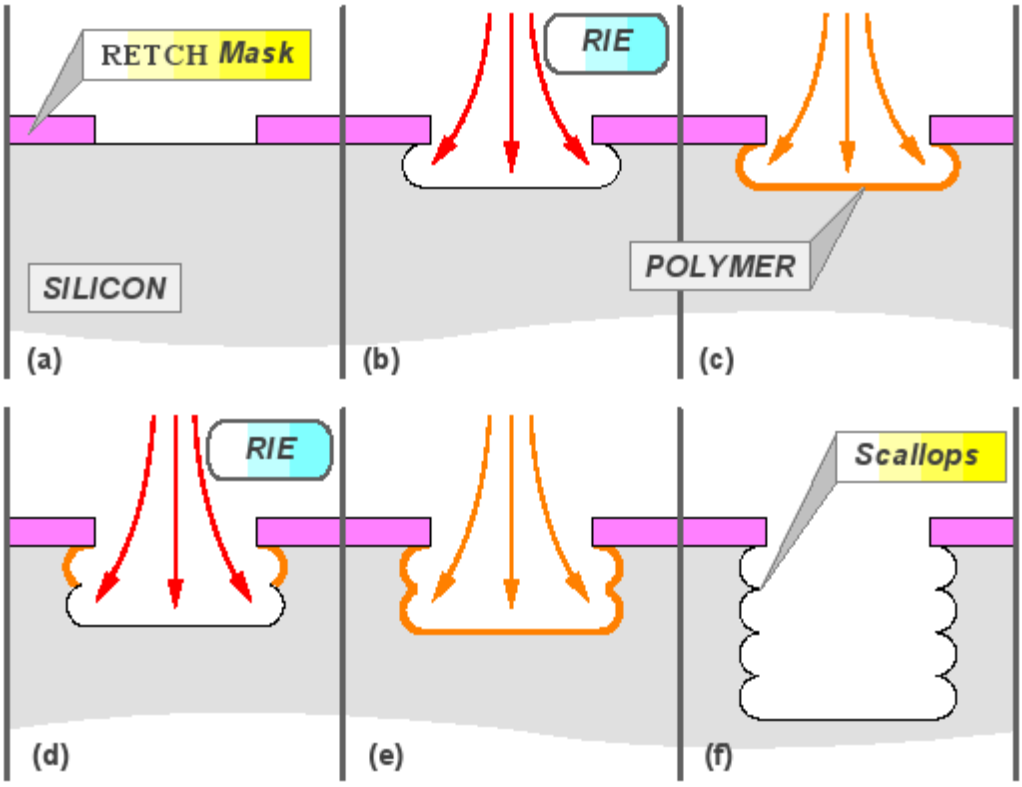


Figura 4-9. Técnica DRIE [32].

4.3 Extended Kalman Filter

El uso de la versión extendida del filtro de Kalman se debe a la presencia de no linealidades en las relaciones entre las diferentes entradas, salidas y vectores de observación que se definen en el mismo. Antes de explicar por qué es necesario utilizar la versión extendida del filtro, vamos a definir qué vamos a considerar entradas y qué vamos a considerar salidas. En un sistema, siguiendo las ecuaciones (3-3) y (3-4) y las definiciones de los componentes posteriormente desarrolladas en el capítulo 3, tenemos una entrada de control, u , un vector observación, x, una medida, z, unos ruidos w y v que afectan al proceso y a la medida respectivamente, y una serie de matrices que los relaciona. En nuestro sistema, tenemos las medidas de un giróscopo, un magnetómetro y un acelerómetro como entradas y tenemos como valores que nos gustaría saber, es decir como salida, la orientación del dispositivo y los errores que tienen los sensores. En el tema de errores profundizaremos en el siguiente capítulo.

Hemos determinado que utilizaremos las tres medidas del giróscopo en los tres ejes, como entrada de control, y las medidas de acelerómetro y magnetómetro como vector de medidas. En el vector observación tendremos el roll, pitch y yaw, acompañados de las velocidades de giro y de los errores tanto del giróscopo como del acelerómetro. No vamos a tener en cuenta los errores del magnetómetro, ya que es un sensor que al medir campo magnético puede estar influido por muchas perturbaciones como edificios hechos de materiales ferrosos y otro tipo de estructuras que puedan modificar el campo magnético significativamente en un punto.

De este modo tenemos definidos los parámetros de nuestro sistema, un vector u de 3 elementos, un vector x de 12 y un vector z de 6, aunque veremos en el desarrollo del proyecto que finalmente, al no necesitar el componente en z del magnetómetro, lo que hacemos es obtener un vector z de 5 elementos.

Las relaciones entre el valor del vector observación y su valor anterior están definidas por la matriz A, y las relaciones entre el valor de la entrada de control y del vector observación, por la matriz B. Parece obvio que la relación entre un valor de x y su valor anterior tengan una relación lineal, de hecho, los únicos elementos que son diferentes de 1, 0 o -1 son los que hacen referencia a la relación entre la velocidad de giro y los ángulos de Euler, cuyo valor será igual al tiempo de muestreo, ya que calculamos el valor de la posición multiplicando el

tiempo de muestreo la velocidad. En las ecuaciones que definen estas matrices observaremos que esta relación sí es lineal, por lo que para obtener las matrices A y B no hará falta linealizarlas.

No podemos decir lo mismo de la relación entre las medidas y el vector de estado, definida por la matriz H. Las medidas del acelerómetro y magnetómetro no se relacionan directamente con el vector de observación, salta a la vista, por ello, tenemos que linealizar el vector h(x) para obtener la matriz H. Hay dos formas de hacerlo, pero ambas empiezan con un paso común y es calcular la relación entre las medidas del acelerómetro y magnetómetro por separado y los ángulos de Euler. Solo lo hacemos en función de los ángulos de Euler porque el resto de valores no interaccionan con las mediciones, excepto, por supuesto, el bias[21] o error del acelerómetro. Una vez calculado $h_a(x)$ y $h_m(x)$, siendo los subíndices indicativos de acelerómetro y magnetómetro respectivamente, puede o bien juntarse en una función h(x) y después linealizar, o bien linealizar por separado y luego juntarlo. El resultado es el mismo, pero nosotros optaremos por el primer método, para después poder hacer también la linealización del vector V que si recordamos era la asociada al ruido asociado a la medida.

Como hemos visto, existen no linealidades que deben ser calculadas, por tanto, tendremos que utilizar el EKF, y para linealizar calcularemos matrices jacobianas.

4.4 Lenguajes de programación

4.4.1 Matlab

Matlab es una herramienta de software matemático que ofrece un IDE (entorno de desarrollo integrado) con un lenguaje de programación propio, el lenguaje M. Actualmente está disponible para Unix, Windows, Mac y GNU [33].

El hecho de que sea un software matemático hace que resulte muy útil en programación cuando utilizamos matrices, en C++, por ejemplo, para tratar con matrices he tenido que descargar una librería aparte. Además, el lenguaje ofrecido por Matlab es de alto nivel, de modo que podemos utilizarlo sin tener en cuenta registros u otro tipo de información que debemos conocer cuando programamos en lenguajes de bajo nivel, centrándonos así en la resolución de problemas.

El IDE que presenta Matlab es muy intuitiva y tremendamente útil a la hora de depurar código, de modo que en este trabajo, además de realizar la implementación del filtro de Kalman en C++, lo he hecho en Matlab de forma paralela para comparar resultados y para poder depurar algún tipo de error en Matlab, ya que es más rápido y sencillo.

Las labores que he realizado en Matlab han sido el análisis de Allan Variance, la implementación del filtro de Kalman y la impresión de gráficas para comparar los resultados obtenidos.

4.4.2 C++

C++ es un lenguaje de programación diseñado por Bjarne Stroustrup en 1980 [34]. Lo que hace que este lenguaje sea uno de los que considero más importantes de la actualidad es que combina la potencia que tiene el lenguaje C y la versatilidad de la que dispone un lenguaje orientado a objetos como Java. Lo que hace de C ser tan potente es que, a pesar de ser de medio nivel, puede operar a nivel de bit como los lenguajes de bajo nivel y utilizar estructuras complejas típicas de los lenguajes de alto nivel. La programación orientada a objetos, por otra parte, aporta a este lenguaje la posibilidad de definir clases y otras características que hacen que lenguajes como Java, sean muy útiles a la hora de desarrollar multitud de tareas.

[21] En este proyecto utilizaremos bias para referirnos al error que afecta a los giróscopos y acelerómetro, suponiendo que el giróscopo tiene además una varianza y el bias también tiene su propia varianza.

En este trabajo, he utilizado este lenguaje para realizar la implementación del KF, incluyendo el modelado de errores correspondiente. El programa está compilado con el compilador g++ versión 4.8.4. y lo he llevado a cabo en un entorno Linux (Ubuntu 14.04.3), utilizando como editor de textos el simple pero eficiente gedit.

Pese a todas las ventajas que he comentado anteriormente sobre C++, no dispone por sí mismo de operadores ni clases matriciales por defecto. Esto no ha sido problema alguno, ya que existe una gran cantidad de librerías de código abierto disponibles para el usuario. En este caso he utilizado la librería armadillo [35]. Para tomar la decisión de qué librería escoger y qué librerías complementarias instalar, observé comparaciones en diversas webs y finalmente, me decidí por usar armadillo + openBLAS (versión 0.2.18) [36].

5. ESTUDIO DE ERRORES

But slight mistakes accumulate, and grow to gross errors if unchecked.

- Jacqueline Carey -

5.1 Errores en sensores inerciales

Como hemos visto, existen técnicas de compensación de errores y de filtrado dentro de la propia IMU, lo que hace que algunos de dichos errores estén muy disminuidos y no requieran la necesidad de realizar un modelo del mismo. Sin embargo, es necesario conocer los tipos de errores que afectan a nuestra IMU, ya que algunos sí deberán ser compensados. En este punto, presentaremos algunos de los tipos más importantes de errores que afectan a las IMU y después nos centraremos en los modelos de giróscopo y de acelerómetro que muestren dichos errores.

Básicamente, los errores que afectan a una IMU se pueden agrupar en dos grandes grupos: estocásticos y deterministas. Como se explicó previamente en el capítulo dedicado a los filtros de Kalman, vamos a trabajar en un espacio de estados. Dicho esto, podemos decir que un error determinista es aquel en el que no interviene la aleatoriedad, tal que para un valor de estado, producirá siempre el mismo valor de estado siguiente. En cambio, un error estocástico es, por definición, no determinista, por lo que hará referencia a las componentes aleatorias de los errores que afectan a los sensores. En términos probabilísticos, y a grandes rasgos, podríamos decir que los errores deterministas representan un valor medio y los estocásticos una varianza.

5.1.1 Deterministas

5.1.1.1 Scale factor

El factor de escala es un error en la relación entre entrada y salida. Algunos artículos consideran que el factor de escala debe ser un número en torno a 1, siendo 1 un sistema no afectado por el factor de escala, mientras que otros consideran que debe ser un valor cercano a 0, es decir, este valor anterior, menos 1. Particularmente considero que la segunda opción es más correcta, por lo que en los modelos posteriores será esta la que utilicemos. Generalmente, se expresará en porcentaje, de forma que un *scale factor* de 4% sería 1,04 (o 0,04) En términos matemáticos, el factor de escala está multiplicado por la entrada (o 1+factor de escala), de forma que no afecta si vale 1. Podemos observarlo en la ecuación (5-1), donde S es el scale factor. Esta ecuación está expresada en la figura 5-1. En ocasiones, existe una componente no lineal en el factor de escala, aunque nunca suele ser significativo y los fabricantes suelen incluirlo cuando hacen referencia al factor de escala. El *scale factor*, cuando no está compensado, es un error muy significante y su aparición se puede deber a diversos factores. Un análisis del impacto que tienen los materiales de ensamblado en un giróscopo MEMS es mostrado en [37], centrándose en el *scale factor.*

$$x = Sf(x) + b \tag{5-1}$$

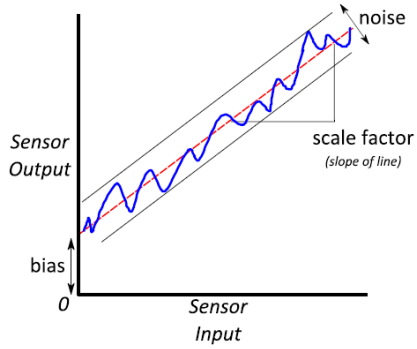


Figura 5-1. Algunos errores de los sistemas inerciales [38].

5.1.1.2 Misalignment

Este error se debe a que los sensores que se supone que deberían estar dispuestos ortogonalmente entre sí no lo están por problemas de fabricación. Es difícil que formen perfectamente un ángulo de 90 grados, y esto hay que compensarlo metiendo una serie de factores que afectan a la relación entre las medidas en diferentes ejes, se expresará, como veremos más adelante, como una matriz con los elementos de la diagonal igual a 0 [38].

5.1.1.3 Bias

El bias será el error más característico de estos sensores, tanto es así que puede reducirse gran parte del modelado de errores a tener un sensor, su varianza y un bias con la suya. El bias es un valor de continua que se añade al valor que estamos midiendo, sería algo así como un offset. Es independiente de la entrada por lo que podría definirse como lo que se mide en ausencia de entrada [12]. En la (5-1) está representado con la letra b y en la figura 6-1 también se muestra. Existe un error denominado *turn-on to turn-on bias variation* que hace referencia a la variación que sufre el bias entre cada encendido del dispositivo. Esto es coherente ya que al iniciar una IMU se deben realizar varias instalaciones y puestas en marcha que podrían variar este valor. No obstante, aunque lo mencione aquí, ese error es estocástico, ya que se expresa como una varianza. Conocido el valor que tenía el bias al apagar el dispositivo, se conoce un rango de valores entre los que se encontrará cuando dicho dispositivo se encienda de nuevo. Esta varianza que genera incertidumbre en el bias es el *turn-on to turn-on bias variation*.

5.1.1.4 G-Sensitivity

En los giróscopos, a pesar de que no miden la aceleración lineal, existe cierta dependencia con ella, debido principalmente a la masa que forma el giróscopo. Esto es más común en los MEMS, sobre todo en los que utilizan una masa en movimiento, como los CVG. Dicha relación se puede expresar con un factor que multiplica a la aceleración que sufre, modelando de este modo este ruido [12].

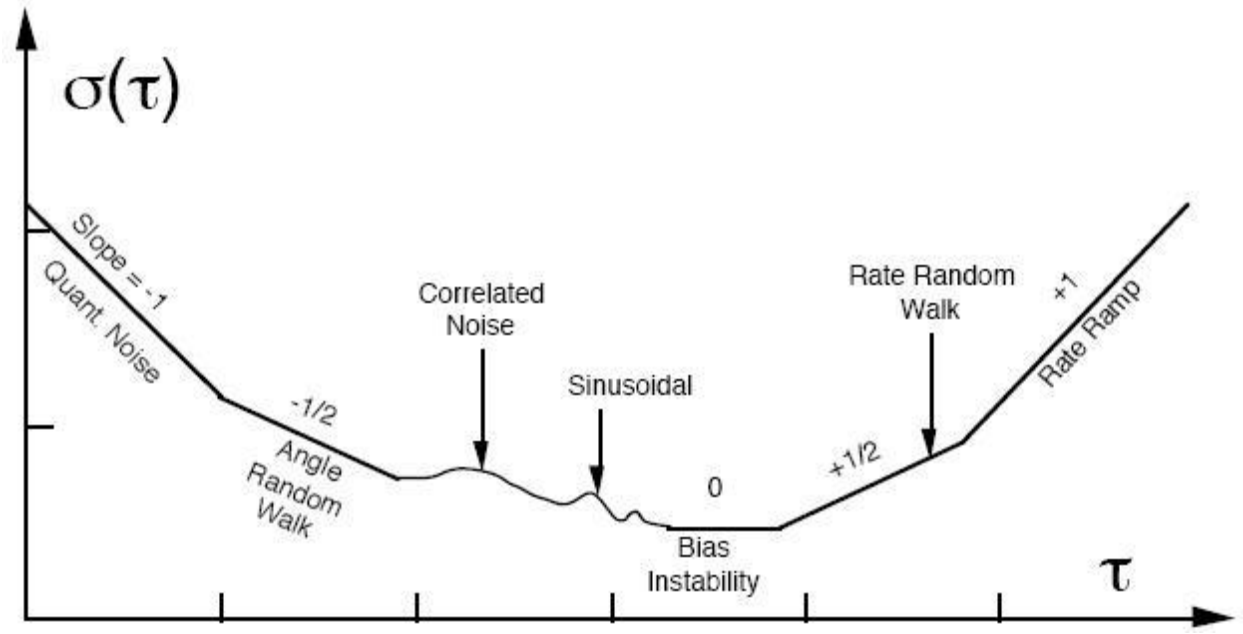
5.1.1.5 Efectos de la temperatura

La temperatura es un factor que influye en gran parte de los errores tanto estocásticos como deterministas. Podemos poner como ejemplos los valores de nuestra IMU para entender dichas variaciones [30]. El factor de escala varía un 0.2% y el bias también varía hasta 20 º/s con la temperatura. El bias no es un valor fijo, sino que va variando con el tiempo, y con la temperatura esta variación puede cambiar, de forma que la relación entre temperatura y bias es no lineal. Además de los errores deterministas, la temperatura puede también influir en los errores estocásticos, de forma que la varianza del sensor y la del bias pueden variar con la temperatura. En algunos sistemas, esta varianza con la temperatura se modela como un coeficiente constante que se multiplica por un incremento de temperatura, el cual marca la diferencia entre la temperatura a la que el instrumento fue calibrado y la temperatura medida actual [39]. En dispositivos como algunos sensores MEMS que utilizan variaciones en la frecuencia de resonancia para realizar medidas, la temperatura puede hacer que esta frecuencia de resonancia varíe. Existen muchos trabajos dedicados a estudiar la relación de la temperatura con los errores de las IMU, por ejemplo en [40] podemos ver un análisis exhaustivo de la dependencia que tiene la temperatura con los procesos estocásticos en una IMU basada en MEMS, y en [41] se realiza una compensación del bias de un giróscopo MEMS estableciendo su relación con la temperatura.

5.1.2 Estocásticos

Los errores deterministas son muy fáciles de compensar, ya que basta con restar el valor de un bias para obtener el valor deseado, dividir entre un factor de escala determinado u otro tipo de consideraciones. Sin embargo, cuando entra en juego la aleatoriedad, la incertidumbre, lo que hace que las medidas sean menos precisas y se pueda confiar menos en ellas (como comentaba en el capítulo de los filtros de Kalman) es necesario recurrir a los modelos estadísticos. Si modelásemos todos los errores como uno solo distribuido de forma normal o gaussiana, podríamos definir la suma de todos los errores deterministas como media y la suma de los estocásticos como varianza. Existen multitud de términos que influyen a los errores estocásticos de un sensor, por ello, en este punto nos centraremos en localizar cada uno de esos términos y, si es posible, identificar de donde vienen.

El método para analizar los errores estocásticos más conocido y usado por su simplicidad es el método de Allan Variance (ver anexo B). Este método utiliza un gran número de muestras del sensor y, utilizando diferentes tamaño de *clúster*, va imprimiendo una gráfica que dependiendo de τ mostrará un tipo de error u otro (figura B-0-2).



Una vez realizado el Allan Variance, lo que obtenemos es una curva de estas características, de forma que podemos identificar los errores que forman el ruido del giróscopo o del acelerómetro dependiendo de la pendiente de la recta.

5.1.2.1 Quantization Noise

El error de cuantización está causado por el paso de una señal analógica a una señal digital [42]. Es debido a la existencia de variaciones entre el valor de la señal y su representación digital, donde los niveles son discretos. Depende, por tanto, de la resolución del ADC, ya que cuanta mayor resolución de bit tenga, menor será este error. Pese a que está presente, generalmente no es un error que se tenga en cuenta a la hora de hacer el modelado, de hecho, como comenta [43], en un acelerómetro, es mucho más determinante el *Bias Instability* del giróscopo, que su propio error de cuantización, por lo que no debe preocuparnos. En la curva del Allan Variance se muestra con una inclinación de -1 en la primera parte. Para su cálculo se mira en el punto que $T=\sqrt{3}$ [42].

5.1.2.2 Angle Random Walk (Velocity Random Walk para el acelerómetro)

El ARW (o VRW) incluye los términos de alta frecuencia con un tiempo de correlación mucho más pequeño que el tiempo de muestreo [44]. En la gráfica aparece como una zona con inclinación -1/2. Este error está debido al ruido termo-mecánico que sufren los sensores. Este error se debe a la agitación que sufren los electrones en un conductor cuando se aplica un determinado voltaje y se genera una corriente. En teoría de circuitos, este error se suele expresar como una fuente de voltaje o de corriente determinada fuente de ruido puesta en serie o en

paralelo respectivamente de una resistencia, por ejemplo. De este modo, obtenemos una resistencia "sin ruido" y expresamos el ruido mediante una fuente. Podemos ver esto explicado en la figura 5-2. De este modo, las corrientes que circulan por nuestro sensor generan este ruido.

Generalmente el ARW se modela como ruido blanco o *White noise* y representa la varianza intrínseca al sensor. Para calcular su valor numérico, basta con mirar el valor de la gráfica en $\tau = 1$. De esta forma obtenemos el coeficiente N en unidades de º/s/$\sqrt{Hz}$ para el giróscopo o m/$s^2/\sqrt{Hz}$ para el acelerómetro, donde los Hz hacen referencia a la frecuencia de muestreo (si las muestras de los sensores están en º/s y m/s^2 respectivamente).

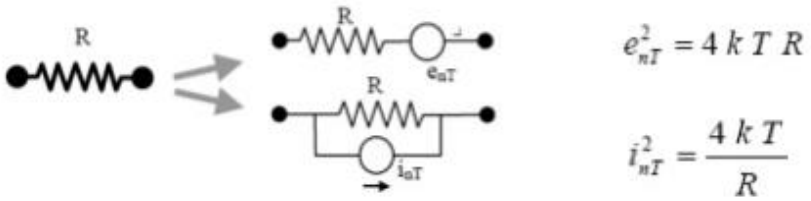


Figura 5-2. Modelo de ruido de una resistencia. T es la temperatura y k la constante de Boltzmann.

5.1.2.3 Bias Instability

El *bias instability o bias stability* es un error que surge por la sensibilidad de algunas partes del circuito al ruido *flicker*. El ruido *flicker*, también denominado ruido rosa es un ruido de baja frecuencia, ya que a frecuencias medias y altas el ruido blanco hace que éste no tenga importancia. El origen exacto de dicho ruido no se ha descubierto aún, pero aparece en la inmensa mayoría de circuitos electrónicos en los que se trabaja a baja frecuencia [45]. Para calcular su coeficiente, basta con observar el valor más bajo de la curva de Allan Variance, y anotar ese valor que obtenemos en la unidad en la que esté la medida del sensor. El nombre viene de que dicho valor es la mejor estabilidad que el sensor puede alcanzar. El BI hace referencia a las variaciones que el bias del que hablábamos en los errores deterministas sufre [44]. Normalmente, este factor se modela (sobre todo en MEMS) como un proceso de Gauss-Markov de primer orden definido por la ecuación (5-2), donde x es el proceso GM y v es el ruido blanco de entrada o *driven noise* [46].

$$x'(t) = -\frac{1}{T_c}x(t) + v(t) \tag{5-2}$$

En nuestro caso, nos interesa el modelo en tiempo discreto, por lo que discretizaremos (5-2) para obtener (5-3).

$$x(k+1) = (1 - \frac{\Delta T}{T_c})x(k) + \eta(k) \tag{5-3}$$

Donde η será un ruido blanco en tiempo discreto de varianza desconocida, T_c sería el tiempo de correlación, que viene definido como el instante en el que el valor del Allan Variance es menor, es decir, donde medimos el coeficiente del Bias Instability. ΔT sería la tasa de muestreo $\Delta T = 1/f_m$ siendo f_m la frecuencia de muestreo. En este proceso GM, tenemos que la desviación típica de x es igual al valor que obtenemos en la gráfica, que llamaremos BI, de modo que $\sigma_x^2 = BI^2$. Suponiendo un sistema lineal invariante en el tiempo, las relaciones entre la matriz de covarianza de x y la del ruido blanco hacen que la relación entre sus varianzas también sea directa, de modo que podemos calcular la varianza de η, que inicialmente es desconocida, sabiendo el valor de la varianza de x. De este modo obtenemos que $\sigma_\eta^2 = \sigma_x^2\left(1 - e^{-\frac{2\Delta T}{T_c}}\right)$. Este será nuestro valor de varianza de los bias de giróscopo y acelerómetro [46].

Aunque hayamos dicho que el valor de la desviación típica del bias es σ_η, realmente no es verdad. La varianza del bias va incrementándose con el tiempo, y la medida del *bias instability* lo que nos muestra es en qué medida esta varianza se incrementa, por lo que habrá que sumarle el valor de varianza obtenido al anterior.

5.1.2.4 Rate Random Walk

Se trata de un proceso aleatorio con un gran tiempo de correlación que aparece como una curva de inclinación

+1/2 en la gráfica de Allan Variance, y para calcular su coeficiente K se mira la gráfica donde T=3 [42]. Debido a su alto tiempo de correlación (teóricamente cero), el Allan Variance no es el método más apropiado para medir este tipo de errores. Por ello, en la mayoría de los casos no se llega a ver la parte relativa al RRW en las gráficas del Allan Variance [43].

Este error aparece cuando el bias varía con el tiempo muy lentamente. Algunos piensan que es debido a variaciones en la temperatura [45] o a degradación o irregularidades en los sensores. Como vimos en el proceso de fabricación, se emplean técnicas de DRIE para moldear la capa que constituirá los sensores, lo que generará posibles irregularidades en la materia. Existe, sin embargo, discrepancia en cuanto a definir su origen, ya que otros piensan que puede deberse a la presencia de ruido blanco en aceleración angular [44]. Puede ser modelado como un proceso GM al igual que el ruido flicker, pero, a diferencia de éste, sería un proceso "sin memoria", esto es que su valor en k+1 depende sólo de su valor en k, y no de otros valores anteriores o posteriores [45].

5.1.2.5 Rate Ramp

El término *Rate Ramp o Drift Rate Ramp* podría no considerarse ni tan siquiera estocástico. Realmente, podría formar parte de los errores deterministas que antes definimos. Sin embargo, puede ser muy útil a la hora de definir el comportamiento del análisis de Allan Variance ante la presencia de errores deterministas [42]. En la curva de Allan Variance viene representado como una zona final de la curva que presenta una inclinación de +1, aunque en nuestro caso, no deberá aparecer [43].

5.1.2.6 Varianza total

La varianza total del sensor sería una suma de todas las componentes que hemos definido anteriormente. Existen muchas formas de expresar estas varianzas, como por ejemplo obtener la varianza integrada ([47] ecuación 2.11), sumar una serie de coeficientes calculados previamente ([44] ecuación (3) y tabla 1) o simplemente una suma de los que se consideren más significativos ([48] ecuación (9)). En nuestro caso no expresaremos la varianza total del sistema como tal, sino que tendremos en cuenta el valor del giróscopo, su bias y sus varianzas correspondientes. En todos los papers antes mencionados se intenta unificar todo como varianza del giróscopo (o acelerómetro), sin embargo, debido a la implementación del EKF que vamos a realizar, es más apropiado conocer los valores de la varianza que afectan a cada uno (bias del giróscopo, giróscopo, bias del acelerómetro y acelerómetro) por separado.

Entre los errores que acabamos de describir, algunos son más predominantes que otros. Particularmente, en los MEMS, los términos más significativos son el RRW, ARW y el *Bias Instability* [42]. Sin embargo, la estructura recursiva del EKF hace que no sea necesario tener en cuenta el RRW, ya que causa imprecisiones a largo plazo, pero no a corto.

Tenemos dos términos que hay que definir, la varianza del sensor y la varianza de su bias. Hemos comentado que el *Bias Instability* hace referencia a la varianza del bias y el ARW (VRW) alude a la varianza del sensor.

- **Varianza del sensor**. El coeficiente del ARW que hemos calculado anteriormente nos da un valor expresado en º/s/$\sqrt{Hz}$. En un proceso, si el valor medio tiene unas unidades concretas, las unidades de la desviación típica[22] deben ser las mismas y las de la varianza el cuadrado de dichas unidades, de modo que si las muestras son en º/s, la desviación típica debe estar en º/s. Para poder calcular la desviación típica asociada a un sensor habrá que multiplicar el valor obtenido en el ARW por la raíz cuadrada de la frecuencia de muestreo. De esta forma tenemos el valor de la desviación típica (y de la varianza) del sensor en cuestión en las unidades correctas. En nuestro caso, los datos del giróscopo estarán en rad/s y los del acelerómetro en g. Por tanto, podemos decir que la varianza del giróscopo en un eje concreto es de $\sigma_g^2 = \left(ARW * \sqrt{f_m(Hz)}\right)^2$ [43].
- **Varianza del bias**. En el caso de la varianza del bias, no obtenemos el valor de la desviación típica como tal, sino que obtenemos un incremento de la misma. El valor que hemos obtenido de la gráfica ya

[22] Sabemos que la varianza es el cuadrado de la desviación típica. En este documento generalmente utilizaremos σ para referirnos a desviación típica y σ^2 para la varianza.

tiene las unidades correctas, por lo que sólo sería necesario un factor de corrección $\left(1 - e^{-\frac{2\Delta T}{T_c}}\right)$. Como se puede deducir por el factor de corrección, el *bias instability* es un valor de la estabilidad del sistema que viene dado para un tiempo de correlación T_c y una frecuencia de muestreo concretos. Este valor se añadirá al que ya tiene la varianza, de modo que conocida la varianza del giróscopo en un eje concreto *Varianza_anterior* podemos definir que en el siguiente instante, la varianza tendrá un valor igual a $Varianza\_anterior + BI^2\left(1 - e^{-\frac{2\Delta T}{T_c}}\right)$. El *bias instability* es un factor a tener en cuenta a la hora de escoger un sensor, como vemos en la tabla 5-1 y más específicamente en la tabla 5-2.

Estos dos valores, el ARW y el *bias instability,* son dos valores muy importantes a la hora de escoger una IMU, tanto es así, que dependiendo de dichos valores, se pueden clasificar los giróscopos en diferentes rangos dependiendo de su calidad. La tabla 5-1 muestra una posible clasificación sacada de [13] donde podemos observar que los valores fundamentales que se tienen en cuenta son el factor de escala, el ARW, el *bias instability*, el rango, el ancho de banda y la tolerancia al impacto.

Tabla 5-1. Tipos de giróscopo.

Parameter	***Rate Grade***	***Tactical Grade***	***Inertial Grade***
Angle Random Walk, °/√h	>0.5	0.5-0.05	<0.001
Bias Drift, °/h	10-1000	0.1-10	<0.01
Scale Factor Accuracy, %	0.1-1	0.01-0.1	<0.001
Full Scale Range (°/sec)	50-1000	>500	>400
Max. Shock in 1msec, g's	10^3	10^3-10^4	10^3
Bandwidth, Hz	>70	~100	~100

Tabla 5-2. Calidad del giróscopo dependiendo de su *bias instability* [49].

Performance Grade	**Bias Stability**
Consumer	30-1000°/hr
Industrial	1-30°/hr
Tactical	0.1-30°/hr
High-end Tactical	0.1-1°/hr
Navigation	0.01-0.1°/hr
Strategic	0.0001-0.01°/hr

Una vez definidos los errores fundamentales que afectan a los sensores inerciales de una IMU, vamos a realizar un correcto modelado de dichos errores para cada uno de los sensores, giróscopo y acelerómetro.

5.2 Giróscopo

Para realizar un correcto modelado del giróscopo, se ha de tener en cuenta los siguientes errores: factor de escala, *misalignment,* bias, *g-sensitivity,* las variaciones de la temperatura y la parte de errores estocásticos. En nuestro caso, como haremos en el desarrollo del proyecto, utilizaremos las unidades de rad/s para el giróscopo. Un buen resumen de estos valores lo muestra (5-4) [39].

$$K_g\omega_{gyro} = \left(I + S_g + M_g\right)\omega + b_g + G_g a + T_g \Delta T + \eta_g + \beta_g \tag{5-4}$$

Donde los elementos que forman dicho modelo se definen así:

- ω_{gyro} (LSB[23]) es la salida digital del giróscopo, el valor que da la IMU.
- K_g (rad/s/LSB) es el factor de escala de sensibilidad del giróscopo y el encargado de convertir el valor que salía del giróscopo en unidades de velocidad angular. Realmente esta parte del modelo no es un error determinista ni estocástico, simplemente es un acondicionamiento de la señal de salida del dispositivo.
- I es la matriz identidad.
- S_g es una matriz que representa el factor de escala. Será una matriz de la forma

  $$\begin{pmatrix} S_{g,x} & 0 & 0 \\ 0 & S_{g,y} & 0 \\ 0 & 0 & S_{g,z} \end{pmatrix}$$

  Donde los valores harán referencia al factor de escala en cada uno de sus ejes y serán valores cercanos a 0.
- $M_g$es la matriz de acoplamiento cruzado, que modelará el error de *misalignment*. Será antisimétrica y sus valores representarán la relación que existe entre los diferentes ejes debido a que no estén perfectamente alineados entre ellos. La matriz tendrá una forma así

  $$\begin{pmatrix} 0 & M_{g,xy} & M_{g,xz} \\ M_{g,yx} & 0 & M_{g,yz} \\ M_{g,zx} & M_{g,zy} & 0 \end{pmatrix}$$, donde $M_{g,xy} = -M_{g,yx}$.
- b_g(rad/s) es el bias del giróscopo, su valor de continua.
- G_g(rad/s/g) es la matriz que representa la sensibilidad del giróscopo a la aceleración.
- a (g) es la aceleración que sufre el giróscopo.
- T_g (rad/s/ºC) es el coeficiente de temperatura del giróscopo.
- ΔT(ºC) es la diferencia de temperatura entre el valor de temperatura medido y el valor para el que ha sido calibrado el sensor.
- η_g(rad/s) es un ruido blanco con media cero y la desviación típica del giróscopo. En nuestro caso, podemos expresarlo de la siguiente manera. $\eta_g \sim N(0, \sigma_g^2)$.
- β_g(rad/s) hace referencia al valor de la varianza del bias. Puede definirse como una secuencia *random walk* de ruido blanco tal que $\beta_g = \sum_{i=1}^{k} N_i$ donde N_i son secuencias de ruido blanco. Esto lo que viene a decir es que la varianza del bias va creciendo con el tiempo.

Finalmente, tenemos una ecuación con dos partes, la parte estocástica, formada por los dos últimos elementos, y la determinista compuesta por el resto. A la hora de realizar nuestro modelo, que posteriormente incluiremos

[23] La salida digital del giróscopo y del acelerómetro (codificada posiblemente en Complemento a 2) será un entero con signo de 16 bits.

en el EKF, vamos a tener en cuenta una serie de simplificaciones.

Como hemos visto anteriormente, el *scale factor* es un valor importante y a tener en cuenta, sin embargo, también hemos visto que las propias IMU ya lo calibran desde dentro, por lo que en este caso, aunque podría hacerse, no va a incluirse en el modelo. Respecto al error de *misalignment*, me remito al punto 4.2.3.3 de fabricación de nuestro dispositivo, en el que detallamos que, mediante técnicas de fabricación MEMS-CMOS, integramos todos los giróscopos dentro de un mismo dado formando parte de la misma estructura eliminando así en gran medida el desalineamiento que podría causar el error del que hablamos, por tanto, no creo que sea necesaria su inclusión en el modelo. Aunque es correcto que la temperatura influye en el bias tal y como hemos expresado en este modelo, no es así como trataremos nosotros la temperatura. En el punto 5.4 detallaré cómo haremos frente a las variaciones que la temperatura realiza en nuestro sensor, eliminando el factor $Tg\Delta T$ de nuestro modelo. Es cierto que el giróscopo sufre perturbaciones que vienen de la aceleración lineal que le afecta, no obstante, y me remito de nuevo al proceso de fabricación MEMS-CMOS, el dispositivo se encuentra en una escala muy pequeña, que por consiguiente hará que la masa sea menor, lo que hará que este factor $G_g a$, aunque existente, pueda ser considerado despreciable en nuestro caso.

De este modo, nuestro modelo final sería algo semejante a esto:

$$K_g \omega_{gyro} = \omega + \eta_g + b_g + \beta_g \tag{5-5}$$

Donde los dos primeros elementos hacen referencia al sensor como tal y los dos últimos al bias del mismo.

5.3 Acelerómetro

Similar al caso que teníamos en el giróscopo, debemos tener en cuenta varios errores definidos arriba. De esta forma podemos considerar que el modelo [39] del acelerómetro esté formado por los siguientes factores

$$K_a a_{acc} = (I + S_a + M_a)a + b_a + T_a \Delta T + \eta_a + \beta_a \tag{5-6}$$

Como se puede observar, no existe dependencia con la aceleración lineal, esto es obvio ya que sería absurdo que la propia medida que el acelerómetro está midiendo sea una perturbación para el sensor.

Los elementos que forman dicho modelo se definen así:

- a_{acc} (LSB) es la salida digital del acelerómetro que da la IMU.
- K_a (g/LSB) es el factor de escala de sensibilidad del acelerómetro y el encargado de convertir el valor que salía del acelerómetro en unidades de g.
- S_a es una matriz que representa el factor de escala. Será una matriz de la forma

  $$\begin{pmatrix} S_{a,x} & 0 & 0 \\ 0 & S_{a,y} & 0 \\ 0 & 0 & S_{a,z} \end{pmatrix}$$

  Donde los valores harán referencia al factor de escala en cada uno de sus ejes y serán valores cercanos a 0.
- $M_a$es la matriz de acoplamiento cruzado, que modelará el error de *misalignment*. Será antisimétrica y sus valores representarán la relación que existe entre los diferentes ejes debido al desalineamiento entre ellos. La matriz tendrá una forma así

  $$\begin{pmatrix} 0 & M_{a,xy} & M_{a,xz} \\ M_{a,yx} & 0 & M_{a,yz} \\ M_{a,zx} & M_{a,zy} & 0 \end{pmatrix}$$, donde $M_{a,xy} = -M_{a,yx}$.
- b_a(g) es el bias del acelerómetro.
- T_a (g/ºC) es el coeficiente de temperatura del acelerómetro.

- η_a(g) es un ruido blanco con media cero y la desviación típica del acelerómetro. En nuestro caso, podemos expresarlo de la siguiente manera. $\eta_a \sim N(0, \sigma_a^2)$. Siendo $\sigma_a^2 = \left(VRW * \sqrt{f_m(Hz)}\right)^2$
- β_a(g) hace referencia al valor de la varianza del bias. Puede definirse como una secuencia *random walk* de ruido blanco tal que $\beta_a = \sum_{i=1}^{k} N_i$ donde N_i son secuencias de ruido blanco.

El modelo que utilizaremos del acelerómetro, igual que hicimos en el giróscopo, no será igual al mostrado, sino que realizaremos una serie de simplificaciones.

Tomando las mismas consideraciones que se tuvieron en cuenta en el giróscopo, despreciamos el scale factor, el error de *misalignment* y la dependencia con la temperatura, ya que son extrapolables al acelerómetro. De ese modo obtenemos el siguiente modelo.

$$K_a a_{acc} = a + \eta_a + b_a + \beta_a \tag{5-7}$$

Igual que en el giróscopo, la primera parte alude al sensor y la segunda a su bias.

5.4 Modelado de la temperatura

En los modelos presentados del giróscopo y el acelerómetro hemos considerado oportuno eliminar el factor temperatura, sin embargo, no vamos a obviarlo, sino que vamos a modelarlo de otra manera.

Tener en cuenta la temperatura es fundamental a la hora de diseñar un sensor, ya que, principalmente, afecta a la medida de dicho sensor perturbándola y haciendo que no sea válida. Puesto que en un sensor el elemento que más validez adquiere es sin duda la medida, no podemos permitir que algo tan presente en todas las aplicaciones sea a su vez tan determinante. Si nos vamos a los dispositivos que más precisión requieren y en los que más dinero se invierte, es decir, en los dispositivos militares, observamos que el rango de temperatura es mucho mayor que el que utilizamos en dispositivos cotidianos, lo que nos hace darnos cuenta de la importancia de la temperatura.

En este trabajo nos centraremos en los efectos que la temperatura tiene sobre las varianzas, tanto del giróscopo y del acelerómetro como de sus bias, siendo el efecto en la varianzas de los bias más significativos [43]. La forma en que lo haremos será la siguiente, realizaremos análisis de Allan Variance para diferentes temperaturas, obteniendo ARW, VRW y BI para diferentes temperaturas y, por tanto, estas varianzas para diferentes temperaturas. De este modo en cada iteración podremos darle al filtro de Kalman un valor de las varianzas que hará que se adecúe más al comportamiento del sistema.

Uno de los problemas que se pretenden solucionar es que en estos dispositivos, la relación entre el crecimiento de la varianza del bias y la temperatura es generalmente no lineal.

6. IMPLEMENTACIÓN DEL PROYECTO

I think computer viruses should count as life. I think it says something about human nature that the only form of life we have created so far is purely destructive. We've created life in our own image

- Stephen Hawking -

6.1 Fases del desarrollo del Proyecto

El desarrollo de este Proyecto consta de varias fases que se han ido realizando en orden cronológico:

- **Documentación.** Recopilación de información sobre filtros de Kalman y sus extensiones, unidades de medidas inerciales y, sobre todo, los errores que afectan a los sensores inerciales.
- **Estudio de errores que afectan a los sensores inerciales.** Definición de los errores más significativos que pueden perturbar las señales de giróscopo y acelerómetro.
- **Diseño del filtro de Kalman.** Se lleva a cabo un diseño de dicho filtro, que se encargará de transformar medidas de tres sensores inerciales en variables que nos indiquen la orientación del dispositivo, incluyendo el modelado correspondiente de ruidos que afectan al sistema.
- **Implementación del filtro.** Programado en C++ y Matlab, utilizará los datos de la IMU y los tratará como hemos mencionado.
- **Resultados.** Una vez implementado el filtro, se obtienen una serie de resultados que serán debidamente interpretados.

6.2 Diseño del filtro de Kalman

El diseño del filtro es una de las partes más importante del proyecto, ya que incluye profundizar en filtros de Kalman, incluir los modelos de ruido que hemos comentado en el capítulo 5 y realizar los programas en Matlab y en entorno Linux (programando en C++). Para comenzar, aunque ya definimos en el punto 4.3 las entradas y salidas de nuestro sistema, profundizaremos en ello.

Tenemos un sistema que vamos a representar en espacio de estados, utilizando el algoritmo del filtro de Kalman. Nuestro sistema estará entonces definido por las siguientes ecuaciones:

$$x_k = A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + w_{k-1} \tag{6-1}$$

$$z_k = H_k x_k + v_k \tag{6-2}$$

Donde podemos definir:

$$x = \begin{bmatrix} r_x & r_y & r_z & w_x & w_y & w_z & b_{gx} & b_{gy} & b_{gz} & b_{ax} & b_{ay} & b_{az} \end{bmatrix}^T$$

El vector de estado, cuyos componentes, de tres en tres son, valores de los ángulos de Euler (roll, pitch, yaw), velocidad angular, bias del giróscopo y bias del acelerómetro.

$$u = \begin{bmatrix} g_x & g_y & g_z \end{bmatrix}^T$$

La entrada de control, cuyos componentes son las medidas del giróscopo.

$$z = \begin{bmatrix} a_x & a_y & a_z & m_x & m_y & m_z \end{bmatrix}^T$$

El vector de medidas, formado por las medidas del acelerómetro y del magnetómetro.

$w_k \; y \; v_k$

Los errores asociados al proceso y a la medida respectivamente. Supondremos que estos errores son gaussianos de media 0 y una varianza que concretaremos más adelante.

Para obtener el valor de las matrices que definen el sistema, habrá que estudiar la relación entre estos tres vectores. Para empezar, la relación entre x, su valor pasado y u, que viene dada por A y B, puede obtenerse de estas ecuaciones.

$$r_x = r_x + w_x * \Delta t \qquad r_y = r_y + w_y * \Delta t \qquad r_z = r_z + w_z * \Delta t$$
$$w_x = g_x - b_{gx} \qquad w_y = g_y - b_{gy} \qquad w_z = g_z - b_{gz}$$
$$b_{gx} = b_{gx} \qquad b_{gy} = b_{gy} \qquad b_{gz} = b_{gz}$$
$$b_{ax} = b_{ax} \qquad b_{ay} = b_{ay} \qquad b_{az} = b_{az}$$

Donde Δt hace referencia a la tasa de muestreo que utilizan nuestros ADCs. De aquí se obtienen las matrices A y B.

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Esta parte es sencilla, ya que es la parte lineal de nuestro filtro, que recordamos que es un EKF. Para terminar con la parte de la ecuación (6-1), vamos a definir W. Como estudiamos, para determinar W es necesario realizar el jacobiano indicado en la ecuación (3-22). Sabemos que w_k es el ruido asociado al proceso, y en nuestro caso, como mencionamos, lo consideraremos como AWGN. Al considerarlo aditivo, sus componentes serán $w_x, w_y \;\; y \;\; w_z$ y se añadirán a las medidas del giróscopo. De este modo, se puede demostrar que el resultado de la ecuación (3-22) es W=B.

Para el cálculo de H tenemos que dividir la matriz en dos partes, la parte relacionada con el acelerómetro y la parte relacionada con el magnetómetro. En la parte anterior, todo era lineal, por lo que la matriz que relaciona los valores era igual que su jacobiana, pero en este caso habrá que calcular la matriz h tal que $z = h(x, v)$ y hacerle el jacobiano para obtener H. También, en la fase de actualización del EKF utilizaremos $h(x^-, 0)$ como estimación de las medidas. Como hemos hecho con w_k, v_k también es AWGN, por lo que podremos expresar la ecuación anterior de la forma $z = h(x) + v$.

Antes de comenzar, vamos a presentar las matrices de rotación, que nos ayudarán a determinar h. Una matriz de rotación R(θ) es una matriz que depende de un ángulo θ tal que, dado dos vectores cuya relación es $v_1 = R(\pi)v_2$, se puede decir que si el vector v_2 es rotado π radianes en el ángulo θ, el resultante es el vector v_1. Las matrices de rotación suelen denominarse R_x donde el subíndice indica el eje sobre el que se produce el giro. También se pueden agrupar entre sí, formando matrices de varios giros en varios ejes multiplicando matrices en el sentido de z a x tal que una matriz de rotación en tres ejes en el sistema cartesiano sería $R = R_z R_y R_x$. En [50] podemos encontrar tanto las tres matrices de rotación en los tres ejes que antes mencionamos, como la matriz de rotación para dos dimensiones, que también utilizaremos. Cabe destacar que los ángulos roll, pitch y yaw corresponden a los giros en x, y y z respectivamente.

El acelerómetro es más sencillo, por lo que empezaermos por él. Sabiendo que vamos a medir aceleraciones en g, podemos expresar la relación entre las medidas del acelerómetro y los ángulos de Euler de la siguiente manera.

$$\begin{bmatrix}0\\0\\1\end{bmatrix} = R_{yx}\begin{bmatrix}a_x - b_{ax}\\a_y - b_{ay}\\a_z - b_{az}\end{bmatrix} \tag{6-3}$$

Esto viene a expresar que si giramos en pitch y en roll podemos conseguir que el acelerómetro esté recto. Le restamos al acelerómetro su bias, para obtener la medida real. La ecuación (6-3) puede despejarse para obtener (6-4). De este modo ya tenemos $ha(x)$, que alude a la parte de $h(x)$ que corresponde al acelerómetro.

$$\begin{bmatrix}a_x\\a_y\\a_z\end{bmatrix} = R_{yx}^T\begin{bmatrix}0\\0\\1\end{bmatrix} + \begin{bmatrix}b_{ax}\\b_{ay}\\b_{az}\end{bmatrix} \tag{6-4}$$

En la parte del magnetómetro tenemos dos problemas. El primero es que, debido a lo impreciso que es el sensor, intentaremos que afecte lo menos posible al sistema y por ello lo usaremos únicamente como brújula. Tendremos que convertir el vector de 3 coordenadas en uno de 2 sobre el plano de la tierra y además normalizado. El segundo problema es que la frecuencia de muestreo del magnetómetro es menor que el de los otros dos sensores, por lo que lo que hemos hecho en el código (anexo A) es plantear dos opciones a la hora de realizar la iteración. Medida con magnetómetro o medida sin ella. Desde el creador de logs (imu_logger.cpp) vamos a escribir en fichero el valor 10000.0 en las tres coordenadas del magnetómetro cuando no exista medida del mismo para poder entrar en una u otra opción. La diferencia es que cuando iteramos sin medida del magnetómetro, la $h_m(x)$ que vamos a calcular ahora no se incluiría en la H total, lo que conlleva a utilizar dimensiones diferentes en la fase de actualización como vemos en los códigos de KF.cpp y KF.m.

Procedamos al cálculo de ese vector bidimensional. En la función *magnet* del código lo que haremos será este cáclulo. El procedimiento es el siguiente:

Giramos el vector actual del magnetómetro en roll y pitch, siguiendo un procedimiento similar al utilizado en el acelerómetro (6-5). El nuevo vector tendrá un giro en x y en y diferente del original, no obstante, el yaw será el mismo, así que sigue aportándonos la dirección del polo norte magnético. Las medidas de la forma m_x'' son las originales mientras que las que carezcan de una de las tildes son las que obtenemos.

De la medida actual, sólo nos interesa los componentes en x y en y, por lo que eliminamos el componente en z, de este modo el magnetómetro no corrompe las predicciones que el giróscopo hace en roll y pitch, afectando solamente al yaw. Podría decirse que es el acelerómetro el encargado de actualizar las predicciones de roll y pitch y el magnetómetro las de yaw. Obtenemos así un vector bidimensional, al que realizamos una

normalización, obteniendo así el nuevo vector $z = \begin{bmatrix} m_x & m_y \end{bmatrix}^T$.

A este proceso lo denominaremos acondicionamiento del magnetómetro.

$$\begin{bmatrix} m'_x \\ m'_y \\ m'_z \end{bmatrix} = R_{yx} \begin{bmatrix} m''_x \\ m''_y \\ m''_z \end{bmatrix} \tag{6-5}$$

Una vez tenemos este vector bidimensional normalizado, de manera análoga a lo realizado con el acelerómetro, podemos calcular h_m mediante la ecuación (6-6). Donde R_{2D} es la matriz de rotación en dos dimensiones que incluirá como ángulo el yaw.

$$\begin{bmatrix} m_x \\ m_y \end{bmatrix} = R_{2D}^T \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6-6}$$

Con una fusión entre h_a y h_m, finalmente obtenemos h, en la derecha de la igualdad en la ecuación (6-7).

$$\begin{bmatrix} a_x \\ a_y \\ a_z \\ m_x \\ m_y \end{bmatrix} = \begin{bmatrix} R_{yx}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} b_{ax} \\ b_{ay} \\ b_{az} \end{bmatrix} \\ R_{2D}^T \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} \tag{6-5}$$

Para calcular H utilizamos la ecuación (3-23), y obtenemos (6-8).

$$H = \begin{bmatrix} 0 & -\cos(r_y) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \cos(r_x)\cos(r_y) & -\sin(r_x)\sin(r_y) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -\cos(r_y)\sin(r_x) & -\cos(r_x)\sin(r_y) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -\sin(r_z) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\cos(r_z) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6-8}$$

En cada iteración, como indica la ecuación (3-23) los valores de los ángulos de Euler serán los estimados previamente en la fase de predicción por el vector de estado. Al igual que ocurría con el cálculo de W, el cálculo de V resulta más sencillo de lo que aparenta. Siguiendo la ecuación (3-24), puede demostrarse que, debido a que tratamos v_k como AWGN, la matriz V es igual a la matriz identidad de 5x5. Al igual que H, cuando haya medida sin magnetómetro, las dos últimas filas de dichas matrices no formarán parte de las mismas. En el código esta nueva H y V se llamarán H2 y V2.

Prácticamente hemos definido la totalidad del sistema, ya sólo haría falta implementar el algoritmo de Kalman y obtener los valores correspondientes de roll, pitch y yaw. No obstante, nos queda un par de matrices por definir, las matrices Q y R, encargadas de modelar la varianza del sistema. Sería absurdo que después de todo un trabajo hablando de los errores que afectan a los sensores los ignorásemos ahora. Si bien la matriz de covarianza P lleva en su diagonal las varianzas de los elementos del vector de estado, las diagonales de Q y R albergarán las varianzas de los sensores.

La matriz Q es una matriz diagonal de 3x3 que representa la varianza del ruido asociado al proceso. En nuestro caso, estas varianzas serán las del giróscopo. La matriz R es una matriz diagonal de 5x5 que representa la varianza del ruido asociado a la medida, en nuestro caso, el acelerómetro y el magnetómetro. No nos hemos centrado mucho en el magnetómetro en este trabajo, por lo que simplemente le asociaremos una varianza constante de 0.002. Respecto al cálculo de las varianzas de los sensores, nos remontamos al estudio de errores, a la parte final en la que definíamos que la varianza del giróscopo se calcularía con la fórmula $\sigma_g^2 = \left(ARW * \sqrt{f_m(Hz)}\right)^2$ y la del acelerómetro con $\sigma_a^2 = \left(VRW * \sqrt{f_m(Hz)}\right)^2$ siendo $f_m = 1/\Delta t$ la frecuencia de muestreo, 200 hz. No nos olvidemos de la varianza del bias de los sensores, que había que calcular basándonos en el *Bias Instability*. Como hemos mencionado anteriormente, en cada iteración, después de la fase de predicción, sumaremos un valor a las últimas 6 posiciones de la diagonal de la matriz de covarianza. Estas

últimas 6 posiciones hacen referencia a las varianzas de los bias de giróscopo y acelerómetro, y, como podemos recordar del estudio del ruido *Flicker,* van incrementándose con el tiempo.

Esto podría ser muy sencillo con un análisis de Allan Variance obteniendo los valores de BI, VRW y ARW. Sin embargo, lo que estamos haciendo es un modelado en función de la temperatura, por lo que para conocer el incremento de varianza en los bias y las varianzas de los vectores para una temperatura dada, el procedimiento ha sido el siguiente:

En primer lugar se han realizado varios logs a diferentes temperaturas, cuatro en nuestro caso, con el objetivo de obtener un análisis de Allan Variance para cada temperatura concreta. Dejando previamente media hora el dispositivo acondicionarse al ambiente, se han tomado logs de en torno a una hora de duración. Para realizar el Allan Variance después y que de unos valores coherentes es importante que el dispositivo permanezca inmóvil durante la captación de datos. Puesto que el rango de temperaturas es de -40 a +85 ºC, he tomado medidas a -2, 18, 33 y 48 ºC. El resultado es el mostrado en las figuras de la 6-5 a la 6-12.

En segundo lugar, para cada una de esas temperaturas se ha creado un vector de 12 elementos con las varianzas correspondientes, de forma que ya tenemos las varianzas a diferentes temperaturas. Con estos 4 puntos podremos establecer 12 gráficas (Figuras de la 6-1 a la 6-4) que representen la variación de las varianzas. Comparando nuestras gráficas con las que obtiene [41], podemos observar como la evolución de la varianza del bias tiene una tendencia similar.

Mediante triangulación, calcularemos las varianzas intermedias utilizando la ecuación $v_3 = \frac{T_3 - T_1}{T_2 - T_1}(v_2 - v_1) + v_1$ donde v es un vector de varianzas y T es temperatura. Los subíndices 1 y 2 hacen referencia a los dos puntos utilizados para la triangulación, mientras que el 3 hace referencia a la temperatura medida y a la varianza correspondiente a esa temperatura. Todo esto está implementado en el código. De esta forma en cada iteración se mide la temperatura y se calcula el valor de las varianzas, se crean asi las matrices Q y R y se modifica la matriz de covarianza P.

Tedríamos definida la totalidad del sistema. La programación del filtro no tiene más misterio que implementar el algoritmo de Kalman como está hecho en el código (anexo A).

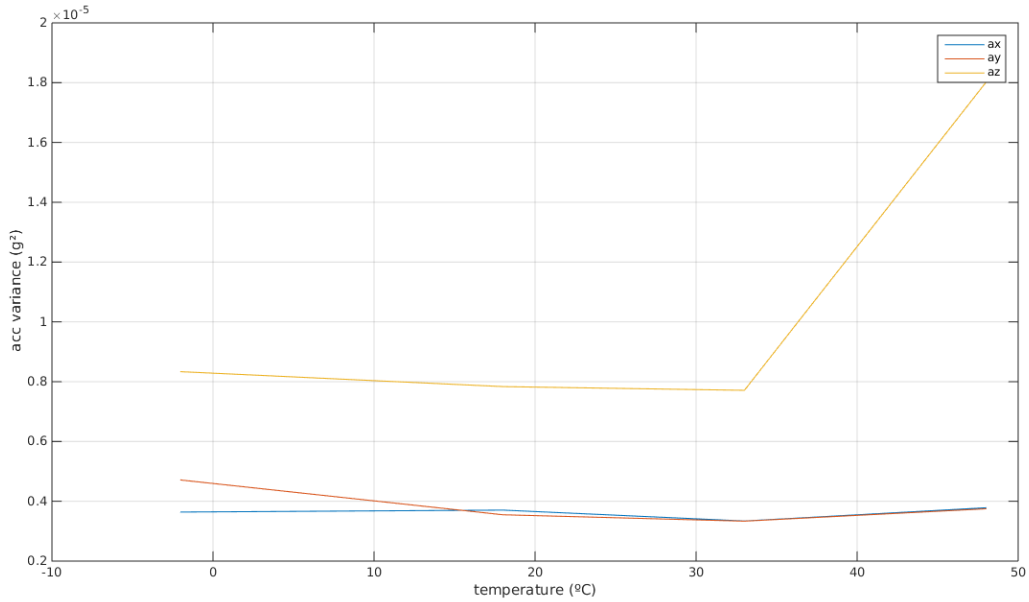


Figura 6-1. Varianza del acelerómetro.

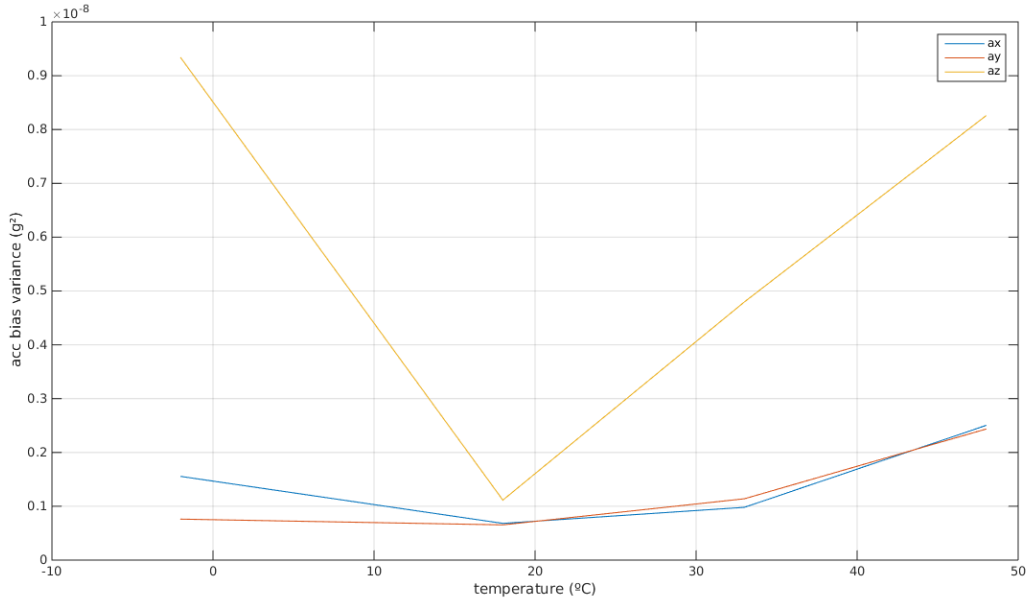


Figura 6-2. Varianza del bias del acelerómetro.

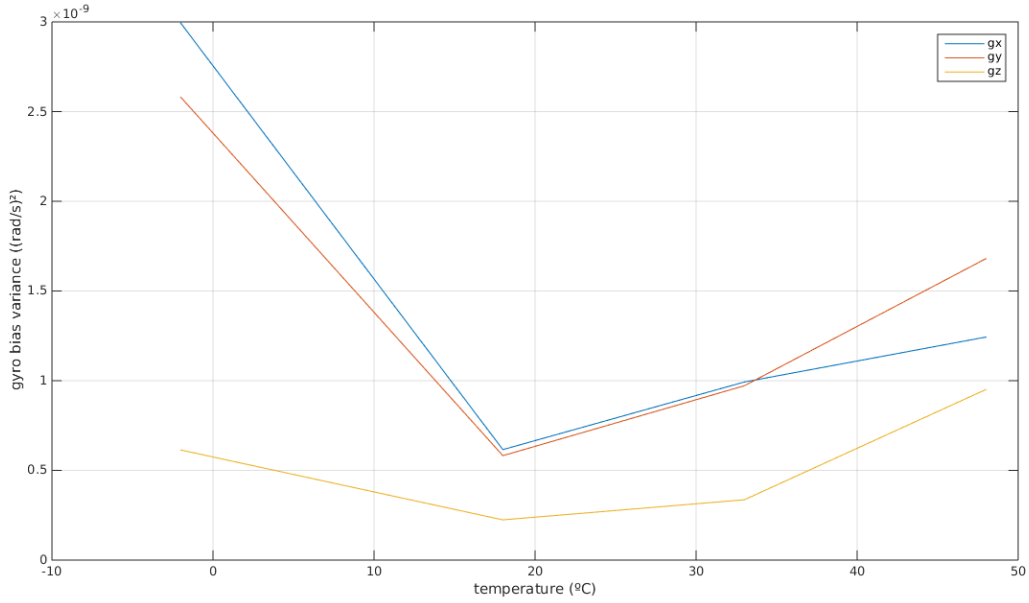


Figura 6-3. Varianza del bias del giróscopo.

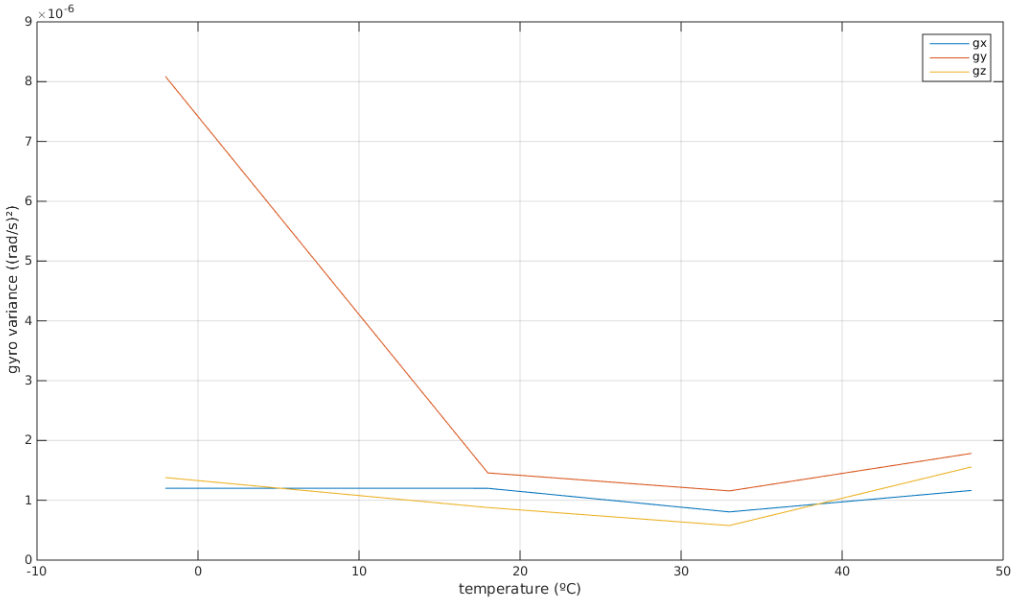


Figura 6-4. Varianza del giróscopo.

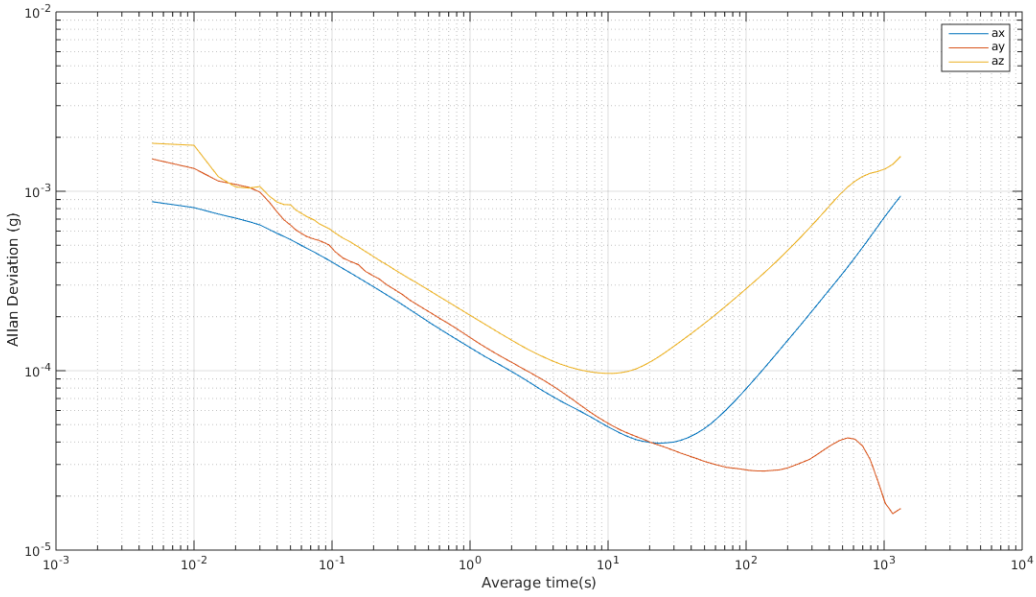


Figura 6-5. Allan Variance del acelerómetro a -2ºC.

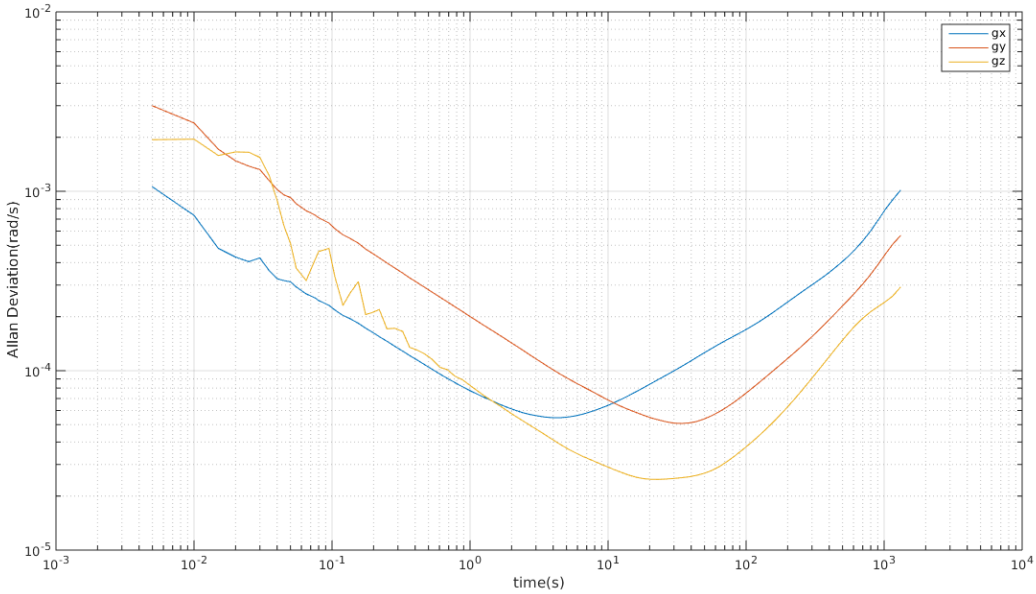


Figura 6-6. Allan Variance del giróscopo a -2ºC.

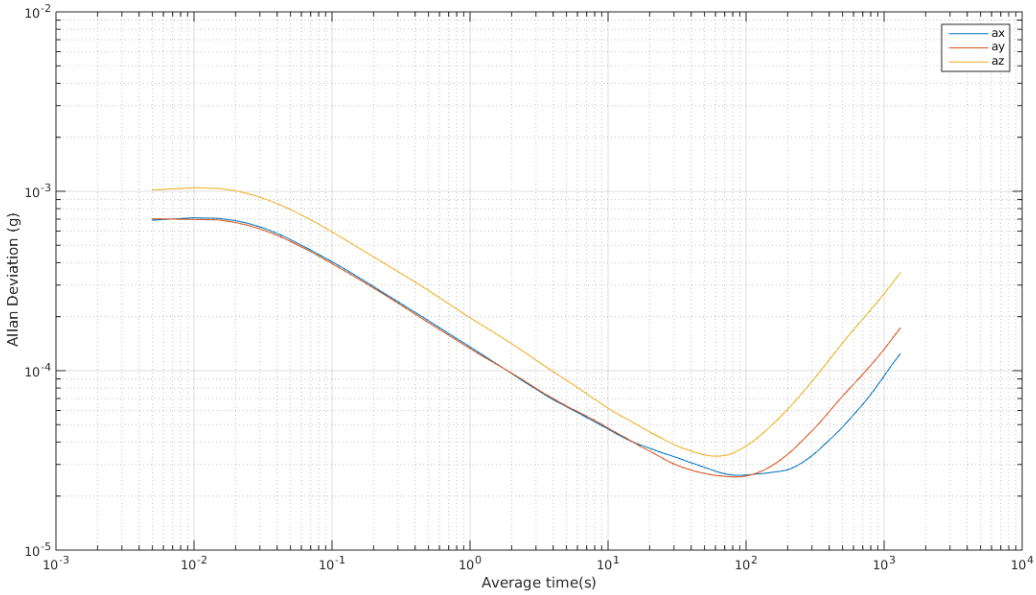


Figura 6-7. Allan Variance del acelerómetro a 18ºC.

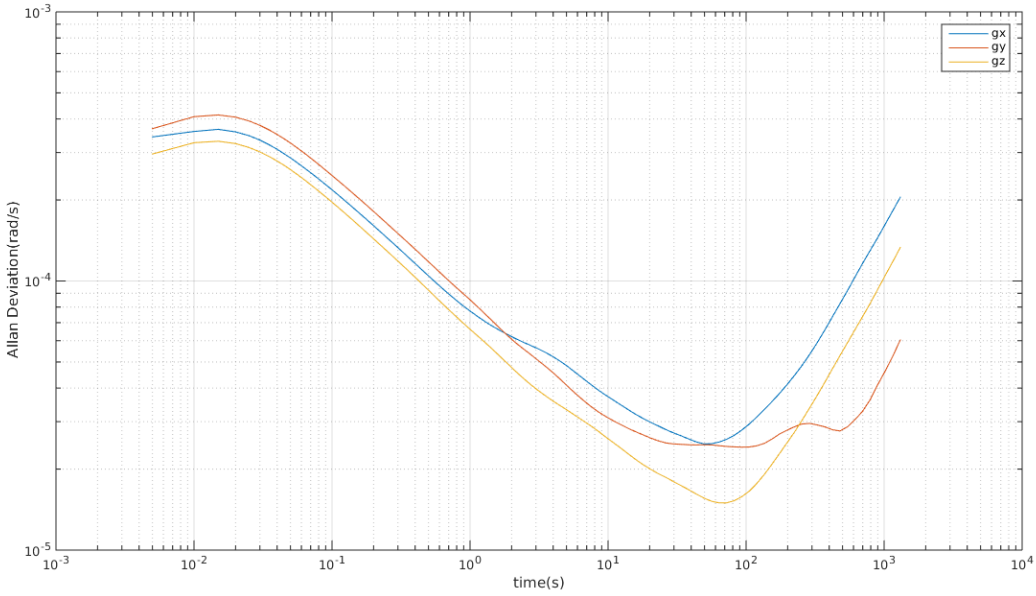


Figura 6-8. Allan Variance del giróscopo a 18ºC.

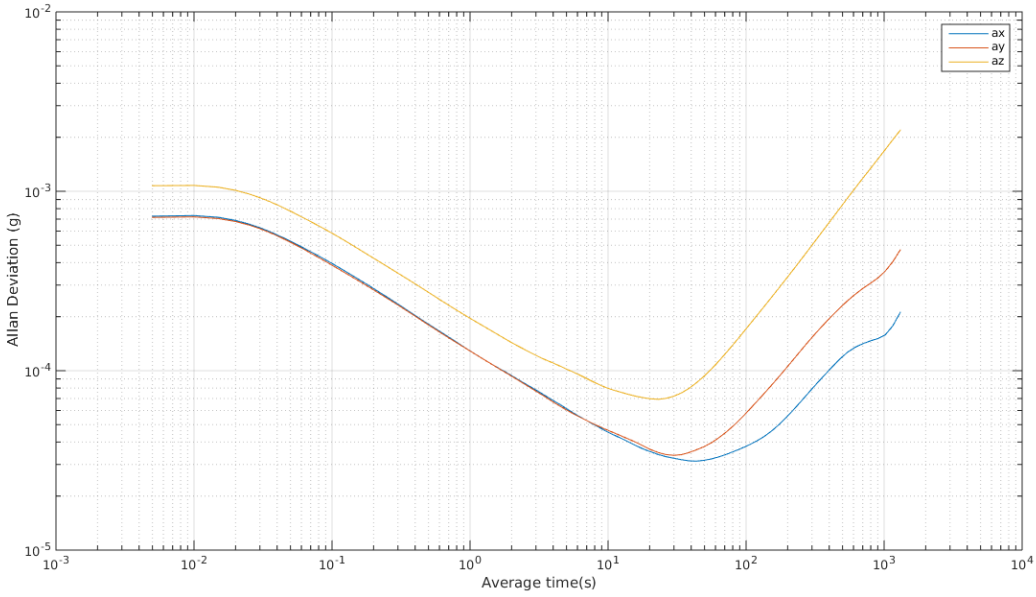


Figura 6-9. Allan Variance del acelerómetro a 33ºC.

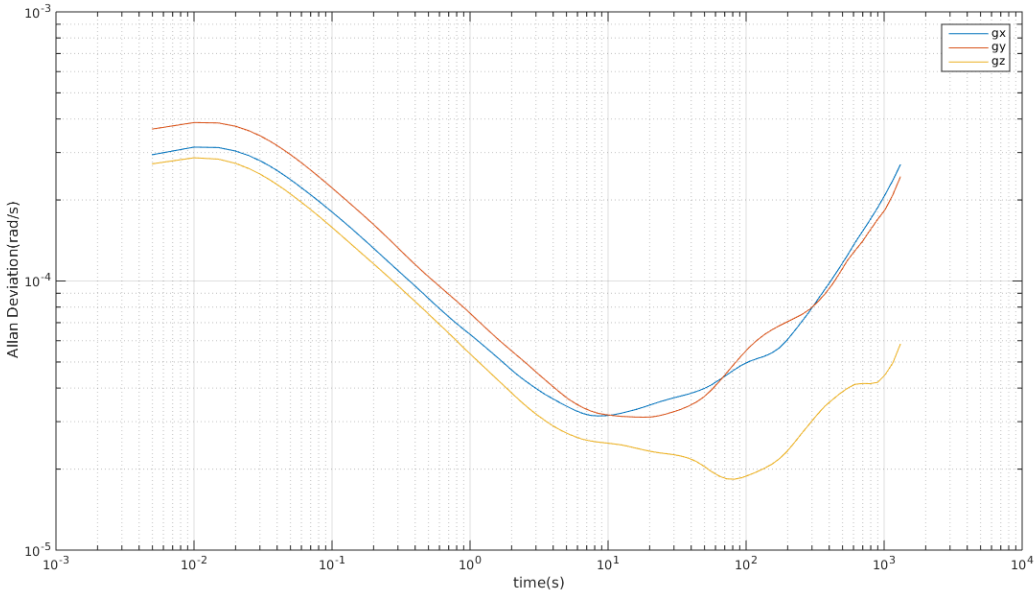


Figura 6-10. Allan Variance del giróscopo a 33ºC.

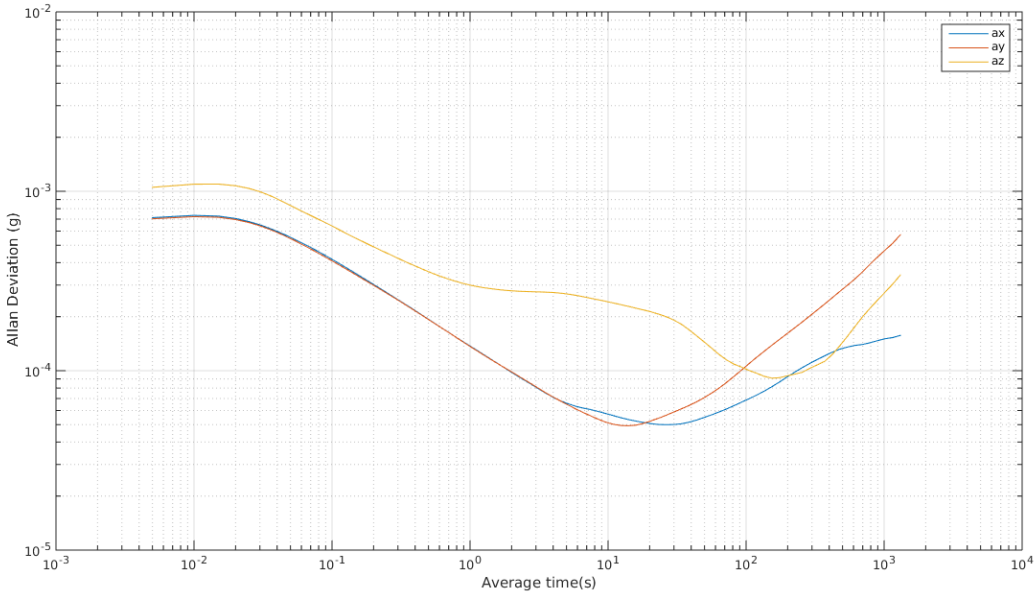


Figura 6-11. Allan Variance del acelerómetro a 48ºC.

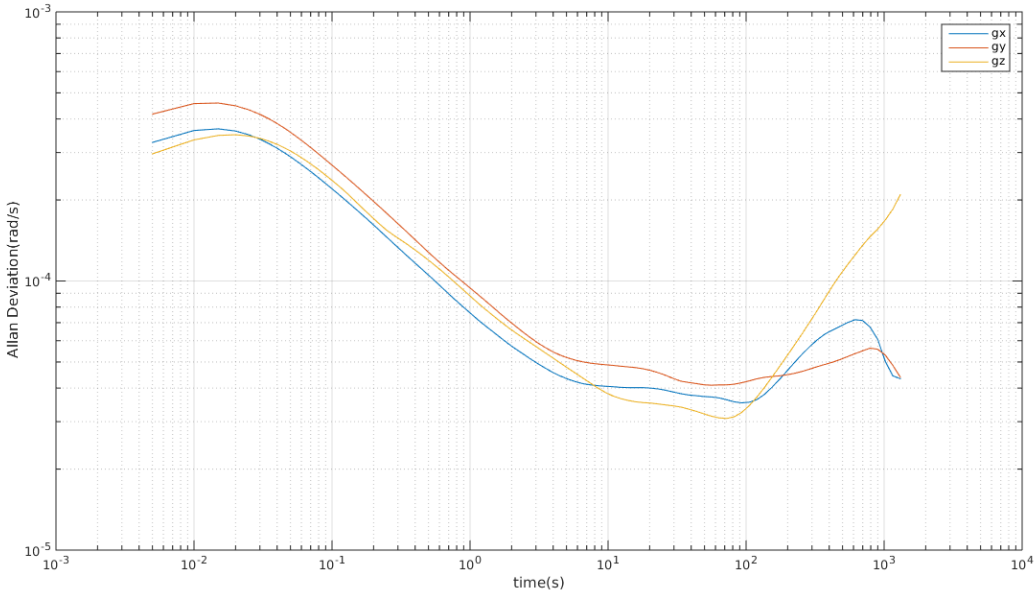


Figura 6-12. Allan Variance del giróscopo a 48ºC.

6.3 Flujo de datos entre programas

Hemos utilizado varios programitas para la correcta realización del proyecto, por lo que convendría hacer una breve explicación del flujo de datos realizado entre ellos. Hay dos partes del proyecto, recopilación de datos y calibración del modelo, y la implementación del filtro de Kalman con esos modelos.

Para la extracción de datos he utilizado ***imu_logger.cpp*** que incluye a ***arduimu_v3.hpp***. Esto me generaba tanto logs estáticos como dinámicos. Considero logs estáticos los utilizados para realizar el análisis de Allan Variance a diferentes temperaturas, con el dispositivo quieto. Están escritos en formato *imu_log_xxx.txt* donde xxx identifica el rango de temperatura, siendo bajo para -2, media para 18, media alta para 33 y alto para 48. Los logs dinámicos registrarán movimiento en la imu y su formato es *imu_log_PRUEBAx.txt* siendo x indicativo de si la medida se ha hecho a temperatura ambiente (1) o a temperatura mayor de la habitual (2).

Para el análisis de Allan Variance utilizo ***ReadAllan.m*** y ***allan.m.*** La primera lee un log estático, llama a la segunda, que implementa el algoritmo, y calcula el valor de las varianzas para una temperatura dada.

Para el filtro de Kalman utilizo tanto ***ReadKalman.m, Kalman.m*** y todas las demás de Matlab como ***KF.cpp.*** Éstas reciben logs dinámicos y generan una serie de salidas, en el caso de Matlab un par de variables x y P, y en el caso de C++ un fichero con el formato *Salida_pruebax.log* con x correspondiente al de la entrada. La implementación del filtro en Matlab ha sido más que un instrumento de resultados, una forma de depurar posibles errores, ya que es más fácil de depurar código en Matlab que código en C++.

He utilizado otros dos para imprimir por pantalla tanto las gráficas de las temperaturas ***(plotvar.m)*** como las salidas de ***KF.cpp*** (***Plotter.m***).

7. RESULTADOS Y CONCLUSIONES

Everything should be made as simple as posible. But not simpler

- Albert Einstein -

7.1 Resultados

En este trabajo se realiza una pequeña introducción a filtros de Kalman, Sistemas de Navegación Inercial y los errores que afectan a una IMU.

Hemos diseñado un filtro de Kalman y hemos incluido en él el modelo de temperatura que hemos realizado. Mediante el análisis de los errores que afectan a los sensores inerciales de la IMU hemos podido caracterizar dichos sensores de bajo coste, una fase muy importante en el diseño de Sistemas de Navegación Inerciales. Un análisis de dichos errores se ha realizado en este trabajo, escogiendo después cuales serán más determinantes y, por tanto, cuales formarán parte de nuestro modelo. El modelo se trata de un modelo de las varianzas del sistema dependientes de la temperatura, temperatura que hemos medido gracias al sensor de temperatura que nuestro sistema incluye.

Esta caracterización de la que hablábamos podría resumirse en la tabla 7-1, que no es totalmente significativa ya que se ha calculado a partir de las medias de los valores de las varianzas en los tres ejes y a diferente temperatura, pero nos puede dar una estimación de las características de nuestra IMU.

Tabla 7-1. Características de la MPU-6000.

ARW	$9.4 \times 10^{-5}\ rad/s\ /\sqrt{Hz}$
VRW	$16.6 \times 10^{-5}\ g/\sqrt{Hz}$
BI gyroscope	$3.4 \times 10^{-5}\ rad/s$
BI accelerometer	$5.34 \times 10^{-5}\ g$

Para comprobar el correcto funcionamiento del sistema se han hecho un par de logs dinámicos a diferentes temperaturas. El primero (prueba 1) se ha realizado a una temperatura media de 18.7ºC, mientras que el segundo (prueba 2) se ha realizado a una temperatura media de 42.2 ºC. Los resultados se muestran en las figuras de la 7-1 a la 7-6, donde los valores del bias del giróscopo y del acelerómetro se muestran en rad/s y en g respectivamente. La variación que ha sufrido la temperatura a lo largo del experimento la mostramos en las figuras 7-7 y 7-8. Estas variaciones de temperatura eran relativas al lugar en el que se realizan los experimentos. Podemos observar como en la de temperatura ambiente varía poco más de un grado, mientras que en la de altas temperaturas varía hasta 6, esto hace que el sistema vaya cambiando las varianzas en todo momento para

adecuarse a la temperatura.

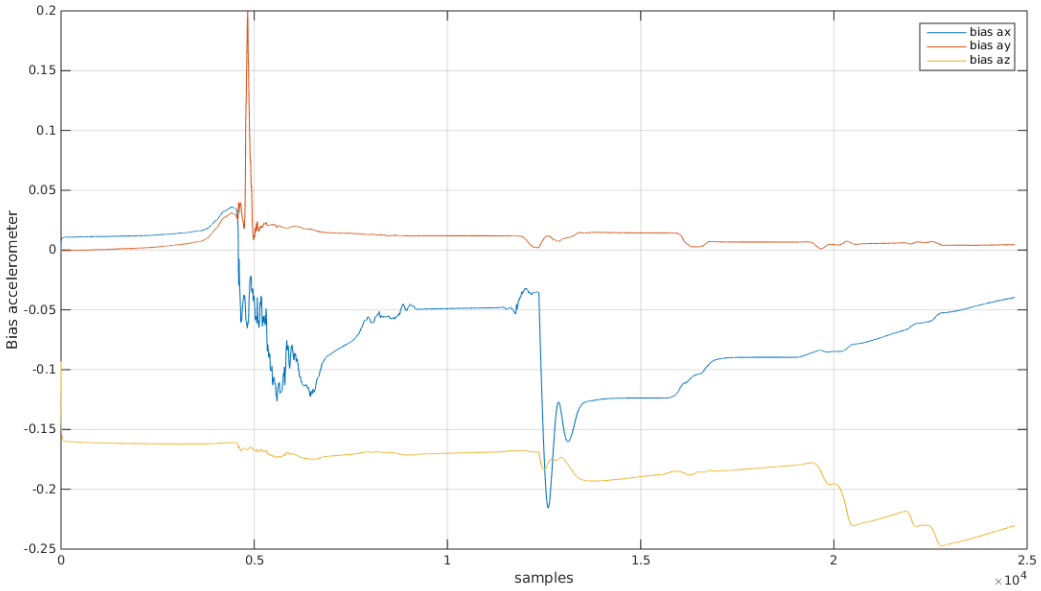


Figura 7-1. Bias del acelerómetro en prueba 1.

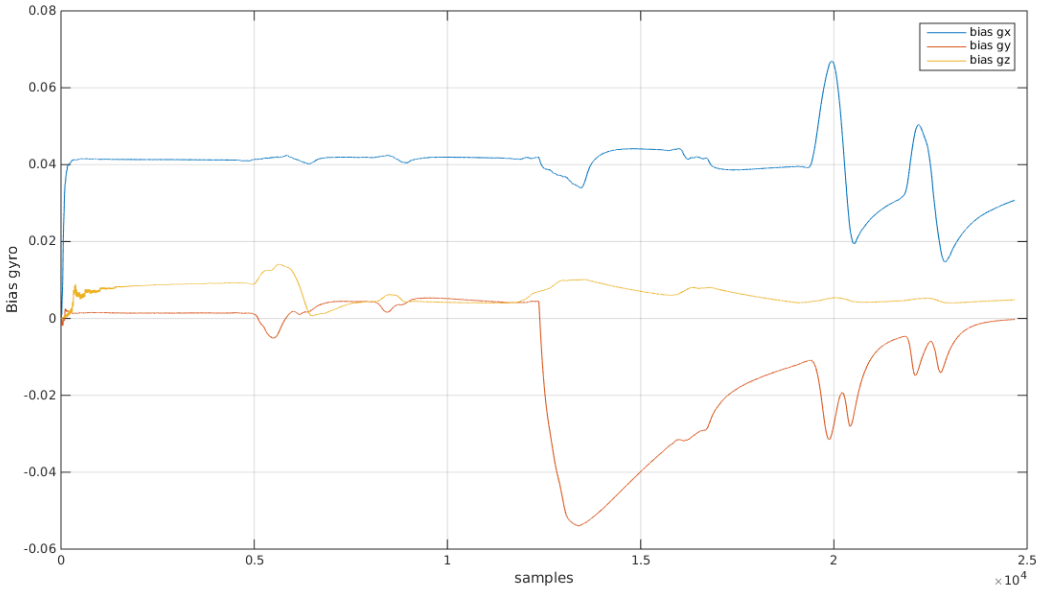


Figura 7-2. Bias del giróscopo en prueba 1.

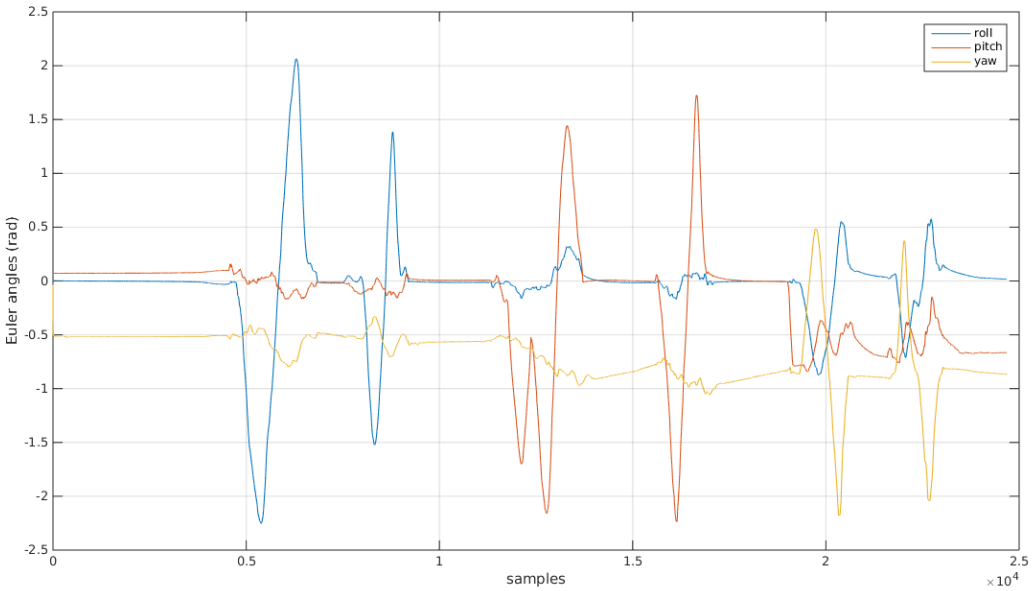


Figura 7-3. Prueba 1 a temperatura ambiente.

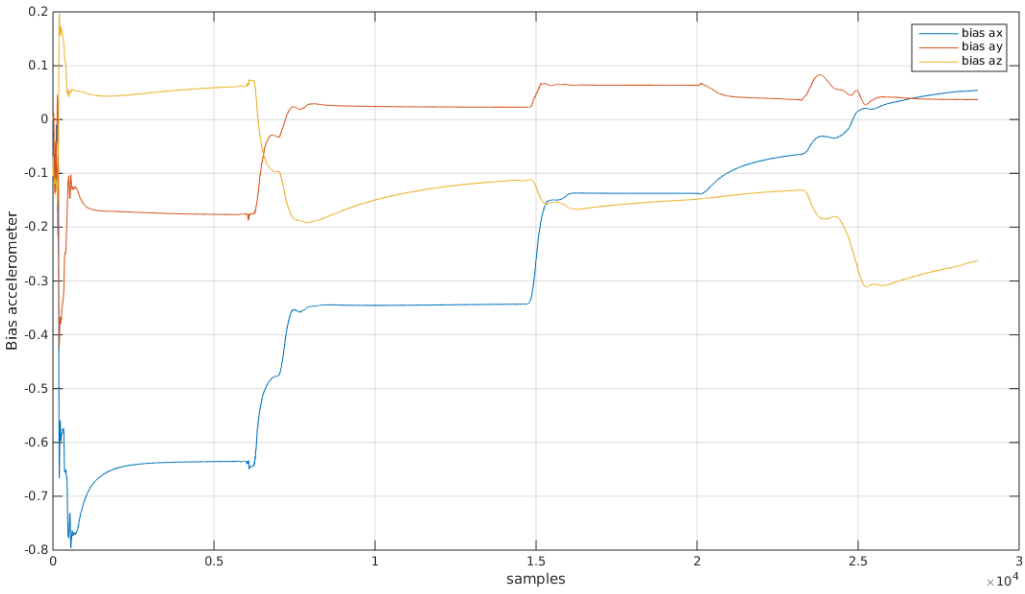


Figura 7-4. Bias del acelerómetro en prueba 2.

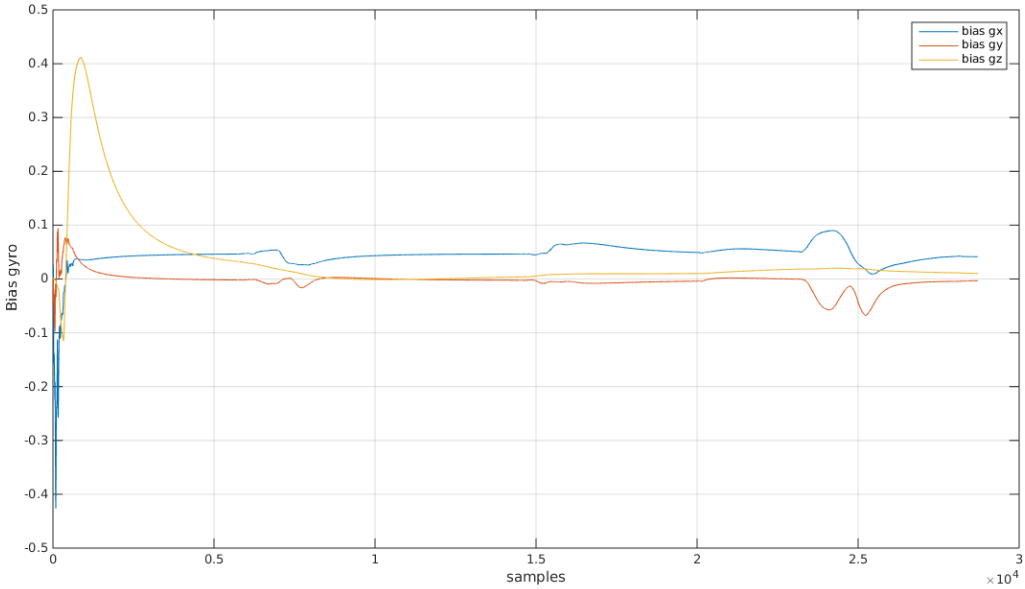


Figura 7-5. Bias el giróscopo en prueba 2.

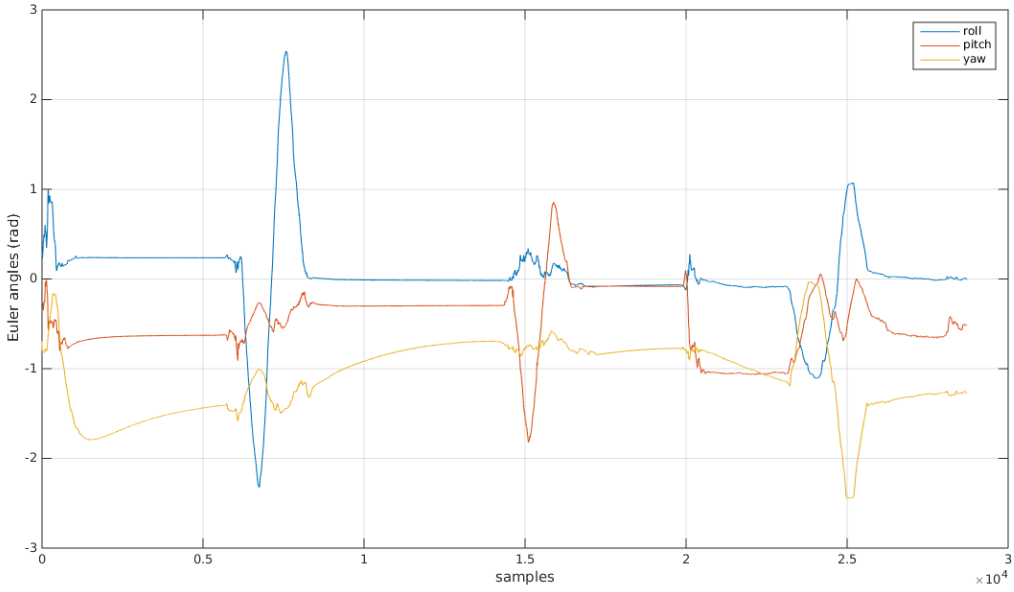


Figura 7-6. Prueba 2 a temperatura alta.

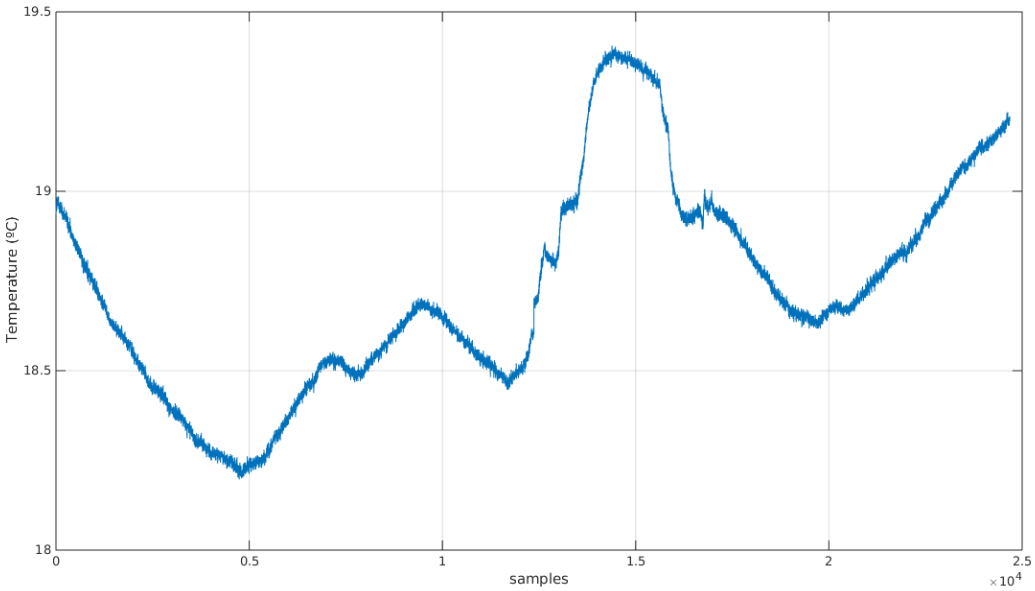


Figura 7-7. Evolución de la temperatura en prueba 1. La temperatura media es de 18.7ºC.

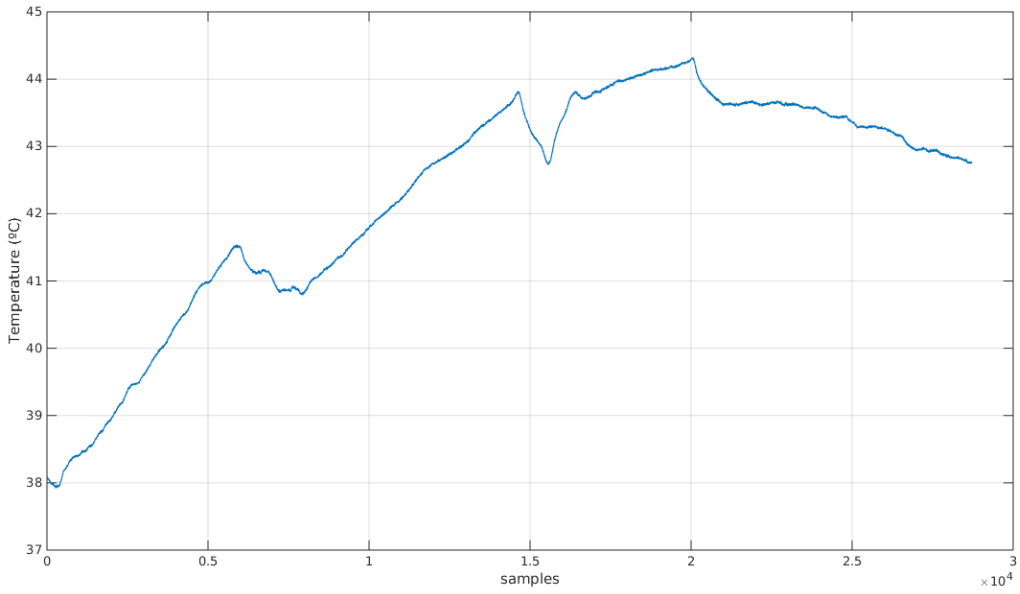


Figura 7-8. Evolución de la temperatura en prueba 2. La temperatura media es de 42.2ºC.

En estos experimentos hemos tratado de realizar giros en roll, pitch y yaw para simular el comportamiento en los tres ejes. En las gráficas que hacen referencia a la variación del bias del acelerómetro y del giróscopo, podemos ver que, aunque sufren variaciones bruscas en los momentos de giros, estas variaciones son de una magnitud pequeña y se ve como convergen finalmente a un valor.

En cuanto a las que aluden a los ángulos roll, pitch y yaw, vemos como varían en un sentido y otro y finalmente convergen. Hemos realizado uno o dos giros en roll, después en pitch, y finalmente en yaw. Obviamente existen cambios en los otros ángulos cuando giramos respecto a uno, esto es así porque este experimento está realizado a mano, con su inexactitud correspondiente.

7.2 Conclusiones y trabajo futuro

Los sensores inerciales basados en MEMS se han hecho un hueco en el panorama internacional debido a su bajo coste y su pequeño tamaño, pero también sufren una serie de errores que los hacen vulnerables. Por esta razón, es necesario realizar un análisis de los errores que afectan a nuestros sensores, principalmente los estocásticos, para poder utilizar sin problema el dispositivo.

En este trabajo la parte más importante sin duda ha sido la documentación, importante para desarrollar el pequeño modelo realizado e imprescindible para conocer los errores que afectan al sistema. Profundizar no sólo en cómo calcular un error sino en lo que lo causa, enriquece la visión sobre nuestro sensor, de modo que sepamos en todo momento con qué estamos trabajando. En esta parte ha sido especialmente importante el documento [29] facilitado por InvenSense en el que se describe la fabriación de sus sensores.

En el caso de este trabajo, a pesar de que se ha implementado un modelo de temperatura, no era el único objetivo, sino que también lo era el estudiar estos errores y sus posibles causas. Han quedado demostradas algunas de sus causas al igual como los errores que más afectan a los sensores inerciales, siendo estos últimos objeto de modelado.

En cuanto al modelo realizado, es válido para conseguir un sistema que responda de una manera más precisa de lo que lo hacía el componente lineal del factor de temperatura que incluía la ecuación (5-4) o (5-6). Se modela así algo muy importante sobre todo en sensores de bajo coste, y es que la dependencia entre la variación del bias y la temperatura es altamente no lineal. En las gráficas referentes a varianzas en diferentes temperaturas se ve como baja de forma abrupta y luego crece poco a poco en la mayoría de los casos. Este modelo no pretende ser un modelo super preciso que consiga una IMU perfecta, sino que responda a las no linealidades de la relación temperatura-bias.

Trabajo futuro

En esta área existen muchas posibilidades de expansión en cuanto a trabajo futuro se refiere. Concretamente, este trabajo se podría considerar una base para proyectos más concretos y específicos. Varias vías de futuro que considero apropiadas serían las siguientes:

- Pese a que hemos visto que los más fundamentales son el ruido blanco y el *flicker*, podría también tenerse en cuenta el resto de errores que hemos estudiado. Esto estaría más orientado a sensores que requieran más precisión.
- Implementar dicho modelado y filtrado en tiempo real, programándolo en la propia ArduIMU.
- Tener en cuenta la aceleración que sufre el acelerómetro debido a su propio movimiento. No hemos tenido en cuenta que la aceleración de dicho sensor no sólo está afectada por la gravedad cuando hemos hecho el cálculo de h_a.
- Estudiar más a fondo la configuración interna de la MPU y del magnetómetro para compensar errores. Considerar también el modelo de ruido que afectaría al magnetómetro, que no se trata en este documento.
- Sobre este mismo proyecto, se podría hacer un modelado de temperatura más exhaustivo. La precisión de mi modelado es baja y esto podría mejorarse de dos maneras, no exclusivas ente sí:
    - Podrían tomarse más medidas, ya que en nuestro caso tomamos únicamente 4, con lo que nuestra curva está lejos de ser una curva suave que se adapte mejor a las no linealidades de esta relación. Además podrían realizarse varias medidas a la misma temperatura y escoger un valor medio, con el objetivo de mitigar posibles errores puntuales.

- Las medidas podrían tomarse en un ambiente más controlado. En mi caso, las medidas las he realizado en mi piso y en la terraza. He tratado de que durante el tiempo que dura el experimento ninguna vibración externa lo pertube, como pueda ser pasos fuertes o portazos, sin embargo, los lugares no mantenían una temperatura fija sino que oscilaba en torno a un rango. Realizado en un laboratorio estas mismas medidas habrían resultado más precisas y fiables. De hecho, en [43] se comenta que el análisis de Allan Variance pierder validez si varía la temperatura.

REFERENCIAS

[1] O. J. Woodman, An introduction to inertial navigation, Cambridge: Technical report number 696 published by the University of Cambridge, 2007.

[2] H. Zao y Z. Y. Wang, «Motion measurement using inertial sensors, ultrasonic sensors, and magnetometers with extended kalman filter for data fusion,» *IEEE Sensors Journal,* vol. 12, nº 5, pp. 943-953, Mayo 2012.

[3] E. Gai, «The century of inertial navigation technology,» *Aerospace Conference Proceedings, 2000 IEEE,* vol. 1, pp. 59-60, 18-25 Marzo 2000.

[4] R. González Carvajal y C. Luján, Apuntes de Sistemas Embebidos, Sevilla: Departamento de Ingeniería Electrónica, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla, 2015.

[5] A. Zul Zafar y D. Hazry, «A simple approach on implementing IMU sensor fusion in PID controller for stabilizing quadrator flight control,» de *Proc. IEEE 7th Int. Colloq. Signal Processing and Its Applications*, Penang,Malaysia, Marzo 2011, pp. 28-32.

[6] M. Bloesch, M. Hutter, M. A. Hoepflinger, S. Leutenegger, C. Gehring, C. D. Remy y R. Siegwart, «State estimation for legged robots – consistent fusion of leg kinematics and IMU,» de *Proc. Robotics: Science and Systems*, vol. 2, Sydney, NSW, Australia, Julio 2012.

[7] W. W. Wang y L. C. Fu, «Mirror therapy with an exoskeleton upper-limb robot based on IMU measurement system,» de *IEEE International Workshop on Medical Measurements and Applications Proceedings (MeMeA)*, Bari, Mayo 30-31, 2011, pp. 370-375.

[8] R. S. McGinnis, «Miniaturized wireless IMU enables lowcost baseball pitching training aid,» de *Proc. 35th Ann. Meeting of the American Society of Biomechanics*, Long Beach, CA, Agosto 10-13, 2011.

[9] G. T. Schmidt, «INS/GPS Technology Trends,» [En línea]. Available: http://ewh.ieee.org/r10/queensland/v2/lib/exe/fetch.php/:chapters:aess:2009insgpsfinal.pdf. [Último acceso: 31 05 2016].

[10] X. Li, W. Xiao y Y. Fei, «Status Quo and Developing Trend of MEMS-Gyroscope Technology,» de *2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC)* , Qinhuangdao, Hebei, China, 18-20 Septiembre 2015.

[11] G. Welch y G. Bishop, An Introduction to the Kalman Filter, Chapel Hill, NC: Department of Computer Science, University of North Carolina at Chapell Hill, Actualizado: 24 Julio 2006.

[12] D. H. Titterton y J. L. Weston, Strapdown Inertial Navigation Technology - 2nd Edition, Institution of Engineering and Technology, 2004.

[13] S. Nasiri, A Critical Review of MEMS Gyroscopes Technology and Commercialization Status, Coronado Drive, Santa Clara, California: InvenSense, 2006.

[14] N. Lobontiu, «5.6: Tuning Forks. From Mechanical Design of Microresonators: Modeling and Applications,» 2006. [En línea]. Available: http://www.globalspec.com/reference/69969/203279/5-6-tuning-forks. [Último acceso: 31 05 2016].

[15] R. Sanders, «UC Berkeley physicists develop ultrasensitive gyroscope based on superfluid helium,» 4 10 1997. [En línea]. Available: http://www.berkeley.edu/news/media/releases/97legacy/gyroscope.html. [Último acceso: 31 05 2016].

[16] X. Rottenberg, «Silicon-germanium (SiGe) technology for MEMS and NEMS fabrication,» 15 03 2012. [En línea]. Available: http://www.memsjournal.com/2012/03/silicon-germanium-sige-technology-for-mems-and-nems-fabrication.html. [Último acceso: 31 05 2016].

[17] «Wikipedia - Magnetometer,» 17 05 2016. [En línea]. Available: https://en.wikipedia.org/wiki/Magnetometer. [Último acceso: 01 06 2016].

[18] T. R. McGuire y R. I. Potter, «Anisotropic Magnetoresistance in Ferromagnetic 3d Alloys,» *IEEE Transactions on Magnetics,* vol. 11, nº 4, pp. 1018-1038, Julio 1975.

[19] «3-Axis Digital Compass IC HMC5883L (Datasheet),» Honeywell International Inc., Plymouth,MN, Febrero 2013.

[20] R. E. Kálmán, «A new approach to linear filtering and prediction problems,» *J. Basic Eng.,* vol. 82, nº 1, pp. 35-45, Marzo 1960.

[21] «Wikipedia - Espacio de estados,» 09 11 2015. [En línea]. Available: https://es.wikipedia.org/wiki/Espacio_de_estados. [Último acceso: 01 06 2016].

[22] «Wikipedia - filtro de Kalman,» 24 05 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Filtro_de_Kalman. [Último acceso: 01 06 2016].

[23] S. J. Julier y J. K. Uhlmann, «Unscented filtering and nonlinear estimation,» *Proceedings of the IEEE,* vol. 92, nº 3, pp. 401-422, 2004.

[24] «ArduIMU v3 Schematic,» [En línea]. Available: http://cdn.sparkfun.com/datasheets/Robotics/ArduIMU328-v3.pdf. [Último acceso: 03 06 2016].

[25] J. Muñoz, «ArduIMU V3 is finally here!,» 02 11 2011. [En línea]. Available: http://diydrones.com/profiles/blogs/arduimu-v3-is-finally-here. [Último acceso: 03 06 2016].

[26] «Atmega328 Datasheet,» [En línea]. Available: https://www.sparkfun.com/datasheets/Components/SMD/ATMega328.pdf. [Último acceso: 03 06 2016].

[27] «LCC IC Package Drawing,» [En línea]. Available: http://www.interfacebus.com/ic-package-lcc-drawing.html.

[28] «MPU-6000 and MPU-6050 Register Map and Descriptions,» InvenSense Inc., Borregas Ave, Sunnyvale, CA, 2012.

[29] D. K. Shaeffer, «MEMS inertial sensors: a tutorial overview,» *IEEE Communications Magazine,* vol. 51, nº 4, pp. 100-109, Abril 2013.

[30] «MPU-6000 and MPU-6050 Product Specification,» InvenSense Inc., Borregas Ave, Sunnyvale, CA,

2013.

[31] «InvenSense - MPU-6050,» 2016. [En línea]. Available: https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/. [Último acceso: 04 06 2016].

[32] A. Luque Estepa y J. M. Quero Reboul, Apuntes de Microsistemas, Sevilla: Departamento de Ingeniería Electrónica, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla, 2015.

[33] «Wikipedia - Matlab,» 11 05 2016. [En línea]. Available: https://es.wikipedia.org/wiki/MATLAB. [Último acceso: 03 06 2016].

[34] B. Stroustrup, The C++ Programming Language, 1997.

[35] «Armadillo - C++ linear algebra library,» [En línea]. Available: http://arma.sourceforge.net/. [Último acceso: 04 05 2016].

[36] N. Ho, «OpenCV vs. Armadillo vs. Eigen vs. more! Round 3: pseudoinverse test,» 25 11 2012. [En línea]. Available: http://nghiaho.com/?p=1726. [Último acceso: 04 05 2016].

[37] A. Filipe, M. Vincent, V. Volant y C. Kergueris, «Impact of die-attach materials,» *International Symposium on Inertial Sensors and Systems (ISISS),* pp. 1-2, 2014.

[38] «IMU errors and their effects,» 21 02 2014. [En línea]. Available: http://www.novatel.com/assets/Documents/Bulletins/APN064.pdf. [Último acceso: 05 06 2016].

[39] D. R. Greenheck, «Design and Characterization of a Low Cost MEMS IMU CLuster for Precision Navigation,» Marquette University, Milwaukee, Wisonsin, Agosto 2015.

[40] M. El-Diasty y S. Pagiatakis, «A Rigorous Temperature-Dependent Stochastic Modelling and Testing for MEMS-Based Inertial Sensor Errors,» *Sensors,* vol. 9, nº 11, pp. 8474-8489, 2009.

[41] D. Xia, S. Chen, S. Wang y H. Li, «Microgyroscope Temperature Effects and Compensation-Control Methods,» *Sensors,* vol. 9, nº 10, pp. 8349-8376, 2009.

[42] N. El-Sheimy, H. Hou y X. Niu, «Analysis and modeling of inertial sensors using Allan Variance,» *IEEE Transactions on Instrumentation and Measurement,* vol. 57, nº 1, pp. 140-149, Enero 2008.

[43] Y. Yuksel y H. Burak Kaygisiz, «Notes on Stochastic Errors of Low Cost MEMS Inertial Units,» [En línea]. Available: http://www.instk.org/web/static/bibliography/Introduction_to_Sensor_Errors.pdf. [Último acceso: 08 06 2016].

[44] G. Yanning, H. Fei, D. Shaohe, M. Guangfu y Z. Liangkuan, «Performance Analysis of MEMS Gyro and Improvement using Kalman filter,» *34th Chinese Control Conference (CCC),* pp. 4789-4794, 2015.

[45] M. Kirkko-Jaakkola, J. Collin y J. Takala, «Bias Prediction for MEMS Gyroscopes,» *IEEE Sensors Journal,* vol. 12, nº 6, pp. 2157-2163, Junio 2012.

[46] A. G. Quinchia, G. Falco, E. Falletti, F. Dovis y C. Ferrer, «A Comparison between Different Error Modeling of MEMS Applied to GPS/INS Integrated Systems,» *Sensors,* vol. 13, nº 8, pp. 9549-9588, 2013.

[47] J. Cemenska, Sensor Modelling and Kalman Filtering Applied to Satellite Attitude Determination,

University of California at Berkeley, 2003.

[48] P. Petkov y T. Slavov, «Stochastic Modeling of MEMS Inertial Sensors,» *CYBERNETICS AND INFORMATION TECHNOLOGIES,* vol. 10, nº 2, pp. 31-40, 2010.

[49] *Guide to Comparing Gyro and IMU Technologies - Micro-Electro-Mechanical Systems and Fiber Optic Gyros,* Middletown, RI: KVH Industries, Inc..

[50] «Wikipedia - matrices de rotación,» 02 06 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Matriz_de_rotaci%C3%B3n. [Último acceso: 23 04 2016].

[51] D. W. Allan, «Statistics of atomic frequency standards,» *Proceedings of the IEEE,* vol. 54, nº 2, pp. 221-230, Febrero 1966.

[52] L. C. Ng, On the application of Allan variance method for ring laser gyro performance characterization, Livermore, CA, USA: Lawrence Livermore Nat. Lab., Octubre 1993.

[53] J. Li y J. Fang, «Sliding Average Allan Variance for Inertial Sensor,» *IEEE Transactions on Instrumentation and Measurement,* vol. 62, nº 12, pp. 3291-3300, Diciembre 2013.

[54] FreescaleSemiconductor, «Allan Variance: Noise Analysis for Gyroscopes,» 02 2015. [En línea]. Available: http://cache.freescale.com/files/sensors/doc/app_note/AN5087.pdf. [Último acceso: 02 06 2016].

[55] IEEE Standard Specification Format Guide and Test Procedure for Single-Axis Interferometric Fiber Optic Gyros, IEEE Std 952-1997, 1998.

ANEXO A: CÓDIGO

A.1 Matlab

A.1.1 Allan Variance

ReadAllan.m

```
function [tiempos,varianzas,Temp] = ReadAllan()
%% Script to Read data from a file and use allan
tic
fileID = fopen('imu_log_media.txt');
C = textscan(fileID,'%f %f %f %f %f %f %f %f %f %f');
fclose(fileID);
% C{1} contains gx measurements, c{2} gy and so on
omega(:,1)=C{1};
omega(:,2)=C{2};
omega(:,3)=C{3};
omega(:,4)=C{4};
omega(:,5)=C{5};
omega(:,6)=C{6};
temperatura=C{10};

fs=200;
pts=100;
[VRW,ARW,BiasIns,T,sigma] = allan(omega,fs,pts);
% We obtain ARW in rad/s/sqrt(hz), VRW in g/sqrt(hz) and biases in rad/s
% and g. To calculate gyro and accelerometer variances we have to do
% ARW*sqrt(hz) and VRW*sqrt(hz). And hz is the sample frecuency (200hz).

% we show the allan deviation plot
format long
figure
loglog(T,sigma(:,1),T,sigma(:,2),T,sigma(:,3))
legend('gx','gy','gz')
xlabel('time(s)')
ylabel('Allan Deviation(rad/s)')
grid on
figure
loglog(T,sigma(:,4),T,sigma(:,5),T,sigma(:,6))
legend('ax','ay','az')
xlabel('Average time(s)')
ylabel('Allan Deviation (g)')
grid on

% varianzas vector has variance of gyro, acc,
% biasgyro y biasacc

varianzas(1,1:3)=(ARW*sqrt(200)).^2;
varianzas(2,1:3)=(VRW*sqrt(200)).^2;
varianzas(3,1:3)=BiasIns(1:3,1).^2;
varianzas(4,1:3)=BiasIns(4:6,1).^2;
```

```
% Temperature to asign to this variances
figure
hist(temperatura)
Temp=mean(temperatura);

% Correlation time
tiempos(1,1:3)=BiasIns(1:3,2);
tiempos(2,1:3)=BiasIns(4:6,2);

[filas,columnas]=size(temperatura);
t=1:1:filas;
figure
plot(t,temperatura)
xlabel('samples')
ylabel('Temperature (ºC)')
grid on

%Vargyro=(ARW*sqrt(200)).^2;
%Varacc=(VRW*sqrt(200)).^2;
%VarBiasgyro=BiasIns(1:3,1).^2;
%VarBiasacc=BiasIns(4:6,1).^2;

toc
end
```

allan.m

```
function [VRW,ARW,BiasIns,T,sigma] = allan(omega,fs,pts)
[N,M] = size(omega);     % calculate the data lenght
n = 2.^(0:floor(log2(N/2)))';    % determine largest bin size
maxN = n(end);
endLogInc = log10(maxN);
m = unique(ceil(logspace(0,endLogInc,pts)))';    % create log spaced vector
average factor
t0 = 1/fs;   % t0 = sample interval
T = m*t0;    % T = length of time for each cluster
theta = cumsum(omega)/fs; % integration of samples over time to obtain output
angle θ
sigma2 = zeros(length(T),M); % array of dimensions (cluster periods) X
(#variables)
for i=1:length(m)
    % loop over the various cluster sizes
for k=1:N-2*m(i)
    % implements the summation in the AV equation
sigma2(i,:) = sigma2(i,:) + (theta(k+2*m(i),:) - 2*theta(k+m(i),:) +
theta(k,:)).^2;
end
end
sigma2 = sigma2./repmat((2*T.^2.*(N-2*m)),1,M);
sigma = sqrt(sigma2);

%Calculate the Bias Instability for the 3 axis gyro(deg/s) and the
%corresponding time where this variance works
[BiasIns(1,1),posgx]=min(sigma(:,1));
BiasIns(1,2)=T(posgx);
[BiasIns(2,1),posgy]=min(sigma(:,2));
BiasIns(2,2)=T(posgy);
[BiasIns(3,1),posgz]=min(sigma(:,3));
BiasIns(3,2)=T(posgz);
```

```
[BiasIns(4,1),posax]=min(sigma(:,4));
BiasIns(4,2)=T(posax);
[BiasIns(5,1),posay]=min(sigma(:,5));
BiasIns(5,2)=T(posay);
[BiasIns(6,1),posaz]=min(sigma(:,6));
BiasIns(6,2)=T(posaz);


%Calculate the ARW (deg/s/sqrt(hz)=deg/sqrt(s))
% every log make a different T, so we have to find were is the
% value of T that is closer to 1.
[unuseful,columnas]=size(T);
aux = abs(T-ones(1,columnas));
indice=find(aux==min(aux));

ARW(:,1)=sigma(indice,1);
ARW(:,2)=sigma(indice,2);
ARW(:,3)=sigma(indice,3);
%Calculate the VRW (m/s/sqrt(s))
VRW(:,1)=sigma(indice,4);
VRW(:,2)=sigma(indice,5);
VRW(:,3)=sigma(indice,6);
end
```

A.1.2 Kalman Filter

ReadKalman.m

```
%% Script to Read data from a file and use RealKalman
tic
fileID = fopen('imu_log_PRUEBA1.txt');
C = textscan(fileID,'%f %f %f %f %f %f %f %f %f %f');
fclose(fileID);
% C{1} contains gx measurements, c{2} gy and so on
u(1,:)=C{1}; %gyro
u(2,:)=C{2};
u(3,:)=C{3};
zout(1,:)=C{4}; % accelerometer
zout(2,:)=C{5};
zout(3,:)=C{6};
zout(4,:)=C{7}; % magnetometer
zout(5,:)=C{8};
zout(6,:)=C{9};
temp=C{10};
[x,P]=Kalman(u,zout,temp);
toc
```

Kalman.m

```
function [x,P]=Kalman( u,zout,temp )
% u is the control input (gyroscope), zout is accelerometer and
% magnetometer measurement, but we have to calibrate the signal before use
% it. x and P are state vector and covariance matrix

%zout is z before calibration
%check the dimensions of u and z
[entradas,muestrasu] = size(u);
[observacion,muestrasz]= size(zout);
if(entradas==3 && observacion==6)
    if(muestrasu==muestrasz)
        N=muestrasu;
    else
        sprintf('Diferencia en número de muestras\n');
        return;
    end
else
    sprintf('número de elementos en u o z incorrectos\n');
    return;
end
% fs = 200hz  --> sample rate = 0.005s
At=0.005;

% We define the temperature
temperatura=temp;

    %Defining matrices
    %A
    A=eye(12);
    A(4:6,4:6)=zeros(3);
    A(1:3,4:6)=At*eye(3);
    A(4:6,7:9)=-eye(3);

    %B
    B=zeros(12,3);
    B(4:6,1:3)=eye(3);


% x and P are big matrices that contain the whole values of state vector
% and covariance matrix

xo=zeros(12,1);                                          % Initial conditions
for x and P
Po=pi*pi*eye(12);
Po(4:6)=0.01;
Po(7:9)=0.000008462;
Po(10:12)=0.00001;
x=zeros(12,N);
P=zeros(12,12*N);

[ x(:,1),P(:,1:12) ] = KF(xo,u(:,1),zout(:,1),Po,temperatura(1),A,B);

for i=2:N
    [ x(:,i),P(:,12*(i-1)+1:12*(i-1)+12) ] = KF(x(:,i-
1),u(:,i),zout(:,i),P(:,12*(i-2)+1:12*(i-2)+12),temperatura(i),A,B);
end

end
```

KF.m

```
function [ xest,Pest ] = KF(x,u,zout,P,temperatura,A,B)
%KF algorithm
    %  Calculate Q and R in function of temperature
    [Q, R,Biasvar]= VarianceQR(temperatura);
    %Prediction
    xest=A*x+B*u;
%x'(k+1)=Ax(k)+Bu(k)
    Pest=A*P*A'+B*Q*B';                                    %P'(k+1)=AP(k)A^T
+ BQB^T
    %Modifying bias variances
    Pest(7,7)=Pest(7,7)+Biasvar(1);
    Pest(8,8)=Pest(8,8)+Biasvar(2);
    Pest(9,9)=Pest(9,9)+Biasvar(3);
    Pest(10,10)=Pest(10,10)+Biasvar(4);
    Pest(11,11)=Pest(11,11)+Biasvar(5);
    Pest(12,12)=Pest(12,12)+Biasvar(6);

    %prediction of state vector values
    rx=xest(1,1);
    ry=xest(2,1);
    rz=xest(3,1);
    wx=xest(4,1);
    wy=xest(5,1);
    wz=xest(6,1);
    bgx=xest(7,1);
    bgy=xest(8,1);
    bgz=xest(9,1);
    bax=xest(10,1);
    bay=xest(11,1);
    baz=xest(12,1);
    %Correction or update
    if ((zout(4)~=100000)||(zout(5)~=100000)||(zout(6)~=100000))
        % There is magnetometer measurement
        z=magnet(zout,xest);

        %H
        H=zeros(5,12);
        H(1,2)= -cos(ry); H(1,10)= 1;
        H(2,1)= cos(rx)*cos(ry); H(2,2)= -sin(rx)*sin(ry); H(2,11)= 1;
        H(3,1)= -cos(ry)*sin(rx); H(3,2)= -cos(rx)*sin(ry); H(3,12)= 1;
        H(4,3)= -sin(rz);
        H(5,3)= -cos(rz);
        V=eye(5);

        %define relation between z and x with h(x) with the equation z=h(x)+v
        h(1,1)=-sin(ry)+bax;
        h(2,1)=cos(ry)*sin(rx)+bay;
        h(3,1)=cos(rx)*cos(ry)+baz;
        h(4,1)=cos(rz);
        h(5,1)=-sin(rz);

        y=z-h;                                             %y(k+1)=z(k+1)-
h(xest,0)

        K=Pest*H'/(H*Pest*H'+V*R*V');
%K(k+1)=P'(k+1)H^T/(HP'(k+1)H^T+VR(k+1)V^T)
```

```
        xest=xest+K*(y);
%x(k+1)=x'(k+1)+K(k+1)[y(k+1)]
        Pest=(eye(12)-K*H)*Pest;
%P(k+1)=[I-KH]P'(k+1)
    elseif ((zout(4)==100000)&&(zout(5)==100000)&&(zout(6)==100000))
        % There is not magnetometer measurement
        % Just read accelerometer measurement
        z2(1:3,1)=zout(1:3);

        %H
        H2=zeros(3,12);
        H2(1,2)= -cos(ry); H2(1,10)= 1;
        H2(2,1)= cos(rx)*cos(ry); H2(2,2)= -sin(rx)*sin(ry); H2(2,11)= 1;
        H2(3,1)= -cos(ry)*sin(rx); H2(3,2)= -cos(rx)*sin(ry); H2(3,12)= 1;

        V2=[eye(3), zeros(3,2)];

        %define relation between z and x with h(x) with the equation z=h(x)+v
        h2(1,1)=-sin(ry)+bax;
        h2(2,1)=cos(ry)*sin(rx)+bay;
        h2(3,1)=cos(rx)*cos(ry)+baz;

        y2=z2-h2;
%y(k+1)=z(k+1)-h(xest,0)

        K2=Pest*H2'/(H2*Pest*H2'+V2*R*V2');
%K(k+1)=P'(k+1)H^T/(HP'(k+1)H^T+VR(k+1)V^T)
        xest=xest+K2*(y2);
%x(k+1)=x'(k+1)+K(k+1)[y(k+1)]
        Pest=(eye(12)-K2*H2)*Pest;
%P(k+1)=[I-KH]P'(k+1)
    else
        sprintf('error en la medida del magnetometro\n');
    end
end
```

VarianceQR.m

```
function [Q,R,Biasvar] = VarianceQR(temp)
% This function returns the Matrices Q and R and the
% Bias variance, depending on the temperature and time. We will sum bias
% variance to covariance matrix

%This function recieve the actual temperature measured by the MPU-6000 and
the time when it's measured and
%returns the variation of the bias of the gyro and accelerometer depending
%on this factors. Q and R are the covariance matrices that contains the
variance of u and z on its diagonals

% This matrix contains the variance for 4 different temperatures
A=randn(12,4);

% t=-2 in raw 1, t=18 in raw 2 , t=33 in raw 3 and t=48 in raw 4
A(:,1)=1.0e-05 *[0.120182664207550   0.808012904036915   0.137916548785348
0.363961314341654   0.471210625715531   0.833419171094181   0.000299271414032
0.000258001995447   0.000061362749832   0.000155469127782   0.000076120809999
0.000933252339286];
A(:,2)=1.0e-05 *[0.120049479213774   0.145764649741926   0.087911860495552
0.370490608727625   0.354923532939822   0.783599377362748
```

```
0.000061575896382   0.000058201058891   0.000022419476809
0.000068214842821   0.000065419891193   0.000111426476794];
A(:,3)=1.0e-05 *[0.080586203384274   0.115727475484456   0.057764710951732
0.334138631030756   0.333571603146316   0.771110741070530   0.000099277666022
0.000097191126895   0.000033624649422   0.000098017650130   0.000113944967906
0.000479659353145];
A(:,4)=1.0e-05 *[0.116372970289676   0.178301924385547   0.155548416916616
0.378388991437665   0.374916110597673   1.801167286704032   0.000124309611446
0.000168029206333   0.000095087320340   0.000250130848670   0.000243383100933
0.000825537171562];

A(7,1)=A(7,1)*(1-exp(-2*0.005/4.515));
A(8,1)=A(8,1)*(1-exp(-2*0.005/33.905));
A(9,1)=A(9,1)*(1-exp(-2*0.005/23.235));
A(10,1)=A(10,1)*(1-exp(-2*0.005/23.235));
A(11,1)=A(11,1)*(1-exp(-2*0.005/135.6));
A(12,1)=A(12,1)*(1-exp(-2*0.005/10.91));

A(7,2)=A(7,2)*(1-exp(-2*0.005/49.485));
A(8,2)=A(8,2)*(1-exp(-2*0.005/92.925));
A(9,2)=A(9,2)*(1-exp(-2*0.005/72.22));
A(10,2)=A(10,2)*(1-exp(-2*0.005/92.925));
A(11,2)=A(11,2)*(1-exp(-2*0.005/81.92));
A(12,2)=A(12,2)*(1-exp(-2*0.005/63.67));

A(7,3)=A(7,3)*(1-exp(-2*0.005/8.48));
A(8,3)=A(8,3)*(1-exp(-2*0.005/18.055));
A(9,3)=A(9,3)*(1-exp(-2*0.005/81.92));
A(10,3)=A(10,3)*(1-exp(-2*0.005/43.625));
A(11,3)=A(11,3)*(1-exp(-2*0.005/29.895));
A(12,3)=A(12,3)*(1-exp(-2*0.005/23.235));

A(7,4)=A(7,4)*(1-exp(-2*0.005/92.925));
A(8,4)=A(8,4)*(1-exp(-2*0.005/56.13));
A(9,4)=A(9,4)*(1-exp(-2*0.005/72.22));
A(10,4)=A(10,4)*(1-exp(-2*0.005/26.355));
A(11,4)=A(11,4)*(1-exp(-2*0.005/14.035));
A(12,4)=A(12,4)*(1-exp(-2*0.005/153.835));


% Calculate the new variance depending on the Temperature with the equation
% v3=((T3-T1)/(T2-T1))(v2-v1)+v1 where v is variance vector and T is the
% temperature.
% subindex 1 and 2 refer to the two temperatures that choose the slope

Q=eye(3);
R=eye(5);
v3=zeros(12,1);
switch temp
    case -2
        v3=A(:,1);
    case 18
        v3=A(:,2);
    case 33
        v3=A(:,3);
    case 48
        v3=A(:,4);
    otherwise
        if temp<18
            T1=-2;
            T2=18;
            v1=A(:,1);
```

```
            v2=A(:,2);
            v3=((temp-T1)/(T2-T1))*(v2-v1)+v1;
        else if ((temp>18)&&(temp<33))
                T1=18;
                T2=33;
                v1=A(:,2);
                v2=A(:,3);
                v3=((temp-T1)/(T2-T1))*(v2-v1)+v1;
            else if temp>33
                    T1=33;
                    T2=48;
                    v1=A(:,3);
                    v2=A(:,4);
                    v3=((temp-T1)/(T2-T1))*(v2-v1)+v1;
                end
            end
        end
end



Q(1,1)=v3(1);
Q(2,2)=v3(2);
Q(3,3)=v3(3);
R(1,1)=v3(4);
R(2,2)=v3(5);
R(3,3)=v3(6);
Biasvar(1)=v3(7);
Biasvar(2)=v3(8);
Biasvar(3)=v3(9);
Biasvar(4)=v3(10);
Biasvar(5)=v3(11);
Biasvar(6)=v3(12);



% suposing magnetometer variance is 0.002
R(4:5,4:5)=0.002*eye(2);

% the three first elements of Biasvar are gyrobiasvar and the latter three
% are accelerometerbiasvar

end
```

magnet.m

```
function [ z ] = magnet( zout,xest )
%Recieves a 6 vector zout and returns a 5-vector with the accelerometer and
%the normalized magnetometer
%accelerometer
z(1:3,1)=zout(1:3);

%sustitute the values of xest on the rotation matrices
rx=xest(1,1);
ry=xest(2,1);
rz=xest(3,1);
%define the rotation matrices
Rx=[1 0 0;0 cos(rx) -sin(rx);0 sin(rx) cos(rx)];
```

```
Ry=[cos(ry) 0 sin(ry);0 1 0;-sin(ry) 0 cos(ry)];

%magnetometer
aux(1:3,1)=(Ry*Rx)*zout(4:6);
%check if the mz value is too high
if aux(3,1)>10000000, sprintf('valor excesivo de mz\n'), end
%da un error the conversion to logical from sym is not possible
%normalizar
z(4:5,1)=aux(1:2,1)./sqrt(aux(1,1)^2 + aux(2,1)^2);
end
```

A.1.3 Plot

Plotter.m

```
%% Script to read and print the output of KF made in C++
% This script is usefull to see how the filter act against a movement
clear all
tic
fileID = fopen('Salida_prueba2.log');
C = textscan(fileID,'%f %f %f %f %f %f %f %f %f %f %f %f');
fclose(fileID);
% C{1} to C{3} contain pitch, roll and yaw measurements
r(1,:)=C{1};
r(2,:)=C{2};
r(3,:)=C{3};
% C{7} to C{12} contain bias of gyro and accelerometer
bias(1,:)=C{7};
bias(2,:)=C{8};
bias(3,:)=C{9};
bias(4,:)=C{10};
bias(5,:)=C{11};
bias(6,:)=C{12};
[filas,columnas]=size(r);
t=1:1:columnas;
figure
plot(t,r(1,:),t,r(2,:),t,r(3,:))
legend('roll','pitch','yaw')
xlabel('samples')
ylabel('Euler angles (rad)')
grid on
figure
plot(t,bias(1,:),t,bias(2,:),t,bias(3,:))
legend('bias gx','bias gy','bias gz')
xlabel('samples')
ylabel('Bias gyro')
grid on
figure
plot(t,bias(4,:),t,bias(5,:),t,bias(6,:))
legend('bias ax','bias ay','bias az')
xlabel('samples')
ylabel('Bias accelerometer')
grid on

toc
```

plotvar.m

```
%% Script to plot the variances for different temperature
% values of gyro, accelerometer and its bias for different temperatures
% t=-2 in raw 1, t=18 in raw 2 , t=33 in raw 3 and t=48 in raw 4
var(:,1)=1.0e-05 *[0.120182664207550   0.808012904036915   0.137916548785348
0.363961314341654   0.471210625715531   0.833419171094181   0.000299271414032
0.000258001995447   0.000061362749832   0.000155469127782   0.000076120809999
0.000933252339286];
var(:,2)=1.0e-05 *[0.120049479213774   0.145764649741926   0.087911860495552
0.370490608727625   0.354923532939822   0.783599377362748
0.000061575896382   0.000058201058891   0.000022419476809
0.000068214842821   0.000065419891193   0.000111426476794];
var(:,3)=1.0e-05 *[0.080586203384274   0.115727475484456   0.057764710951732
0.334138631030756   0.333571603146316   0.771110741070530   0.000099277666022
0.000097191126895   0.000033624649422   0.000098017650130   0.000113944967906
0.000479659353145];
var(:,4)=1.0e-05 *[0.116372970289676   0.178301924385547   0.155548416916616
0.378388991437665   0.374916110597673   1.801167286704032   0.000124309611446
0.000168029206333   0.000095087320340   0.000250130848670   0.000243383100933
0.000825537171562];
t = [-2 18 33 48];
% plot gyro variance
figure
plot(t,var(1,:),t,var(2,:),t,var(3,:))
legend('gx','gy','gz')
xlabel('temperature (°C)')
ylabel('gyro variance ((rad/s)²)')
grid on
% plot acc variance
figure
plot(t,var(4,:),t,var(5,:),t,var(6,:))
legend('ax','ay','az')
xlabel('temperature (°C)')
ylabel('acc variance (g²)')
grid on
% plot gyro bias variance
figure
plot(t,var(7,:),t,var(8,:),t,var(9,:))
legend('gx','gy','gz')
xlabel('temperature (°C)')
ylabel('gyro bias variance ((rad/s)²)')
grid on
% plot acc bias variance
figure
plot(t,var(10,:),t,var(11,:),t,var(12,:))
legend('ax','ay','az')
xlabel('temperature (°C)')
ylabel('acc bias variance (g²)')
grid on
```

A.1 C++

KF.cpp

```
/*****************************************************
 *   Kalman filter for ArduIMU+ V3                   *
 *   Francisco González Mañero                       *
 *   Escuela Técnica Superior de Ingenieros                *
 *   GITT                                            *
 *****************************************************/

#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
//using armadillo library
#include <armadillo>
#include <fstream>
#include <iostream>

#define TIMESTEP 0.005     // Working at 200 Hz

using namespace arma;
using namespace std;

float temperature,T1,T2,T3;
float gx,gy,gz;                                     // control input
float ax,ay,az,mx,my,mz;                            // measurements
float rx,ry,rz,wx,wy,wz,bgx,bgy,bgz,bax,bay,baz;  // State Vector
int i=0;
int j,k;

    //Define the matrices
    mat A = eye<mat>(12,12);
    mat B = zeros<mat>(12,3);
    mat H = zeros<mat>(5,12);
    mat H2= zeros<mat>(3,12);      // without magnetometer
    mat C = eye<mat>(6,6);
    mat V = eye<mat>(5,5);
    mat V2= eye<mat>(3,5);
    mat Q = eye<mat>(3,3); // Process noise covariance
    mat R = eye<mat>(5,5); // Measurement noise covariance
    mat K(12,5);           // Kalman Gain
    mat K2(12,3);
    mat P(12,12);          // Covariance matrix
    mat Pest(12,12);       // Estiamtion of Covariance matrix
    mat Rx(3,3,fill::zeros);// Rotation matrices we are going to use below
    mat Ry(3,3,fill::zeros);
    mat I(12,12,fill::eye);
    mat Var(12,4,fill::zeros); // Matrix with the variances for diferent
temperatures(-20,0,30,40)

    //Define the vectors
    vec z(5);              // Measurement (Accelerometer and magnetometer)
    vec z2(3);             // Measuremnt without magnetometer
    vec y(5);              // y=z-h(x)
```

```
        vec y2(3);              // error without magnetometer
        vec x(12);              // State Vector (rx ry rz wx wy wz bgx bgy bgz
bax bay baz)
        vec u(3);               // Control input
        vec xest(12);           // Estimation of State Vector
        vec measurement(6);     // Measurement before calibration
        vec h(5);               // z=h(x)
        vec h2(3);              // without magnetometer
        vec Biasvar(6,fill::zeros);    // Increment of Bias variance for gyro
(0,1,2) and accelerometer (3,4,5)
        vec v1(12);                    // variance vectors used in noise()
function
        vec v2(12);
        vec v3(12);
        vec aux(6);


/*******************************************************
*              Functions definitions                   *
*******************************************************/


    void initKF()
        {
        // Initializes the matrices A and B
        A(3,3)=0;
        A(4,4)=0;
        A(5,5)=0;
        A(0,3)=TIMESTEP;
        A(1,4)=TIMESTEP;
        A(2,5)=TIMESTEP;
        A(3,6)=-1;
        A(4,7)=-1;
        A(5,8)=-1;
        B(3,0)=1;
        B(4,1)=1;
        B(5,2)=1;
        // Set x and P
        x.fill(0.0);
        P.eye();
        // The variance of angles are set to pi, the ones of rate are set to
0.01
        // The variance of gyro bias and acc bias are set using the turn-on to
turn-on bias from the datasheet
        P(0,0)=3.1415*3.1415;
        P(1,1)=3.1415*3.1415;
        P(2,2)=3.1415*3.1415;
        P(3,3)=0.01;
        P(4,4)=0.01;
        P(5,5)=0.01;
        P(6,6)=0.000008462;
        P(7,7)=0.000008462;
        P(8,8)=0.000008462;
        P(9,9)=0.00001;
        P(10,10)=0.00001;
        P(11,11)=0.00001;
        // Set the matrix of variances. temp=-2 in col 0, temp=18 in col 1 ,
temp=33 in col 2 and temp=48 in col 3
        // -2°C
        Var(0,0)=0.120182664207550;
        Var(1,0)=0.808012904036915;
        Var(2,0)=0.137916548785348;
```

```
Var(3,0)=0.363961314341654;
Var(4,0)=0.471210625715531;
Var(5,0)=0.833419171094181;
Var(6,0)=0.000299271414032*(1- pow(2.71828,(-2*TIMESTEP/4.515)));
Var(7,0)=0.000258001995447*(1- pow(2.71828,(-2*TIMESTEP/33.905)));
Var(8,0)=0.000061362749832*(1- pow(2.71828,(-2*TIMESTEP/23.235)));
Var(9,0)=0.000155469127782*(1- pow(2.71828,(-2*TIMESTEP/23.235)));
Var(10,0)=0.000076120809999*(1- pow(2.71828,(-2*TIMESTEP/135.6)));
Var(11,0)=0.000933252339286*(1- pow(2.71828,(-2*TIMESTEP/10.91)));

// 18°C
Var(0,1)=0.120049479213774;
Var(1,1)=0.145764649741926;
Var(2,1)=0.087911860495552;
Var(3,1)=0.370490608727625;
Var(4,1)=0.354923532939822;
Var(5,1)=0.783599377362748;
Var(6,1)=0.000061575896382*(1- pow(2.71828,(-2*TIMESTEP/49.485)));
Var(7,1)=0.000058201058891*(1- pow(2.71828,(-2*TIMESTEP/92.925)));
Var(8,1)=0.000022419476809*(1- pow(2.71828,(-2*TIMESTEP/72.22)));
Var(9,1)=0.000068214842821*(1- pow(2.71828,(-2*TIMESTEP/92.925)));
Var(10,1)=0.000065419891193*(1- pow(2.71828,(-2*TIMESTEP/81.92)));
Var(11,1)=0.000111426476794*(1- pow(2.71828,(-2*TIMESTEP/63.67)));

// 33°C
Var(0,2)=0.080586203384274;
Var(1,2)=0.115727475484456;
Var(2,2)=0.057764710951732;
Var(3,2)=0.334138631030756;
Var(4,2)=0.333571603146316;
Var(5,2)=0.771110741070530;
Var(6,2)=0.000099277666022*(1- pow(2.71828,(-2*TIMESTEP/8.48)));
Var(7,2)=0.000097191126895*(1- pow(2.71828,(-2*TIMESTEP/18.055)));
Var(8,2)=0.000033624649422*(1- pow(2.71828,(-2*TIMESTEP/81.92)));
Var(9,2)=0.000098017650130*(1- pow(2.71828,(-2*TIMESTEP/43.625)));
Var(10,2)=0.000113944967906*(1- pow(2.71828,(-2*TIMESTEP/29.895)));
Var(11,2)=0.000479659353145*(1- pow(2.71828,(-2*TIMESTEP/23.235)));

// 48°C
Var(0,3)=0.116372970289676;
Var(1,3)=0.178301924385547;
Var(2,3)=0.155548416916616;
Var(3,3)=0.378388991437665;
Var(4,3)=0.374916110597673;
Var(5,3)=1.801167286704032;
Var(6,3)=0.000124309611446*(1- pow(2.71828,(-2*TIMESTEP/92.925)));
Var(7,3)=0.000168029206333*(1- pow(2.71828,(-2*TIMESTEP/56.13)));
Var(8,3)=0.000095087320340*(1- pow(2.71828,(-2*TIMESTEP/72.22)));
Var(9,3)=0.000250130848670*(1- pow(2.71828,(-2*TIMESTEP/26.355)));
Var(10,3)=0.000243383100933*(1- pow(2.71828,(-2*TIMESTEP/14.035)));
Var(11,3)=0.000825537171562*(1- pow(2.71828,(-2*TIMESTEP/153.835)));
Var=Var*0.00001;
// initialize the variances with the value for 18 °C

v3=Var.col(1);
Q(0,0)=v3(0);
Q(1,1)=v3(1);
Q(2,2)=v3(2);
R(0,0)=v3(3);
R(1,1)=v3(4);
R(2,2)=v3(5);
R(3,3)=0.002;// variance of magnetometer
```

```
        R(4,4)=0.002;
        // Define values for Biasvar, the three first values for gyro bias
variance and the latter three for acc bias variance
        // The initial values are set by the covariance matrix
        Biasvar(0)=0;
        Biasvar(1)=0;
        Biasvar(2)=0;
        Biasvar(3)=0;
        Biasvar(4)=0;
        Biasvar(5)=0;
        }

    void noise()
        {
        // This function choose the values of Q and R depending on the
temperature
        // The values in the diagonals of Q are related to the gyro variance
and the ones in R are related to the acc and the magnet's

        // We will access this value through a matrix with the variances and
the temperature
        // if our temperature is not one of the values we have, we obtain this
variance with the equation
        // v3=((T3-T1)/(T2-T1))(v2-v1)+v1 where T is temperature and v is
variance vector. 1 and 2 indicate the values that form the
        // straight line to calculate 3.

        switch ((int)temperature)
                {
                case -2:
                        {v3=Var.col(0);
                        break;}
                case 18:
                        {v3=Var.col(1);
                        break;}
                case 33:
                        {v3=Var.col(2);
                        break;}
                case 48:
                        {v3=Var.col(3);
                        break;}
                default:
                        {
                        // If temperature is not one of this values, we
calculate the variances vector v3
                        if(temperature<18)
                                {
                                T1=-2;
                                T2=18;
                                v1=Var.col(0);
                                v2=Var.col(1);
                                v3=((temperature-T1)/(T2-T1))*(v2-v1)+v1;
                                }
                        else if ((temperature>18)&&(temperature<33))
                                {
                                T1=18;
                                T2=33;
                                v1=Var.col(1);
                                v2=Var.col(2);
                                v3=((temperature-T1)/(T2-T1))*(v2-
v1)+v1;

                                }
```

```
                        else if (temperature>33)
                                {
                                T1=33;
                                T2=48;
                                v1=Var.col(2);
                                v2=Var.col(3);
                                v3=((temperature-T1)/(T2-T1))*(v2-v1)+v1;
                                }
                        break;
                        }
                }
        Q(0,0)=v3(0);
        Q(1,1)=v3(1);
        Q(2,2)=v3(2);
        R(0,0)=v3(3);
        R(1,1)=v3(4);
        R(2,2)=v3(5);
        R(3,3)=0.002;// variance of magnetometer
        R(4,4)=0.002;

        // Define values for Biasvar, the three first values for gyro bias
variance and the latter three for acc bias variance


        Biasvar(0)=v3(6);
        Biasvar(1)=v3(7);
        Biasvar(2)=v3(8);
        Biasvar(3)=v3(9);
        Biasvar(4)=v3(10);
        Biasvar(5)=v3(11);
        }

    void BiasIns()
        {
        // This function vary the 6 last elements on the diagonal of P
depending of the variation of bias in the gyro and the acc
        Pest(6,6)=Pest(6,6)+Biasvar(0);               // Variance of the gyro
bias(6,7 and 8) and variance of the accelerometer bias(9,10 and 11)
        Pest(7,7)=Pest(7,7)+Biasvar(1);
        Pest(8,8)=Pest(8,8)+Biasvar(2);
        Pest(9,9)=Pest(9,9)+Biasvar(3);
        Pest(10,10)=Pest(10,10)+Biasvar(4);
        Pest(11,11)=Pest(11,11)+Biasvar(5);
        }


    void magnet()
        {
        // This function takes the measurement vector and change the 3 last
values (magnetometer)

        vec aux(3);
        vec aux2(3);

        z(0)=measurement(0);    // No changes in the accelerometer
        z(1)=measurement(1);
        z(2)=measurement(2);

        Rx(0,0)=1;              // Set the rotation matrices
        Rx(1,1)=cos(rx);
        Rx(1,2)=-sin(rx);
        Rx(2,1)=sin(rx);
```

```
        Rx(2,2)=cos(rx);

        Ry(0,0)=cos(ry);
        Ry(0,2)=sin(ry);
        Ry(1,1)=1;
        Ry(2,0)=-sin(ry);
        Ry(2,2)=cos(ry);

        aux(0)=measurement(3);
        aux(1)=measurement(4);
        aux(2)=measurement(5);

        aux2=Ry*Rx*aux;

        if(fabs(aux2(2))>50000*(fabs(aux2(1))+fabs(aux2(0))))
                {
                printf("Valor de mz muy grande en la iteración %d\n",i);
                }
        z(3)=aux2(0)/(sqrt(aux2(0)*aux2(0) +
aux2(1)*aux2(1)));                              // Normalize
        z(4)=aux2(1)/(sqrt(aux2(0)*aux2(0) + aux2(1)*aux2(1)));
        }



    void checkP()
        {
        // This function checks if the covariance matrix P is symmetric and
all terms of the diagonal are positive
        for(j=0;j<12;j++)
                {
                for(k=0;k<12;k++)
                        {
                        if((j==k)&&(P(k,j)<=0))
                                {
                                printf("Elemento de la diagonal 0 o menor que
0\n");

                                break;
                                }
                        else if((j!=k)&&(fabs(P(k,j)-P(j,k))>0.00001))
                                {
                                printf("Matriz P no simétrica\n");
                                break;
                                }
                        }
                }
        }


    void Kalman()
        {

        u(0)=gx;
        u(1)=gy;
        u(2)=gz;

        // Prediction
        xest = A*x + B*u;                       // xest=A*x + B*u
        Pest = A*P*A.t() + B*Q*B.t();           // Pest=A*P*A' + B*Q*B'
        BiasIns();                              // Modify P depending on the
temperature (Bias Instability)
```

```
        rx=xest(0);                                  // Predict the values of State
Vector
        ry=xest(1);
        rz=xest(2);
        wx=xest(3);
        wy=xest(4);
        wz=xest(5);
        bgx=xest(6);
        bgy=xest(7);
        bgz=xest(8);
        bax=xest(9);
        bay=xest(10);
        baz=xest(11);
        // Update

        measurement(0)=ax;                           // Measurement of the
accelerometer and magnetometer
        measurement(1)=ay;
        measurement(2)=az;
        measurement(3)=mx;
        measurement(4)=my;
        measurement(5)=mz;


        if(!((measurement(3)==100000.0)&&(measurement(4)==100000.0)&&(measurem
ent(5)==100000.0)))
                {
                h(0)=-sin(ry)+bax;                   // This
functions show the relation between z and x
                h(1)=cos(ry)*sin(rx)+bay;
                h(2)=cos(rx)*cos(ry)+baz;
                h(3)=cos(rz);
                h(4)=-sin(rz);

                magnet();                            // Calibration
of the magnetometer values and fill z with the measurement

                y=z-h;


                H(0,1)= -cos(ry);                    // H is the
jacobian of h against x
                H(0,9)=1;
                H(1,0)=cos(rx)*cos(ry);
                H(1,1)=-sin(rx)*sin(ry);
                H(1,10)=1;
                H(2,0)=-cos(ry)*sin(rx);
                H(2,1)=-cos(rx)*sin(ry);
                H(2,11)=1;
                H(3,2)=-sin(rz);
                H(4,2)=-cos(rz);

                K = Pest*H.t()*inv(H*Pest*H.t()+V*R*V.t());
// K=Pest*H'/(H*Pest*H'+V*R*V')
                x = xest + K*(y);
// x=xest+K*(z-h(xest,0))      x is x_k+1 and the same happens with P
                P = (I-K*H)*Pest;
// P=(I-K*H)*Pest
                }
        else
if((measurement(3)==100000.0)&&(measurement(4)==100000.0)&&(measurement(5)==1
00000.0))
```

```
                {
                // If enter here, there's not magnetometer measurement
                h2(0)=-sin(ry)+bax;                             // This
functions show the relation between z and x
                h2(1)=cos(ry)*sin(rx)+bay;
                h2(2)=cos(rx)*cos(ry)+baz;

                z2(0)=measurement(0);
                z2(1)=measurement(1);
                z2(2)=measurement(2);

                y2=z2-h2;


                H2(0,1)= -cos(ry);                              // H2 is the
jacobian of h2 against x
                H2(0,9)=1;
                H2(1,0)=cos(rx)*cos(ry);
                H2(1,1)=-sin(rx)*sin(ry);
                H2(1,10)=1;
                H2(2,0)=-cos(ry)*sin(rx);
                H2(2,1)=-cos(rx)*sin(ry);
                H2(2,11)=1;


                K2 = Pest*H2.t()*inv(H2*Pest*H2.t()+V2*R*V2.t());
// K=Pest*H'/(H*Pest*H'+V*R*V')
                x = xest + K2*(y2);
// x=xest+K*(z-h(xest,0))      x is x_k+1 and the same happens with P
                P = (I-K2*H2)*Pest;
// P=(I-K*H)*Pest
                }
        else
                {
                printf("ERROR: Medida del magnetómetro problemática\n");
                }
        rx=x(0);                                            // Update the values of
State Vector
        ry=x(1);
        rz=x(2);
        wx=x(3);
        wy=x(4);
        wz=x(5);
        bgx=x(6);
        bgy=x(7);
        bgz=x(8);
        bax=x(9);
        bay=x(10);
        baz=x(11);
        checkP();

        noise();                                    // Define the Q and R matrices
depending on temperature (ARW and VRW)
        }




/*********************************************************************
*               Here is where the main function starts                *
*********************************************************************/
```

```
    int main ()
        {
        initKF();
        char buffer[500];
        char bufferout[500];
        ifstream inputFile("imu_logger/imu_log_PRUEBA2.txt");
        // The first 3 values are the control input and the following 6 are
the measurements

        ofstream
outputF("/home/fran/MATLAB/code/TFG/KF/FilterTemp/Salida_prueba2.log");
        // The output, that is the state vector in every moment
        while (!inputFile.eof())
                {
                inputFile.getline(buffer,500);
                sscanf(buffer,"%f %f %f %f %f %f %f %f %f
%f\n",&gx,&gy,&gz,&ax,&ay,&az,&mx,&my,&mz,&temperature);
                // We recieve the values in rad/s ,g and mG
                Kalman(); // Here is where the KF algorithm is
                i++;
                sprintf(bufferout,"%f %f %f %f %f %f %f %f %f %f %f
%f",rx,ry,rz,wx,wy,wz,bgx,bgy,bgz,bax,bay,baz);
                outputF<<bufferout<<endl;
                }
        cout << "Matriz de covarianza P \n" <<P <<endl;
        outputF.close();
        return 1;
      }
```

imu_logger.cpp

```
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include "arduimu_v3.hpp"

#define DEG2RAD(X) ( (X) * 0.01745329251)

static volatile bool keepRunning = true;

void intHandler(int dummy)
{
    keepRunning = false;
}

int main( int argc, char **argv)
{
        // Setup control+C handler
        signal(SIGINT, intHandler);

        // Check input parameters
        if(argc != 2)
        {
                printf("Error: wrong parameters. Usage:\n\timu_logger
/dev/ttyUSBx\n");
                return -1;
```

```
	}

	// Setup IMU
	ArduimuDev imuDev(argv[1]);
	ArduImuRawData rawData;

	// Open log file
	char fileName[480];
	struct timespec time;
	clock_gettime(CLOCK_REALTIME, &time);
	sprintf(fileName, "imu_log_%lu.txt", time.tv_sec);
	FILE *pf = fopen(fileName, "w");
	if(pf == NULL)
	{
		printf("Error: cannot open file %s for writing\n", fileName);
		return -1;
	}

	// Read IMU data
	int count = 0;
	while(keepRunning)
	{
		// Read raw imu data
		if(imuDev.readData(rawData) != 1)
			continue;

		// Save on disk
		if(rawData.mag_update)
		{
			fprintf(pf, "%f %f %f %f %f %f %f %f %f %f\n",
DEG2RAD(rawData.gyr_x), DEG2RAD(rawData.gyr_y), DEG2RAD(rawData.gyr_z),

				rawData.acc_x, rawData.acc_y, rawData.acc_z,

				rawData.mag_x, rawData.mag_y, rawData.mag_z,

				rawData.temp);
		}
		else
		{
			fprintf(pf, "%f %f %f %f %f %f 10000.0 10000.0 10000.0
%f\n", DEG2RAD(rawData.gyr_x), DEG2RAD(rawData.gyr_y),
DEG2RAD(rawData.gyr_z),

				rawData.acc_x, rawData.acc_y, rawData.acc_z,

				rawData.temp);
		}

		// Display some debug info
		count++;
		if(count == 10)
		{
			count = 0;
			printf("%f %f %f %f %f %f %f %f %f %f\n",
DEG2RAD(rawData.gyr_x), DEG2RAD(rawData.gyr_y), DEG2RAD(rawData.gyr_z),

				rawData.acc_x, rawData.acc_y, rawData.acc_z,

				rawData.mag_x, rawData.mag_y, rawData.mag_z,

				rawData.temp);
```

```
                }
        }

        // Close log file
        fclose(pf);

        return 0;
```

Arduimu_v3.hpp

```
#ifndef __ARDUIMU_V3_HPP__
#define __ARDUIMU_V3_HPP__

#include <termios.h>
#include <unistd.h>
#include <stdint.h>
#include <fcntl.h>
#include <stdio.h>

//!Arduimu raw data
struct ArduImuRawData
{
        uint32_t        time;
        bool            accgyro_update;
        bool            mag_update;
        double          acc_x;                  // Accel in g
        double          acc_y;
        double          acc_z;
        double          gyr_x;                  // Gyro in ยน/s
        double          gyr_y;
        double          gyr_z;
        double          mag_x;                  // Mag in mG
        double          mag_y;
        double          mag_z;
        double          temp;                   // temperature
};

class ArduimuDev
{
public:

        //!Intialize the serial port to the given values
        ArduimuDev(const char *pDev)
        {
                struct termios my_termios;

                // Open the port in read-write mode
                m_portHandler = open(pDev, O_RDWR | O_NOCTTY);
                if(m_portHandler < 0)
                        return;

                // Get the port attributes and flush all data on queues
                tcgetattr(m_portHandler, &my_termios);
                tcflush(m_portHandler, TCIOFLUSH);
```

```
                // Setup the communication
                my_termios.c_iflag &= ~(BRKINT | IGNPAR | PARMRK | INPCK
|ISTRIP | IXON | INLCR | IGNCR | ICRNL);
                my_termios.c_iflag |= IGNBRK | IXOFF;
                my_termios.c_oflag &= ~(OPOST);
                my_termios.c_cflag |= CLOCAL | CREAD;
                my_termios.c_cflag &= ~PARENB;
                my_termios.c_cflag |= CS8;
                my_termios.c_cflag &= ~CSTOPB;
                my_termios.c_cflag &= ~CRTSCTS;
                my_termios.c_lflag &= ~(ECHO | ECHOE | ECHOK | ECHONL | ICANON
| NOFLSH | TOSTOP | ISIG | IEXTEN);
                my_termios.c_cc[VMIN]=1;        //Each simple read call will be
blocked until recive at least one byte
                my_termios.c_cc[VTIME]=0;       //No timeout for reading
                cfsetispeed(&my_termios, B115200);
                cfsetospeed(&my_termios, B115200);
                tcsetattr(m_portHandler, TCSANOW, &my_termios);
        }

        //!Default destructor
        ~ArduimuDev(void)
        {
                ArduimuDev::finish();
        }

        //!Read (blocking) a complete message from the serial port
        int readData(ArduImuRawData &data)
        {
                int dataRead, size;
                char c;

                // Find the biginning of the stream 0x23
                c = 0;
                while(c != '#')
                {
                        dataRead = read(m_portHandler, &c, 1);
                        if(dataRead < 0)
                                return -5;
                }

                // Conitue reading until 0x2A is received
                c = 0;
                size = 0;
                while(c != '*')
                {
                        dataRead = read(m_portHandler, &c, 1);
                        if(dataRead < 0)
                                return -5;

                        m_readBuff[size++] = c;
                        if(size == 256)
                                size = 0;
                }

                // Parse message
                if(size != 25 && size != 19)
                        return -1;
                data.time =  ((uint32_t *)m_readBuff)[0];
                data.acc_x = -((int16_t *)m_readBuff)[2]/8192.0;
                data.acc_y = -((int16_t *)m_readBuff)[3]/8192.0;
```

```
                data.acc_z = ((int16_t *)m_readBuff)[4]/8192.0;
                data.gyr_x = -((int16_t *)m_readBuff)[5]/65.5;
                data.gyr_y = -((int16_t *)m_readBuff)[6]/65.5;
                data.gyr_z = ((int16_t *)m_readBuff)[7]/65.5;
                data.temp = (((int16_t *)m_readBuff)[8] + 521)/340.0 + 21.0;
                data.mag_x = 0;
                data.mag_y = 0;
                data.mag_z = 0;
                data.accgyro_update = true;
                data.mag_update = false;
                if(size == 25)
                {
                        data.mag_x = -((int16_t *)m_readBuff)[9];
                        data.mag_y = -((int16_t *)m_readBuff)[10];
                        data.mag_z = ((int16_t *)m_readBuff)[11];
                        data.mag_update = true;
                }

                return 1;
        }

        //!Finish the serial communication
        void finish(void)
        {
                // Close the port
                if(m_portHandler > 0)
                {
                        close(m_portHandler);
                        m_portHandler = 0;
                }
        }

protected:

        //!Serial port hadler
        int m_portHandler;

        //!Reading buffer
        char m_readBuff[256];
};
```

ANEXO B: ALLAN VARIANCE

B.1 Introducción

El Allan Variance fue desarrollado por David Allan en 1966 para estudiar la estabilidad frecuencial de osciladores de precisión [51]. Un análisis de Allan Variance lo que hace básicamente es, tomando una gran cantidad de muestras en dominio temporal, realizar un análisis de los errores estocásticos que afectan al dispositivo en cuestión. El resultado del análisis es una gráfica donde podremos identificar los diferentes errores que afectan al dispositivo, dependiendo de la inclinación de la misma y de otros factores que definiremos más adelante.

Es un método muy utilizado en la caracterización de los ruidos estocásticos de los sensores inerciales [42] y se combina con otros métodos como técnicas de muestreado de *clústers* [52]. Para la caracterización de los sensores inerciales, es necesario contar con un gran número de muestras, ya algunos de los errores tienen un tiempo de correlación muy alto.

B.2 Metodología

Existen diversos tipos de métodos de Allan Variance, pero en este trabajo el que utilizaremos será el *fully overlapped Allan Variance* [53], que definiremos a continuación. La descripción de la metodología está basada en [54] al igual que la implementación del Allan Variance en Matlab que mostramos en el anexo anterior. La descripción está orientada a un giróscopo, pero podría hacerse de igual forma para cualquier sensor inercial.

1. Para comenzar, obtenemos el historial temporal del giróscopo, que llamaremos Ω(t). El sampleo se realizará con un periodo de muestreo τ_0 y un número de muestras *N*.
2. Escogemos una τ tal que $\tau = m\tau_0$. Donde m será un valor menor que (N-1)/2.
3. El historial temporal lo dividimos en *clústers* de tamaño τ, de modo que la diferencia entre el comienzo de un *clúster* y el comienzo del siguiente sea τ_0 como expresa la figura B-0-1.
4. Una vez que se generan los *clústers,* se procede al cálculo del Allan Variance.

   Para cada muestra, se calcula θ, tal que (en discreto) es la suma acumulativa de esa muestra y todas las anteriores multiplicadas por el periodo de muestreo. Tendremos por tanto una secuencia de N valores de θ, considerando que θ_k es la relacionado a la muestra *k*. Una vez calculado esto, utilizamos la ecuación (B-1) para el cálculo del Allan Variance.

$$\sigma^2(\tau) = \frac{1}{2\tau^2(N-2m)} \sum_{k=1}^{N-2m} (\theta_{k+2m} - 2\theta_{k+m} + \theta_k)^2 \tag{B-1}$$

5. Se calcula el Allan Deviation, que no es más que la raíz cuadrada del Allan Variance, y se repite el proceso para diferentes valores de m, en este caso los valores de m estarán logarítmicamente espaciados. Los valores de la Allan Deviation se imprimirán y de este modo obtendremos la gráfica en cuestión.

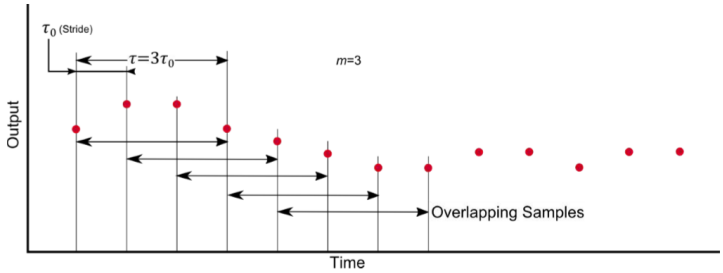


Figura B-0-1. *Clústers* para m=3.

Lo que obtenemos es una gráfica con la forma de la figura B-0-2. En esta gráfica podemos ver cuáles son los principales errores estocásticos que afectan a los sensores inerciales. En el capítulo 5 estudiaremos más a fondo estos errores, y basándonos en esta figura, conseguiremos calcularlos y estimarlos.

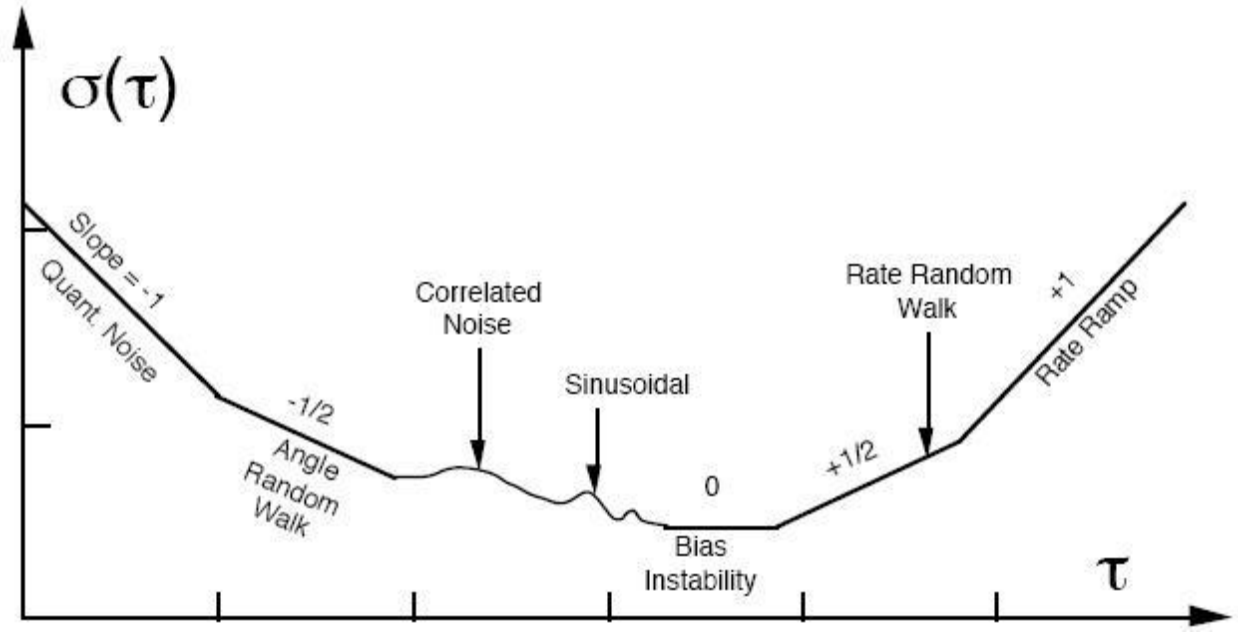


Figura B-0-2. Resultado del análisis de Allan Variance [55].