Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

BISKY TEAM

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

PROCESADOR AERONÁUTICO DE TRAYECTORIA Y ORIENTACIÓN

Alumno/Alumna: Elgezabal Núñez, Aritz

Director/Directora: Ortiz Álvarez-Cienfuegos, Javier

Curso: 2020 - 2021

Fecha: Junio, 2021>

Índice

1. Introducción ... 3
2. Contexto ... 3
3. Objetivos y planificación temporal ... 4
4. Desarrollo ... 5
4.1 Aeromodelo ... 5
4.1.1 Funcionamiento de un Aeromodelo de ala fija ... 5
4.2 Perfil aerodinámico ... 7
4.3 Aviónica ... 13
4.3.1 Hardware ... 13
4.3.2 Software ... 20
5 Resultados ... 42
5.1 Validación del algoritmo ... 42
5.1.1 XSens MTi-7 ... 42
5.1.2 MPU-9250 ... 42
5.1.3 Prueba en banco de pruebas ... 43
5.1.4 Pruebas dinámicas ... 45
5.2 Tiempos de los subprocesos ... 46
5.3 Prueba con aeromodelo ... 48
6 Conclusiones ... 52
6.1 Autonomía ... 52
6.1.1 Turbofan o Turbojet ... 52
6.1.2 Paneles solares ... 52
6.2 Aeromodelo ... 53
6.2.1 Estabilidad ... 53
6.2.2 Sustentación, entrada en pérdida e hipersustentadores ... 53
6.2.3 Velocidad ... 53
6.3 Posicionamiento global ... 54
6.3.1 Precisión ... 54
6.3.2 Pérdida de la señal ... 54
6.4 Algoritmos de control ... 55
6.4.1 Control Guiado ... 55

6.4.2 Control de trayectoria 55
6.4.3 Modo Repetidor 55
6.5 Lazo de control de velocidad aerodinámica 56
7 Bibliografía 57
7.1 Aerodinámica y Aeronáutica 57
7.2 Software y Control 57
7.3 Hardware 57
8 Anexos 58
8.1 Micromod Estándar M.2 58
8.2 Filtro de orientación 69
8.2.1 Representación por "Cuaterniones" 69
8.2.2 Orientación por ratio angular 71
8.2.3 Orientación por observación vectorial 71
8.2.4 Algoritmo de fusión de filtros 74
8.2.5 Compensación de la distorsión magnética 76
8.2.6 Compensación de la deriva giroscópica 76
8.3 XSens MTi-7 78
8.3.1 Procedimiento de envío SPI (MTi 1 – series) 78
8.3.2 MTSSP Protocolo serial síncrono 78
8.4 Microcontrolador ATMEGA2560 83
8.4.1 Diagrama de bloques del microcontrolador 83
8.4.2 Periférico SPI 84
8.4.3 Periférico USART 88
8.4.4 Especificaciones del TWI compatible con I2C 94
8.5 Módulo receptor GNSS 99
8.6 MARG 112

1. Introducción

El equipo BiSky Team de la Escuela de Ingeniería de Bilbao se plantea el uso de aeromodelos autogobernados para mantener una supervisión activa durante los lanzamientos de cohetes, así como para el reconocimiento de áreas poco accesibles con intención de localizar los cohetes cuando ya hayan aterrizado.

La idea del proyecto es la siguiente: la aviónica dentro del aeromodelo, mediante una unidad de instrumentación y un sistema G.N.S.S., es capaz de conocer y corregir tanto su posición como orientación y velocidad respecto al plano terrestre, actuando sobre el motor y las superficies de control. Antes del despegue se carga una lista de puntos G.N.S.S. por los que se encargará de navegar en orden de forma automática. En caso necesario, el supervisor puede cambiar el modo de vuelo y manejar remotamente la aeronave. Esta aeronave servirá de plataforma para diferentes cargas (radio repetidores, cámaras, experimentos… etc.) durante su vuelo.

2. Contexto

Durante los lanzamientos de cohetes de BiSky Team, el equipo se ha encontrado con retos a la hora de recuperar los lanzadores. El reto principal a superar es la pérdida de cobertura del radioenlace de telemetría. Cuando el cohete cae es posible que caiga detrás de un obstáculo y que éste produzca sombra (electromagnética). El cohete transmite su posición GNSS vía enlace de telemetría, es por eso que se presenta esta solución.

El aeromodelo con control automático puede realizar la búsqueda del aparato una vez haya caído, para así servir de plataforma a un repetidor entre el cohete y la estación de tierra. Una vez encontrado el aparato, el aeromodelo puede permanecer en el aire sin necesidad de intervención, simplificando y facilitando así la labor de búsqueda y recuperación.

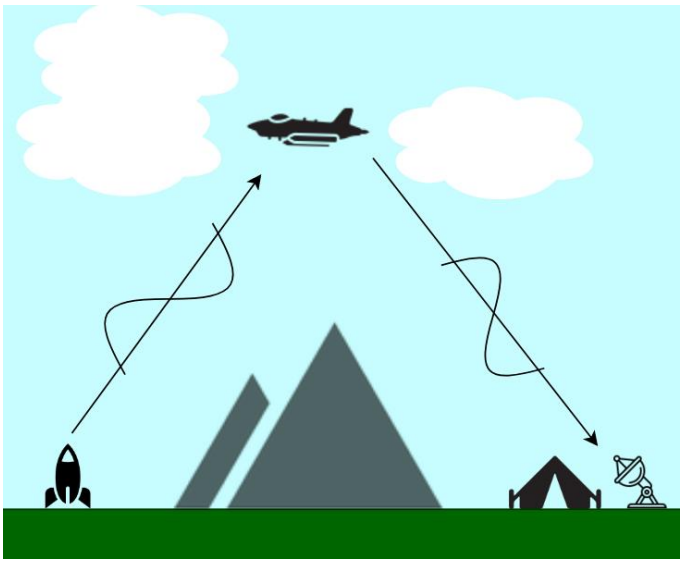


Ilustración 1: Propuesta al problema del radioenlace

3. Objetivos y planificación temporal

En este proyecto, centrado en la electrónica industrial y automática, los objetivos generales son:

1. Diseñar la aviónica de un aeromodelo.
    1. Escoger el tipo de aeromodelo.
    2. Escoger la instrumentación.
    3. Escoger y programar el filtro digital de orientación.
    4. Desarrollar un programa funcional y universal.

Se comienza recopilando información teórica acerca de aerodinámica general, de la instrumentación utilizada en la electrónica de aviación y métodos de orientación electrónicos. Se busca una relación de compromiso entre los métodos más económicos y los más computacionalmente eficientes.

Una vez escogidos los métodos a utilizar, se accede a información sobre aeronavegación y se simplifica el problema. Esta simplificación trae consigo una dificultad añadida, puesto que el sistema de referencia del sistema de orientación y el de trayectoria no tienen por qué ser el mismo, por lo tanto, será necesario buscar una forma de relacionar, directa o indirectamente, ambos sistemas de referencia.

En la siguiente etapa se diseñan las maniobras de vuelo y el/los lazo(s) de control que gobernarán la aeronave. El modelo del avión se hará lo más simple y universal posible, con el fin de que pueda ser reutilizado en futuras simulaciones externas a este proyecto.

Después, se diseña un aeromodelo sencillo, cuyas principales características serán su estabilidad y lenta reacción. Esto facilitará las primeras pruebas y la afinación de los PIDs que se programarán.

Finalmente, se termina el software y se realizan tanto pruebas estáticas como dinámicas del prototipo.

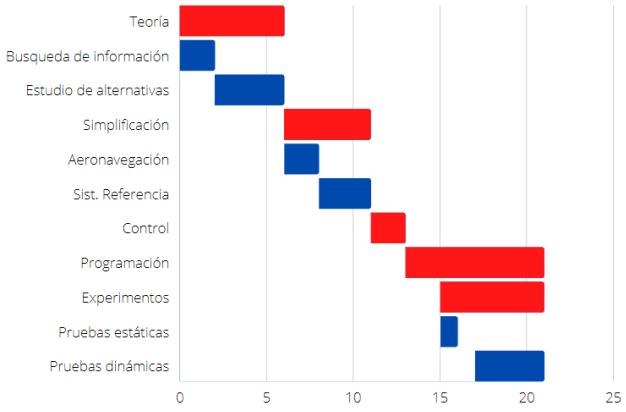


Ilustración 2: Diagrama Gantt, en semanas

4. Desarrollo

El desarrollo del proyecto se divide en 2 partes principalmente: primero el diseño y desarrollo de un aeromodelo de ala fija, segundo el diseño y programación de la aviónica que gobierna el aparato.

4.1 Aeromodelo

Con la intención de partir de un aeromodelo cuya estabilidad es conocida, se plantea diseñar desde cero un aeromodelo de ala fija para baja velocidad de vuelo. Para realizar el diseño se realiza una iteración de diseño y en función de los resultados de las simulaciones, se modifican las diferentes dimensiones y posiciones de las superficies aerodinámicas. Esto inicia un nuevo proceso iterativo hasta el momento en que el aeromodelo cumpla los requisitos exigibles para posibilitar el desempeño de sus funciones.

Para simular el diseño del aeromodelo se utiliza el programa XFLR5. Primero se introducen diferentes perfiles aerodinámicos para analizarlos mediante el programa. Después, los datos recabados por el programa se utilizan para realizar las simulaciones pertinentes a la hora de diseñar el avión. Antes de empezar con el diseño del aeromodelo es importante entender el funcionamiento de los aviones.

4.1.1 Funcionamiento de un Aeromodelo de ala fija

También conocidos como aeroplanos o aviones, son vehículos capaces de volar siendo más pesados que el aire, cuya característica principal para poder hacerlo es el uso de alas fijas. La razón de que las alas permitan al avión volar es el concepto aerodinámico de "Downwash".

En aerodinámica en general se define **downwash** como el cambio en la dirección del aire desviado por la acción aerodinámica de una superficie (como la pala del rotor de un helicóptero o el ala de un avión en movimiento), como parte del proceso de producción de sustentación. Es por eso que un ala debe verse como una herramienta diseñada para lanzar el aire a su paso hacia abajo. Esto es algo bastante intuitivo si miramos la forma del ala y vemos lo que ocurre con el efecto Coanda en una cucharilla.

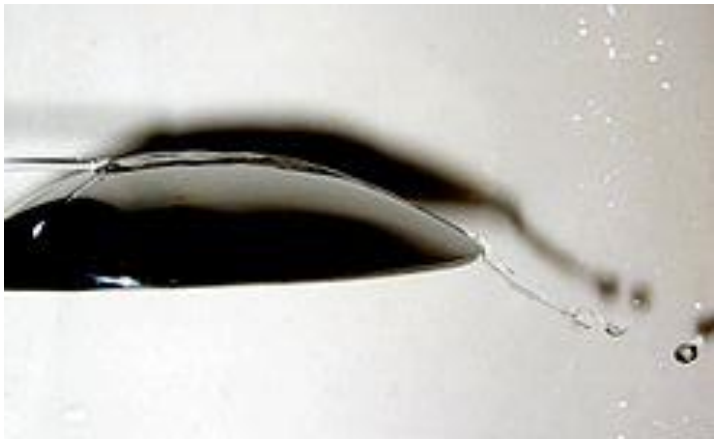


Ilustración 3: Efecto Coanda demostrado con una cuchara y agua.

"El efecto Coanda es el fenómeno físico por el cual una corriente de fluido, gaseosa o líquida, tiende a ser atraída por una superficie vecina a su trayectoria. El término fue acuñado por Albert Metral en honor al ingeniero aeronáutico rumano **Henri Coandă,** quien descubrió el efecto en su prototipo de un avión a reacción." – Wikipedia 06/04/2021

Que un avión se sostenga en el aire o que un perfil aerodinámico genere sustentación es un ejemplo perfecto de la aplicación de la tercera ley de movimiento de Newton: Al crear el flujo descendente, como reacción aparece una fuerza de igual magnitud y sentido opuesto, que se aplica sobre el avión: es la fuerza de sustentación. La sustentación es una fuerza y, como tal, se puede medir aplicando la segunda ley de Newton.

La sustentación producida por un ala es proporcional a la cantidad de aire que ésta desvía por unidad de tiempo, multiplicada por la velocidad vertical del aire que ha sido desviado. De forma similar al principio de operación de un helicóptero, uno puede aumentar la sustentación de un ala aumentando la velocidad vertical del aire, la cantidad de aire que se desvía, o una combinación de las dos.

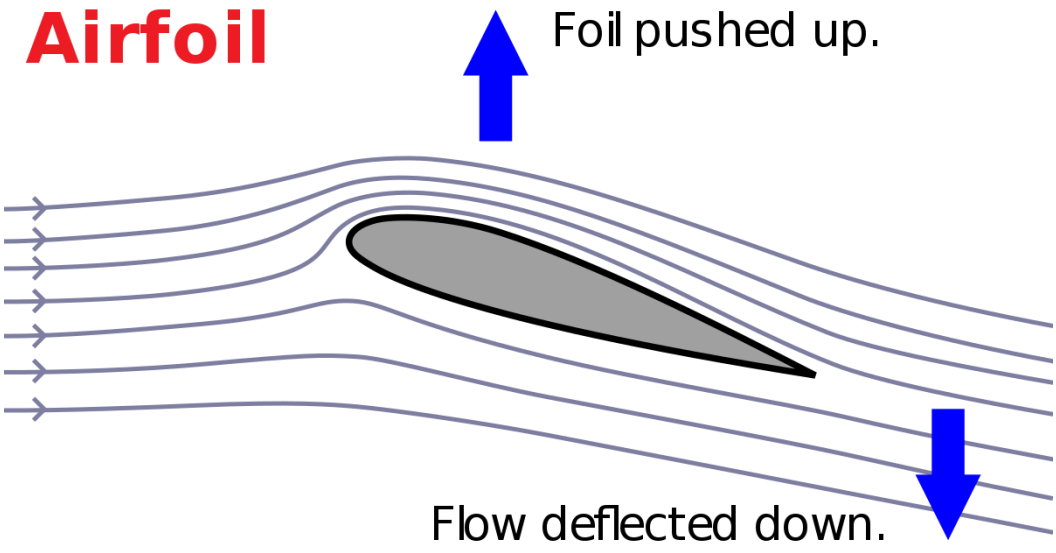


Ilustración 4: Principio del Downwash ilustrado

4.2 Perfil aerodinámico

Previo al diseño del aeromodelo es necesario escoger los perfiles aerodinámicos que se utilizarán en sus alas, timón de dirección y timón de profundidad.

4.2.1 Nomenclatura de los perfiles NACA

Existen diversas publicaciones donde se catalogan las propiedades de familias de perfiles y se facilita información sobre sus características aerodinámicas. Posiblemente los perfiles más utilizados en aeronáutica sean los perfiles NACA, cuya nomenclatura de 4 cifras se explica en la tabla siguiente, recogida de [1].

Familia	Perfiles NACA de cuatro cifras
Espesor	Es la misma ley de espesores que la de perfiles Clark Y y Göttingen 398, con un factor de afinidad que proporciona el valor del espesor relativo.
Curvatura	Dos parábolas de segundo grado tangentes en el punto de tangente horizontal
Nomenclatura	NACA XYZZ (ejemplo NACA 2415)
X	100f/c donde f es la flecha máxima de la curvatura y c la cuerda.
Y	$10x_f/c$ donde x_f es la posición de la flecha máxima de la curvatura.
ZZ	100 e/c, donde e es el espesor máximo y c la cuerda.

Ilustración 5: NACA de 4 dígitos

4.2.2 Simulación

Para los timones de profundidad y dirección se utiliza el perfil aerodinámico NACA 0012. Un perfil aerodinámico simétrico que hace de control en este aeromodelo.

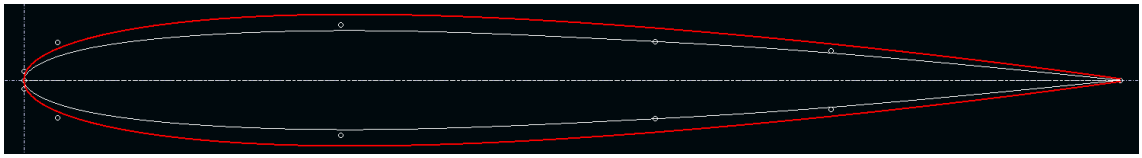


Ilustración 6: Perfil NACA 0012 introducido en XFLR5

Para las alas se ha escogido el perfil aerodinámico Eppler E214. Es un perfil aerodinámico para bajos valores de Reynolds. Para poder simular este perfil aerodinámico mediante el programa XFLR5, se ha utilizado la información disponible en: https://m-selig.ae.illinois.edu/

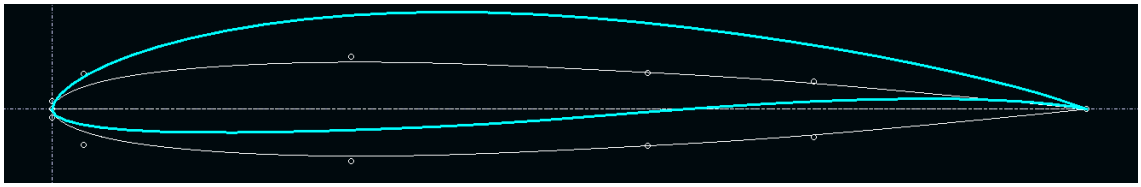


Ilustración 7: Perfil Eppler E214 introducido en XFLR5

Para continuar, se simulan ambos perfiles aerodinámicos ante diferentes números de Reynolds y ángulos de ataque. Para ello se configura un conjunto de análisis para Reynolds entre 100.000 y 150.000 a incrementos de 10.000, y para ángulos de ataque entre -2 y 16 grados con incrementos de un grado.

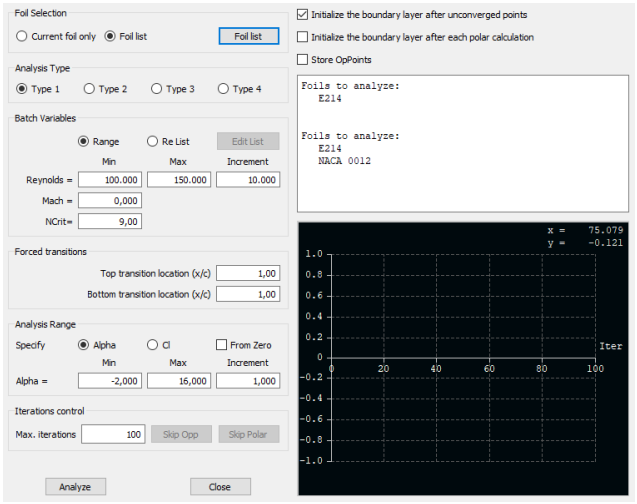


Ilustración 8: Configuración del análisis

Resultados gráficos de los coeficientes: Cd, Cl y Cm.

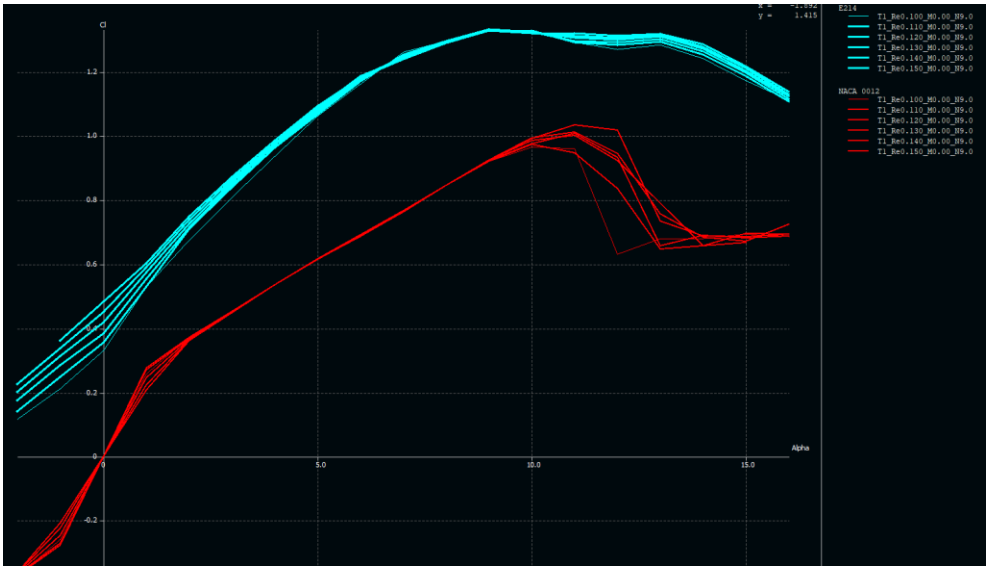


Ilustración 9: Coeficiente de sustentación en relación al ángulo de ataque

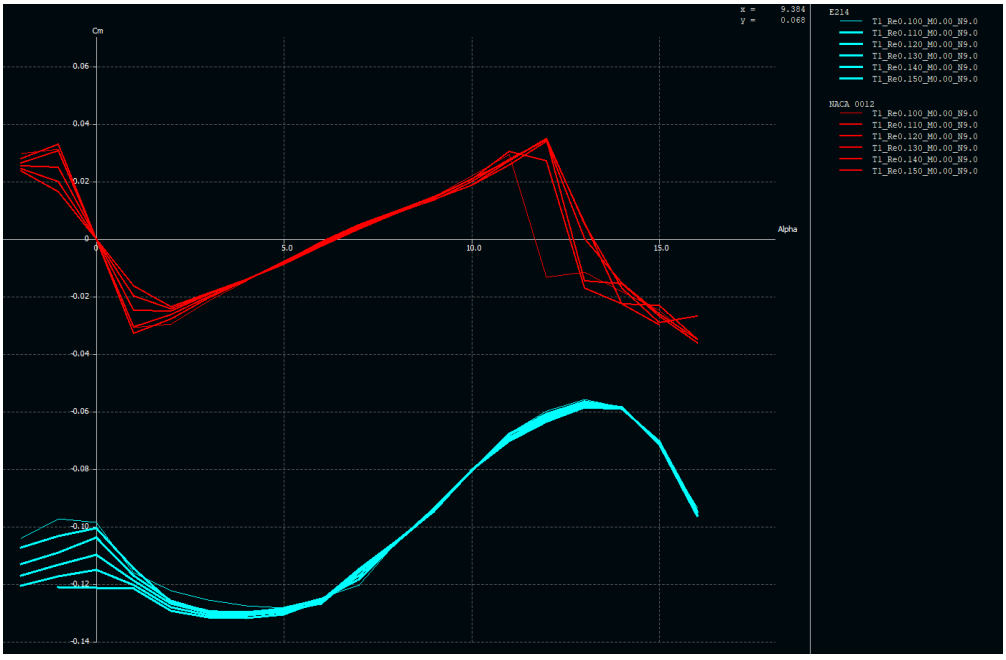


Ilustración 10: Coeficiente de movimiento en función del ángulo

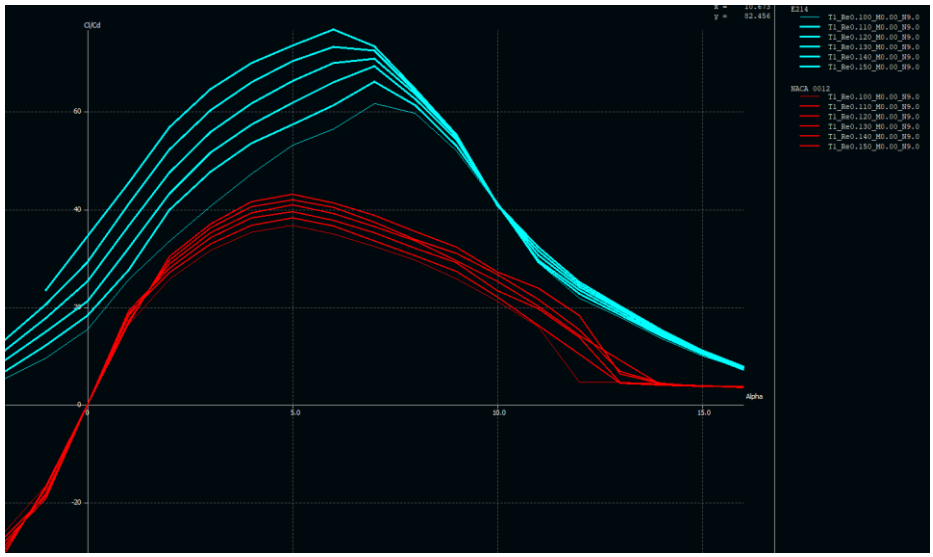


Ilustración 11: Cl (Sustentación)/Cd (Drag) en relación al ángulo de ataque

Esta gráfica muestra el coeficiente de sustentación dividido por el coeficiente de arrastre (drag), en otras palabras, la eficiencia del perfil. Al relacionarlo con el ángulo de ataque, se puede conocer la eficiencia teórica del perfil para diferentes ángulos de ataque.

Se procede a diseñar el avión.

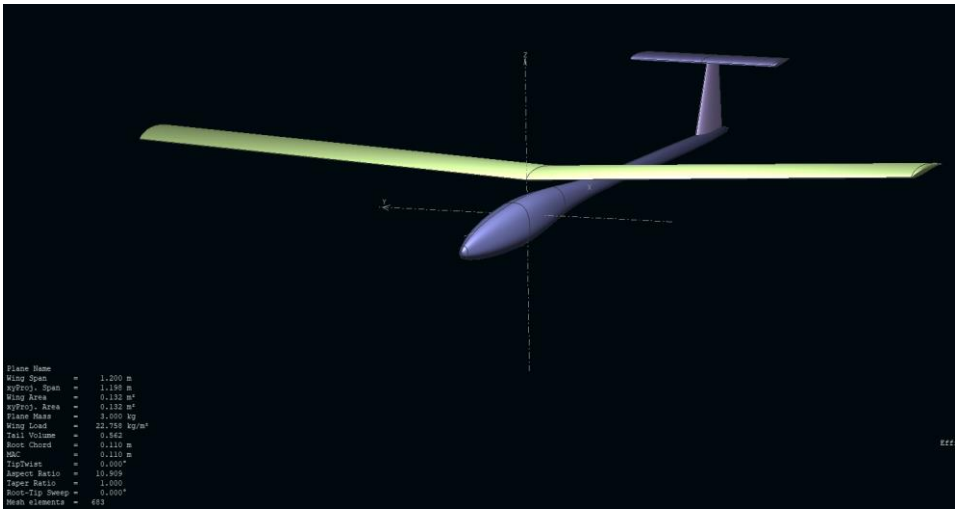


Ilustración 12: Primera iteración del aeromodelo

Se realiza una simulación viscosa a 10 m/s para analizar la estabilidad del diseño. Si se observa la gráfica Cm la pendiente es positiva, lo que hace al aeromodelo inestable. Para corregir esto, se realizan modificaciones.

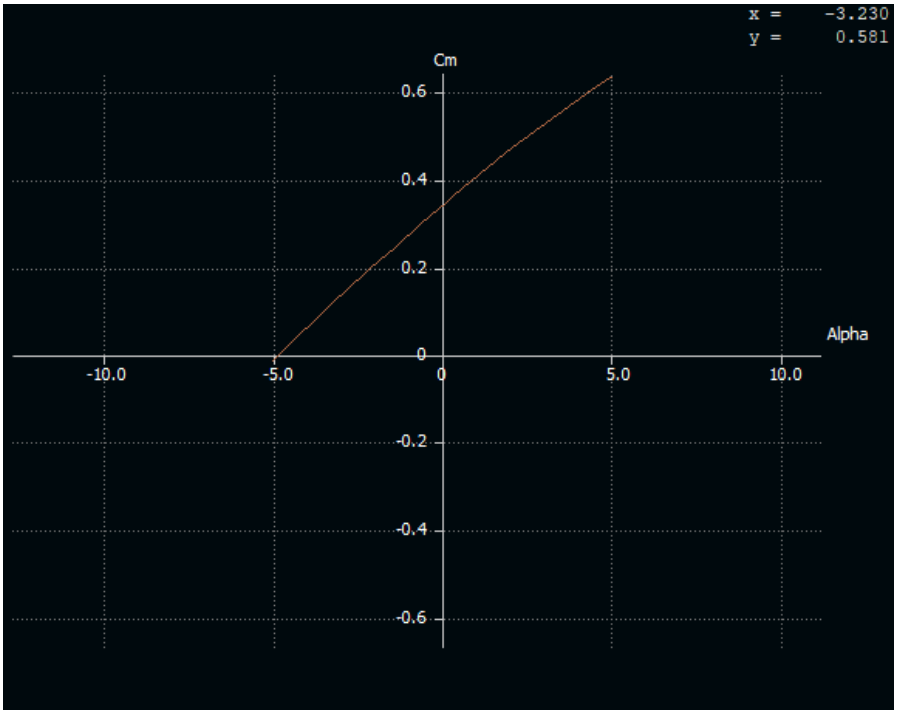


Ilustración 13: Gráfica Cm de la primera Iteración

La gráfica Cm da información acerca de la estabilidad estática del aeromodelo. Una pendiente positiva indica inestabilidad mientras que una pendiente negativa indica estabilidad (Más información acerca de la estabilidad estática en [3]).

Para que la pendiente sea negativa, se modifican la posición y ángulos de ataque de las alas y timón de profundidad. Esto consigue el siguiente resultado.

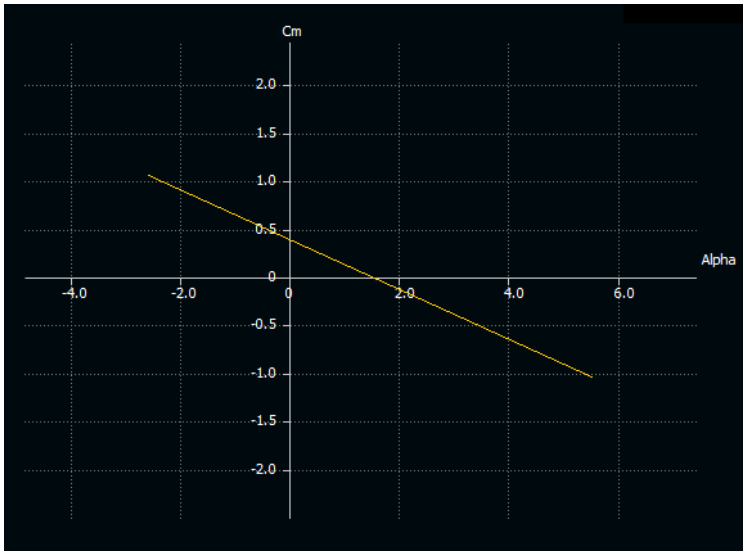


Ilustración 14: Gráfica Cm de la iteración final (Estable estáticamente)

Se utiliza fabricación aditiva para construir el aeromodelo.

Ilustración 15: Impresora Anet A6 fabricando piezas del aeromodelo

4.3 Aviónica

La electrónica aplicada a la aviación recibe el nombre de aviónica. El diseño de la electrónica de este proyecto se divide en 3 partes: primera, el diseño hardware, donde se escogen los diferentes componentes electrónicos, segunda, el diseño software (donde se desarrollan los diferentes lazos de control, filtros de orientación…) y tercera, la implementación del software que gobierna el prototipo.

Debido a la alta velocidad de transferencia de datos por los buses de comunicación, se siguen las pautas para diseño de circuitos impresos disponibles en [9], [10] y [11].

4.3.1 Hardware

Con el fin de conseguir que el proyecto funcione con un hardware universal, se ha decidido utilizar un sistema de tarjetas modulares cuya conexión siga la norma del Estándar M.2.

Ilustración 16: Tarjeta SAMD51 para MicroMod Estándar M.2 (https://www.sparkfun.com/products/16791)

Dado que el estándar M.2 fue diseñado para la conexión de esclavos modulares, es necesario adaptar la norma para que también permita la conexión de controladores modulares. Este proyecto aprovecha las especificaciones creadas para MicroMod y, por lo tanto, utiliza el pinout MicroMod M.2.

En los anexos se encuentran disponibles una serie de tablas que especifican las conexiones según esta norma.

Mediante el uso de este sistema, se divide el hardware en 4 módulos conectables y una placa base que los une.

4.3.1.1 Placa base

La función de la placa base es permitir la interconexión de los 4 módulos y aportar una alimentación de 3.3 Voltios. Para ello, debe ser capaz de regular la tensión de la batería que se le va a conectar y unir los diferentes buses de comunicación, según la normativa que se está utilizando.

Alimentación

Para alimentar el sistema se utiliza una batería de 12 V de Ni-MH. Para convertir la tensión de la batería a 3.3 V se utiliza el convertidor DC-DC "TPS562201DDCR", de Texas Instruments, como se ve en el siguiente esquema:

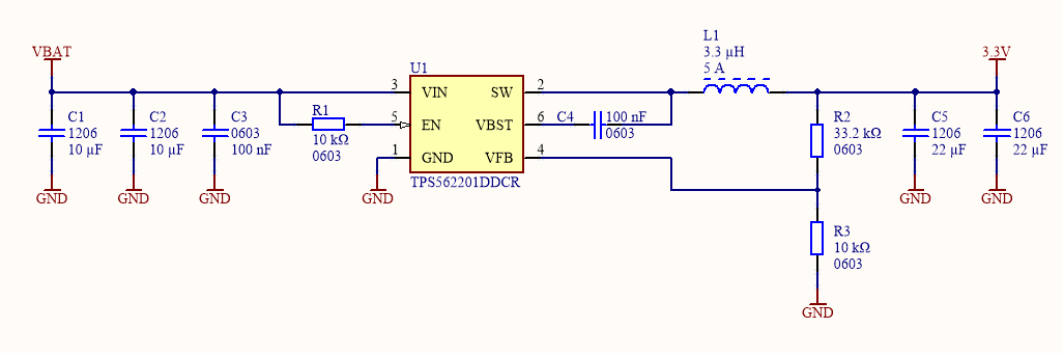


Ilustración 17: Convertidor Buck para alimentar la electrónica

Ilustración 18: Batería 12V Ni-MH

4.3.1.2 Módulo Microcontrolador

El módulo Microcontrolador incorpora un microcontrolador, los componentes pasivos necesarios para su funcionamiento y las conexiones al conector M.2 Estándar correspondientes. Cualquier microcontrolador con la memoria suficiente, velocidad de reloj que permita la ejecución del programa y sus interrupciones, y buses de comunicación I2C, SPI y UART puede utilizarse en este módulo.

ATMEGA2560

Para el prototipo se ha escogido un microcontrolador ATMEGA2560 alimentado a 3.3 Voltios. Éste es un microcontrolador de 8 bits que tiene como trabajo el cálculo de orientación y trayectoria de la aeronave a partir de los sensores disponibles, así como el mando sobre las superficies de control. En la siguiente imagen se puede observar un esquemático reducido de la alimentación para este microcontrolador.

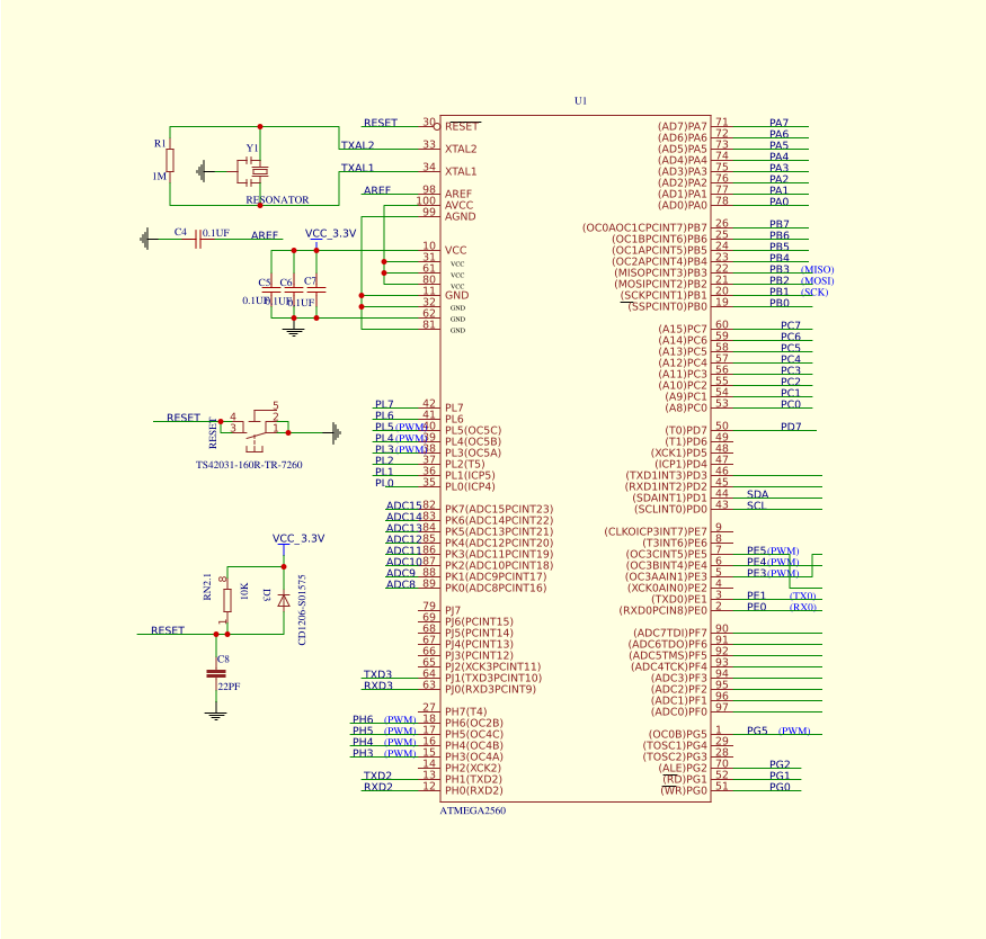


Ilustración 19: Conexiones de alimentación del ATMEGA2560

4.3.1.3 Módulo de instrumentación

El módulo de instrumentación es el módulo donde se encuentran el receptor GNSS y el MARG (IMU + Magnetómetro). Este módulo necesita recoger del conector M.2 Estándar: alimentación, bus I2C (para la comunicación entre Microcontrolador y MARG) y bus UART (para la comunicación entre Microcontrolador y receptor GNSS).

MPU-9250

El IC MPU-9250 es un conjunto IMU + magnetómetro que se vende como módulo para plataformas de desarrollo de hardware y es aproximadamente 200 veces más barato que un XSens MTi-7 (AHRS que se ha utilizado como referencia comercial durante los ensayos). Debido a su bajo coste y su fácil implementación en Arduino (ya existen librerías que lo manejan), este módulo ha sido escogido como unidad MARG de P.A.T.O.

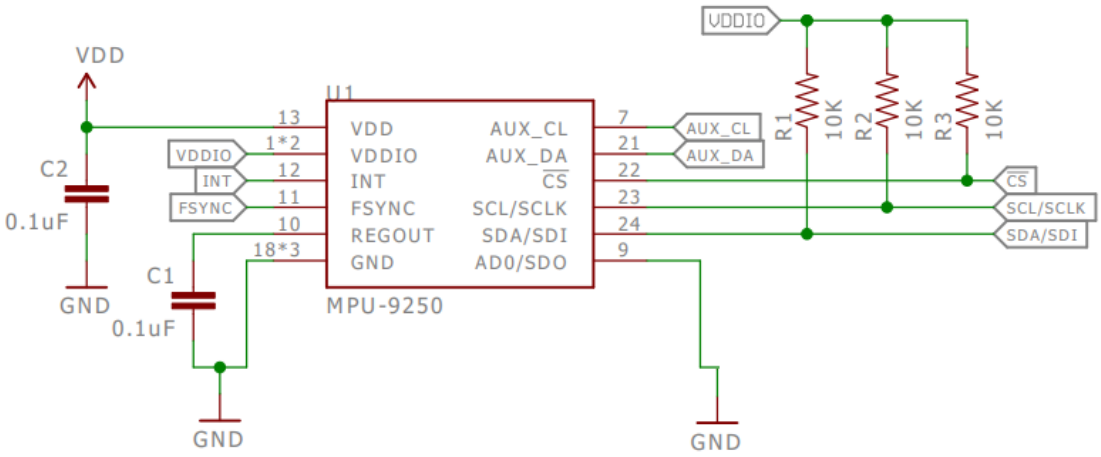


Ilustración 20: Esquema del módulo MARG MPU9250

Receptor GNSS

El receptor GNSS de la familia NEO-6 ofrece los datos a través de UART, a una velocidad de 9.600 bps. Los datos transmitidos son sentencias definidas en el protocolo NMEA (National Marine Electronics Association). La intensidad de corriente necesaria es de unos 37 mA en modo de medición continuo. Se utiliza un modelo NEO-6M, puesto que se alimentará a 3.3 V (También sería posible utilizar los modelos NEO-6Q que funcionan con la misma tensión).

4.3.1.4 Módulo de Actuadores

Los servomotores encargados de gobernar las superficies de control del aeromodelo necesitan tanto alimentación como control, procedente de la aviónica. Se conectan a este módulo.

4.3.1.5 Módulo de telemetría

El Módulo de telemetría se encarga de mantener una comunicación bidireccional con la estación de tierra.

Frecuencia

A la hora de escoger la frecuencia de operación es importante tener en cuenta los efectos de difracción, absorción y refracción. Se pueden separar en 4 grupos de propagación: ondas terrestres, directas, de satélite e ionosféricas.

Las **ondas terrestres** (**Surface Waves**): Al enviar más allá del horizonte, en teoría debería haber sombra (electromagnética), las ondas inducen otras corrientes en la tierra haciendo que éstas aprovechen el efecto de refracción sobre esta nueva zona y que lleguen muy lejos. La distancia depende del tipo de suelo, sobre el que se inducen las corrientes. Las frecuencias de las ondas terrestres se encuentran entre 30 Hz y 3 MHz.

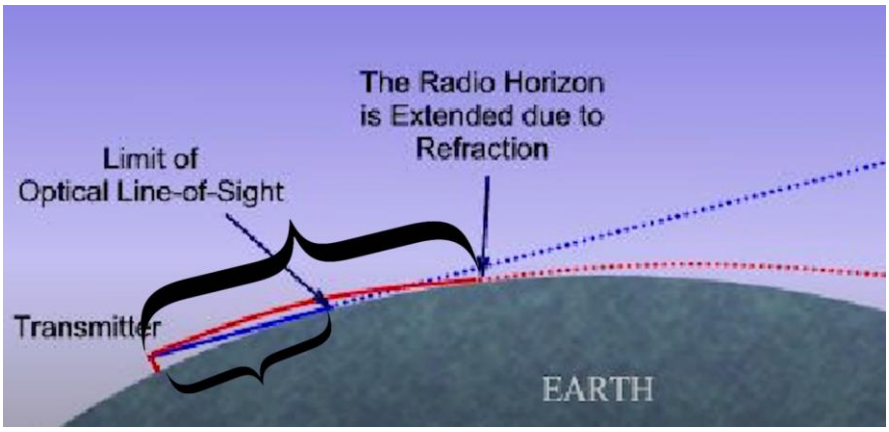


Ilustración 21: Surface Waves

Las ondas directas tienen como característica que se propagan hasta que se encuentran un obstáculo o el horizonte. Se utilizan en todo el espectro electromagnético. La distancia máxima de propagación dependerá de la altura a la que se encuentren las antenas.

$$distancia(km) = \sqrt{13 \cdot h_{transmisor}(m)} + \sqrt{13 \cdot h_{receptor}(m)}$$

Ondas ionosféricas (Sky Waves) y de satélite (Space Waves): La atmósfera tiene varias capas. Durante los 20 primeros kilómetros se encuentra la troposfera, después está la estratosfera durante otros 30 kilometros y después está la ionosfera; esta parte de la atmósfera está situada entre los 50 y 400 km y es menos densa. La ionosfera hace de capa eléctrica, al reaccionar a la radiación solar, y por lo tanto afecta a las transmisiones electromagnéticas.

Desde los *50 km* están la capa D de la ionosfera, la capa E, la capa F1 y la capa F2. De noche, al no reaccionar al sol sólo está la capa F (F1 + F2. La más externa).

- Para las ondas de UHF (*mayores de 300 MHz*) la ionosfera es transparente. Éstas son las **Ondas de satélite o Space Waves**.
- Las frecuencias muy bajas son absorbidas por la ionosfera.
- De *3 KHz a 3 MHz* las ondas rebotan en la ionosfera (reflexión). Es importante tener en cuenta que la variación de las capas entre noche y día hace que la distancia de rebote cambie, siendo mayor de noche. Cualquier onda reflejada cambiará su polarización y por lo tanto se verá atenuada.
- Las llamadas **Ondas ionosféricas**, de frecuencia entre *30 KHz y 30 MHz,* refractan en la ionosfera.

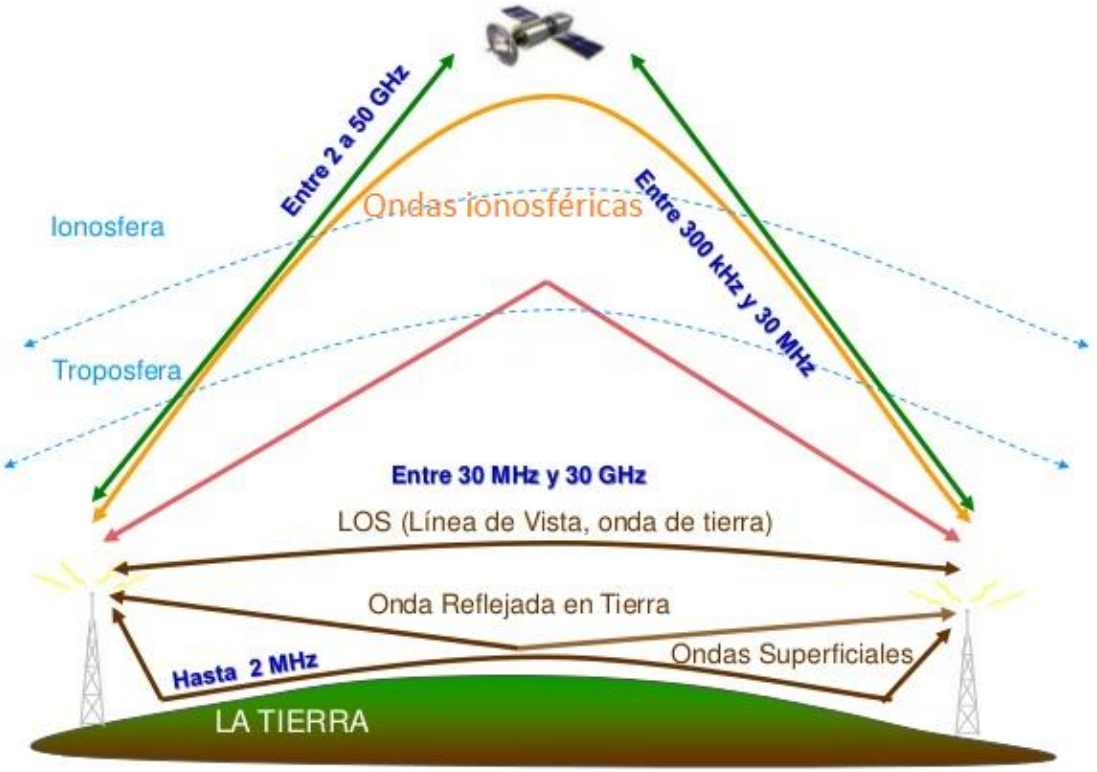


Ilustración 22: Esquema con las diferentes ondas

Dado que la aeronave debe estar siempre en Línea de Visión, según las normativas actuales, se escoge una frecuencia entre 3 MHz y 30 GHz para el funcionamiento de la telemetría.

En caso de que aparezca un obstáculo puntual (un pájaro, un globo…) entre ambas antenas el radioenlace podría verse afectado y por tanto, se decide que la frecuencia será preferiblemente baja. Esto se debe a que a frecuencias más bajas, los obstáculos en Línea de Visión pueden ser mitigados gracias al efecto de la difracción.

Modulación GFSK

Se escoge una modulación GFSK para hacer el trabajo.

Teniendo todo lo anterior en cuenta se escoge una frecuencia de trabajo 431 MHz. Para este trabajo se utiliza un APC220 cuyo datasheet se encuentra disponible en los anexos. Se utiliza otro APC220 con un adaptador de UART a USB para recibir la telemetría mediante el monitor serie de Arduino IDE.

Ilustración 23: Módulos APC220 + adaptador USB/UART para funcionar de telemetría

4.3.2 Software

El software a desarrollar se separa principalmente en 2 partes diferentes. Por un lado está el *Control Guiado* de la aeronave, que permite la aeronavegación mediante la orientación respecto a tierra del aeromodelo, y por otro lado está el *Control de Trayectoria*, que mediante un sistema GNSS es capaz de conocer su posición y trayectoria para seguir una serie de puntos GNSS previamente programados. Estos dos lazos se superponen, siendo el *Control de Trayectoria* el que da las consignas al *Control Guiado*.

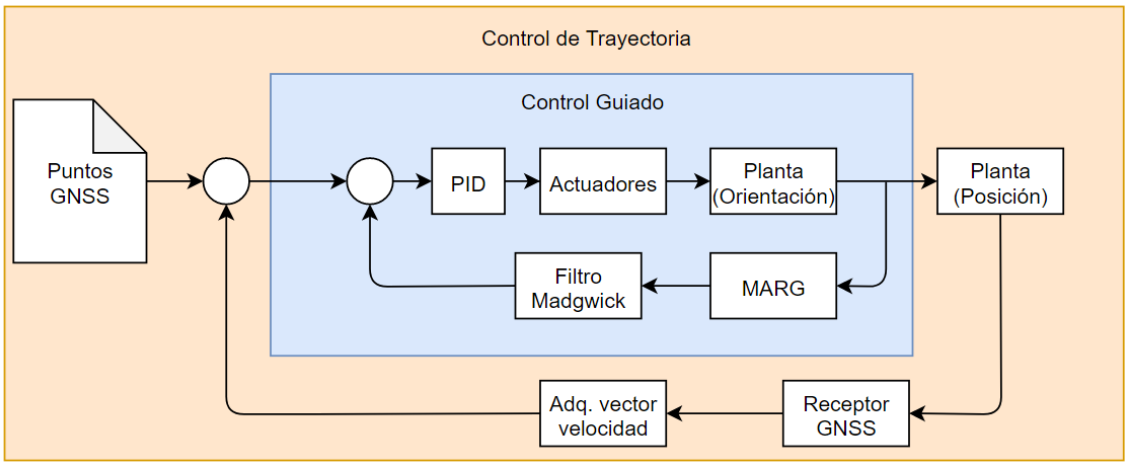


Ilustración 24: Diagrama simplificado de los lazos de control de P.A.T.O.

4.3.2.1 Control Guiado

El *Control Guiado*, es el lazo de control encargado de orientar la aeronave y por lo tanto de gobernar las superficies de control para, por medio de la instrumentación a bordo, corregir las reacciones del avión hasta obtener la orientación solicitada.

Navegación aérea para aeromodelos de 2 G.D.L.

Un aeromodelo de 2 grados de libertad utiliza el timón de dirección y el timón de profundidad para controlar su vuelo. Más detalles acerca de la aeronavegación en [2].

Para controlar el ángulo de inclinación (*pitch* en inglés, normalmente θ) utiliza el timón de profundidad. La acción de esta superficie de control disminuye o aumenta la fuerza descendente creada por la parte trasera del avión. Una mayor fuerza descendente en la cola, producida por un timón de profundidad *hacia arriba*, fuerza a la cola del avión a ir hacia abajo y a la nariz a ir hacia arriba y la velocidad se reduce. Una disminución de fuerza descendente en la cola, producida por un timón *hacia abajo*, permite que la cola se eleve y la nariz baje.

Para controlar la dirección del avión se utiliza el timón de dirección. Es importante que para que este control funcione el ángulo superior de las alas debe ser inferior a 180º. Esto es llamado ángulo de diedro y necesita ser positivo o lo que es lo mismo, un ángulo inferior a 180º por la parte de arriba. Un efecto adicional del ángulo de diedro es la contribución a la estabilidad del avión.

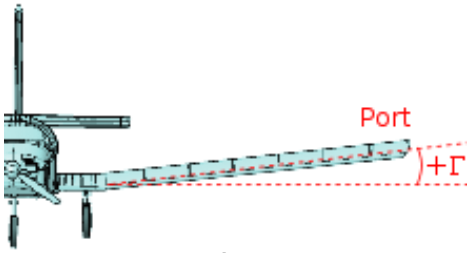


Ilustración 25: Ángulo de diedro

La acción del timón de dirección hace girar el aeromodelo en su eje de *deriva* (*yaw* en inglés, normalmente Ψ), lo que hace que exista un deslizamiento entre la orientación del avión y la dirección del mismo (y por lo tanto de la dirección de la que reciben el viento las alas). Este deslizamiento provoca (entre otras cosas) que el ángulo de ataque del ala hacia donde se gira disminuya mientras que el ángulo de ataque del ala contraria aumente. Esta modificación de ángulos de ataque hace que la sustentación del ala con más ángulo de ataque aumente provocando que el avión gire en su eje de alabeo (*roll* en inglés, normalmente Φ).

Cuando el avión se encuentra girado en su eje de alabeo y la dirección del viento coincide con la orientación del avión, el ángulo de ataque en ambas alas es el mismo, por lo que la fuerza de sustentación es la misma. Dado que el avión está girado y una de las alas no realiza su fuerza en vertical (porque está girado) el avión vuelve de forma natural a su posición de reposo (posición estable para un vuelo recto y nivelado).

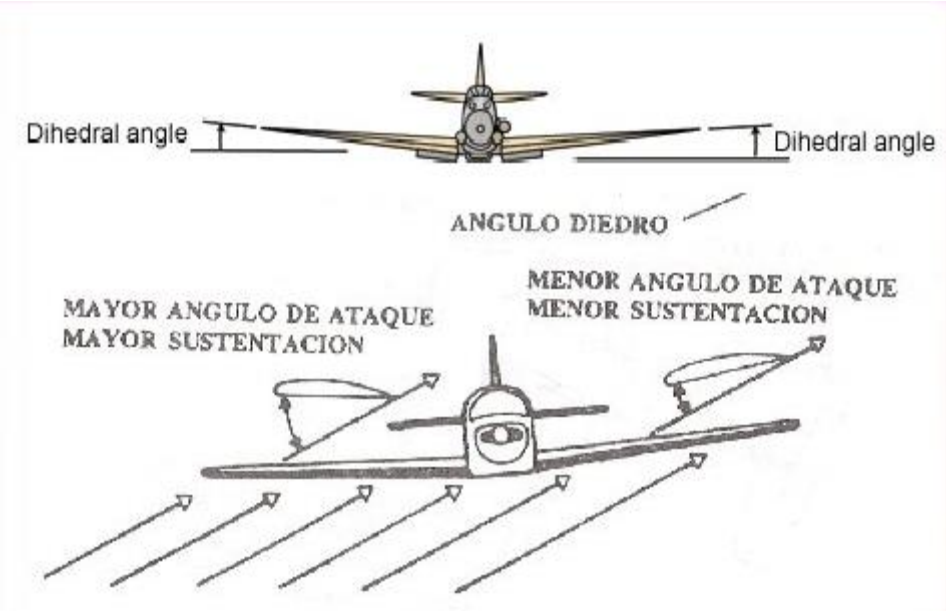


Ilustración 26: Demostración del autonivelado mediante ángulo de diedro

Lazo de control para dos grados de libertad

El lazo de control para dos grados de libertad se divide en 3 bloques principales: Control Guiado 2 G.D.L., Planta y Realimentación AHRS.

A la hora de realizar la planta del aeromodelo, se plantea un modelo matemático simplificado, donde el aeromodelo es un sólido rígido. Éste reacciona a los pares generados por una fuerza constante, sobre la superficie frontal aparente de las superficies de control. Adicionalmente estos pares son sometidos a perturbaciones que representan, no sólo perturbaciones durante el vuelo, sino las auténticas fuerzas aerodinámicas a las que el avión estará sometido también en un vuelo normal. Es posible hacer esto porque se conoce la gran estabilidad de este aeromodelo, lo que permite que cualquier irregularidad (como es el modelo excesivamente simplificado que se propone aquí) se corrija automáticamente.

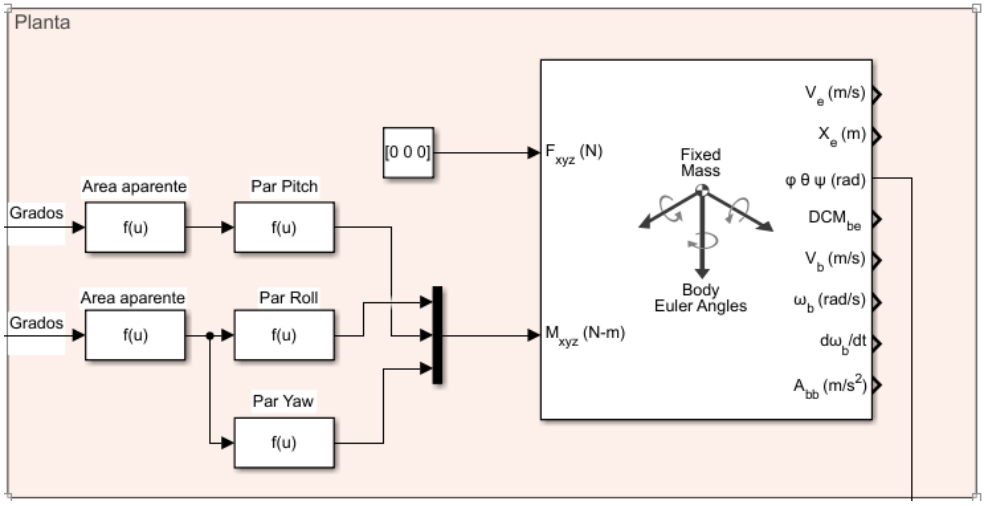


Ilustración 27: Planta simplificada de aeromodelo de 2 G.D.L.

Las superficies de control se mueven gracias a servomotores; éstos se representan como lazos cerrados (junto con una saturación como limitación mecánica de posición) tal y como se ve en la siguiente ilustración:

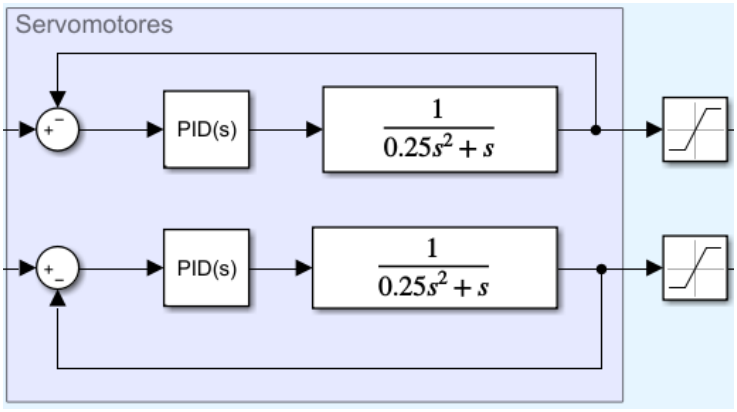


Ilustración 28: Servomotores con limitación mecánica de posición

Los servomotores se encuentran dentro del bloque “Control guiado 2 G.D.L.”. Este bloque recibe consignas de giro e inclinación que posteriormente se adaptan. El error entre estas consignas y el ángulo Euler correspondiente llega a un PID que gobierna los servomotores. Cabe destacar que el valor de la consigna una vez adaptado no puede superar valores de seguridad asignados heurísticamente con el fin de que el aeromodelo trabaje con patrones de vuelo ya conocidos y que, por lo tanto, no entre en pérdida.

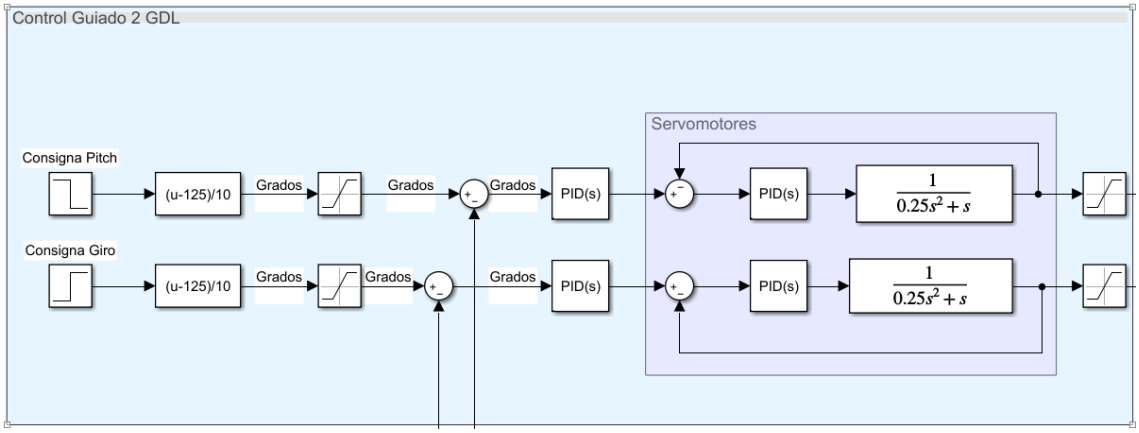


Ilustración 29: Bloque "Control Guiado 2 G.D.L."

El control sigue los principios de navegación aérea para aeronaves de 2 G.D.L. anteriormente comentados. Para ello, el ascenso será establecido por la inclinación del avión y por lo tanto el error entre consigna de elevación adaptada y la inclinación será el que gobierne el timón de profundidad. Por otro lado el giro será establecido por el Roll del avión y será el error entre éste y la consigna quien determine la posición del timón de dirección.

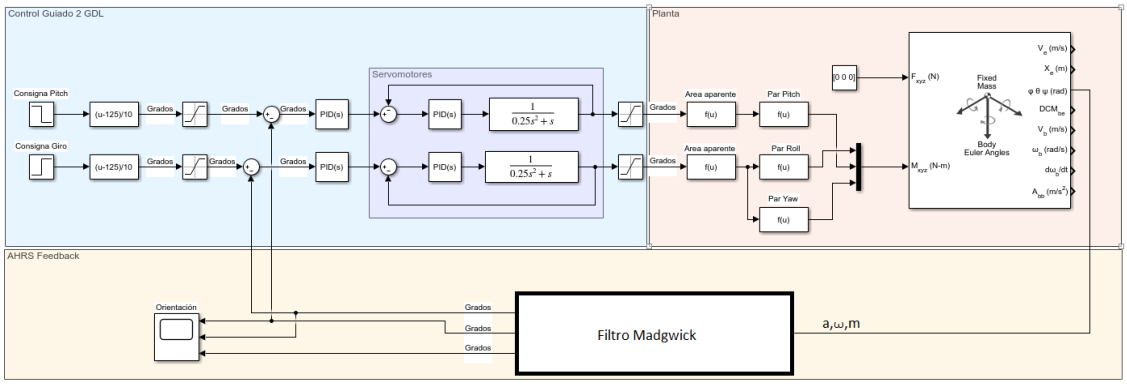


Ilustración 30: Lazo de control 2 G.D.L. simplificado en Simulink

Navegación aérea para aeromodelos de 3 G.D.L.

Un aeromodelo de 3 grados de libertad utiliza timón de profundidad, timón de dirección y alerones para controlar su vuelo. Más detalles acerca de la aeronavegación en [2].

Para controlar el ángulo de inclinación utiliza el timón de profundidad. La acción de esta superficie de control disminuye o aumenta la fuerza descendente creada por la parte trasera del avión, igual que en el aeromodelo de 2 G.D.L.

En aerodinámica, el giro es una maniobra básica muy compleja e implica el uso de timón de dirección, timón de profundidad y alerones. Durante el vuelo recto y nivelado, la sustentación total actúa en vertical y directamente opuesta a la gravedad. Al alabear el avión, la sustentación (que sigue siendo perpendicular al eje transversal del aeromodelo) actúa ahora en un plano inclinado.

Si se descompone esta sustentación en dos vectores (uno vertical y otro horizontal), el componente vertical de la sustentación se opone a la gravedad. El componente horizontal actúa como fuerza centrípeta, tirando del avión hacia el centro de un eje imaginario. Esta fuerza centrípeta lo impulsa a girar alrededor de dicho eje, contribuyendo la sección de cola a mantener el aeromodelo alineado con el viento relativo en una trayectoria curvada.

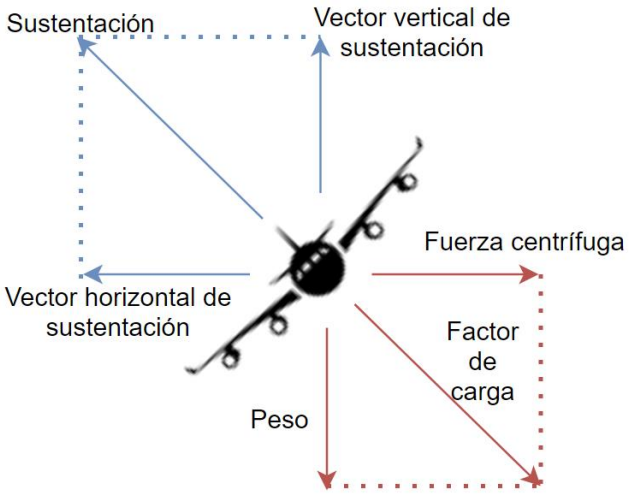


Ilustración 31: Fuerzas que actúan en un giro

Por lo tanto, se alabea el avión para inclinar la sustentación. Esta inclinación de la sustentación hace que además de soportar el peso del avión, se provea de fuerza centrípeta, que mantiene al avión alrededor del eje vertical de giro, contrarrestando la fuerza centrífuga que tiende a expulsar al avión de la trayectoria curvada.

Maniobra de giro

Para iniciar el giro se comienza a girar gradualmente hacia el lado al cual se quiere girar el avión, hasta conseguir una actitud de alabeo apropiada para el giro.

Al alabear el avión cambia la trayectoria de vuelo, por lo que lo hacen también los ángulos de ataque de las dos alas. Como el ala del lado contrario al giro tiene menor ángulo de ataque que el ala del lado del giro, desarrolla menos sustentación.

Puesto que se quiere que el viraje sea coordinado (ni resbale ni derrape), el eje longitudinal debe apuntar en esa trayectoria. Para ello, el timón de dirección anulará la aceleración que aparece en el eje formado por las alas.

Puesto que el morro tiende a caer, es necesaria la acción del timón de profundidad durante la maniobra.

Lazo de control para 3 G.D.L.

Nuevamente, el lazo de control se puede dividir en los mismos 3 bloques, pero tanto la planta como el bloque "Control Guiado 3 GDL" son ahora distintos.

La planta ahora tiene 3 superficies de control diferentes que ejercen un par de fuerzas cada una. Esto hace necesario un sistema de coordinación de giro, tal y como se explicó en el apartado anterior.

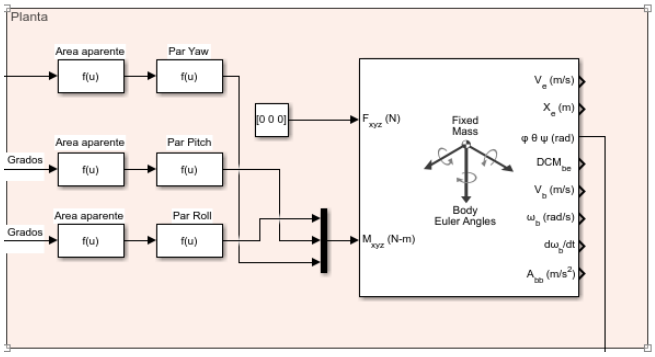


Ilustración 32: Planta con modelo 3 GDL simplificado

Puesto que ahora hay 3 superficies de control se añade un servomotor nuevo.

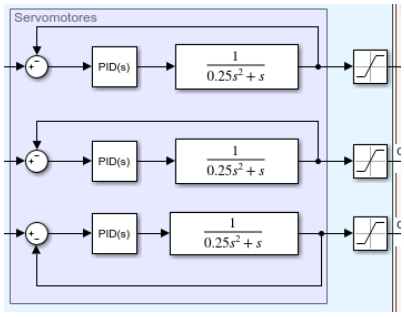


Ilustración 33: Los 3 Servomotores que controlan las superficies de control

Finalmente, el bloque *Control Guiado 3GDL* no tiene 3 consignas. Mantiene el sistema de 2 consignas anteriormente empleado para el sistema de dos grados de libertad, lo que hace que el sistema se pueda utilizar independientemente de la opción que se escoja.

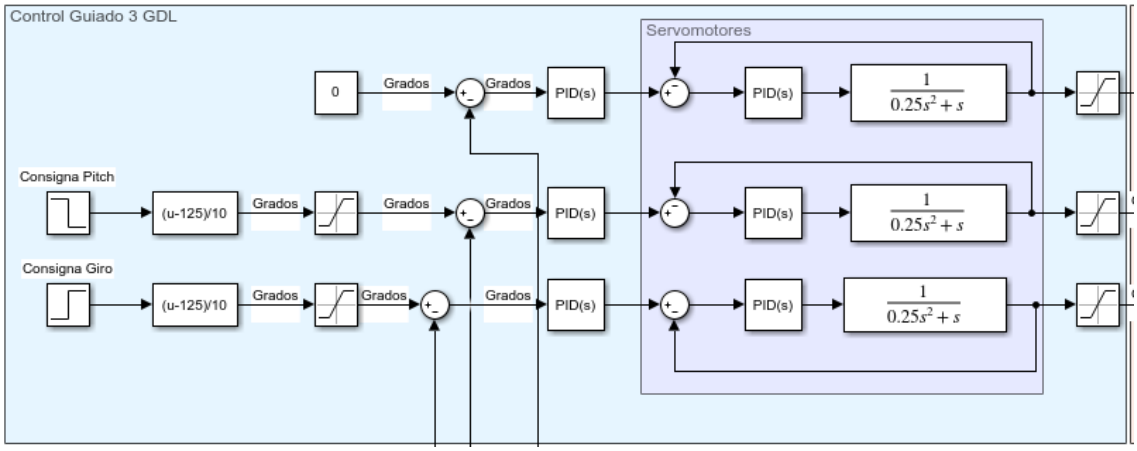


Ilustración 34: Bloque "Control Guiado 3 GDL"

El lazo de control sigue el procedimiento de aeronavegación para aeronaves de 3 GDL explicado anteriormente. El control de ascenso/descenso permanece inalterado, el control de giro por el contrario ha cambiado.

Para controlar el giro es necesario mantener una coordinación entre las 3 superficies de control, lo que se llama coordinación de giro. Para conseguirlo se ha optado por separar los 3 controles y darle el gobierno de giro al Roll. Por lo tanto, el giro se realiza de la siguiente forma: el error entre el Roll y la consigna de giro controla los alerones. Esto hace que el avión se ladee. Al igual que en el sistema de 2 GDL, el control de ascenso/descenso mantiene la coordinación del timón de profundidad de forma automática. El timón de dirección es gobernado por el error en la aceleración lineal en el eje Y del aeromodelo (es decir, mantiene la aceleración en este eje con un valor nulo), dado que tanto durante un vuelo recto y nivelado como durante un giro el valor de la aceleración debe ser cero. Todo esto consigue de forma eficaz la coordinación de giro con 3 G.D.L.

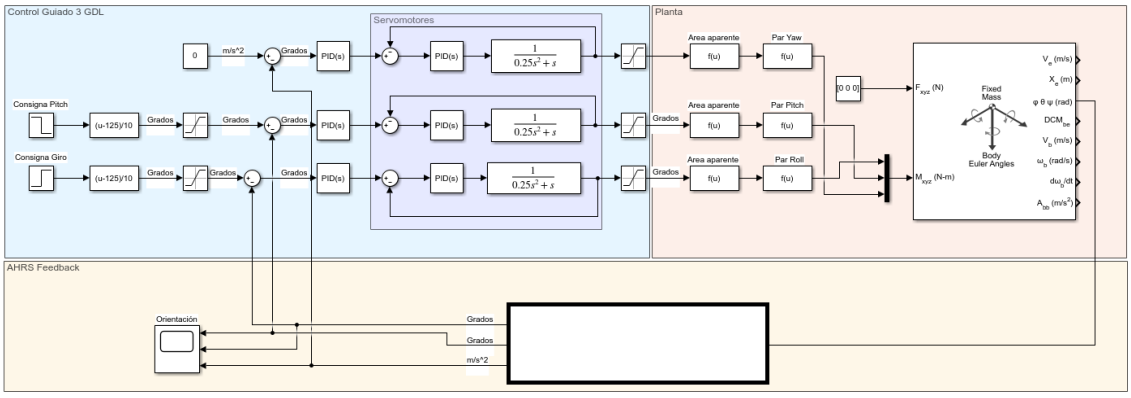


Ilustración 35: Lazo de control simplificado para 3 G.D.L. en Simulink

4.3.2.3 Control de Trayectoria

Para realizar el control de trayectoria se supone la diferencia entre la trayectoria del aeromodelo y su orientación despreciable. Por lo tanto, se puede suponer que, tras normalizarlos, tanto el vector de trayectoria como el de orientación son iguales. Para obtener el vector de trayectoria se utiliza un sistema GNSS, en este caso se utiliza el módulo GY_GPS6MV2.

G.N.S.S.

Existen varias redes GNSS actualmente en funcionamiento: GPS (Estados Unidos), Galileo (Unión Europea), GLONASS (Rusia), BeiDou (China)…

El funcionamiento de los sistemas de navegación por satélite se basa en la medida de las distancias existentes entre el receptor cuya posición se quiere determinar y un conjunto de satélites cuya posición se conoce con gran precisión. Este proceso es muy usado en el entorno topográfico y se conoce como Trilateración inversa 3D.

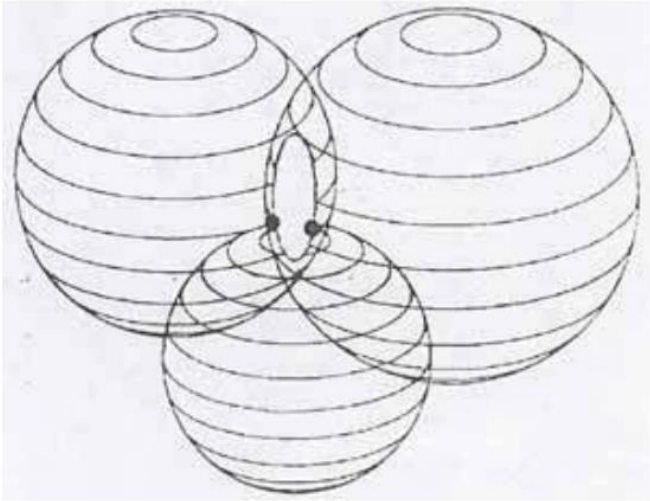


Ilustración 36: Localización con tres satélites, triangulo esférico de incertidumbre.

Matemáticamente se necesitan 4 mediciones de distancia a los satélites para determinar la posición exacta, pues se debe añadir la incógnita del estado o retardo de reloj del receptor.

En tierra, el módulo receptor GNSS tiene un almacenaje programado que le informa dónde está cada satélite en el espacio, en cada momento. Las órbitas básicas son muy exactas pero con el fin de mantenerlas así, los satélites son monitorizados de manera constante. Los errores que se controlan son los llamados errores de efemérides, es decir, los producidos por la evolución orbital de los satélites. Estos errores se generan por influencias gravitacionales del sol y de la luna y por la presión de la radiación solar sobre los satélites. Son errores generalmente muy sutiles pero si se quiere gran exactitud se deben tener en cuenta. Estos errores son enviados al receptor por el satélite en el código, con información sobre la órbita exacta del satélite.

Puntos de viaje GNSS

El sistema tiene almacenados en memoria una serie de puntos GNSS en un orden preestablecido. Esto determina el patrón de vuelo del aeromodelo, que seguirá estos puntos GNSS sin importar lo que tenga delante, por lo que será necesario un análisis topográfico previo de la zona con el fin de evitar accidentes.

Consigna Ascenso/Descenso para Control Guiado

El control de trayectoria será el encargado de dar la consigna de ascenso/descenso al lazo de control de Control guiado de la aeronave.

Para entregar la consigna, se tomará la altura del próximo punto GNSS y se le restará la altura GNSS actual. El resultado será la consigna, que posteriormente el propio Control Guiado adaptará.

$$Consigna_{inclinación} = h_{próximo\ punto} - h_{actual}$$

Consigna de Giro para Control Guiado

Para calcular la consigna de Giro a entregar al Control Guiado, es necesario obtener dos vectores: el vector de trayectoria actual y el vector a destino.

El vector de trayectoria actual lo proporciona el módulo GNSS. Por otro lado, el vector a destino es el vector obtenido entre el punto actual y el destino posteriormente normalizado. Estos vectores son bidimensionales y la altura no es tenida en cuenta. Para calcular el ángulo entre estos dos vectores se recurre a la siguiente fórmula:

$$\cos(\alpha) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}| \cdot |\vec{v}|}$$

Para determinar la dirección de giro se hace lo siguiente: Si el ángulo es inferior a 180º, se entrega el valor del ángulo como consigna. En caso contrario se restan 360º al valor y este nuevo valor se entrega como consigna. Posteriormente el Control Guiado adaptará este valor para trabajar con él.

Una vez alcanzada una distancia al objetivo (variable a la hora de configurarlo), se continúa al siguiente punto de la lista. Para conseguir esto se plantea un cubo, orientado según el sistema de coordenadas GNSS, cuyo centro es el punto GNSS y los lados se encuentran a la distancia configurada del centro.

Ilustración 37: El área que se reconoce como punto es en realidad un cubo tridimensional

4.3.3 Rotaciones, ángulos de Euler, cuaterniones y Gimbal Lock

El modo "tradicional" de especificar una rotación es descomponiéndola en tres rotaciones, cada una alrededor de un eje cartesiano.

A cada uno de los ángulos que representan las rotaciones alrededor de los ejes X,Y,Z, se le llaman Ψ, θ, Φ. Esta tripleta se denomina *ángulos de Euler*, equivalente a la notación pitch, yaw, roll. Estos ángulos se utilizan con mucha frecuencia para representar rotaciones con vectores tridimensionales. Cada tripleta de vectores ortonormales, definiendo los ejes en un movimiento determinado, se denomina marco, F_i, y una sucesión de movimientos – representados como giros mediante ángulos de Euler en las tripletas – se obtiene componiendo distintos marcos. Pese a esto, es preferible utilizar *cuaterniones* a *ángulos de Euler* para determinar la orientación.

Utilizar *cuaterniones* en lugar de los *ángulos de Euler* está más que justificado. Los *ángulos de Euler* dependen de la elección de los ejes que van a definir la rotación. Su mayor desventaja como herramienta para representar las rotaciones es su propensión a sufrir lo que se denomina *Gimbal Lock*.

Como ejemplo práctico puede mencionarse lo que ocurrió con el viaje del Apollo 11 a la Luna (https://www.hq.nasa.gov/alsj/gimbals.html), que al entrar en *Gimbal Lock* perdió toda estabilidad y empezó a dar bandazos sin sentido, hasta que mediante el control manual pudo salir de dicho estado. No es difícil detectar que cuando un *ángulo de Euler* se aproxime a $\frac{\pi}{2}$, su coseno lo hará a cero, y esto derivará en graves problemas si los cómputos los está llevando a cabo un ordenador o microcontrolador, incapaz de dividir entre 0.

Cuando se pierde uno de los grados de libertad por entrar en *Gimbal Lock*, los ángulos que definen la rotación respecto a los otros dos ejes pierden su independencia y desemboca en un caos del sistema sobre el que están aplicando las rotaciones.

Los *cuaterniones* mantienen siempre las derivadas parciales linealmente independientes, ρ_x, ρ_y, ρ_z , por lo que se evita perder un grado de libertad y caer en *Gimbal Lock.*

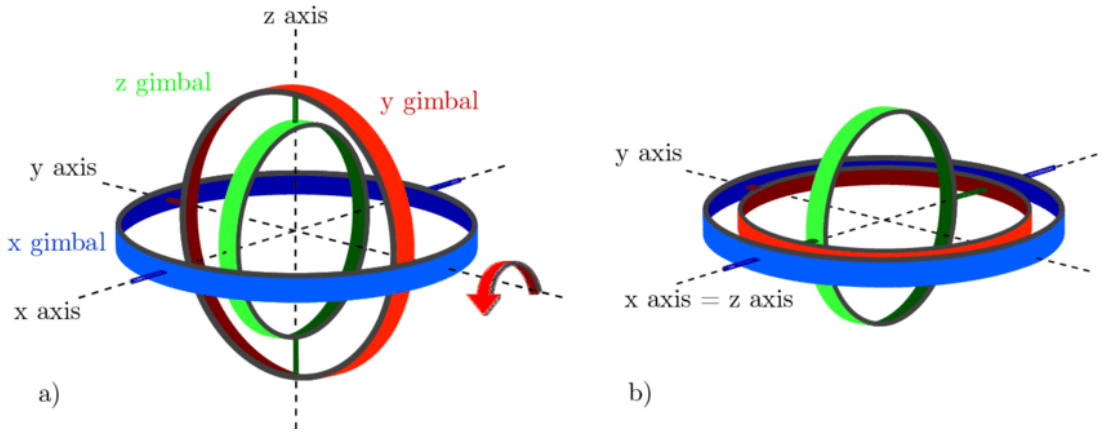


Ilustración 38: A la izquierda estado inicial con los 3 ejes perpendiculares. A la derecha entrada en Gimbal Lock del eje y (rojo) tras hacer un giro de 90º

4.3.4 Filtro de Orientación Madgwick

Basado principalmente en álgebra de *cuaterniones* y en el artículo [1], se desarrolla un algoritmo capaz de calcular la orientación utilizando acelerómetros, giróscopos y magnetómetros. La explicación del funcionamiento del filtro se encuentra en los anexos. El diagrama de bloques del algoritmo se puede ver a continuación.

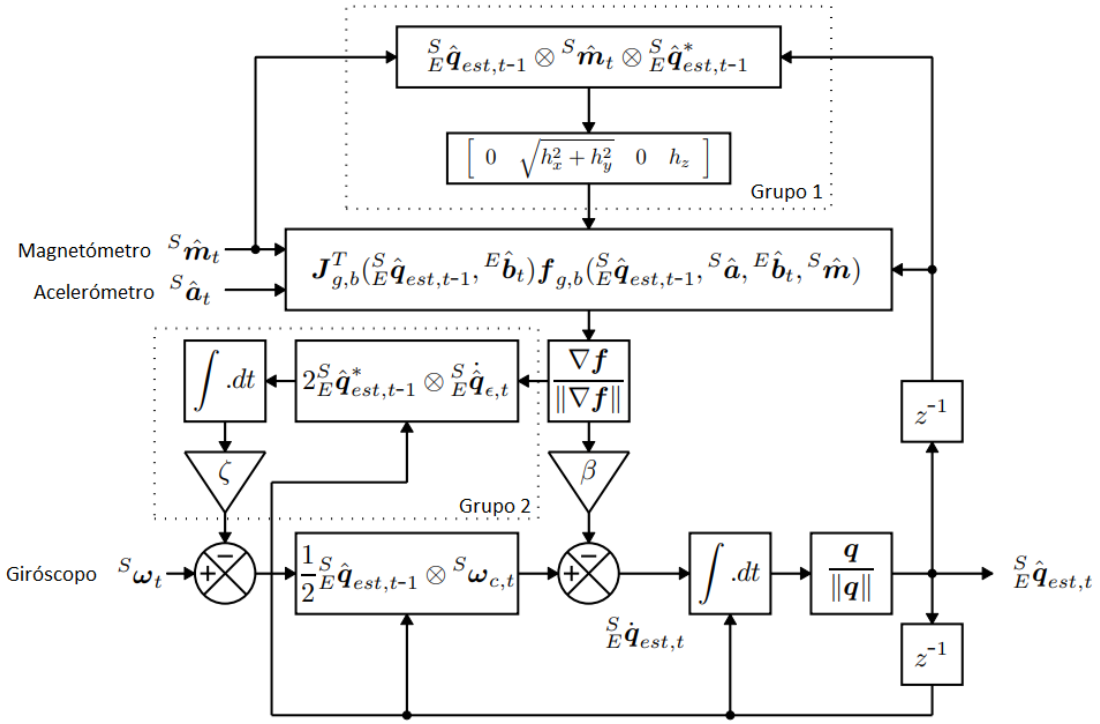


Ilustración 39: Diagrama de bloques del filtro de orientación MARG, incluyendo compensación por distorsión magnética (grupo 1) y compensación de la deriva giroscópica (grupo 2)

4.3.5 Programación del microcontrolador

A la hora de programar el Microcontrolador es importante tener en cuenta que hay parte del programa que se ejecuta normalmente y otra (gran) parte del programa se ejecuta de forma cíclica y temporizada. Por lo tanto se empieza por el programa principal. Antes de eso se añaden una serie de librerías:

```
#include <Wire.h>               //Librería para trabajar con el bus I2C
#include <math.h>               //Librería necesaria para el filtro de orientación
#include <SPI.h>                //Librería para trabajar con el bus SPI
#include <TimerOne.h>           //Librería para trabajar con interrupciones cíclicas
```

Configuración

Los datos obtenidos del receptor GNSS NEO-6 están en formato **$GPGSV, $GPVGT, $GPGLL, $GPGSA, $GPRMC y $GPGGA**, que son algunas de las secuencias disponibles en el protocolo NMEA. Estas secuencias tienen esta forma:

*$GPGSV,NoMsg,MsgNo,NoSv,{,sv,elv,az,cno}*cs*

*$GPVTG,cogt,T,cogm,M,sog,N,kph,K,mode*cs*

*$GPGLL,Latitude,N,Longitude,E,hhmmss.ss,Valid,Mode*cs*

*$GPGSA,Smode,FS{,sv},PDOP,HDOP,VDOP*cs*

*$GPRMC,hhmmss.ss,A,llll.ll,a,yyyy.yy,a,x.x,x.x,ddmmyy,x.x,a*cs*

*$GPGGA,hhmmss.ss,Latitude,N,Longitude,E,FS,NoSV,HDOP,msl,m,Altref,m,DiffAge,DiffStation*cs*

Antes de comenzar a obtener los datos, conviene configurar el módulo GNSS para que transmita las tramas que convienen al proyecto. En este caso se escogen dos: **$GPRMC y $GPGGA.**

```
//Configuración del GNSS

GPS.begin(9600)
while (GPS.available())
    GPS.read();
// Se inhabilita la trama GPGLL
strcpy(linea,"$PUBX,40,GLL,0,0,0,0,0,0*");
CheckSum();
GPS.println(linea);
// Se inhabilita la trama GPGSA
linea[10] = 'S';
linea[11] = 'A';
linea[25] = 0;
CheckSum();
GPS.println(linea);
// Se inhabilita la trama GPGSV
linea[11] = 'V';
CheckSum();
GPS.println(linea);
// Se inhabilita la trama GPVGT
linea[9] = 'V';
linea[10] = 'G';
linea[11] = 'T';
linea[25] = 0;
CheckSum();
GPS.println(linea);
// Sólo quedan habilitadas las secuencias $GPGGA y $GPRMC.
// Selección del puerto UART y su configuración
linea[0] = 0xB5; linea[1] = 0x62;               // Cabecera UBX
linea[2] = 0x06; linea[3] = 0x00;               // Configuración de Puerto
linea[4] = 0x14; linea[5] = 0x00;               // Longitud (Set/Get)
linea[6] = 0x01;                                // Puerto UART 1
linea[7] = 0x00;                                // Reservado
linea[8] = 0x00; linea[9] = 0x00;               // Tx Ready
linea[10] = 0xD0; linea[11] = 0x08;             // Modo UART (1)
linea[12] = 0x00; linea[13] = 0x00;             // Modo UART (2)
linea[14] = 0x00; linea[15] = 0x96;             // Baudrate 38.400 bps (1)
linea[16] = 0x00; linea[17] = 0x00;             // Baudrate 38.400 bps (2)
linea[18] = 0x03; linea[19] = 0x00;             // inProtoMask
linea[20] = 0x03; linea[21] = 0x00;             // outProtoMask
linea[22] = 0x00; linea[23] = 0x00;             // Reservado
linea[24] = 0x00; linea[25] = 0x00;             // Reservado
linea[26] = 0x8F; linea[27] = 0x70;             // CheckSum
for (i = 0; i < 28; i++)                        // Transmite
    GPS.write(linea[i]);                        // Los 28 bytes
GPS.end();                                      // Cierra el Puerto GPS
GPS.begin(38400);                               // Abre el puerto GPS a 38.400 bps
```

Para configurar el MPU9250 se puede utilizar la librería o mediante registros. Para ahorrar memoria del Microcontrolador se utilizan los registros directamente:

```
//Se reinicia el módulo para asegurar un estado conocido:
Wire.begin();                                   // Inicio del I2C
Wire.beginTransmission(MPU);                    // Conexión al MPU
Wire.write(0x6B);                               // Escribe registro Power Management 1
Wire.write(0);                                  // RESET
Wire.endTransmission(TRUE);                     // Fin de transmisión
```

Se empieza por la configuración del giróscopo. El fondo de escala se puede configurar a 250 dps, 500 dps, 1000 dps o 2000 dps, escribiendo en su registro el valor 0x00, 0x08, 0x10 o 0x18 respectivamente.

```
// Configuración del Giróscopo:
Wire.beginTransmission(MPU);                    // Conexión al MPU
Wire.write(0x1B);                               // Registro Config. Giroscopo
Wire.write(0x08);                               // Escribe fondo de escala 500 dps para giros.
Wire.endTransmission(TRUE);                     // Fin de la transmisión
```

Después se continúa con la configuración del acelerómetro. El fondo de escala del acelerómetro se puede ajustar a 2 G, 4 G, 8 G o 16 G, escribiendo en su registro 0x00, 0x08, 0x10 ó 0x18 respectivamente.

```
//Configuración del acelerómetro:
Wire.beginTransmission(MPU);                    // Conexión al MPU
Wire.write(0x1C);                               // Registro Config. Acelerometro
Wire.write(0x08);                               // Escribe fondo de escala a 4 Gs
Wire.endTransmission(TRUE);                     // Fin de la transmisión
```

Por último, el magnetómetro no es necesario configurarlo, pero es necesario acceder a él. Es necesario que se escriba el bit “BYPASS_EN” para poder tener acceso a él. El magnetómetro es un segundo chip dentro del encapsulado del módulo IMU y por lo tanto hay que avisar a éste, que es el dueño del I2C, de que se quiere usar ese dispositivo:

```
//Acceso al chip interno magnetómetro:
Wire.beginTransmission(MPU);                    // Conexión al MPU
Wire.write(0x37);                               // Registro INT Pin/Bypass
Wire.write(2);                                  // Activar el bypass (Acceso a AK8963)
Wire.endTransmission(TRUE);                     // Fin de la transmisión
```

Se configura la interrupción cíclica al periodo de muestreo escogido en las simulaciones con Simulink utilizando la librería TimerOne.h:

```
Timer1.attachinterrupt(ejecucion_ciclica);
Timer1.initialize(PERIODO_MUESTREO_US); //Valor en us
```

Con esto se termina el setup y se comienza con el loop del programa:

Para interpretar la secuencia se puede programar un parser que separe los valores que sean de interés. Para ello se presenta un código que aprovecha las particularidades de ASCIIZ para hacer más fácil el trabajo. Para esto se buscan los separadores (comas) y se sustituyen por el carácter de control *null* (0x00), a la vez que se escribe en cada posición del array de punteros "*dato*" la dirección de inicio de cada uno de ellos, hasta encontrar el asterisco, que indica la llegada del *checksum*. Con esto cada puntero apunta a un dato de la sentencia leída.

```
i = 5;
j = 0;
dato[j ++] = &(linea[3])];
while (linea[i]) != '*') {
    if (linea[i] == ',') {
        dato[j++] = &(linea[i + 1]);
        linea[i] = 0;
    }
    i++;
    if (i > 89)
        break;
}
```

Ahora se comprueba si la trama es $GPRMC o &GPGGA. En caso de ser &GPRMC se obtiene el valor de la dirección del vector de trayectoria. Si la trama es &GPGGA, se recoge la posición actual (latitud, longitud y altura). Todo se guarda en un array de nombre "*registro*".

```
if (j > 8) {
    if (!strcmp(dato[0], 'GGA')) {
        // Sentencia GGA ************************************************************
        strcpy(registro[HORA], dato[1]);
        if (dato[6][0] != '0') {
            strcpy (registro[LATITUD_N], dato[2]);
            strcpy (registro[LATITUD_A], dato[3]);
            strcpy (registro[LONGITUD_N], dato[4]);
            strcpy (registro[LONGITUD_A], dato[5]);
            strcpy (registro[ALTITUD], dato[9]);
        }
    }
    else if (!strcmp(dato[0], 'RMC')) {
        //Sentencia RMC  ************************************************************
        strcpy(registro[HORA], dato[1]);
        if (dato[6][0] != '0') {
            strcpy (registro[CURSO], dato[7]);
        }
    }
}
```

Ahora, para evitar que la interrupción coja los datos a medias se utiliza un flag y un array llamado *registro_copia* en el que volcar los datos. De este modo se evita que cuando se esté escribiendo en uno de ellos, los datos recogidos por el algoritmo cíclico puedan ser erróneos.

```
flag_volcado = 0;
strcpy (registro_copia[LATITUD_N], registro[LATITUD_N]);
strcpy (registro_copia[LATITUD_A], registro[LATITUD_A]);
strcpy (registro_copia[LONGITUD_N], registro[LONGITUD_N]);
strcpy (registro_copia[LONGITUD_N], registro[LONGITUD_N]);
strcpy (registro_copia[ALTITUD], registro[ALTITUD]);
flag_volcado = 1;
```

Con esto se acaba la función loop. Sólo quedan funciones de usuario. La primera de ellas es la función *CheckSum* la cual se encarga de calcular el CheckSum de los mensajes de configuración del receptor GNSS.

```
void CheckSum() {
    cs = 0;
    for (posicion = 1; linea[posicion] != '*'; posicion++)      // XOR con cs para cada byte
        cs = cs ^ línea[posición];
    l = cs % 16;                                                // Hexadecimal bajo
    h = (cs – l) / 16;                                          // Hexadecimal alto
    l += 48;                                                    // Conversión a char
    if (l > 57)                                                 // Si es mayor a 9
        l += 7;                                                 // Escribe letra (A-F)
    h += 48;                                                    // Conversión a char
    if (h > 57)                                                 // Si es mayor a 9
        h += 7;                                                 // Escribe letra (A-F)
    linea[++ posición] = h;                                     // Se escribe el CheckSum
    linea[++ posición] = l;
    linea[++ posición] = 0;
}
```

Después va la función del filtro de orientación:

A la hora de implementar este algoritmo con Arduino IDE, hace falta incluir la librería *math.h*. Se empieza por crear una función que utilice los valores recogidos por el MARG para su funcionamiento:

```
void madgwick_actualizarMARG(float gx, float gy, float gz,
                             float ax, float ay, float az,
                             float mx, float my, float mz) {
```

Dentro de la función se empieza por declarar todas las variables que se utilizarán. Todas las variables que se declaran son tipo float.

Se obtiene el valor de la derivada del cuaternión (el cuaternión está formado por variables globales).

```
qDot1 = 0.5f * (-q1 * gx - q2 * gy - q3 * gz);
qDot2 = 0.5f * ( q0 * gx + q2 * gz - q3 * gy);
qDot3 = 0.5f * ( q0 * gy - q1 * gz + q3 * gx);
qDot4 = 0.5f * ( q0 * gz + q1 * gy - q2 * gx);
```

Lo siguiente se calcula sólo en caso de que el valor recibido del acelerómetro sea correcto, por lo tanto se utiliza una condición:

```
if(!((ax == 0.0f) && (ay == 0.0f) && (az == 0.0f)))
```

Primero se normalizan las medidas de acelerómetro y magnetómetro:

```
//Normalización del acelerómetro
recipNorm = invSqrt(ax * ax + ay * ay + az * az);
ax *= recipNorm;
ay *= recipNorm;
az *= recipNorm;

// Normalización del magnetómetro
recipNorm = invSqrt(mx * mx + my * my + mz * mz);
mx *= recipNorm;
my *= recipNorm;
mz *= recipNorm;
```

Ahora, para evitar calcular repetidamente, se calculan una serie de variables auxiliares.

```
_2q0mx = 2.0f * q0 * mx;
_2q0my = 2.0f * q0 * my;
_2q0mz = 2.0f * q0 * mz;
_2q1mx = 2.0f * q1 * mx;
_2q0 = 2.0f * q0;
_2q1 = 2.0f * q1;
_2q2 = 2.0f * q2;
_2q3 = 2.0f * q3;
_2q0q2 = 2.0f * q0 * q2;
_2q2q3 = 2.0f * q2 * q3;
q0q0 = q0 * q0;
q0q1 = q0 * q1;
q0q2 = q0 * q2;
q0q3 = q0 * q3;
q1q1 = q1 * q1;
q1q2 = q1 * q2;
q1q3 = q1 * q3;
q2q2 = q2 * q2;
q2q3 = q2 * q3;
q3q3 = q3 * q3;
```

Con las variables auxiliares ya calculadas, se empieza por calcular la referencia terrestre mediante el campo magnético:

```
hx = mx*q0q0 - _2q0my*q3 + _2q0mz*q2 + mx*q1q1 + _2q1*my*q2 + _2q1*mz*q3 - mx*q2q2 - mx*q3q3;
hy = _2q0mx*q3 + my*q0q0 - _2q0mz*q1 + _2q1mx*q2 - my*q1q1 + my*q2q2 + _2q2*mz*q3 - my*q3q3;
_2bx = sqrtf(hx * hx + hy * hy);
_2bz = -_2q0mx*q2 + _2q0my*q1 + mz*q0q0 + _2q1mx*q3 - mz*q1q1 + _2q2*my*q3 - mz*q2q2 + mz*q3q3;
_4bx = 2.0f * _2bx;
_4bz = 2.0f * _2bz;
```

Se continúa con el algoritmo gradiente descendente y su normalización:

```
// Algoritmo correctivo de gradiente descendente
s0 = -_2q2 * (2.0f * q1q3 - _2q0q2 - ax) + _2q1 * (2.0f * q0q1 + _2q2q3 - ay)
     - _2bz * q2 * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx)
     + (-_2bx * q3 + _2bz * q1) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my)
     + _2bx * q2 * (_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);
s1 = _2q3 * (2.0f * q1q3 - _2q0q2 - ax) + _2q0 * (2.0f * q0q1 + _2q2q3 - ay)
     - 4.0f * q1 * (1 - 2.0f * q1q1 - 2.0f * q2q2 - az)
     + _2bz * q3 * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx)
     + (_2bx * q2 + _2bz * q0) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my)
     + (_2bx * q3 - _4bz * q1) * (_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);
s2 = -_2q0 * (2.0f * q1q3 - _2q0q2 - ax) + _2q3 * (2.0f * q0q1 + _2q2q3 - ay)
     - 4.0f * q2 * (1 - 2.0f * q1q1 - 2.0f * q2q2 - az)
     + (-_4bx * q2 - _2bz * q0) * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx)
     + (_2bx * q1 + _2bz * q3) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my)
     + (_2bx * q0 - _4bz * q2) * (_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);
s3 = _2q1 * (2.0f * q1q3 - _2q0q2 - ax) + _2q2 * (2.0f * q0q1 + _2q2q3 - ay)
     + (-_4bx * q3 + _2bz * q1) * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx)
     + (-_2bx * q0 + _2bz * q2) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my)
     + _2bx * q1 * (_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);
recipNorm = invSqrt(s0 * s0 + s1 * s1 + s2 * s2 + s3 * s3);       // Normalizar magnitud
s0 *= recipNorm;
s1 *= recipNorm;
s2 *= recipNorm;
s3 *= recipNorm;
```

Finalmente se aplica la realimentación:

```
// Aplicar realimentación
qDot1 -= beta * s0;
qDot2 -= beta * s1;
qDot3 -= beta * s2;
qDot4 -= beta * s3;
```

Ahora se sale de la condición de los acelerómetros (por lo tanto, lo que se calcule a partir de ahora se ejecutará sean correctos los valores o no). Se incluye la derivada del *cuaternión* al *cuaternión*:

```
q0 += qDot1 * invSampleFreq;
q1 += qDot2 * invSampleFreq;
q2 += qDot3 * invSampleFreq;
q3 += qDot4 * invSampleFreq;
```

Y finalmente se normaliza el cuaternión:

```
recipNorm = invSqrt(q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3);
q0 *= recipNorm;
q1 *= recipNorm;
q2 *= recipNorm;
q3 *= recipNorm;
anglesComputed = 0;
```

Con esto las variables *q0*, *q1*, *q2* y *q3* tienen el valor del *cuaternión* de orientación calculado. Para que esta función pueda ser calculada se utiliza la función *invSqrt*. En las siguientes líneas de código puede verse la función *invSqrt*:

```
float invSqrt(float x) {
    float halfx = 0.5f * x;
    float y = x;
    long i = *(long *)&y;
    i = 0x5f3759df - (i>>1);
    y = *(float *)&i;
    y = y * (1.5f - (halfx * y * y));
    y = y * (1.5f - (halfx * y * y));
    return y;
}
```

Adicionalmente, se utilizarán los *ángulos de Euler* durante los experimentos, ya que resultan más sencillos de entender, por lo tanto la siguiente función será la encargada de realizar el cambio de *cuaterniones* a *ángulos de Euler*:

```
void computeAngles() {
  roll = atan2f(q0*q1 + q2*q3, 0.5f - q1*q1 - q2*q2);
  pitch = asinf(-2.0f * (q1*q3 - q0*q2));
  yaw = atan2f(q1*q2 + q0*q3, 0.5f - q2*q2 - q3*q3);
  anglesComputed = 1;
}
```

La siguiente es la función que se ejecutará cíclicamente, primero se recogen los datos del MPU 9250, empezando por los acelerómetros. Habrá que realizar un ajuste en función de la sensibilidad escogida en el acelerómetro.

```
void ejecucion_ciclica() {
    //Acelerómetros:
    Wire.beginTransmission(MPU);
    Wire.write(0x3B);                                   //Solicita acelerómtetros
    Wire.endTransmission(FALSE);
    Wire.requestFrom(MPU, 6, true);
    // Acelerómetro OX:
    lectura = (Wire.read() <<8 | Wire.read()) / ajuste[accel];
    frac2alfa();
    strcpy(registro[ACCEL_X], inter);
    // Acelerómetro OY:
    lectura = (Wire.read() <<8 | Wire.read()) / ajuste[accel];
    frac2alfa();
    strcpy(registro[ACCEL_Y], inter);
    // Acelerómetro OZ:
    lectura = (Wire.read() <<8 | Wire.read()) / ajuste[accel];
    frac2alfa();
    strcpy(registro[ACCEL_Z], inter);
```

Se repite el mismo procedimiento para los giróscopos:

```
// Giróscopos:
Wire.beginTransmission(MPU);
Wire.write(0x43);                                          //Solicita giróscopos
Wire.endTransmission(FALSE);
Wire.requestFrom(MPU, 6, true);
// Giróscopo OX:
lectura = (Wire.read() <<8 | Wire.read()) / ajuste[accel];
frac2alfa();
strcpy(registro[GYRO_X], inter);
// Giróscopo OY:
lectura = (Wire.read() <<8 | Wire.read()) / ajuste[accel];
frac2alfa();
strcpy(registro[GYRO_Y], inter);
// Giróscopo OZ:
lectura = (Wire.read() <<8 | Wire.read()) / ajuste[accel];
frac2alfa();
strcpy(registro[GYRO_Z], inter);
```

Para los magnetómetros, tras pedir acceso al chip del magnetómetro se lee el registro 0x02 hasta que contenga el bit 1 activo, en este momento se puede acceder a los registros que contienen los valores de los magnetómetros.

```
Wire.beginTransmission(MAG);
Wire.write(0x0A);
Wire.write(0x11);
Wire.endTransmission(true);
data_ready=0;
while (! (data_ready & 1)) {
    Wire.beginTransmission(MAG);
    Wire.write(0x02);
    Wire.endTransmisission(false);
    Wire.requestFrom(MAG, 1, TRUE);
    data_ready = (Wire.read())
}
Wire.beginTransmission(MAG);
Wire.write(0x03);
Wire.endTransmission(false);
Wire.requestFrom(MAG, 6, true);
itoa((Wire.read() | Wire.read() <<8), sbuffer, 10);
strcpy(registro[MAG_X], sbuffer);
itoa((Wire.read() | Wire.read() <<8), sbuffer, 10);
strcpy(registro[MAG_Y], sbuffer);
itoa((Wire.read() | Wire.read() <<8), sbuffer, 10);
strcpy(registro[MAG_Z], sbuffer);
```

Una vez obtenidos los datos se llama a la función del filtro de orientación, a la se darán los datos para calcular la orientación. Se le solicita también que haga la conversión a *ángulos de Euler* al final:

```
//Calcular orientación:
madgwick_actualizarMARG(registro[GYRO_X], registro[GYRO_Y], registro[GYRO_Z],
registro[ACCEL_X],registro[ACCEL_Y],registro[ACCEL_Z],
registro[MAG_X],registro[MAG_Y],registro[MAG_Z]);
Conversión a ángulos de Euler:
madgwick_computeAngles();
```

Las variables con los *ángulos de Euler* son variables globales, por lo que el resto de funciones pueden acceder a ellas.

Después se procede a programar los PIDs necesarios para el control. Para programar PIDs en microcontroladores se sigue el artículo "Cómo implementar un PID con Microcontroladores" incluido en los anexos. El PID se ejecuta en una interrupción periódica, que asegura que la frecuencia de muestreo es siempre la misma.

Por lo tanto, para el PID, en el bucle principal no hay nada. Todo se encuentra dentro de la propia interrupción (a excepción de la declaración de variables). En caso de saturar la salida, este PID puede dar problemas, por lo que se le incluye un anti wind-up para evitar este problema. Se empieza por el PID que gobierna el timón de profundidad:

```
// Cálculo de controlador:
unsigned long now_pitch = millis();
int timeChange_pitch = (now_pitch – lastTime_pitch);
double error_pitch = altura_consigna_pitch - pitch;
ITerm_pitch += (ki_pitch * error_pitch);
If (ITerm_pitch > outMax)
    ITerm_pitch = outMax;
else if (ITerm_pitch < outMin)
    ITerm_pitch = outMin;
double dInput_pitch = (pitch – lastInput_pitch);
// Calcular salida del PID:
Output_pitch = kp_pitch * error_pitch + ITerm_pitch – kd_pitch * dInput_pitch;
//Se satura la salida para que el número no siga creciendo
If (Output_pitch > outMax)
    Output_pitch = outMax;
else if (Output_pitch < outMin)
    Output_pitch = outMin;
/*Memoria para el próximo ciclo*/
lastInput_pitch = pitch;
lastTime_pitch = now_pitch;
// escribir la salida del controlador
analogWrite(TIMON_PROFUNDIDAD, output);
```

Después se utiliza el mismo método para el PID que controla el timón de dirección:

```
// Calculo de controlador:
unsigned long now_roll = millis();
int timeChange_roll = (now – lastTime_roll);
double error_roll = consigna_giro - roll;
ITerm_roll += (ki * error);
If (ITerm_roll > outMax)
    ITerm_roll = outMax;
else if (ITerm_roll < outMin)
    ITerm_roll = outMin;
double dInput_roll = (roll – lastInput_roll);
// Calcular salida del PID:
Output_roll = kp_roll * error_roll + ITerm_roll – kd_roll * dInput_roll;
//Se satura la salida para que el número no siga creciendo
If (Output_roll > outMax)
    Output_roll = outMax;
else if (Output_roll < outMin)
    Output_roll = outMin;
/*Memoria para el próximo ciclo*/
lastInput_roll = roll;
lastTime_roll = now_roll;
// escribir la salida del controlador
analogWrite(TIMON_DIRECCION, output);
```

Para calibrar estos PIDs se utiliza el apoyo de Simulink, con los lazos de control propuestos antes. Se busca una primera respuesta lenta y sobreamortiguada, para garantizar la integridad del aeromodelo durante las primeras pruebas. Esto se debe a que el aeromodelo escogido es muy estable y una respuesta sobreamortiguada da un mayor margen para errores, siendo el peor caso una respuesta extremadamente lenta por parte del control. En este caso el aeromodelo se podría salvar sin problemas para reajustar el PID heurísticamente.

5 Resultados

Primero se ha validado el algoritmo del filtro de orientación, con el fin de contrastar su funcionamiento: primero en un banco de ensayos y después en movimiento. Después se ha realizado un estudio de los tiempos de los subprocesos que constituyen el programa final. Finalmente se ha llevado a cabo una prueba del programa principal con un aeromodelo diferente.

5.1 Validación del algoritmo

Para validar el algoritmo del filtro de orientación, se realizan una serie de experimentos utilizando un sistema comercial junto con un MARG (Magnetic, Angular Rate and Gravity), al que se le aplica el filtro anteriormente presentado. El sistema comercial escogido para la comparación es un XSens MTi-7 mientras que el MARG es un MPU-9250, el cual incluye una unidad de medida inercial (acelerómetro y giróscopo) junto con un magnetómetro.

5.1.1 XSens MTi-7

El sistema AHRS investigado en este caso es MTi 7 de XSens. Utiliza un filtro Kalman para determinar la orientación a partir de la información recibida en acelerómetros, giróscopos y magnetómetros. También dispone de conexión a chip GNSS externo para recibir información de la posición y velocidad e incluirla en la trama de datos que facilita para el microcontrolador que se desee usar en la aviónica.

Se ha decidido utilizar comunicación SPI para determinar la viabilidad de usar este chipset instrumental. Para recibir la trama de datos vía SPI hay que tener en cuenta que:

Para permitir comunicación SPI el firmware del MTi 1-series debe ser superior a 1.0.3; de lo contrario no es posible actualizar el firmware a una versión que permita este tipo de comunicación, según indica el datasheet de la serie.

La explicación detallada del funcionamiento de la librería que se ha diseñado se encuentra en los anexos.

Ilustración 40: XSens MTi-7

5.1.2 MPU-9250

El IC MPU-9250 es un conjunto IMU + magnetómetro que se vende como módulo para plataformas de desarrollo de hardware y es aproximadamente 200 veces más barato que un XSens MTi-7.

5.1.3 Prueba en banco de pruebas

Se sitúa en una superficie plana un servomotor con su eje móvil perpendicular a la superficie. Unido mecánicamente al cuerpo del servomotor, una caja con: un Arduino MEGA, un XSens MTi-7 y un MPU9250. XSens MTi-7 está conectado al Arduino MEGA mediante SPI mientras que MPU9250 se conecta mediante I²C. El servomotor también es gobernado por el Arduino MEGA.

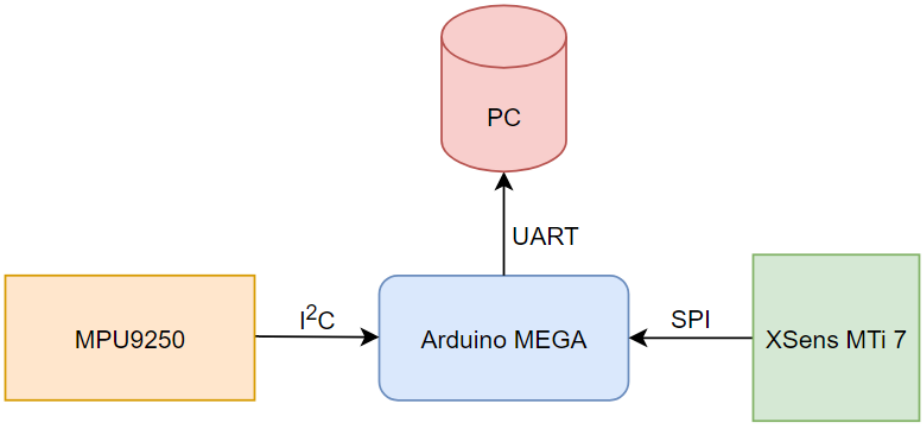


Ilustración 41: Diagrama de comunicaciones del banco de ensayos

Durante esta prueba el servomotor empieza centrado y se mueve de forma controlada en una dirección y en la otra, para finalmente volver a la posición de inicio.

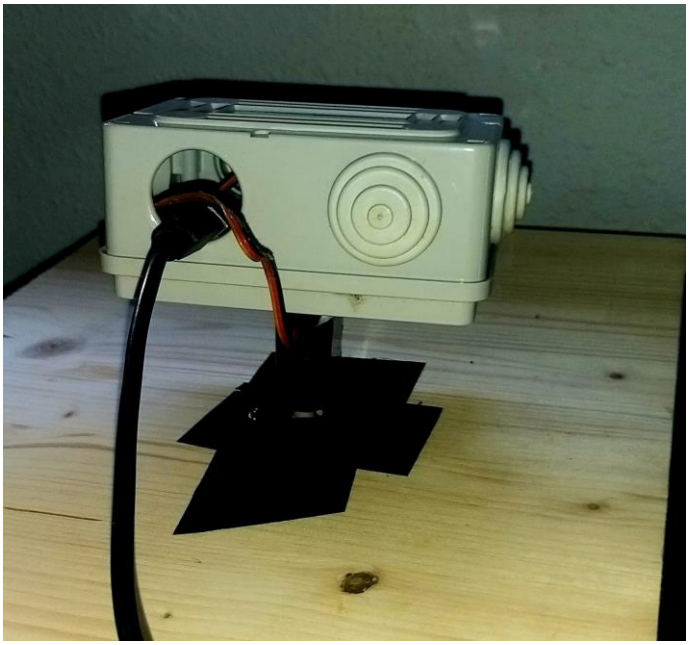


Ilustración 42: Banco de pruebas

Mediante el cable USB, que también se utiliza para programar el Arduino MEGA, se transfiere la información vía UART al PC para su posterior análisis utilizando Excel.

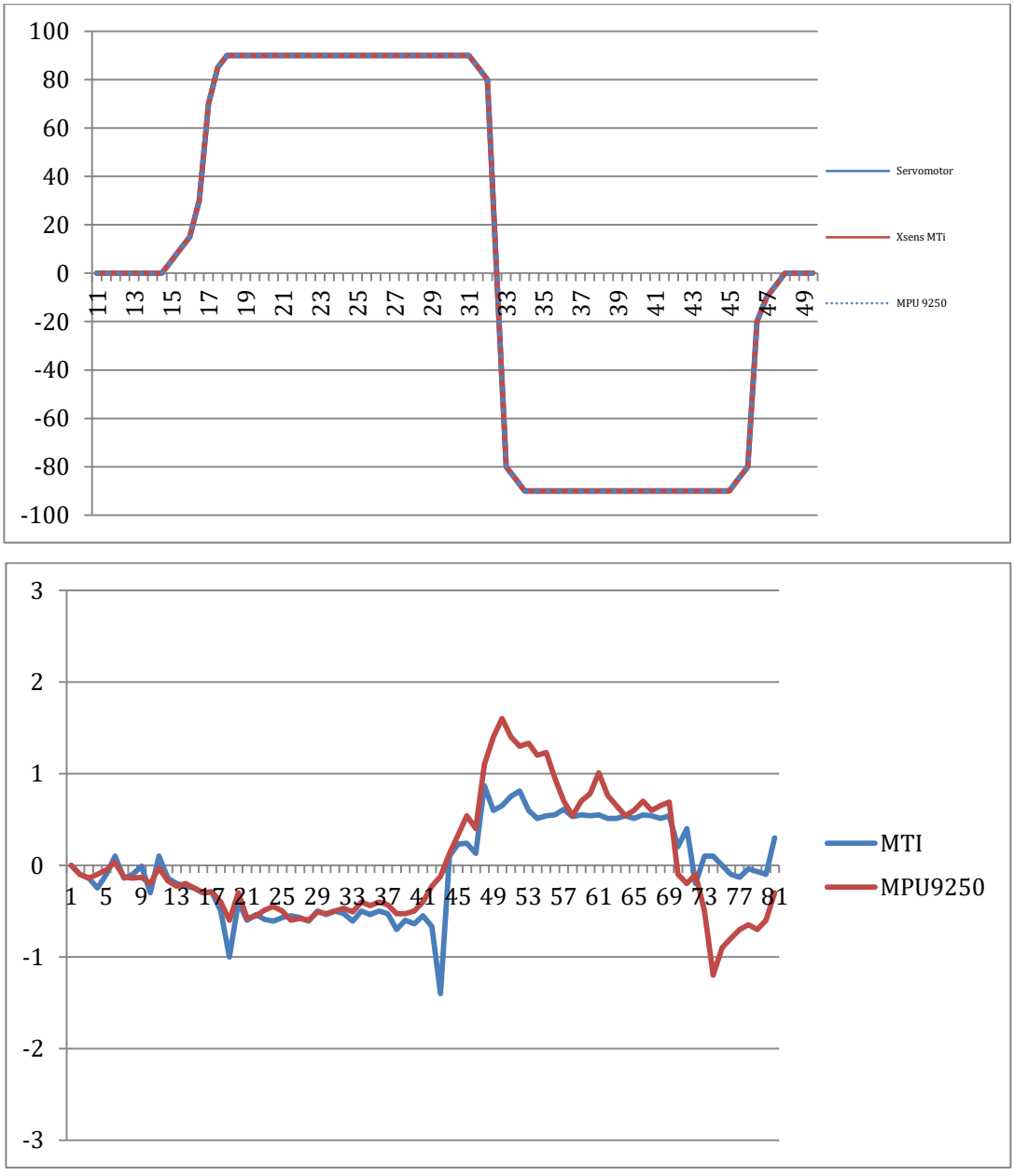


Ilustración 43: Resultados de la prueba. Ángulo medido en grados (Arriba) y Error (Abajo)

Los resultados de la prueba del sistema comercial y el planteado son muy similares y por lo tanto en esta memoria el resultado se considera positivo y se procede a continuar con un experimento más dinámico.

5.1.4 Pruebas dinámicas

Un Arduino UNO junto con el MPU9250 son introducidos en una caja. Esta caja incluye alimentación y un módulo de tarjeta SD para el almacenamiento de los datos.

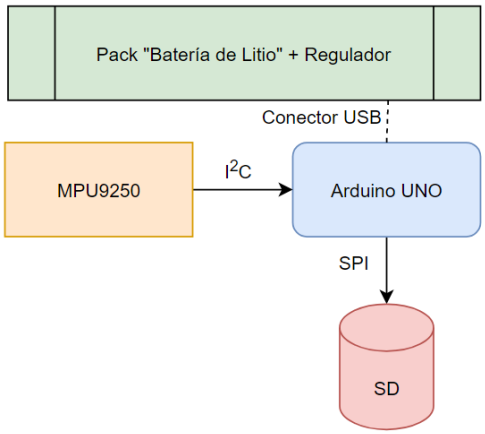


Ilustración 44: Diagrama de comunicaciones del experimento dinámico

La caja se sujeta a las *aerobarras* del manillar de una bicicleta de triatlón, con el fin de que mantenga una buena estabilidad. El experimento se realiza dando vueltas por el Bidegorri de Barakaldo (Bizkaia) en la bicicleta, con el sistema encendido para recoger datos.

Una vez terminado el experimento se deduce que el filtro es apto para realizar el seguimiento de la orientación de la bicicleta durante el trayecto. Por analogía, debido al paralelismo que hay entre las fuerzas a las que se ven sometidos tanto la bicicleta como un avión, el filtro es apto para su uso en el seguimiento de orientación en aeromodelos.

Ilustración 45: Experimento dinámico con bicicleta. Se puede observar la caja que contiene la electrónica encima de las aerobarras

5.2 Tiempos de los subprocesos

Las primeras pruebas se hicieron con el receptor GNSS. Leyendo datos del receptor a 38.400 bps, se termina en 19,25 ms. Los procesos de parseado y extracción de los datos desde el buffer consumen unos 150 us. Por lo tanto el proceso de lectura del receptor GNSS es de 19,4 ms.

Con la radio se transmiten datos de 8 bits, más uno de arranque y otro de parada a una velocidad de 9.600 bps (960 bytes por segundo) . Teniendo en cuenta que la trama es de 44 bytes (Carácter de arranque, 3 float para la posición GNSS, 3 float para los ángulos de Euler, un float para el ángulo de la trayectoria y carácter de parada), se calcula que su transmisión tardará 49 ms.

Las medidas con el MARG indican tiempos en torno a los 12 ms para leer y 2,5 ms para la conversión de los datos. Esto hace un total de 14,5 ms.

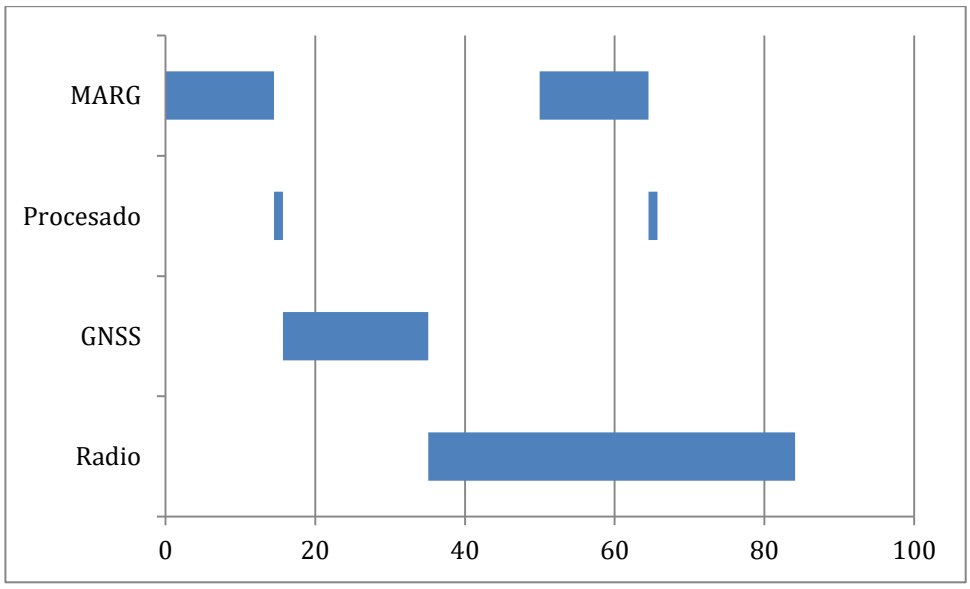


Ilustración 46: Representación del tiempo de los subprocesos

A simple vista puede parecer que la transmisión por radio se solapa con la interrupción cíclica, pero, dado que la escritura en el buffer UART es más rápida que la propia transmisión, esto no es así.

Por un lado, si da tiempo a entregar todo al buffer, la transmisión seguirá sin intervención del microcontrolador, tal y como se ve en la ilustración anterior. En caso contrario, habrá un retardo, entre dos de los bytes transmitidos por radio, de 15,7 ms. En cualquier caso la comunicación no se ve mermada y los tiempos se cumplen.

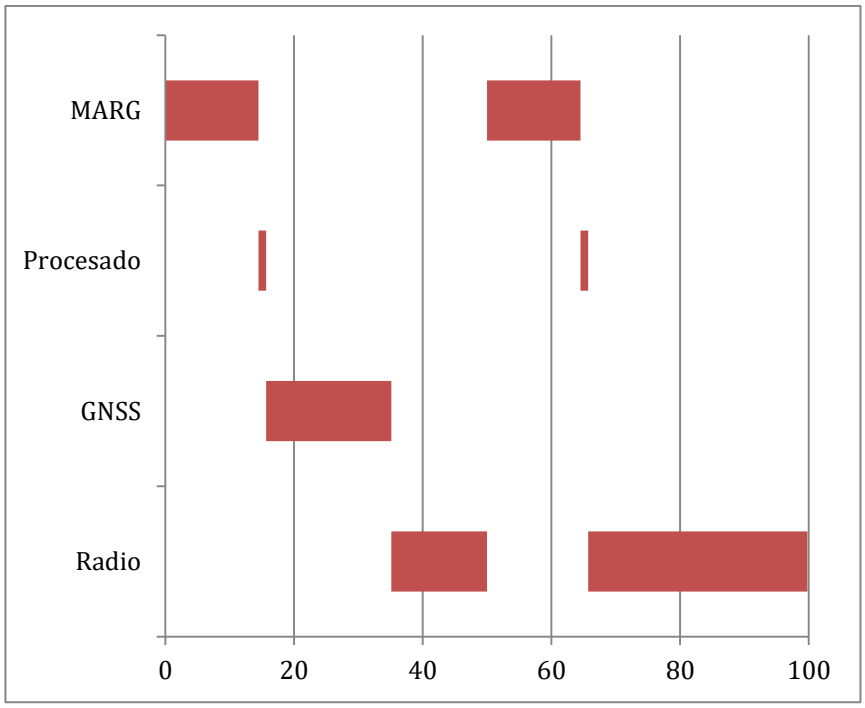


Ilustración 47: Representación del tiempo de los subprocesos con interrupción intermedia

5.3 Prueba con aeromodelo

Debido a problemas con la impresora 3D, las primeras pruebas de vuelo se realizan con un aeromodelo comercial (Una réplica del Piper J-3 Cub concretamente).

Ilustración 48: Piper J-3 Cub

Tanto las pruebas de vuelo recto y nivelado como de giro son satisfactorias y los algoritmos de control hacen correctamente su trabajo. Pese a esto, se observa un movimiento extraño a la hora de volar con vientos fuertes.

Cuando no hay mucho viento, el aeromodelo se dirige directamente hacia el punto de destino, tal y como cabe esperar.

Ilustración 49: Vuelo sin viento

Cuando un viento constante y fuerte golpea al aeromodelo, éste empieza a actuar raro: comienza a girar en dirección al viento para después empezar a hacer zig-zag. Las conclusiones de los experimentos son las siguientes.

Primero hay que tener en cuenta el escenario, el avión va directamente al punto marcado y recibe viento fuerte desde un costado.

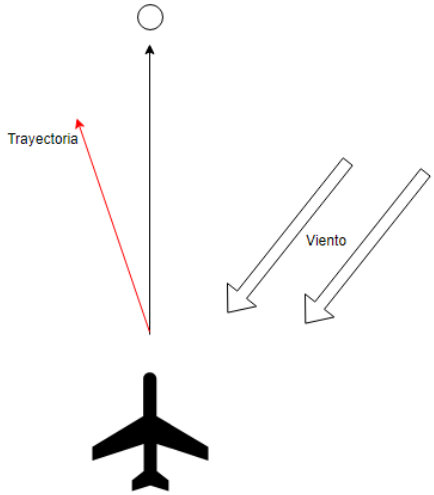


Ilustración 50: Avión en vuelo recibe viento fuerte

Para compensar el cambio de trayectoria, éste gira contra el viento.

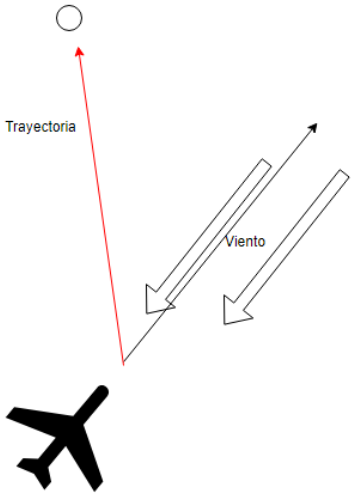


Ilustración 51: Compensación de la trayectoria

Si el viento es lo suficientemente fuerte, el avión girará más aún. Este giro excesivo desemboca en que el algoritmo de control de trayectoria solicite un giro en sentido contrario. Cuando este segundo giro se excede nuevamente el sentido de giro vuelve a cambiar.

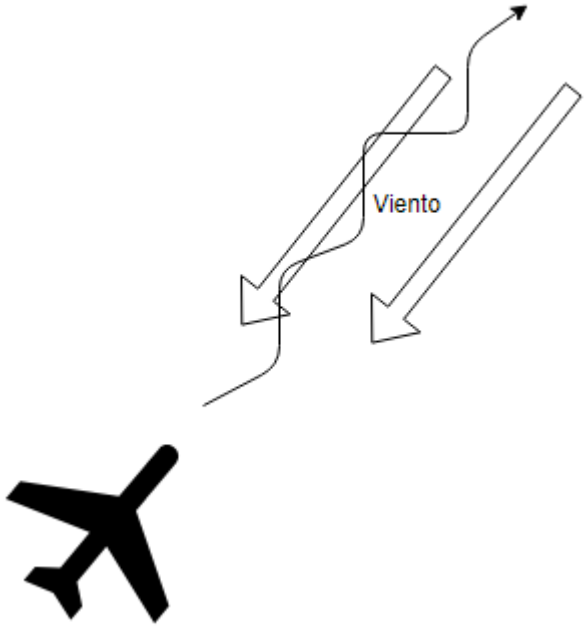


Ilustración 52: Reacción del avión volando contra fuertes vientos

La razón de este comportamiento se debe a que la velocidad del viento es mayor a la velocidad de movimiento (con respecto al aire) del aeromodelo. Este hecho hace que la suposición de que la diferencia entre el vector de trayectoria y el de orientación sea despreciable no se cumpla, debido a que el vector de trayectoria bien podría tener una diferencia de 180º con la orientación de la aeronave en este caso.

Cuando el aeromodelo vuela contra el viento fuerte, decide girar en una de las dos direcciones.

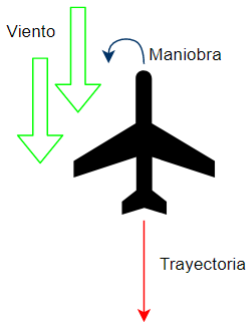


Ilustración 53: Comportamiento volando frente al viento

Este giro provoca un cambio en la trayectoria.

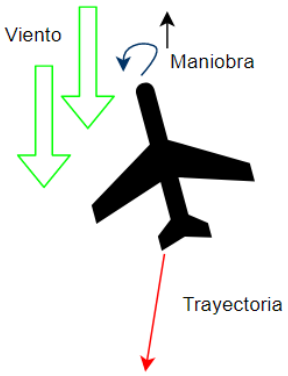


Ilustración 54: Cambio del vector de trayectoria

Este cambio en la trayectoria afecta al lazo de control de trayectoria puesto que ahora es, según el algoritmo que se ha programado, más eficiente girar en la otra dirección debido a que el ángulo es menor entre el vector de trayectoria y el de destino. El cambio de giro continúa hasta que el viento fuerte desaparece.

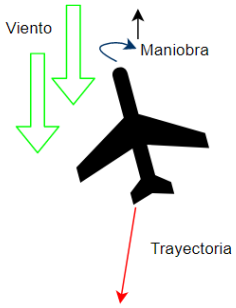


Ilustración 55: Cambio del sentido de giro

6 Conclusiones

El sistema P.A.T.O. es un sistema de bajo coste capaz de realizar tanto vuelos de búsqueda como vuelos experimentales para la carga en la bahía (del aeromodelo de ala fija que controla) de forma autónoma y sin necesidad de la intervención de un operador (aunque es necesaria su supervisión por motivos de seguridad). Pero como todo sistema de bajo coste, éste tiene sus limitaciones.

6.1 Autonomía

En lo referente al "alcance y autonomía" del vehículo escogido para P.A.T.O., no dispone de demasiada autonomía debido a la poca energía que es capaz de cargar (en parte se debe a ser eléctrica su propulsión).

6.1.1 Turbofan o Turbojet

Como propuesta de mejora se propone el diseño y fabricación de un motor turbojet o un turbofan. Esto presentaría nuevos problemas debido al movimiento del combustible dentro del tanque y al cambio de la masa total a lo largo del vuelo, así como el movimiento del centro de masas a lo largo del mismo.

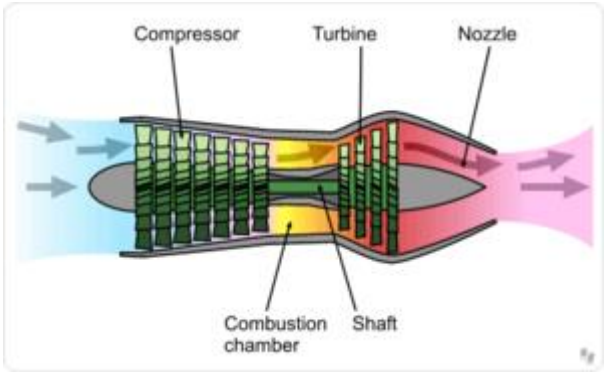


Ilustración 56: Esquema básico de un turbojet

6.1.2 Paneles solares

Otra opción sería incorporar paneles solares en la parte superior de la superficie del aeromodelo. Se sugiere ponerlos en las alas y, mediante un convertidor DC-DC, recargar las baterías (principalmente las baterías de la propulsión, pero la aviónica debe recargarse también, así que esto desemboca en un diseño de BMS o Batery Management System muy interesante). Como desventaja esta solución sólo puede utilizarse durante el día y para operaciones nocturnas únicamente añade peso al aeromodelo.

6.2 Aeromodelo

El aeromodelo escogido para llevar el prototipo P.A.T.O. es perfectamente capaz de llevar a cabo las tareas que se le han asignado: reconocimiento, búsqueda de objetivos, vuelo prediseñado para experimentos…

6.2.1 Estabilidad

Esto se debe en gran parte a que las respuestas del aeromodelo son lentas, la velocidad no es muy alta y que el propio diseño del modelo se puede catalogar como estable. Esta estabilidad implica que el avión tiende a mantener un vuelo recto y nivelado por sí solo. Es similar a tener una canica sobre una superficie en forma de "U": por mucho que la canica se mueva hacia uno de los extremos, en el instante que deje de intervenir la fuerza que la ha movido, ella misma volverá al centro de forma natural; también tiene un cierto coste energético mover la canica y es una buena forma de ilustrar la lentitud del sistema. Un ejemplo de un aeromodelo estable sería una avioneta.

Sería interesante desarrollar una siguiente versión del sistema P.A.T.O. con un aeromodelo más críticamente estable y que por lo tanto sea más rápido en reacciones. Un símil del sistema críticamente estable sería una canica en una llanura. No requiere mucha energía mover la canica (por lo tanto velocidad de reacción más alta) y ésta no volvería al centro por sí sola. Un ejemplo sería un caza, como el F-22.

No se aconseja el uso de P.A.T.O. en un aeromodelo inestable. Éste tendría reacciones excesivamente rápidas y presentaría muchos problemas para tan sólo mantener un vuelo recto y nivelado. Un símil sería una canica en una "U" invertida. No hay costo energético al desplazar la canica fuera del centro, pero costaría volver a él (y no volvería sola, más bien lo contrario). Como ilustración de este caso, la NASA necesitó 3 ordenadores en paralelo para realizar las correcciones necesarias para el vuelo recto y nivelado en su X-29, cosa que justifica este consejo sobradamente.

6.2.2 Sustentación, entrada en pérdida e hipersustentadores

El coeficiente de sustentación máximo determina la carga alar del avión. Por lo tanto, desarrollar dispositivos hipersustentadores (Flaps y Slats) para aumentar el valor del coeficiente de sustentación máximo sería una ampliación interesante. Más detalles acerca del tema en [1].

6.2.3 Velocidad

La aeronave que P.A.T.O. gobierna está diseñada para vuelos lentos de cara a una mejor observación durante las operaciones de reconocimiento y búsqueda. Puede ser interesante rediseñar la aeronave para permitir vuelos más rápidos con la intención de ampliar las posibilidades de los vuelos con fines experimentales (experimentos en la bahía de carga), así como su alcance.

6.3 Posicionamiento global

En principio P.A.T.O. dispone de un sistema G.N.S.S. conectado a la red GPS norteamericana para conocer su ubicación en el globo (así como la altura).

6.3.1 Precisión

La precisión de una sola red de satélites G.N.S.S. es suficiente para la aeronavegación, pero puede ser insuficiente para un buen aterrizaje. Para mejorar la precisión se pueden usar 2 redes de satélites simultáneamente (con 2 receptores G.N.S.S.) a costa de un mayor consumo eléctrico.

6.3.2 Pérdida de la señal

Cuando las antenas de base se encuentran en mantenimiento, la señal G.N.S.S. perderá precisión. Esto es inevitable, pero generalmente se tiene cobertura de hasta 3 antenas de base, por lo que no afectaría. Éstas sirven para corregir errores y conseguir una mayor precisión.

Ilustración 57: Antena base GNSS

En caso de tormenta eléctrica es muy probable que la tormenta produzca interferencias y por lo tanto se pierda señal G.N.S.S. Es una de las principales razones por las que no está permitido usar P.A.T.O. cuando exista la posibilidad de tormenta eléctrica.

Pese a esto, un rediseño del conjunto antena + amplificador, del receptor GNSS puede ser interesante como mejora.

6.4 Algoritmos de control

Los algoritmos de control de la aeronave cumplen su función según lo estipulado. Pese a ello, es posible mejorar ambos algoritmos de control.

6.4.1 Control Guiado

El control guiado descrito en esta memoria realiza los giros tal y como se explica en los manuales de aeronavegación (ver [2] para más detalle). Las maniobras de ascenso y descenso, por el contrario, no siguen estas normas al pie de la letra; sería interesante modificar el algoritmo de control guiado para que haga estas maniobras según el manual de aeronavegación.

6.4.2 Control de trayectoria

El control de trayectoria descrito aquí supone que la diferencia entre el vector de trayectoria y el de orientación son despreciables. Para la mayoría de trayectos este control es adecuado, pero, en caso de que tenga viento de frente a una velocidad mayor que su velocidad aerodinámica, el vector de trayectoria apuntaría en dirección opuesta.

Dado el diseño del control, P.A.T.O. intentaría corregir su trayectoria girando a un lado. Este giro modificaría el vector de trayectoria lo suficiente como para que el lazo de control aplique un giro al otro lado. Esto provocaría un movimiento en *zig-zag* en dirección al destino escogido.

Haciendo que la maniobra sea progresiva este zig-zag se vería minimizado, debido al continuo cambio de sentido. Modificar el control de trayectoria de esta manera, supondría un aumento de eficiencia para el proyecto.

6.4.3 Modo Repetidor

Cuando P.A.T.O. recibe la señal de radio del objetivo que está buscando, sigue su trayectoria. Añadir una funcionalidad llamada Modo Repetidor es un paso lógico para este proyecto.

En el momento en el que la bahía de carga (equipada con un radiorrepetidor para el objetivo a buscar) recibe la señal de baliza del objetivo, podrían hacerse dos cosas:

- La posición GNSS actual es almacenada como una variable nueva y la máquina de estados cambia a un estado nuevo que incluye esta posición. Esto implica que el aeromodelo girará en círculos intentando ir al punto una y otra vez. Tras recibir el *ACK* desde base, se continúa con la máquina de estados original (lista original de puntos GNSS).
- El radiorrepetidor recibe y desencripta la posición del objetivo. La posición desencriptada se entrega a P.A.T.O., que crea una nueva variable con esta nueva posición y repite el proceso descrito en el párrafo anterior.

Es difícil saber cuál de los dos métodos es mejor. En el caso de recibir un rebote de la señal momentáneamente, dar vueltas alrededor del punto actual no tendría mucho sentido; en el caso de que el objetivo esté cubierto por algo metálico (un tejado metálico por ejemplo), posicionarse sobre él llevaría al radiorrepetidor a un punto de sombra. El desarrollo de este modo de funcionamiento, así como la relación de compromiso entre los dos métodos (o incluso algún método alternativo) serán objeto de la próxima iteración de P.A.T.O.

6.5 Lazo de control de velocidad aerodinámica

El control de motor actualmente en este aeromodelo consiste en un "todo o nada". Añadir un control de velocidad que sea capaz de mantener la velocidad aerodinámica del aparato, e incluso de asegurar una velocidad aerodinámica mínima para mantener la sustentación del avión, es algo a tener en cuenta como ampliación del sistema.

Como instrumento de medición de la velocidad aerodinámica se utilizaría un tubo de *Pitot* y como motor se utilizaría un motor síncrono o un brushless (siendo este segundo el más lógico por su bajo coste). Para gobernar el motor se propone el integrado DRV8343-Q1 de Texas Instruments, que permite incluso el control de motores brushless con sensores Hall.

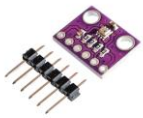


Ilustración 58: Módulo Arduino compatible BMP280 (Sensor Barométrico)

Para el tubo de *Pitot* se propone el uso de los sensores barométrico BMP280. Es posible realizar un pequeño montaje en el que se midan tanto la presión total (con uno de los sensores) como la presión estática (con otro). Teniendo ambas, se puede obtener la presión dinámica y desde ahí calcular la velocidad aerodinámica del avión. En caso de querer incorporar el sensor a una PCB propia el siguiente esquema puede servir como referencia.

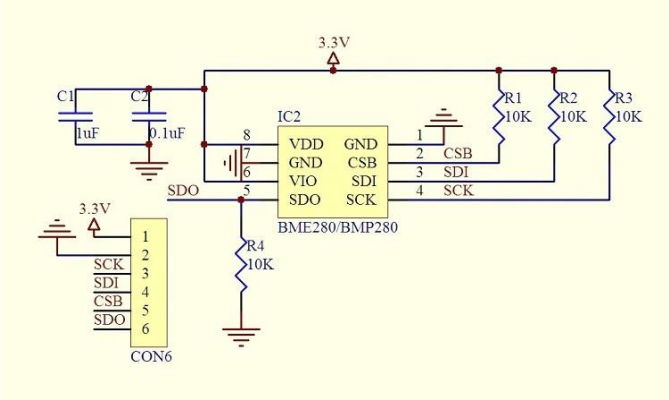


Ilustración 59: Esquemático BME280 (Chip de BMP280)

7 Bibliografía

7.1 Aerodinámica y Aeronáutica

[1] J. Messeguer Ruiz & A. Sanz Andrés, "Aerodinámica básica 2ª Edición", Gareeta

[2] R. Machado, Escuela básica de Rod Machado, Microsoft Flight Simulator 2002. http://flightsimulator-ra.com.ar/download/aeronautica%20-%20manual%20de%20vuelo%20(1).pdf

[3] Sergio Hidalgo, "Cómo diseñar un AVIÓN para que sea ESTABLE ? TE ENSEÑO A HACERLO !!". https://youtu.be/owMY_eJGt3o

7.2 Software y Control

[4] Sebastian O.H. Madgwick, Andrew J.L. Harrison, Ravi Vaidyanathan , "Estimation of IMU and MARG orientation using a gradient descent algorithm", IEEE International Conference on Rehabilitation Robotics 2011.

[5] Filtro Madgwick modificado: http://www.ijcee.org/vol10/974-T4004.pdf

[6] Rocío Carratalá Sáez, Los cuaterniones y su importancia en la representación gráfica por ordenador, Julio 2015. http://repositori.uji.es/xmlui/bitstream/handle/10234/139036/TFG_2015_CarratalaSaezR.pdf?sequence=1&isAllowed=y

[7] Sebastian O.H. Madgwick, An efficient orientation filter for inertial and inertial/magnetic sensor arrays, Abril 2010. https://www.x-io.co.uk/res/doc/madgwick_internal_report.pdf

[8] A. Elgezabal Núñez, Cómo implementar un PID con microcontroladores, Diciembre 2020

7.3 Hardware

[9] Ford Motor Company, EMC Design Guide for Printed Circuit Boards, Octubre 2002

[10] Infineon, EMC and System-ESD Design Guidelines for Board Layout, Febrero 2016

[11] Toradex, Layout Design Guide, Abril 2015

8 Anexos

8.1 Micromod Estándar M.2

Tablas Pin-Out Micromod Estándar M.2:

Función				Pin inferior
			NC	
			3.3 V	74
			RTC_3V_BATT	72
		SPI_CS1#	SDIO_DATA3(I/O)	70
			SDIO_DATA2(I/O)	68
			SDIO_DATA1(I/O)	66
		SPI_CIPO1	SDIO_DATA0(I/O)	64
		SPI_COPI1	SDIO_CMD(I/O)	62
		SPI_SCK1	SDIO_SCK(O)	60
			AUD_MCLK(O)	58
CAM_MCLK	PCM_OUT	I2S_OUT	AUD_OUT	56
CAM_PCLK	PCM_IN	I2S_IN	AUD_IN	54
PDM_DATA	PCM_SYNC	I2S_WS	AUD_LRCLK	52
PDM_CLK	PCM_CLK	I2S_SCK	AUD_BCLK	50
			G4/BUS4	48
			G3/BUS3	46
			G2/BUS2	44
			G1/BUS1	42
			G0/BUS0	40
			A1	38
			GND	36
			A0	34
			PWM0	32
			Module Key	30
			Module Key	28
			Module Key	26
			Module Key	24
			UART_TX2 (O)	22
			UART_RX2 (I)	20
		CAM_TRIG	D1	18
			I2C_INT#	16
			I2C_SCL(I/O)	14
			I2C_SDA (I/O)	12
			D0	10
		SW0	G11	8
			RESET# (I- Open Drain)	6
			3.3 V_EN	4
			3.3 V	2

Pin superior	Función		
75	**GND**		
73	G5/BUS5		
71	G6/BUS6		
69	G7/BUS7		
67	G8		
65	G9	ADC_D-	CAM_HSYNC
63	G10	ADC_D+	CAM_VSYNC
61	SPI_CIPO (I)		
59	SPI_COPI (O)	LED_DAT	
57	SPI_SCK (O)	LED_CLK	
55	SPI_CS#		
53	I2C_SCL1 (I/O)		
51	I2C_SDA1 (I/O)		
49	**BATT_VIN / 3 (I-ADC) (0 to 3.3 V)**		
47	PWM1		
45	**GND**		
43	CAN_TX		
41	CAN_RX		
39	**GND**		
37	USBHOST_D-		
35	USBHOST_D+		
33	**GND**		
31	Module Key		
29	Module Key		
27	Module Key		
25	Module Key		
23	SWDIO		
21	SWDCK		
19	UART_RX1 (I)		
17	UART_TX1 (O)		
15	UART_CTS1 (I)		
13	UART_RTS (O)		
11	BOOT (I – Open Drain)		
9	USB_VIN		
7	**GND**		
5	USB_D-		
3	USB_D+		
1	**GND**		

Tabla descriptiva de pines MicroMod Estándar M.2:

Grupo	Señal	I/O	Descripción	Tensión
Power	3.3 V	I	3.3 V Source	3.3 V
	GND		Gnd	0 V
	USB_VIN	I	USB VIN compliant to USB 2.0 specification Connect to pin son precessor board that require 5V for USB functionality	4.8 – 5.2 V
	RTC_3V_BATT	I	3 V procided by external coin cell or mini battery. Max draw = 100uA. Connect to pins mantaining an RTC during power loss. Can be left NC.	3 V
	3.3_EN	O	Controls the carrier board's main voltage regulator, Voltage above 1 V will enable 3.3 power path.	3.3 V
	BATT_VIN/3	I	Carrier boards raw voltage over 3. 1/3 resistor dividr is implemente don carrier board. Amplify the analog signal as neede for full 0-3.3 V range	3.3 V

VGrupo	Señal	I/O	Descripción	Tensión
Reset	Reset	I	Input to processor. Open drain with pullup on processor board. Pulling low resets processor.	3.3 V
	Boot	I	Input to proceso. Open drain with pullup on processor board. Pulling low puts processor into special boot mode. Can be left NC.	3.3 V

Grupo	Señal	I/O	Descripción	Tensión
USB	USB_D	I/O	USB Data. Diferential serial data interface compliant to USB 2.0 specification. If UART is required for programming USB must be routed to USB-to-serial conversión IC on the processor board.	
USB Host	USBHOST_D	I/O	For processors that support USB Host Mode. USB Dat Differential serial data interface copliant to USB 2.0 specification. Can be left NC.	

Grupo	Señal	I/O	Descripción	Tensión
CAN	CAN_RX	I	CAN Bus receive data.	3.3 V
	CAN_TX	O	CAN Bus transmit data.	3.3 V
UART	UART_RX1	I	UART receive data	3.3 V
	UART_TX1	O	UART transmit data	3.3 V
	UART_RTS1	O	UART ready to send	3.3 V
	UART_CTS1	I	UART clear to send	3.3 V
	UART_RX2	I	2nd UART receive data	3.3 V
	UART_TX2	O	2nd UART transmit data	3.3 V

Grupo	Señal	I/O	Descripción	Tensión
I2C	I2C_SCL	I/O	I2C clock. Open drain with pullup on carrier board	3.3 V
	I2C_SDA	I/O	I2C data. Open drain with pullup on carrier board.	3.3 V
	I2C_INT#	I	Interrupt notification from carrier board to processor. Open drain with pullup on carrier board. Active LOW.	3.3 V
	I2C_SCL1	I/O	2nd I2C clock. Open drain with pullup on carrier board.	3.3 V
	I"C_SDA1	I/O	2nd I2C data. Open drain with pullup on carrier board.	3.3 V

Grupo	Señal	I/O	Descripción	Tensión
SPI	SPI_COPI	O	SPI Controller Output/Peripheral Input	3.3 V
	SPI_CIPO(I)	I	SPI Controller Input/Peripheral Output	3.3 V
	SPI_SCK	O	SPI Clock	3.3 V
	SPI_CS#	O	SPI Chip Select. Active LOW. Can be routed to GPIO if hardware CS is unused.	3.3 V
SPI/SDIO	SPI_SCK1/SDIO_CLK	O	2nd SPI Cloc. Secondary use is SDIO Clock	3.3 V
	SPI_COPI1/SDIO_CMD	I/O	2nd SPI Controller Output/Peripheral Input. Secondary use is SDIO command interface.	3.3 V
	SPI_CIPO1/SDIO_DATA0	I/O	2nd SPI Peripheral Input/Controller Output. Secondary use is SDIO data Exchange bit 0.	3.3 V
	SDIO_DATA1	I/O	SDIO data Exchange bit 0.	3.3 V
	SDIO_DATA2	I/O	SDIO data Exchange bit 0.	3.3 V
	SPI_CS1/SDIO_DATA3	I/O	2nd SPI Chip Select. Secondary use is SDIO data Exchange bit 0.	3.3 V

Grupo	Señal	I/O	Descripción	Tensión
Audio	AUD_MCLK	O	Audio master clock	3.3 V
	AUD_OUT/PCM_OUT/I2S_OUT/CAM_MCLK	O	Audio data output. PCM synchronus data output. I2S serial data out. Camera master clock	3.3 V
	AUD_IN/PCM_IN/I2S_IN/CAM_PCLK	I	Audio data input. PCM synchronus data input. I2S serial data in. Camera peripheral clock	3.3 V
	AUD_LRCLK/PCM_SYNC/I2S_WS/PDM_DATA	I/O	Audio left/right clock. PCM Synchronous data SYN. I2S Word select. PDM data.	3.3 V
	AUD_BCLK/PCM_CLK/I2S_CLK/PDM_CLK	O	Audio bit clock. PCM clock. I2S continous serial clock. PDM clock.	3.3 V
SWD	SWDIO	I/O	Serial Wire Debug I/O. Connect if processor board suppors SWD. Can be left NC.	3.3 V
	SWDCK	I	Serial Wire Debug clock. Connect if processor board supports SWD. Can be left NC.	3.3 V

Grupo	Señal	I/O	Descripción	Tensión
ADC	A0	I	Analog to digitial converter 0. Amplify the analog signal as needed to enable full 0-3.3V range.	3.3 V
	A1	I	Analog to digital converter 1. Amplify the analog signal to enable full 0-3.3V range.	3.3 V
PWM	PWM0	O	Pulse Width Modulated Output 0	3.3 V
	PWM1	O	Pulse Width Modulated Output 1	3.3 V
Digital	D0	I/O	General digital input/output pin.	3.3 V
	D1/CAM_TRIG	I/O	General digital input/output pin. Camera Trigger.	3.3 V

Grupo	Señal	I/O	Descripción	Tensión
General/Bus	G0/BUS0	I/O	General purpose pins. Any unusd processor pins should be assigned to Gx with ADC + PWM capable pins given priority (0, 1, 2, etc.) positions. The intent is to guarantee PWM ADC and Digital Pin functionality on respective ADC/PWM/Digital pins. Gx pins do not guarantee ADC/PWM function. Alternativ use is pins can support a fast read/write 8-bit or 4-bit wide bus.	3.3 V
	G1/BUS1	I/O		3.3 V
	G2/BUS2	I/O		3.3 V
	G3/BUS3	I/O		3.3 V
	G4/BUS4	I/O		3.3 V
	G5/BUS5	I/O		3.3 V
	G6/BUS6	I/O		3.3 V
	G7/BUS7	I/O		3.3 V
	G8	I/O	General purpose pin	3.3 V
	G9/ADC_D-/CAM_HSYNC	I/O	Differential ADC input if available. Camera horizontal syn	3.3 V
	G10/ADC_D+/CAM_VSYNC	I/O	Fieferential ADC input if available. Camera vertical sync.	3.3 V
	G11/SWO	I/O	General purpose pin. Serial Write Output	3.3 V

8.2 Filtro de orientación

8.2.1 Representación por "Cuaterniones"

La siguiente información se ha obtenido de [7]. Un cuaternión es un número complejo de cuatro dimensiones que puede utilizarse para representar la orientación de un cuerpo fijo o el marco de coordenadas en un espacio tridimensional. Una orientación arbitraria del marco B relativo al marco A se puede conseguir mediante una rotación del ángulo θ sobre el eje A_r definido en el marco A. Esto se representa en la siguiente figura, donde los vectores mutuamente ortogonales x_A, y_A y z_A y x_B, y_B y z_B definen los principales ejes de coordenadas de los marcos A y B.

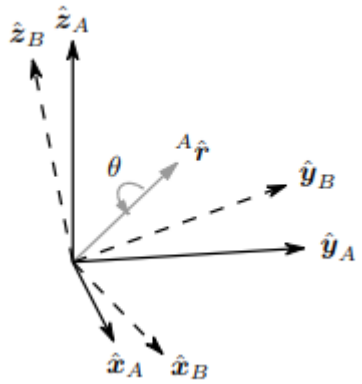


Ilustración 60: Representación gráfica de los marcos A y B

El cuaternión que describe la rotación ${}^A_B\hat{q}$, se define en la ecuación:

$$ {}^A_B\hat{q} = [q_1 \quad q_2 \quad q_3 \quad q_4] = \left[cos\frac{\theta}{2} \quad -r_x sin\frac{\theta}{2} \quad -r_y sin\frac{\theta}{2} \quad -r_z sin\frac{\theta}{2}\right] $$

Donde r_X, r_y, r_z definen los componentes del vector unitario A_r en x, y, z respectivamente sobre el marco A. El sistema de superíndice y subíndice utilizado se usa para indicar el marco de referencia como superíndice y el subíndice como el marco que está siendo descrito.

Por ejemplo ${}^A_B\hat{q}$ describe la orientación del marco B relativo al marco A y A_r es un vector descrito en el marco A. La aritmética de cuaterniones generalmente requiere que el cuaternión que describe una orientación sea antes normalizado. Por lo tanto, es lo convencional que todos los cuaterniones tengan longitud unidad.

El conjugado de un cuaternión, indicado por "*", se puede usar para intercambiar marcos relativos descritos por una orientación. Por ejemplo, ${}^{B}_{A}\hat{q}$ es el conjugado de ${}^{A}_{B}\hat{q}$ y describe la orientación de A relativa al marco B.

$$\widehat{{}^{A}_{B}q^{*}} = {}^{B}_{A}\hat{q} = [q_1 \quad -q_2 \quad -q_3 \quad -q_4]$$

El producto de cuaterniones, indicado por ⊗, se puede usar para definir orientaciones compuestas. Por ejemplo, para dos orientaciones descritas por ${}^{A}_{B}\hat{q}$ y ${}^{B}_{C}\hat{q}$, la orientación compuesta ${}^{A}_{C}\hat{q}$ se define como:

$${}^{A}_{C}\hat{q} = {}^{B}_{C}\hat{q} \otimes {}^{A}_{B}\hat{q}$$

Para dos cuaterniones a y b, el producto de cuaternión se puede determinar usando la regla de Hamilton y se define como se puede ver en la siguiente ecuación. Un producto de cuaternión no es conmutativo, por lo que a ⊗ b no es igual que b ⊗ a.

$$a \otimes \mathrm{b} = [a_1 \quad a_2 \quad a_3 \quad a_4] \otimes [b_1 \quad b_2 \quad b_3 \quad b_4] =$$

$$= \begin{bmatrix} a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4 \\ a_1b_2 + a_2b_1 + a_3b_4 - a_4b_3 \\ a_1b_3 - a_2b_4 + a_3b_1 + a_4b_2 \\ a_1b_4 + a_2b_3 - a_3b_2 + a_4b_1 \end{bmatrix}^{T}$$

Un vector tridimensional puede ser rotado por un cuaternión utilizando la relación descrita en la siguiente ecuación. ${}^{A}v$ y ${}^{B}v$ son el mismo vector descrito en el marco A y en el marco B respectivamente, donde cada vector contiene un cero como primer elemento para convertirlo en un vector de 4 filas.

$${}^{B}v = {}^{A}_{B}\hat{q} \otimes {}^{A}v \otimes {}^{A}_{B}\hat{q}$$

La orientación descrita por ${}^{A}_{B}\hat{q}$ se puede representar como una matriz de rotación ${}^{A}_{B}R$ definida por:

$${}^{A}_{B}R = \begin{bmatrix} 2{q_1}^2 - 1 + 2{q_2}^2 & 2(q_2q_3 + q_1q_4) & 2(q_2q_4 - q_1q_3) \\ 2(q_2q_3 - q_1q_4) & 2{q_1}^2 - 1 + 2{q_3}^2 & 2(q_3q_4 + q_1q_2) \\ 2(q_2q_4 + q_1q_3) & 2(q_3q_4 - q_1q_2) & 2{q_1}^2 - 1 + 2{q_4}^2 \end{bmatrix}$$

Los ángulos Ψ, θ, Φ en la llamada secuencia aeroespacial describen la orientación del marco B por las rotaciones secuenciales, en alineación con el marco A, de Ψ alrededor de z_B, θ sobre y_B y Φ sobre x_B. La representación por ángulos de Euler de ${}^{A}_{B}\hat{q}$ se define por:

$$\Psi = \mathrm{atan2}\,(2q_2q_3 - q_1q_4, 2{q_1}^2 - 1 + 2{q_2}^2)$$

$$\theta = -sin^{-1}(2q_2q_4 + 2q_1q_3)$$

$$\Phi = \mathrm{atan2}\,(2q_3q_4 - q_1q_2, 2{q_1}^2 - 1 + 2{q_2}^2)$$

8.2.2 Orientación por ratio angular

Un giróscopo de 3 ejes mide el ratio angular sobre los ejes x, y, z nombrados ω_x, ω_y, ω_z respectivamente. Si estos parámetros (en radianes^{-1}) se distribuyen en el vector ${}^{s}\omega$, la derivada del cuaternión que describe el ratio de cambio de orientación del marco terrestre relativo al marco del sensor $\dot{{}^{S}_{E}q}$ se puede calcular:

$$ {}^{S}\omega = \begin{bmatrix} 0 & \omega_x & \omega_y & \omega_z \end{bmatrix} $$

$$ \dot{{}^{S}_{E}q} = \frac{1}{2} {}^{S}_{E}\hat{q} \otimes {}^{S}\omega $$

La orientación del marco terrestre relativo al marco del sensor en el tiempo t, ${}^{S}_{E}q_{\omega,t}$, puede ser computado integrando numéricamente la derivada de cuaternión $\dot{{}^{S}_{E}q_{\omega,t}}$ siempre que las condiciones iniciales sean conocidas. En estas ecuaciones, ${}^{s}\omega_t$ es el ratio angular medido en el instante t, Δt es el periodo de muestreo y ${}^{S}_{E}\hat{q}_{est,t-1}$ es la estimación de la orientación anterior. El subíndice ω indica que el cuaternión es calculado a partir de ratios angulares.

$$ \dot{{}^{S}_{E}q_{\omega,t}} = \frac{1}{2} {}^{S}_{E}\hat{q}_{est,t-1} \otimes {}^{S}\omega_t $$

$$ {}^{S}_{E}q_{\omega,t} = {}^{S}_{E}\hat{q}_{est,t-1} + \dot{{}^{S}_{E}q_{\omega,t}}\Delta t $$

8.2.3 Orientación por observación vectorial

Un acelerómetro de 3 ejes mide la magnitud y dirección del campo gravitatorio en el marco del sensor compuesto por aceleraciones lineales. De forma parecida un magnetómetro de 3 ejes mide la dirección y magnitud del campo magnético de la tierra respecto al marco del sensor compuesto por el flujo magnético local y distorsiones. En el contexto de un filtro de orientación se asume que un acelerómetro sólo medirá la gravedad y un magnetómetro sólo medirá el campo magnético terrestre.

Si la dirección de un campo terrestre es conocida en el marco terrestre, una medida de la dirección del campo en el marco del sensor permitirá el cálculo de una orientación del marco del sensor relativa al campo terrestre. Sin embargo, para cualquier medida dada no habrá una única solución de orientación, en su lugar habrá infinitas soluciones representadas por todas las orientaciones conseguidas debidas a la rotación de la auténtica orientación alrededor de un eje paralelo al campo.

Una representación vía cuaterniones requiere encontrar una solución completa. Esto se puede conseguir a través de la formulación de un problema de optimización, donde la orientación de un sensor ${}^{S}_{E}\hat{q}$ es alineada con una dirección predefinida al campo del marco terrestre ^{E}d, con la dirección del marco del sensor ^{S}s, utilizando la operación de rotación anteriormente descrita. Por lo tanto ${}^{S}_{E}\hat{q}$ se puede encontrar como la solución a:

$$\min_{{}^{S}_{E}\hat{\boldsymbol{q}}\in\Re^4} \boldsymbol{f}({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{d}}, {}^{S}\hat{\boldsymbol{s}})$$

Donde la ecuación siguiente define la función objeto:

$$f({}^{S}_{E}\hat{q}, {}^{E}\hat{d}, {}^{S}\hat{s}) = {}^{S}_{E}\widehat{q^*} \otimes {}^{E}\hat{d} \otimes {}^{S}_{E}\hat{q} - {}^{S}\hat{s}$$

Los componentes de cada vector se describen como:

$${}^{S}_{E}\hat{q} = [q_1 \quad q_2 \quad q_3 \quad q_4]$$
$${}^{E}\hat{d} = [0 \quad d_x \quad d_y \quad d_z]$$
$${}^{S}\hat{s} = [0 \quad s_x \quad s_y \quad s_z]$$

Existen varios algoritmos de optimización pero el algoritmo de gradiente descendente es uno de los más simples de implementar y computar. La siguiente ecuación describe el algoritmo para "n" iteraciones en una estimación de la orientación de ${}^{S}_{E}\hat{q}_{n+1}$ y un tamaño de paso μ.

$${}^{S}_{E}\hat{q}_{n+1} = {}^{S}_{E}\hat{q}_{k} - \mu\frac{\nabla f({}^{S}_{E}\widehat{q_k}, {}^{E}\hat{d}, {}^{S}\hat{s})}{\left\|\nabla f({}^{S}_{E}\widehat{q_k}, {}^{E}\hat{d}, {}^{S}\hat{s})\right\|}, \quad k = 0,1,2,\dots,n$$

La siguiente ecuación computa el gradiente de la solución de superficie definida por la función objetivo y su Jacobiano.

$$\nabla f({}^{S}_{E}\widehat{q_k}, {}^{E}\hat{d}, {}^{S}\hat{s}) = \mathrm{J}^T({}^{S}_{E}\widehat{q_k}, {}^{E}\hat{d}) f({}^{S}_{E}\widehat{q_k}, {}^{E}\hat{d}, {}^{S}\hat{s})$$

$$f({}^{S}_{E}\widehat{q_k}, {}^{E}\hat{d}, {}^{S}\hat{s}) = \begin{bmatrix} 2dx\left(\frac{1}{2} - {q_3}^2 - {q_4}^2\right) + 2dy(q_1q_4 + q_2q_3) + 2dz(q_2q_4 - q_1q_3) - 5x \\ 2dx(q_2q_3 - q_1q_4) + 2dy\left(\frac{1}{2} - {q_2}^2 - {q_4}^2\right) + 2dz(q_1q_2 + q_3q_4) - 5y \\ 2dx(q_1q_3 + q_2q_4) + 2dy(q_3q_4 - q_1q_2) + 2dz\left(\frac{1}{2} - {q_2}^2 - {q_3}^2\right) - 5z \end{bmatrix}$$

$$\mathrm{J}({}^{S}_{E}\widehat{q_k}, {}^{E}\hat{d}) = \begin{bmatrix} 2dyq_4 - 2dzq_3 & 2dyq_3 + 2dzq_4 & -4dxq_3 + 2dyq_2 - 2dzq_1 & -4dxq_4 + 2dyq_1 + 2dzq_2 \\ -2dxq_4 + 2dzq_2 & 2dxq_3 - 4dyq_2 + 2dzq_1 & 2dxq_2 + 2dzq_4 & -2dxq_1 - 4dyq_4 + 2dzq_3 \\ 2dxq_3 - 2dyq_2 & 2dxq_4 - 2dyq_1 - 4dzq_2 & 2dxq_1 + 2dyq_4 - 4dzq_3 & 2dxq_2 + 2dyq_3 \end{bmatrix}$$

Estas ecuaciones describen de forma general el algoritmo aplicable a un campo predefinido en cualquier dirección. Sin embargo, si es posible asumir que la dirección del campo sólo tiene componentes en 1 ó 2 de los ejes principales del marco de orientación global, las ecuaciones se simplifican.

Asumiendo que la gravedad se define en el eje vertical, el eje z, obtenemos esta ecuación:

$$^{E}\hat{g} = [0 \quad 0 \quad 0 \quad 1]$$

Sustituyendo $^{E}\hat{g}$ y normalizando la medida de acelerómetro $^{S}\hat{a}$ para $^{E}\hat{d}$ y $^{S}\hat{s}$ respectivamente en las ecuaciones anteriores:

$$^{S}\hat{a} = \begin{bmatrix} 0 & a_x & a_y & a_z \end{bmatrix}$$

$$f(\widehat{{}^{S}_{E}q_k}, {}^{S}\hat{a}) = \begin{bmatrix} 2(q_2q_4 - q_1q_3) - a_x \\ 2(q_1q_2 + q_3q_4) - a_y \\ 2\left(\frac{1}{2} - {q_2}^2 - {q_3}^2\right) - a_z \end{bmatrix}$$

$$\mathrm{J}(\widehat{{}^{S}_{E}q}) = \begin{bmatrix} -2q_3 & 2q_4 & -2q_1 & 2q_2 \\ 2q_2 & 2q_1 & 2q_4 & 2q_3 \\ 0 & -4q_2 & -4q_3 & 0 \end{bmatrix}$$

El campo magnético terrestre se puede considerar que tiene componentes en un eje horizontal y el eje vertical. Esto se puede representar como:

$$^{E}\hat{b} = [0 \quad b_x \quad 0 \quad b_z]$$

Sustituyendo $^{E}\hat{d}$ por $^{E}\hat{b}$ y la medida del magnetómetro normalizada $^{S}\widehat{m}$ en $^{S}\hat{s}$:

$$^{S}\widehat{m} = \begin{bmatrix} 0 & m_x & m_y & m_z \end{bmatrix}$$

$$f\left(\widehat{{}^{S}_{E}q_k}, {}^{E}\hat{b}, {}^{S}\widehat{m}\right) = \begin{bmatrix} 2b_x\left(\frac{1}{2} - {q_3}^2 - {q_4}^2\right) + 2b_z(q_2q_4 - q_1q_3) - m_x \\ 2b_x(q_2q_3 - q_1q_4) + 2b_z(q_1q_2 + q_3q_4) - m_y \\ 2b_x(q_1q_3 + q_2q_4) + 2b_z\left(\frac{1}{2} - {q_2}^2 - {q_3}^2\right) - m_z \end{bmatrix}$$

$$\mathrm{J}\left(\widehat{{}^{S}_{E}q_k}, {}^{E}\hat{b}\right) = \begin{bmatrix} -2b_zq_3 & 2b_zq_4 & -4b_xq_3 - 2b_zq_1 & -4b_xq_4 + 2b_zq_2 \\ -2b_xq_4 + 2b_zq_2 & 2b_xq_3 + 2b_zq_1 & 2b_xq_2 + 2b_zq_4 & -2b_xq_1 + 2b_zq_3 \\ 2b_xq_3 & 2b_xq_4 - 4b_zq_2 & 2b_xq_1 - 4b_zq_3 & 2b_xq_2 \end{bmatrix}$$

La medición de la gravedad o el campo magnético terrestre no provee una solución única de la orientación del sensor. Por lo tanto, las medidas y referencias de ambos campos podrían combinarse de la siguiente manera:

$$f_{g,b}\left(\widehat{{}^{S}_{E}q_k}, {}^{S}\hat{a}, {}^{E}\hat{b}, {}^{S}\widehat{m}\right) = \begin{bmatrix} f_g\left(\widehat{{}^{S}_{E}q_k}, {}^{S}\hat{a}\right) \\ f_b\left(\widehat{{}^{S}_{E}q_k}, {}^{E}\hat{b}, {}^{S}\widehat{m}\right) \end{bmatrix}$$

$$\mathrm{J}_{g,b}\left(\widehat{{}^{S}_{E}q}, {}^{E}\hat{b}\right) = \begin{bmatrix} \mathrm{J}_g^T\left(\widehat{{}^{S}_{E}q}\right) \\ \mathrm{J}_b^T\left(\widehat{{}^{S}_{E}q}, {}^{E}\hat{b}\right) \end{bmatrix}$$

Donde la solución de superficie por la función objetivo de las anteriores ecuaciones tiene un mínimo definido por una línea, la nueva ecuación $f_{g,b}$ tiene un mínimo definido por un punto siempre que b_x sea distinto de cero.

La siguiente ecuación calcula la orientación estimada ${}^{S}_{E}q_{\nabla,t}$ computada en tiempo t basada en una estimación de orientación ${}^{S}_{E}\hat{q}_{\text{est,t}-1}$ y el gradiente función objetivo ∇f definido por las mediciones de sensor ${}^{S}\hat{a}_t$ y ${}^{S}\hat{m}_t$ muestreadas en tiempo t.

La forma de ∇f se escoge de acuerdo a los sensores en uso, según la segunda ecuación. El subíndice ∇ indica que el cuaternión se ha calculado usando el algoritmo de gradiente descendente.

$$ {}^{S}_{E}q_{\nabla,t} = {}^{S}_{E}\hat{q}_{\text{est,t}-1} - \mu_t \frac{\nabla f}{\|\nabla f\|} $$

$$ \nabla f = \begin{cases} J_g^T\left({}^{S}_{E}\hat{q}_{\text{est,t}-1}\right) f_g\left({}^{S}_{E}\hat{q}_{\text{est,t}-1}, {}^{S}\hat{a}\right) \\ J_{g,b}\left({}^{S}_{E}\hat{q}_{\text{est,t}-1}, {}^{E}\hat{b}\right) f_{g,b}\left({}^{S}_{E}\hat{q}_{\text{est,t}-1}, {}^{S}\hat{a}, {}^{E}\hat{b}, {}^{S}\hat{m}\right) \end{cases} $$

Se puede definir un valor óptimo de μ_t que asegure que el ratio de convergencia de ${}^{S}_{E}q_{\nabla,t}$ es limitado al ratio de orientación física, puesto que esto evita excederse en las estimaciones debido a un innecesario tamaño de paso. Por lo tanto μ_t puede ser calculado por la siguiente ecuación, donde Δt es el periodo de muestreo y ${}^{S}_{E}\dot{q}_{\omega,\text{t}}$ es el ratio de orientación física medida por giróscopos y α es una aumentación de μ para tener en cuenta el ruido en las medidas de acelerómetro y magnetómetro.

$$ \mu_t = \alpha \left\| {}^{S}_{E}\dot{q}_{\omega,\text{t}} \right\| \Delta\text{t}, \quad \alpha > 1 $$

8.2.4 Algoritmo de fusión de filtros

Una orientación estimada del marco sensor relativa al marco terrestre, ${}^{S}_{E}\hat{q}_{\text{est,t}}$, se obtiene de la fusión de los cálculos de orientación, ${}^{S}_{E}q_{\omega,\text{t}}$ y ${}^{S}_{E}q_{\nabla,t}$. Se describe la fusión ${}^{S}_{E}q_{\omega,\text{t}}$ y ${}^{S}_{E}q_{\nabla,t}$ en la siguiente ecuación donde γ_t y $(1-\gamma_t)$ son el "peso" aplicado a cada cálculo de orientación.

$$ {}^{S}_{E}\hat{q}_{\text{est,t}} = \gamma_t {}^{S}_{E}q_{\nabla,t} + (1-\gamma_t) {}^{S}_{E}q_{\omega,\text{t}}, \quad 0 \le \gamma_t \le 1 $$

Un valor óptimo de γ_t se define como uno que asegure la divergencia de ${}^{S}_{E}q_{\omega}$ igual a la convergencia de ${}^{S}_{E}q_{\nabla}$. Esto se representa en la siguiente ecuación, donde $\frac{\mu_t}{\Delta\text{t}}$ es el ratio de convergencia de ${}^{S}_{E}q_{\nabla}$ y β es el ratio de divergencia de ${}^{S}_{E}q_{\omega}$ expresado como la magnitud de una derivada de cuaternión correspondiente al error de medición del giróscopo.

$$ (1-\gamma_t)\beta = \gamma_t \frac{\mu_t}{\Delta\text{t}} \quad \rightarrow \gamma_t = \frac{\beta}{\frac{\mu_t}{\Delta\text{t}} + \beta} $$

Las ecuaciones anteriores aseguran la fusión óptima de ${}^{S}_{E}q_{\omega,\text{t}}$ y ${}^{S}_{E}q_{\nabla,t}$ asumiendo que el ratio de convergencia de ${}^{S}_{E}q_{\nabla}$ gobernado por α es igual o mayor que el ratio de cambio físico de la orientación. Si se asume α muy grande, entonces μ_t, definido anteriormente, también se vuelve muy grande y las ecuaciones de filtro de la orientación se simplifican. Un valor grande de μ_t implica que ${}^{S}_{E}\hat{q}_{\text{est,t}-1}$ se vuelve despreciable y la ecuación se puede reescribir como:

$$ {}^{S}_{E}q_{\nabla,t} \approx -\mu_t \frac{\nabla f}{\|\nabla f\|} $$

La definición de γ_t también se ve simplificada, puesto que el término β en el denominador se vuelve despreciable y se puede reescribir la ecuación como:

$$\gamma_t \approx \frac{\beta \Delta t}{\mu_t}$$

Sustituyendo, se consigue la siguiente ecuación:

$${}^{S}_{E}q_{est,t} = \frac{\beta \Delta t}{\mu_t}\left(-\mu_t \frac{\nabla f}{\|\nabla f\|}\right) + (1-0)\left({}^{S}_{E}\hat{q}_{est,t} + {}^{S}_{E}q_{\omega,t}\right)$$

La ecuación anterior se puede simplificar como:

$${}^{S}_{E}q_{est,t} = {}^{S}_{E}\hat{q}_{est,t-1} + {}^{S}_{E}\dot{q}_{est,t}\Delta t$$

Donde

$${}^{S}_{E}\dot{q}_{est,t} = {}^{S}_{E}\dot{q}_{\omega,t} + \beta\, {}^{S}_{E}\dot{q}_{\epsilon,t}$$

$${}^{S}_{E}\dot{q}_{\epsilon,t} = \frac{\nabla f}{\|\nabla f\|}$$

El filtro calcula la orientación ${}^{S}_{E}q_{est}$ integrando numéricamente el ratio estimado de orientación ${}^{S}_{E}\dot{q}_{est}$. El filtro computa ${}^{S}_{E}\dot{q}_{est}$ como el ratio de cambio de orientación medido por los giróscopos, ${}^{S}_{E}\dot{q}_{\omega}$, con la magnitud de error de medida del giróscopo, β, eliminando en la dirección del error estimado, ${}^{S}_{E}\hat{q}_{\epsilon}$, computado de las medidas de acelerómetros y magnetómetros.

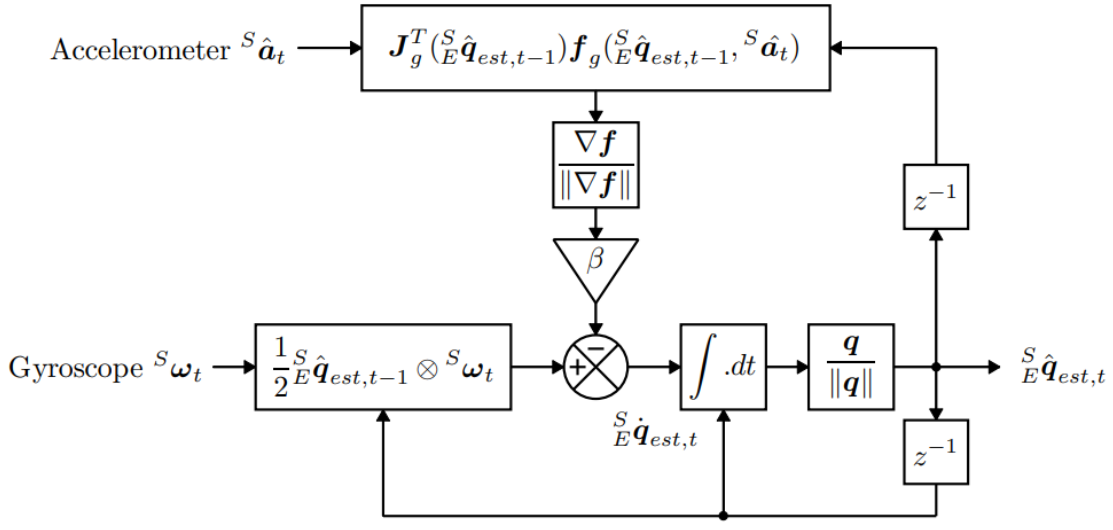


Ilustración 61: Diagrama de bloques de la implementación del filtro para una IMU

8.2.5 Compensación de la distorsión magnética

Las mediciones del campo magnético terrestre pueden verse distorsionadas por la presencia de elementos ferromagnéticos en las cercanías del magnetómetro. Investigaciones en el efecto de distorsiones magnéticas en la eficacia del sensor de orientación muestran errores substanciales que pueden haber sido producidos por aplicaciones eléctricas, mobiliario metálico y estructuras metálicas en la construcción de edificios. Errores de declinación, relativos al plano horizontal de la superficie terrestre, no pueden ser corregidos sin referencias adicionales de dirección. Errores de inclinación, relativos al plano vertical de la superficie terrestre, pueden ser compensados, puesto que el acelerómetro ofrece una medida adicional de la orientación del sensor.

La dirección medida del campo magnético terrestre en el marco de la tierra en tiempo t, ${}^{E}\hat{h}_t$, puede ser computada como la medida normalizada del magnetómetro, ${}^{S}\hat{m}_t$, rotado por la orientación estimada del sensor obtenido del filtro, ${}^{S}_{E}\hat{q}_{\text{est,t}-1}$:

$${}^{E}\hat{h}_t = \begin{bmatrix} 0 & h_x & h_y & h_z \end{bmatrix} = {}^{S}_{E}\hat{q}_{\text{est,t}-1} \otimes {}^{S}\hat{m}_t \otimes {}^{S}_{E}\widehat{q^*}_{\text{est,t}-1}$$

El efecto de un error en la medición de la orientación del campo magnético terrestre, ${}^{E}\hat{h}_t$, puede ser corregido si la referencia de la dirección del campo magnético terrestre, ${}^{E}\hat{b}_t$, tiene la misma inclinación. Esto se consigue computando ${}^{E}\hat{b}_t$ como ${}^{E}\hat{h}_t$ normalizado para obtener componentes sólo en los ejes x y z del marco terrestre:

$${}^{E}\hat{b}_t = \begin{bmatrix} 0 & \sqrt{{h_x}^2 + {h_y}^2} & 0 & h_z \end{bmatrix}$$

Compensar las distorsiones magnéticas de esta manera asegura que las perturbaciones se encuentran limitadas y sólo afectan al componente de dirección estimada de la orientación. También se elimina la necesidad de una orientación de referencia predefinida del campo magnético terrestre, una desventaja potencial al usar otros filtros de orientación.

8.2.6 Compensación de la deriva giroscópica

El cero giroscópico se verá desplazado debido al tiempo, temperatura y movimiento. Cualquier implementación práctica de un conjunto sensor IMU o MARG debe tenerlo en cuenta. Una ventaja de las soluciones basadas en Kalman es que puede estimar el bias giroscópico como un estado más dentro del modelo del sistema. Sin embargo, Mahony "et al" demostró que la deriva giroscópica puede ser compensada por filtros de orientación más sencillos a través de realimentación integral del error en el ratio de cambio de la orientación. Aquí se utiliza un método similar.

La dirección normalizada del error estimado del ratio de cambio de orientación, ${}^{S}_{E}\dot{q}_{\epsilon}$, se puede expresar como el error angular en cada eje del giróscopo:

$${}^{S}_{E}q_{\omega,\text{t}} = 2\,{}^{S}_{E}\widehat{q^*}_{\text{est,t}-1} \otimes {}^{S}_{E}\dot{q}_{\epsilon,\text{t}}$$

El bias del giróscopo, ${}^{S}_{E}\dot{q}_{\epsilon}$, se representa como el componente DC de ${}^{S}\omega_{\epsilon}$ y se puede eliminar de la integral de ${}^{S}\omega_{\epsilon}$ ajustada con la ganancia apropiada ξ. El primer elemento de ${}^{S}\omega_{c}$ se asume que siempre es cero.

$$^{S}\omega_{b,t} = \xi \sum_{t} {}^{S}\omega_{\epsilon,t}\Delta t$$

$$^{S}\omega_{c,t} = {}^{S}\omega_{t} - {}^{S}\omega_{b,t}$$

Las medidas giroscópicas compensadas, $^{S}\omega_{c}$, pueden ser utilizadas en lugar de las medidas giroscópicas, $^{S}\omega$. La magnitud del error angular en cada eje, $^{S}\omega_{\epsilon}$ es igual a la derivada cuaternión unidad. Por lo tanto, la ganancia integral ξ define de forma directa el ratio de convergencia del bias giroscópico estimado, $^{S}\omega_{b}$, expresado como la magnitud de la derivada de un cuaternión. Como este proceso requiere el uso de la estimación completa de orientación, $^{S}_{E}\hat{q}_{\mathrm{est,t}}$, sólo es posible aplicarlo en filtro para unidad MARG.

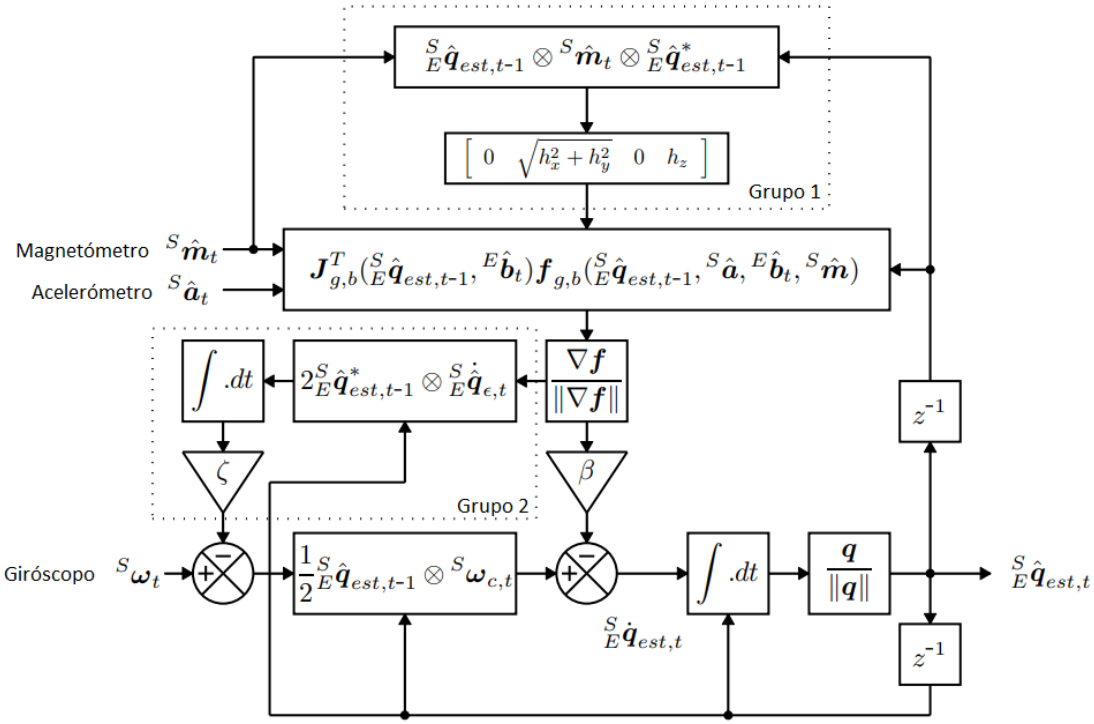


Ilustración 62: Diagrama de bloques del filtro de orientación MARG incluyendo compensación por distorsión magnética (grupo 1) y compensación de la deriva giroscópica (grupo 2)

8.3 XSens MTi-7

El sistema AHRS investigado en este caso es MTi 7 de XSens. Utiliza un filtro Kalman para determinar la orientación a partir de la información recibida en acelerómetros, giróscopos y magnetómetros. También dispone de conexión a chip GNSS externo para recibir información de la posición y velocidad e incluirla en la trama de datos que facilita para el microcontrolador que se desee usar en la aviónica.

Se ha decidido utilizar comunicación SPI para determinar la viabilidad de usar este chipset instrumental. Para recibir la trama de datos vía SPI hay que tener en cuenta que:

Para permitir comunicación SPI el firmware del MTi 1-series debe ser superior a 1.0.3 de lo contrario no es posible actualizar el firmware a una versión que permita este tipo de comunicación según dice el datasheet de la serie.

8.3.1 Procedimiento de envío SPI (MTi 1 – series)

El maestro empieza la transferencia poniendo a nivel bajo *SPI_nCS*, para seleccionar el "esclavo". El maestro debe mantener la línea *SPI_nCS* baja durante la transferencia. El esclavo interpretará el flanco ascendente del pin *SPI_nCS* como el fin de la transferencia. El maestro coloca los datos que quiere transmitir en la línea *SPI_MOSI*. El esclavo pone la información en *SPI_MISO*.

El maestro primero transmite el "*opcode*". El "*opcode*" determina qué clase de dato transmite el maestro, y qué clase de datos éste quiere leer del esclavo. Del segundo al cuarto bytes enviados por el maestro durante la comunicación SPI son "bytes de relleno". Estos bytes son necesarios para darle tiempo al esclavo a seleccionar la información que tiene que enviar.

Tanto maestro como esclavo son libres de seleecionar el valor de las palabras de relleno y el receptor deberá ignorarlo. Sin embargo, los primeros 4 bytes enviados por el módulo MTi 1-series son siempre: 0xFA, 0xFF, 0xFF, 0xFF.

Los siguientes contienen la información que interesa. Es responsabilidad del maestro determinar cuántos bytes se van a transmitir (hace falta saber de antemano cuántos enviará el esclavo). Para leer mensajes de *Notification_Pipe* y *Measurement_Pipe*, el número de bytes que el maestro debe leer depende del tamaño del mensaje que el esclavo enviará.

Para determinar el número correcto de bytes, el maestro puede leer primero "*Pipe Status*" y así obtener el tamaño de los mensajes.

8.3.2 MTSSP Protocolo serial síncrono

8.3.2.1 Flujo de datos

La comunicación MTSSP se efectúa de acuerdo al modelo maestro-esclavo. El maestro siempre inicia y administra la comunicación. El módulo MTi 1-series siempre tendrá rol de esclavo, mientras que el usuario del módulo será siempre el maestro. El maestro puede enviar mensajes al módulo o recibir mensajes del esclavo.

Recordar que para algunos OPCODES, el protocolo MTSSP utiliza mensajes *xbus* reducidos. El cálculo del checksum de un mensaje *xbus* reducido incluye el BiD (que no se envía en los mensajes *xbus* reducidos) y se supone que es 0xFF.

El mensaje *xbus* reducido queda tal que (en orden de recepción):

1. MID (1 byte) --> Identificativo de mensaje.
2. LEN (1 byte) --> Longitud en bytes de los datos.
3. DataID (2 bytes) --> Identificativo del dato enviado.
4. Data_LEN (1 byte) --> Longitud del dato enviado (referente al DataID)
5. Datos (Data_LEN bytes) --> Datos enviados

[...] --> (Puede enviar más datos y se repetiría desde el punto 3)

6. CheckSum (1 byte) --> CheckSum del mensaje suponiendo el *BusID* = 0xFF.

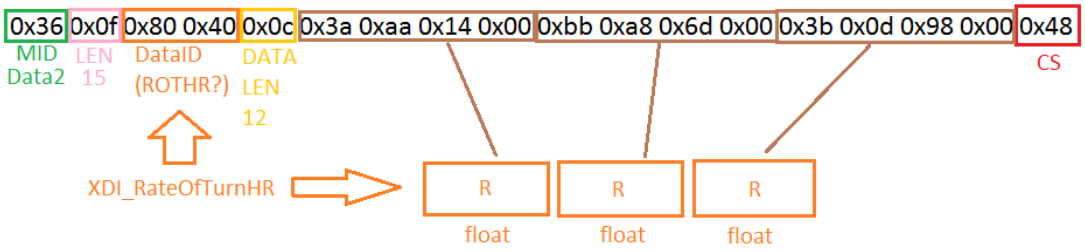


Ilustración 63: Estructura de la trama SPI de MTi 1 - series

8.3.2.2 Opcode

El primer byte que llega vía SPI es el "*opcode*". Según el *opcode* el formato del mensaje varía. Por lo tanto, se cataloga el mensaje primero según el *opcode*:

- OPCODE 0x01 --> Lectura --> Definido por *opcode*
- OPCODE 0x02 --> Escritura --> Definido por *opcode*
- OPCODE 0x03 --> Escritura --> *xbus* reducido
- OPCODE 0x04 --> Lectura --> Definido por *opcode*
- OPCODE 0x05 --> Lectura --> *xbus* reducido
- OPCODE 0x06 --> Lectura --> *xbus* reducido

8.3.2.3 Configuración SPI

Modo 4 - hilos SPI:

- Chip Select / Slave Select (SPI_nCS)
- Serial Clock (SPI_SCK)
- Master data In, Slave data Out (SPI_MISO)
- Master data Out, Slave data In (SPI_MOSI)

El módulo utiliza el modo 3 de SPI:

Captura datos en el flanco ascendente de reloj y propaga los datos en el flanco descendente. (CPOL = 1 y CPHA=1). La información se envía MSB primero. El módulo utiliza un formato de 8 bits.

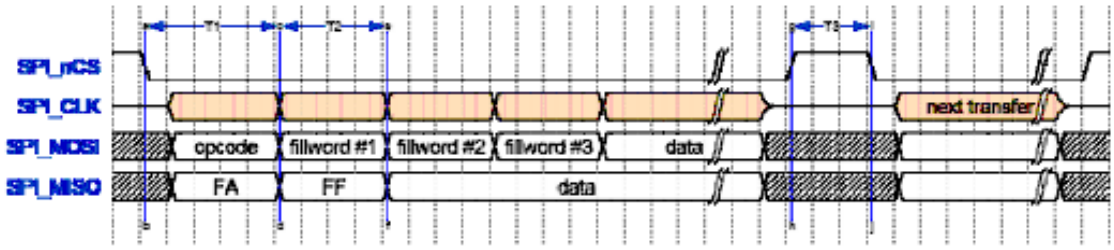


Ilustración 64: Funcionamiento SPI de MTi 1 - series

- **T1:** Tiempo entre flanco descendente de Slave Select y primer paquete enviado: Mín 4 us
- **T2:** Tiempo de envío de un byte: Mín 4 us
- **T3:** Tiempo para envío consecutivo: Mín 3 us
- **Bitrate máximo del SPI:** 2 Mbits/s

8.3.2.4 Funciones de la librería desarrollada

xsens_SPI_init(struct spi_device *mti_spi)

Primero reinicia la MTi 1-series accionando el pin reset durante un tiempo determinado. Realiza una comunicación con la IMU vía SPI en la que se envía el OPCODE "PIPE_STATUS" de modo que devuelva 2 uint16 que se leen, pero que por ahora quedan en el olvido. Lo ideal sería esperar recibir confirmación de que la IMU se ha reiniciado (posible mejora). Una vez se ha terminado de recibir la información esperada en el bus SPI la función termina. Necesita como variables:

- Puntero a la estructura del *spi_device* que vaya a usar.

xsens_SPI_actualizarMedidas(struct spi_device *mti_spi)

Llama a la función "xbus_spi_read()" y le da información del bus *spi_device* que se está utilizando y el pin *SS* que necesitará para realizar su función. Es posible ampliar más adelante esta función para que se encargue de actualizar estructuras de datos donde se almacene la información de los sensores. Necesita como variables:

- Puntero a la estructura del *spi_device* que vaya a usar.

xbus_spi_read(struct spi_device *mti_spi)

Realiza llamadas a las funciones "*xbus_spi_readPipeStatus()*", "*xbus_spi_readPipeNotif()*", "*xbus_spi_parseData()*" y les da la información que necesitan para funcionar.

xbus_spi_readPipeStatus(struct spi_device *mti_spi)

Solicita vía SPI el tamaño de la infromación preparada para enviar vía SPI por parte de la MTi 1-series, tanto de mensajes de notificación como de mensajes de mediciones. Guarda los valores en los *arrays notificationSize* y *measurementSize*. Necesita como variables:

- Puntero a la estructura del *spi_device* que vaya a usar.

xbus_spi_readPipeNotif(struct spi_device *mti_spi)

Solicita y recibe los mensajes de notificación preparados de la MTi 1-series. Necesita como variables:

- Puntero a la estructura del *spi_device* que vaya a usar.

xbus_spi_readPipeMeas(struct spi_device *mti_spi)

Solicita y recibe los mensajes de mediciones preparados de la MTi 1-series. Necesita como variables:

- Puntero a la estructura del *spi_device* que vaya a usar.

xbus_spi_dataswapendian(uint8_t* data, uint8_t length)

Cambia el orden de los bytes (4 bytes cada vez) del array que se le hayan enviado. Es utilizado por la función "*xbus_spi_parseData*". Necesita como variables:

- Puntero al array de datos (En este caso del dato que se quiere leer, no el array completo)
- Tamaño del dato a leer del array.

xbus_spi_parseData(uint8_t* data, uint8_t datalength)

Realiza el parseo del mensaje de mediciones recibido. La función de momento guarda los datos recibidos en sus variables correspondientes, pero sería más conveniente que estas variables formasen parte de una estructura de datos ya predefinida (que todo esté más ordenado). Necesita como variables:

- Puntero al array de datos (Mensaje leído del *measurement pipe*)
- Tamaño del array de datos (Tamaño del mensaje recibido por *measurement pipe*)

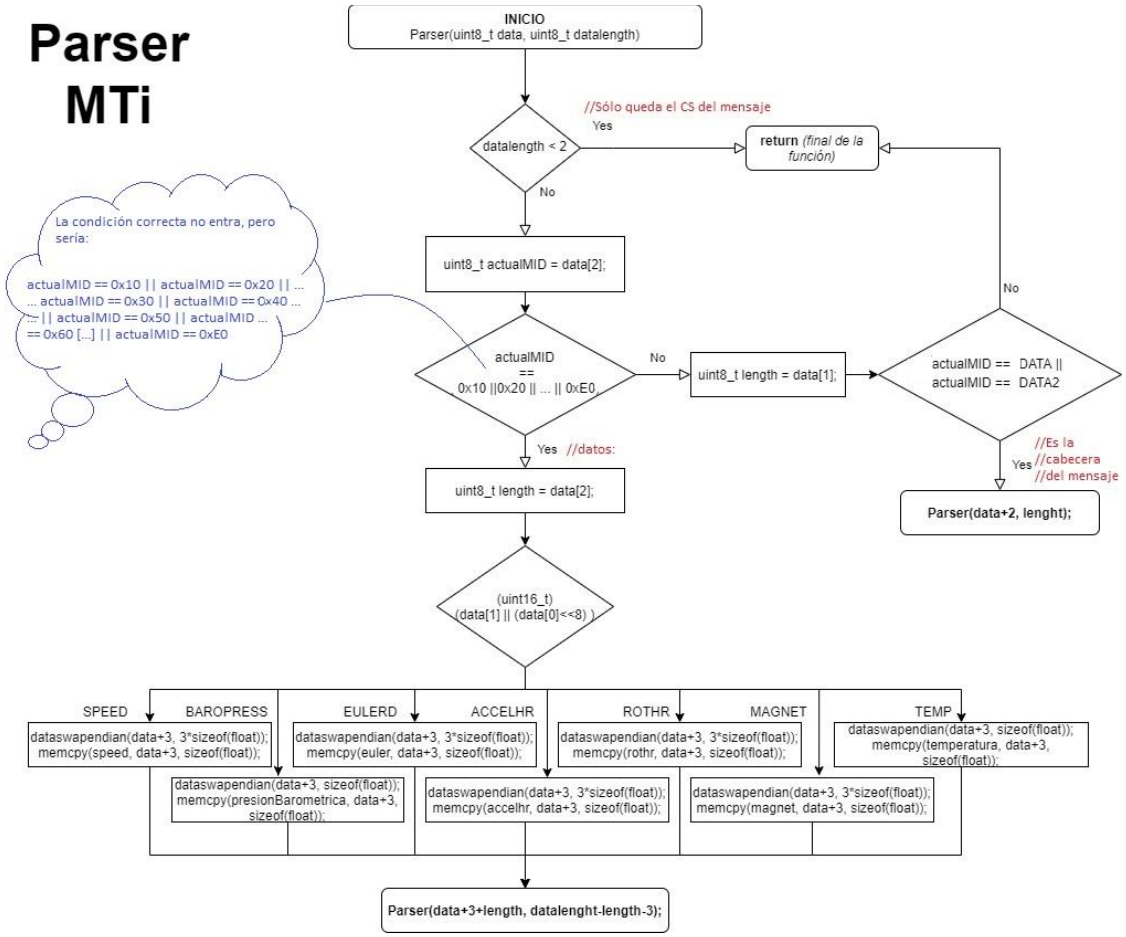


Ilustración 65: Diagrama de flujo del funcionamiento del Parser de la trama SPI

8.4 Microcontrolador ATMEGA2560

8.4.1 Diagrama de bloques del microcontrolador

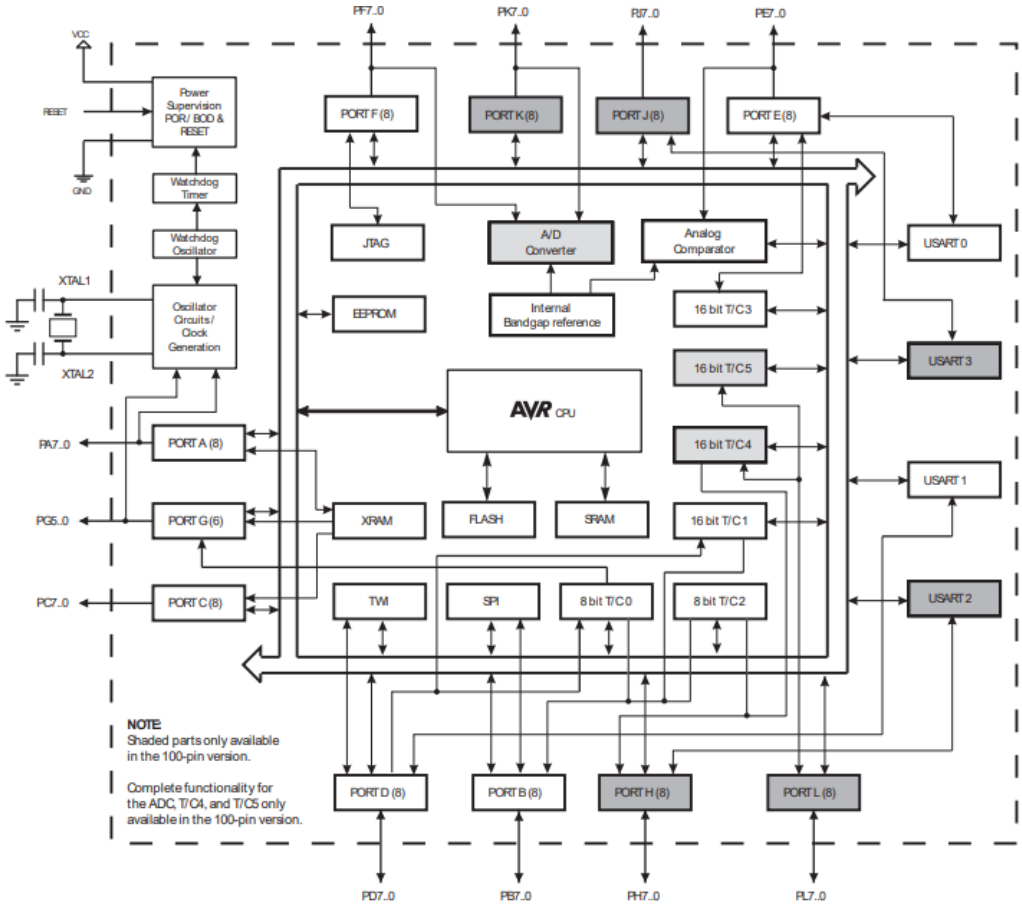
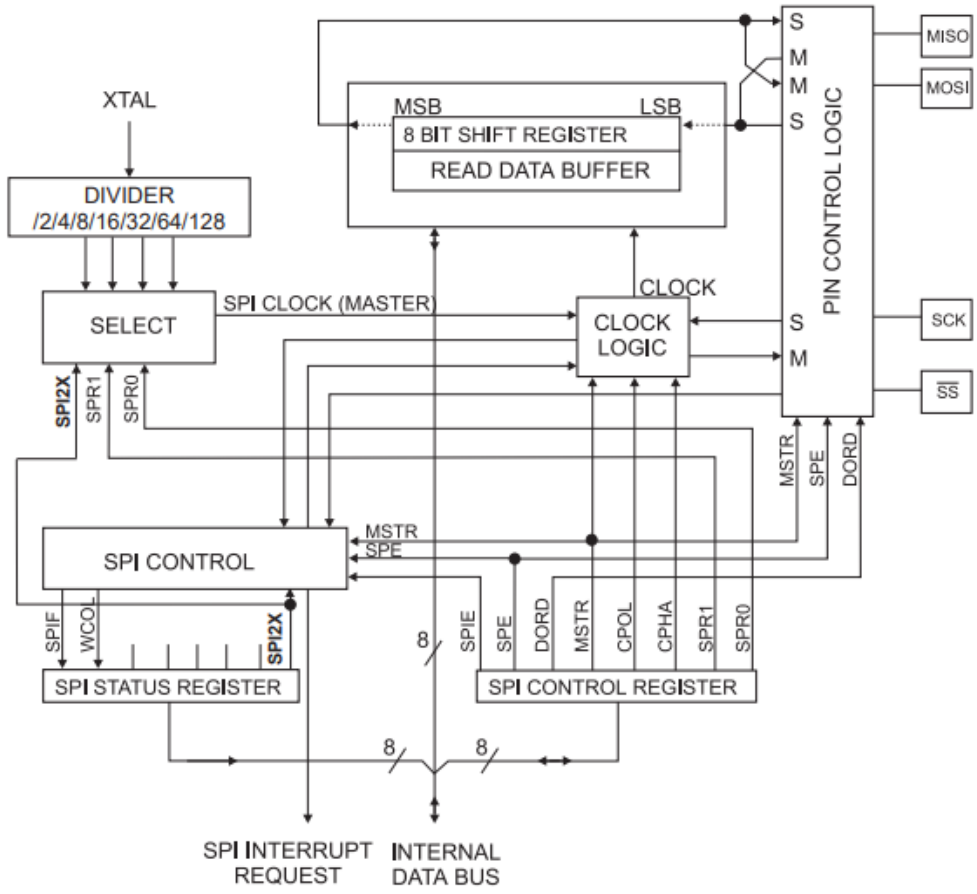


Ilustración 66:Diagrama de bloques del Microcontrolador

8.4.2 Periférico SPI

Figure 21-1. SPI Block Diagram[1]



Note: 1. Refer to Figure 1-1 on page 2, and Table 13-6 on page 76 for SPI pin placement.

Ilustración 67: Diagrama de bloques del periférico SPI del Microcontrolador

Funcionalidad /SS y modos de funcionamiento:

21.1 $\overline{SS}$ Pin Functionality

21.1.1 Slave Mode

When the SPI is configured as a Slave, the Slave Select ($\overline{SS}$) pin is always input. When $\overline{SS}$ is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs. When $\overline{SS}$ is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI logic will be reset once the $\overline{SS}$ pin is driven high.

The $\overline{SS}$ pin is useful for packet/byte synchronization to keep the slave bit counter synchronous with the master clock generator. When the $\overline{SS}$ pin is driven high, the SPI slave will immediately reset the send and receive logic, and drop any partially received data in the Shift Register.

21.1.2 Master Mode

When the SPI is configured as a Master (MSTR in SPCR is set), the user can determine the direction of the $\overline{SS}$ pin.

If $\overline{SS}$ is configured as an output, the pin is a general output pin which does not affect the SPI system. Typically, the pin will be driving the $\overline{SS}$ pin of the SPI Slave.

If $\overline{SS}$ is configured as an input, it must be held high to ensure Master SPI operation. If the $\overline{SS}$ pin is driven low by peripheral circuitry when the SPI is configured as a Master with the $\overline{SS}$ pin defined as an input, the SPI system interprets this as another master selecting the SPI as a slave and starting to send data to it. To avoid bus contention, the SPI system takes the following actions:

1. The MSTR bit in SPCR is cleared and the SPI system becomes a Slave. As a result of the SPI becoming a Slave, the MOSI and SCK pins become inputs.
2. The SPIF Flag in SPSR is set, and if the SPI interrupt is enabled, and the I-bit in SREG is set, the interrupt routine will be executed.

Thus, when interrupt-driven SPI transmission is used in Master mode, and there exists a possibility that $\overline{SS}$ is driven low, the interrupt should always check that the MSTR bit is still set. If the MSTR bit has been cleared by a slave select, it must be set by the user to re-enable SPI Master mode.

21.1.3 Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 21-3 on page 196 and Figure 21-4 on page 196. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. This is clearly seen by summarizing Table 21-3 on page 197 and Table 21-4 on page 197 in Table 21-2.

Table 21-2. CPOL Functionality

	Leading Edge	Trailing Edge	SPI Mode
CPOL=0, CPHA=0	Sample (Rising)	Setup (Falling)	0
CPOL=0, CPHA=1	Setup (Rising)	Sample (Falling)	1
CPOL=1, CPHA=0	Sample (Falling)	Setup (Rising)	2
CPOL=1, CPHA=1	Setup (Falling)	Sample (Rising)	3

Ilustración 68: Funcionalidad del pin /SS y modos de funcionamiento

Registros de control y estado:

21.2 Register Description

21.2.1 SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIE: SPI Interrupt Enable**

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the if the Global Interrupt Enable bit in SREG is set.

- **Bit 6 – SPE: SPI Enable**

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: Master/Slave Select**

This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If $\overline{SS}$ is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.

- **Bit 3 – CPOL: Clock Polarity**

When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. Refer to Figure 21-3 on page 196 and Figure 21-4 on page 196 for an example. The CPOL functionality is summarized in Table 21-3.

Table 21-3. CPOL Functionality

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

- **Bit 2 – CPHA: Clock Phase**

The settings of the Clock Phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to Figure 21-3 on page 196 and Figure 21-4 on page 196 for an example. The CPOL functionality is summarized in Table 21-4.

Table 21-4. CPHA Functionality

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

Ilustración 69: Registros de control de SPI

- **Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0**

These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on the Slave. The relationship between SCK and the Oscillator Clock frequency f_{osc} is shown in Table 21-5.

Table 21-5. Relationship Between SCK and the Oscillator Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

21.2.2 SPSR – SPI Status Register

Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIF: SPI Interrupt Flag**

When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If $\overline{SS}$ is an input and is driven low when the SPI is in Master mode, this will also set the SPIF Flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the SPI Data Register (SPDR).

- **Bit 6 – WCOL: Write COLlision Flag**

The WCOL bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

- **Bit 5:1 – Res: Reserved Bits**

These bits are reserved bits and will always read as zero.

- **Bit 0 – SPI2X: Double SPI Speed Bit**

When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode (see Table 21-5). This means that the minimum SCK period will be two CPU clock periods. When the SPI is configured as Slave, the SPI is only guaranteed to work at $f_{osc}/4$ or lower.

The SPI interface on the ATmega640/1280/1281/2560/2561 is also used for program memory and EEPROM downloading or uploading. See "Serial Downloading" on page 338 for serial programming and verification.

Ilustración 70: Registros de control y estado

8.4.3 Periférico USART

Formato de las tramas:

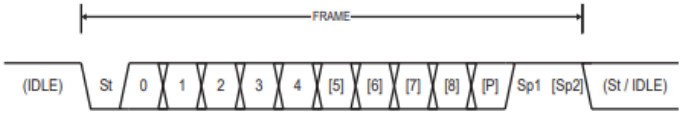
22.4 Frame Formats

A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The USART accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. Figure 22-4 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

Figure 22-4. Frame Formats



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

The frame format used by the USART is set by the UCSZn2:0, UPMn1:0 and USBSn bits in UCSRnB and UCSRnC. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

The USART Character SiZe (UCSZn2:0) bits select the number of data bits in the frame. The USART Parity mode (UPMn1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the USART Stop Bit Select (USBSn) bit. The Receiver ignores the second stop bit. An FE (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

22.4.1 Parity Bit Calculation

The parity bit is calculated by doing an exclusive-or of all the data bits. If odd parity is used, the result of the exclusive or is inverted. The parity bit is located between the last data bit and first stop bit of a serial frame. The relation between the parity bit and data bits is as follows:

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

P_{even}	Parity bit using even parity.
P^{odd}	Parity bit using odd parity.
d_n	Data bit n of the character.

Ilustración 71: Formato de las tramas

Registros del USART:

22.10.2 UCSRnA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – RXCn: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (that is, does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushed and consequently the RXCn bit will become zero. The RXCn Flag can be used to generate a Receive Complete interrupt (see description of the RXCIEn bit).

- **Bit 6 – TXCn: USART Transmit Complete**

This flag bit is set when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDRn). The TXCn Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXCn Flag can generate a Transmit Complete interrupt (see description of the TXCIEn bit).

- **Bit 5 – UDREn: USART Data Register Empty**

The UDREn Flag indicates if the transmit buffer (UDRn) is ready to receive new data. If UDREn is one, the buffer is empty, and therefore ready to be written. The UDREn Flag can generate a Data Register Empty interrupt (see description of the UDRIEn bit).

UDREn is set after a reset to indicate that the Transmitter is ready.

- **Bit 4 – FEn: Frame Error**

This bit is set if the next character in the receive buffer had a Frame Error when received, that is, when the first stop bit of the next character in the receive buffer is zero. This bit is valid until the receive buffer (UDRn) is read. The FEn bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRnA.

- **Bit 3 – DORn: Data OverRun**

This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

- **Bit 2 – UPEn: USART Parity Error**

This bit is set if the next character in the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UPMn1 = 1). This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

- **Bit 1 – U2Xn: Double the USART Transmission Speed**

This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.

Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

- **Bit 0 – MPCMn: Multi-processor Communication Mode**

This bit enables the Multi-processor Communication mode. When the MPCMn bit is written to one, all the incoming frames received by the USART Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the MPCMn setting. For more detailed information see "Multi-processor Communication Mode" on page 216.

22.10.3 UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – RXCIEn: RX Complete Interrupt Enable n**

Writing this bit to one enables interrupt on the RXCn Flag. A USART Receive Complete interrupt will be generated only if the RXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXCn bit in UCSRnA is set.

- **Bit 6 – TXCIEn: TX Complete Interrupt Enable n**

Writing this bit to one enables interrupt on the TXCn Flag. A USART Transmit Complete interrupt will be generated only if the TXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXCn bit in UCSRnA is set.

- **Bit 5 – UDRIEn: USART Data Register Empty Interrupt Enable n**

Writing this bit to one enables interrupt on the UDREn Flag. A Data Register Empty interrupt will be generated only if the UDRIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDREn bit in UCSRnA is set.

- **Bit 4 – RXENn: Receiver Enable n**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxDn pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FEn, DORn, and UPEn Flags.

- **Bit 3 – TXENn: Transmitter Enable n**

Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxDn pin when enabled. The disabling of the Transmitter (writing TXENn to zero) will not become effective until ongoing and pending transmissions are completed, that is, when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxDn port.

- **Bit 2 – UCSZn2: Character Size n**

The UCSZn2 bits combined with the UCSZn1:0 bit in UCSRnC sets the number of data bits (Character SiZe) in a frame the Receiver and Transmitter use.

- **Bit 1 – RXB8n: Receive Data Bit 8 n**

RXB8n is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDRn.

- **Bit 0 – TXB8n: Transmit Data Bit 8 n**

TXB8n is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDRn.

22.10.4 UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

- **Bits 7:6 – UMSELn1:0 USART Mode Select**

These bits select the mode of operation of the USARTn as shown in Table 22-4.

Table 22-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)(1)

Note: 1. See "USART in SPI Mode" on page 227 for full description of the Master SPI Mode (MSPIM) operation.

- **Bits 5:4 – UPMn1:0: Parity Mode**

These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPMn setting. If a mismatch is detected, the UPEn Flag in UCSRnA will be set.

Table 22-5. UPMn Bits Settings

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- **Bit 3 – USBSn: Stop Bit Select**

This bit selects the number of stop bits to be inserted by the Transmitter. The Receiver ignores this setting.

Table 22-6. USBS Bit Settings

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

- **Bit 2:1 – UCSZn1:0: Character Size**

The UCSZn1:0 bits combined with the UCSZn2 bit in UCSRnB sets the number of data bits (Character SiZe) in a frame the Receiver and Transmitter use.

Table 22-7. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- **Bit 0 – UCPOLn: Clock Polarity**

This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOLn bit sets the relationship between data output change and data input sample, and the synchronous clock (XCKn).

Table 22-8. UCPOLn Bit Settings

UCPOLn	Transmitted Data Changed (Output of TxDn Pin)	Received Data Sampled (Input on RxDn Pin)
0	Rising XCKn Edge	Falling XCKn Edge
1	Falling XCKn Edge	Rising XCKn Edge

22.10.5 UBRRnL and UBRRnH – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRR[11:8]				UBRRHn
	UBRR[7:0]								UBRRLn
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

- **Bit 15:12 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRH is written.

- **Bit 11:0 – UBRR11:0: USART Baud Rate Register**

This is a 12-bit register which contains the USART baud rate. The UBRRH contains the four most significant bits, and the UBRRL contains the eight least significant bits of the USART baud rate. Ongoing transmissions by the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing UBRRL will trigger an immediate update of the baud rate prescaler.

22.11 Examples of Baud Rate Setting

For standard crystal and resonator frequencies, the most commonly used baud rates for asynchronous operation can be generated by using the UBRR settings in Table 22-9 to Table 22-12 on page 226. UBRR values which yield an actual baud rate differing less than 0.5% from the target baud rate, are bold in the table. Higher error ratings are acceptable, but the Receiver will have less noise resistance when the error ratings are high, especially for large serial frames (see "Asynchronous Operational Range" on page 215). The error values are calculated using the following equation:

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \bullet 100\%$$

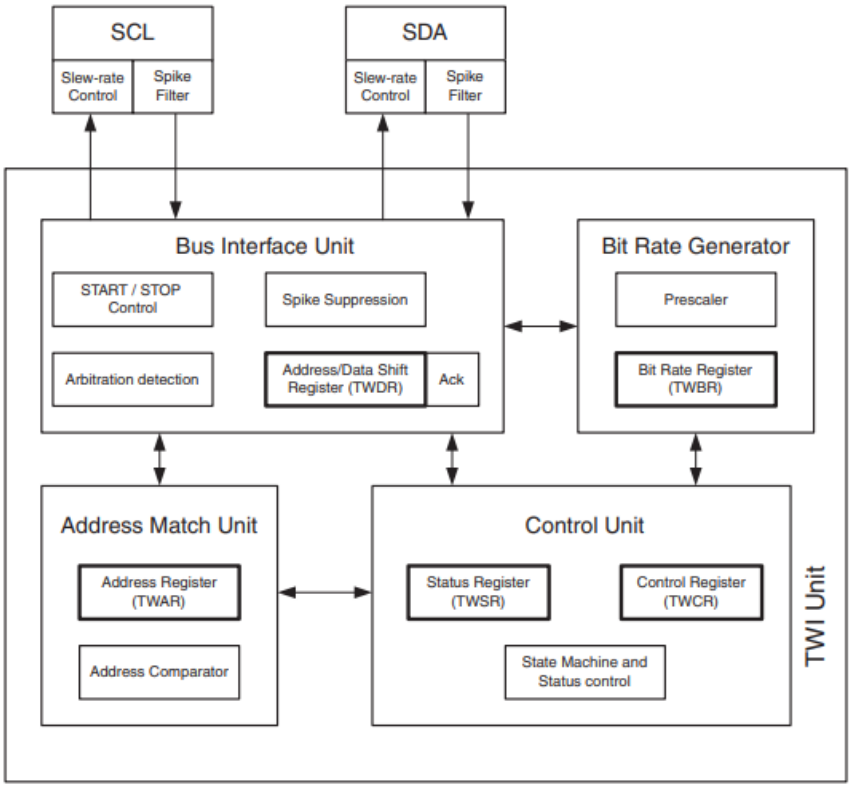
Table 22-9. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies

Baud Rate [bps]	f_{osc} = 1.0000MHz				f_{osc} = 1.8432MHz				f_{osc} = 2.0000MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4K	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2K	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8K	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4K	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6K	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8K	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2K	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4K	–	–	–	–	–	–	0	0.0%	–	–	–	–
250K	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max.(1)	62.5Kbps		125Kbps		115.2Kbps		230.4Kbps		125Kbps		250Kbps	

Note: 1. UBRR = 0, Error = 0.0%

8.4.4 Especificaciones del TWI compatible con I2C

Figure 24-9. Overview of the TWI Module



24.5.1 SCL and SDA Pins

These pins interface the AVR TWI with the rest of the MCU system. The output drivers contain a slew-rate limiter in order to conform to the TWI specification. The input stages contain a spike suppression unit removing spikes shorter than 50ns. Note that the internal pull-ups in the AVR pads can be enabled by setting the PORT bits corresponding to the SCL and SDA pins, as explained in the I/O Port section. The internal pull-ups can in some systems eliminate the need for external ones.

24.5.2 Bit Rate Generator Unit

This unit controls the period of SCL when operating in a Master mode. The SCL period is controlled by settings in the TWI Bit Rate Register (TWBR) and the Prescaler bits in the TWI Status Register (TWSR). Slave operation does not depend on Bit Rate or Prescaler settings, but the CPU clock frequency in the Slave must be at least 16 times higher than the SCL frequency. Note that slaves may prolong the SCL low period, thereby reducing the average TWI bus clock period.

The SCL frequency is generated according to the following equation:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

Ilustración 72: Vista preliminar del módulo TWI

Registros TWI:

24.9 Register Description

24.9.1 TWBR – TWI Bit Rate Register

Bit	7	6	5	4	3	2	1	0	
(0xB8)	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – TWI Bit Rate Register**

TWBR selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the Master modes. See "Bit Rate Generator Unit" on page 242 for calculating bit rates.

24.9.2 TWCR – TWI Control Register

Bit	7	6	5	4	3	2	1	0	
(0xBC)	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The TWCR is used to control the operation of the TWI. It is used to enable the TWI, to initiate a Master access by applying a START condition to the bus, to generate a Receiver acknowledge, to generate a stop condition, and to control halting of the bus while the data to be written to the bus are written to the TWDR. It also indicates a write collision if data is attempted written to TWDR while the register is inaccessible.

- **Bit 7 – TWINT: TWI Interrupt Flag**

This bit is set by hardware when the TWI has finished its current job and expects application software response. If the I-bit in SREG and TWIE in TWCR are set, the MCU will jump to the TWI Interrupt Vector. While the TWINT Flag is set, the SCL low period is stretched. The TWINT Flag must be cleared by software by writing a logic one to it. Note that this flag is not automatically cleared by hardware when executing the interrupt routine. Also note that clearing this flag starts the operation of the TWI, so all accesses to the TWI Address Register (TWAR), TWI Status Register (TWSR), and TWI Data Register (TWDR) must be complete before clearing this flag.

- **Bit 6 – TWEA: TWI Enable Acknowledge Bit**

The TWEA bit controls the generation of the acknowledge pulse. If the TWEA bit is written to one, the ACK pulse is generated on the TWI bus if the following conditions are met:

1. The device's own slave address has been received.
2. A general call has been received, while the TWGCE bit in the TWAR is set.
3. A data byte has been received in Master Receiver or Slave Receiver mode.

By writing the TWEA bit to zero, the device can be virtually disconnected from the 2-wire Serial Bus temporarily. Address recognition can then be resumed by writing the TWEA bit to one again.

- **Bit 5 – TWSTA: TWI START Condition Bit**

The application writes the TWSTA bit to one when it desires to become a Master on the 2-wire Serial Bus. The TWI hardware checks if the bus is available, and generates a START condition on the bus if it is free. However, if the bus is not free, the TWI waits until a STOP condition is detected, and then generates a new START condition to claim the bus Master status. TWSTA must be cleared by software when the START condition has been transmitted.

- **Bit 4 – TWSTO: TWI STOP Condition Bit**

Writing the TWSTO bit to one in Master mode will generate a STOP condition on the 2-wire Serial Bus. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically. In Slave mode, setting the TWSTO bit can be used to recover from an error condition. This will not generate a STOP condition, but the TWI returns to a well-defined unaddressed Slave mode and releases the SCL and SDA lines to a high impedance state.

- **Bit 3 – TWWC: TWI Write Collision Flag**

The TWWC bit is set when attempting to write to the TWI Data Register – TWDR when TWINT is low. This flag is cleared by writing the TWDR Register when TWINT is high.

- **Bit 2 – TWEN: TWI Enable Bit**

The TWEN bit enables TWI operation and activates the TWI interface. When TWEN is written to one, the TWI takes control over the I/O pins connected to the SCL and SDA pins, enabling the slew-rate limiters and spike filters. If this bit is written to zero, the TWI is switched off and all TWI transmissions are terminated, regardless of any ongoing operation.

- **Bit 1 – Res: Reserved Bit**

This bit is a reserved bit and will always read as zero.

- **Bit 0 – TWIE: TWI Interrupt Enable**

When this bit is written to one, and the I-bit in SREG is set, the TWI interrupt request will be activated for as long as the TWINT Flag is high.

24.9.3 TWSR – TWI Status Register

Bit	7	6	5	4	3	2	1	0	
(0xB9)	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

- **Bits 7:3 – TWS: TWI Status**

These five bits reflect the status of the TWI logic and the 2-wire Serial Bus. The different status codes are described later in this section. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the prescaler bits to zero when checking the Status bits. This makes status checking independent of prescaler setting. This approach is used in this datasheet, unless otherwise noted.

- **Bit 2 – Res: Reserved Bit**

This bit is reserved and will always read as zero.

- **Bits 1:0 – TWPS: TWI Prescaler Bits**

These bits can be read and written, and control the bit rate prescaler.

Table 24-7. TWI Bit Rate Prescaler

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

To calculate bit rates, see "Bit Rate Generator Unit" on page 242. The value of TWPS1:0 is used in the equation.

24.9.4 TWDR – TWI Data Register

Bit	7	6	5	4	3	2	1	0	
(0xBB)	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

In Transmit mode, TWDR contains the next byte to be transmitted. In Receive mode, the TWDR contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when the TWI Interrupt Flag (TWINT) is set by hardware. Note that the Data Register cannot be initialized by the user before the first interrupt occurs. The data in TWDR remains stable as long as TWINT is set. While data is shifted out, data on the bus is simultaneously shifted in. TWDR always contains the last byte present on the bus, except after a wake up from a sleep mode by the TWI interrupt. In this case, the contents of TWDR is undefined. In the case of a lost bus arbitration, no data is lost in the transition from Master to Slave. Handling of the ACK bit is controlled automatically by the TWI logic, the CPU cannot access the ACK bit directly.

- **Bits 7:0 – TWD: TWI Data Register**

These eight bits constitute the next data byte to be transmitted, or the latest data byte received on the 2-wire Serial Bus.

24.9.5 TWAR – TWI (Slave) Address Register

Bit	7	6	5	4	3	2	1	0	
(0xBA)	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

The TWAR should be loaded with the 7-bit Slave address (in the seven most significant bits of TWAR) to which the TWI will respond when programmed as a Slave Transmitter or Receiver, and not needed in the Master modes. In multimaster systems, TWAR must be set in masters which can be addressed as Slaves by other Masters.

The LSB of TWAR is used to enable recognition of the general call address (0x00). There is an associated address comparator that looks for the slave address (or general call address if enabled) in the received serial address. If a match is found, an interrupt request is generated.

- **Bits 7:1 – TWA: TWI (Slave) Address Register**

These seven bits constitute the slave address of the TWI unit.

- **Bit 0 – TWGCE: TWI General Call Recognition Enable Bit**

If set, this bit enables the recognition of a General Call given over the 2-wire Serial Bus.

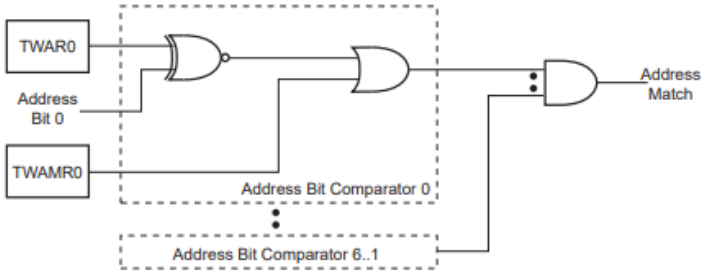
24.9.6 TWAMR – TWI (Slave) Address Mask Register

Bit	7	6	5	4	3	2	1	0	
(0xBD)	TWAM[6:0]							–	TWAMR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:1 – TWAM: TWI Address Mask**

The TWAMR can be loaded with a 7-bit Slave Address mask. Each of the bits in TWAMR can mask (disable) the corresponding address bit in the TWI Address Register (TWAR). If the mask bit is set to one then the address match logic ignores the compare between the incoming address bit and the corresponding bit in TWAR. Figure 24-22 shows the address match logic in detail.

Figure 24-22. TWI Address Match Logic, Block Diagram



- **Bit 0 – Res: Reserved Bit**

This bit is reserved and will always read as zero.

8.5 Módulo receptor GNSS

Descripción del formato de las tramas NMEA:

u-blox locate, communicate, accelerate

NMEA Protocol

16 Protocol Overview

NMEA messages sent by the GPS receiver are based on NMEA 0183 Version 2.3. The following picture shows the structure of a NMEA protocol message.

NMEA Protocol Frame

Checksum range

$	<Address>	{,<value>}	*<checksum>	<CR><LF>
Start character	Address field.	Data field(s)	Checksum field	End sequence
Always '$'	Only digits and uppercase letters, cannot be null. This field is subdivided into 2 fields: <XX> Talker Identifier, always **GP** for a GPS receiver, **P** for proprietary Messages; <XXX> Sentence Formatter Defines the message content	Delimited by a ','. Length can vary, even for a certain field.	Starts with a '*' and consists of 2 characters representing a hex number. The checksum is the exclusive OR of all characters between '$' and '*'.	Always <CR><LF>

Example:

$	GP	ZDA	,141644.00,22,03,2002,00,00	*67	<CR><LF>

For further information on the NMEA Standard please refer to *NMEA 0183 Standard For Interfacing Marine Electronic Devices*, Version 2.30, March 1, 1998. See http://www.nmea.org/ for ordering instructions.
The NMEA standard allows for proprietary, manufacturer-specific messages to be added. These shall be marked with a manufacturer mnemonic. The mnemonic assigned to u-blox is UBX and is used for all non-standard messages. These proprietary NMEA messages therefore have the address field set to PUBX. The first data field in a PUBX message identifies the message number with two digits.

Lista de mensajes NMEA:

u-blox locate, communicate, accelerate

An exception from the above default are dead reckoning fixes, which are also output when invalid (user limits exceeded).

In Antaris firmware versions older than 3.0, the receiver did output invalid data and marked it with the 'Invalid/Valid' Flags. If required, this function can still be enabled in later firmware versions, using the UBX protocol message CFG-NMEA.

Differing from the NMEA standard, u-blox reports valid dead reckoning fixes with user limits met (not exceeded) as valid (A) instead of invalid (V).

19 NMEA Messages Overview

When configuring NMEA messages using the UBX protocol message CFG-MSG, the Class/Ids shown in the table shall be used.

Page	Mnemonic	Cls/ID	Description
	NMEA Proprietary Messages		**Proprietary Messages**
68	**UBX,00**	0xF1 0x00	Poll a PUBX,00 message
69	**UBX,00**	0xF1 0x00	Lat/Long Position Data
71	**UBX,03**	0xF1 0x03	Poll a PUBX,03 message
72	**UBX,03**	0xF1 0x03	Satellite Status
74	**UBX,04**	0xF1 0x04	Poll a PUBX,04 message
75	**UBX,04**	0xF1 0x04	Time of Day and Clock Information
76	**UBX,05**	0xF1 0x05	Poll a PUBX,05 message
77	**UBX,05**	0xF1 0x05	Lat/Long Position Data
79	**UBX,06**	0xF1 0x06	Poll a PUBX,06 message
80	**UBX,06**	0xF1 0x06	Lat/Long Position Data
82	**UBX,40**	0xF1 0x40	Set NMEA message output rate
83	**UBX,41**	0xF1 0x41	Set Protocols and Baudrate
	NMEA Standard Messages		**Standard Messages**
54	**DTM**	0xF0 0x0A	Datum Reference
55	**GBS**	0xF0 0x09	GNSS Satellite Fault Detection
56	**GGA**	0xF0 0x00	Global positioning system fix data
57	**GLL**	0xF0 0x01	Latitude and longitude, with time of position fix and status
58	**GPQ**	0xF0 0x40	Poll message
59	**GRS**	0xF0 0x06	GNSS Range Residuals
60	**GSA**	0xF0 0x02	GNSS DOP and Active Satellites
61	**GST**	0xF0 0x07	GNSS Pseudo Range Error Statistics
62	**GSV**	0xF0 0x03	GNSS Satellites in View
63	**RMC**	0xF0 0x04	Recommended Minimum data
64	**THS**	0xF0 0x0E	True Heading and Status
65	**TXT**	0xF0 0x41	Text Transmission
66	**VTG**	0xF0 0x05	Course over ground and Ground speed

Descripción del mensaje NMEA que activa o desactiva el envío de cada mensaje:

u-blox locate, communicate, accelerate

21.11 UBX,40

Message	**UBX,40**		
Description	**Set NMEA message output rate**		
Firmware	Supported on u-blox 6 from firmware version 6.00 up to version 7.03.		
Type	Set Message		
Comment	Set/Get message rate configuration (s) to/from the receiver. • Send rate is relative to the event a message is registered on. For example, if the rate of a navigation message is set to 2, the message is sent every second navigation solution.		
Message Info	*ID for CFG-MSG*	*Number of fields*	
	0xF1 0x40	11	

Message Structure:

```
$PUBX,40,msgId,rddc,rus1,rus2,rusb,rspi,reserved*cs<CR><LF>
```

Example:

```
$PUBX,40,GLL,1,0,0,0,0,0*5D
```

Field No.	*Example*	*Format*	*Name*	*Unit*	*Description*
0	$PUBX	string	`$PUBX`	-	Message ID, UBX protocol header, proprietary sentence
1	40	numeric	`ID`	-	Proprietary message identifier
2	GLL	string	`MsgId`	-	NMEA message identifier
3	1	numeric	`rddc`	cycles	output rate on DDC - 0 disables that message from being output on this port - 1 means that this message is output every epoch
4	1	numeric	`rus1`	cycles	output rate on USART 1 - 0 disables that message from being output on this port - 1 means that this message is output every epoch
5	1	numeric	`rus2`	cycles	output rate on USART 2 - 0 disables that message from being output on this port - 1 means that this message is output every epoch
6	1	numeric	`rusb`	cycles	output rate on USB - 0 disables that message from being output on this port - 1 means that this message is output every epoch
7	1	numeric	`rspi`	cycles	output rate on SPI - 0 disables that message from being output on this port - 1 means that this message is output every epoch
8	0	numeric	`reserved`	-	Reserved, Always fill with 0
9	*5D	hexadecimal	`cs`	-	Checksum
10	-	character	`<CR><LF>`	-	Carriage Return and Line Feed

Se utilizan dos tramas GNSS para obtener los datos necesarios para el funcionamiento del sistema: GPGGA y GPMRC

20.3 GGA

Message	**GGA**		
Description	**Global positioning system fix data**		
Firmware	Supported on u-blox 6 from firmware version 6.00 up to version 7.03.		
Type	Output Message		
Comment	**The output of this message is dependent on the currently selected datum (Default: WGS84)** Time and position, together with GPS fixing related data (number of satellites in use, and the resulting HDOP, age of differential data if in use, etc.).		
Message Info	*ID for CFG-MSG*	*Number of fields*	
	0xF0 0x00	17	

Message Structure:

```
$GPGGA,hhmmss.ss,Latitude,N,Longitude,E,FS,NoSV,HDOP,msl,m,Altref,m,DiffAge,DiffStation*cs<CR><LF>
```

Example:

```
$GPGGA,092725.00,4717.11399,N,00833.91590,E,1,8,1.01,499.6,M,48.0,M,,0*5B
```

Field No.	*Example*	*Format*	*Name*	*Unit*	*Description*
0	$GPGGA	string	$GPGGA	-	Message ID, GGA protocol header
1	092725.00	hhmmss.sss	hhmmss.ss	-	UTC Time, Current time
2	4717.11399	ddmm.mmmm	Latitude	-	Latitude, Degrees + minutes, see Format description
3	N	character	N	-	N/S Indicator, N=north or S=south
4	00833.91590	dddmm.mmmm	Longitude	-	Longitude, Degrees + minutes, see Format description
5	E	character	E	-	E/W indicator, E=east or W=west
6	1	digit	FS	-	Position Fix Status Indicator, See Table below and Position Fix Flags description
7	8	numeric	NoSV	-	Satellites Used, Range 0 to 12
8	1.01	numeric	HDOP	-	HDOP, Horizontal Dilution of Precision
9	499.6	numeric	msl	m	MSL Altitude
10	M	character	uMsl	-	Units, Meters (fixed field)
11	48.0	numeric	Altref	m	Geoid Separation
12	M	character	uSep	-	Units, Meters (fixed field)
13	-	numeric	DiffAge	s	Age of Differential Corrections, Blank (Null) fields when DGPS is not used
14	0	numeric	DiffStation	-	Diff. Reference Station ID
15	*5B	hexadecimal	cs	-	Checksum
16	-	character	<CR><LF>	-	Carriage Return and Line Feed

20.10 RMC

Message	**RMC**		
Description	**Recommended Minimum data**		
Firmware	Supported on u-blox 6 from firmware version 6.00 up to version 7.03.		
Type	Output Message		
Comment	**The output of this message is dependent on the currently selected datum (Default: WGS84)** The Recommended Minimum sentence defined by NMEA for GPS/Transit system data.		
Message Info	*ID for CFG-MSG*	*Number of fields*	
	0xF0 0x04	15	

Message Structure:

```
$GPRMC,hhmmss,status,latitude,N,longitude,E,spd,cog,ddmmyy,mv,mvE,mode*cs<CR><LF>
```

Example:

```
$GPRMC,083559.00,A,4717.11437,N,00833.91522,E,0.004,77.52,091202,,,A*57
```

Field No.	*Example*	*Format*	*Name*	*Unit*	*Description*
0	$GPRMC	string	$GPRMC	-	Message ID, RMC protocol header
1	083559.00	hhmmss.sss	hhmmss.ss	-	UTC Time, Time of position fix
2	A	character	Status	-	Status, V = Navigation receiver warning, A = Data valid, see Position Fix Flags description
3	4717.11437	ddmm.mmmm	Latitude	-	Latitude, Degrees + minutes, see Format description
4	N	character	N	-	N/S Indicator, hemisphere N=north or S=south
5	00833.91522	dddmm.mmmm	Longitude	-	Longitude, Degrees + minutes, see Format description
6	E	character	E	-	E/W indicator, E=east or W=west
7	0.004	numeric	Spd	knots	Speed over ground
8	77.52	numeric	Cog	degrees	Course over ground
9	091202	ddmmyy	date	-	Date in day, month, year format
10	-	numeric	mv	degrees	Magnetic variation value, not being output by receiver
11	-	character	mvE	-	Magnetic variation E/W indicator, not being output by receiver
12	-	character	mode	-	Mode Indicator, see Position Fix Flags description
13	*57	hexadecimal	cs	-	Checksum
14	-	character	<CR><LF>	-	Carriage Return and Line Feed

Descripción del formato de los mensajes UBX:

UBX Protocol

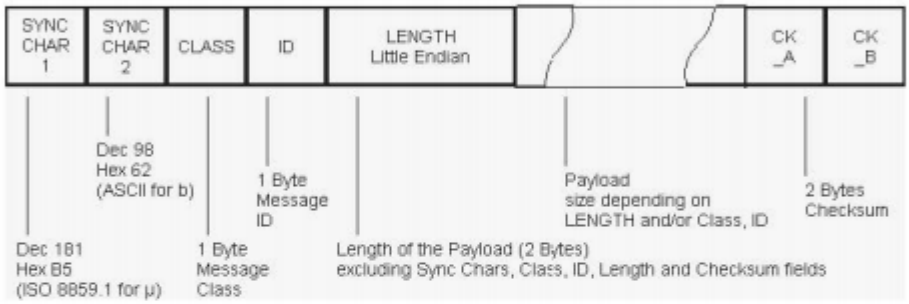
22 UBX Protocol Key Features

u-blox GPS receivers use a u-blox proprietary protocol to transmit GPS data to a host computer using asynchronous RS232 ports. This protocol has the following key features:

- Compact - uses 8 Bit Binary Data.
- Checksum Protected - uses a low-overhead checksum algorithm
- Modular - uses a 2-stage message identifier (Class- and Message ID)

23 UBX Packet Structure

A basic UBX Packet looks as follows:



- Every Message starts with 2 Bytes: 0xB5 0x62
- A 1 Byte Class Field follows. The Class defines the basic subset of the message
- A 1 Byte ID Field defines the message that is to follow
- A 2 Byte Length Field is following. Length is defined as being the length of the payload, only. It does not include Sync Chars, Length Field, Class, ID or CRC fields. The number format of the length field is an unsigned 16-Bit integer in Little Endian Format.
- The Payload is a variable length field.
- CK_A and CK_B is a 16 Bit checksum whose calculation is defined below.

24 UBX Class IDs

A Class is a grouping of messages which are related to each other. The following table gives the short names, description and Class ID Definitions.

Name	*Class*	*Description*
NAV	0x01	Navigation Results: Position, Speed, Time, Acc, Heading, DOP, SVs used
RXM	0x02	Receiver Manager Messages: Satellite Status, RTC Status
INF	0x04	Information Messages: Printf-Style Messages, with IDs such as Error, Warning, Notice
ACK	0x05	Ack/Nack Messages: as replies to CFG Input Messages
CFG	0x06	Configuration Input Messages: Set Dynamic Model, Set DOP Mask, Set Baud Rate, etc.
MON	0x0A	Monitoring Messages: Comunication Status, CPU Load, Stack Usage, Task Status
AID	0x0B	AssistNow Aiding Messages: Ephemeris, Almanac, other A-GPS data input
TIM	0x0D	Timing Messages: Timepulse Output, Timemark Results

Lista de mensajes UBX:

28 UBX Messages Overview

Page	Mnemonic	Cls/ID	Length	Type	Description
UBX Class ACK				**Ack/Nack Messages**	
91	**ACK-ACK**	0x05 0x01	2	Answer	Message Acknowledged
91	**ACK-NAK**	0x05 0x00	2	Answer	Message Not-Acknowledged
UBX Class AID				**AssistNow Aiding Messages**	
92	**AID-ALM**	0x0B 0x30	0	Poll Request	Poll GPS Aiding Almanac Data
92	**AID-ALM**	0x0B 0x30	1	Poll Request	Poll GPS Aiding Almanac Data for a SV
93	**AID-ALM**	0x0B 0x30	(8) or (40)	Input/Output Message	GPS Aiding Almanac Input/Output Message
93	**AID-ALPSRV**	0x0B 0x32	16	Output Message	ALP client requests AlmanacPlus data from server
94	**AID-ALPSRV**	0x0B 0x32	16 + 1*dataSize	Input Message	ALP server sends AlmanacPlus data to client
95	**AID-ALPSRV**	0x0B 0x32	8 + 2*size	Output Message	ALP client sends AlmanacPlus data to server
95	**AID-ALP**	0x0B 0x50	0 + 2*N	Input message	ALP file data transfer to the receiver
96	**AID-ALP**	0x0B 0x50	1	Input message	Mark end of data transfer
96	**AID-ALP**	0x0B 0x50	1	Output message	Acknowledges a data transfer
96	**AID-ALP**	0x0B 0x50	1	Output message	Indicate problems with a data transfer
97	**AID-ALP**	0x0B 0x50	24	Periodic/Polled	Poll the AlmanacPlus status
97	**AID-AOP**	0x0B 0x33	0	Poll request	Poll AssistNow Autonomous data
98	**AID-AOP**	0x0B 0x33	1	Poll request	Poll AssistNow Autonomous data for one satellite
98	**AID-AOP**	0x0B 0x33	(48) or (192)	Input/Output Message	AssistNow Autonomous data
99	**AID-DATA**	0x0B 0x10	0	Poll	Polls all GPS Initial Aiding Data
99	**AID-EPH**	0x0B 0x31	0	Poll Request	Poll GPS Aiding Ephemeris Data
99	**AID-EPH**	0x0B 0x31	1	Poll Request	Poll GPS Aiding Ephemeris Data for a SV
100	**AID-EPH**	0x0B 0x31	(8) or (104)	Input/Output Message	GPS Aiding Ephemeris Input/Output Message
101	**AID-HUI**	0x0B 0x02	0	Poll Request	Poll GPS Health, UTC and ionosphere parameters
101	**AID-HUI**	0x0B 0x02	72	Input/Output Message	GPS Health, UTC and ionosphere parameters
102	**AID-INI**	0x0B 0x01	0	Poll Request	Poll GPS Initial Aiding Data
103	**AID-INI**	0x0B 0x01	48	Polled	Aiding position, time, frequency, clock drift
104	**AID-REQ**	0x0B 0x00	0	Virtual	Sends a poll (AID-DATA) for all GPS Aiding Data
UBX Class CFG				**Configuration Input Messages**	
105	**CFG-ANT**	0x06 0x13	0	Poll Request	Poll Antenna Control Settings
105	**CFG-ANT**	0x06 0x13	4	Get/Set	Get/Set Antenna Control Settings
106	**CFG-CFG**	0x06 0x09	(12) or (13)	Command	Clear, Save and Load configurations
108	**CFG-DAT**	0x06 0x06	0	Poll Request	Poll Datum Setting
108	**CFG-DAT**	0x06 0x06	2	Set	Set Standard Datum
108	**CFG-DAT**	0x06 0x06	44	Set	Set User-defined Datum
109	**CFG-DAT**	0x06 0x06	52	Get	Get currently selected Datum
110	**CFG-EKF**	0x06 0x12	0	Poll Request	Poll EKF Module Settings
110	**CFG-EKF**	0x06 0x12	16	Get/Set	Get/Set EKF Module Settings - LEA-6R

UBX Messages Overview continued

Page	Mnemonic	Cls/ID	Length	Type	Description
112	**CFG-ESFGWT**	0x06 0x29	44	Get/Set message	Get/Set settings of gyro+wheel tick sol (GWT) - LEA-
113	**CFG-FXN**	0x06 0x0E	0	Poll Request	Poll FXN configuration
113	**CFG-FXN**	0x06 0x0E	36	Command	RXM FixNOW configuration
114	**CFG-INF**	0x06 0x02	1	Poll Request	Poll INF message configuration for one protocol
115	**CFG-INF**	0x06 0x02	0 + 10*N	Set/Get	Information message configuration
116	**CFG-ITFM**	0x06 0x39	8	Command	Jamming/Interference Monitor configuration
117	**CFG-MSG**	0x06 0x01	2	Poll Request	Poll a message configuration
117	**CFG-MSG**	0x06 0x01	8	Set/Get	Set Message Rate(s)
118	**CFG-MSG**	0x06 0x01	3	Set/Get	Set Message Rate
118	**CFG-NAV5**	0x06 0x24	0	Poll Request	Poll Navigation Engine Settings
119	**CFG-NAV5**	0x06 0x24	36	Get/Set	Get/Set Navigation Engine Settings
120	**CFG-NAVX5**	0x06 0x23	0	Poll Request	Poll Navigation Engine Expert Settings
120	**CFG-NAVX5**	0x06 0x23	40	Get/Set	Get/Set Navigation Engine Expert Settings
122	**CFG-NMEA**	0x06 0x17	0	Poll Request	Poll the NMEA protocol configuration
122	**CFG-NMEA**	0x06 0x17	4	Set/Get	Set/Get the NMEA protocol configuration
123	**CFG-NVS**	0x06 0x22	13	Command	Clear, Save and Load non-volatile storage data
125	**CFG-PM2**	0x06 0x3B	0	Poll Request	Poll extended Power Management configuration
125	**CFG-PM2**	0x06 0x3B	44	Set/Get	Extended Power Management configuration
127	**CFG-PM**	0x06 0x32	0	Poll Request	Poll Power Management configuration
127	**CFG-PM**	0x06 0x32	24	Set/Get	Power Management configuration
129	**CFG-PRT**	0x06 0x00	0	Poll Request	Polls the configuration of the used I/O Port
129	**CFG-PRT**	0x06 0x00	1	Poll Request	Polls the configuration for one I/O Port
129	**CFG-PRT**	0x06 0x00	20	Get/Set	Get/Set Port Configuration for UART
132	**CFG-PRT**	0x06 0x00	20	Get/Set	Get/Set Port Configuration for USB Port
133	**CFG-PRT**	0x06 0x00	20	Get/Set	Get/Set Port Configuration for SPI Port
136	**CFG-PRT**	0x06 0x00	20	Get/Set	Get/Set Port Configuration for DDC Port
138	**CFG-RATE**	0x06 0x08	0	Poll Request	Poll Navigation/Measurement Rate Settings
138	**CFG-RATE**	0x06 0x08	6	Get/Set	Navigation/Measurement Rate Settings
139	**CFG-RINV**	0x06 0x34	0	Poll Request	Poll contents of Remote Inventory
139	**CFG-RINV**	0x06 0x34	1 + 1*N	Set/Get	Set/Get contents of Remote Inventory
140	**CFG-RST**	0x06 0x04	4	Command	Reset Receiver / Clear Backup Data Structures
141	**CFG-RXM**	0x06 0x11	0	Poll Request	Poll RXM configuration
141	**CFG-RXM**	0x06 0x11	2	Set/Get	RXM configuration
142	**CFG-SBAS**	0x06 0x16	0	Poll Request	Poll contents of SBAS Configuration
142	**CFG-SBAS**	0x06 0x16	8	Command	SBAS Configuration
144	**CFG-TMODE2**	0x06 0x3D	0	Poll Request	Poll Time Mode Settings
144	**CFG-TMODE2**	0x06 0x3D	28	Get/Set	Time Mode Settings 2
145	**CFG-TMODE**	0x06 0x1D	0	Poll Request	Poll Time Mode Settings

UBX Messages Overview continued

Page	Mnemonic	Cls/ID	Length	Type	Description
145	**CFG-TMODE**	0x06 0x1D	28	Get/Set	Time Mode Settings
146	**CFG-TPS**	0x06 0x31	0	Poll Request	Poll Timepulse Parameters
146	**CFG-TPS**	0x06 0x31	1	Poll Request	Poll TimePulse Parameters
147	**CFG-TPS**	0x06 0x31	32	Get/Set	Get/Set TimePulse Parameters
148	**CFG-TP**	0x06 0x07	0	Poll Request	Poll TimePulse Parameters
148	**CFG-TP**	0x06 0x07	20	Get/Set	Get/Set TimePulse Parameters
149	**CFG-USB**	0x06 0x1B	0	Poll Request	Poll a USB configuration
149	**CFG-USB**	0x06 0x1B	108	Get/Set	Get/Set USB Configuration
UBX Class ESF				**External Sensor Fusion Messages**	
151	**ESF-MEAS**	0x10 0x02	(8 + 4*N) or (12 + 4*N)	Input/Output Message	External Sensor Fusion Measurements (LEA-6R)
152	**ESF-STATUS**	0x10 0x10	16 + 4*numSens	Periodic/Polled	Sensor Fusion Status Information (LEA-6R)
154	**ESF-STATUS**	0x10 0x10	16 + 4*numSens	Periodic/Polled	Sensor Fusion Status Information (LEA-6R)
UBX Class INF				**Information Messages**	
157	**INF-DEBUG**	0x04 0x04	0 + 1*N	Output	ASCII String output, indicating debug output
157	**INF-ERROR**	0x04 0x00	0 + 1*N	Output	ASCII String output, indicating an error
158	**INF-NOTICE**	0x04 0x02	0 + 1*N	Output	ASCII String output, with informational contents
158	**INF-TEST**	0x04 0x03	0 + 1*N	Output	ASCII String output, indicating test output
159	**INF-WARNING**	0x04 0x01	0 + 1*N	Output	ASCII String output, indicating a warning
UBX Class MON				**Monitoring Messages**	
160	**MON-HW2**	0x0A 0x0B	28	Periodic/Polled	Extended Hardware Status
161	**MON-HW**	0x0A 0x09	68	Periodic/Polled	Hardware Status
162	**MON-HW**	0x0A 0x09	68	Periodic/Polled	Hardware Status
163	**MON-IO**	0x0A 0x02	0 + 20*N	Periodic/Polled	I/O Subsystem Status
164	**MON-MSGPP**	0x0A 0x06	120	Periodic/Polled	Message Parse and Process Status
164	**MON-RXBUF**	0x0A 0x07	24	Periodic/Polled	Receiver Buffer Status
165	**MON-RXR**	0x0A 0x21	1	Get	Receiver Status Information
165	**MON-TXBUF**	0x0A 0x08	28	Periodic/Polled	Transmitter Buffer Status
166	**MON-VER**	0x0A 0x04	70 + 30*N	Answer to Poll	Receiver/Software/ROM Version
UBX Class NAV				**Navigation Results**	
167	**NAV-AOPSTATUS**	0x01 0x60	20	Periodic/Polled	AssistNow Autonomous Status
167	**NAV-CLOCK**	0x01 0x22	20	Periodic/Polled	Clock Solution
168	**NAV-DGPS**	0x01 0x31	16 + 12*numCh	Periodic/Polled	DGPS Data Used for NAV
169	**NAV-DOP**	0x01 0x04	18	Periodic/Polled	Dilution of precision
169	**NAV-EKFSTATUS**	0x01 0x40	36	Periodic/Polled	Dead Reckoning Software Status
171	**NAV-POSECEF**	0x01 0x01	20	Periodic/Polled	Position Solution in ECEF
172	**NAV-POSLLH**	0x01 0x02	28	Periodic/Polled	Geodetic Position Solution
172	**NAV-SBAS**	0x01 0x32	12 + 12*cnt	Periodic/Polled	SBAS Status Data
174	**NAV-SOL**	0x01 0x06	52	Periodic/Polled	Navigation Solution Information

UBX Messages Overview continued

Page	*Mnemonic*	*Cls/ID*	*Length*	*Type*	*Description*
175	**NAV-STATUS**	0x01 0x03	16	Periodic/Polled	Receiver Navigation Status
177	**NAV-SVINFO**	0x01 0x30	8 + 12*numCh	Periodic/Polled	Space Vehicle Information
179	**NAV-TIMEGPS**	0x01 0x20	16	Periodic/Polled	GPS Time Solution
179	**NAV-TIMEUTC**	0x01 0x21	20	Periodic/Polled	UTC Time Solution
180	**NAV-VELECEF**	0x01 0x11	20	Periodic/Polled	Velocity Solution in ECEF
181	**NAV-VELNED**	0x01 0x12	36	Periodic/Polled	Velocity Solution in NED
UBX Class RXM				**Receiver Manager Messages**	
182	**RXM-ALM**	0x02 0x30	0	Poll Request	Poll GPS Constellation Almanach Data
182	**RXM-ALM**	0x02 0x30	1	Poll Request	Poll GPS Constellation Almanach Data for a SV
183	**RXM-ALM**	0x02 0x30	(8) or (40)	Poll Answer / Periodic	GPS Aiding Almanach Input/Output Message
183	**RXM-EPH**	0x02 0x31	0	Poll Request	Poll GPS Constellation Ephemeris Data
184	**RXM-EPH**	0x02 0x31	1	Poll Request	Poll GPS Constellation Ephemeris Data for a SV
184	**RXM-EPH**	0x02 0x31	(8) or (104)	Poll Answer / Periodic	GPS Aiding Ephemeris Input/Output Message
185	**RXM-PMREQ**	0x02 0x41	8	Input	Requests a Power Management task
185	**RXM-RAW**	0x02 0x10	8 + 24*numSV	Periodic/Polled	Raw Measurement Data
186	**RXM-SFRB**	0x02 0x11	42	Periodic	Subframe Buffer
187	**RXM-SVSI**	0x02 0x20	8 + 6*numSV	Periodic/Polled	SV Status Info
UBX Class TIM				**Timing Messages**	
189	**TIM-SVIN**	0x0D 0x04	28	Periodic/Polled	Survey-in data
189	**TIM-TM2**	0x0D 0x03	28	Periodic/Polled	Time mark data
191	**TIM-TP**	0x0D 0x01	16	Periodic/Polled	Timepulse Timedata
192	**TIM-VRFY**	0x0D 0x06	20	Polled/Once	Sourced Time Verification

Descripción UBX que establece la velocidad del puerto:

31.16 CFG-PRT (0x06 0x00)

31.16.1 Polls the configuration of the used I/O Port

Message	**CFG-PRT**				
Description	**Polls the configuration of the used I/O Port**				
Firmware	Supported on u-blox 6 from firmware version 6.00 up to version 7.03.				
Type	Poll Request				
Comment	Polls the configuration of the I/O Port on which this message is received				
	Header	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
Message Structure	0xB5 0x62	0x06 0x00	0	*see below*	CK_A CK_B
No payload					

31.16.2 Polls the configuration for one I/O Port

Message	**CFG-PRT**				
Description	**Polls the configuration for one I/O Port**				
Firmware	Supported on u-blox 6 from firmware version 6.00 up to version 7.03.				
Type	Poll Request				
Comment	Sending this message with a port ID as payload results in having the receiver return the configuration for the specified port.				
	Header	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
Message Structure	0xB5 0x62	0x06 0x00	1	*see below*	CK_A CK_B
Payload Contents:					

Byte Offset	*Number Format*	*Scaling*	*Name*	*Unit*	*Description*
0	U1	-	`PortID`	-	Port Identifier Number (see the other versions of CFG-PRT for valid values)

31.16.3 Get/Set Port Configuration for UART

Message	**CFG-PRT**				
Description	**Get/Set Port Configuration for UART**				
Firmware	Supported on u-blox 6 from firmware version 6.00 up to version 7.03.				
Type	Get/Set				
Comment	Several configurations can be concatenated to one input message. In this case the payload length can be a multiple of the normal length (see the other versions of CFG-PRT). Output messages from the module contain only one configuration unit.				
	Header	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
Message Structure	0xB5 0x62	0x06 0x00	20	*see below*	CK_A CK_B
Payload Contents:					

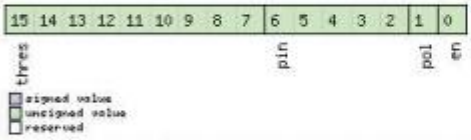
Byte Offset	*Number Format*	*Scaling*	*Name*	*Unit*	*Description*
0	U1	-	`portID`	-	Port Identifier Number (= 1 or 2 for UART ports)
1	U1	-	`reserved0`	-	Reserved
2	X2	-	`txReady`	-	reserved (Alwyas set to zero) up to Firmware 7.01, TX ready PIN configuration (since Firmware 7.01) (see graphic below)

CFG-PRT continued

Byte Offset	*Number Format*	*Scaling*	*Name*	*Unit*	*Description*
4	X4	-	`mode`	-	A bit mask describing the UART mode (see graphic below)
8	U4	-	`baudRate`	Bits/s	Baudrate in bits/second
12	X2	-	`inProtoMask`	-	A mask describing which input protocols are active. Each bit of this mask is used for a protocol. Through that, multiple protocols can be defined on a single port. (see graphic below)
14	X2	-	`outProtoMask`	-	A mask describing which output protocols are active. Each bit of this mask is used for a protocol. Through that, multiple protocols can be defined on a single port. (see graphic below)
16	U2	-	`reserved4`	-	Always set to zero
18	U2	-	`reserved5`	-	Always set to zero

Bitfield txReady

This Graphic explains the bits of `txReady`



Name	*Description*
`en`	Enable TX ready feature for this port
`pol`	Polarity 0 High-active 1 Low-active
`pin`	PIO to be used (must not be in use already by another function)
`thres`	Threshold The given threshold is multiplied by 8 bytes. The TX ready PIN goes active after >= thres*8 bytes are pending for the port and going inactive after the last pending bytes have been written to hardware (0-4 bytes before end of stream). 0x000 no threshold 0x001 8byte 0x002 16byte ... 0x1FE 4080byte 0x1FF 4088byte

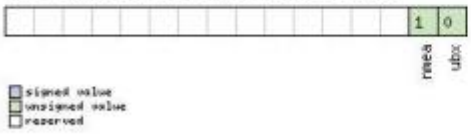
Bitfield mode

This Graphic explains the bits of mode

Name	Description
reserved1	Default 1 for compatibility with A4
charLen	Character Length 00 5bit (not supported) 01 6bit (not supported) 10 7bit (supported only with parity) 11 8bit
parity	000 Even Parity 001 Odd Parity 1XX No Parity X1X Reserved
nStopBits	Number of Stop Bits 00 1 Stop Bit 01 1.5 Stop Bit 10 2 Stop Bit 11 0.5 Stop Bit

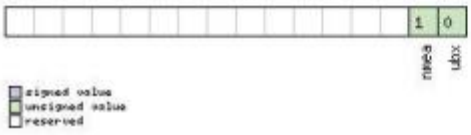
Bitfield inProtoMask

This Graphic explains the bits of inProtoMask



Bitfield outProtoMask

This Graphic explains the bits of outProtoMask



8.6 MARG

Mapa de registros del MPU 9250:

InvenSense | MPU-9250 Register Map and Descriptions | Document Number: RM-MPU-9250A-00 Revision: 1.4 Release Date: 9/9/2013

3 Register Map for Gyroscope and Accelerometer

The following table lists the register map for the gyroscope and accelerometer in the MPU-9250 MotionTracking device.

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00	0	SELF_TEST_X_GYRO	R/W	xg_st_data [7:0]							
01	1	SELF_TEST_Y_GYRO	R/W	yg_st_data [7:0]							
02	2	SELF_TEST_Z_GYRO	R/W	zg_st_data [7:0]							
0D	13	SELF_TEST_X_ACCEL	R/W	XA_ST_DATA [7:0]							
0E	14	SELF_TEST_Y_ACCEL	R/W	YA_ST_DATA [7:0]							
0F	15	SELF_TEST_Z_ACCEL	R/W	ZA_ST_DATA [7:0]							
13	19	XG_OFFSET_H	R/W	X_OFFS_USR [15:8]							
14	20	XG_OFFSET_L	R/W	X_OFFS_USR [7:0]							
15	21	YG_OFFSET_H	R/W	Y_OFFS_USR [15:8]							
16	22	YG_OFFSET_L	R/W	Y_OFFS_USR [7:0]							
17	23	ZG_OFFSET_H	R/W	Z_OFFS_USR [15:8]							
18	24	ZG_OFFSET_L	R/W	Z_OFFS_USR [7:0]							
19	25	SMPLRT_DIV	R/W	SMPLRT_DIV[7:0]							
1A	26	CONFIG	R/W	-	FIFO_MODE	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		
1B	27	GYRO_CONFIG	R/W	XGYRO_Cten	YGYRO_Cten	ZGYRO_Cten	GYRO_FS_SEL [1:0]		-	FCHOICE_B[1:0]	
1C	28	ACCEL_CONFIG	R/W	ax_st_en	ay_st_en	az_st_en	ACCEL_FS_SEL[1:0]		-		
1D	29	ACCEL_CONFIG 2	R/W	-				ACCEL_FCHOICE_B	A_DLPF_CFG		
1E	30	LP_ACCEL_ODR	R/W	-				Lposc_clksel [3:0]			
1F	31	WOM_THR	R/W	WOM_Threshold [7:0]							
23	35	FIFO_EN	R/W	TEMP_FIFO_EN	GYRO_XOUT	GYRO_YOUT	GYRO_ZOUT	ACCEL	SLV2	SLV1	SLV0
24	36	I2C_MST_CTRL	R/W	MULT_MST_EN	WAIT_FOR_ES	SLV_3_FIFO_EN	I2C_MST_P_NSR	I2C_MST_CLK[3:0]			
25	37	I2C_SLV0_ADDR	R/W	I2C_SLV0_RNW	I2C_ID_0 [6:0]						
26	38	I2C_SLV0_REG	R/W	I2C_SLV0_REG[7:0]							
27	39	I2C_SLV0_CTRL	R/W	I2C_SLV0_EN	I2C_SLV0_BYTE_SW	I2C_SLV0_REG_DIS	I2C_SLV0_GRP	I2C_SLV0_LENG[3:0]			
28	40	I2C_SLV1_ADDR	R/W	I2C_SLV1_RNW	I2C_ID_1 [6:0]						
29	41	I2C_SLV1_REG	R/W	I2C_SLV1_REG[7:0]							
2A	42	I2C_SLV1_CTRL	R/W	I2C_SLV1_EN	I2C_SLV1_BYTE_SW	I2C_SLV1_REG_DIS	I2C_SLV1_GRP	I2C_SLV1_LENG[3:0]			
2B	43	I2C_SLV2_ADDR	R/W	I2C_SLV2_RNW	I2C_ID_2 [6:0]						
2C	44	I2C_SLV2_REG	R/W	I2C_SLV2_REG[7:0]							
2D	45	I2C_SLV2_CTRL	R/W	I2C_SLV2_EN	I2C_SLV2_BYTE_SW	I2C_SLV2_REG_DIS	I2C_SLV2_GRP	I2C_SLV2_LENG[3:0]			
2E	46	I2C_SLV3_ADDR	R/W	I2C_SLV3_RNW	I2C_ID_3 [6:0]						
2F	47	I2C_SLV3_REG	R/W	I2C_SLV3_REG[7:0]							
30	48	I2C_SLV3_CTRL	R/W	I2C_SLV3_EN	I2C_SLV3_BYTE_SW	I2C_SLV3_REG_DIS	I2C_SLV3_GRP	I2C_SLV3_LENG [3:0]			
31	49	I2C_SLV4_ADDR	R/W	I2C_SLV4_RNW	I2C_ID_4 [6:0]						
32	50	I2C_SLV4_REG	R/W	I2C_SLV4_REG[7:0]							
33	51	I2C_SLV4_DO	R/W	I2C_SLV4_DO[7:0]							
34	52	I2C_SLV4_CTRL	R/W	I2C_SLV4_EN	SLV4_DONE_INT_EN	I2C_SLV4_REG_DIS	I2C_MST_DLY[4:0]				

InvenSense	MPU-9250 Register Map and Descriptions	Document Number: RM-MPU-9250A-00 Revision: 1.4 Release Date: 9/9/2013

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
35	53	I2C_SLV4_DI	R	I2C_SLV4_DI[7:0]							
36	54	I2C_MST_STATUS	R	PASS_THROUGH	I2C_SLV4_DONE	I2C_LOST_ARB	I2C_SLV4_NACK	I2C_SLV3_NACK	I2C_SLV2_NACK	I2C_SLV1_NACK	I2C_SLV0_NACK
37	55	INT_PIN_CFG	RAW	ACTL	OPEN	LATCH_INT_EN	INT_ANYRD_2CLEAR	ACTL_FSYNC	FSYNC_INT_MODE_EN	BYPASS_EN	-
38	56	INT_ENABLE	RAW	-	WOM_EN	-	FIFO_OFLOW_EN	FSYNC_INT_EN	-	-	RAW_RDY_EN
3A	58	INT_STATUS	R	-	WOM_INT	-	FIFO_OFLOW_INT	FSYNC_INT	-	-	RAW_DATA_RDY_INT
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT_H[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT_L[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT_H[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT_L[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT_H[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT_L[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT_H[15:8]							
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT_L[7:0]							
49	73	EXT_SENS_DATA_00	R	EXT_SENS_DATA_00[7:0]							
4A	74	EXT_SENS_DATA_01	R	EXT_SENS_DATA_01[7:0]							
4B	75	EXT_SENS_DATA_02	R	EXT_SENS_DATA_02[7:0]							
4C	76	EXT_SENS_DATA_03	R	EXT_SENS_DATA_03[7:0]							
4D	77	EXT_SENS_DATA_04	R	EXT_SENS_DATA_04[7:0]							
4E	78	EXT_SENS_DATA_05	R	EXT_SENS_DATA_05[7:0]							
4F	79	EXT_SENS_DATA_06	R	EXT_SENS_DATA_06[7:0]							
50	80	EXT_SENS_DATA_07	R	EXT_SENS_DATA_07[7:0]							
51	81	EXT_SENS_DATA_08	R	EXT_SENS_DATA_08[7:0]							
52	82	EXT_SENS_DATA_09	R	EXT_SENS_DATA_09[7:0]							
53	83	EXT_SENS_DATA_10	R	EXT_SENS_DATA_10[7:0]							
54	84	EXT_SENS_DATA_11	R	EXT_SENS_DATA_11[7:0]							
55	85	EXT_SENS_DATA_12	R	EXT_SENS_DATA_12[7:0]							
56	86	EXT_SENS_DATA_13	R	EXT_SENS_DATA_13[7:0]							
57	87	EXT_SENS_DATA_14	R	EXT_SENS_DATA_14[7:0]							
58	88	EXT_SENS_DATA_15	R	EXT_SENS_DATA_15[7:0]							
59	89	EXT_SENS_DATA_16	R	EXT_SENS_DATA_16[7:0]							
5A	90	EXT_SENS_DATA_17	R	EXT_SENS_DATA_17[7:0]							
5B	91	EXT_SENS_DATA_18	R	EXT_SENS_DATA_18[7:0]							
5C	92	EXT_SENS_DATA_19	R	EXT_SENS_DATA_19[7:0]							
5D	93	EXT_SENS_DATA_20	R	EXT_SENS_DATA_20[7:0]							
5E	94	EXT_SENS_DATA_21	R	EXT_SENS_DATA_21[7:0]							
5F	95	EXT_SENS_DATA_22	R	EXT_SENS_DATA_22[7:0]							
60	96	EXT_SENS_DATA_23	R	EXT_SENS_DATA_23[7:0]							
63	99	I2C_SLV0_DO	RAW	I2C_SLV0_DO[7:0]							
64	100	I2C_SLV1_DO	RAW	I2C_SLV1_DO[7:0]							
65	101	I2C_SLV2_DO	RAW	I2C_SLV2_DO[7:0]							

InvenSense	MPU-9250 Register Map and Descriptions	Document Number: RM-MPU-9250A-00 Revision: 1.4 Release Date: 9/9/2013

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
66	102	I2C_SLV3_DO	R/W	I2C_SLV3_DO[7:0]							
67	103	I2C_MST_DELAY_CTRL	R/W	DELAY_ES_SHADOW	-	-	I2C_SLV4_DLY_EN	I2C_SLV3_DLY_EN	I2C_SLV2_DLY_EN	I2C_SLV1_DLY_EN	I2C_SLV0_DLY_EN
68	104	SIGNAL_PATH_RESET	R/W	-	-	-	-	-	GYRO_RST	ACCEL_RST	TEMP_RST
69	105	MOT_DETECT_CTRL	R/W	ACCEL_INTEL_EN	ACCEL_INTEL_MODE	-		-		-	
6A	106	USER_CTRL	R/W	-	FIFO_EN	I2C_MST_EN	I2C_IF_DIS	-	FIFO_RST	I2C_MST_RST	SIG_COND_RST
6B	107	PWR_MGMT_1	R/W	H_RESET	SLEEP	CYCLE	GYRO_STANDBY	PD_PTAT	CLKSEL[2:0]		
6C	108	PWR_MGMT_2	R/W	-		DIS_XA	DIS_YA	DIS_ZA	DIS_XG	DIS_YG	DIS_ZG
72	114	FIFO_COUNTH	R/W	-			FIFO_CNT[12:8]				
73	115	FIFO_COUNTL	R/W	FIFO_CNT[7:0]							
74	116	FIFO_R_W	R/W	D[7:0]							
75	117	WHO_AM_I	R	WHOAMI[7:0]							
77	119	XA_OFFSET_H	R/W	XA_OFFS[14:7]							
78	120	XA_OFFSET_L	R/W	XA_OFFS[6:0]							-
7A	122	YA_OFFSET_H	R/W	YA_OFFS[14:7]							
7B	123	YA_OFFSET_L	R/W	YA_OFFS[6:0]							-
7D	125	ZA_OFFSET_H	R/W	ZA_OFFS[14:7]							
7E	126	ZA_OFFSET_L	R/W	ZA_OFFS[6:0]							-

Table 1 MPU-9250 mode register map for Gyroscope and Accelerometer

Note: Register Names ending in _H and _L contain the high and low bytes, respectively, of an internal register value.

In the detailed register tables that follow, register names are in capital letters, while register values are in capital letters and italicized. For example, the ACCEL_XOUT_H register (Register 59) contains the 8 most significant bits, *ACCEL_XOUT*[15:8], of the 16-bit X-Axis accelerometer measurement, *ACCEL_XOUT*.

The reset value is 0x00 for all registers other than the registers below.

- Register 107 (0x01) Power Management 1
- Register 117 (0x71) WHO_AM_I

Registros de configuración del acelerómetro y giróscopo:

4.6 Register 27 – Gyroscope Configuration

Serial IF: R/W

Reset value: 0x00

BIT	NAME	FUNCTION
[7]	XGYRO_Cten	X Gyro self-test
[6]	YGYRO_Cten	Y Gyro self-test
[5]	ZGYRO_Cten	Z Gyro self-test
[4:3]	GYRO_FS_SEL[1:0]	Gyro Full Scale Select: 00 = +250dps 01= +500 dps 10 = +1000 dps 11 = +2000 dps
[2]	-	Reserved
[1:0]	Fchoice_b[1:0]	Used to bypass DLPF as shown in table 1 above. NOTE: Register is Fchoice_b (inverted version of Fchoice), table 1 uses Fchoice (which is the inverted version of this register).

4.7 Register 28 – Accelerometer Configuration

Serial IF: R/W

Reset value: 0x00

BIT	NAME	FUNCTION
[7]	ax_st_en	X Accel self-test
[6]	ay_st_en	Y Accel self-test
[5]	az_st_en	Z Accel self-test
[4:3]	ACCEL_FS_SEL[1:0]	Accel Full Scale Select: ±2g (00), ±4g (01), ±8g (10), ±16g (11)
[2:0]	-	Reserved

Registros de configuración del ByPass. Permite acceder a través del bus I2C a los registros del magnetómetro incluidos en el encapsulado:

InvenSense	MPU-9250 Register Map and Descriptions	Document Number: RM-MPU-9250A-00 Revision: 1.4 Release Date: 9/9/2013

4.19 Register 55 – INT Pin / Bypass Enable Configuration

Serial IF: R/W

Reset value: 0x00

BIT	NAME	FUNCTION
[7]	ACTL	1 – The logic level for INT pin is active low. 0 – The logic level for INT pin is active high.
[6]	OPEN	1 – INT pin is configured as open drain. 0 – INT pin is configured as push-pull.
[5]	LATCH_INT_EN	1 – INT pin level held until interrupt status is cleared. 0 – INT pin indicates interrupt pulse's is width 50us.
[4]	INT_ANYRD_2CLEAR	1 – Interrupt status is cleared if any read operation is performed. 0 – Interrupt status is cleared only by reading INT_STATUS register
[3]	ACTL_FSYNC	1 – The logic level for the FSYNC pin as an interrupt is active low. 0 – The logic level for the FSYNC pin as an interrupt is active high.
[2]	FSYNC_INT_MODE_EN	1 – This enables the FSYNC pin to be used as an interrupt. A transition to the active level described by the ACTL_FSYNC bit will cause an interrupt. The status of the interrupt is read in the I2C Master Status register PASS_THROUGH bit. 0 – This disables the FSYNC pin from causing an interrupt.
[1]	BYPASS_EN	When asserted, the i2c_master interface pins(ES_CL and ES_DA) will go into 'bypass mode' when the i2c master interface is disabled. The pins will float high due to the internal pull-up if not enabled and the i2c master interface is disabled.
[0]	RESERVED	

4.20 Register 56 – Interrupt Enable

Serial IF: R/W

Reset value: 0x00

BIT	NAME	FUNCTION
[7]	RESERVED	

Registros de lectura de acelerómetro y giróscopo:

4.22 Registers 59 to 64 – Accelerometer Measurements

Name: ACCEL_XOUT_H

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	High byte of accelerometer x-axis data.

Name: ACCEL_XOUT_L

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	Low byte of accelerometer x-axis data.

Name: ACCEL_YOUT_H

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	High byte of accelerometer y-axis data.

Name: ACCEL_YOUT_L

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	Low byte of accelerometer y-axis data.

Name: ACCEL_ZOUT_H

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	High byte of accelerometer z-axis data.

Name: ACCEL_ZOUT_L

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	Low byte of accelerometer z-axis data.

4.23 Registers 65 and 66 – Temperature Measurement

Name: TEMP_OUT_H

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	High byte of the temperature sensor output

Name: TEMP_OUT_L

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	Low byte of the temperature sensor output: **TEMP_degC** = ((TEMP_OUT – RoomTemp_Offset)/Temp_Sensitivity) + 21degC Where Temp_degC is the temperature in degrees C measured by the temperature sensor. TEMP_OUT is the actual output of the temperature sensor.

4.24 Registers 67 to 72 – Gyroscope Measurements

Name: GYRO_XOUT_H

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	High byte of the X-Axis gyroscope output

Name: GYRO_XOUT_L

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION

BIT	NAME	FUNCTION
[7:0]	D[7:0]	Low byte of the X-Axis gyroscope output **GYRO_XOUT =** Gyro_Sensitivity * X_angular_rate Nominal FS_SEL = 0 Conditions Gyro_Sensitivity = 131 LSB/(°/s)

Name: GYRO_YOUT_H

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	High byte of the Y-Axis gyroscope output

Name: GYRO_YOUT_L

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	Low byte of the Y-Axis gyroscope output **GYRO_YOUT =** Gyro_Sensitivity * Y_angular_rate Nominal FS_SEL = 0 Conditions Gyro_Sensitivity = 131 LSB/(°/s)

Name: GYRO_ZOUT_H

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	High byte of the Z-Axis gyroscope output

Name: GYRO_ZOUT_L

Serial IF: SyncR

Reset value: 0x00 (if sensor disabled)

Registro del manejo de energía:

InvenSense	MPU-9250 Register Map and Descriptions	Document Number: RM-MPU-9250A-00 Revision: 1.4 Release Date: 9/9/2013

BIT	NAME	FUNCTION
[7]	Reserved	
[6]	FIFO_EN	1 – Enable FIFO operation mode. 0 – Disable FIFO access from serial interface. To disable FIFO writes by dma, use FIFO_EN register. To disable possible FIFO writes from DMP, disable the DMP.
[5]	I2C_MST_EN	1 – Enable the I2C Master I/F module; pins ES_DA and ES_SCL are isolated from pins SDA/SDI and SCL/ SCLK. 0 – Disable I2C Master I/F module; pins ES_DA and ES_SCL are logically driven by pins SDA/SDI and SCL/ SCLK. NOTE: DMP will run when enabled, even if all internal sensors are disabled, except when the sample rate is set to 8Khz.
[4]	I2C_IF_DIS	1 – Reset I2C Slave module and put the serial interface in SPI mode only. This bit auto clears after one clock cycle.
[3]	Reserved	
[2]	FIFO_RST	1 – Reset FIFO module. Reset is asynchronous. This bit auto clears after one clock cycle.
[1]	I2C_MST_RST	1 – Reset I2C Master module. Reset is asynchronous. This bit auto clears after one clock cycle. NOTE: This bit should only be set when the I2C master has hung. If this bit is set during an active I2C master transaction, the I2C slave will hang, which will require the host to reset the slave.
[0]	SIG_COND_RST	1 – Reset all gyro digital signal path, accel digital signal path, and temp digital signal path. This bit also clears all the sensor registers. SIG_COND_RST is a pulse of one clk8M wide.

4.34 Register 107 – Power Management 1

Name: PWR_MGMT_1

Serial IF: R/W

Reset value: (Depends on PU_SLEEP_MODE bit, see below)

BIT	NAME	FUNCTION
[7]	H_RESET	1 – Reset the internal registers and restores the default settings. Write a 1 to set the reset, the bit will auto clear.
[6]	SLEEP	When set, the chip is set to sleep mode (After OTP loads, the PU_SLEEP_MODE bit will be written here)

BIT	NAME	FUNCTION
[5]	CYCLE	When set, and SLEEP and STANDBY are not set, the chip will cycle between sleep and taking a single sample at a rate determined by LP_ACCEL_ODR register NOTE: When all accelerometer axis are disabled via PWR_MGMT_2 register bits and cycle is enabled, the chip will wake up at the rate determined by the respective registers above, but will not take any samples.
[4]	GYRO_STANDBY	When set, the gyro drive and pll circuitry are enabled, but the sense paths are disabled. This is a low power mode that allows quick enabling of the gyros.
[3]	PD_PTAT	Power down internal PTAT voltage generator and PTAT ADC
[2:0]	CLKSEL[2:0]	Code Clock Source 0 Internal 20MHz oscillator 1 Auto selects the best available clock source – PLL if ready, else use the Internal oscillator 2 Auto selects the best available clock source – PLL if ready, else use the Internal oscillator 3 Auto selects the best available clock source – PLL if ready, else use the Internal oscillator 4 Auto selects the best available clock source – PLL if ready, else use the Internal oscillator 5 Auto selects the best available clock source – PLL if ready, else use the Internal oscillator 6 Internal 20MHz oscillator 7 Stops the clock and keeps timing generator in reset (After OTP loads, the inverse of PU_SLEEP_MODE bit will be written to CLKSEL[0])

4.35 Register 108 – Power Management 2

Name: PWR_MGMT_2

Serial IF: R/W

Reset value: 0x00

BIT	NAME	FUNCTION
[7:6]	Reserved	

Mapa de registros del magnetómetro:

InvenSense	MPU-9250 Register Map and Descriptions	Document Number: RM-MPU-9250A-00 Revision: 1.4 Release Date: 9/9/2013

5 Register Map for Magnetometer

The register map for the MPU-9250's Magnetometer (AK8963) section is listed below.

Name	Address	READ/ WRITE	Description	Bit width	Explanation
WIA	00H	READ	Device ID	8	
INFO	01H	READ	Information	8	
ST1	02H	READ	Status 1	8	Data status
HXL	03H	READ	Measurement data	8	X-axis data
HXH	04H			8	
HYL	05H			8	Y-axis data
HYH	06H			8	
HZL	07H			8	Z-axis data
HZH	08H			8	
ST2	09H	READ	Status 2	8	Data status
CNTL	0AH	READ/ WRITE	Control	8	
RSV	0BH	READ/ WRITE	Reserved	8	DO NOT ACCESS
ASTC	0CH	READ/ WRITE	Self-test	8	
TS1	0DH	READ/ WRITE	Test 1	8	DO NOT ACCESS
TS2	0EH	READ/ WRITE	Test 2	8	DO NOT ACCESS
I2CDIS	0FH	READ/ WRITE	I^2C disable	8	
ASAX	10H	READ	X-axis sensitivity adjustment value	8	Fuse ROM
ASAY	11H	READ	Y-axis sensitivity adjustment value	8	Fuse ROM
ASAZ	12H	READ	Z-axis sensitivity adjustment value	8	Fuse ROM

Table 2 Register Table

Addresses from 00H to 0CH and from 10H to 12H are compliant with automatic increment function of serial interface respectively. Values of addresses from 10H to 12H can be read only in Fuse access mode. In other modes, read data is not correct.

Registro de estado del magnetómetro:

5.2 Detailed Descriptions for Magnetometer Registers

This section details each register within the MPU-9250's Magnetometer section.

5.3 WIA: Device ID

Addr	Register name	D7	D6	D5	D4	D3	D2	D1	D0
Read-only register									
00H	WIA	0	1	0	0	1	0	0	0

Device ID of AKM. It is described in one byte and fixed value.

48H: fixed

5.4 INFO: Information

Addr	Register name	D7	D6	D5	D4	D3	D2	D1	D0
Read-only register									
01H	INFO	INFO7	INFO6	INFO5	INFO4	INFO3	INFO2	INFO1	INFO0

INFO[7:0]: Device information for AKM.

5.5 ST1: Status 1

Addr	Register name	D7	D6	D5	D4	D3	D2	D1	D0
Read-only register									
02H	ST1	0	0	0	0	0	0	0	DRDY
Reset		0	0	0	0	0	0	0	0

DRDY: Data Ready

"0": Normal

"1": Data is ready

DRDY bit turns to "1" when data is ready in single measurement mode or self-test mode. It returns to "0" when any one of ST2 register or measurement data register (HXL to HZH) is read.

DOR: Data Overrun

"0": Normal

"1": Data overrun

DOR bit turns to "1" when data has been skipped in continuous measurement mode or external trigger measurement mode. It returns to "0" when any one of ST2 register or measurement data register (HXL~HZH) is read.

Registro de control del magnetómetro:

5.7 ST2: Status 2

Addr	Register name	D7	D6	D5	D4	D3	D2	D1	D0
Read-only register									
09H	ST2	0	0	0	BITM	HOFL	0	0	0
Reset		0	0	0	0	0	0	0	0

HOFL: Magnetic sensor overflow

"0": Normal

"1": Magnetic sensor overflow occurred

In single measurement mode, continuous measurement mode, external trigger measurement mode and self-test mode, magnetic sensor may overflow even though measurement data regiseter is not saturated. In this case, measurement data is not correct and HOFL bit turns to "1". When next measurement stars, it returns to "0".

BITM: Output bit setting (mirror)

"0": 14-bit output

"1": 16-bit output

Mirror data of BIT bit of CNTL1 register.
ST2 register has a role as data reading end register, also. When any of measurement data register is read in continuous measurement mode or external trigger measurement mode, it means data reading start and taken as data reading until ST2 register is read. Therefore, when any of measurement data is read, be sure to read ST2 register at the end.

5.8 CNTL1: Control 1

Addr	Register name	D7	D6	D5	D4	D3	D2	D1	D0
Read-only register									
0AH	CNTL1	0	0	0	BIT	MODE3	MODE2	MODE1	MODE0
Reset		0	0	0	0	0	0	0	0

MODE[3:0]: Operation mode setting

"0000": Power-down mode
"0001": Single measurement mode
"0010": Continuous measurement mode 1
"0110": Continuous measurement mode 2
"0100": External trigger measurement mode
"1000": Self-test mode
"1111": Fuse ROM access mode
Other code settings are prohibited

BIT: Output bit setting

"0": 14-bit output
"1": 16-bit output

Registros de lectura del magnetómetro

InvenSense	MPU-9250 Register Map and Descriptions	Document Number: RM-MPU-9250A-00 Revision: 1.4 Release Date: 9/9/2013

5.6 HXL to HZH: Measurement Data

Addr	Register name	D7	D6	D5	D4	D3	D2	D1	D0
Read-only register									
03H	HXL	HX7	HX6	HX5	HX4	HX3	HX2	HX1	HX0
04H	HXH	HX15	HX14	HX13	HX12	HX11	HX10	HX9	HX8
05H	HYL	HY7	HY6	HY5	HY4	HY3	HY2	HY1	HY0
06H	HYH	HY15	HY14	HY13	HY12	HY11	HY10	HY9	HY8
07H	HZL	HZ7	HZ6	HZ5	HZ4	HZ3	HZ2	HZ1	HZ0
08H	HZH	HZ15	HZ14	HZ13	HZ12	HZ11	HZ10	HZ9	HZ8
Reset		0	0	0	0	0	0	0	0

Measurement data of magnetic sensor X-axis/Y-axis/Z-axis

HXL[7:0]: X-axis measurement data lower 8bit
HXH[15:8]: X-axis measurement data higher 8bit
HYL[7:0]: Y-axis measurement data lower 8bit
HYH[15:8]: Y-axis measurement data higher 8bit
HZL[7:0]: Z-axis measurement data lower 8bit
HZH[15:8]: Z-axis measurement data higher 8bit

Measurement data is stored in two's complement and Little Endian format. Measurement range of each axis is from -32760 ~ 32760 decimal in 16-bit output.

Measurement data (each axis) [15:0]			Magnetic flux density [μT]
Two's complement	Hex	Decimal	
0111 1111 1111 1000	7FF8	32760	4912(max.)
|	|	|	|
0000 0000 0000 0001	0001	1	0.15
0000 0000 0000 0000	0000	0	0
1111 1111 1111 1111	FFFF	-1	-0.15
|	|	|	|
1000 0000 0000 1000	8008	-32760	-4912(min.)

Table 4 Measurement data format

8.7 APC220

8.7.1 Pinout del Módulo

Installation and Use

APC220-43 module has 9 pins. Refers to the Table 1：

APC220-43		
Pin NO.	Pin Name	Description
1	GND	Grounding of Power Supply
2	VCC	Power supply DC 3.5V-5.5V
3	EN	Power enable, ≧1.6V or empty, ≦0.5Vsleep.
4	RXD	URAT input, TTL
5	TXD	URAT output, TTL
6	MUX	The pin is expanded for other functions
7	SET	Setting parameters, setting online supportted
8	NC	Not connected
9	NC	Not connected

Table 1 Interface definition

Dimensions – PIN Assignments

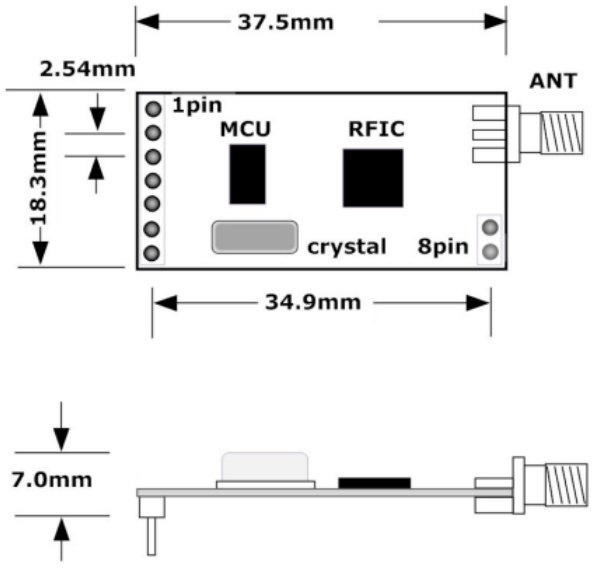


Figure 1: Size of Module

8.7.2 Parámetros de configuración

Setup parameters

With series COM or through the software Rf-Magic ,user can set up all parameters which include work frequency, UART rate, air rate,checkout mode and so on.

It is very convenient to set APC220-43 . Different options can be selected based on user needs . Please refer to Table 2 and Figure 2.

The instruction of setting parameters of module APC220-43		
Setting	options	default
UART rate	1200,2400,4800,9600b,19200,38400,57600	9600bps
Series Parity	Disable,Even Parity,Odd Parity	Disable
Frequency	418MHz-455MHz	434 MHz
Air Rate	2400bps,4800bps,9600bps,19200bps	9600bps
RF Power	0-9(9 for 20mw)	9(20mw)

Table 2 Setting Parameters of Modules

The Parameters Table:

The parameters table		
Parameters	Bytes	Instruction
Frequency	6	Unit is KHz,for example 434MHz is 434000.
Rf data rate	1	1,2,3 and 4 refer to 2400,4800,9600,19200bps separetely.
Output power	1	0 to 9, 9 means 13dBm(20mW).
UART rate	1	0,1,2,3,4,5 and 6 refers to 1200,2400,4800,9600, 19200,38400,57600bps separately.
Series checkout	1	Series checkout：0 means no check,1 means even parity,2 means odd parity.

8.7.3 Especificaciones técnicas del APC220

Specifications

The technical specifications of APC220-43:	
Work frequency	418MHz to 455MHz
Modulation	GFSK
Frequency interval	200KHz
Transmitted power	20mw (10 levels)
Received sensitivity	-113dBm@9600bps
Air rate	2400 - 19200bps
UART rate	1200 - 57600bps
The parity of series COM	8E1/8N1/8O1
The buffer of COM	256bytes
Humidity	10%～90%
Temperature	-30℃ - 85℃
Supply voltage	3.5 – 5.5V (the ripple is ±50mV)
Transmit current	≦42mA@20mW
Receiving current	≦28mA
Sleeping current	≦5uA
Transfers distance	1000m (open space)
Dimension	37.5mm x 18.3mm x 7.0mm

Cómo implementar un PID con Microcontroladores

A. Elgezabal Núñez, *BiSKY Team, Bilbao*

***Abstracto*— Este documento trata el tema de los controladores PID en microcontroladores. Se empieza con una breve resumen de la teoría detrás del control automático y los controladores PID para terminar con la programación en microcontroladores de los mismos.**

En la primera sección se hace una breve introducción de lo que es el control automático. Después, en la segunda sección se explican los fundamentos de la teoría de control moderna. En la tercera y cuarta sección se verá una aproximación intuitiva a lo que es un controlador PID y su tunning. En la quinta sección se verá la forma de implementar un PID básico así como un método eficaz de corregir uno de sus defectos más recurrentes, el wind-up. En la sexta sección se ven las conclusiones del documento y finalmente, en la séptima sección se ven una serie de referencias bibliográficas del tema tratado en este documento.

I. Introducción

El control automático es una rama de la ingeniería que se ocupa del control de un proceso en un estado determinado; por ejemplo, mantener la temperatura de una caldera, el rumbo de un cohete o la velocidad de un motor.

Los controladores automáticos comparan el valor de salida de una planta con el valor deseado (consigna) y produce una señal de control que reduce la desviación a cero, o en su defecto a un valor pequeño. La forma en que este control produce la señal de control recibe el nombre de acción de control.

Es posible clasificar en diferentes tipos de controladores de acuerdo a su acción de control:

a) Controladores Todo o Nada
b) Controladores Proporcionales (P)
c) Controladores Integrales (I)
d) Controladores Proporcionales e Integrales (PI)
e) Controladores Proporcionales y derivativos (PD)
f) Controladores Proporcionales, derivativos e integrales (PID)

En este documento se hará especial énfasis en los controladores PID.

II. Introducción a teoría de control

Aquí se intentará hacer algo que no es tan frecuente de encontrar, **dar una visión intuitiva de la teoría de control** y de la motivación que hay detrás, de forma que sea fácilmente comprensible, y sin tener que usar transformadas de Laplace, lugares de las raíces, diagramas de Bode, ni teoremas de Nyquist.

A. El sistema

Se denomina sistema a una representación simplificada de la realidad que es abstraída para poder trabajar con él. Normalmente se representa una parte de la realidad que aislamos.

Se puede considerar sistema a casi cualquier cosa. La presión de una caldera, el sistema de climatización de un coche, el equipo de bombeo de un depósito, los controles de un avión, la propulsión de una nave espacial, un robot... hasta el sistema digestivo de un pato.

Un sistema, al que normalmente se le hará referencia como planta, **tiene una o varias entradas** que representan acciones sobre las que se puede actuar. También **tiene una o varias salidas** que son los parámetros que se pueden observar y medir. En general un sistema tiene muchas entradas y salidas, pero normalmente se van a simplificar a las más importantes.

La planta, además, tiene un cierto comportamiento. Es decir, **realizar una acción tendrá una determinada consecuencia sobre sus salidas**. Por supuesto, el comportamiento puede ser todo lo complicado que se pueda imaginar. Puede ser no lineal, tener inercias, incluso retrasos puros (un tiempo entre que se aplica la acción y el sistema actúa), efectos entre varias entradas.

El comportamiento de la planta también puede ser abstraído y modelado de forma simplificada, en lo que se conoce como función de trasferencia. Y se puede representar, por ejemplo, mediante una ecuación, un diagrama de bloques, una representación gráfica o frecuencial.

Además la **planta puede tener perturbaciones**, es decir, modificaciones en el entorno que modifican su comportamiento. Por ejemplo, en el edificio la temperatura exterior va a cambiar, en el coche se puede subir o bajar una cuesta, y en el embalse se puede poner a llover.

En resumen, **un sistema es un modelo de una porción del universo, que tiene entradas y salidas, cuya relación entre éstas sigue un determinado comportamiento y que está sometido a posibles perturbaciones**.

.

Y esto se suele representar así:

Figure 1: Representación de un sistema

B. El controlador

El controlador es **un elemento que se pone antes de las entradas de nuestra planta para actuar sobre ellas** con objeto de conseguir que la salida tenga unas determinadas características.

Por ejemplo, si se quiere controlar la temperatura de un edificio actuando sobre la caldera, la velocidad del coche actuando sobre el motor, o el nivel del depósito actuando sobre el bombeo.

Lógicamente, **la entrada sobre la que se va a actuar debe tener una influencia en la salida que se quiere controlar**. Si se está controlando una variable que es totalmente independiente de lo que se está midiendo, no se conseguirá nada.

El controlador puede ser desde una persona, hasta un sistema analógico, o un sistema digital. En sistemas automáticos lógicamente se hace referencia a algún tipo de máquina controlada por el sistema de forma autónoma sin intervención manual.

El controlador, representado junto al sistema, quedaría así.

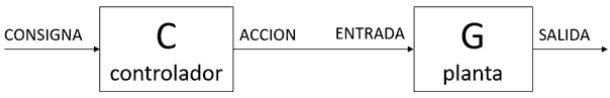


Figure 2: Representación de Controlador y sistema

C. Características a la salida

Lo más evidente **es que la salida tenga un valor determinado** que se denomina consigna. Por ejemplo, se quiere que la temperatura sea de 24ºC, que el coche vaya a 80km/h, o que el nivel del agua de un embalse sea 175m.

Por supuesto, **la consigna no tiene que ser la misma siempre**. Se puede querer que la temperatura sea 17ºC durante la noche y 24ºC durante el día, que el coche pase a 50km/h al pasar por una travesía, o querer vaciar a 160m embalse porque tengo que suministrar agua potable a la red.

Pero, además de la consigna, hay **otras características deseadas para la salida**. En general, se va a querer que la salida sea estable a largo plazo, que converja al valor de la consigna, que el tiempo de respuesta sea rápido, que no oscile, que no sobrepase el valor de consigna en ningún momento.

No se va a poder tenerlas todas a la vez, así que se tiene que **llegar a un compromiso entre todas**. Aunque, llegado el caso puede, que se esté dispuesto a sacrificar unas respecto a otras.

Por ejemplo, quizás no importa que la temperatura de un edificio alcance brevemente 24.5ºC, si así el tiempo de respuesta es menor. Pero si a 180m de agua el embalse se rompe, inundando un pueblo y matando a cientos de personas, quizás no sea tolerable el desbordamiento y se prefiera que suba el nivel lento pero seguro.

Por tanto, **las características deseadas para la salida dependen totalmente del sistema y de lo que se quiera hacer**, y por tanto el controlador que se tiene que emplear.

D. Lazo Cerrado

Para que un controlador funcione tiene que tener realimentación. Básicamente **el controlador tiene que poder monitorizar la salida que intenta controlar** directa o indirectamente.

Sin realimentación, un controlador no es un controlador.

Está claro que para controlar una salida hay que poder medirla. Así que se **coge la salida y se compara con la consigna para ver el error** que hay. Se emplea este error como entrada del controlador. Esto se expresa así.

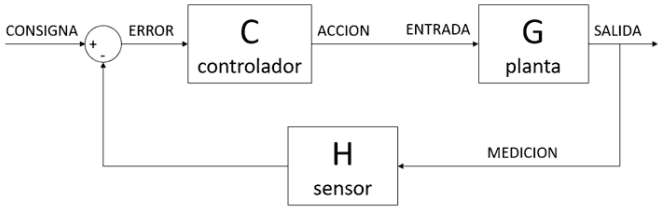


Figure 3 Representación de un Lazo Cerrado (L.C.)

El error entre la medición y la consigna puede ser bien porque aún no se ha conseguido que la salida alcance a la consigna, o porque se ha cambiado la consigna.

Con realimentación el sistema global (controlador + planta) se comporta de forma totalmente distinta. La salida puede desde tender obedientemente a la consigna, a ponerse a oscilar como una loca hasta que rompa algo.

Y la diferencia entre una salida (éxito absoluto en el sistema de control) y la otra (rotura, despido, y posible destrucción del universo) depende totalmente del control que se diseñe.

III. Aproximación intuitiva al Controlador PID

En este capítulo se **verá el controlador PID**, uno de los controladores más extendidos por su sencillez y por ser capaz de conseguir un buen comportamiento en una gran variedad de situaciones.

En el capítulo anterior se vieron algunas generalidades y definiciones sobre teoría de sistemas.

Por supuesto, no se va a entrar en profundidad en los detalles del PID, dado que es tema muy extenso (e interesante). Si existe interés se puede consultar la abundante documentación disponible.

En este capítulo se intenta dar una **visión intuitiva del controlador y de la motivación** que explica su comportamiento. Sin usar ecuaciones, ni matemáticas. De hecho, es una de las cosas geniales del PID, que no es necesario saber en detalle cómo funciona, para hacer que funcione.

A. ¿Qué es el PID?

El controlador PID es **uno de los más empleados en la industria** para el control de sistemas realimentados. Algunas de sus fortalezas son su sencillez y que es capaz de dar un **buen comportamiento en una gran variedad de situaciones** sin necesidad de conocer con detalle la planta a controlar.

El controlador PID es conocido desde hace tiempo. Sus primeros usos datan de 1911, y su primer análisis teórico de 1922 de la mano de Nicolas Minorsky. En esos tiempos, el control PID era exclusivamente analógico. Sin embargo, resulta fácil implementar en programación un PID digital y los cálculos que requiere son sencillos y eficientes. Además, es "relativamente sencillo" extrapolar la teoría de los PID analógicos a su equivalente digital.

Pese a su popularidad, hay que decir que actualmente el PID no es el mejor controlador disponible. Pero en la mayoría de los casos es más que suficiente. Por otro lado, muchos de los controladores más "modernos" no dejan de ser versiones mejoradas de un PID, como por ejemplo las distintas familias basadas en PID con parámetros adaptativos.

B. Funcionamiento del PID

El algoritmo PID (proporcional, integral, derivativo) está formado por la **suma de tres componentes, Proporcional, Integral, y Derivativo**. Matemáticamente, un controlador PID tiene la siguiente formulación.

$$Output(t) = K_p \cdot e(t) + K_i \int_0^t e(t) \cdot dt + K_d \cdot \frac{de}{dt}$$

Cada componente del PID es "independiente" de los demás, en el sentido de que cada uno calcula una salida de lo que "para él" debería hacer para obtener la respuesta adecuada.

Los tres componentes se suman para dar la salida del controlador. Cada uno cumple una cierta función y mejoran cierta parte de la respuesta. Y cuando los tres componentes trabajan juntos, en la proporción adecuada, consiguen un gran comportamiento.

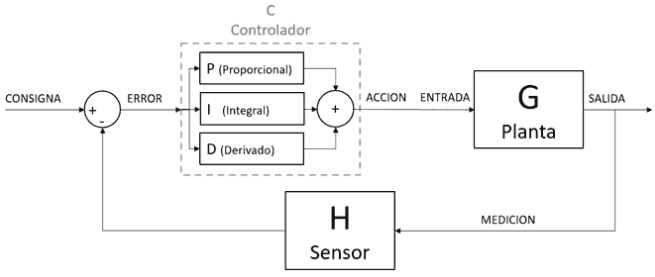


Figure 4: PID en un Lazo Cerrado

Cada componente tiene un parámetro Kp, Ki y Kd, respectivamente. Estos parámetros indican la ponderación (o "la fuerza") que tiene en el resultado final. Que la respuesta del PID sea buena, cosas rotas, muerte y destrucción, **reside en el ajuste correcto de estos tres parámetros**.

Y aquí viene la parte "graciosa" del PID. En los tres parámetros, si se pone un valor muy bajo no se va a notar el efecto del componente en la salida. Y si se pone demasiado alto… pues eso, muerte, destrucción, cosas rotas, etc.

Es más, en la respuesta global del controlador los tres componentes trabajan juntos e influyen uno sobre otros, por lo que **no vale con ajustar cada uno de los parámetros de forma independiente**. Existe una cierta "zona" dentro de los tres parámetros, donde el comportamiento es más o menos bueno.

Lógicamente, queda claro que la dificultad (que tampoco es para tanto) de un PID es ajustar los parámetros K, Ki y Kd, para que el comportamiento sea bueno.

Como se ha explicado, el controlador PID se basa en tres componentes, PID. Su fortaleza reside en el papel que cada uno de estos componentes tiene en la respuesta. A modo resumen:

- El componente proporcional reacciona **al presente.**
- El componente integral reacciona **al pasado**, y aporta "memoria" al controlador.
- El componente derivado reacciona **al futuro**, y aporta "predicción" al controlador

C. Componente Proporcional – P

1) Motivación

La motivación del componente proporcional seguramente sea el más intuitivo de explicar. Si se entra en la habitación y ven 12ºC, parece lógico que hay que darle más a la calefacción que si pone 23ºC.

Pues esa es la motivación del componente proporcional, que tiene sentido que **se aplique más acción cuanto más lejos está del valor deseado**, y viceversa.

2) Formulación

El componente proporcional se calcula simplemente como un factor K multiplicado por el error (diferencia entre consigna y valor real).

3) Comportamiento

El factor proporcional tiene una gran influencia en la **velocidad de respuesta del sistema**. Si se tiene una K pequeña, el sistema va a tardar mucho en alcanzar la consigna, porque le está aplicando poco al actuador. Si se aumenta, se consigue disminuir el tiempo de respuesta.

Pero si aumenta la K demasiado se puede **sobrepasar la consigna u oscilar**.

Si se tiene una K demasiado grande, se le va a dar demasiado a la calefacción. Así al ver 18ºC, se le da a la calefacción y el siguiente valor es 25. se ha pasado. Baja la calefacción, 21, ahora se ha pasado hacia abajo. Sube la calefacción, 23. Bien, ya lo tiene.

Pero si lleva una K aún más grande, podría pasar que al ver 18º le dé MUCHO a la calefacción, y lo siguiente que se vea podría ser 32ºC. ¡Ostras, se he pasado una barbaridad, BAJA LA CALEFACCIÓN! 14º ¡OSTRAS SUBE SUBE! 45º... 8º... 56º... Acaba de hacer que el sistema oscile y sea inestable.

Otra característica es que, en general, el componente proporcional **no elimina por completo el error a largo plazo**. En este ejemplo, supongamos que a 22ºC y una "posición" de la calefacción determinada, la energía que aportamos al edificio es exactamente la que necesita el edificio para mantener la temperatura. Se habría conseguido que la temperatura sea estable, pero como la "posición" de la calefacción viene dada por el error (que se ha estabilizado) nunca se podrá subir los 2ºC que faltan.

D. Componente Integral – I

1) Motivación

La función del componente integral quizás sea el más complicado de explicar dentro del PID. Imaginar que se está en la sala, y hay 22ºC. El proporcional dice que se ponga la calefacción en una "posición" y el controlador lo pone ahí. Pasa un minuto, pasa otro, pasa otro... y eso sigue en 22ºC, 2ºC por debajo de los que se quiere.

El componente integral es el que dice... ¿oye chicos, igual hay que subir un poco la calefacción no? Porque llevamos un buen rato en 22ºC sin mover la calefacción, y eso no tiene pinta de que vaya a irse. Pues esa voz con tanto sentido, **que reacciona a la memoria del error pasado**, es el componente integral.

2) Formulación

El componente integral aplica una acción que es proporcional a la integral del error a lo largo del tiempo. Equivalentemente, responde proporcionalmente a la suma de todos los errores anteriores.

Gráficamente corresponde con el área encerrado bajo la curva del error, que también es lo mismo que el área entre la salida y la consigna. En el campo discreto, la integral se sustituye por un método discreto para su cálculo, como el cálculo mediante rectángulos o trapecios.

3) Comportamiento

El componente integral permite al controlador **eliminar por completo el error a largo plazo**. Sin embargo, si Ki es muy pequeño, el sistema tardará mucho en eliminar el error.

Por otro lado, hay que tener en cuenta que el término integral únicamente responde al área entre la salida y la consigna. Una consecuencia de ello es que si hemos acumulado (por ejemplo) un error por estar por debajo de la consigna, la única forma que tiene el término integral de compensarlo es estar un tiempo por encima.

Y efectivamente, **el componente integral tiene tendencia a sobrepasar y oscilar**, más incluso que el término proporcional.

E. Componente Derivativo – D

1) Motivación

El componente derivado también es sencillo de explicar. Suponiendo que se está tranquilamente con la temperatura a la consigna a 24ºC. ¡Todo perfecto! La temperatura baja a 23ºC, luego a 22ºC, y se sube un poquito la palanca. Todo va tal y como se espera.

Ahora imaginando que, cuando se estaba a 24ºC la temperatura baja a 18ºC de golpe. Ostras, eso es una buena bajada. Se subes la calefacción. Marca 12ºC... Han abierto una ventana, pero ¡La temperatura sigue bajando muy rápido! Siguiente valor 2ºC.

Ésa es la función del componente derivado, reaccionar ante variaciones del error (por cambio de consigna o de la variable). Porque no es lo mismo estar a 12ºC y subiendo lentamente, que si la temperatura se está desplomando a toda velocidad. Es decir, el término derivativo **responde a la velocidad de cambio del error**.

2) Formulación

El componente derivado se calcula de forma proporcional a la derivada del error respecto del tiempo en el momento presente. Equivalentemente, responde proporcionalmente a la diferencia entre el error actual y el error en el instante anterior.

En el campo discreto, la derivada se sustituye por la diferencia entre el error actual y el error anterior, dividido por el tiempo de muestreo (o se obvia esta división por completo y se incluye en el coeficiente Kd, si el tiempo de muestreo es constante).

3) Comportamiento

El componente derivado **mejora la respuesta general de muchos sistemas** para valores de Kd moderados. Sin embargo, si se excede de Kd se verá que aparece una falta de "suavidad" en la respuesta, y otros comportamientos "raros".

Además, el componente derivado **responde muy mal al ruido de la medición**. El ruido, una variación de alta frecuencia, supone variaciones muy rápidas, aunque sean de pequeña amplitud. Estas variaciones son amplificadas por el componente derivado.

Y, por último, el componente derivado a veces es un poco "bestia" porque **pide acciones muy grandes**. Imaginar, por ejemplo, un cambio instantáneo en la consigna. El componente derivado demandaría una acción infinita que el accionador no podría satisfacer. Como el accionador no podría dar la acción que solicita el controlador, se tendrían desviaciones respecto a lo calculado, o incluso se podría dañar el accionador.

IV. TUNNING DEL PID

En función del valor que se ponga a los parámetros, se obtendrá una buena respuesta, o una respuesta lenta, u oscilante, o incluso ... muerte, destrucción.

Para hacerlo más complicado, el efecto de los factores en la respuesta no es independiente. Su aportación se mezcla y entremezcla. Por lo que **no se pueden ajustar los parámetros de "uno a uno"**.

El éxito del controlador **depende totalmente del ajuste**. Para ello, es importante que se conozcan las características y efectos de cada factor en la respuesta del sistema.

A. Resumen de los efectos del PID

1) Componente Proporcional

- Poco K, respuesta lenta.
- Mucho K, desbordamiento, oscilación, e incluso inestabilidad.
- No consigue eliminar el error en estacionario.

2) Componente Integral

- Elimina el error estacionario
- Demasiado Ki, oscilación e inestabilidad

3) Componente Derivativa

- Mejora el comportamiento general
- Demasiado Kd, comportamiento "raro" en la salida
- Muy sensible al ruido
- Muy sensible a cambios bruscos en el error (perturbaciones o cambio de consigna)

B. *Ajustando el PID*

Existen muchas formas de ajustar un PID, algunas más o menos teóricas.

1) *Caracterización de la planta*

Consiste en determinar la función de transferencia de la planta. Prácticas habituales son aplicar una entrada escalón o frecuencial, y analizar las características de la salida.

Después se puede operar matemáticamente para realizar un cálculo exacto del controlador para la respuesta requerida.

2) *Aplicación de reglas de sintonización*

De forma similar, consiste en aplicar una determinada entrada al sistema, y medir la salida. A continuación, se aplican una de las muchas reglas de sintonización (ejemplo famoso, reglas Ziegler-Nichols) para obtener los parámetros del PID.

3) *Auto Tunning*

En similar al anterior, pero realizado de forma automática por el controlador. Los reguladores PID han avanzado mucho y, en la actualidad, muchos tienen estupendos algoritmos de auto ajuste.

Precisamente, esta capacidad de auto ajuste es uno de los motivos del éxito comercial de los controladores PID.

4) *Ajuste manual (Heurístico)*

Finalmente, está el ajuste "a mano". Que, aunque suene fatal, es una opción muy válida y muy popular. En muchas ocasiones, un operador humano puede conseguir un ajuste tan bueno (o incluso superior) a un Auto tuning genérico.

C. *Ajuste manual del PID*

Para ajustar el PID manualmente se debe poder actuar sobre la entrada del sistema, sobre los parámetros del PID, y visualizar la respuesta. Y aquí, "visualizar" no significa necesariamente poner un osciloscopio. En el caso de un motor, por ejemplo, se puede "ver" la respuesta del sistema simplemente mirando cómo gira (o cómo suena). Hay dos métodos principalmente:

1) *Ajuste Proporcional/Integral/Derivativo*

- Ajustar K hasta que el sistema sobre pase o empiece a sobre oscilar.
- Ajustar Ki hasta eliminar error estacionario.
- Aumentar Kd hasta que empiezan a pasar cosas raras.

2) *Ajuste Proporcional/Derivativo/Integral*

- Ajustar K hasta que el sistema sobre pase o empiece a sobre oscilar
- Aumentar Kd hasta que empiezan a pasar cosas raras
- Ajustar Ki hasta eliminar error estacionario

Con los dos se van a conseguir resultados similares, en último término. *El autor* ***aconseja el primero***.

El motivo es que el término derivativo es muy sensible al ruido de la medición y, por tanto, es difícil de ajustar en un caso real.

V. Implementar un Controlador con Microcontroladores

Ahora toca dejar la teoría, desempolvar el teclado y **ver cómo se implementa un controlador PID** en un microcontrolador. Afortunadamente, es bastante sencillo hacer un PID básico.

A. *PID a Mano*

Primero se va a ver **cómo implementar un controlador sencillo**. En realidad, el código no es excesivamente complejo. Supongamos que el controlador toma la entrada de la variable controlada en A0, y efectúa la salida mediante una señal PWM en el pin 3.

El código de un PID básico podría ser el siguiente:

```
// Asignaciones pins
const int PIN_INPUT = A0;
const int PIN_OUTPUT = 3;

// Constantes del controlador
double Kp=2, Ki=5, Kd=1;

// variables externas del controlador:
double Input, Output, Setpoint;

// variables internas del controlador:
unsigned long currentTime, previousTime;
double elapsedTime;
double error, lastError, cumError, rateError;

void setup()
{
  Input = analogRead(PIN_INPUT);
  Setpoint = 100;
}

void loop()
{
  pidController.Compute();

  // leer una entrada del controlador
  Input = analogRead(PIN_INPUT);
```

```
  // calcular el controlador
  Output = computePID(Input);

   delay(100);

    // escribir la salida del controlador
   analogWrite(PIN_OUTPUT, Output);
}

double computePID(double inp)
{
     // obtener el tiempo actual
     currentTime = millis();

      // calcular el tiempo transcurrido
     elapsedTime = (double)(currentTime - previousTime);

     // determinar el error entre la consigna y la medición
     error = Setpoint - Input;

     // calcular la integral del error
     cumError += error * elapsedTime;

     // calcular la derivada del error
     rateError = (error - lastError) / elapsedTime;

     // calcular la salida del PID
     double output = kp*error + ki*cumError + kd*rateError;

      // almacenar error anterior
     lastError = error;

     // almacenar el tiempo anterior
     previousTime = currentTime;

     return output;
}
```

Como se puede observar, no es demasiado difícil. Hay una función *computePID()* que realiza todo el trabajo. En esta función se calcula el tiempo transcurrido entre llamadas, que es necesario para calcular tanto la derivada como la integral del error.

A continuación, **se compara la entrada del controlador con la consigna para determinar el error** y se realizan las 'pseduo' integrales y derivadas (su equivalente en discreto). Finalmente, se calcula la respuesta del sistema mediante la fórmula del PID, y se guardan los valores para el siguiente ciclo.

Sin embargo, aunque es completamente funcional, **este controlador PID es bastante simple**. Por tanto, tiene ciertas carencias ante situaciones que ocurren frecuentemente en la realidad.

B. *Reset wind-up*

Reset windup es una trampa en la que caen muchos principiantes. Ocurre cuando el PID cree que está haciendo algo que no puede. Por ejemplo, un PWM que acepta valores entre 0 y 255. El PID por defecto no sabe esto. Si cree que los valores 300, 400, 500 funcionarán, probará esos valores esperando conseguir lo que necesita. Dado que en realidad el valor está saturado a 255 seguirá intentando valores cada vez más alto sin llegar a ninguna parte.

El problema se presenta en forma de retrasos raros.

C. *La Solución*

1) *Primer paso*

Existen diversas maneras para mitigar el Wind-up, pero la forma más intuitiva es indicarle al PID cuáles son los límites de salida. En el código que se ve más adelante hay una function SetOutputLimits. Cuando uno de los limites es alcanzado, el PID deja de sumar (Integrar).

2) *Segundo paso*

Hay aún diferencia entre lo que el PID cree que está enviando y lo que en realidad envía. Esto es debido al componente Proporcional y (en menor medida) al termino derivativo.

Aunque el componente integral ha sido saturado de forma segura, P y D están aún contribuyendo al wind-up. Esto puede ser inaceptable en según que aplicación. Así que como segundo paso se satura el valor de salida también.

```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; //1 sec
double outMin, outMax;
void Compute()
{
  unsigned long now = millis();
  int timeChange = (now - lastTime);
  if(timeChange>=SampleTime)
  {
    /*Compute all the working error variables*/
    double error = Setpoint - Input;
    ITerm+= (ki * error);
    if(ITerm> outMax) ITerm= outMax;
    else if(ITerm< outMin) ITerm= outMin;
    double dInput = (Input - lastInput);

    /*Compute PID Output*/
    Output = kp * error + ITerm- kd * dInput;
    if(Output > outMax) Output = outMax;
    else if(Output < outMin) Output = outMin;

    /*Remember some variables for next time*/
    lastInput = Input;
    lastTime = now;
  }
```

```
}

void SetTunings(double Kp, double Ki, double Kd)
{
 double SampleTimeInSec =
((double)SampleTime)/1000;
  kp = Kp;
  ki = Ki * SampleTimeInSec;
  kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
  if (NewSampleTime > 0)
  {
    double ratio  = (double)NewSampleTime
            / (double)SampleTime;
    ki *= ratio;
    kd /= ratio;
    SampleTime = (unsigned long)NewSampleTime;
  }
}

void SetOutputLimits(double Min, double Max)
{
  if(Min > Max) return;
  outMin = Min;
  outMax = Max;

  if(Output > outMax) Output = outMax;
  else if(Output < outMin) Output = outMin;

  if(ITerm> outMax) ITerm= outMax;
  else if(ITerm< outMin) ITerm= outMin;
}
```

VI. Conclusiones

En este documento se ha visto de manera resumida e intuitiva lo que es la teoría de control moderna y el papel que cumple en ella el PID.

Se ha aprendido a hacer el tunning heurístico de un PID sin conocer la función de transferencia del sistema a controlar (planta).

Posteriormente, se ha aprendido a programar un PID básico y se ha conocido uno de los errores más recurrentes en estos PIDs, el wind-up.

Finalmente, se ha visto un método sencillo para eliminar éste error y el razonamiento detrás de él.

Referencias

[1] Katsuhiko Ogata, "Ingeniería de control moderna 5ª Edición", 2010.
[2] Oscar Barambones, "Sistemas Digitales de Control", 2004.
[3] Brett Beauregard, **"**Improving the Beginner's PID – Introduction" http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/
[4] Luis Llamas, "Introducción a la teoría de controladores en Arduino" https://www.luisllamas.es/introduccion-a-la-teoria-de-controladores-en-arduino/
[5] Luis Llamas, "Teoría de control en Arduino: El Controlador PID" https://www.luisllamas.es/teoria-de-control-en-arduino-el-controlador-pid/
[6] Luis Llamas, "Cómo ajustar un controlador PID en Arduino" https://www.luisllamas.es/como-ajustar-un-controlador-pid-en-arduino/
[7] Luis Llamas, "Cómo implementar un controlador PID en Arduino" https://www.luisllamas.es/como-implementar-un-controlador-pid-en-arduino/