Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Aplicación Electrónica para UAV: integración de IMU y GPS.

Autor: Manuel Bravo Escudero

Tutora: Juana María Martínez Heredia

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Aplicación Electrónica para UAV: integración de IMU y GPS

Autor:

Manuel Bravo Escudero

Tutora:

Juana María Martínez Heredia

Dpto. de Ingeniería Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2015

Trabajo Fin de Grado: Aplicación Electrónica para UAV: integración de IMU y GPS

Autor: Manuel Bravo Escudero
Tutora: Juana María Martínez Heredia

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Sevilla, 2015

Resumen

La gran expansión que han sufrido en los últimos años los vehículos aéreos no tripulados ha cambiado las reglas del juego en lo que se conocía hasta el momento del entorno aeronáutico. Tanto a nivel de sistemas y de aplicaciones como a nivel de normativa se han producido modificaciones y la aparición de una serie de nuevos recursos aplicables a estos sistemas. La consecuencia de estos cambios ha sido la creación de nuevas líneas de desarrollo en el ya de por sí complicado y en constate cambio mundo de la aeronáutica.

Una de las dificultades que se encuentra alguien externo a las compañías desarrolladoras de estos productos es la falta de información disponible en el dominio público. Los avances punteros en este sector no se detallan en su totalidad debido a la gran competencia que existe entre las empresas. Todo esto es la clave que actúa como motivación de nuestro trabajo, moviéndonos a profundizar más en la aviónica que se aplica en los UAVs.

Siendo conscientes de lo rápido que avanzan y han avanzado estos desarrollos en los últimos tiempos, se ha considerado oportuno realizar un breve estudio previo del estado del arte en el que nos encontramos actualmente. Como el resultado que se tiene en el presente es el fruto de todo lo ocurrido en el pasado, comenzaremos comentando el desarrollo de estos sistemas desde su primera aparición. Esto permitirá entender por qué estos sistemas evolucionaron lentamente a lo largo de 100 años y han experimentado un sorprendente impulso que los ha hecho llegar, incluso, hasta el ámbito doméstico. En el presente por tanto contamos con un amplio rango de sistemas a nuestro alcance en los que nos podemos basar para justificar nuestros desarrollos posteriores. Por ello se continuará exponiendo la jerarquía de la vasta variedad de diferentes vehículos no tripulados que se conocen, para obtener una visión más clara de todas las clases que se pueden encontrar en la realidad. Todo ello no sería posible sin los equipos de aviónica que realizan ciertas funciones en los UAVs propias de la tripulación. En este punto podremos afirmar que se comprende el estado actual de estos sistemas y que se entiende la criticidad que la aviónica supone para un UAV, por lo que se procede a analizar sistemas reales que las compañías ofrecen. Para ello se ha realizado un estudio de mercado, desglosando las características de los diferentes sistemas. Realizado este paso, se comprobará que una parte vital de la electrónica aplicada son los sensores de posicionamiento, tanto inerciales como GPS, apareciendo en la inmensa mayoría de los mismos. Ya que no podemos abarcar todo el espectro de sistemas de aviónica, decidimos centrar nuestro estudio en estos instrumentos tan necesarios para la autonomía de dichos vehículos. Y

como no es lo mismo quedarse en la teoría que pasar a la práctica, se decidió construir un prototipo que fusionase una unidad de medidas inerciales con un receptor de señales GPS, permitiéndonos conocer su funcionamiento más en profundidad. La plataforma que nos permitirá construir nuestro prototipo, ofreciéndonos la suficiente flexibilidad y potencia como para poder integrar ambos dispositivos será Arduino. En concreto se empleará un Arduino Due, que cuenta con unas características óptimas para nuestro objetivo

Como ejemplo de IMU para nuestro estudio se ha seleccionado la placa *10-DOF IMU Breakout - L3GD20H + LSM303 + BMP180* de Adafruit. Este instrumento agrupa tres acelerómetros en los tres ejes, tres giróscopos en los tres ejes, tres magnetómetros en los tres ejes, un sensor de temperatura y un sensor de presión. Para su correcto funcionamiento la compañía ofrece una serie de códigos compatibles con Arduino que permiten extraer los datos de cada sensor por separado y otros que integran dichos datos para obtener información más interesante a la hora del posicionamiento. Finalmente se seleccionará, para la integración posterior, un código que devuelve los ángulos que definen el rumbo del vehículo, y otro que extrae las aceleraciones que captan los acelerómetros.

El otro pilar de nuestro proyecto será el receptor de GPS *Ultimate GPS Breakout* de la compañía Adafruit igualmente. El objetivo de este instrumento será recibir las sentencias NMEA (protocolo de comunicación empleado) que emiten los satélites, y el código que ofrece Adafruit se encarga de depurarlas para conseguir al final datos legibles. Este código será el que posteriormente se integrará con los indicados anteriormente para obtener el prototipo deseado.

Finalmente se detallarán los pasos que se han seguido para completar la fusión de ambos dispositivos, obteniendo un equipo que consigue emitir tanto los datos inerciales como el posicionamiento GPS coordinadamente. Con esto se ha conseguido obtener un sistema a partir de dos, ahorrando en espacio, tanto físico como digital.

Con todo ello hemos podido incrementar nuestros conocimientos sobre la electrónica que emplean estos dispositivos, ya que son necesarios para comprender correctamente su funcionamiento, al igual que aquellos relacionados con la programación en el lenguaje empleado, como paso necesario para realizar la correcta integración del software, y con todo ello conocer más en profundidad la aviónica que se aplica en los UAVs.

Finalmente, se exponen algunas líneas de mejoras futuras que se podrían aplicar como continuación del desarrollo llevado a cabo en el presente Trabajo de Fin de Grado.

Índice

Índice de figuras.

Índice de tablas

1 Introducción

Es una realidad que los vehículos aéreos no tripulados conocidos como UAVs han experimentado un aumento de popularidad y facilidad de desarrollo en los últimos años. Ya no solo es posible encontrarlos en el ámbito militar, si no que algunos están destinados a ayudarnos en nuestro día a día. Por todo ello son uno de los focos de estudio más importantes que nos podemos encontrar en el sector aeroespacial.

Su aparición ha transformado muchos de los criterios que se consideraban a la hora de diseñar un vehículo aéreo. Entre ellos se encuentran los siguientes:

- La supresión de la tripulación a bordo del vehículo obliga a instalar multitud de equipos que desarrollen las mismas funciones que debe realizar un humano. Incluso en las aeronaves tripuladas han sido instalados multitud de dispositivos que facilitan ciertas tareas al personal en cuestión, aumentando la autonomía de la aeronave, aunque no se otorga total libertad de operación a la electrónica, a pesar de que la tendencia es que esta vaya en aumento.

- Los UAVs han ampliado considerablemente el rango de aplicaciones que es posible realizar con vehículos voladores. Tareas repetitivas son cómodas de realizar y además es más sencillo obtener un vehículo programado para realizar siempre las mismas operaciones. También tareas de alto riesgo debido a entornos bélicos o arduos pueden ser llevadas a cabo sin poner en peligro vidas humanas. Este hecho ha incrementado el interés que se tiene hacia ellos, permitiendo que se invierta más en desarrollar las características involucradas.

- Además, la normativa es diferente para los vehículos tripulados y no tripulados. La segunda resulta ser bastante menos estricta, ya que cuando el vehículo toma vuelo no se ponen en riesgo tanto factores como ocurre con el otro grupo. Por ello se permite que se tenga una actitud de innovación constante y más rápida, ya que existen menos factores limitantes a la hora de introducir nuevos elementos.

Con esto se intenta ilustrar que estos vehículos han abierto un nuevo campo de desarrollo en todos los aspectos que involucran. Y entre ellos cabe destacar un ámbito que desarrolla uno de los papeles más fundamentales de estos sistemas: la aviónica, la que podemos considerar la fuente de la que surge la verdadera autonomía que los caracteriza.

Sin embargo, el entorno de constante desarrollo que vivimos en la actualidad genera que las compañías mantengan como confidenciales los detalles de sus desarrollos más recientes. Los avances de hace cinco años pueden considerarse desactualizados, lo que ilustra la situación que pretendemos mostrar. Si esto no fuese así, bastaría con estudiar y copiar lo que sí se nos muestra para desarrollar reproducciones de los dispositivos que deseásemos. Esta situación nos deja en una posición de conocimiento poco profundo del tema, teniendo que tomar una parte activa en esta cuestión, debido a lo difícil que resulta encontrar información de esta índole en libros o recursos digitales.

Por todo ello se propuso este trabajo, para estudiar la aviónica que hoy en día se aplica en los UAVs. Como es necesario conocer el proceso que se ha seguido hasta alcanzar el punto actual, también se cree conveniente repasar la historia de los UAVs, analizando cuales han sido los momentos fundamentales que impulsaron la creación de mejoras. Tras esto se cree conveniente estudiar sistemas que se encuentren actualmente en el mercado, para tomarlos como referencia para los siguientes pasos de nuestro trabajo.

Una vez que consigamos una buena base teórica sobre la actualidad de los UAVs, y debido a que tomar parte en el asunto y pasar a la práctica permite mayor involucración en el tema, nos hemos marcado como objetivo construir un prototipo de dispositivo de aviónica a partir de varios elementos más simples. Como se justificará más adelante, unas de las partes fundamentales en todo sistema de posicionamiento son los sensores. Existen dos categorías de navegación:

- Navegación a estima: tratan de hallar la posición de la aeronave conociendo las condiciones iniciales y datos que ofrecen ciertos sensores autónomos, como aquellos que miden propiedades inerciales.

- Navegación por posicionamiento: obtienen, mediante la información que reciben de sistemas no embarcados, la posición absoluta en los ejes geográficos.

Estas dos categorías son básicas en los vehículos aéreos de hoy en día, por lo que decidimos emplear para nuestro proyecto una unidad de medidas inerciales y un receptor de GPS. Ayudándonos de una placa Arduino, nuestro objetivo final será integrar ambos instrumentos al nivel de la recepción de los datos, coordinando la información de ambos.

Finalmente se pretenderá desarrollar la manera en que estos datos pueden ser extraídos de manera realista. A expensas de mejorar la presentación

física del prototipo y el procesamiento de datos llevado a cabo, creemos que así resultará un dispositivo perfectamente presentable en el mercado

2 Sistemas UAV

“Airplanes are now built to carry a pilot and a dog in the cockpit. The pilot's job is to feed the dog, and the dog's job is to bite the pilot if he touches anything."

Arlen Rens, a Lockheed Martin test pilot.

2.1 Introducción a los UAVs

A pesar de que recientemente los conocidos como UAVs han sufrido su expansión más impactante, esta categoría de vehículos aéreos no tripulados existe desde hace más tiempo del que uno cabe esperar, aunque si es verdad que hasta hace poco el estado de su tecnología estaba muy poco desarrollada. En nuestros tiempos, sin embargo, podemos encontrarnos UAVs destinados únicamente al ocio, e incluso, como juguetes. Esto es reflejo de lo que han avanzado los sistemas, pudiendo abaratar hasta este punto los mencionados dispositivos.

Las siglas “UAV” proceden de su denominación en inglés: *Unmanned Aerial Vehicles*, aunque debido a su reciente *“boom”* la manera estandarizada en la que podemos referirnos a ellos no ha sido establecida aún. Así, podemos encontrarnos con nombres como UMAs (Unmanned Air Vehicles), RPV (Remote Piloted Vehicles), drones o UCAVs (Uninhabited Combat Air Vehicles).

El último de los nombres se corresponde lógicamente con aquellos drones destinados a tener un papel ofensivo en el campo militar. Y es al sector militar de la aeronáutica al que la humanidad debe la mayoría de los avances que en su día fueron punteros y, por tanto, aquellos que lo son ahora mismo y se comienzan a emplear en la aviación civil. Esto es debido a que la normativa militar no es tan estricta como la que se aplica al otro ámbito, de modo que a la hora de implementar un nuevo sistema, la primera deja más margen.

Como veremos más adelante, las guerras han sido las primeras impulsoras de todo tipo de tecnología, contribuyendo de esta manera al sector

aeroespacial. Y es dentro de este sector donde se enmarcan los UAVs, y por consecuencia, también han sido desarrollados durante estos periodos. No es difícil imaginarse el interés que puede despertar un dispositivo que puede ser guiado automáticamente o de forma remota, hasta la zona del otro bando, y allí desarrollar la misión para la que está destinado, ya sea ofensiva, de vigilancia, etc. De esta manera, un trabajo puede desarrollarse sin poner en peligro la vida de la persona destinada a tripular el vehículo. Sin embargo, durante las dos guerras mundiales, la tecnología no estaba tan desarrollada como lo está hoy en día.

Esto nos lleva a pensar que no es casualidad el auge que los vehículos no tripulados están sufriendo. Desde el momento en que se vio que su construcción era completamente factible desde el punto de vista tecnológico, que podían ser dedicados a múltiples aplicaciones y que el precio final era muy asequible, han surgido multitud de empresas que ofrecen pequeños drones destinados al ocio, sin nada más necesario que el dinero para realizar su compra (un dron más grande y pesado estará dedicado a aplicaciones más serias, prácticamente en ningún caso recreativas). Estos a su vez suelen integrar una cámara o algún otro instrumento de funcionalidades diferentes. O simplemente son objetos de aeromodelismo con bajos requerimientos para su control. La ventaja que ofrecen es una forma rápida, sencilla y barata de alcanzar mayores alturas y disfrutar del vuelo estable.

En el campo civil también existen drones que realizan misiones, básicamente de supervisión y monitorización. Entre ellas podemos destacar la vigilancia de fronteras ante el tráfico de drogas y la inmigración ilegal, inspección de líneas de potencia eléctrica, supervisión de viaductos de gas o petróleo, monitorización de entornos forestales, fumigación de cultivos y supervisión de tráfico y entornos urbanos. En este caso, los drones ayudan a realizar tareas repetitivas o de largo alcance. De este modo, que la tarea sea repetitiva, facilita su diseño. Y su capacidad para volar permite que cubran grandes distancias de la manera más rápida y eficiente posible. Los únicos impedimentos que deben resolver son los de carácter legislativo y de seguridad.

Sin embargo, los mayores avances no se realizan en vehículos destinados a ser vendidos en el mercado civil. Como ya se dijo, esto ocurre en el ámbito militar, ya que las tareas a desarrollar son inesperadas y requieren de gran capacidad de adaptación y decisión. Ya sean sistemas de mejora de la estabilidad, sistemas de guerra electrónica, armamento, etc., existen líneas de investigación en cada uno de ellos para estar a la altura de los últimos avances. Una de las ventajas de su aplicación en este sector, consecuencia de la eliminación del piloto, es la posibilidad de aumentar la cantidad de *g's* a las que la aeronave puede estar sometida. El piloto establece una limitación

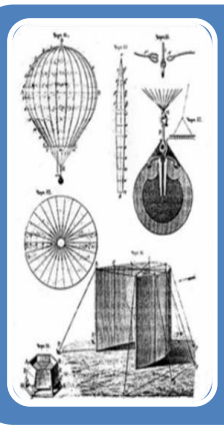
de 9*g*, por lo que un UAV puede aumentar su maniobrabilidad y su capacidad de escape ante ataques. Además, el vehículo acaba siendo más pequeño y barato, ya que al ser suprimido el habitáculo del piloto, parte de este espacio puede ser ocupado por otros sistemas. A modo de ejemplo, se estima que un UCAV supone una tercera parte del coste total de una nave de combate pilotada, y el coste de operación y mantenimiento resulta del 25% menor.

Excepto pequeñas variaciones, todos ellos poseen una arquitectura de su hardware muy similar, ya que aunque no todos están dedicados para lo mismo, todos precisan de unos requisitos mínimos: el vuelo debe ser estable en todo momento, y es complicado que esto se consiga desde la estación de tierra; la comunicación con esta estación debe efectuarse por medio de determinados canales y siguiendo determinados protocolos para que sea correcta; las órdenes que el controlador desea dar deben llegar al dron, ser recibidas adecuadamente y reinterpretadas para traducirse en deflexiones de actuadores o lo que corresponda; debe controlarse en todo momento la batería de la que se dispone para asegurarse de que el dron no pare en pleno vuelo, con los daños que esto desencadenaría. Así podríamos continuar hasta completar el conjunto de funcionalidades básicas que el dron precisaría, y que finalmente se materializaría en un sistema de dispositivos hardware fácilmente identificables en la mayoría de UAVs.

Ligado inexorablemente a esta configuración compleja, aparece el conjunto de tecnologías conocidas como aviónica. Dada la experiencia que se tiene en la aplicación de diferentes tipos de sistemas (hidráulico, neumático, etc.), la aplicación de la electrónica ha demostrado ser la única que hace posible el desarrollo de todas las funciones necesarias con el mínimo coste y peso. Sería impensable fabricar un dispositivo que realice todas las funciones que desempeña uno electrónico actual, pero accionado mediante potencia hidráulica, por ejemplo.

2.2 Reseña histórica

Para hacernos una idea de lo que son los UAVs en la actualidad y el mérito que supone disponer de un vehículo en el aire, cuyo vuelo y misión sean completamente autónomos, es indispensable conocer la trayectoria que se ha seguido hasta llegar al punto en que nos encontramos actualmente. Para ello se desarrolla la línea temporal siguiente:



Globos Austriacos, 22 Agosto 1849

- Uno de los primeros registros que se tienen de algo parecido a un UAV es el ataque por parte de Austria a la ciudad de Venecia con globos cargados de explosivos, no tripulados. Aunque el concepto de UAV no es claro en este ejemplo, tuvo el suficiente impacto para alentar a desarrollar aviones no tripulados cuando esto fue posible.

Primera Guerra Mundial

- Es durante este periodo que apareció el primer avión de la armada americana sin piloto. Conocido como *"flying bomb"*, pretendía ser un *"torpedo aéreo"* o la primera versión de los actuales *"cruise missiles"*. Su control se basada en giróscopos.

Periodo entre guerras

- Durante este periodo se relizaron numerosos avances. Conversión de aviones a UAVs que volaban con pilotos automáticos, aparición de los primeros misiles de crucero, estudio del control mediante ondas de radio... Durante este periodo apareció la denominación de *"dron"*.

Segunda Guerra Mundial

- Reginald Denny construye la primera factoría que produce drones radiocontrolados para cualquier propósito. Consigue demostrar sus aplicaciones militares y firma un acuerdo con el ejército americano.
- La armada estadounidense continúa desarrollando sus drones, haciéndolos cada vez mayores y mejorando sus prestaciones.
- La aparición del pulsorreactor facilita la implementación de la propulsión de los UAVs.

Guerra Fría

- Evolución de los drones de blanco, empleados como señuelos, apareciendo infinidad de modelos debido al avanzado estado de las tecnologías. De hecho, muchos podían alcanzar velocidades supersónicas (Mach 2).
- Algunos de ellos fueron modificados para recolectar datos de radioactividad y pruebas nucleares.
- El éxito de los drones como blancos los impulsó a ser usados como vehículos de reconocimiento. Una de sus más importantes pruebas fue la Guerra de Vietnam.
- La URSS comenzó a desarrollar sus propios prototipos, pero mantuvo estos proyectos en secreto.

Era Moderna

- La guerra entre Siria e Israel cambió por completo la visión que se tenía de los drones, en la que los israelíes derribaron numerosos aviones sirios con muy pocas pérdidas, empleando drones con alta inteligencia militar. Esto les dio importantes perspectivas.
- Los EEUU están muy centrados en desarrollar avanzados drones miitares.
- Aparición de drones de larga autonomía y microUAVs.
- Uso de nuevas fuentes de energía, como haces de láser o microondas, o energía solar.

Como podemos observar, y como se ha dicho con anterioridad, la mayor parte de los desarrollos asociados a los UAVs se debe al impulso tanto económico como táctico que se les dio y se les sigue dando en periodos bélicos. Sus comienzos, sus primeras aplicaciones, sus grandes logros, todos ellos están fuertemente ligados al sector militar. No es casualidad, ya que son innumerables las ventajas que se les pueden encontrar, que ya fueron comentadas anteriormente.

Desde el punto de vista tecnológico, al igual que ocurre con el resto del mundo de la aeronáutica, los impulsos de sus diseños están muy limitados por la tecnología del momento. Debido a que son máquinas que involucran una gran cantidad de aspectos de la ciencia, se tuvo que esperar a que todos ellos se encontrasen en el punto adecuado de su desarrollo para conseguir resultados óptimos. Véase por ejemplo los desarrollos que se tuvieron que hacer desde el punto de vista de las comunicaciones remotas para que pudiesen ser controlados de forma idónea. Del mismo modo, tuvieron que ser desarrollados novedosos sistemas propulsivos (entre ellos

ha sido destacado el pulsorreactor) cuya relación $^{Potencia}/_{Peso}$ fuese la adecuada para ser implementado en dichos vehículos.

2.3 Clasificación de los UAVs

Como se puede deducir hasta el momento, entre las grandes ventajas que los UAVs tienen se encuentran la posibilidad de aplicarlos a múltiples propósitos, su gran dinamicidad y su baja limitación por factores humanos (limitaciones del entorno, de *g*s, de periodos de vuelos grandes, tareas repetitivas, etc.). Por ello es importante hacerse a la idea de los diferentes tipos que existen respecto a diferentes clasificaciones, para poder tener conocimiento de la amplia gama que se conoce en la actualidad. Ello posibilitará comparar las diferentes categorías, ya que para diferentes propósitos, las características no pueden ser las mismas. Además es interesante conocer diferentes clasificaciones, ya que la gran expansión que los UAV han tenido recientemente, trascendiendo del mundo militar a una situación mucho más cercana a la vida cotidiana, ha acabado resultando en una mayor variedad de modelos.

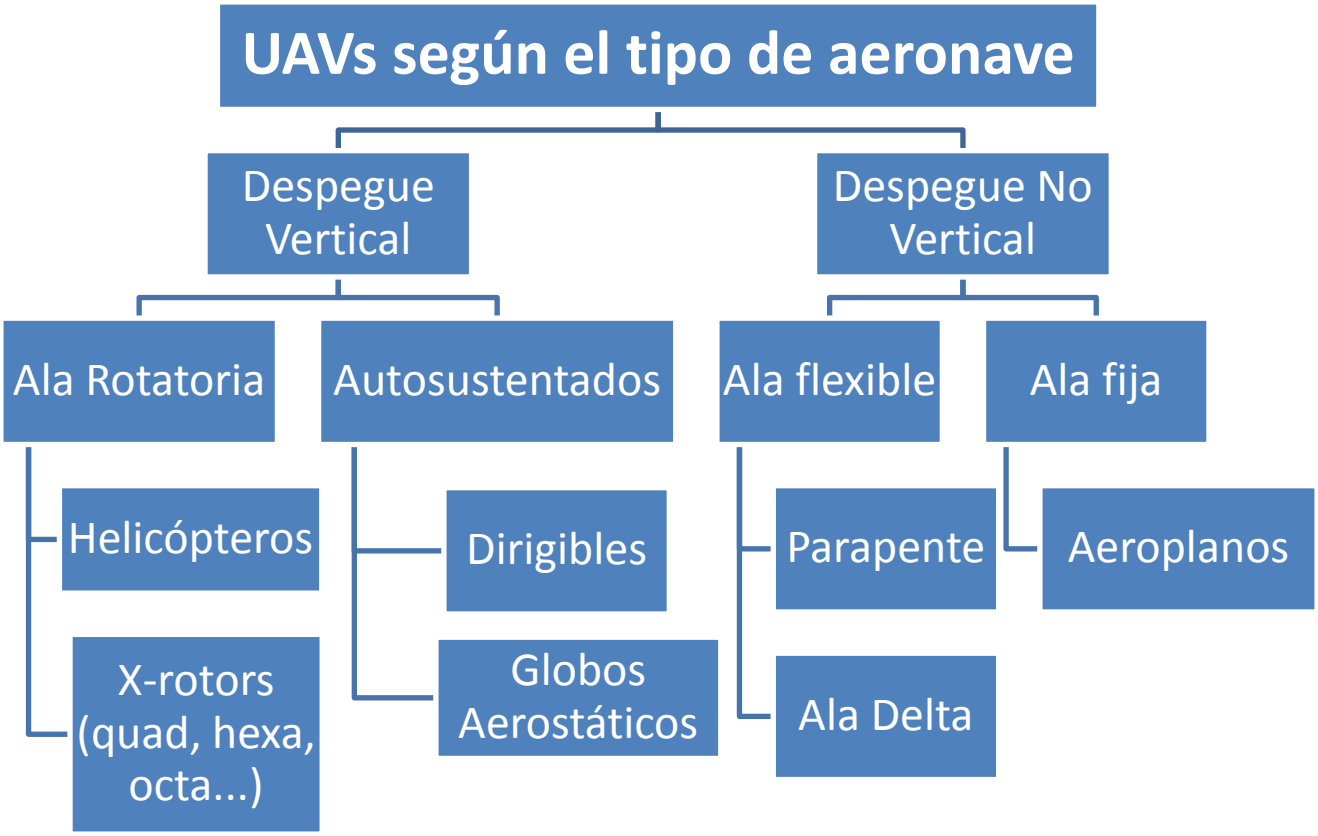


Figura 1. Clasificación de los UAVs según el tipo de ala empleada

Para empezar, la clasificación más simple que existe, que enmarca los UAVs según el tipo de ala que empleen. Se incluyen todos aquellos que

realizan una misión sin necesidad de llevar tripulación a bordo, poseyendo una controlabilidad total o parcial. De esta manera, globos meteorológicos, que no son controlables, quedan al margen. Esta clasificación se muestra en la Figura 1.

De este modo podemos comprobar que, a pesar de que los tipos más usuales de UAVs son los aeroplanos y los clasificados como ala rotatoria, pueden darse muchos más casos de los que uno podría imaginarse. Llegados a este punto, podría decirse que todo vehículo que alguna vez ha sido pilotado y ha conseguido realizar un vuelo controlado, puede modificarse para convertirse en UAV, cada vez con menos esfuerzo gracias a la experiencia que se tiene en ellos. Hay que comentar que los globos que se incluyen en el apartado de vehículos auto-sustentados sí poseen cierta controlabilidad. Un caso conocido es el denominado vehículo *Tandem*, sustentado por globos estratosféricos, zona donde se desarrolla su misión, y siendo controlado por hélices específicamente diseñadas para este entorno.

Otra manera más detallada e interesante de clasificar los UAVs es según sus capacidades de vuelo, es decir, alcance, autonomía, altitud de vuelo, etc. Debido al deseo reciente de estandarizar y regular las distintas posibilidades que permiten los actuales UAVs, la OTAN ha realizado la clasificación, mostrada en la Tabla 1.

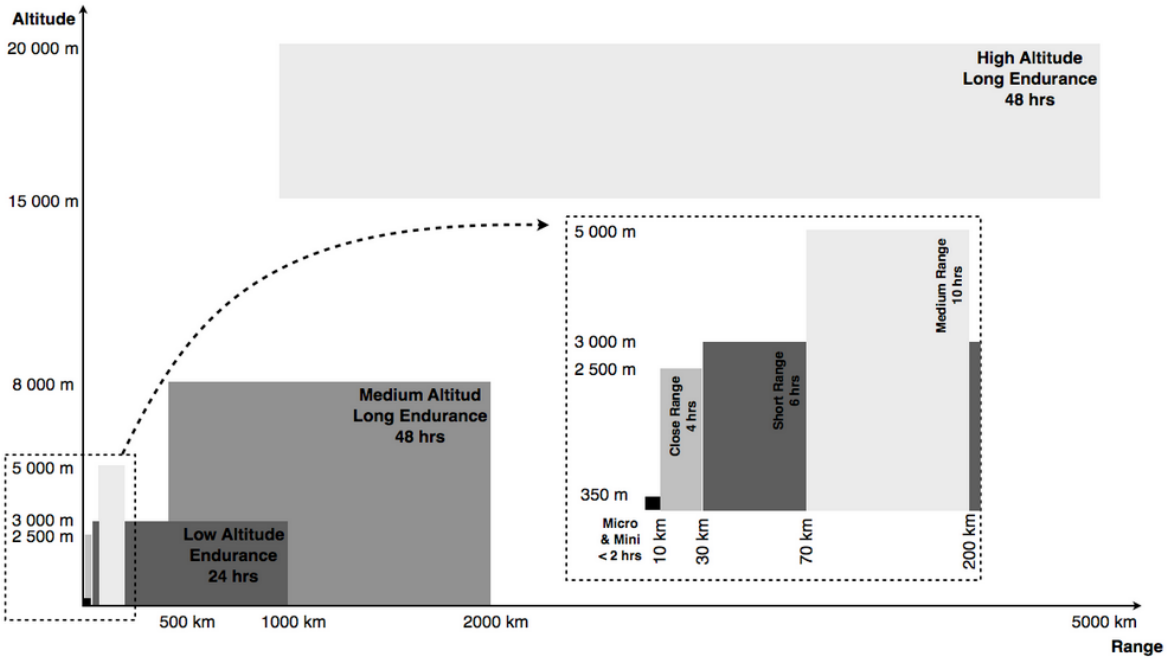


Figura 2. Clasificación de Blyenburgh

Clase	Categoría	Aplicación	Altitud	Radio de Misión	Ejemplo
Clase I: menos de 150kg	**SMALL**: >20kg	Unidad Táctica (sistema de lanzamiento)	Hasta 5000 ft	50 km	
	MINI: 2-20kg	Sub-unidad táctica (lanzamiento manual)	Hasta 3000 ft	25 km	
	MICRO: <2kg	Táctico, individual (un solo operador)	Hasta 200 ft	5 km	
Clase II: entre 150kg y 600kg	**TACTICAL**	Formación táctica	Hasta 10000 ft	200 km	
Clase III: más de 600kg	**Strike/ Combat**	Estratégico/ Nacional	Hasta 65000 ft	Ilimitado	
	HALE	Estratégico/ Nacional	Hasta 65000 ft	Ilimitado	
	MALE	Operacional	Hasta 45000 ft	Ilimitado	

Tabla 1. Clasificación de los UAVs según sus capacidades de vuelo

Resulta interesante igualmente la Figura 2, (*http://apeironuav.org/?p=205*), que expresa de forma visual y comparativa dónde se sitúa cada vehículo respecto a las características comentadas. La OTAN se ha basado en dicho

gráfico para realizar su clasificación, introduciendo como variable adicional el peso en despegue.

Se han indicado en la tabla posibles aplicaciones que podrían otorgarse a cada tipo de UAV. Lógicamente, aquellos con un peso y dimensiones mayores no suelen desarrollar tareas relacionadas con lo cotidiano, debido a los requisitos que muestran. Aquellos que se embarcan dentro de este rango serían los de la zona superior de la tabla, mientras que los de la zona inferior están pensados para aplicaciones militares y civiles complejas.

2.4 Aviónica en los UAVs

En un UAV actual, se podría decir que la aviónica lo es casi todo. La palabra aviónica procede del término inglés *Avionics* que surge como contracción de las palabras *Aviation* y *Electronics*, de modo que hace referencia al uso de la electrónica en todos los campos posibles de un vehículo aéreo. Es tal su importancia que el coste de la aviónica de una aeronave puede variar entre el 30% para el caso de un avión militar o civil moderno, hasta el 75% para un AWACS (Airborne Warning and Control System).

Cuando la aviónica se comenzó a aplicar y se vio que podía emplearse para ayudar a la tripulación en sus tareas, el siguiente paso a dar fue inmediato: reducir el número de personal en el avión, ya que se podía sustituir su trabajo por este tipo de sistemas. Si continuamos de esta manera, es fácil darse cuenta de que el punto final de este proceso desemboca en los UAVs, donde ninguna persona es necesaria a bordo para el desarrollo de la misión del vehículo.

A pesar de que otras fuentes de potencia podrían haber sido aplicadas para realizar las funciones que realiza la aviónica en el avión, la electrónica ha sido la única capaz de demostrar que consigue los resultados más eficientes. Por lo que la hegemonía que la electrónica posee en el entorno aeroespacial está más que justificada. Además, la electrónica de un sistema en tierra y la electrónica que se considera como aviónica difieren en gran manera, por lo que no es aleatorio que la aviónica, aunque muy ligada con la electrónica, forme una rama de estudio independiente de ella. Y existen varias razones por lo que esto es así:

- *La importancia de conseguir el mínimo peso*: en aeronáutica existe un efecto de apalancamiento debido al peso innecesario que se estima del orden de 10:1; es decir, un ahorro en el peso de la carga de pago de 10kg equivale a un ahorro del peso total de la aeronave de 100kg. Esto se debe a que un incremento de peso que se tiene que transportar está ligado a un aumento del peso de la estructura que

tiene que soportarlo. Además, para realizar la misión con mayor peso, debemos incluir más combustible para generar el incremento de empuje necesario. Este proceso puede entenderse como un ciclo vicioso. Por tanto, podemos deducir por qué se realizan tantos esfuerzos en reducir el peso de los sistemas del avión.

- *Requerimientos ambientales*: este problema es más severo y recurrente en aplicaciones militares, aunque en el entorno civil puede llegar a igualarse. Podemos comenzar hablando de los grandes gradientes de temperatura que los equipos deben soportar, las vibraciones a las que estarán sometidos o del número de *g*'s que el avión sufrirá durante las maniobras (9*g*'s en el caso de cazas modernos, que exigen al equipo soportar hasta 20). La compatibilidad electromagnética es un factor muy limitante, ya que estos equipos deben ser prácticamente insensibles ante un gran rango de frecuencias externas (como las tormentas eléctricas).

- *Altos niveles de seguridad, confianza e integridad*: ante cualquier fallo, estos equipos no pueden ser reparados en vuelo, por lo que se requieren altos niveles de estos conceptos para evitar todos los errores y sobrecostes que podría acarrear la pérdida de uno de ellos.

- *Restricciones de espacio*: muy importante en aviones militares, donde se hace especial énfasis en la miniaturización de cualquier elemento y la compacidad de alta densidad.

Como se trató de ejemplificar previamente, los UAVs son el resultado de ampliar el concepto de autonomía de una aeronave, por lo que no son una categoría completamente al margen de la aviación tripulada. Por esta razón, la mayor parte de la aviónica aplicable a una aeronave tripulada puede ser implementada en un UAV. De hecho, algunos elementos *hardware* están diseñados para poder ser implementados en ambas categorías. Sin embargo, existen algunas diferencias que deben ser comentadas. A continuación se describen los campos más importantes de la aviónica en general, destacando lo relevante de cada uno de ellos:

- *Displays e interacción hombre-máquina*: obviamente un dispositivo HUD (*Head Up Display*) no es necesario en un UAV; sin embargo, los elementos *Head Down Displays* son necesarios para el personal de supervisión en la estación de tierra desde donde se controle el vehículo, o estación en una nave de base.

- *Aerodinámica y Control de Vuelo*: las funciones de transferencia matemáticas que determinaban la respuesta del vehículo en función de las deflexiones de las superficies de control son idénticas tanto para

naves tripuladas como autónomas. Sin embargo, un UAV puede ser inestable aerodinámicamente, debido a la maniobrabilidad extra que se consigue con estas configuraciones.

- *Fly-by-Wire flight control*: los UAVs llevan inherentemente la tecnología *fly-by-wire* (sistema en que la interacción con las superficies de control y demás dispositivos se realiza a través de señales eléctricas generadas al mover las palancas del piloto), de modo que todo lo aplicable a aeronaves tripuladas es equivalente para UAVs. Una característica a destacar es que en ambos campos se exigen *Flight Control Systems* que superen fallos; sin embargo, en los UAVs la probabilidad de fallo para que el sistema sea aceptado puede ser algo mayor. La ausencia de la tripulación hace que la ley sea más permisiva en este sentido.

- *Sensores inerciales*: en UAVs los giróscopos de fibra óptica y los sensores más punteros ofrecen muchas ventajas en términos de costes, seguridad, actuación, peso, robustez y disponibilidad, resultando interesantes para estos vehículos.

- *Sistemas de Navegación*: en cuanto a comunicaciones con satélites tipo GPS, estas han permitido un gran avance en el posicionamiento tanto de naves tripuladas como no tripuladas: sin embargo, su real impacto será percibido en el futuro, ya que aún queda trabajo por hacer. Estas comunicaciones, además, pueden ser interferidas y alteradas. Respecto a la navegación TRN (*Terrain Reference Navigation*), esta ha tenido un mayor impacto en los UAVs, permitiéndolos volverse autónomos por completo e impidiendo cualquier tipo de infiltración externa. Esta navegación también cuenta con mejoras en la precisión.

- *Sistema de Datos de Aire*: este punto es idéntico en ambos campos, ya que es vital para el guiado del vehículo aéreo.

- *Autopilotos y Sistemas de Gestión del Vuelo*: si los autopilotos se han vuelto imprescindibles en aeronaves tripuladas, aún más en UAVs. En cuanto a los Sistemas de Gestión del Vuelo, estos se integran hasta en el más pequeño de los UAVs, ya que son los que muestran las directrices de la misión.

- *Sistemas de Aviónica Integrados*: los UAVs son los primeros candidatos para desarrollar e implementar nuevas tecnologías de aviónica integrada, así como la explotación de sistemas COTS (“Comercial Off-The-Shelf”) tanto hardware como software.

2.5 Estudio de Mercado.

Una vez que conocemos algunas relevantes clasificaciones de UAVs y hemos resaltado el papel tan vital que juegan los sistemas de aviónica en los vehículos bajo estudio, es momento de estudiar qué nos encontramos en la realidad. Para ello, se ha realizado una búsqueda de empresas que ofertan sistemas de aviónica, así como ejemplos de UAVs reales de los que disponemos información sobre su configuración electrónica.

Existe un gran número de componentes diferentes de aviónica embarcados en un UAV: sistemas de gestión de la carga de pago, computadores de vuelo, controladores de interfaz de comunicaciones, computadores de control de los motores, unidad de gestión de energía, etc. Sin embargo, por motivos relacionados con el enfoque de este trabajo, nos centraremos en aquellos sistemas relacionados con el control de vuelo, ligados íntimamente con los sistemas de navegación. Los elementos más representativos de lo que estamos hablando son los conocidos autopilotos. Pero también se encuentran otras configuraciones en la práctica.

2.5.1 Robocóptero ganador del International Aerial Robotic Competition (1997).

Este dispositivo fue diseñado para participar en el concurso indicado, para una misión de reconocimiento de material radiactivo y diversos propósitos adicionales. Como características generales, poseía una carga en vacío de 6.8kg y podía llevar una carga de pago de 4.08kg. La arquitectura de sus componentes de aviónica se muestra en la Figura 3.

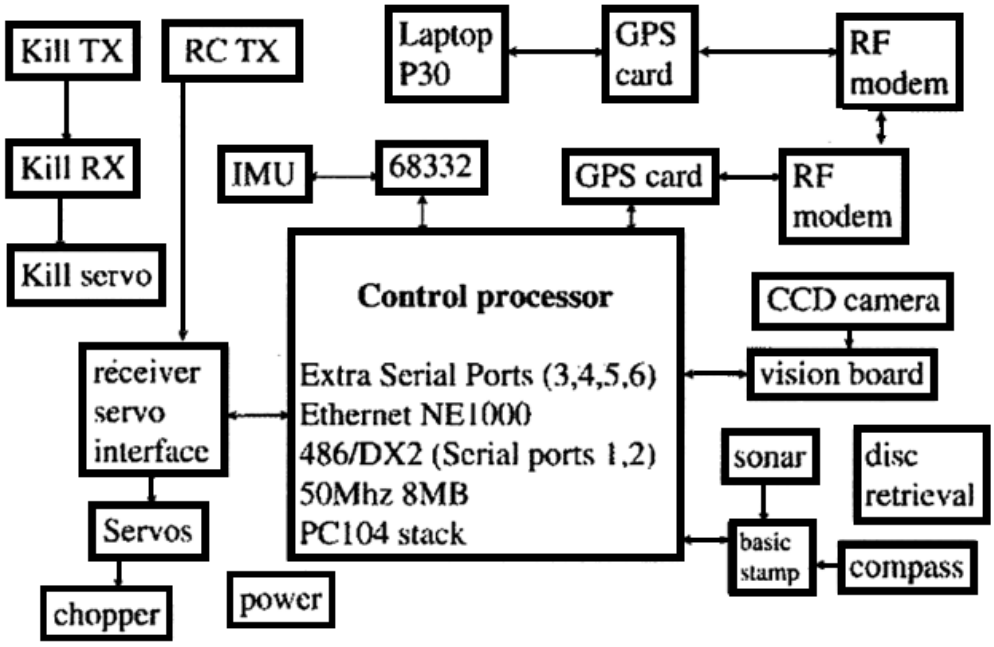


Figura 3. Diagrama de bloques del Robocóptero ganador del International Aerial Robotic Competition.

Podemos comprobar que posee comunicación GPS para posicionamiento absoluto, una cámara que le permite navegación mediante imagen, un sonar para reconocimiento de obstáculos, una brújula, la interfaz con los servos, comunicaciones remotas, la unidad de medidas inerciales y en el centro el controlador, que cuenta con comunicación Ethernet.

2.5.2 Vehículo de Vigilancia (MIT, 1996).

Este vehículo fue desarrollado con el objetivo de constituir un Wide Area Surveillance Projectile (WASP) que responda a la necesidad de una respuesta más rápida en la vigilancia de grandes zonas. Su alcance es de aproximadamente 15 km. Podemos observar la estructura de su hardware en la Figura 4.

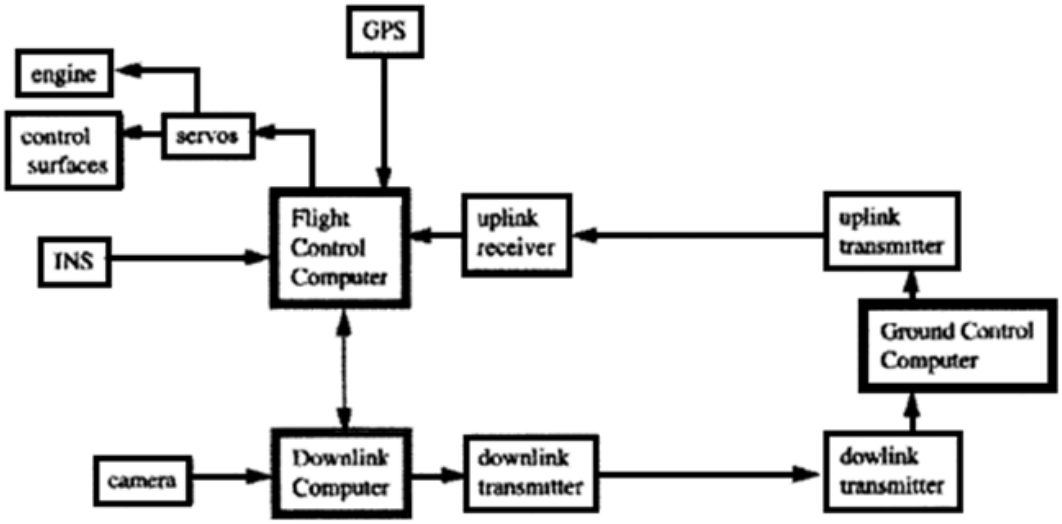


Figura 4. Diagrama de bloques del vehículo de vigilancia desarrollado por el MIT

Como muestra la imagen, posee un computador de control del vuelo como núcleo del sistema. Este está directamente comunicado con los servos, a los que envía instrucciones, y el motor. También se comunica con la unidad de medidas inerciales (INS) y el receptor GPS, por lo que se cubre la navegación inercial y absoluta. También cuenta con una cámara, y sistemas de comunicaciones con la estación de tierra. Además, se desea destacar que las comunicaciones se efectúan a través de una red Ethernet.

Todos estos sistemas consumen una potencia de 19.64 W, siendo aquellos que más consumen, la CPU (4.90W), los sistemas encargados de la comunicación Ethernet (2.00 W), los puertos serie (2.00 W) y el receptor GPS (1.40 W). Aunque sea el cuarto, podemos concluir que el GPS es de los sistemas que más consumen, suponiendo el 28.57% de lo que consume la CPU (muy elevado si comparamos la criticidad de ambos sistemas) y un 7.13% del consumo total. Es interesante estudiar el consumo de cada

sistema, ya que el ahorro del mismo es un requisito muy buscado en la actualidad.

2.5.3 Innovative Defense Electronics (RADA)

RADA es una empresa israelí que ofrece una gama muy variada de componentes de aviónica, desde minúsculos componentes para pequeños UAVs, hasta aviónica para aeronaves pesadas. En este caso, nos centramos en un conjunto de componentes compactos destinados a UAVs clasificados como MALE y HALE. Están denominados como LRU (*Line Replacement Unit*), ya que son unidades modulares fácilmente implementables para cualquier configuración requerida. Como se comentó con anterioridad, esta línea de aviónica modular está muy en auge por la facilidad y estandarización que supondría. Se muestran en la Figura 5 los diferentes productos ofertados por la empresa RADA en cuestión de electrónica para UAV.

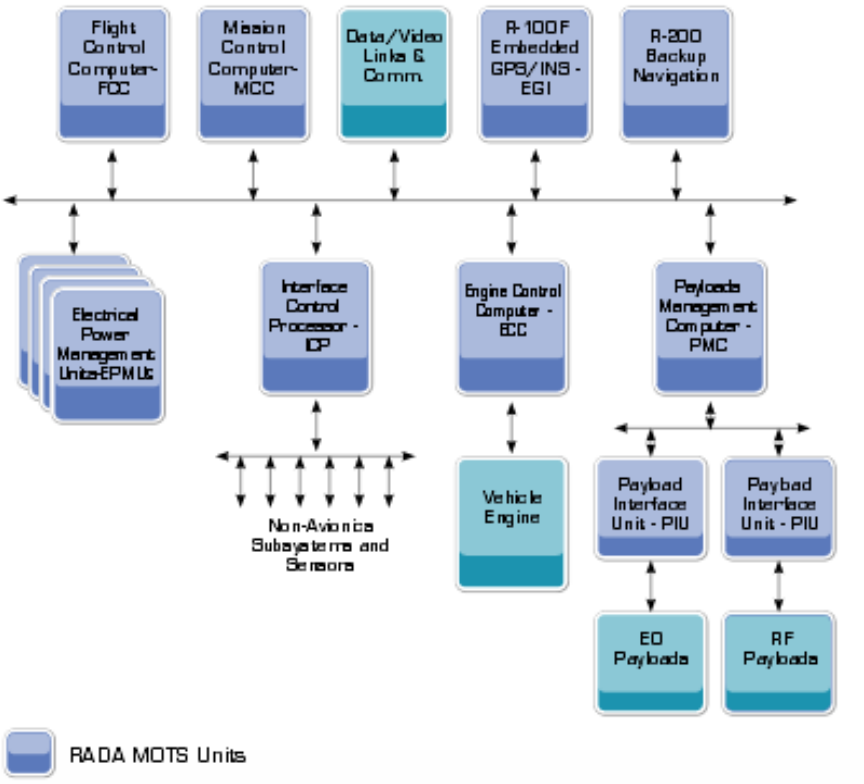


Figura 5. Diferentes productos con los que trabaja la empresa RADA (en violeta)

Los componentes que se muestran en la Figura 5 se describen como:

- ICP (*Interface Control Processor*): proporciona un puente a través de varios canales de comunicación MIL-STD-1553B, entre el computador de control de vuelo y cientos de puertos I/O. Consumo de 17 W y peso de 1.25 Kg.

- PMC (*Payload Management Computer*): permite la gestión de múltiples cargas de pago a través de una única plataforma, siendo compatible con el bus MIL-STD-1553. Consumo de 15 W y peso de 1.5 Kg.

- PIU (*Payload Interface Unit*): unidad dirigida por la PMC a través de control digital. Amplía la comunicación con la carga de pago, devolviendo información añadida y su estado. Incluye electrónica redundante que incrementa la seguridad. Consumo de 27 W y peso de 4 Kg.

- ECC (*Engien Control Computer*): proporciona una interfaz completa entre el motor y el FCC, incluyendo sensores, monitorización, control de actuadores y funciones de control discreto del motor. La aplicación de software modular permite la configuración externa de más de 300 parámetros. Consumo de 15 W y peso de 1.15 Kg.

- EPMU (*Electrical Power Management Unit*): RADA ofrece 6 tipos diferentes de estos dispositivos para el control de interruptores, protección y sensores.

2.5.4 Mavins (RADA)

Es el segundo producto de la empresa RADA dedicado a UAVs. En este caso se trata de una integración de computador de control de vuelo, autopiloto, sensores, etc. Su alta compacidad se justifica por estar destinado a UAVs mini y micro, además de como sistema de respaldo para vehículos pilotados. Sus sensores incluyen las tecnologías MEMS. Su peso es de 1 kg, y sus dimensiones de 110 x 100 x 10 mm. La apariencia del sistema es la que se muestra en la Figura 6.

Figura 6. Dispositivo MAVINS

En la Figura 7 se muestra el diagrama de bloques interno, obtenido del sitio web de la compañía. Como se puede ver en él, posee dos módulos iguales

que aportan redundancia, aumentando la fiabilidad del sistema. Cada uno de ellos cuenta con una CPU y una memoria como núcleo, de modo que este está comunicado con el resto a través de los buses CAN, RS422A, RS232C y Ethernet rápido. Observamos que también está duplicada la unidad de suministro de potencia y la unidad de medidas inerciales, GPS y un sistema de datos de aire. Finalmente vemos que se comunica con unidades externas como son dos antenas GPS, un sensor de presión estática y otro de presión total.

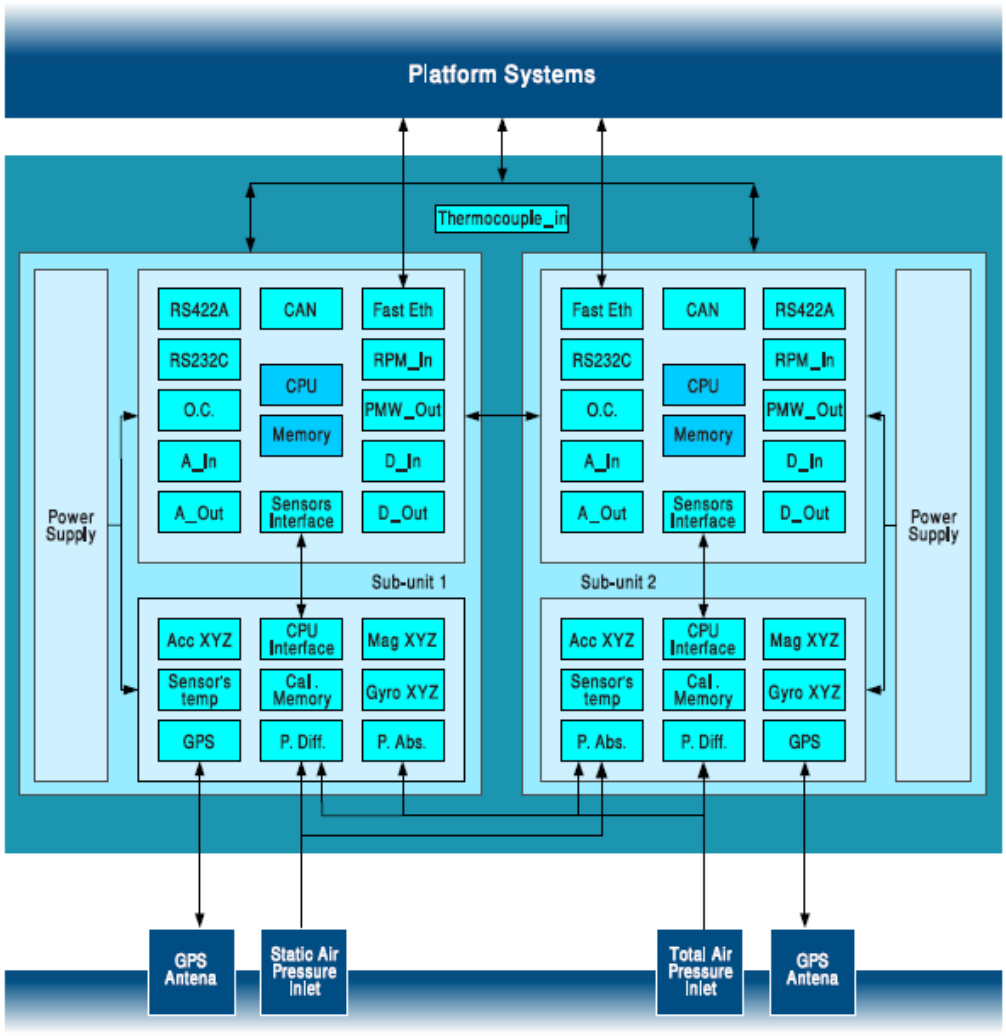


Figura 7. Diagrama de bloques del dispositivo MAVINS.

2.5.5 Piccolo (Cloud Cap Technology)

Esta empresa ofrece un autopiloto muy integrado, y además con miras a poder aplicarse a cualquier configuración, pasando de aviónica personalizable a aviónica estandarizada. Está orientado a UAVs pequeños y medianos. Posee un peso de 212g, incluyendo GPS y carcasa.

En la Figura 8 se muestra el diagrama de bloques del dispositivo en cuestión. En él se puede ver que posee un microprocesador como núcleo de todo el sistema, y este está conectado al resto de elementos. Por la izquierda observamos que recibe la fuente de energía. En la esquina inferior derecha se sitúa todo el conjunto de sensores, contando con giróscopos, un termopar, acelerómetros, sensor de la velocidad aerodinámica y GPS.

En cuanto a las comunicaciones que emplea el sistema, podemos comprobar que en la esquina superior derecha encontramos un bus CAN.

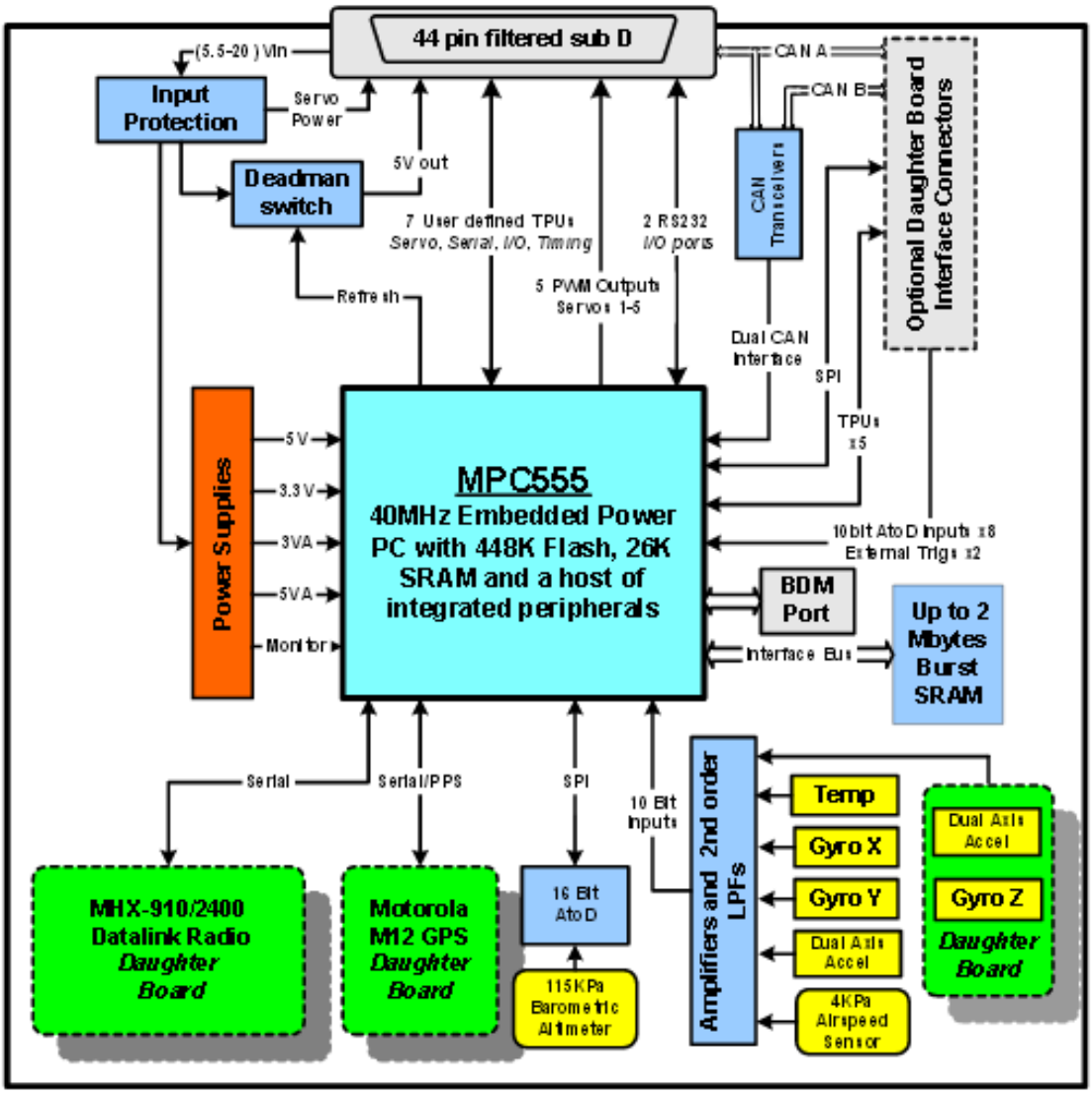


Figura 8. Diagrama de bloques del dispositivo Piccolo.

2.5.6 Pulse Aerospace

Esta empresa de aviónica ofrece diversos equipos entre los que encontramos una familia de autopilotos, que agrupamos debido a las similitudes que existen entre ellos:

- WePilot1000. Sistema de control de vuelo para pequeños helicópteros autónomos. Consiste en un sistema encapsulado que cuenta con receptor GPS, acelerómetros y giróscopos, ambos en tres ejes. Del mismo modo posee barómetros y conexión para un magnetómetro externo.

- WePilot1000FW. Sistema basado en el equipo anterior, pero con la aplicación de algoritmos específicos para sistemas de ala fija.

- WePilot3000. Sistema de control de vuelo doblemente redundante para vehículos autónomos que cuentan con un MTOW entre 10 y 500 Kg. Cuenta con dos CPUs, dos receptores GPS/Glonass, unidades de medidas inerciales en 5 ejes, dos magnetómetros, así como sensores de presión absoluta y relativa.

Las características generales medias se muestran en la Tabla 2

.

Dimensiones	**165x116x61 mm**
Masa (con carcasa)	**670 g**
Masa (sin carcasa)	**300 g**
Consumo	**450 x 12 V = 5.4 W**

Tabla 2. Características generales del los autopilotos de la empresa Pulse Aerospace.

2.5.7 Vector (UAV Navigation)

Autopiloto ofertado por la empresa UAV Navigation, destinado a UAVs medianos o grandes. Es capaz de controlar tanto UAVs de ala fija como de ala rotatoria, a través de un amplio rango de actuaciones. Cuenta con elementos redundantes para evitar fallos mayores ante errores de los sensores. Las dimensiones del dispositivo se muestran en la imagen Figura 9

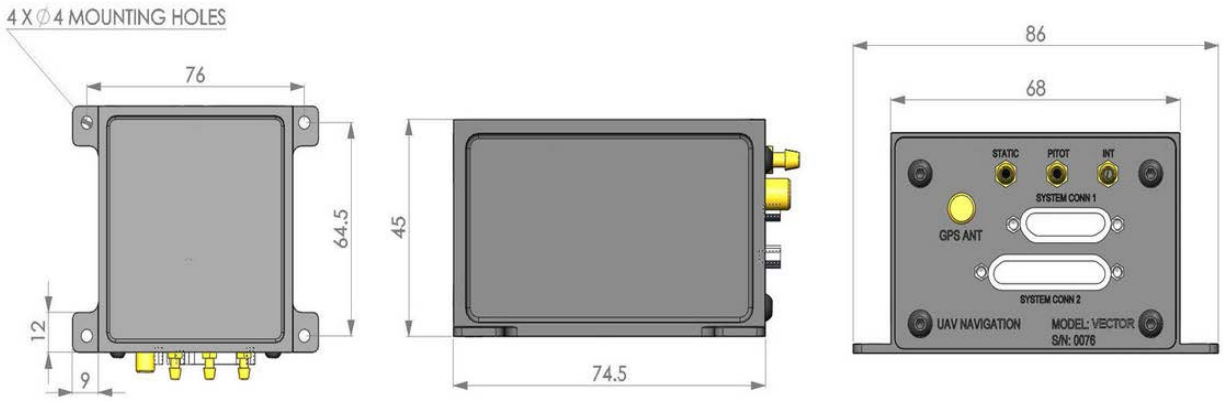


Figura 9. Dimensiones del dispositivo Vector de la empresa UAV Navigation

En cuanto a los sensores con los que trabaja, lleva integrada la placa denominada POLAR AHRS-INS que cuenta con numerosos dispositivos de navegación. Entre ellos, un sistema AHRS (Attitude, Heading & Reference System), una unidad de medidas inerciales, un sistema de navegación inercial, de los que se espera que posean los acelerómetros, giróscopos y magnetómetros necesarios, una unidad de datos de aire (ADS) y GPS. En la Figura 10 se incluye el diagrama de bloques de la tarjeta de sensores.

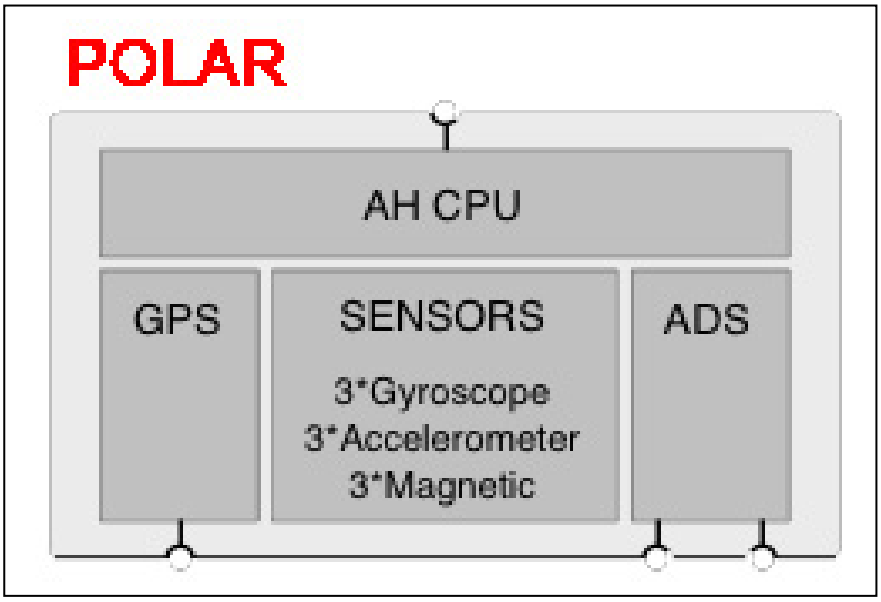


Figura 10. Diagrama de bloques de la placa de sensores POLAR AHRS-INS

También sabemos que su peso es de 300 g, y que consume 2.5 W para su correcto funcionamiento.

2.5.8 Proton (UAV Navigation)

De la misma empresa anterior también encontramos este autopiloto, cuyo objetivo es reducir el coste, el peso y el tamaño, mediante la fabricación en masa y la simplificación de los dispositivos con los que cuenta. Puede ser implementado en dispositivos de ala fija y rotatoria. Mostramos en la Figura 11 las dimensiones en milímetros del autopiloto en cuestión con el encapsulado final.

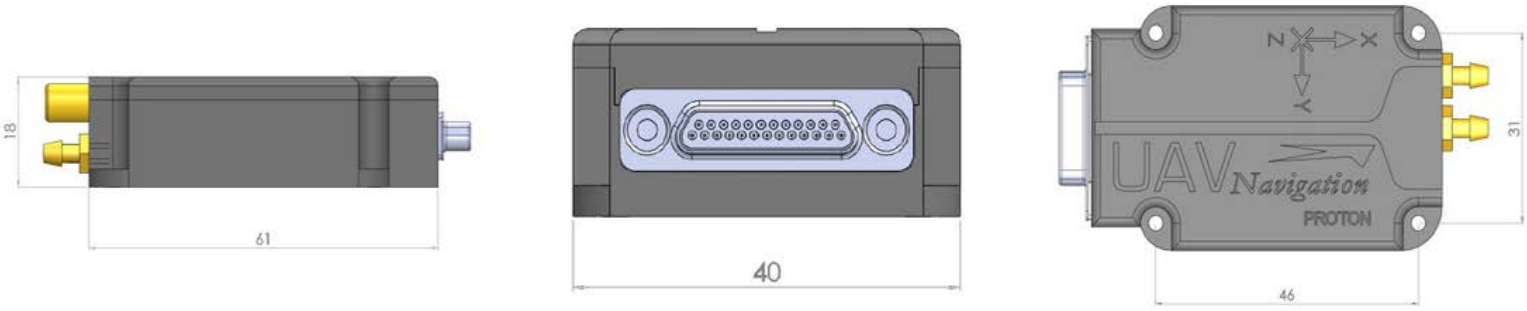


Figura 11. Dimensiones del dispositivo Proton de la empresa UAV Navigation

Todo lo necesario para el posicionamiento se proporciona encapsulado en la placa ATOM AHRS-INS, que al igual que el conjunto del autopiloto, está diseñada para reducir peso (menos de 2 g), tamaño y consumo (90 mW). Consta de 3 giróscopos, 3 acelerómetros y 3 magnetómetros, para determinar todas las posiciones asociadas a todos los grados de libertad. Se incluye en la Figura 12 su diagrama de bloques.

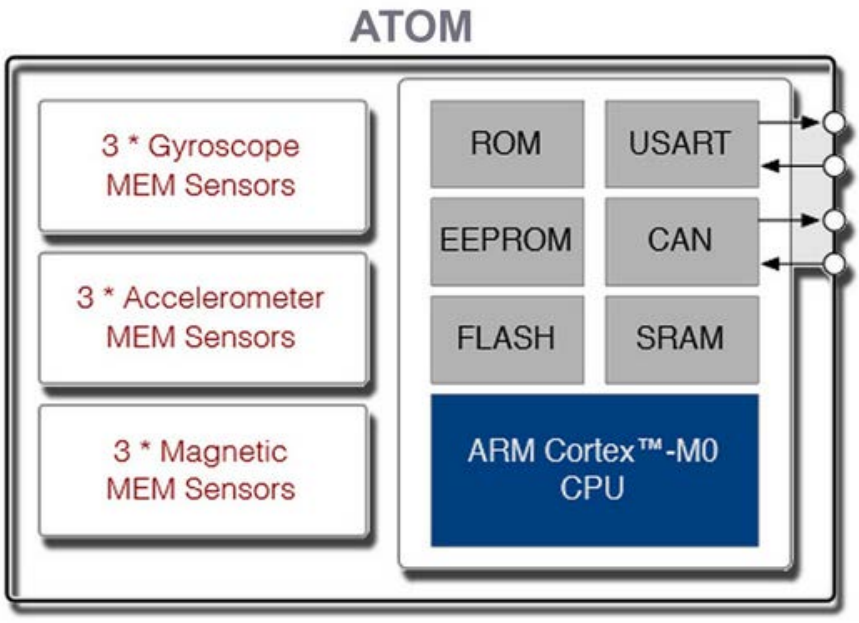


Figura 12. Diagrama de bloques de la placa ATOM AHRS-INS

2.5.9 OpenPilot.

OpenPilot está concebida como una plataforma de software libre, enfocada a ser desarrollada de forma comunitaria, o sea, cualquiera puede intervenir en el desarrollo, consiguiendo un dispositivo flexible de aviónica para UAVs pequeños. Inicialmente se proporciona la plataforma OpenPilot, común para cualquier propósito, y después esta es personalizada para uno u otro mediante la implementación de diferentes tarjetas adicionales. Este sistema está constantemente en desarrollo, pero actualmente se dispone de las diferentes estructuras:

- Plataforma CopterControl: este dispositivo está pensado como una plataforma de estabilización más que como un autopiloto, de modo que no es el producto con más aplicaciones de la compañía. Para desempeñar su función está provista de una tarjeta de sensores denominada CC3D que consta de 3 giróscopos y 3 acelerómetros, además de poder recibir señales de GPS.

- Plataforma Revolution: está actualmente en desarrollo, y su función final será constituir un autopiloto aplicable a cualquier vehículo no tripulado. Debido a las mayores exigencias de su diseño, consta de un dispositivo INS completo, desarrollando todas las funciones asociadas a un autopiloto.

La Figura 13 muestra la apariencia de la plataforma OpenPilot, además de sus diferentes componentes:

Figura 13. Apariencia de la plataforma OpenPilot y sus componentes.

2.5.10 Familia de Autopilotos MP2x28 (MicroPilot)

La empresa MicroPilot ofrece una familia de autopilotos de modo que entre todos se cubre gran rango de las aplicaciones de los UAVs pequeños y micro, y también diferentes precios. Todos ellos poseen una serie de características técnicas comunes y alguna que los diferencia de los demás. De este modo respecto a especificaciones de navegación podemos comentar que todos poseen recepción de GPS, giróscopos en los tres ejes, acelerómetros en los tres ejes, aunque no todos constan de magnetómetros, pero siempre existe posibilidad de incluirlo. Todos pesan 24g y el consumo máximo que pueden presentar ronda los 5 W. Los diferentes productos que se ofrecen son:

- **MP2128-Heli2**: destinado para vehículos de ala fija y rotatoria (helicópteros o multi-rótores), posee el mayor rango de características de la familia.

- **MP2128-g2**: solo se puede aplicar a multi-rótores y vehículos de ala fija, aunque cuenta con mejores características para aviones de ala fija. Es compatible con un amplio rango de temperaturas y con diferentes opciones de comunicación.

- **MP2028-g2**: vuela vehículos de ala fija y multi-rótores. Su principal motivación es ofrecer la suficiente flexibilidad para ser aplicado a cualquier aplicación. De este modo cuenta con la mejor relación precio/actuación.

- **MP1028-g2**: igualmente, solo vuela UAVs de ala fija y multi-rótores, y su simple propósito es cubrir las necesidades de nivel más bajo. Por ello está asociado al autopiloto de menor coste, para las ocasiones en que obtener un dispositivo económico es lo primordial.

Además se cuenta con unas versiones mejoradas denominadas MP2128-LRC2 y MP2128-Heli-LRC2. Las siglas LRC significan “Long Range Communication”, de modo que la característica mejorada de estos dispositivos son las comunicaciones, permitiendo que estas alcancen mayores distancias y confiriéndoles mayor redundancia al hardware asociado a ellas, resultando un vehículo más flexible.

También está disponible el autopiloto MP2128-3X, que consta de triple redundancia (tres autopilotos MP2128-Heli2 integrados y otros dispositivos de respaldo), consiguiendo el elemento de mayor confianza.

2.5.11 AirElectronics

Esta empresa española ofrece todo tipo de elementos electrónicos de aplicación en UAVs, entre los que encontramos algunos asociados al control del vuelo de dichos vehículos:

- U-Pilot: constituye el núcleo de la aviónica de cualquier UAV donde sea implementado, ya que está diseñado como un autopiloto. Permite controlar vehículos de ala rotatoria y fija en todo momento. Posee sensores MEMS de alta sensibilidad, constituyendo la unidad que recoge los datos de posicionamiento, y que se disponen según el siguiente diagrama:

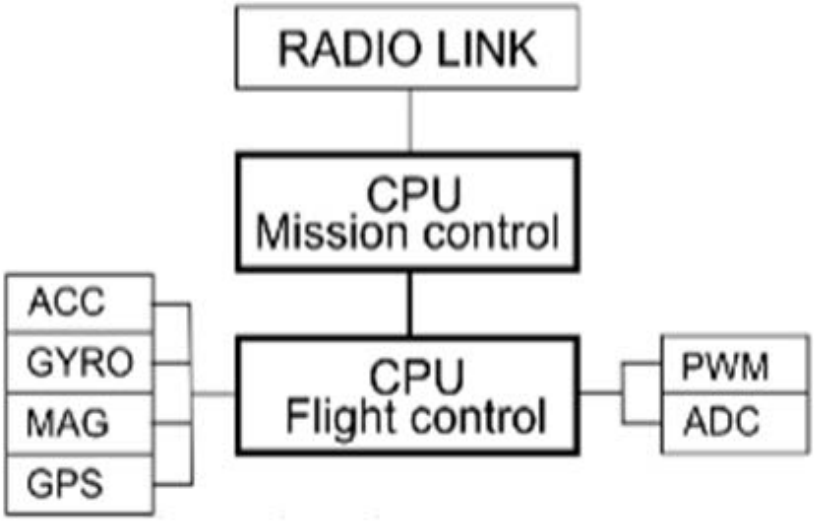


Figura 14. Diagrama de bloques del dispositivo U-Pilot

Como podemos ver, dispone de acelerómetros, giróscopos, sensores de presión estática para mejorar la actuación en diferentes altitudes y sensores de presión dinámica.

- U-Pilot 0EM: es exactamente el mismo dispositivo que el U-Pilot, pero presentado en versión delgada, debido a un encapsulado menos resistente. Presenta los mismos sensores y elementos, pudiendo desempeñar las mismas funciones e instalarse en los mismos UAVs.

2.5.12 Resumen de los datos obtenidos.

Debido a la gran cantidad de información recogida mediante el anterior estudio de mercado se hace necesario procesarla y presentarla de forma que sea más cómoda su interpretación. En este caso se ha decidido tabularla quedándonos solamente con los datos más relevantes para el desarrollo que se realizará con posterioridad. Aún así, las empresas

no ofrecen los mismos datos para sus productos, de modo que aunque no contamos con la totalidad de todos ellos, contamos con los suficientes de cada categoría para sacar nuestras propias conclusiones. Presentamos en la Tabla 3 la información a la que nos referimos.

Para comenzar, cada artículo está definido por su denominación y la empresa u organización a la que se debe su desarrollo. En segundo lugar, hemos indicado las dimensiones que posee cada dispositivo, para hacernos una idea del espacio que ocupan en el vehículo al que van destinado. En relación con esto podemos ver en la tercera columna el peso, cuando se especifica en el sitio web de cada empresa. El tamaño y el peso son dos parámetros muy determinantes para el diseño de la aviónica de un sistema, como se comentó en el apartado “2.4 Aviónica en los UAVs”. Las columnas siguientes hacen referencia a los equipos de posicionamiento y sensores de la aviónica, de modo que son relevantes igualmente, sobre todo para el estudio posterior que se tratará en este trabajo. Así estudiamos el número de giróscopos, acelerómetros, magnetómetros (o sea, elementos típicos de una IMU aunque no siempre constituyan esta unidad), comunicación con GPS y sistema de datos de aire. Finalmente se presenta las comunicaciones existentes entre la unidad de datos inerciales y el resto de elementos de aviónica del sistema. Esto permitirá detectar cuales son los protocolos de comunicaciones más empleados en la práctica, para enfocar nuestros pasos siguientes a una visión más realista.

Como conclusiones, podemos destacar, esta vez ya con total certeza, el papel tan vital que juega la unidad de sensores inerciales en la aviónica de estos sistemas. De modo que la mayoría de los sistemas cuentan con 3 giróscopos y 3 acelerómetros. En cuanto a los magnetómetros, si no se cuenta con ellos, al menos existe la posibilidad de conectar uno externo. Además, de igual manera, en prácticamente todos se dispone de compatibilidad con la red GPS o al menos existe la posibilidad de conexión externa.

En cuanto a las comunicaciones que se emplean, destacamos el BUS CAN, Ethernet y RS422, apareciendo recurrentemente en estos sistemas.

Designación	Desarrollador	Destinatario	Dimensiones (mm)	Peso (kg)	IMU			GPS	ADS	Comunicaciones
					Giróscopos	Acelerómetros	Magnetómetro			
Robocóptero reconocimiento	MIT	A.R. Competition	-	6,8	SI	SI	SI	SI	NO	Ethernet NE100; PC104 stack
WASP	MIT	-	-	-	SI	SI	SI	SI	NO	Ethernet NE100; PC104 stack
Componentes Aviónica	RADA	MALE, HALE	152,5x185,5x70	1,975	-	-	-	-	-	MIL-STD-1553B; ARINC 429; Fast Ethernet; RS422; RS485;
Autopiloto Mavins	RADA	MINI, MICRO	110x100x78	1	3	3	SI	SI	PE, PD, T	RS422A; CAN BUS; Ethernet
Autopiloto Piccolo	Cloud Cap Technology	Pequeños, Medianos	121,92x60,96x38,1	0,212	3	3	NO	SI	T, Asp, BAlt	-
Autopiloto WePilot1000	Pulse Aerospace	Pequeños, Medianos	-	-	3	3	CON. EXT.	SI	P	-
Autopiloto WePilot3000	Pulse Aerospace	MTOW 10-500 Kg	165x116x61	0,67	2	3	SI	SI	PE, PD	-
Autopilot Vector	UAV Navigation	Pequeños, Medianos	68x45x74,5	0,3	3	3	SI	SI	SI	CAN; Ethernet; RS-232(3); RS-422/485(2);
Autopiloto Protón	UAV Navigation	Pequeños	61x40x18	0,05	3	3	3	NO	NO	CAN; Ethernet; RS-232(1); RS-422/485(1);
Autopiloto Copter Control Platform	Open Pilot	Pequeños	36x36	-	3	3	NO	NO: Externo	NO	-
Autopiloto Revolution Platform	Open Pilot	Pequeños	-	-	3	3	3	SI	P	-
Autopiloto MP2x28 (familia)	MicroPilot	Pequeños-Micros	100x40x15	0,024	3	3	SI	SI	Altitude, Asp	-
Autopiloto U-Pilot	AirElectronics	Pequeños	89x35x155	0,28	SI	SI	CON. EXT.	SI	SI (PE, PD)	RS-332
Autopiloto U-Pilot OEM	AirElectronics	Pequeños	58x21x138	0,077	SI	SI	CON. EXT.	SI	SI (PE, PD)	RS-333

Leyenda	
PE	Presión Estática
PD	Presión Dinámica
P	Presión
T	Temperatura
Asp	Air Speed
BAlt	Barometric Altitude
CON. EXT.	Conexión Externa

Tabla 3. Tabla resumen de los datos obtenidos mediante la investigación de mercado.

2.6 Conclusiones

Toda la información que se ha presentado hasta ahora respecto a los UAVs se ha expuesto con el objetivo de ilustrar el marco actual en relación al estado de estos vehículos. Una vez mostrados todos estos datos, y sobre todo, los sistemas tomados como ejemplo de la aviónica que se emplea en los UAVs, estamos en disposición de continuar.

Como se ha comentado, la unidad de medidas inerciales y el GPS son dos elementos muy importantes en la navegación de estos vehículos, de modo que en estos dispositivos estarán centrados nuestros siguientes objetivos: desarrollar el prototipo de un sistema de sensores de navegación integrados.

Se comenzará analizando una placa que hará el papel de IMU, incluyendo acelerómetros, giróscopos, magnetómetros y algunos sensores de datos de aire.

Posteriormente se estudiará una placa receptora de datos GPS, que complementará el posicionamiento que puede lograr la IMU.

Finalmente se estudiará cómo se han integrado ambos sistemas para obtener un dispositivo que compagine tanto datos de posicionamiento relativo como absoluto, para demostrar que el estado de la tecnología permite por tanto realizar proyectos de aviónica con bajo coste y complejidad, pero igualmente efectivos. Nuestro alcance final será obtener un dispositivo capaz de devolvernos coordinadamente tanto datos de la IMU como del GPS.

3 Descripción de la Unidad de Medidas Inerciales

3.1 Introducción y Motivación

Antes de describir nuestra placa debemos decir que el prototipo que se desarrollará se apoyará en el uso de la plataforma Arduino, en concreto se usará un Arduino Due. Este dispositivo nos ofrece mayor potencia de procesamiento debido al uso de una CPU Atmel SAM3X8E ARM Cortex-M3, convirtiéndose en la primera placa de Arduino en utilizar un microcontrolador de 32-bits con arquitectura ARM. Se destacan las siguientes características:

- Cuenta con 54 pines digitales de salida/entrada, 12 entradas y 2 salidas analógicas y varios pines para dar potencia a diferentes voltajes. Destacamos la presencia de los pines del protocolo I2C, denominados SDA-SCL correspondiéndose con las dos líneas que emplea, y los pines que permiten la conexión con un bus CAN. Estos serán importantes en nuestro prototipo, ya que serán empleados; concretamente, la IMU emplea la comunicación I2C.

- A diferencia de otros Arduinos, que funcionan alimentados con 5 V, el Arduino Due es alimentado con una tensión de 3.3 V, lo que se deberá tener en cuenta en la conexión con el resto de componentes.

- La alimentación del Arduino durante el desarrollo de nuestro trabajo se realizará a través de un puerto USB del ordenador. Pero no será problema a la hora de implementarlo para su propósito final, ya que la alimentación también puede realizarse a través de otros medios, siempre y cuando se realice entre 7 y 12 V.

- El código cargado en la placa es almacenado en una memoria Flash de 512 KB.

- Posee dos puertos USB. En nuestro caso se empleará el puerto de programación, recomendado para usarlo mediante la programación del Arduino. El otro es denominado puerto nativo, y generalmente se emplea para utilizar el Arduino como un periférico.

Dicho esto pasamos a presentar nuestra IMU, que desempeñará el papel de receptor de datos inerciales y de aire tan importantes para la correcta navegación de un vehículo no tripulado (podríamos decir que serían parte de los sentidos haciendo una analogía con el cuerpo de un ser vivo). Se ha

elegido la placa *10-DOF IMU Breakout - L3GD20H + LSM303 + BMP180*, de la compañía de componentes electrónicos Adafruit. Esta designación se debe a que la compañía ha integrado, tanto a nivel hardware como a nivel software, tres de sus productos anteriores: los giróscopos L3DG20H, los acelerómetros y magnetómetros LSM303DLHC y los sensores de temperatura y presión BMP180. Todos ellos emplean la comunicación I2C, por lo que todo su conjunto puede comunicarse con la placa Arduino mediante este protocolo, que como se dijo, ofrece la comodidad de utilizar solo dos cables. La placa IMU puede ser alimentada mediante 5 V, para ser empleada por otros Arduinos, y mediante 3.3 V, que será nuestro caso, lo que supondrá una gran comodidad. Como características destacamos:

- Las dimensiones de la placa son 38 x 23 x 3 mm.
- Su peso es de 2.8 g.
- El acelerómetro/magnetómetro LSM303DLHC actúa en un rango de ±16 g y ±8.1 gauss.
- El giróscopo L3GD20 actúa entre los ±2000 grados por segundo.

Se muestra en la Figura 15 la placa en cuestión, indicando los pines que trataremos en nuestro desarrollo posterior.

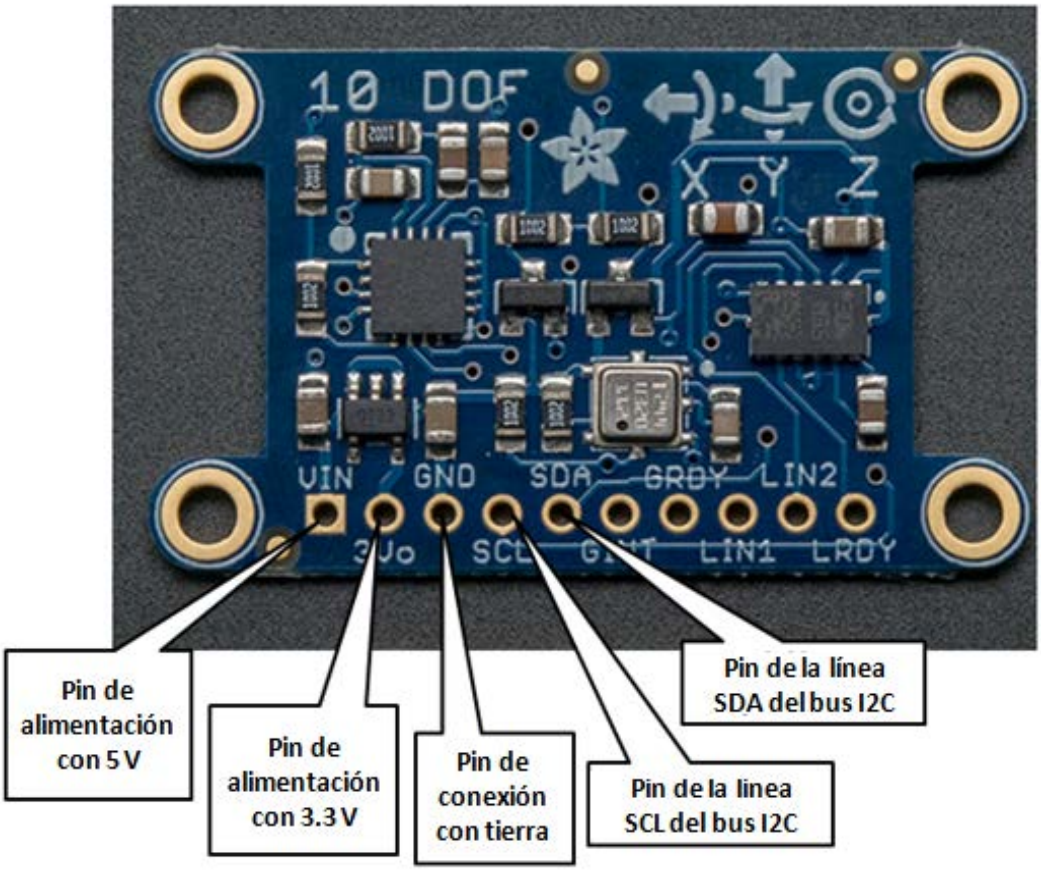


Figura 15. Apariencia de la IMU indicando los pines más relevantes.

El objetivo de emplear una IMU y, más concretamente, los datos que nos ofrece, es obtener la información críticamente necesaria para la correcta navegación del vehículo autónomo. Desde el momento en que se comienzan a eliminar funciones que realizaba el piloto antiguamente o en aviones menos avanzados, para cedérselas a sistemas de aviónica, dichos equipos deben realizar las mismas funciones pero mediante la implementación de otro tipo de tecnología. Mientras que el piloto visualmente analizaba los dispositivos de orientación y estabilización con sus ojos y procesaba la información con su cerebro, ahora es necesario implementar procesadores y sensores que analicen y capten la información del entorno. Con todo esto se pretende una vez más hacer énfasis en la importancia de este tipo de unidades en la electrónica de los UAVs. Por lo que, además, habiendo comprobado que cualquier dispositivo real del mercado lleva implementado algún tipo de sensor con estas características, ya sea constituyendo una IMU, un AHRS o simplemente una placa con determinados sensores, queda más que justificado centrar nuestro estudio en desarrollar el dispositivo ya comentado, desde el momento en que estamos interesados en conocer su funcionamiento.

3.2 Conexión con Arduino Due

La conexión con la placa Arduino es muy sencilla, pero hay que tener en cuenta las características especiales asociadas a nuestro Arduino Due, entre las que principalmente se encuentra el uso de niveles de tensión a 3.3 V. Se muestra en la siguiente tabla la conexión entre pines:

Pin de la IMU	Pin del Arduíno
SCL	SCL
SDA	SDA
GND	GND
3V0	3.3 V power supply

Tabla 4. Conexiones entre el Arduino Due y la IMU

A pesar de que nosotros hemos optado por conectar la fuente de tensión de 3.3V al pin 3V0, la arquitectura de la IMU es flexible, permitiendo conectarla a placas Arduinos con otros niveles de tensión e incluso de diferentes maneras para cada placa. El pin *Vin* cuenta con un regulador de tensión de modo que es posible conectar a él una entrada de 3.3 V o de 5V. Sin embargo, conectando la fuente de 3.3 V al pin 3V0, se realiza un bypass a este regulador. La compañía recomienda alimentar la placa con 3.3 V, por lo que esta es la justificación de nuestra decisión.

Finalmente resulta un cableado como el de la Figura 16.

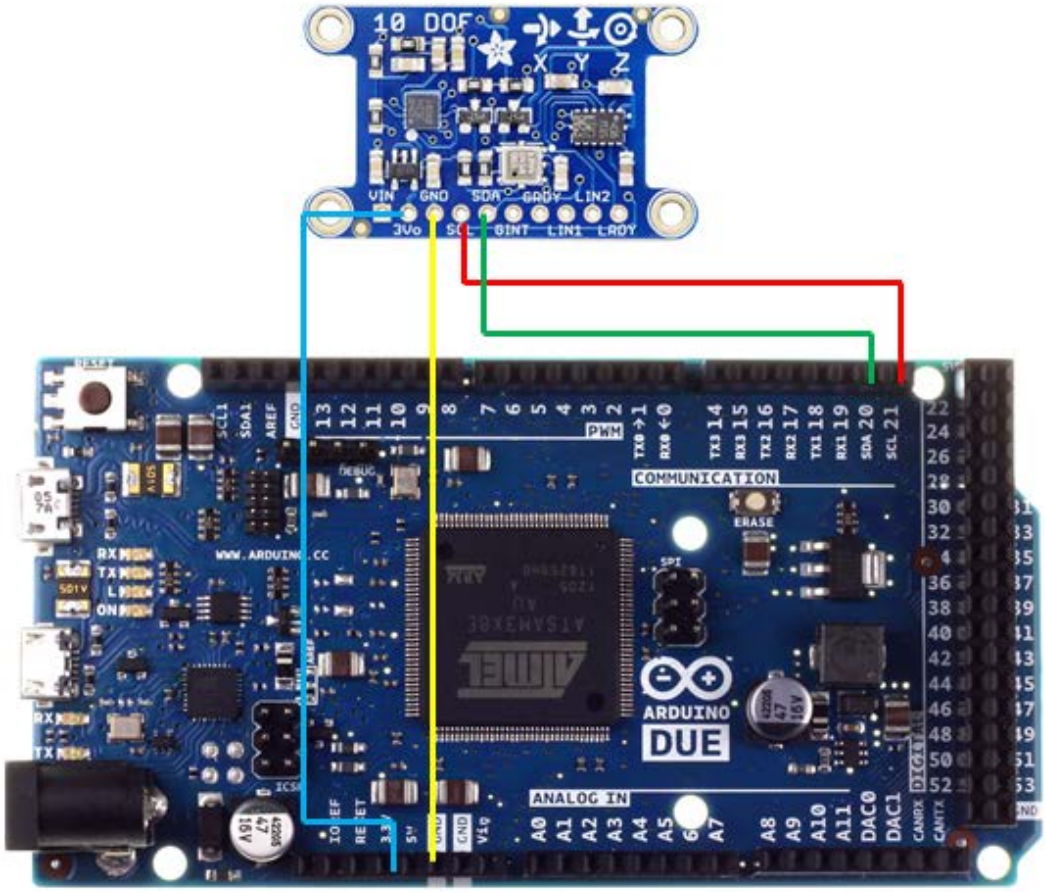


Figura 16. Gráfico de las conexiones entre la placa Arduino Due y la IMU.

3.3 Software para el uso de la IMU

Como ya se ha dicho, la IMU es una integración de varios sensores, los cuales devuelven numerosos valores: aceleración en los tres ejes, velocidad angular en los tres ejes, y orientación de los tres ejes respecto al campo magnético de la Tierra. Pero además, es posible manipular estas magnitudes para obtener otras. Por ejemplo, integrando numéricamente la aceleración podemos obtener la velocidad lineal en los tres ejes, e integrando dicha velocidad podemos obtener la distancia recorrida en los tres ejes desde la posición inicial. Lo mismo ocurre con los giros, de modo que derivando la velocidad angular podemos obtener la aceleración angular, e integrándola podemos obtener el giro en grados desde la posición inicial. Este giro, que tras la manipulación matemática tendrá errores, podemos compararlo con la variación de los ángulos obtenida por el magnetómetro y mediante la implementación de algún filtro de errores ir corrigiendo dichos valores.

Todas estas operaciones pueden ser realizadas mediante el uso de los códigos proporcionados por la compañía Adafruit para el correcto

funcionamiento de los sensores de la IMU. Algunos de ellos están enfocados a algunos de los sensores, de modo que solo recogen los datos proporcionados por parte de ellos. Sin embargo, hay un conjunto de códigos que son los que trabajan realmente como integradores de la IMU. Presentamos a continuación los códigos que ofrece la compañía.

- **Código del sensor de presión y temperatura BMP180, denominado *sensorapi***: este código extrae la información que obtienen los sensores de presión y temperatura integrados en la IMU. Además, el código aplica un modelo de atmósfera empleando la presión y la temperatura para estimar la altitud (por definición, respecto al nivel del mar). Esta resulta muy necesaria a la hora de calcular la distancia entre el UAV y el suelo. Los datos son mostrados en el *Monitor Serial* de Arduino, obteniéndose una visualización como la que se muestra en la Figura 17.

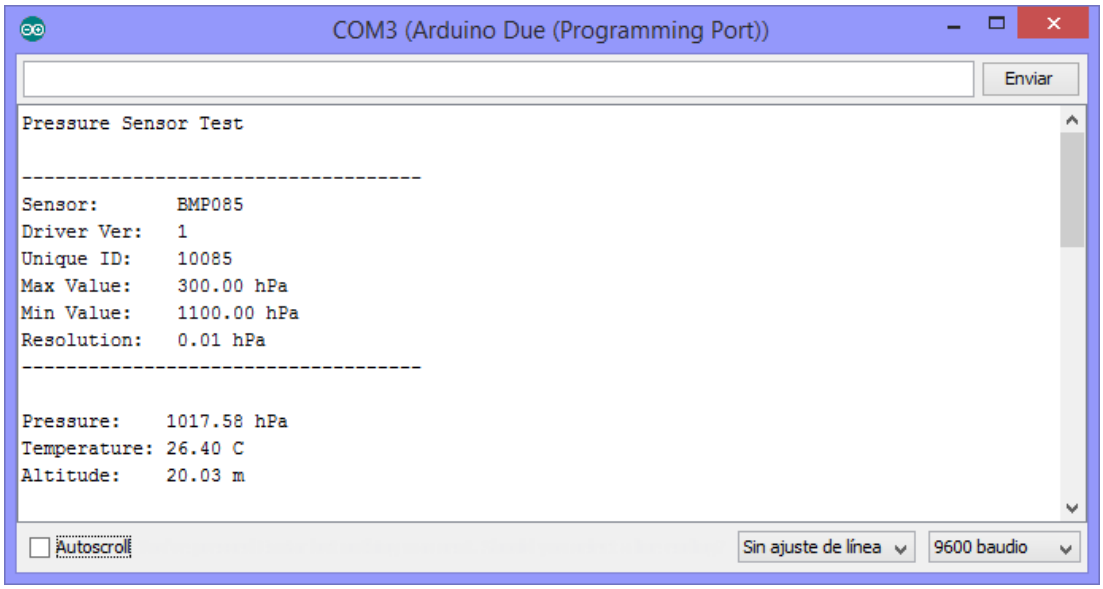


Figura 17. Resultados mostrados por el Monitor Serial del código *sensorapi*

Se muestra un primer bloque de iniciación y verificación del sensor, donde se observa el rango y la resolución de presiones posibles. Posteriormente se imprime, actualizándose recurrentemente, los valores de presión, temperatura y la estimación de la altitud que se obtienen de los sensores. Las pruebas se han realizado en Sevilla en Verano, de modo que los valores de temperatura y presión son muy razonables. Para obtener una buena estimación de la altitud es necesario actualizar la presión a nivel de mar en la función *Adafruit_Sensor.h*, incluida en la librería que debemos tener guardada.

- **Código de los giróscopos L3GD20, denominado *sensorapi*:** en este caso se extraen los valores de velocidades angulares en los tres ejes que recogen los tres giróscopos dispuestos de tal manera. Esos datos son imprescindibles para que el vehículo autónomo conozca la velocidad de giro en cualquier eje, y en caso de integrar estos datos, conocer la inclinación del vehículo en cualquier dirección. Sin embargo, no están implementados estos pasos en este código. Finalmente se presentan los datos en el *Monitor Serial* como se muestra en la Figura 18.

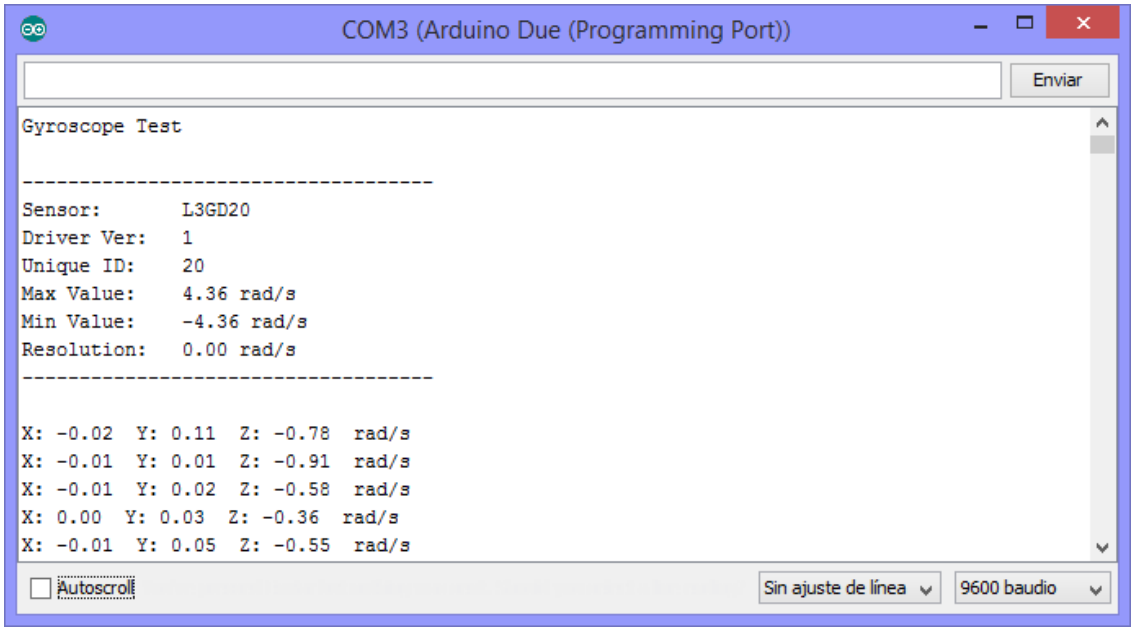


Figura 18. Resultados mostrados por el Monitor Serial del código *sensorapi*

Al inicio se muestra el bloque de iniciación y verificación del sensor, donde se puede comprobar que todo funciona correctamente, y además se muestra el rango de valores para la velocidad angular. Después se muestra dicha velocidad angular en los tres ejes. Para obtener unos datos más ilustrativos se ha realizado esta prueba girando la placa lo más uniformemente posible alrededor del eje Z, por lo que se puede comprobar que los valores de los giros alrededor de los ejes X e Y tienden a ser nulos, y sin embargo, existe velocidad angular alrededor del eje Z

- **Código del acelerómetro/ magnetómetro LSM303, denominado accelsensor**: a pesar de que el sensor LSM303 incluye una serie de acelerómetros y magnetómetros, este código solamente está destinado a leer los valores de las aceleraciones que obtienen los acelerómetros situados en los tres ejes. Al igual que en el resto de casos, estas aceleraciones se podrían integrar para obtener la

velocidad que experimenta el vehículo, pero al igual que anteriormente, esto no está implementado en este código. Finalmente las aceleraciones se presentan en el Monitor Serial como se muestra en la Figura 19.

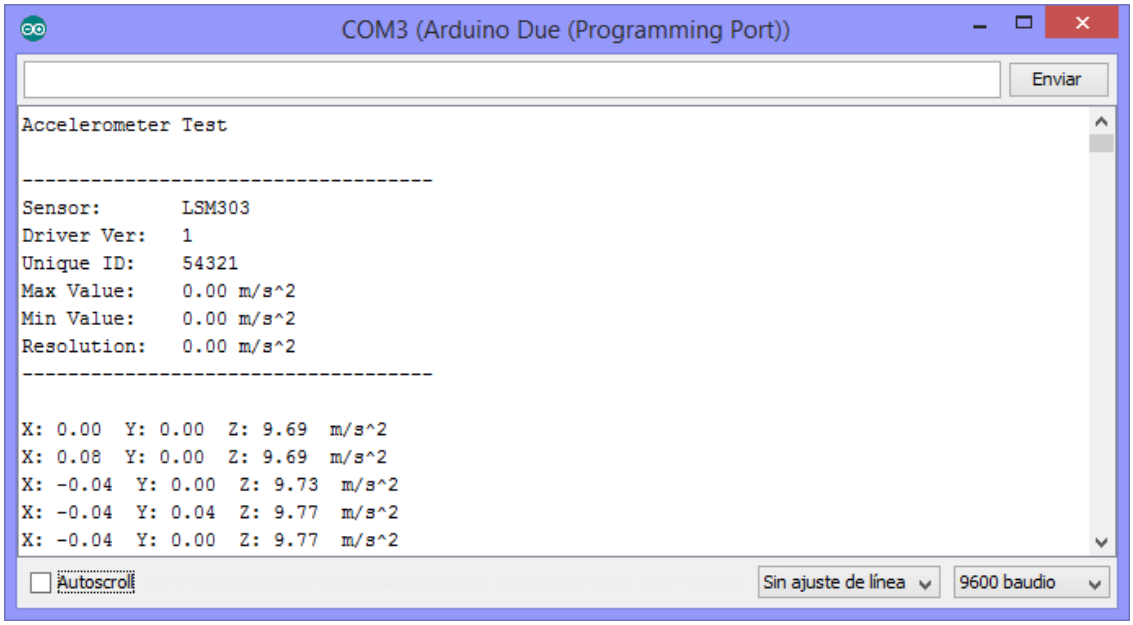


Figura 19. Resultados mostrados por el Monitor Serial del código *accelsensor*

Como en los casos anteriores, primeramente se muestra el bloque de iniciación y validación de los sensores. Debajo de este se pueden ver las líneas con los valores, actualizándose en intervalos regulares, de las aceleraciones en los tres ejes, correspondientes a cada uno de los acelerómetros. En este caso se ha situado la placa sobre una superficie nivelada horizontalmente, de modo que podemos comprobar que sobre los ejes X e Y no actúa ninguna aceleración, sin embargo, podemos observar cómo sobre el eje Z se refleja la acción de la fuerza de la gravedad, con unos valores bastante aceptables.

- **Código del acelerómetro/ magnetómetro LSM303, denominado *magsensor:*** como en el caso anterior solo disponíamos de un código para extraer los valores de las aceleraciones, necesitamos este código para conocer los datos que ofrecen los magnetómetros en los tres ejes. Igualmente, los datos se muestran en el *Monitor Serial*, como ilustra en la Figura 20. Como en el caso anterior, observamos al inicio un bloque de iniciación y comprobación del sensor, y después los valores del campo medido en micro-Tesla, debido al campo magnético terrestre, actualizándose repetitivamente.

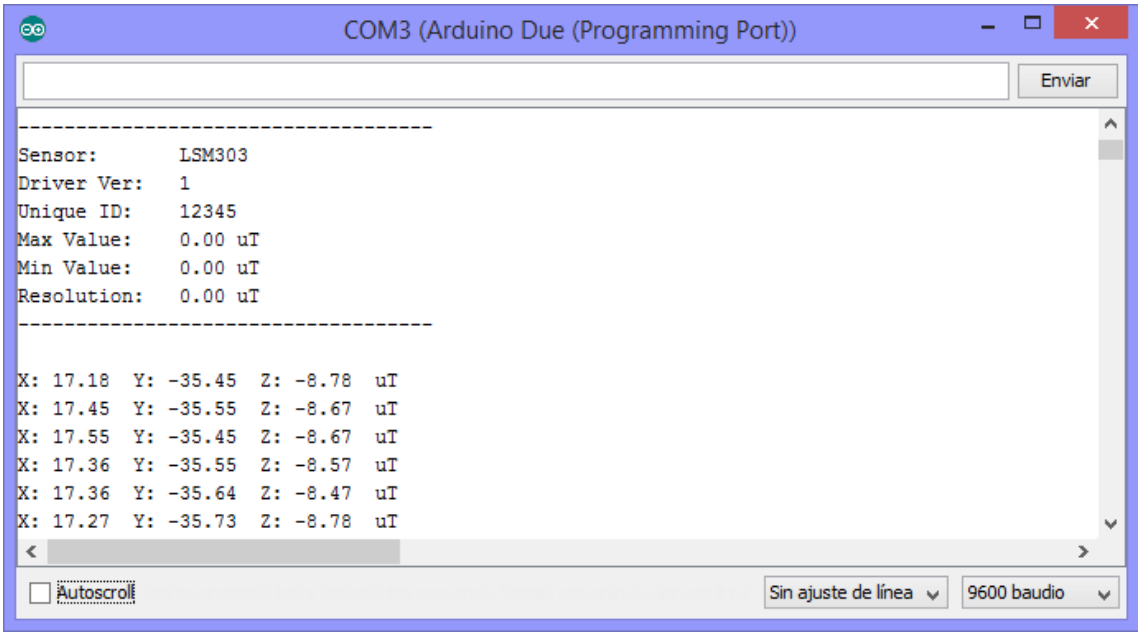


Figura 20. Resultados mostrados por el Monitor Serial del código *magsensor*

- **Código de integración de la IMU, denominado *tester*:** *en* este caso, se presenta el primero de los códigos que no solo obtiene los datos de uno de los sensores, si no que recoge los datos de todos al mismo tiempo. Como su nombre indica, no realiza ninguna operación aparte de actuar como un test de cada uno de los sensores por separado y presentar todos los datos recogidos en conjunto. Por ello, al igual que sucedía en los códigos anteriores, inicialmente en el *Monitor Serial* se muestra el bloque de inicio y verificación de cada uno de los sensores por separado, idénticos a los presentados hasta ahora. Después se muestra un bloque con todos los datos, que se actualiza a intervalos regulares. El resultado se ejemplifica en la Figura 21. Como se puede comprobar se muestran, por este orden, las aceleraciones, el campo magnético, las velocidades angulares, presión, temperatura y altitud estimada.

- **Código de integración de la IMU, denominado *ahrs*:** este es el primer código que procesa los datos de todos los sensores para devolver una información más útil desde el punto de vista de la estabilización de nuestro vehículo o la obtención de la actitud de algún dispositivo. Además se proporciona la temperatura y la presión, datos que igualmente son muy importantes para obtener propiedades del movimiento del vehículo, como el número de Mach. En la Figura 22 mostramos el resultado que se ve en el *Monitor Serial*.

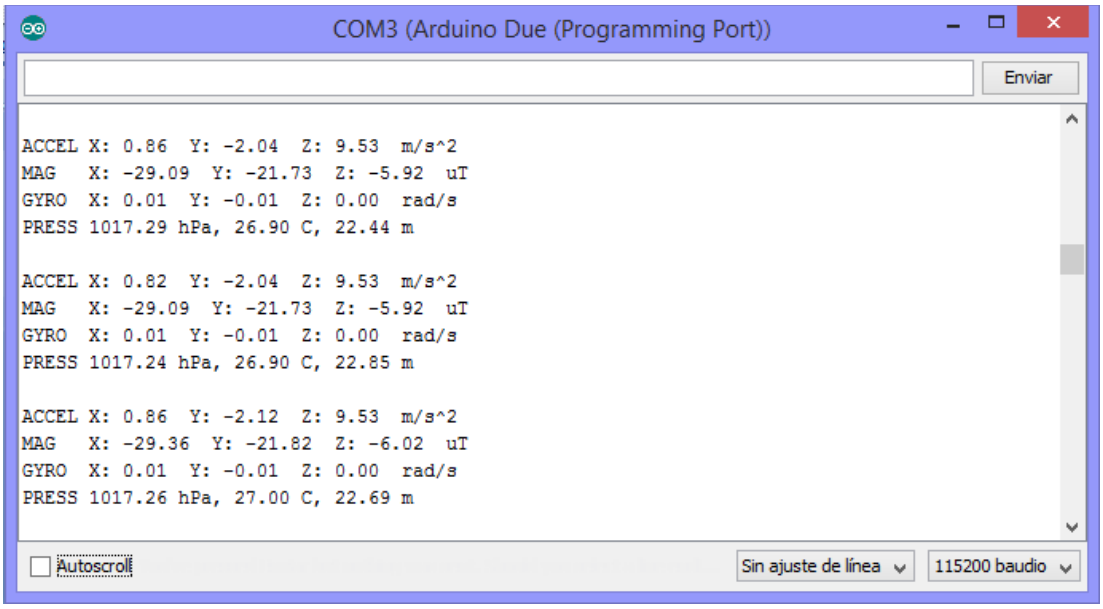


Figura 21. Resultados mostrados por el Monitor Serial del código tester

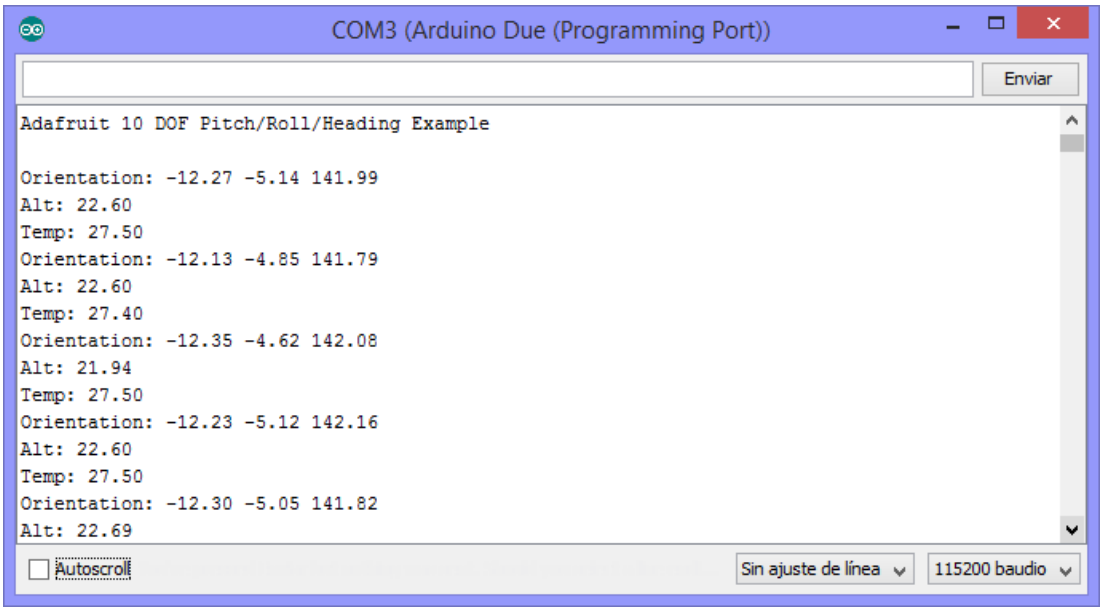


Figura 22. Resultados mostrados por el Monitor Serial del código *ahrs*

Como se puede observar, en primer lugar se obtienen los valores asociados a lo que se denomina *Orientation*, y que se corresponden con los ángulos de alabeo, cabeceo y guiñada que experimentaría un vehículo, según el eje X. Se ilustran estos ángulos en la Figura 23.

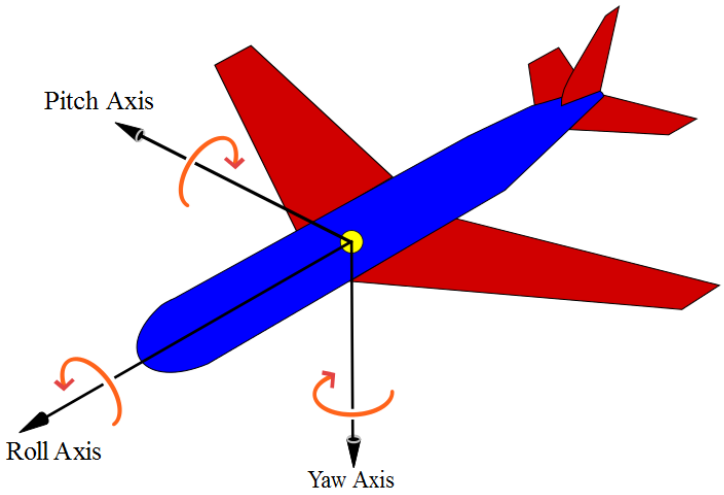


Figura 23. Ángulos de cabeceo (*pitch*), *balance* (*roll*) y guiñada (*yaw o heading*).

De este modo la IMU detecta alabeo nulo cuando el eje Z se alinea con la vertical, devolviendo ángulos positivos y negativos cuando se inclina a uno u otro lado de dicha posición, llegando hasta los 180 grados, momento en el que cambia de signo. Obtenemos cabeceo nulo cuando el eje X se posiciona horizontal, de modo que cuando este eje se sitúa verticalmente se alcanzan 90 grados positivos o negativos, y comienza a decrecer si el dispositivo continua girando, hasta alcanzar de nuevo un ángulo nulo. Y resulta una guiñada nula cuando el eje X se orienta hacia el norte, de modo que los ángulos varían entre 180 y -180 grados. Hay que tener en cuenta que el eje X que se muestra en la Figura 15 no corresponde con el eje X verdadero que considera el dispositivo, teniendo este que apuntar en el sentido opuesto, debido a un problema estético indicado por el fabricante. Finalmente se muestra la altitud estimada a partir de los datos de aire y la temperatura del entorno.

- **Código de integración de la IMU, denominado *pitchrollheading*:** este es el último de los programas que la compañía Adafruit ofrece para implementarlos junto a la IMU que empleamos. Como hemos dicho, estos códigos integrados combinan los datos de todos los sensores para obtener información relevante. Más en profundidad, este código obtiene la misma información que el código anterior, pero la presenta de manera diferente. Por lo que conociendo la función de uno de los programas, el otro es redundante. Igualmente, se obtienen los ángulos de Alabeo, Cabeceo y Guiñada, en este orden, y además la temperatura y la altitud estimada. El resultado obtenido en el *Monitor Serial* se muestra en la Figura 24.

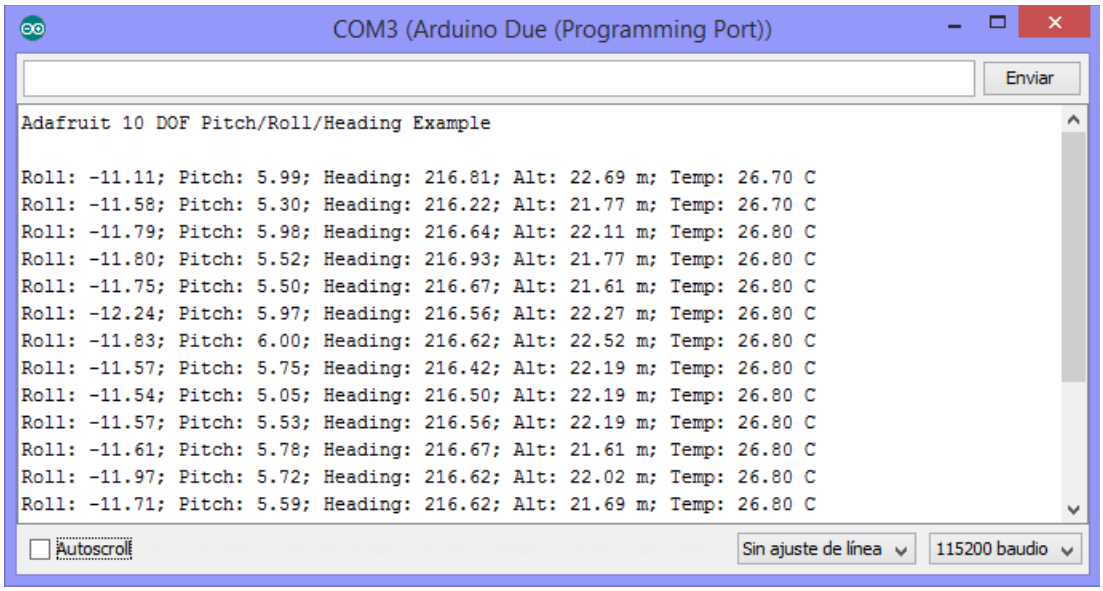


Figura 24. Resultados mostrados por el Monitor Serial del código *pitchrollheading*

3.4 Descripción del software relevante

Debido a que agrupar la totalidad de los programas de la IMU ofrecería información redundante, es importante seleccionar algunos de ellos para, una vez integrados, que ofrezcan todos los datos necesarios para la correcta navegación de nuestro vehículo. Principalmente esto es debido a que uno de los códigos que consiguen obtener un AHRS de nuestra IMU, tras combinar los datos de todos los sensores instalados, solamente ofrece los ángulos de guiñada, cabeceo y alabeo de la situación actual. Esto es perfecto para la estabilización, pero no involucra la trayectoria seguida en cada punto, por lo que vemos necesario incluir explícitamente los datos de las aceleraciones experimentadas por el vehículo. Para este propósito hemos seleccionado los dos programas siguientes, de los que se presenta el código. Además, se han dividido en bloques para ir comentando la función que realiza cada uno de ellos por separado.

3.4.1 pitchrollheading

Anteriormente se comentó la función que realiza este código, pero no se especificó como realiza su función. Para ello exponemos a continuación el código involucrado, tal y como se puede leer en el IDE. La única modificación ha sido la división en bloques comentada en el párrafo anterior.

```
/*------------------------------------------------------------------------------------------------------
*/
/*Bloque 1*/
#include <Wire.h>
#include <Adafruit_Sensor.h>
```

```
#include <Adafruit_LSM303_U.h>
#include <Adafruit_BMP085_U.h>
#include <Adafruit_10DOF.h>
/*-------------------------------------------------------------------------------------------------------
*/
/*Bloque 2*/
/* Assign a unique ID to the sensors */
Adafruit_10DOF                dof   = Adafruit_10DOF();
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified   mag   = Adafruit_LSM303_Mag_Unified(30302);
Adafruit_BMP085_Unified       bmp   = Adafruit_BMP085_Unified(18001);

/* Update this with the correct SLP for accurate altitude measurements */
float seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;

/*-------------------------------------------------------------------------------------------------------
*/
/*Bloque 3*/
/*!
   @brief  Initialises all the sensors used by this example
*/
void initSensors()
{
  if(!accel.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println(F("Ooops, no LSM303 detected ... Check your wiring!"));
    while(1);
  }
  if(!mag.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println("Ooops, no LSM303 detected ... Check your wiring!");
    while(1);
  }
  if(!bmp.begin())
  {
    /* There was a problem detecting the BMP180 ... check your connections */
    Serial.println("Ooops, no BMP180 detected ... Check your wiring!");
    while(1);
  }
}
/*-------------------------------------------------------------------------------------------------------
*/
/*Bloque 4*/
void setup(void)
{
```

```
 Serial.begin(115200);
 Serial.println(F("Adafruit 10 DOF Pitch/Roll/Heading Example")); Serial.println("");

 /* Initialise the sensors */
 initSensors();
}
/*-------------------------------------------------------------------------------------------------------
*/
```

/*Bloque 5*/

```
/*!
    @brief  Constantly check the roll/pitch/heading/altitude/temperature
*/
void loop(void)
{
 sensors_event_t accel_event;
 sensors_event_t mag_event;
 sensors_event_t bmp_event;
 sensors_vec_t   orientation;

 /* Calculate pitch and roll from the raw accelerometer data */
 accel.getEvent(&accel_event);
 if (dof.accelGetOrientation(&accel_event, &orientation))
 {
  /* 'orientation' should have valid .roll and .pitch fields */
  Serial.print(F("Roll: "));
  Serial.print(orientation.roll);
  Serial.print(F("; "));
  Serial.print(F("Pitch: "));
  Serial.print(orientation.pitch);
  Serial.print(F("; "));
 }

 /* Calculate the heading using the magnetometer */
 mag.getEvent(&mag_event);
 if (dof.magGetOrientation(SENSOR_AXIS_Z, &mag_event, &orientation))
 {
  /* 'orientation' should have valid .heading data now */
  Serial.print(F("Heading: "));
  Serial.print(orientation.heading);
  Serial.print(F("; "));
 }

 /* Calculate the altitude using the barometric pressure sensor */
 bmp.getEvent(&bmp_event);
```

```
 if (bmp_event.pressure)
 {
  /* Get ambient temperature in C */
  float temperature;
  bmp.getTemperature(&temperature);
  /* Convert atmospheric pressure, SLP and temp to altitude    */
  Serial.print(F("Alt: "));
  Serial.print(bmp.pressureToAltitude(seaLevelPressure,
                                      bmp_event.pressure,
                                      temperature));
  Serial.print(F(" m; "));
  /* Display the temperature */
  Serial.print(F("Temp: "));
  Serial.print(temperature);
  Serial.print(F(" C"));
 }

 Serial.println(F(""));
 delay(1000);
}
/*--------------------------------------------------------------------------------------------------------
*/
```

- *Bloque 1*. En esta sección solamente se llama a las funciones indicadas, igualmente proporcionadas por el fabricante, y necesarias para el correcto funcionamiento de la IMU.

- *Bloque 2*. En este bloque se definen las variables que serán resultado de las funciones definidas en el bloque 1. Como se puede observar, dichas magnitudes serán *dof*, magnitud asociada a una función necesaria para el funcionamiento del conjunto de la IMU, *accel*, que recoge el resultado de las aceleraciones leídas por el acelerómetro, *mag*, que almacenará el valor del campo leído por el magnetómetro, y *bmp*, para la lectura del barómetro. Finalmente se indica que para mayor precisión de la estimación de la altitud se recoge en la variable *seaLevelPressure* la presión a nivel del mar.

- *Bloque 3.* En este se define una función auxiliar, que será llamada posteriormente, y cuyo funcionamiento se emplea para detectar errores de conexión en el cableado entre la IMU y la placa Arduino. Simplemente comprueba que todo se encuentre en buen estado, y en caso contrario muestra el siguiente mensaje: *"Ooops, no LSM303 detected ... Check your wiring!"*. Este mensaje se da en el caso de que los acelerómetros-magnetómetros de la placa fallen, pero para el resto de casos, el mensaje es análogo simplemente

modificando la denominación del sensor en cuestión que está dando problemas.

- *Bloque 4.* Este bloque abarca la función *void setup*, que se procesará una vez al inicio del programa, y que se encarga de iniciarlo correctamente. En primer lugar se inicia la comunicación con el *Monitor Serial* a 115200 baud y justamente después se imprime en dicha ventana el mensaje *"Adafruit 10 DOF Pitch/Roll/Heading Example"* como muestra de que el programa se ha cargado en la placa y ha comenzado a funcionar. Por último se llama a la función descrita en el bloque tres, ya que es lógico que el siguiente paso sea comprobar el estado de los sensores. Dicha función se ejecuta, y si todo está en correcto funcionamiento se pasa al siguiente bloque sin respuesta alguna, siendo esto buena señal.

- *Bloque 5.* Este bloque involucra al código que se repetirá repetitivamente, y que constituirá la parte más crucial de nuestro programa, obteniendo los datos que se indicaron en la sección anterior. Para ello se comienza leyendo todos los datos proporcionados por los sensores. Posteriormente, como se indica en los comentarios proporcionados por el fabricante, se obtienen mediante una función el alabeo y el cabeceo de los datos proporcionados por los acelerómetros, de modo que si dicha operación se finaliza con éxito se muestran estos valores por pantalla. Para ellos ambos deben estar dentro del rango válido de valores. Después se calcula la guiñada, con ayuda de los datos registrados por los magnetómetros, y de la misma manera, si los datos obtenidos son posibles, se muestran por pantalla. Finalmente se calcula la altitud estimada a partir de la presión atmosférica y la temperatura obtenida, y se muestra por pantalla. Como se puede observar no se emplean los datos proporcionados por los giróscopos, ya que es suficiente la información que se extrae del resto de sensores. Finalmente se incluye un *delay* para ayudar a que la información se muestre más lentamente y sea más legible, aunque se puede adaptar a cualquier velocidad, por los requisitos que necesitemos en cada ocasión.

3.4.2 accelsensor

Una vez que disponemos de los datos que nos ofrecen la orientación en la que se encuentra nuestro dispositivo, parece lógico incluir en los datos necesarios las aceleraciones que este sufre, para cubrir la información de los otros tres grados de libertad. Para ello se integrará el código que lee los datos de los acelerómetros con el código que calcula los

ángulos de alabeo, cabeceo y guiñada, para determinar en cualquier punto el movimiento de nuestro vehículo. Presentamos a continuación el código denominado *accelsensor.*

```
/*-----------------------------------------------------------------------------------------------------------
*/
/*Bloque 1*/
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>

/* Assign a unique ID to this sensor at the same time */
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(54321);

/*-----------------------------------------------------------------------------------------------------------
*/
/*Bloque 2*/
void displaySensorDetails(void)
{
  sensor_t sensor;
  accel.getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
  Serial.print    ("Max   Value:        ");  Serial.print(sensor.max_value);  Serial.println("
m/s^2");
  Serial.print    ("Min   Value:        ");  Serial.print(sensor.min_value);  Serial.println("
m/s^2");
  Serial.print     ("Resolution:        ");   Serial.print(sensor.resolution);   Serial.println("
m/s^2");
  Serial.println("------------------------------------");
  Serial.println("");
  delay(500);
}

/*-----------------------------------------------------------------------------------------------------------
*/
/*Bloque 3*/
void setup(void)
{
  Serial.begin(9600);
  Serial.println("Accelerometer Test"); Serial.println("");

  /* Initialise the sensor */
  if(!accel.begin())
  {
```

```
  /* There was a problem detecting the ADXL345 ... check your connections */
  Serial.println("Ooops, no LSM303 detected ... Check your wiring!");
  while(1);
 }

 /* Display some basic information on this sensor */
 displaySensorDetails();
}

/*----------------------------------------------------------------------------------------------------
*/
/*Bloque 4*/
void loop(void)
{
 /* Get a new sensor event */
 sensors_event_t event;
 accel.getEvent(&event);

 /* Display the results (acceleration is measured in m/s^2) */
 Serial.print("X: "); Serial.print(event.acceleration.x); Serial.print("  ");
 Serial.print("Y: "); Serial.print(event.acceleration.y); Serial.print("  ");
 Serial.print("Z: ");  Serial.print(event.acceleration.z);  Serial.print("
");Serial.println("m/s^2 ");
 delay(500);
}
/*----------------------------------------------------------------------------------------------------
*/
```

- *Bloque 1*. El funcionamiento de este código es muy similar al anterior, y además, mucho más simple, tanto a nivel de la cantidad de información que se procesa como a nivel de procesamiento, ya que lo único que se realiza es una lectura de los sensores, sin obtener información a partir de ella. Por ello existe cierta correspondencia entre los bloques de los que consta, pero aún así algunos están incluidos en otros, por su mayor simplicidad. Para comenzar, este bloque, al igual que el *Bloque 1* de la función anterior, se encarga de llamar a las funciones necesarias incluídas en la librería, y además involucra la función del *Bloque 2* anterior de definir las variables, que en este caso solo es *accel*, asociada a los acelerómetros del sensor.

- *Bloque 2*. Este bloque es el encargado de mostrar por pantalla la información inicial del sensor que se mostraba en la imagen de la sección anterior, donde se describía la función de este código. Como se puede comprobar se muestra, por este orden, el nombre del sensor y la versión, su código de identificación, su rango de

valores posibles y su resolución. Finalmente se incluye un *delay* de medio segundo. Esta función será llamada posteriormente.

- *Boque 3*. Involucra la función *void setup*. Lo primero que se hace es iniciar la comunicación con el *Monitor Serial* a 9600 baudios, valor que habrá que compatibilizar en la integración con el programa anterior, para acto seguido mostrar un mensaje de iniciación: "*Accelerometer Test".* Tras esto se realiza el test del estado del sensor, al igual que hacía el *Bloque 3* del programa anterior, pero solo para el caso de los acelerómetros. Si se presenta algún error en el funcionamiento se mostrará el mensaje *"Ooops, no LSM303 detected ... Check your wiring!".* Finalmente se llama a la función del *Bloque 2* para que muestre la información relativa al sensor.

- *Bloque 4.* Abarca la función del bucle *void loop* cuya única y recurrente misión es obtener la lectura de los acelerómetros y mostrar los datos por pantalla.

4 Descripción del receptor GPS

4.1 Introducción y Motivación.

Los sistemas más antiguos de posicionamiento empleaban dispositivos como giróscopos mecánicos, muy voluminosos y poco precisos, y además, se realizaba poco procesamiento de la información obtenida. Por ello era frecuente una simple inspección visual de los instrumentos por parte del piloto, o contar con un ingeniero de vuelo cuya función era sacar conclusiones de los datos obtenidos. Más adelante, cuando se empezó a involucrar la electrónica en la aviación, se hizo posible la determinación de la trayectoria realizada, o la trayectoria futura estimada a partir de los datos actuales. Sin embargo era necesario eliminar el máximo de errores en las estimaciones realizadas, objetivo que no es posible cumplir en su totalidad, debido a que los sensores siempre ofrecen errores y ruido en las medidas que proporcionan. La redundancia de estos sistemas es un recurso muy empleado, no solo para disponer de sistemas de *back-up* ante el fallo de uno de ellos, sino también para comparar diferentes datos y obtener mejores aproximaciones mediante la computación de todo el conjunto de datos del que se dispone.

El paso siguiente son los sistemas *AHRS* actuales que cuentan con receptores de GPS para mejorar el posicionamiento de los vehículos donde se aplican. Se emplean algoritmos que actualizan repetitivamente la información y combinan los datos de los sensores inerciales y magnetómetros junto con la localización que ofrecen los satélites GPS, minimizando así los errores que estos sistemas pueden presentar.

Por todo ello se considera vital incluir un receptor GPS en cualquier dispositivo que pretenda desempeñar el papel de un AHRS, ya que en la actualidad se considerarían directamente como sistemas desactualizados.

Comenzaremos exponiendo brevemente una serie de datos básicos sobre la red GPS.

- Las siglas GPS significan Sistema de Posicionamiento Global, y consiste en una constelación de satélites desarrollada por el Departamento de Defensa de los Estados Unidos, permitiendo obtener una precisión a un objeto de centímetros en las coordenadas de su posición terrestre, si se utiliza GPS deferencial (DGPS), aunque lo normal son varios metros.

- La constelación está formada por 24 satélites con órbitas a 20200 km y 55 grados de inclinación, coordinadas para cubrir toda la superficie

de la Tierra. Estos determinan la posición del receptor en cuestión mediante la trilateración. Se determina la distancia de cada satélite al objeto empleando el tiempo que tarda una señal en recorrer la distancia entre él y el recetor. Las coordenadas son obtenidas con al menos 3 satélites y la altitud con un cuarto satélite.

- Los receptores de GPS reciben la señal proporcionada por cada satélite, que posee elementos que diferencian las señal de todos ellos, y que incluye la hora de la semana, de acuerdo a un preciso reloj atómico que llevan a bordo, el número de la semana GPS, un informe del estado del satélite pudiendo detectarse errores, y por supuesto, datos referentes a la localización. Cada transmisión dura 30 segundos, transmitiendo 1500 bits de información de datos codificados.

- Los receptores GPS, como el nuestro, emplean un sistema de comunicaciones denominado NMEA, definido por el National Marine Electronics Association. Se utiliza un simple protocolo ASCII de comunicaciones en serie, definiendo como es transmitida la información entre un talker y varios listeners al mismo tiempo. Es típico el uso de un Baud Rate de 4800. Las reglas básicas del protocolo consisten en comenzar cada sentencia con el signo del dólar, seguidos de cinco bits que identifican al emisor y al tipo de mensaje. Después se incluyen campos de información delimitados por una coma.

A pesar de todas las ventajas que ofrece, no es posible contar con él como un sistema primario de navegación, explicación por la que se suele tener como un sistema de apoyo. Principalmente, porque no ofrece una garantía de precisión lo suficientemente buena como para considerarlo como tal. Algunos estudios se han realizado para elaborar un método de asistencia al despegue, aproximación y aterrizaje mediante GPS, pero de momento no han sido lo suficientemente fructuosos como para sustituir los actuales ILS (Instrument Landing System). Y en segundo lugar, debido a motivos políticos, de modo que al ser propiedad de los Estados Unidos, ellos pueden ordenar la interrupción de su señal a su antojo. Además de este sistema, existe una red de posicionamiento ruso, GLONASS, y actualmente están en desarrollo un sistema europeo, Galileo, y uno chino, Beidou. Quizás la situación cambie cuando dispongamos de un sistema de posicionamiento propio en completo funcionamiento.

Visto lo esencial que se hace incluir un receptor de señal GPS en nuestro sistema, se ha seleccionado el Ultimate GPS Breakout - 66 channel w/10 Hz updates - Versión 3 de la compañía Adafruit. Sus características principales se detallan a continuación:

- La placa pesa unos 8.5 gramos y sus dimensiones son de 25.5 x 35 x 6.5.
- Sensibilidad de hasta -165dBm ($3.16 \cdot 10^{-17}\ mW$), actualizaciones a 10 Hz y 66 canales.
- Diseño preparado para alimentación con 5 V y 20 mA de corriente.
- Registro de datos integrado.
- PPS cuando obtiene una señal fix.
- Antena interna y conector u.FL para una antena activa externa.
- LED de estado integrado.
- Arquitectura basada en un chipset denominado MTK3339, que permite rastrear hasta 22 satélites. De este modo se consiguen hasta 10 posiciones por segundo, útil si el dispositivo se mueve a altas velocidades.
- Incluye un regulador de tensión permitiendo alimentarlo con 3.3 V. Resulta un consumo de 66mW.
- Se incluye un adaptador para pilas de botón, en el caso de que la batería interna que incluye se agote.
- La precisión estimada de este dispositivo es de menos de 3 metros, y la precisión en el cálculo de la velocidad es de 0.1 m, ya que los algoritmos incluidos también permiten la obtención de este parámeto, a partir de la tasa de variación de la posición.
- Como se describió anteriormente, la salida se realiza en formato NMEA 0183, por defecto a 9600 baud.
- Soporta los sistemas DGPS (GPS diferencial, que ofrece mayor precisión de hasta 10 cm, en el mejor de los casos), WAAS (sistema que se complementa con estaciones en tierra que minimizan posibles fuentes de error, desarrollado por Estados Unidos) y EGNOS (similar a los anterior pero Europeo, empleando una serie de satélites geoestacionarios y estaciones en tierra).

El diagrama de bloques de la arquitectura interna del receptor resulta como se muestra en la Figura 25, donde se han indicado los principales elementos que influirán nuestro uso.

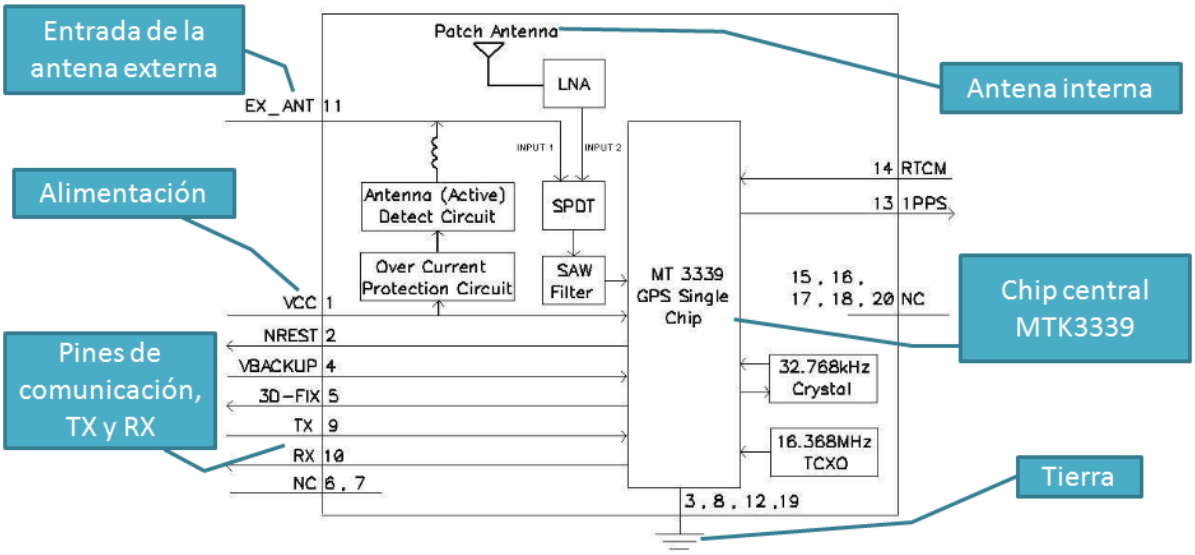


Figura 25. Diagrama de bloques del receptor GPS.

Además se muestra la apariencia del dispositivo en la Figura 26, indicando la correspondencia de los elementos del diagrama de bloques y los de la placa física.

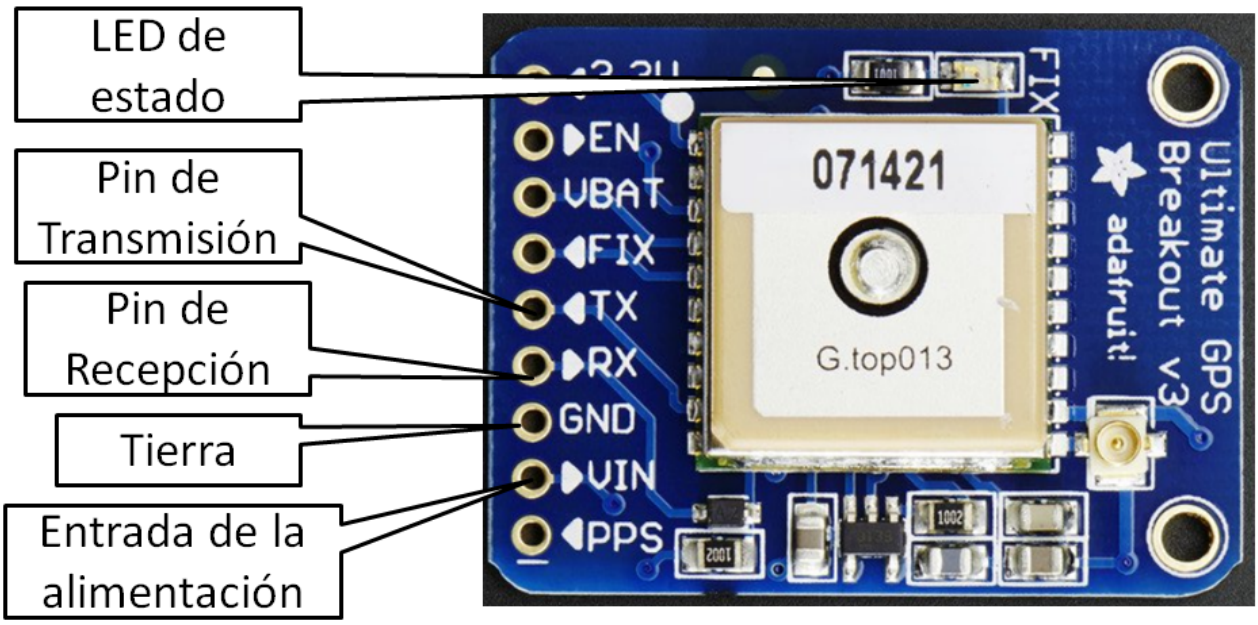


Figura 26. Apariencia del receptor GPS indicando sus pines principales

Como se puede comprobar, todos los pines indicados en el diagrama de bloques son localizables en la placa en sí, disponiendo de los dos pines que enviarán y recibirán información útil, el pin de alimentación y el pin de conexión con tierra. Además, se ha indicado también la existencia del LED de estado, que parpadea con una frecuencia de una vez por segundo cuando el dispositivo está buscando señal de los satélites (*tracking*), y lo hace cada quince segundos cuando consigue fijar la comunicación con estos.

4.2 Conexión con Arduino Due.

Al igual que ocurre con la IMU, este dispositivo es compatible con nuestra placa Arduino Due, sin más que realizar la conexión pertinente y cargar el programa, que al igual que en el caso anterior, proporciona el fabricante. Los pines a conectar se muestran en la Tabla 5.

Pin del GPS	Pin del Arduino
TX	RX pin 19
RX	TX pin 18
GND	GND
VIN	3.3 V power supply

Tabla 5. Conexiones entre la placa Arduino Due y el receptor GPS.

La conexión resulta finalmente como se muestra en la Figura 27.

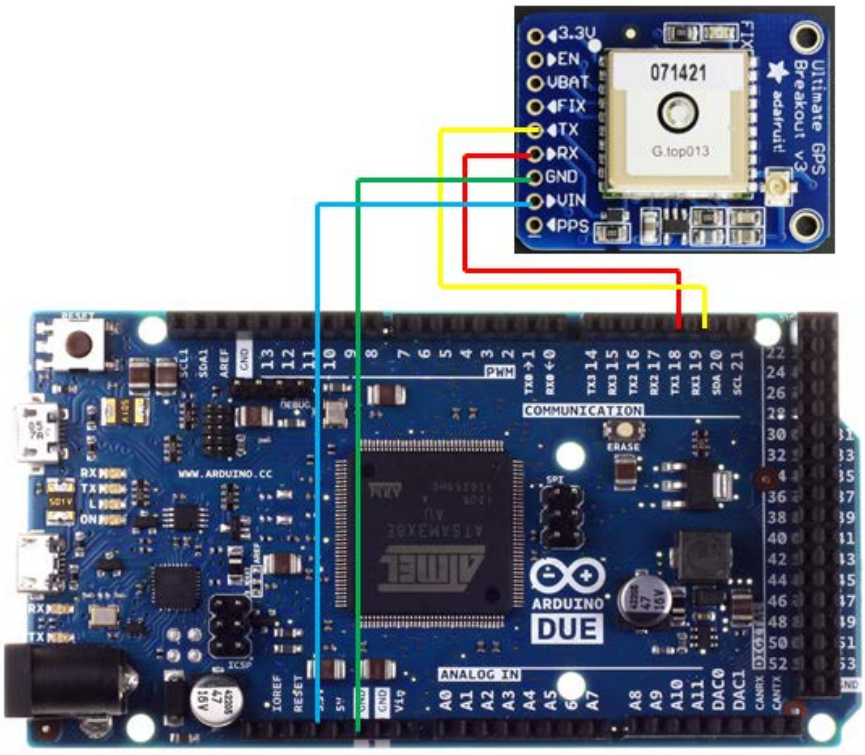


Figura 27. Diagrama de las conexiones entre la placa Arduino Due y el receptor GPS.

Un dato importante a destacar es que ninguno de los pines de comunicación interfiere con los pines que emplea la IMU, de modo que a la hora de integrar ambos, el trabajo resultara mucho más simple. Las comunicaciones de ambos con el microprocesador del Arduino no estarán acopladas en cuanto a cableado.

3.3 Software necesario para el funcionamiento del GPS.

Una vez que el dispositivo receptor de señales consigue comunicarse correctamente con los satélites y recibe la información codificada en las ondas, se extraen de ellas las sentencias NMEA que se presentaron con anterioridad, y que se detallarán más adelante. El protocolo NMEA indica que las sentencias incluyen toda la información posible en los campos de información, solamente separada por comas, por lo que más tarde se mostrará las sentencias en cuestión y toda la información que estas transportan. Por ellos es necesario un código que depure las sentencias y las transforme en datos más fáciles de leer y manipular. La compañía Adafruit ofrece una librería de códigos descargable, dentro de la cual encontramos el apropiado para el Arduino Due que usamos. Se muestra a continuación el código en cuestión separado por bloques, como se hizo en el caso de la IMU. Posteriormente se comentará la función de cada uno.

```
/*-------------------------------------------------------------------------------------------------*/
/*Bloque 1*/
#include <Adafruit_GPS.h>
#define mySerial Serial1

Adafruit_GPS GPS(&mySerial);

// Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial console
// Set to 'true' if you want to debug and listen to the raw GPS sentences.
#define GPSECHO  true

// this keeps track of whether we're using the interrupt
// off by default!
boolean usingInterrupt = false;
void useInterrupt(boolean); // Func prototype keeps Arduino 0023 happy
/*-------------------------------------------------------------------------------------------------*/

/*Bloque 2*/
void setup()
{

 // connect at 115200 so we can read the GPS fast enough and echo without dropping chars
```

```
  // also spit it out
  Serial.begin(115200);
  Serial.println("Adafruit GPS library basic test!");

  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some use 4800
  GPS.begin(9600);
  mySerial.begin(9600);
/*-----------------------------------------------------------------------------------------------*/

  /*Bloque 3*/
  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix data) including altitude
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // uncomment this line to turn on only the "minimum recommended" data
  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  // For parsing data, we don't suggest using anything but either RMC only or RMC+GGA since
  // the parser doesn't care about other sentences at this time

  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);  // 1 Hz update rate
  // For the parsing code to work nicely and have time to sort thru the data, and
  // print it out we don't suggest using anything higher than 1 Hz

  // Request updates on antenna status, comment out to keep quiet
  GPS.sendCommand(PGCMD_ANTENNA);

  // the nice thing about this code is you can have a timer0 interrupt go off
  // every 1 millisecond, and read data from the GPS for you. that makes the
  // loop code a heck of a lot easier!
/*-----------------------------------------------------------------------------------------------*/

/*Bloque 4*/
#ifdef __arm__
  usingInterrupt = false;  //NOTE - we don't want to use interrupts on the Due
#else
  useInterrupt(true);
#endif

  delay(1000);
  // Ask for firmware version
  mySerial.println(PMTK_Q_RELEASE);
}

#ifdef __AVR__
// Interrupt is called once a millisecond, looks for any new GPS data, and stores it
SIGNAL(TIMER0_COMPA_vect) {
```

```
  char c = GPS.read();
  // if you want to debug, this is a good time to do it!
#ifdef UDR0
  if (GPSECHO)
    if (c) UDR0 = c;
    // writing direct to UDR0 is much much faster than Serial.print
    // but only one character can be written at a time.
#endif
}

void useInterrupt(boolean v) {
  if (v) {
    // Timer0 is already used for millis() - we'll just interrupt somewhere
    // in the middle and call the "Compare A" function above
    OCR0A = 0xAF;
    TIMSK0 |= _BV(OCIE0A);
    usingInterrupt = true;
  } else {
    // do not call the interrupt function COMPA anymore
    TIMSK0 &= ~_BV(OCIE0A);
    usingInterrupt = false;
  }
}
#endif //#ifdef__AVR__
/*-----------------------------------------------------------------------------------------------------------*/

/*Bloque 5*/
uint32_t timer = millis();
void loop()                     // run over and over again
{
  // in case you are not using the interrupt above, you'll
  // need to 'hand query' the GPS, not suggested :(
  if (! usingInterrupt) {
    // read data from the GPS in the 'main loop'
    char c = GPS.read();
    // if you want to debug, this is a good time to do it!
    if (GPSECHO)
      if (c) Serial.print(c);
  }

  // if a sentence is received, we can check the checksum, parse it...
  if (GPS.newNMEAreceived()) {
    // a tricky thing here is if we print the NMEA sentence, or data
    // we end up not listening and catching other sentences!
    // so be very wary if using OUTPUT_ALLDATA and trytng to print out data
    //Serial.println(GPS.lastNMEA());   // this also sets the newNMEAreceived() flag to
false
```

```
   if (!GPS.parse(GPS.lastNMEA()))   // this also sets the newNMEAreceived() flag to
false
     return;  // we can fail to parse a sentence in which case we should just wait for
another
  }

  // if millis() or timer wraps around, we'll just reset it
  if (timer > millis())  timer = millis();

  // approximately every 2 seconds or so, print out the current stats
  if (millis() - timer > 2000) {
    timer = millis(); // reset the timer

    Serial.print("\nTime: ");
    Serial.print(GPS.hour, DEC); Serial.print(':');
    Serial.print(GPS.minute, DEC); Serial.print(':');
    Serial.print(GPS.seconds, DEC); Serial.print('.');
    Serial.println(GPS.milliseconds);
    Serial.print("Date: ");
    Serial.print(GPS.day, DEC); Serial.print('/');
    Serial.print(GPS.month, DEC); Serial.print("/20");
    Serial.println(GPS.year, DEC);
    Serial.print("Fix: "); Serial.print((int)GPS.fix);
    Serial.print(" quality: "); Serial.println((int)GPS.fixquality);
    if (GPS.fix) {
      Serial.print("Location: ");
      Serial.print(GPS.latitude, 4); Serial.print(GPS.lat);
      Serial.print(", ");
      Serial.print(GPS.longitude, 4); Serial.println(GPS.lon);

      Serial.print("Speed (knots): "); Serial.println(GPS.speed);
      Serial.print("Angle: "); Serial.println(GPS.angle);
      Serial.print("Altitude: "); Serial.println(GPS.altitude);
      Serial.print("Satellites: "); Serial.println((int)GPS.satellites);
    }
  }
}
/*-------------------------------------------------------------------------------------------------------*/
```

Para comenzar es necesario especificar que, a pesar de que el código está destinado a ser implementado exclusivamente con un Arduino Due, la compañía ha decidido hacerlo más flexible, de modo que se incluyen sentencias que analizan si nuestra arquitectura es ARM o AVR, actuando de un modo u otro debido a las diferencias existentes. Analizamos uno a uno los bloques que hemos definido en el código:

- *Bloque 1*. Este bloque realiza las funciones de iniciación del programa. Para comenzar, como es habitual, se incluye una función que debe estar almacenada en la librería, y que igualmente proporciona la compañía, Adafruit_GPS.h. Posteriormente se definen las comunicaciones entre la placa Arduino y el receptor de GPS. También, como se indica en el comentario, se establece la variable GPSECHO como verdadera, implicando que en nuestro caso las sentencias se mostrarán por pantalla. Para aplicaciones que no requieren esto, se deberá definir dicha variable como falsa. Y por último se crea la variable booleana *usingInterrupt*, como falsa. Esta variable cambiará a verdadera en el caso de que usemos una arquitectura AVR, utilizando una serie de interrupciones para el correcto funcionamiento del dispositivo.

- *Bloque 2*. El bloque dos comienza con el inicio de la función *void setup*. En ella se da comienzo a la comunicación con el *Monitor Serial* a 115200 baud y como indicativo de esto se imprime este mensaje por pantalla: "Adafruit GPS library basic test!". Igualmente se inicia la comunicación con la placa GPS a 9600 baud.

- *Bloque 3*. Dentro aún de la función *void setup*, se establecen una serie de sentencias que como podemos comprobar envían comandos al GPS. Estas son definidas por el fabricante del núcleo del GPS, que no es el mismo que el de la placa final que poseemos. Sus funciones son modificar el modo en que actúa el GPS, los datos que envía, etc. En nuestro caso, no realizaremos ningún cambio en este sentido.

- *Bloque 4*. El bloque 4 es el núcleo fundamental para aquellos que empleen este código con una arquitectura AVR. En nuestro caso no nos afecta, y podríamos eliminar este bloque entero. Como podemos observar se comienza con un bloque condicional, en el que se vuelve a definir la variable *usingInterrupt* como falsa si usamos una arquitectura ARM, como es nuestro caso, o como verdadera para el caso de emplear una arquitectura AVR. Después se inicia otro en el que se toman determinadas acciones en el caso de emplear un dispositivo AVR. Y finalmente se detalla la función *void useInterrupt*. No entraremos en más detalles, porque como hemos dicho, no nos afecta.

- *Bloque 5*. Esta función incluye el bucle del código por completo, y su función será la de leer repetitivamente las sentencias NMEA que se reciben de los satélites de GPS. Antes de entrar en el bucle, se define una variable contador *timer*, y se le da el valor de los milisegundos en el momento de su lectura. Acto seguido entramos en el bucle. En

primer lugar se realiza la lectura de los datos que recibe el GPS, y en caso de que tengamos activada la variable GPSECHO se muestran por pantalla las sentencias NMEA recibidas. Acto seguido se tiene un bloque que, mediante el uso de ciertas funciones predefinidas, analiza si se han obtenido nuevos datos, ya que en caso contrario se muestra por pantalla la última sentencia obtenida para evitarnos mostrar campos vacíos o repetitivos. Ahora aparecen un par de líneas que, debido a que la función *millis()* tiene un límite, resetea la variable *timer* en caso de que se supere este límite. Tras esto se tienen las sentencias que regulan lo que se muestra por pantalla, de modo que se imprimen los datos relacionados con el estado de la señal cada dos segundos, y, en el caso de que se obtenga un *fix,* se muestran datos más específicos de posicionamiento, etc., como se mostrará a continuación.

Finalmente el resultado que se muestra por pantalla es como se ve en la Figura 28.

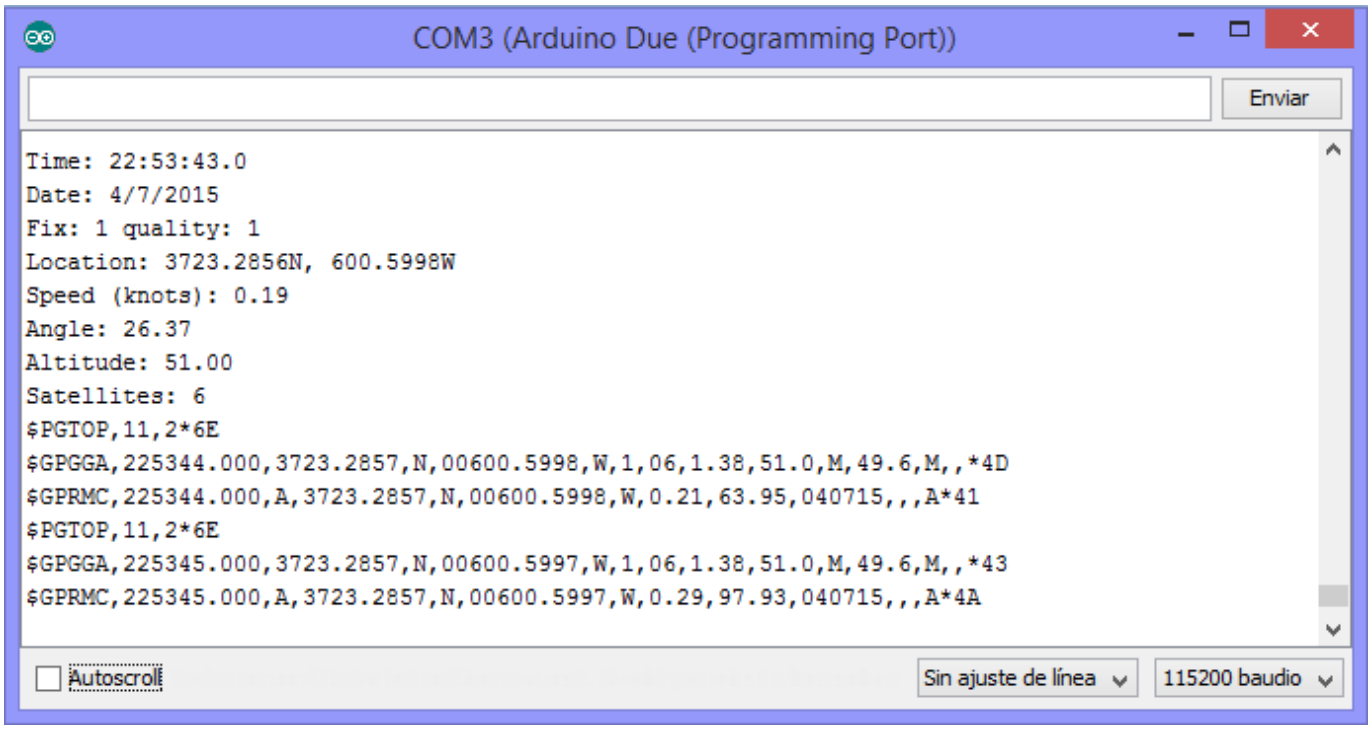


Figura 28. Resultados del código aplicable al receptor GPS, mostrados por el *Monitor Serial*

Los datos que se muestran son los siguientes:

- Hora GMT (Greenwich Mean Time), el que calculan los relojes atómicos a bordo de los satélites, y que utilizan para coordinarse

entre ellos y para calcular con exactitud las distancias necesarias en sus cálculos. Y la fecha actual en el meridiano de Greenwich.

- Tras estos datos se muestra el tipo de *fix* que se ha obtenido, de modo que solamente si se muestra un 1 significa que lo tenemos.

- Una vez obtenido el *fix* podemos consultar los datos de localización, que son los que se muestran a continuación. La estructura de las coordenadas resulta ser dos primeros bits para los grados, y el resto para los minutos con decimales. Tras esto, con la variación de nuestra posición, el código estima nuestra velocidad y nuestro rumbo (en el momento de la toma de la imagen el dispositivo no se encontraba en movimiento, pero los pequeños errores que se producen provoca que el programa calcule una pequeña velocidad y el rumbo). Además, cuando se cuenta con conexión a más de 3 satélites, es posible calcular también nuestra altitud sobre el nivel del mar.

- Finalmente se muestra el número de satélites de los que recibimos información.

Una vez mostrados los datos básicos, de una manera legible y cómoda de interpretar, se muestran las diferentes sentencias NMEA que recibe nuestro dispositivo, contando cada una con una estructura diferente. El conjunto de sentencias que podemos recibir son:

- GPGGA: *Global Positioning System Fixed Data*. Los valores que contiene, en este orden son el ID del mensaje, la hora UTC, la Latitud y Longitud con su indicador Norte/Sur Este/Oeste, el tipo de *fix*, el número de satélites en uso, la dilución de precisión horizontal, la altitud de la antena sobre el nivel del mar con su unidad, la separación geoidal con su unidad, un campo específico de DGPS (vacío si no se usa), y bits para el fin del mensaje.

- GPGSA: *GNSS DOP (dilución de la precisión del sistema global de navegación por satélite) and Active Satellites*. Los datos que se muestran son, en este orden, el ID del mensaje, modo Manual o Automático de posicionamiento 2D o 3D (se calcula la altura), tipo de Fix (0 cuando no existe, 1 para 2D y 2 para 3D), número de satélites usados, dilución de la precisión en posición, dilución horizontal de la precisión, dilución vertical de la precisión, bits de fin de mensaje.

- GPGSV: *GNSS Satellites in View*. Los valores que contiene, en este orden, son el ID del mensaje, el número de mensajes (de 1 a 3 dependiendo del número de satélites rastreados), número de satélites

a la vista, ID del satélite A, Elevación en grados de la posición del satélite A, Azimut de la posición del satélite A, SNR (indicador de la calidad de la señal) en decibelios del satélite A, repetición de estos datos para todos los satélites a la vista, y bits de fin del mensaje.

- GPRMC: *Recommended Minimum Navigation Information*. Los valores que contiene son, en este orden, el ID del mensaje, la hora UTC, el estado de la validez del mensaje, latitud y longitud con su indicador Norte/Sur Este/Oeste, velocidad sobre el suelo, rumbo, fecha, variación magnética, modo (A para autónomo, D para diferencial, y E para estimado) y bits para el fin del mensaje.

- GPVTG: *Course and Speed Information relative to the ground*. Los valores que contiene son, en este orden, el ID del mensaje, el curso en grados con la referencia *True*, el curso en grados con la referencia *Magnetic*, la velocidad y sus unidades, el modo (A para autónomo, D para diferencial, y E para estimado) y los bits de fin de mensaje.

- PGTOP: *Status of antenna*. Los datos que contiene hace alusión a la antena empleada, de modo que en este orden, se presenta el ID del mensaje, el ID del comando y el estado de la antena, siendo 1 para antena activa en cortocircuito, 2 para el uso de la antena interna y 3 para el uso de una antena externa.

3.4 Herramienta *MT3339 PC Tool*

El código anterior permite recibir las señales de los satélites y presentarlas adecuadamente en el *Monitor Serial*. Sin embargo, como ya se comentó anteriormente, el dispositivo GPS consta de un microprocesador programado de fábrica que, aunque es posible modificar algunas de sus características de funcionamiento, no es nada cómodo hacerlo mediante el envío de comandos a través del IDE del Arduino. El fabricante pone a nuestra disposición un Data Sheet del instrumento donde vienen listados todos los comandos y sus funciones, pero no es la manera idónea de actuar, ya que es un lenguaje poco intuitivo. Para ello, además, ofrece una herramienta, *MT3339 PC Tool*, que en primer lugar, muestra una presentación más clara e ilustrativa aún de la información que ya poseíamos con el código anterior. Además ofrece información complementaria y, como ya se ha planteado, permite manipular algunas alternativas interesantes.

Para el funcionamiento correcto de la misma, debemos tener guardadas en las librerías de Arduino el código anterior con todas las funciones complementarias. Además, debemos descargar y descomprimir la carpeta con la herramienta en cuestión. Una vez que el código está en correcto funcionamiento junto con el receptor, podemos abrir la herramienta. Es

importante tener el *Monitor Serial* cerrado, ya que si no se produce un error. En ella se presentarán varias pestañas, ofreciendo cada uno diferentes posibilidades.

- La pantalla principal, denominada *Skyplot*, ofrece la información más relevante, presentada de una manera estructurada y cómoda. Una vez seleccionado el puerto a través del que nos comunicamos con nuestro dispositivo y el Baud Rate adecuado, pulsamos *Open* y comenzamos a visualizar los datos. En la parte superior observamos la localización la hora y la fecha, y cuando está disponible, la información sobre cada satélite. En la zona inferior podemos observar un esquema del cielo visible por nuestro instrumento y la posición de cada satélite, cuando esta información está disponible. También podemos seleccionar el tipo de comienzo de la función TTFF pudiendo elegir entre:
  - *Cold Start*: este arranque se produce después de que el módulo GPS es apagado sin ser alimentado con una fuente adicional de *back-up*. Es el tiempo de inicio más largo de los tres, de media 35 segundos bajo cielo abierto, ya que la información sobre el almanaque (estado de los satélites que cada uno envía de sí mismo y de los demás, y que suele variar cada 4 meses) y la efemérides (posición precisa de cada satélite en un instante que envía cada uno de sí mismo y que suele variar cada 4 horas por las perturbaciones orbitales) debe ser cargada de nuevo con cada inicio del dispositivo receptor.
  - *Hot Start*: es el arranque que se produce asumiendo que el instrumento está siendo alimentado con una fuente externa, dentro de dos horas desde que fue apagado, estando los datos del almanaque y efemérides aún almacenados.
  - *Warm Start*: si el dispositivo es arrancado después de dos horas de su apagado, habiendo sido alimentado con una fuente de *back-up*, se inicia este arranque que simplemente refresca los datos que se posee de los satélites.

  Una vez que el dispositivo funciona correctamente, se puede visualizar una pantalla como la que se muestra en la Figura 29.

- La siguiente pestaña realiza principalmente la función que desempeña el Monitor Serial, mostrando la misma información, y además permite su almacenaje en la memoria del chipset si pulsamos Log NMEA.

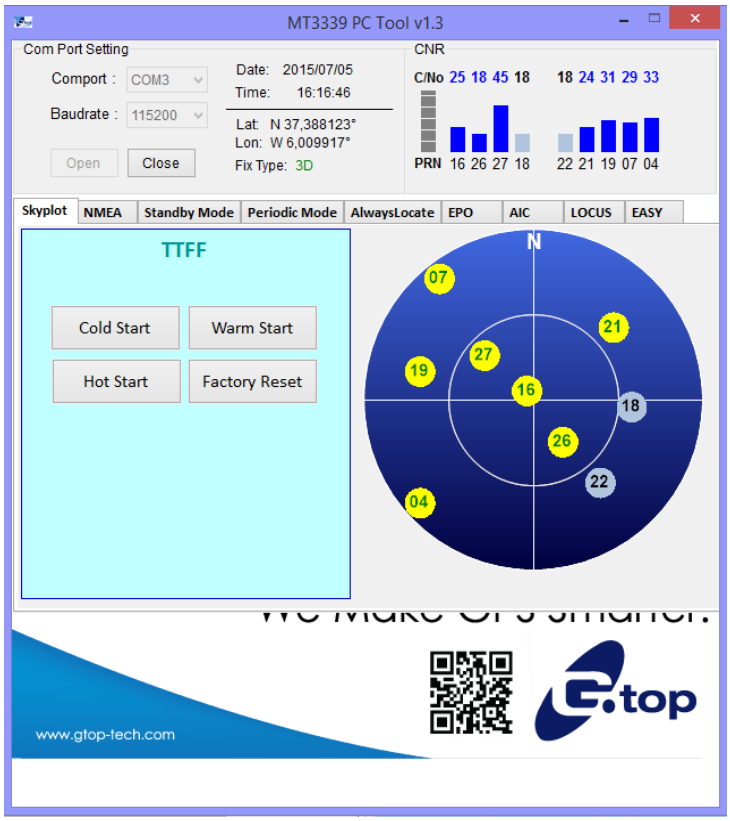


Figura 29. Ventana *Skyplot* de la herramienta MT3339

- La tercera pestaña, denominada *Standby Mode*, es una de las opciones que esta aplicación ofrece, destinada al ahorro de consumo de energía. Simplemente es un botón que permite activar y desactivar el receptor para evitar su consumo cuando no queremos que trabaje, sin necesidad de desconectar por completo el equipo. Se incluye un gráfico que representa el consumo a lo largo del tiempo.

- La cuarta pestaña ofrece el modo *Periodic Mode*. El modo anterior permitía encender y apagar el dispositivo a nuestro gusto. Este modo permite programar un encendido y apagado del dispositivo periódicamente, ajustable a nuestras necesidad. Es posible definir los intervalos de actividad e inactividad para alcanzar el funcionamiento más óptimo. Además el botón Set EPH Receiver permite al receptor prolongar el periodo de recepción de la efemérides.

- La quinta pestaña consta de otro modo de ahorro de consumo, denominado *AlwaysLocate*. Consiste en una programación inteligente que dependiendo del entorno y de las condiciones de movimiento del dispositivo, consigue alcanzar un equilibrio entre ahorro de energía y precisión del posicionamiento.

- El servicio EPO es presentado en la sexta pestaña. Las siglas significan

Extended Prediction Orbit, permitiendo la predicción de las órbitas durante 7 o 14 días mediante el algoritmo pertinente. Esta información se carga directamente en el dispositivo de recepción, mejorando su funcionamiento.

- La séptima pestaña incluye la aplicación AIC (Active Interference Cancellation), que como promete, mejora la navegación mediante la eliminación de las interferencias. Además, se incluye un gráfico que muestra a tiempo real el estado del *fix* del GPS.

- La octava pestaña constituye la función LOCUS, que se define como la solución innata del registro del microprocesador MT3339 incluido en nuestro dispositivo. Lleva asociados varios campos seleccionables para elegir el momento en que deseamos que se registre información, como por ejemplo, solamente cuando se consiga un 3D *fix*. Además el fabricante proporciona una tabla que especifica que información se almacena en cada momento, aunque también contamos con unos botones para modificar esta información. Al final una serie de botones permiten manipular toda la funcionalidad del sistema, como comenzar o detener el almacenamiento, consultar el estado del sistema, exportar los datos almacenados, etc. También se cuenta con una barra que muestra el porcentaje de almacenamiento ocupado.

- La última ventana ofrece la función EASY (Embadded Assist System). Ofrece tres ventajas: *EASY to TTFF*, que acelera la función de TTFF prediciendo la navegación por satélite mediante el empleo de las efemérides recibidas, *EASY to calcúlate*, que permite no dedicar un tiempo concreto a esta función, ya que se ejecuta eficientemente en los tiempos vacios de computación y *EASY to desig-in*, definida como la tecnología que no involucra esfuerzos adicionales de diseño. El botón que se muestra en la ventana permite activar o desactivar esta función.

La Tabla 6 y la Tabla 7 muestran la apariencia del resto de pestañas comentadas.

A pesar de todas las ventajas que esta aplicación ofrece, carece de la flexibilidad que buscamos a la hora de realizar la integración de la IMU y el GPS. Esta característica es aportada por el código principal que rige su funcionamiento, que aunque como ya hemos dicho, no permite la completa manipulación de las características de funcionamiento del dispositivo (aceptamos las especificaciones que trae por defecto, ya que nos son suficientes), nos permitirá trabajar con las sentencias básicas. Procedemos por tanto a presentar el dispositivo final que recibe datos tanto de la IMU como de GPS.

Tabla 6. Apariencia de algunas pestañas de la herramienta MT3339 (I).

Tabla 7. Apariencia de algunas pestañas de la herramienta MT3339 (II)

5 Integración Final

5.1 Introducción.

Como ya se ha dicho, una vez presentada la placa que integrará todos los sensores de medidas inerciales, los sensores de datos de aire, y además, el GPS, solo resta realizar la puesta a punto del conjunto para que trabaje como un sistema interrelacionado.

Para comenzar necesitamos realizar la toma de datos de los acelerómetros, ya que será necesario conocer las aceleraciones que sufre nuestro vehículo en todo momento, para en un futuro, poder estimar la posición en la que nos encontramos respecto a la inicial. Como ya comentamos, disponemos del código que extrae los datos de las aceleraciones de los acelerómetros de nuestra IMU y los muestra por pantalla. Por otro lado interesa determinar los ángulos de alabeo, cabeceo y guiñada para conocer en todo momento el rumbo en las tres dimensiones que está tomando nuestro UAV. Por esto, también contamos con el código referente al rumbo, que no solo calcula los giros sufridos por nuestro instrumento, sino que transforma dichos movimientos en los ángulos comentados. Este paso que ya está programado nos ahorra el trabajo de realizar el procesamiento de dichos datos. Además, obtener dichos ángulos a partir de los datos extraídos de los sensores nos indica que quizás ya se está aplicando implícitamente un filtro del ruido, por ejemplo un simple filtro de paso bajo o alguno similar al filtro Kalman.

En segundo lugar, tendremos que realizar, coordinadamente con el proceso anterior, la toma de sentencias NMEA del receptor de GPS. Ya disponemos del código que realiza esta función de forma independiente, pero ahora habrá que combinar este con el código anterior de manera que se muestre por pantalla el posicionamiento y el resto de datos que calcula el sistema GPS.

En este punto, interesa definir el alcance de nuestro trabajo. Aspiramos a que nuestro dispositivo sea capaz de leer conjuntamente los datos de la IMU y los datos del GPS, evitando que las sentencias colapsen a la hora de su computación, acabando por no obtener un resultado satisfactorio. Y además, una vez obtenidas estas, queremos que se muestren por pantalla o que se cuente con la posibilidad de extraer dicha información a través de cualquier otro pin de la placa Arduino. No pretendemos por tanto combinar ambos datos para que la navegación del UAV sea más satisfactoria, como se realiza en los *AHRS* actuales o cualquier sistema moderno de navegación, ya que no disponemos de los recursos suficientes para ello. Siendo conscientes de que este paso sería fundamental para situar nuestro

dispositivo en el mercado, la dejamos como posible línea de desarrollo futuro.

5.2 Integración del hardware

Lo primero que debemos realizar a la hora de poner en común la pareja de dispositivos que ya hemos expuesto, es diseñar la estructura física que tendrá, en el sentido de las conexiones necesarias para que su función se realice de manera correcta.

En primer lugar, nuestra IMU se comunica con la placa Arduino mediante el protocolo I2C necesitando para ello lo que podríamos denominar como dos líneas de datos. El primero es el SDA, que será a través del cual se transmitan los datos en serie, y el SCL, que actúa como reloj coordinando durante todo el proceso de comunicación.

En segundo lugar, el dispositivo GPS se comunica con la placa Arduino a través de los pines *RX* y *TX*, de comunicaciones en serie.

Por todo ello y afortunadamente, las conexiones entre la placa Arduino y ambos dispositivos no colisionan físicamente. Esto nos permite conectar directamente estos cuatro buses por separado, consiguiendo que las comunicaciones se realicen con normalidad según se programen en el código.

Realmente también hay que incluir otras dos líneas que son necesarias para conectar ambos dispositivos. Estas son la toma de alimentación y la toma de tierra. Como en ambos casos es posible realizar la alimentación a 3.3 V se extraerá una toma común para ambos, al igual que sucederá con la tierra.

Finalmente, la conexión resulta como se muestra en la Figura 30.

5.3 Integración del Software

Una vez que disponemos de los tres códigos, los dos que extraen datos de la IMU y el que interactúa con el GPS, y todas las funciones necesarias incluidas en nuestra librería, podemos proceder a combinar sus sentencias para obtener un código que realice la función deseada. Comentamos punto por punto los pasos seguidos:

- Principalmente se ha combinado ordenadamente ambos códigos, es decir, respetando las funciones que se creaban y manteniendo la estructura existente dentro de cada función *void setup* o *void loop*. De este modo incluíamos todo lo necesario en el código final. Sin embargo, había que eliminar las redundancias e incongruencias que se producían.

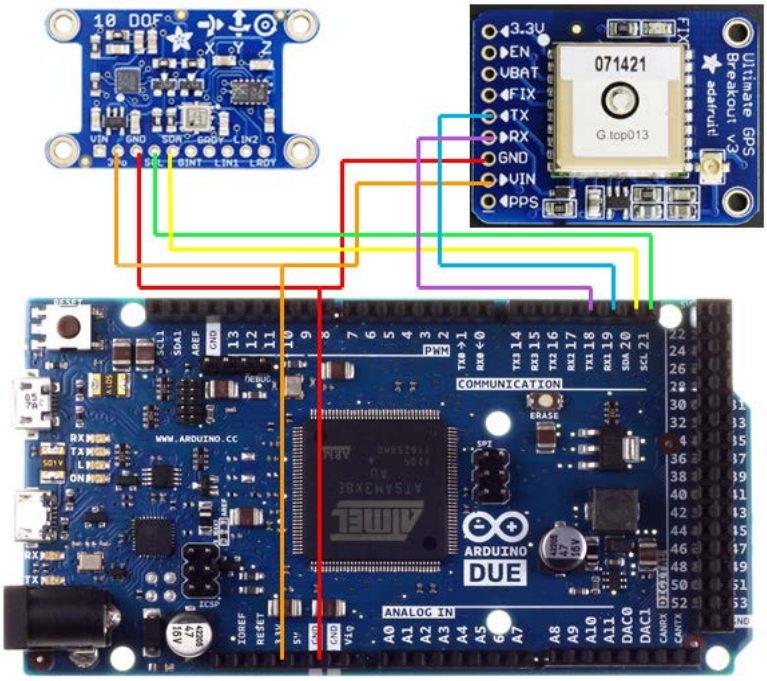


Figura 30. Diagrama de las conexiones entre la placa Arduino, la IMU y el receptor GPS.

- En este punto detectamos que una de las funciones que empleábamos establecía una comunicación con el Monito Serial a un Baud Rate diferente de las otras dos. Por ello se tuvo que definir una sola sentencia de este tipo, finalizando con un Baud Rate de 115200 baudios.

- Además, el código está diseñado para que emita ciertos mensajes de iniciación, que expusimos en su momento, para dar comienzo al funcionamiento de cada sensor por separado. Estos han quedado como comentarios y en su lugar se han sustituido por el mensaje “IMU + GPS”. Aún así se han respetado las funciones que detectaban si el funcionamiento de todo el sistema estaba en correcto estado, devolviendo un mensaje de advertencia en caso contrario.

- Dentro del código del GPS existían unas líneas que comprobaban si estábamos empleando una arquitectura AVR, y en este caso se realizaban unas determinadas operaciones para actuar con esta arquitectura de un modo diferente. Como este no es nuestro caso, ya que utilizamos el procesador del Arduino Due con arquitectura ARM, estas líneas acaban apareciendo como comentarios para posibles modificaciones. De esta forma ahorramos espacio en la memoria del Arduino, el proceso de carga se realiza más velozmente y la visualización del código es más simple.

- La medida más importante que se ha tenido que tomar se encuentra dentro del bucle de funcionamiento. A causa de que las sentencias pueden colisionar a la hora de ser procesadas, esta parte del código ha sido dividida en el tiempo en dos módulos. Este proceso de *scheduling* se ha realizado con ayuda de la función millis(). Así durante los dos primeros segundos desde el inicio del código se ejecuta la parte del código que controla el GPS, recibiendo e interpretando las sentencias NMEA. Se ha comprobado que realizar este proceso dos segundos ha sido óptimo, ya que es el margen que se establece de fábrica para emitir los resultados obtenidos, siendo compatible con el intervalo de tiempo destinado a ello. Tras esto se ejecuta el código relativo a la IMU una vez, que es mucho más simple, ya que solo se tiene que extraer los datos de los sensores que se producen en ese instante, y emitirlos por la vía que corresponda. Estos intervalos de tiempo son fácilmente modificables sin más que cambiar directamente los parámetros que los definen.

A continuación procedemos a presentar el código dividido en bloques como ya se hizo anteriormente:

```
/*-----------------------------------------------------------------------------------------------------------
*/
/*Bloque 1*/
/*Incluir funciones propias del GPS*/
#include <Adafruit_GPS.h>
#define mySerial Serial1
Adafruit_GPS GPS(&mySerial);
#define GPSECHO  false
boolean usingInterrupt = false;
//void useInterrupt(boolean);

/*Incluir funciones propias de la IMU*/
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>
#include <Adafruit_BMP085_U.h>
#include <Adafruit_10DOF.h>

/*Asignar denominaciones únicas a los sensores */
Adafruit_10DOF                dof   = Adafruit_10DOF();
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified   mag   = Adafruit_LSM303_Mag_Unified(30302);
Adafruit_BMP085_Unified       bmp   = Adafruit_BMP085_Unified(18001);

/* Update this with the correct SLP for accurate altitude measurements */
float seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;
```

```
/* Assign a unique ID to this sensor at the same time
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(54321);*/

/*-------------------------------------------------------------------------------------------------------
*/
/*Bloque 2*/
void initSensors()
{
  if(!accel.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println(F("Ooops, no LSM303 detected ... Check your wiring!"));
    while(1);
  }
  if(!mag.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println("Ooops, no LSM303 detected ... Check your wiring!");
    while(1);
  }
  if(!bmp.begin())
  {
    /* There was a problem detecting the BMP180 ... check your connections */
    Serial.println("Ooops, no BMP180 detected ... Check your wiring!");
    while(1);
  }
}

//void displaySensorDetails(void)
//{
//  sensor_t sensor;
//  accel.getSensor(&sensor);
//  Serial.println("------------------------------------");
//  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
//  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
//  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
//  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println("
m/s^2");
//  Serial.print  ("Min Value:    "); Serial.print(sensor.min_value); Serial.println("
m/s^2");
//  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution); Serial.println("
m/s^2");
//  Serial.println("------------------------------------");
//  Serial.println("");
//  delay(500);
//}
```

```
/*-------------------------------------------------------------------------------------------------------
 */
/*Bloque 3*/
void setup() {

  /*Líneas del GPS*/
  Serial.begin(115200);
//  Serial.println("Adafruit GPS library basic test!");

  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some use 4800
  GPS.begin(9600);
  mySerial.begin(9600);

  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix data) including altitude
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // uncomment this line to turn on only the "minimum recommended" data
  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  // For parsing data, we don't suggest using anything but either RMC only or RMC+GGA since
  // the parser doesn't care about other sentences at this time

  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);  // 1 Hz update rate
  // For the parsing code to work nicely and have time to sort thru the data, and
  // print it out we don't suggest using anything higher than 1 Hz

  // Request updates on antenna status, comment out to keep quiet
  GPS.sendCommand(PGCMD_ANTENNA);

/*-------------------------------------------------------------------------------------------------------
 */
/*Bloque 4*/
  /*Líneas de la IMU- PitchRollHeadingTemperature*/
  Serial.println(F("IMU + GPS")); Serial.println("");

  /* Initialise the sensors */
  initSensors();

  /*Líneas de la IMU- Aceleraciones*/
//Serial.println("Accelerometer Test"); Serial.println("");

  /* Initialise the sensor */
  if(!accel.begin())
  {
    /* There was a problem detecting the ADXL345 ... check your connections */
```

```
   Serial.println("Ooops, no LSM303 detected ... Check your wiring!");
   while(1);
  }
//  displaySensorDetails();
}

/*-----------------------------------------------------------------------------------------------------------
 */
/*Bloque 5*/
  // the nice thing about this code is you can have a timer0 interrupt go off
  // every 1 millisecond, and read data from the GPS for you. that makes the
  // loop code a heck of a lot easier!

//#ifdef __arm__
//  usingInterrupt = false;  //NOTE - we don't want to use interrupts on the Due
//#else
//  useInterrupt(true);
//#endif

//  delay(1000);

 // Ask for firmware version
  mySerial.println(PMTK_Q_RELEASE);

//#ifdef __AVR__
//// Interrupt is called once a millisecond, looks for any new GPS data, and stores it
//SIGNAL(TIMER0_COMPA_vect) {
//  char c = GPS.read();
//  // if you want to debug, this is a good time to do it!
//#ifdef UDR0
//  if (GPSECHO)
//    if (c) UDR0 = c;
//    // writing direct to UDR0 is much much faster than Serial.print
//    // but only one character can be written at a time.
//#endif
//}
//
//void useInterrupt(boolean v) {
//  if (v) {
//    // Timer0 is already used for millis() - we'll just interrupt somewhere
//    // in the middle and call the "Compare A" function above
//    OCR0A = 0xAF;
//    TIMSK0 |= _BV(OCIE0A);
//    usingInterrupt = true;
//  } else {
//    // do not call the interrupt function COMPA anymore
//    TIMSK0 &= ~_BV(OCIE0A);
```

```
//    usingInterrupt = false;
//  }
//}
//#endif //#ifdef__AVR__

/*-----------------------------------------------------------------------------------------------------------
*/
/*Bloque 6*/
uint32_t timer = millis();
uint32_t timer2 = millis();

void loop() {

   // if millis() or timer wraps around, we'll just reset it
  if (timer > millis())  timer = millis();
   if (timer2 > millis())  timer2 = millis();

   if (millis()-timer2>1000){
    if (millis()-timer2 < 2000){
// /*Líneas del bucle del GPS*/
  // in case you are not using the interrupt above, you'll
  // need to 'hand query' the GPS, not suggested :(
  if (! usingInterrupt) {
   // read data from the GPS in the 'main loop'
   char c = GPS.read();
   // if you want to debug, this is a good time to do it!
   if (GPSECHO)
     if (c) Serial.print(c);
  }

  // if a sentence is received, we can check the checksum, parse it...
  if (GPS.newNMEAreceived()) {
   // a tricky thing here is if we print the NMEA sentence, or data
   // we end up not listening and catching other sentences!
   // so be very wary if using OUTPUT_ALLDATA and trytng to print out data
   //Serial.println(GPS.lastNMEA());   // this also sets the newNMEAreceived() flag to
false

   if (!GPS.parse(GPS.lastNMEA()))   // this also sets the newNMEAreceived() flag to
false
     return;  // we can fail to parse a sentence in which case we should just wait for
another
   }

  // approximately every 2 seconds or so, print out the current stats
  if (millis() - timer > 2000) {
   timer = millis(); // reset the timer
```

```
  Serial.print("\nTime: ");
  Serial.print(GPS.hour, DEC); Serial.print(':');
  Serial.print(GPS.minute, DEC); Serial.print(':');
  Serial.print(GPS.seconds, DEC); Serial.print('.');
  Serial.println(GPS.milliseconds);
  Serial.print("Date: ");
  Serial.print(GPS.day, DEC); Serial.print('/');
  Serial.print(GPS.month, DEC); Serial.print("/20");
  Serial.println(GPS.year, DEC);
  Serial.print("Fix: "); Serial.print((int)GPS.fix);
  Serial.print(" quality: "); Serial.println((int)GPS.fixquality);
  /*Se eescriben los datos del GPS solo si se consigue un fix*/
  if (GPS.fix)
  {
    Serial.print("Location: ");
    Serial.print(GPS.latitude, 4); Serial.print(GPS.lat);
    Serial.print(", ");
    Serial.print(GPS.longitude, 4); Serial.println(GPS.lon);

    Serial.print("Speed (knots): "); Serial.println(GPS.speed);
    Serial.print("Angle: "); Serial.println(GPS.angle);
    Serial.print("Altitude: "); Serial.println(GPS.altitude);
    Serial.print("Satellites: "); Serial.println((int)GPS.satellites);
     }
    }
   }

/*-------------------------------------------------------------------------------------------------------
 */
/*Bloque 7*/
   if (millis()-timer2>2000){

 sensors_event_t accel_event;
 sensors_event_t mag_event;
 sensors_event_t bmp_event;
 sensors_vec_t   orientation;

 /* Calculate pitch and roll from the raw accelerometer data */
 accel.getEvent(&accel_event);
 if (dof.accelGetOrientation(&accel_event, &orientation))
 {
  /* 'orientation' should have valid .roll and .pitch fields */
  Serial.print(F("Roll: "));
  Serial.print(orientation.roll);
  Serial.print(F("; "));
  Serial.print(F("Pitch: "));
```

```
  Serial.print(orientation.pitch);
  Serial.print(F("; "));
 }

 /* Calculate the heading using the magnetometer */
 mag.getEvent(&mag_event);
 if (dof.magGetOrientation(SENSOR_AXIS_Z, &mag_event, &orientation))
 {
  /* 'orientation' should have valid .heading data now */
  Serial.print(F("Heading: "));
  Serial.print(orientation.heading);
  Serial.print(F("; "));
 }

 /* Calculate the altitude using the barometric pressure sensor */
 bmp.getEvent(&bmp_event);
 if (bmp_event.pressure)
 {
  /* Get ambient temperature in C */
  float temperature;
  bmp.getTemperature(&temperature);
  /* Convert atmospheric pressure, SLP and temp to altitude    */
  Serial.print(F("Alt: "));
  Serial.print(bmp.pressureToAltitude(seaLevelPressure,
                                      bmp_event.pressure,
                                      temperature));
  Serial.print(F(" m; "));
  /* Display the temperature */
  Serial.print(F("Temp: "));
  Serial.print(temperature);
  Serial.print(F(" C"));
 }

 Serial.println(F(""));
sensors_event_t event;
 accel.getEvent(&event);

 /* Display the results (acceleration is measured in m/s^2) */
 Serial.print("X: "); Serial.print(event.acceleration.x); Serial.print("  ");
 Serial.print("Y: "); Serial.print(event.acceleration.y); Serial.print("  ");
 Serial.print("Z:          "); Serial.print(event.acceleration.z);          Serial.print("
");Serial.println("m/s^2 ");
  timer2 = millis(); // reset the timer
  }
}
/*-----------------------------------------------------------------------------------------------------------
*/
```

- *Bloque 1*. Como se realizaba en cada uno de los programas por separado, se comienza incluyendo funciones necesarias para cada uno de ellos, y se definen algunas variables de utilidad. Se comienza con las del GPS y se continúa con las de la IMU. Una modificación que se ha hecho en este punto ha sido establecer la variable *GPSECHO* como falsa, consiguiendo así que las sentencias NMEA no se muestren por pantalla. Ya que no son fácilmente interpretables, y en esta ocasión estamos incluyendo un mayor número de datos, evitamos sobrecargar la información que se muestra. Tras esto se otorgan denominaciones adecuadas a cada variable que se empleará, como ocurrió anteriormente, y se determina la presión a nivel del mar.

- *Bloque 2*. En este caso se define la función que, al igual que en el código sin integrar, se encargaba de alertar si algo iba mal en la información que ofrecían los sensores. Desarrolla el mismo comportamiento que tenía antes. Al final se muestra comentada la función que imprimía los detalles de los sensores de la IMU como motivo de la depuración efectuada a los datos en pantalla.

- *Bloque 3*. Con el bloque tres se da comienzo a la función *void setup*. Al inicio de la misma como es habitual, se ponen en marcha las comunicaciones, a 115200 baud con el *Monitor Serial*, y a 9600 baud las comunicaciones con el GPS. Al finalizar esto se da paso a la serie de sentencia que, como se explicó, envían órdenes al procesador del GPS para definir su comportamiento. Tras esto se incluye varias líneas que se han comentado por no ser útiles en nuestro caso.

- *Bloque 4*. Este módulo se incluye aún dentro de la función *void setup* y comienza imprimiendo por pantalla el mensaje de inicio indicado anteriormente: “IMU + GPS”. Tras esto se define la función que comprueba el estado de los sensores. Finamente aparece comentada la llamada a la función que mostraba las propiedades de los mismos.

- *Bloque 5*. El quinto bloque incluye todo el conjunto de sentencias que redirigen el programa de un modo u otro dependiendo de la arquitectura que empleemos. Todo él aparece comentado, ya que como se dijo, no nos es útil.

- *Bloque 6*. Este bloque comienza dos sentencias antes de la función *void loop*. Dichas sentencias definen dos variables altamente importantes para lo que se muestra después: las variables *timer* y *timer2*. En ambas es almacenado el valor de los milisegundos que la función *milllis()* evalúa en ese instante. Acto seguido se da comienzo al

bucle principal. Y lo primero que se ejecuta es el reseteo de ambos *timer* en el caso de que la función *millis()* sobrepase se valor máximo. Este paso es muy necesario para la integridad del código, y una vez definido podemos continuar. Lo siguiente son dos bloques condicionales que permiten la ejecución del código que contienen en el caso de que nos encontremos entre uno y dos segundos desde la definición de la variable *timer2*. Dentro de ellos se encuentran las sentencias que controlan la transmisión y el procesamiento de la información que capta el GPS. Sin embargo, los datos solo se mostrarán por pantalla una vez cada dos segundos, o sea, una sola vez desde que la secuencia de ejecución entra dentro de este bloque, para lo que se emplea la variable *timer*.

- *Bloque 7*. Una vez que transcurren más de dos segundos desde que se define la variable *timer2*, el código solo nos permite, a través de otro bloque condicional, ejecutar las sentencias que se corresponden con la IMU. Este bloque solo se reproduce una vez desde que se inicia, a diferencia del anterior, que se repetía dentro del margen de tiempo establecido. Esto se debe a que simplemente lee y muestra por pantalla, como ya se comentó por separado en las secciones anteriores, los datos que recibe la IMU. Una vez que esto se ha completado, el *timer2* se redefine con el valor que posee la función *millis()* en este momento, dando paso de nuevo al bloque de sentencias del GPS, repitiéndose el bucle de nuevo.

De esta manera se acaba consiguiendo el resultado esperado. Los datos procedentes de los sensores inerciales y del posicionamiento del GPS se van sucediendo en el monitor serial repetitivamente. Se ha respetado el comportamiento del programa del GPS, que solo ofrecía nuestras coordenadas, altitud, velocidad y demás en el caso de obtener un *fix*, de modo que si no se obtiene simplemente se muestra un bloque más pequeño que el deseado, donde se muestra "*Fix = 0*", dándonos a entender que no contamos con toda la información necesaria.

La Figura 31 muestra cómo se visualiza la información en el *Monitor Serial*. Como se puede comprobar, y como era de esperar después de conocer el funcionamiento del código, se muestran primero los datos del GPS y después los datos de rumbo y aceleraciones de la IMU. Por tanto, podemos asegurar que hemos cumplido nuestro objetivo.

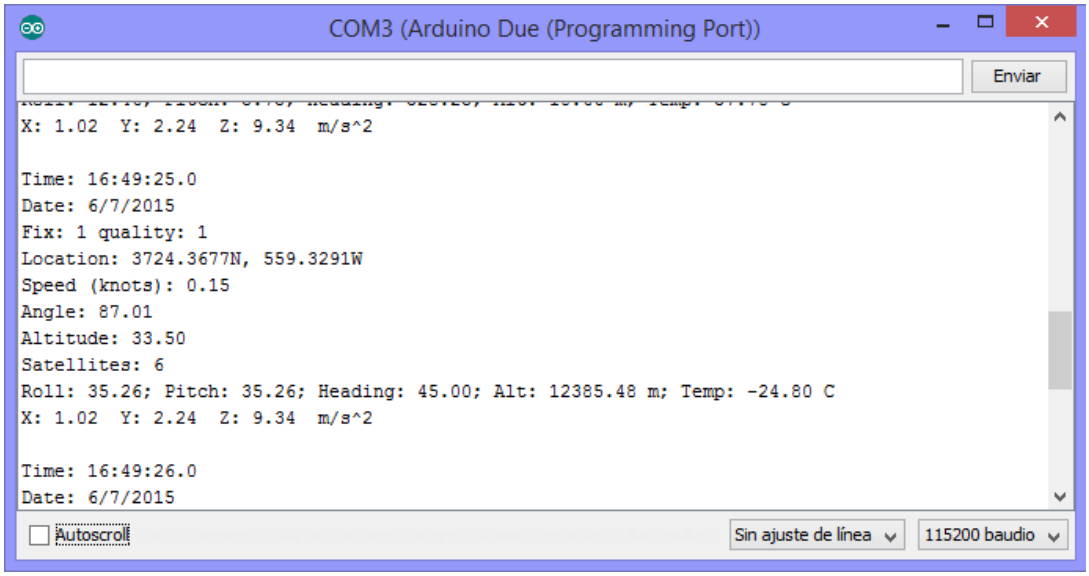


Figura 31. Resultados obtenidos con el código integrado mostrados por el *Monitor Serial*

5.4 Presentación de los resultados

Para finalizar, queda plantear la manera en la que estos datos serían enviados al resto de dispositivos de aviónica que los requieran, como podría ser un Autopiloto o un FCC (Computador de Datos de Vuelo). En nuestro caso, que nos encontramos en el punto del diseño del conjunto, hemos enviado estos datos al computador mediante el comando *Serial.print()*. Para nuestro objetivo esto es suficiente, ya que nos permite analizar los datos obtenidos e implementar las modificaciones pertinentes. Sin embargo, esto no es de aplicación a la hora de comunicarnos con uno de los protocolos empleados en las comunicaciones en la aviónica de los UAVs. Como referente a estos tenemos los detectados durante el estudio de mercado realizado en el apartado 2.

Como solución a la cuestión que se nos plantea, y debido a las posibilidades que la plataforma Arduino nos ofrece, se ha elegido extraer los datos por los pines de comunicaciones CAN que existen en la placa. La justificación de esta decisión ha sido, principalmente, el amplio uso de este protocolo en el sector de la aviónica de los UAV, como quedó demostrado en el análisis de los sistemas existentes. Se muestra en la Figura 32 los pines a los que nos referimos.

Una vez que obtenemos los datos en el protocolo BUS CAN, es más cómodo manipularlos debido a que existen multitud de dispositivos en el mercado que actúan de interfaz entre los diferentes protocolos que se emplean en la realidad. Por tanto podemos asegurar que mediante estos dispositivos tenemos alcance a cualquiera de los demás protocolos de comunicaciones. Un ejemplo de estos sistemas es el *Interface Control*

Processor de la compañía *RADA* que se presentó en el apartado del estudio de mercado.

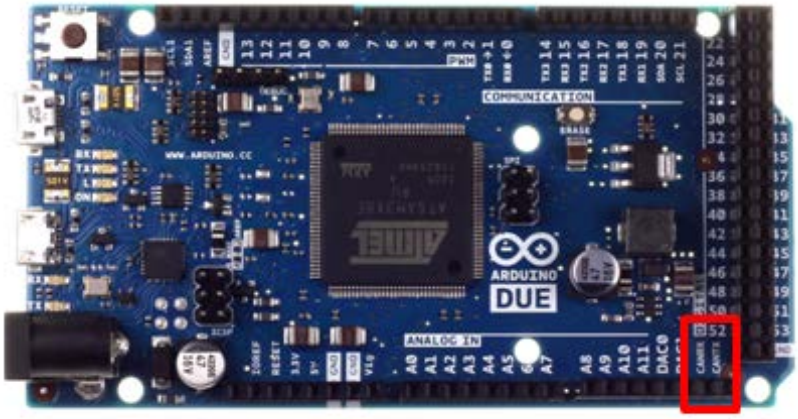


Figura 32. Posición de los pines referentes al BUS CAN.

6 Conclusiones y Líneas Futuras

Con este proyecto se ha pretendido estudiar en profundidad la aviónica que se aplica hoy en día en los UAVs, centrándonos en los dispositivos de posicionamiento. Como se desarrolló al comienzo de la memoria, se empezó estudiando las características de los UAVs, desde su nivel más general hasta los detalles de los sistemas que las empresas sitúan hoy en día en el mercado. De esto modo se ha caracterizado la aviónica que podemos encontrarnos en la actualidad. Una vez justificada la importancia tan crítica que tienen los UAVs en nuestros días, y más en concreto, sus sistemas de aviónica, se consideró oportuno implementar los conocimientos acumulados hasta el momento en el prototipo desarrollado en este Trabajo Fin de Grado. Por ello se pasó al estudio de dos de los dispositivos electrónicos, ya no usuales entre los que se incluyen en un vehículo no tripulado, sino de obligada presencia. Como se pudo ver en la tabla que resume los datos de los sistemas en el mercado, todos ellos cuentan con unos instrumentos que recogen datos inerciales y, la gran mayoría, también, con un receptor de señal GPS. Dentro del tiempo del que se ha dispuesto, se ha realizado dicho estudio tanto a nivel software como a nivel hardware de la IMU y GPS elegidos. Finalmente se ha estudiado cómo integrar ambos dispositivos de forma similar a como se procede con los sistemas más usuales de posicionamiento y detección de la actitud de cualquier vehículo. Asimismo, se ha llevado a cabo su implementación, tanto a nivel de hardware como software.

Con todo ello pretendemos demostrar que con los componentes electrónicos básicos que podemos adquirir a través de Internet, podemos desarrollar prototipos de elementos de aviónica en su primera aproximación, a la altura de los dispositivos electrónicos empleados actualmente en los UAVs. Se puede deducir de esto que estamos viviendo una época en la que todos los recursos a nuestro alcance encierran un potencial tecnológico oculto bajo la gran flexibilidad y posibilidades que involucran. Como se ha pretendido demostrar, basta con conseguir dos dispositivos de precio no elevado para finalmente obtener un sistema de aviónica básico en el entorno en el que nos movemos. Lejos queda la época en que estos elementos eran prácticamente inalcanzables fuera de las empresas que los desarrollaban. Esta facilidad está fomentada por empresas como Adafruit que consiguen hacerlo más sencillo y poner al alcance de cualquiera con conocimientos básicos en el campo la construcción de sistemas electrónicos que realicen funciones destacables. Poseer conocimientos básicos de programación en el lenguaje apropiado y de electrónica así como comprender la información que ofrecen las empresas son los únicos requisitos básicos para dar un paso más allá de la simple puesta en marcha de los dispositivos, tal cual se

reciben de fábrica. Todo ello potenciado lógicamente por las plataformas *Open Sourc*e, como Arduino, que permiten un desarrollo más personalizado y cooperativo de proyectos como el nuestro. De este modo se contribuye a la divulgación de la cultura *DoItYourself*, muy interesante a la hora de desarrollar nuevas tecnologías, al apoyar una mayor globalidad del conocimiento.

Sin embargo, somos conscientes de que estamos muy lejos de haber desarrollado un dispositivo perfecto, por lo que queremos dejar abiertas varias líneas de mejoras:

- El sistema en cuestión no ha sido testado lo suficientemente como para asegurar que la probabilidad de fallos cumpla los requisitos de las estrictas normas de nuestro sector aeronáutico. Debido a que este paso escapa a los objetivos de nuestras intenciones iniciales, es interesante plantear un método para asegurar a priori que la frecuencia de fallo es fácilmente disminuible en cierta medida. Por todo ello se propone la caracterización del dispositivo con cierta redundancia, de modo que sea robusto ante fallos de los sensores que inevitablemente se producirán. A grandes rasgos consistiría en construir uno o varios dispositivos idénticos, modificar nuestro código de lectura de datos para que se capturen los de todos ellos, y además, aplicar un nuevo algoritmo que, ante fallo de uno de los sensores, consiga obtener una lectura satisfactoria. Como ejemplo podemos comentar que, durante nuestro proceso de desarrollo del equipo ocurrieron ciertos problemas en las conexiones del cableado, devolviendo lógicamente errores en los datos. Esto podría haberse evitado si tuviésemos un conjunto de sensores analizando la misma información, ya que este fallo es poco probable que ocurra en todos.

- El error comentado en el punto anterior nos lleva a una nueva línea de mejora a nivel hardware. En el proceso de diseño del prototipo y construcción del mismo, las conexiones realizadas con ayuda de una tabla *Protoboard* han sido suficientes. Pero no lo sería a la hora de continuar con el desarrollo del equipo, donde un fallo en las conexiones es inaceptable. Por ello es imprescindible desarrollar una integración del hardware de mayor calidad.

- Al estudiar los datos necesarios que debería devolver nuestro sistema se consideró vital que entre ellos se encontrase la posición estimada en la que se encuentra nuestro vehículo, a partir de las aceleraciones que captan nuestros sensores. Por ello resulta necesario aplicar un método de integración de las mismas para calcular dichos desplazamientos en las tres direcciones, ya que con ellas no basta a nivel de navegación. Como primer desarrollo nos conformamos con

demostrar que nuestro sistema es capaz de devolver esta información y la del GPS en serie, pero no es suficiente para aplicaciones reales. Este paso no sería trivial, ya que los errores tanto de lectura debido al ruido, como numéricos producidos durante la doble integración a la que hay que someter a estos datos, son una realidad insalvable, y se irían acumulando con el tiempo. Por todo ello nos veríamos obligados a aplicar determinados filtros para minimizar al máximo estas desviaciones.

- A pesar de que la compañía que vende los sensores ofrece documentos sobre cada uno donde se especifica su precisión por separado, no conocemos en qué medida ha podido afectar a dichos valores la integración. Para aumentar la fiabilidad de nuestro sistema será de vital importancia realizar un estudio de cuan finas son las mediciones de nuestro instrumento integrado. Para ello, por ejemplo, habrían de diseñarse experimentos que describiesen el método de realizar precisos movimientos y comprobar si los datos leídos se corresponden con la realidad.

- Por último, se tendría que dar el siguiente paso en la integración de los dos dispositivos en cuestión. Por ahora se ha obtenido un equipo capaz de obtener las aceleraciones en las tres direcciones (importante a nivel estructural y como paso previo a obtener las velocidades y los desplazamientos), los ángulos que definen el rumbo del vehículo (para conocer en qué dirección y sentido nos movemos), la coordenadas geográficas en que nos encontramos, dos estimaciones de la altitud (necesaria a la hora del despegue y aterrizaje; se contaría con una tercera obtenida a partir de las aceleraciones), entre otros. Sin embargo, estos datos son independientes. En los dispositivos actuales, todos ellos interactúan con el objeto de obtener una navegación más satisfactoria. Por todo ello habría que estudiar la aplicación de un algoritmo que, combinando los datos de medidas inerciales y del GPS, obtenga mejores resultados que los que se pueden obtener por separado. A pesar de la idoneidad que este paso supone, resulta ser el menos factible, debido a que estos algoritmos no son mayoritariamente mostrados por las empresas, por lo que la carencia de información supondría una falta de referencias.

En este momento podemos decir que se espera que el punto en el que nuestro proyecto concluye pueda servir como base de siguientes desarrollos, contribuyendo a que más personas puedan formarse en el vasto y joven mundo de la aviónica de los UAV.

Referencias

1. Introduction to Avionics Systems, R.P.G. Collinson, Second Edition
2. http://en.wikipedia.org/wiki/History_of_unmanned_aerial_vehicles
3. http://webdiis.unizar.es/~neira/docs/ABarrientos-CEDI2007.pdf...Grupo de Robótico y Cibernética, Universidad Politécnica de Madrid
4. http://www.jpaerospace.com/Tandem/tandem.html
5. http://apeironuav.org/?p=205
6. http://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=0CDMQFjAC&url=http%3A%2F%2Fdspace.mit.edu%2Fbitstream%2Fhandle%2F1721.1%2F50382%2F40169626-MIT.pdf%3Fsequence%3D2&ei=e3ieVZXaIsHfU4iwu7AL&usg=AFQjCNHbYMqcPU8i9p1zS9duzizspCQrrA&bvm=bv.96952980,d.d24
7. http://www.rada.com/images/brochures/uav_avionics.pdf
8. http://www.rada.com/capabilities/avionics/mavins.html
9. http://www.uadrones.net/systems/research/acrobat/050418.pdf
10. http://pulseaero.com/products/uav-autopilots.php
11. http://www.uavnavigation.com/products/uav-autopilot-vector7
12. http://www.uavnavigation.com/products/uav-autopilot-proton
13. https://www.openpilot.org
14. http://www.micropilot.com/pdf/brochures/brochure-MP2x28.pdf
15. http://www.micropilot.com/pdf/brochures/brochure-MP2128LRC-HELI.pdf
16. www.micropilot.com/pdf/brochures/brochure-MP21283x.pdf
17. http://www.airelectronics.es/products/?PHPSESSID=e5f3d7011u4rt8t6l6bjn8rio1
18. http://www.adafruit.com/product/1604
19. http://www.adafruit.com/products/746
20. https://learn.adafruit.com/adafruit-ultimate-gps
21. https://upload.wikimedia.org/wikipedia/commons/c/c1/Yaw_Axis_Corrected.svg
22. https://es.wikipedia.org/wiki/Sistema_de_posicionamiento_global

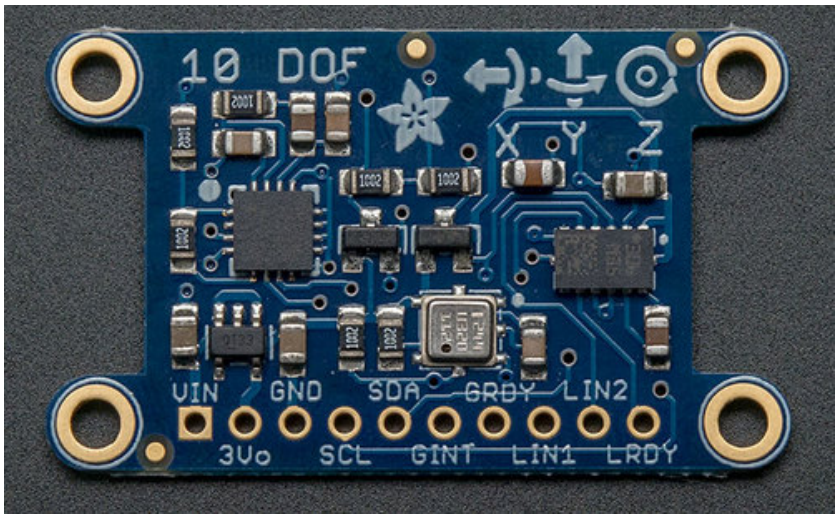
Anexos

- **Anexo 1. Tutorial: Adafruit 10-DOF IMU Breakout**
- **Anexo 2. Data Sheet: GlobalTop-FGPMMOPA6H**

adafruit learning system

Adafruit 10-DOF IMU Breakout

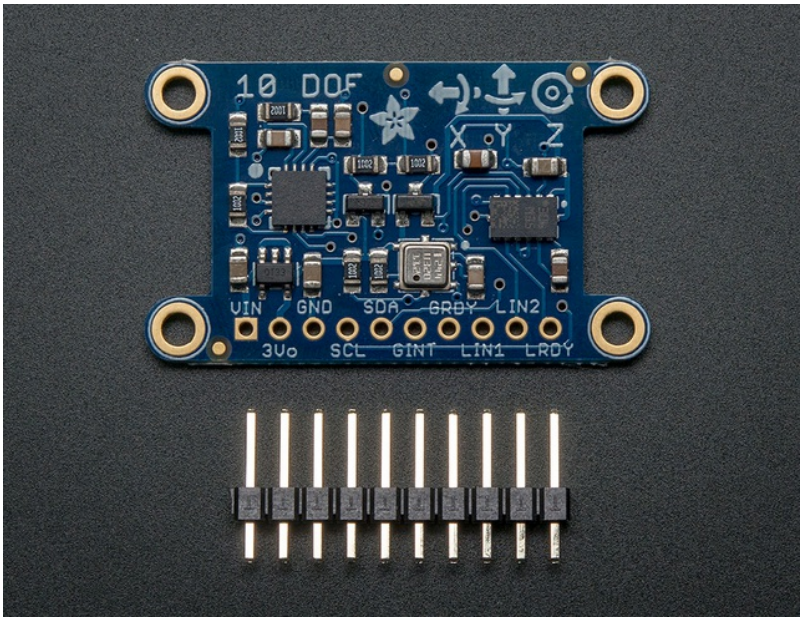
Created by Kevin Townsend



Last updated on 2014-02-07 03:00:14 PM EST

Guide Contents

Introduction



Please note that there is a small cosmetic error on the first revision of these boards (shown above). The straight X arrow for the accelerometer should be pointing in the opposite direction, towards the right. This will be fixed in the net run of these boards.

Adafruit's 10DOF (10 Degrees of Freedom) breakout board allows you to capture ten (err, eleven!) distinct types of motion or orientation related data.

After testing a lot of combinations of sensors, we settled on the following devices that we think offer the best results and the least amount of hassle:

- **LSM303DLHC** - a **3-axis accelerometer** (up to +/-16g) and a **3-axis magnetometer** (up to +/-8.1 gauss) on a single die
- **L3GD20** - a **3-axis gyroscope** (up to +/-2000 dps)
- **BMP180** - A **barometric pressure sensor** (300..1100 hPa) that can be used to calculate altitude, with an additional on-board **temperature sensor**

Related Links

The Adafruit 10DOF board and library reuses the existing Adafruit drivers for the **LSM303DLHC** (accelerometer and magnetometer), the **L3GD20** (gyroscope) and the **BMP180** (pressure/altitude sensor).

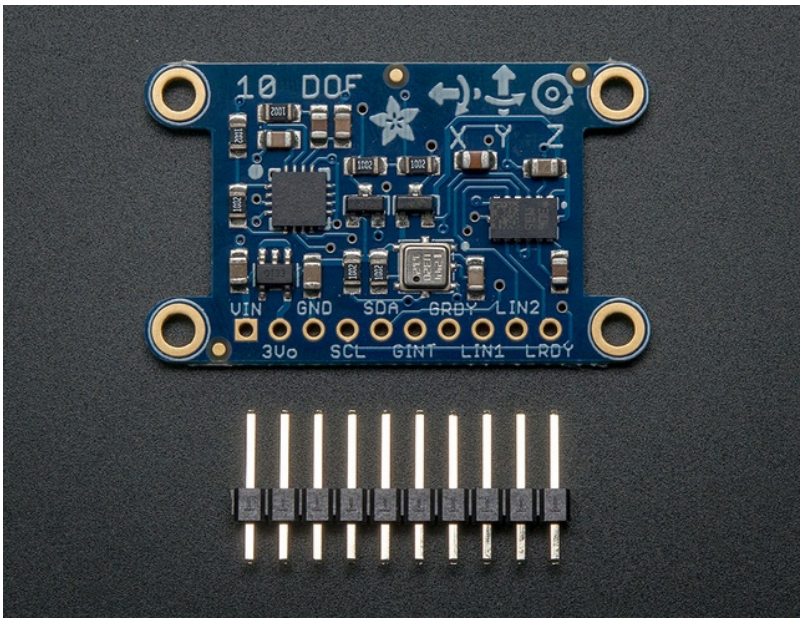
For information about these particular drivers, consult the following learning guides:

- LSM303DLHC Learning Guide (http://adafru.it/cXW)
- L3GD20 Learning Guide (http://adafru.it/cXX)
- BMP180 Learning Guide (http://adafru.it/cXY)

This breakout is basically just a smooshed together version of all three so anything you can do with those libraries/guides will follow here.

Connecting It Up

All of the sensors on the Adafruit 10DOF breakout communicate via a two-pin I2C bus, making it relatively easy to setup with a minimum number of cables:



> To interface with the sensor, you will need to solder in wire or header into the breakout row at the bottom. You cannot 'press fit' or 'twist' wires in, they will not make good contact! Soldering is required

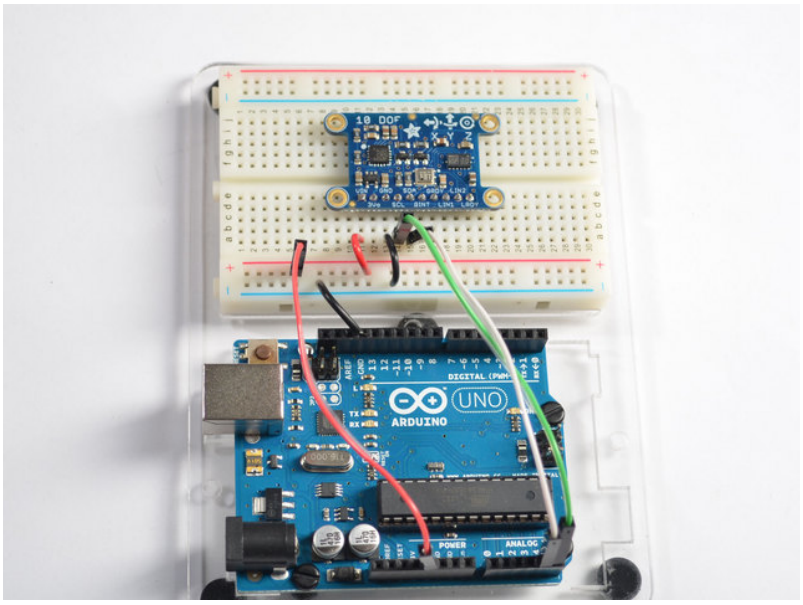
Basic Setup (5V Logic, Arduino Uno, etc.)

We'll be using an Arduino UNO here, but the code will work on a Mega or Leonardo just fine. Most other Arduino compatibles should have no problems either but we only support official Arduinos for code.

- Connect the **SCL** pin on the breakout to the **SCL** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A5**, on a Mega it is also known as **digital 21** and on a Leonardo/Micro, **digital 3**
- Connect the **SDA** pin on the breakout to the **SDA** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A4**, on a Mega it is also known as **digital 20** and on a Leonardo/Micro, **digital 2**
- Connect the **VIN** pin on the breakout to **3.3V or 5V** on your Uno (5V is preferred but if you have a 3V logic Arduino 3V is best)
- Connect the **GND** pin on the breakout to the **GND** pin on your Uno

That's it! With those four wires, you should be able to talk to any of the I2C chips on the board

and run any of the example sketches.



Advanced Setup

While most people probably won't need to use the pins below, we've also broken out a few extra pins for advanced users or for special use cases. If you need to use any of these pins, simply hook them up to a GPIO pin of your choice on the Uno:

- **GINT** - The interrupt pin on the L3GD20 gyroscope
- **GRDY** - The 'ready' pin on the L3GD20 gyroscope
- **LIN1** - Interrupt pin 1 on the LSM303DLHC
- **LIN2** - Interrupt pin 2 on the LSM303DLHC
- **LRDY**- The ready pin on the LSM303DLHC

These pins are all outputs from the 10-DOF breakout and are all 3.3V logic, you can use them with 5V or 3V as 3.3V registers 'high' on 5V systems.

3V3 Setup

If you are using an MCU or board with 3V3 logic (instead of the 5V logic used by the Arduino Uno), you can still power the 10-DOF with the VIN pin *or* you can use the 3Vo pin, which will bypass the on-board 3V3 regulator and level shifting:

- Connect **Vin or 3Vo** on the breakout to the **3.3V** supply on your target MCU
- Connect **GND** on the breakout to **GND** on the target MCU

> Like other breakouts on Adafruit, the 10 DOF Breakout is fully level shifted, and you can safely use it on 3V3 or 5V systems.

Software

LSM303DLHC, L3GD20, BMP180 Drivers

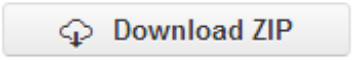
You will need to have all of the following libraries available in your /libraries folder for Arduino to make use of the Adafruit 10DOF breakout:

- Adafruit Unified Sensor Library on Github (http://adafru.it/aZm)
- LSM303DLHC Library on Github (http://adafru.it/aYU)
- L3GD20 Library on Github (http://adafru.it/cZ4)
- BMP180 Library on Github (http://adafru.it/aZq)
- Adafruit 10DOF Library on Github (http://adafru.it/cXZ)

For information on the Adafruit Sensor Library and the Unified Sensor Drivers used for all of these sensors, feel free to have a look at our related learning guide: Using the Adafruit Unified Sensor Driver (http://adafru.it/cZ5)

Downloading Libraries from Github

If you aren't familiar with Github, the easiest way to install the libraries is to click on the links above and find the button that looks like this on the main repository page:



Click this button to download a .zip file containing everything in the repository, and then **place the files in the Arduino Sketch Folder '/libraries' sub-folder.** You should end up with a structure like this:

- *arduinosketches*/libraries/Adafruit_10DOF
- *arduinosketches*/libraries/Adafruit_BMP085
- *arduinosketches*/libraries/Adafruit_L3GD20_U
- *arduinosketches*/libraries/Adafruit_LSM303DLHC
- *arduinosketches*/libraries/Adafruit_Sensor

If you're new to installing libraries, check out our super-awesome detailed tutorial on how to install Arduino libraries (http://adafru.it/aYM)

Adafruit_10DOF Library Helper Functions

The **Adafruit_10DOF.cpp** file (from Adafruit_10DOF (http://adafru.it/cXZ)) includes the following helper functions that are useful when working with typicaly 10DOF projects

bool accelGetOrientation (sensors_event_t *event, sensors_vec_t *orientation)

This function reads the LSM303DLHC accelerometer data (supplied via the 'event' variable), and converts it into equivalent pitch (x) and roll (y) values, populating the supplied 'orientation' variables .pitch and .roll fields accordingly.

Arguments

- **event: The sensors_event_t variable containing the data from the accelerometer**
- **orientation: The sensors_vec_t object that will have its .pitch and .roll fields populated**

Returns

- **true** if the operation was successful,
- **false** if there was an error

Example

> See the 'pitchrollheading' example in the Adafruit_10DOF library for an example of how to use this helper function

```
sensors_event_t accel_event;
sensors_vec_t   orientation;

/* Calculate pitch and roll from the raw accelerometer data */
accel.getEvent(&accel_event);
if (dof.accelGetOrientation(&accel_event, &orientation))
{
  /* 'orientation' should have valid .roll and .pitch fields */
  Serial.print(F("Roll: "));
  Serial.print(orientation.roll);
  Serial.print(F("; "));
  Serial.print(F("Pitch: "));
  Serial.print(orientation.pitch);
  Serial.print(F("; "));
}
```

bool magGetOrientation (sensors_axis_t axis, sensors_event_t *event, sensors_vec_t *mag_orientation)

This function populates the .heading field in mag_orientation with the correct angular data (0-359°). Heading increases when rotating clockwise around the specified axis.

Arguments

- **axis:** The given axis (SENSOR_AXIS_X, SENSOR_AXIS_Y, or SENSOR_AXIS_Z)
- **event:** The raw magnetometer sensor data to use when calculating out heading
- **orientation:** The sensors_vec_t object where we will assign an 'orientation.heading' value

Returns

- **true** if the operation was successful,
- **false** if there was an error

Example

> See the 'pitchrollheading' example in the Adafruit_10DOF library for an example of how to use this helper function

```
sensors_event_t mag_event;
sensors_vec_t   orientation;

/* Calculate the heading using the magnetometer */
mag.getEvent(&mag_event);
if (dof.magGetOrientation(SENSOR_AXIS_Z, &mag_event, &orientation))
{
  /* 'orientation' should have valid .heading data now */
  Serial.print(F("Heading: "));
  Serial.print(orientation.heading);
  Serial.print(F("; "));
}
```

bool magTiltCompensation (sensors_axis_t axis, sensors_event_t *mag_event, sensors_event_t *accel_event)

This function uses the accelerometer data provided in accel_event to compensate the magnetic sensor measurements in mag_event to compensate for situations where the sensor is tilted (the pitch and roll angles are not equal to 0°).

Arguments

- **axis:** The given axis (SENSOR_AXIS_X, SENSOR_AXIS_Y, or SENSOR_AXIS_Z) that is parallel to the gravity of the Earth
- **mag_event:** The raw magnetometer data to adjust for tilt
- **accel_event:** The accelerometer event data to use to determine the tilt when compensating the mag_event values

Returns

- **true** if the operation was successful,
- **false** if there was an error

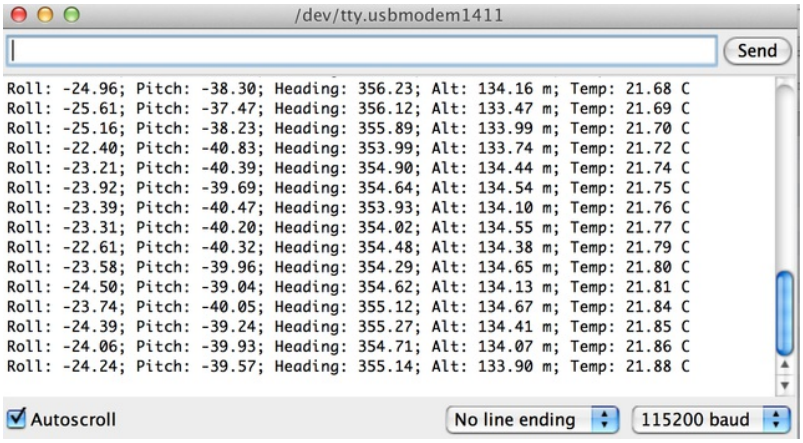
Example

```
sensors_event_t accel_event;
sensors_event_t mag_event;

...

// Get a sensor event from the accelerometer and magnetometer
```

```
accel.getEvent(&accel_event);
mag.getEvent(&mag_event);

if (magTiltCompensation(SENSOR_AXIS_Z, &mag_event, &accel_event))
{
  // Do something with the compensated data in mag_event!
}
else
{
  // Oops ... something went wrong (probably bad data)
}
```

Example Sketch

If you run the **pitchrollheading** sketch in the examples folder, you can see a practical example using these helper functions above, which should result in output similar to the image below:

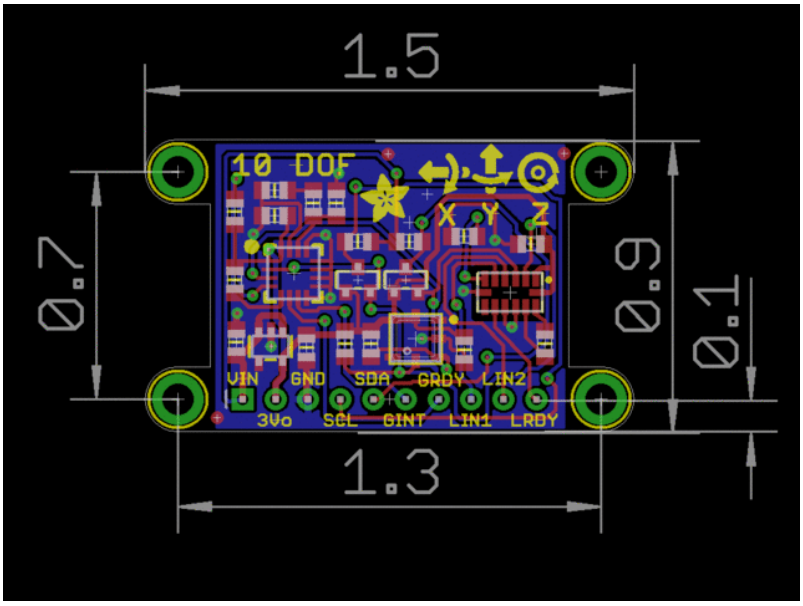


Design Files

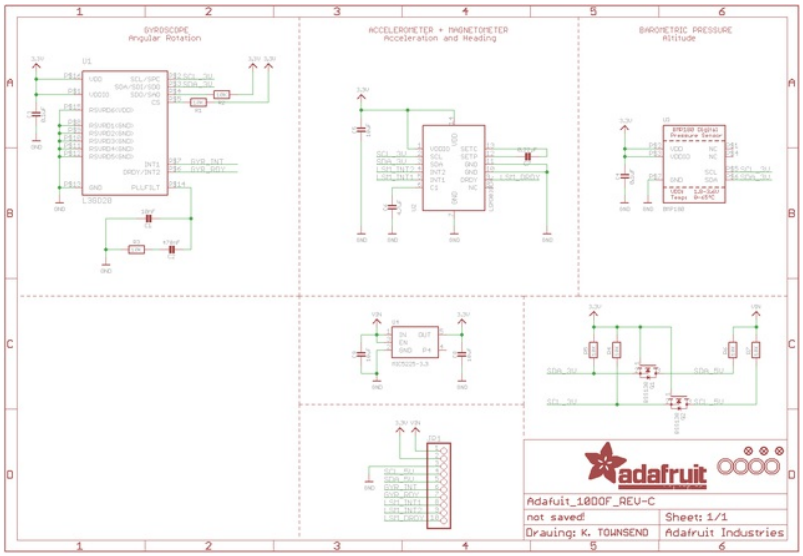
Datasheets

- LSM303 Datasheet (http://adafru.it/d8s)
- L3GD20 Datasheet (http://adafru.it/d8t)
- BMP180 Datasheet (http://adafru.it/d8u)

Dimensions (Inches)



Schematic


GYROSCOPE
Angular Rotation
ACCELEROMETER + MAGNETOMETER
Acceleration and Heading
BAROMETRIC PRESSURE
Altitude
Adafuit_1000F_REV-C
not saved!
Sheet: 1/1
Drawing: K. TOWNSEND
Adafruit Industries

Data Sheet

GlobalTop Technology Inc.

FGPMMOPA6H
GPS Standalone Module
Data Sheet

Revision: V0A

The FGPMMOPA6H is a 4th generation stand-alone GPS module with lightning fast TTFF, ultra high sensitivity (-165dBm), and low power consumption in a small form factor (16*16*4.7mm)

No.16 Nan-ke 9th Rd, Science-Based Industrial Park, Tainan, 741, Taiwan, R.O.C.
Tel: +886-6-5051268 / **Fax**: +886-6-5053381 / **Email**: sales@gtop-tech.com / **Web**: www.gtop-tech.com

Version History

Title:	**GlobalTop FGPMMOPA6H Datasheet**
Subtitle:	**GPS Module**
Doc Type:	**Datasheet**

Revision	Date	Author	Description
V0A	2012-1-31	Delano	Preliminary

Table of Contents

1. Functional Description

1.1 Overview

The FGPMMOPA6H utilizes the MediaTek new generation GPS Chipset MT3339 that achieves the industry's highest level of sensitivity (-165dBm) and instant Time-to-First Fix (TTFF) with lowest power consumption for precise GPS signal processing to give the ultra-precise positioning under low receptive, high velocity conditions.

The module a revision of POT (Patch On Top) GPS Module with an extra embedded function for external antenna I/O and comes with **automatic antenna switching function** and short circuit protection, also features a antenna system called "Antenna Advisor" that helps with the detections and notifications of different antenna statuses, including active antenna connection, antenna open circuit and antenna shortage.

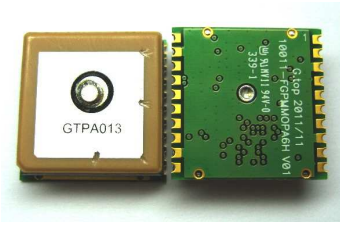
Up to 12 multi-tone active interference canceller (ISSCC2011 award), customer can have more flexibility in system design. Supports up to 210 PRN channels with 66 search channels and 22 simultaneous tracking channels, Module supports various location and navigation applications, including autonomous GPS,QZSS, SBAS(note) ranging (WAAS, EGNO, GAGAN, MSAS), AGPS.

FGPMMOPA6H is excellent low power consumption characteristic (acquisition 82mW, tracking 66mW), power sensitive devices, especially portable applications, need not worry about operating time anymore and user can get more fun.

Note: SBAS can only be enabled when update rate is less than or equal to 5Hz.

Application:

- ✓ Handheld Device
- ✓ Tablet PC/PLB/MID
- ✓ M2M application
- ✓ Asset management
- ✓ Surveillance



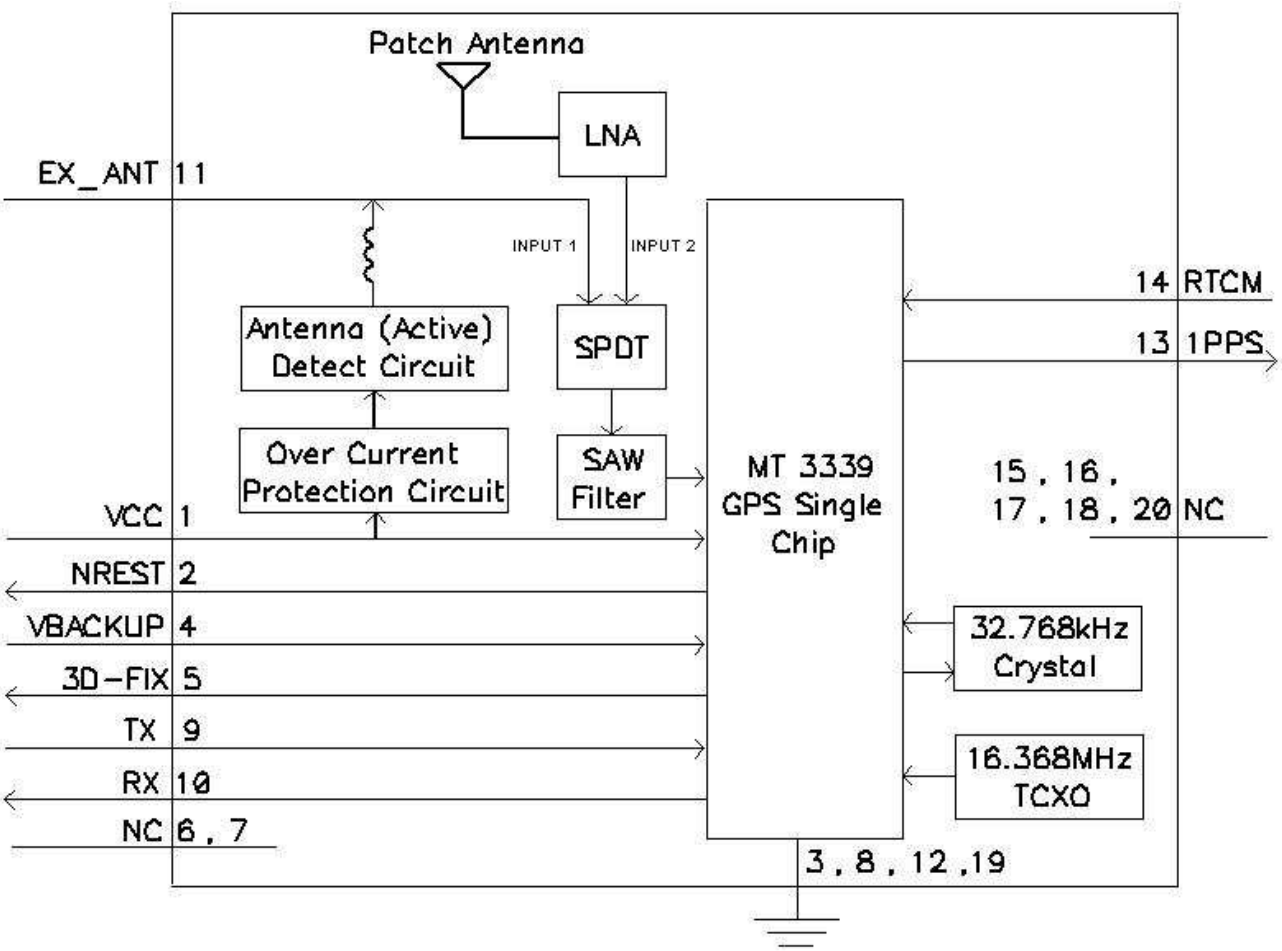
1.2Highlights and Features

- Built-in 15X15X2.5mm ceramic patch antenna on the top of module
- Ultra-High Sensitivity: -165dBm (w/o patch antenna), up to 45dB C/N of SVs in open sky reception.
- High Update Rate: up to 10Hz(note1)
- 12 multi-tone active interference canceller(note2) [ISSCC 2011 Award -Section 26.5] (http://isscc.org/doc/2011/isscc2011.advanceprogrambooklet_abstracts.pdf)
- High accuracy 1-PPS timing support for Timing Applications (10ns jitter)
- AGPS Support for Fast TTFF (EPO™ Enable 7 days/14 days)
- EASY™(note2): Self-Generated Orbit Prediction for instant positioning fix
- AlwaysLocate™(note2) Intelligent Algorithm (Advance Power Periodic Mode) for power saving
- Logger function Embedded(note2)
- Automatic antenna switching function
- Antenna Advisor function
- Gtop Firmware Customization Services
- Consumption current(@3.3V):
  - Acquisition: 25mA Typical
  - Tracking: 20mA Typical
- E911, RoHS, REACH compliant

note 1: SBAS can only be enabled when update rate is less than or equal to 5Hz.

note2: Some features need special firmware or command programmed by customer, please refer to G-top “GPS command List”

1.3 System Block Diagram



1.4 Multi-tone active interference canceller

Because different application (Wi-Fi, GSM/GPRS, 3G/4G, Bluetooth) are integrated into navigation system, the harmonic of RF signal will influence the GPS reception, The multi-tone active interference canceller (abbr: MTAIC) can reject external RF interference which come from other active components on the main board, to improve the capacity of GPS reception without any needed HW change in the design. PA6H can cancel up to 12 independent channels interference continuous wave (CW)

1.5 1PPS

A pulse per second (1 PPS) is an electrical signal that very precisely indicates the start of a second. Depending on the source, properly operating PPS signals have typical accuracy ranging 10ns.

1 PPS signals are used for precise timekeeping and time measurement. One increasingly common use is in computer timekeeping, including the NTP protocol. A common use for the PPS signal is to connect it to a PC using a low-latency, low-jitter wire connection and allow a program to synchronize to it:

PA6H supply the high accurate 1PPS timing to synchronize to GPS time after 3D-Fix.
A power-on output 1pps is also available for customization firmware settings.

1.6 AGPS Support for Fast TTFF (EPO™)

The AGPS (EPO™) supply the predicated Extended Prediction Orbit data to speed TTFF ,users can download the EPO data to GPS engine from the FTP server by internet or wireless network ,the GPS engine will use the EPO data to assist position calculation when the navigation information of satellites are not enough or weak signal zone . About the detail, please link Gtop website .

1.7 EASY™

The EASY™ is embedded assist system for quick positioning, the GPS engine will calculate and predict automatically the single emperies (Max. up to 3 days)when power on ,and save the predict information into the memory , GPS engine will use these information for positioning if no enough information from satellites, so the function will be helpful for positioning and TTFF improvement under indoor or urban condition.

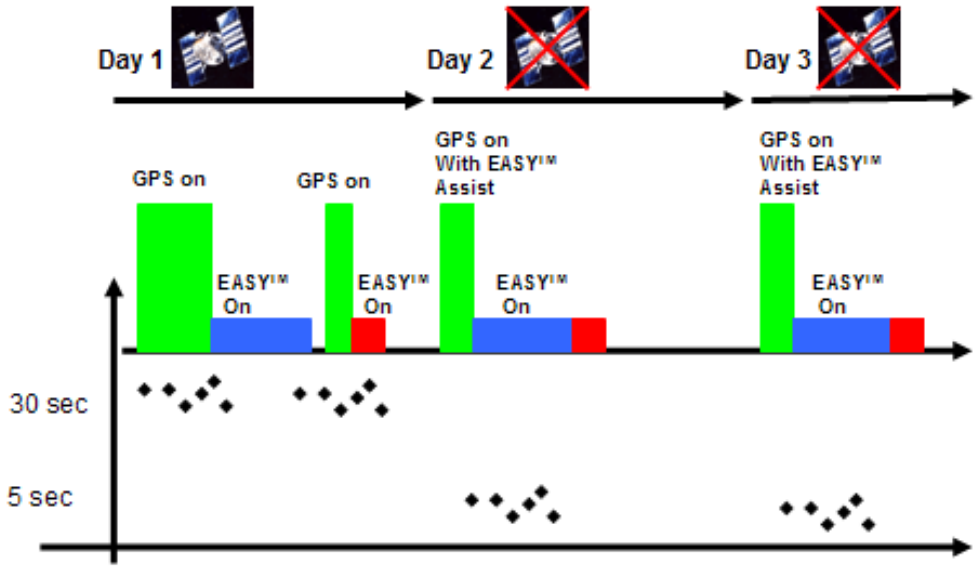


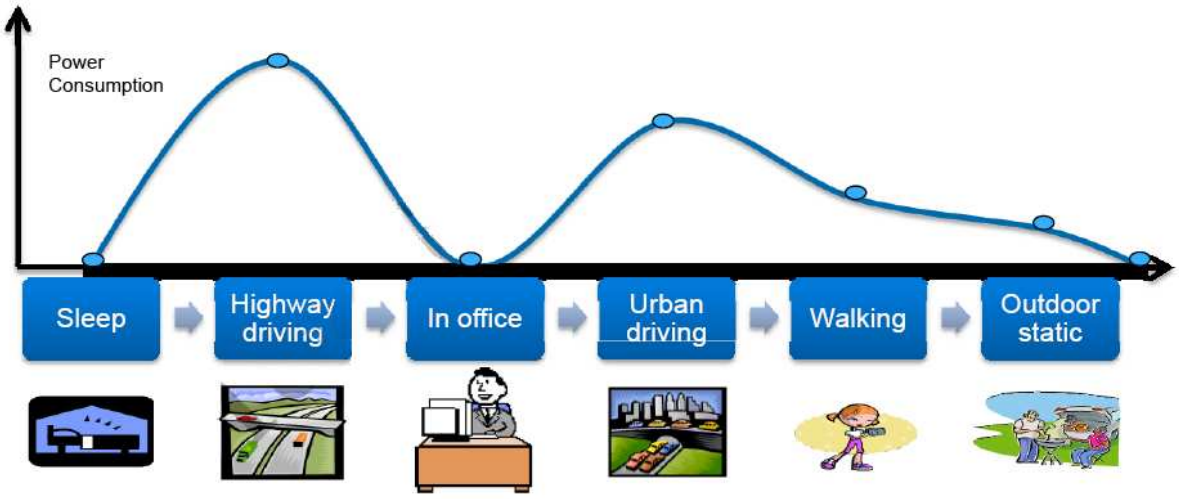
Figure 1.12-1 EASY System operation

Please refer to the Fig 1.12-1, When GPS device great the satellite information from GPS satellites, the GPS engine automatically pre-calculate the predict orbit information for 3 days

The GPS device still can quickly do the positioning with EASY™ function under weak GPS signal.

1.8 AlwaysLocate™ (Advance Power Periodic Mode)

Embedded need to be executed full y all the time , the algorithm can be set by different necessary to decide the operation level of GPS function , reduce power consumption , it will suffer positing accuracy to get the target of power saving and extend the usage time of product . (The positioning accuracy of reporting location < 50m (CEP)



1.9 Embedded Logger function

The Embedded Logger function don't need host CPU (MCU) and external flash to handle the operation , GPS Engine will use internal flash (embedded in GPS chipset) to log the GPS data (Data format : UTC, Latitude , longitude, Valid ,Checksum), the max log days can up to 2 days under AlwaysLocate™ condition .[Note]

Note: Data size per log was shrunk from 24 bytes to 15 bytes.

2.0 Antenna Advisor

"Antenna Advisor" is a brand new antenna system available exclusively for PA6H. It is designed to detect and notify antenna status using software (through proprietary protocol on **Chapter 3.2**).

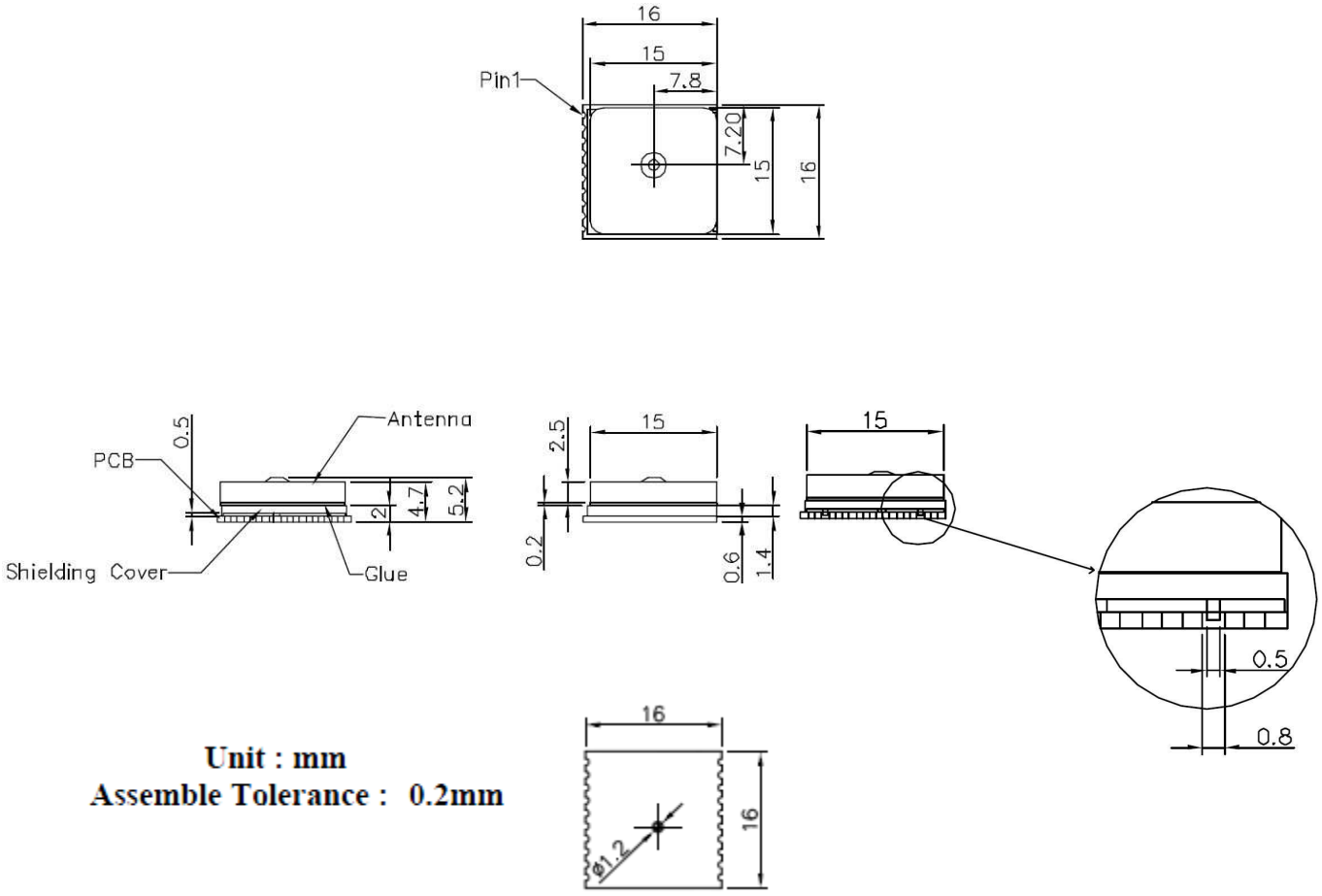
Antenna Advisor can detect and notify the following:

- Active Antenna Shorted
- Using Internal Antenna
- Using Active Antenna

2. Specifications

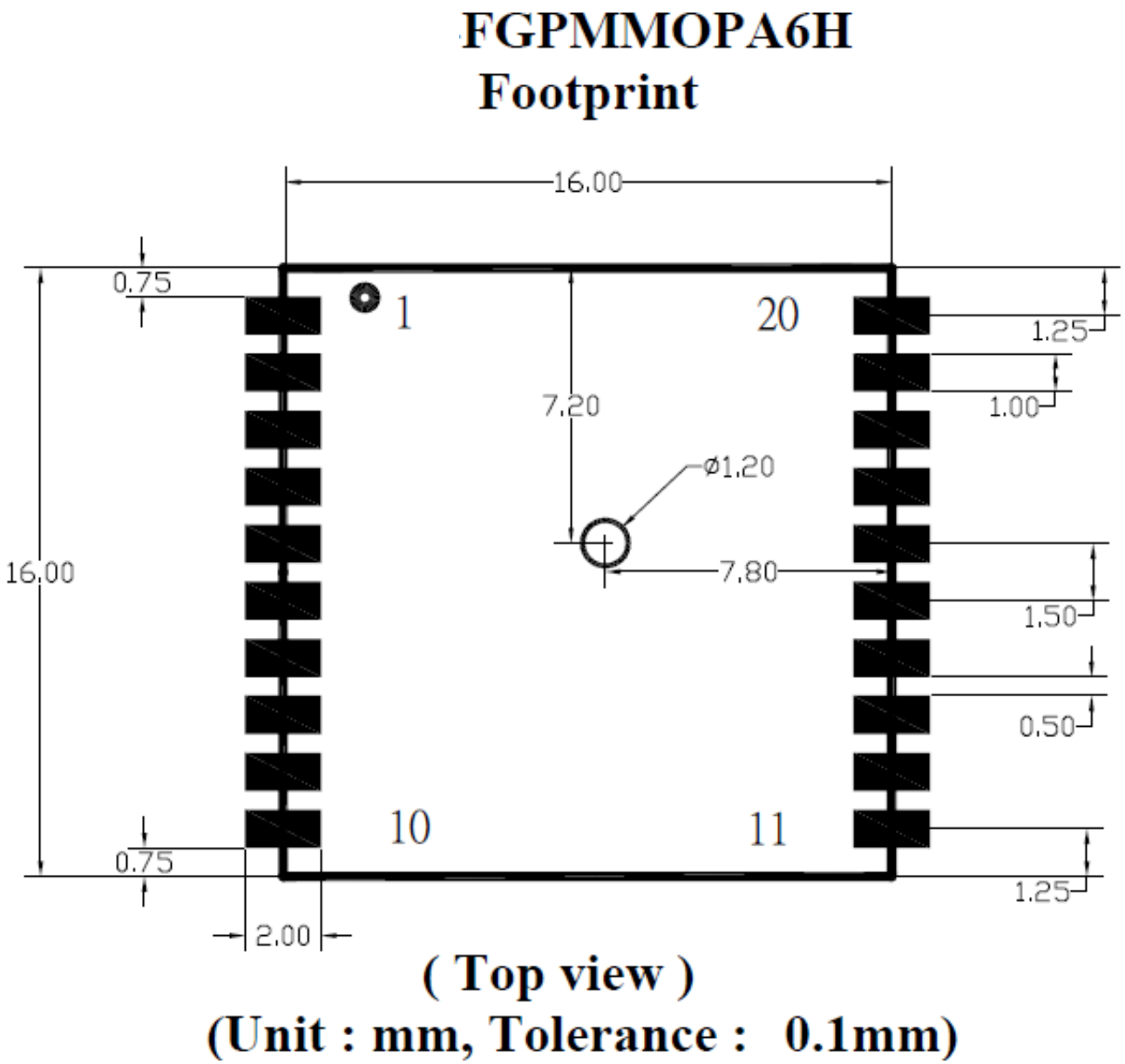
2.1 Mechanical Dimension

Dimension: (Unit: mm, Tolerance: +/- 0.2mm)

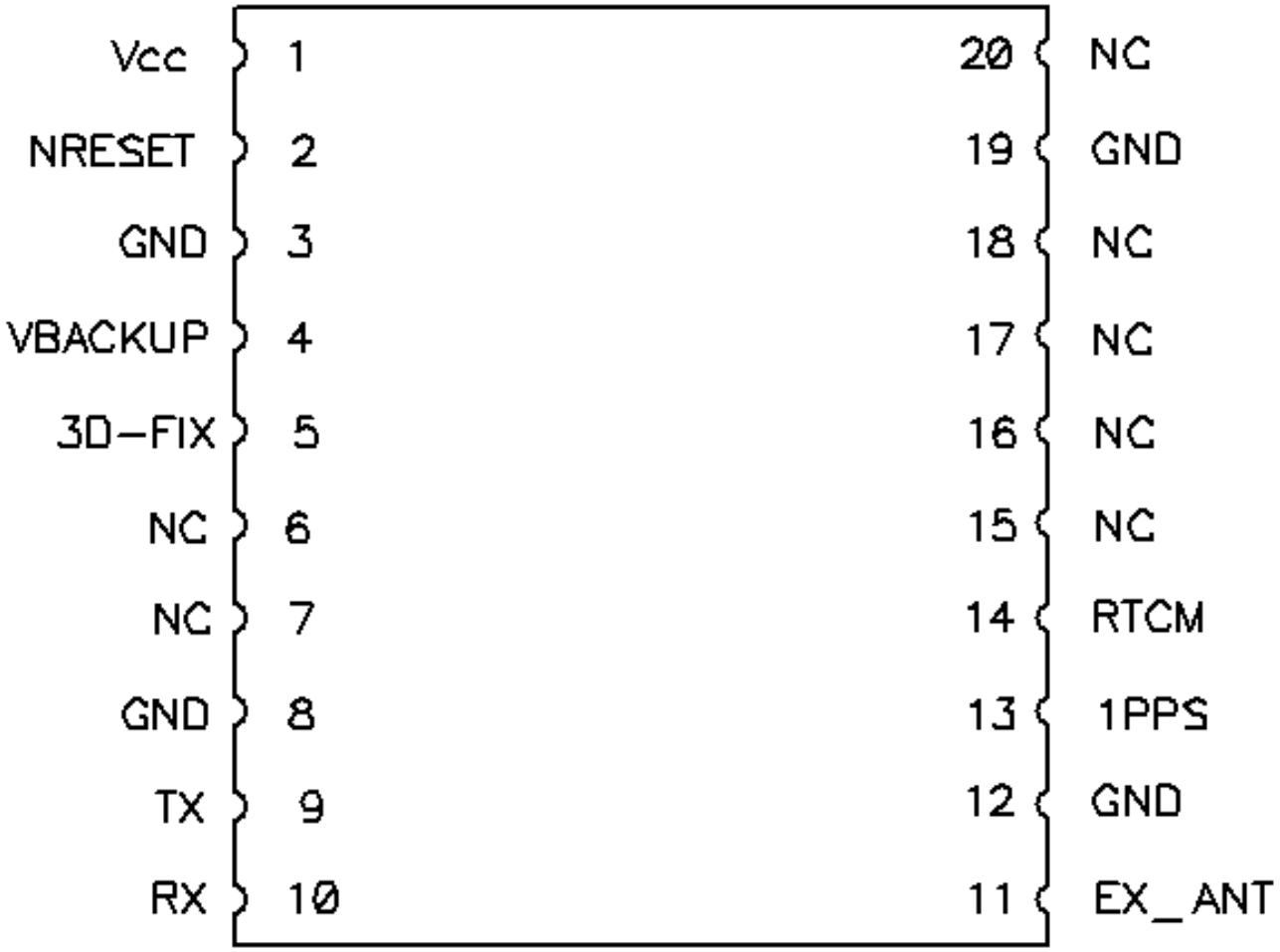


2.2 Recommended PCB pad Layout

(Unit: mm, Tolerance: 0.1mm)



2.3 Pin Configuration



(Top view)

2.4 Pin Assignment

Pin	Name	I/O	Description & Note
1	VCC	PI	Main DC Power Input
2	NRESET	I	Reset Input, Low Active
3	GND	P	Ground
4	VBACKUP	PI	Backup Power Input for RTC & Navigation Data Retention
5	3D-FIX	O	3D-Fix Indicator
6	NC	--	Not Connect
7	NC	--	Not Connect
8	GND	P	Ground
9	TX	O	Serial Data Output for NMEA Output (UART TTL)
10	RX	I	Serial Data Input for Firmware Update (UART TTL)
11	EX_ANT	I PO	External active antenna RF input. DC power from VCC and provide for external active antenna.
12	GND	P	Ground
13	1PPS	O	1PPS Time Mark Output 2.8V CMOS Level
14	RTCM	I	Serial Data Input for DGPS RTCM Data Streaming
15	NC	--	Not Connect
16	NC	--	Not Connect
17	NC	--	Not Connect
18	NC	--	Not Connect
19	GND	P	Ground
20	NC	--	Not Connect

2.5 Description of I/O Pin

VCC (Pin1)

The main DC power supply of the module, the voltage should be kept between from 3.0V to 4.3V.
The Vcc ripple must be controlled under 50mV$_{pp}$ (Typical: 3.3V)

NRESET (Pin2)

With a low level, it causes the module to reset. If not used, keep floating.

GND (Pin3, Pin8, Pin12, Pin19)

Ground

VBACKUP (Pin4)

This connects to the backup power of the GPS module. Power source (such as battery) connected to this pin will help the GPS chipset in keeping its internal RTC running when the main power source is turned off. The voltage should be kept between 2.0V~4.3V, Typical 3.0V.

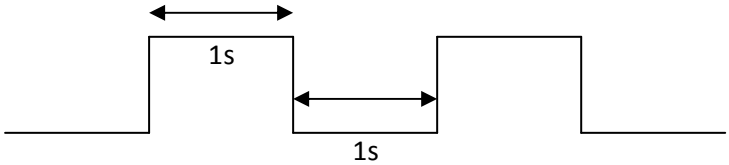
IF VBACKUP power was not reserved, the GPS module will perform a lengthy cold start every time it is powered-on because previous satellite information is not retained and needs to be re-transmitted.

If not used, keep open.

3D-FIX (Pin5)

The 3D-FIX is assigned as a fix flag output. The timing behavior of this pin can be configured by custom firmware for different applications (Example: waking up host MCU). If not used, keep floating.

- Before 2D Fix
  The pin should continuously output one-second high-level with one-second low-level signal.



- After 2D or 3D Fix
  The pin should continuously output low-level signal.

  Low _____

NC (Pin6, Pin7, Pin15, Pin16, Pin17, Pin18, Pin20)

Not connect.

TX (Pin9)

This is the UART transmitter of the module. It outputs the GPS information for application.

RX (Pin10)

This is the UART receiver of the module. It is used to receive software commands and firmware update.

EX_ANT (Pin11)

DC power from VCC and provide for external active antenna (Recommendation: 3.3V).

When a 4mA or higher current is detected, the detect circuit will acknowledge the external antenna as being present and uses external antenna for reception.
In the event of short circuit occurring at external antenna, the module will limit the drawn current to a safe level.
The 3.0V for the GPS external antenna is limited to 25mA.
The 3.3V for the GPS external antenna is limited to 28mA.
The 3.6V for the GPS external antenna is limited to 31mA.

1PPS (Pin13)

This pin provides one pulse-per-second output from the module and synchronizes to GPS time.

Keep floating if not used.

RTCM (Pin14)

This pin receive DGPS data of RTCM protocol (TTL level), if not used keep floating'

RTCM is not enabled by default, please consult GlobalTop support to enable this feature.

2.6 Specification List

	Description
GPS Solution	MTK MT3339
Frequency	L1, 1575.42MHz
Sensitivity[1]	Acquisition: -148dBm, cold start Reacquisition: -163dBm, Hot start Tracking: -165dBm
Channel	66 channels
TTFF	Hot start: 1 second typical Warm start: 33 seconds typical Cold start: 35 seconds typical (No. of SVs>4, C/N>40dB, PDop<1.5)
Position Accuracy	Without aid:3.0m (50% CEP) DGPS(SBAS(WAAS,EGNOS,MSAS)):2.5m (50% CEP)
Velocity Accuracy	Without aid : 0.1m/s DGPS(SBAS(WAAS,EGNOS,MSAS,GAGAN)):0.05m/s
Timing Accuracy (1PPS Output)	10 ns(Typical)
Altitude	Maximum 18,000m (60,000 feet)
Velocity	Maximum 515m/s (1000 knots)
Acceleration	Maximum 4G
Update Rate	1Hz (default), maximum 10Hz
Baud Rate	9600 bps (default)
DGPS	SBAS(defult) [WAAS, EGNOS, MSAS,GAGAN]
QZSS	Support(Ranging)
AGPS	Support
Power Supply	VCC：3.0V to 4.3V；VBACKUP：2.0V to 4.3V
Current Consumption	25mA acquisition, 20mA tracking
Working Temperature	-40 °C to +85 °C
Dimension	16 x 16x 4.7mm, SMD
Weight	**4g**

2.7 Absolute Maximum Ratings

The voltage applied for VCC should not exceed 4.3VDC.

	Symbol	Min.	Typ.	Max.	Unit
Power Supply Voltage	VCC	3.0	3.3	4.3	V
Backup battery Voltage	VBACKUP	2.0	3.0	4.3	V

2.8 Operating Conditions

	Condition	Min.	Typ.	Max.	Unit
Operation supply Ripple Voltage	—	—	—	50	**mVpp**
RX0 TTL H Level	VCC=3.0~4.3V	2.0	—	VCC	**V**
RX0 TTL L Level	VCC=3.0~4.3V	0	—	0.8	**V**
TX0 TTL H Level	VCC=3.0~4.3V	2.4	—	2.8	**V**
TX0 TTL L Level	VCC=3.0~4.3V	0	—	0.4	**V**
Current Consumption @ 3.3V, 1Hz Update Rate	Acquisition	—	25	—	**mA**
	Tracking	—	20	—	**mA**
Backup Power Consumption@ 3V	25°C	—	7	—	**uA**

2.9 GPS External Antenna Specification (Recommended)

It is important that the antenna gets a clear view of the sky and is positioned on a surface level to the horizon for best results. The following specification has to meet for the use reference design.

Characteristic	Specification
Polarization	Right-hand circular polarized
Frequency Received	1.57542GHz +/- 1.023MHz
Power Supply	3V to 3.6V
DC Current	4mA ~ 20mA at 3.3V
Total Gain	>+ 15dBi (Two-stage LNA)
Output VSWR	< 2.5
Impedance	50ohm
Noise Figure	< 1.5dB

3. Protocols

3.1 NMEA Output Sentences

Table-1 lists each of the NMEA output sentences specifically developed and defined by MTK for use within MTK products

Table-1: NMEA Output Sentence	
Option	**Description**
GGA	Time, position and fix type data.
GSA	GPS receiver operating mode, active satellites used in the position solution and DOP values.
GSV	The number of GPS satellites in view satellite ID numbers, elevation, azimuth, and SNR values.
RMC	Time, date, position, course and speed data. Recommended Minimum Navigation Information.
VTG	Course and speed information relative to the ground.

GGA—Global Positioning System Fixed Data. Time, Position and fix related data

Table-2 contains the values for the following example：

$GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.95,39.9,M,17.8,M,,*65

Table-2: GGA Data Format

Name	Example	Units	Description
Message ID	$GPGGA		GGA protocol header
UTC Time	064951.000		hhmmss.sss
Latitude	2307.1256		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12016.4438		dddmm.mmmm
E/W Indicator	E		E=east or W=west
Position Fix Indicator	1		See **Table-3**
Satellites Used	8		Range 0 to 14
HDOP	0.95		Horizontal Dilution of Precision
MSL Altitude	39.9	meters	Antenna Altitude above/below mean-sea-level
Units	M	meters	Units of antenna altitude
Geoidal Separation	17.8	meters	
Units	M	meters	Units of geoids separation
Age of Diff. Corr.		second	Null fields when DGPS is not used
Checksum	*65		
<CR> <LF>			End of message termination

Table-3: Position Fix Indicator

Value	Description
0	Fix not available
1	GPS fix
2	Differential GPS fix

GSA—GNSS DOP and Active Satellites

Table-4 contains the values for the following example：

$GPGSA,A,3,29,21,26,15,18,09,06,10,,,,,2.32,0.95,2.11*00

Table-4: GSA Data Format			
Name	**Example**	**Units**	**Description**
Message ID	$GPGSA		GSA protocol header
Mode 1	A		See **Table-5**
Mode 2	3		See **Table-6**
Satellite Used	29		SV on Channel 1
Satellite Used	21		SV on Channel 2
....	....	....	....
Satellite Used			SV on Channel 12
PDOP	2.32		Position Dilution of Precision
HDOP	0.95		Horizontal Dilution of Precision
VDOP	2.11		Vertical Dilution of Precision
Checksum	*00		
<CR> <LF>			End of message termination

Table-5: Mode 1	
Value	**Description**
M	Manual—forced to operate in 2D or 3D mode
A	2D Automatic—allowed to automatically switch 2D/3D

Table-6: Mode 2	
Value	**Description**
1	Fix not available
2	2D (＜4 SVs used)
3	3D (≧4 SVs used)

GSV—GNSS Satellites in View

Table-7 contains the values for the following example：

$GPGSV,3,1,09,29,36,029,42,21,46,314,43,26,44,020,43,15,21,321,39*7D

$GPGSV,3,2,09,18,26,314,40,09,57,170,44,06,20,229,37,10,26,084,37*77
$GPGSV,3,3,09,07,,,26*73

Table-7: GSV Data Format

Name	Example	Units	Description
Message ID	$GPGSV		GSV protocol header
Number of Messages	3		Range 1 to 3 *(Depending on the number of satellites tracked, multiple messages of GSV data may be required.)*
Message Number1	1		Range 1 to 3
Satellites in View	09		
Satellite ID	29		Channel 1 (Range 1 to 32)
Elevation	36	degrees	Channel 1 (Maximum 90)
Azimuth	029	degrees	Channel 1 (True, Range 0 to 359)
SNR (C/No)	42	dBHz	Range 0 to 99, (null when not tracking)
....	….	….	....
Satellite ID	15		Channel 4 (Range 1 to 32)
Elevation	21	degrees	Channel 4 (Maximum 90)
Azimuth	321	degrees	Channel 4 (True, Range 0 to 359)
SNR (C/No)	39	dBHz	Range 0 to 99, (null when not tracking)
Checksum	*7D		
<CR> <LF>			End of message termination

RMC—Recommended Minimum Navigation Information

Table-8 contains the values for the following example：

$GPRMC,064951.000,A,2307.1256,N,12016.4438,E,0.03,165.48,260406,3.05,W,A*2C

Table-8: RMC Data Format

Name	Example	Units	Description
Message ID	$GPRMC		RMC protocol header
UTC Time	064951.000		hhmmss.sss
Status	A		A=data valid or V=data not valid
Latitude	2307.1256		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12016.4438		dddmm.mmmm
E/W Indicator	E		E=east or W=west
Speed over Ground	0.03	knots	
Course over Ground	165.48	degrees	True
Date	260406		ddmmyy
Magnetic Variation	3.05, W	degrees	E=east or W=west (Need GlobalTop Customization Service)
Mode	A		A= Autonomous mode D= Differential mode E= Estimated mode
Checksum	*2C		
<CR> <LF>			End of message termination

VTG—Course and speed information relative to the ground

Table-9 contains the values for the following example**:**

$GPVTG,165.48,T,,M,0.03,N,0.06,K,A*37

Table-9: VTG Data Format			
Name	**Example**	**Units**	**Description**
Message ID	$GPVTG		VTG protocol header
Course	165.48	degrees	Measured heading
Reference	T		True
Course		degrees	Measured heading
Reference	M		Magnetic (Need GlobalTop Customization Service)
Speed	0.03	knots	Measured horizontal speed
Units	N		Knots
Speed	0.06	km/hr	Measured horizontal speed
Units	K		Kilometers per hour
Mode	A		A= Autonomous mode D= Differential mode E= Estimated mode
Checksum	*06		
<CR> <LF>			End of message termination

3.2 Antenna Status Protocol (Antenna Advisor)

The function is for external active antenna only.

PGTOP—Status of antenna

Table-12 contains the values for the following example**:**

$PGTOP,11,3 *6F

Table-12: PGACK Data Format			
Name	**Example**	**Units**	**Description**
Message ID	$PGTOP		Protocol header
Command ID	11		Function Type
Reference	3		Value of antenna status

Example:

$PGTOP,11,value*checksum

Value: 1=>Active Antenna Shorted

2=>Using Internal Antenna

3=>Using Active Antenna

3.3 MTK NMEA Command Protocols

Packet Type:

103 PMTK_CMD_COLD_START

Packet Meaning:

Cold Start：Don’t use Time, Position, Almanacs and Ephemeris data at re-start.

Example:

$PMTK103*30<CR><LF>

3.4 Firmware Customization Services

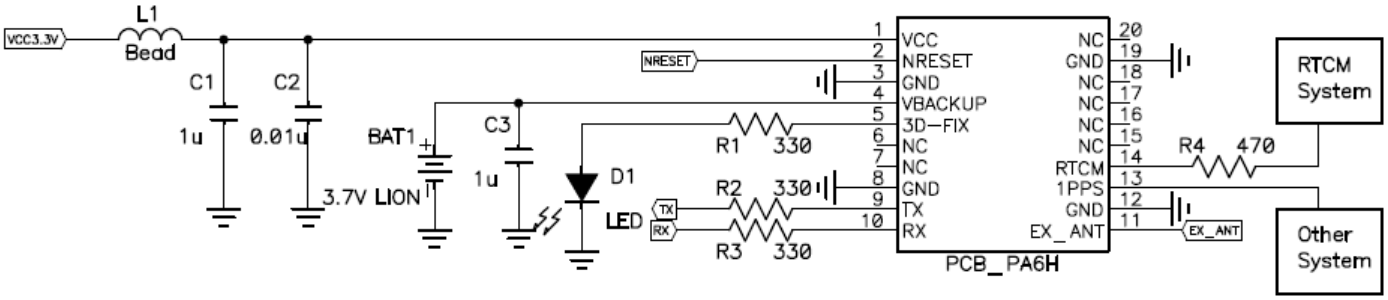
GlobalTop also offers flexible, value-adding GPS firmware customization services that maximizes the over system efficiencies and power consumptions. Latest functions like Binary Mode, 1-Sentence Output, Geo-fencing and Last Position Retention, please see our website at www.gtop-tech.com under Products / GPS Modules / Software Services for more details.

Note: Not all firmware customization services listed below are supported by module. Please contact GlobalTop Sales or Technical Support for more details.

4. Reference Design

This chapter introduces the reference schematic design for the best performance. Additional tips and cautions on design are well documented on Application Note, which is available upon request.

4.1 Reference Design Circuit



Note:

1. Ferrite bead L1 is added for power noise reduction.
2. C1 and C2 bypass should be put near the module.
3. Damping resistors R2,R3,R4 could be modified based on system application for EMI.
4. If you need more support and information on antenna implementation, please directly contact us at sales@gtop-tech.com for further services.

5. Packing and Handling

GPS modules, like any other SMD devices, are sensitive to moisture, electrostatic discharge, and temperature. By following the standards outlined in this document for GlobalTop GPS module storage and handling, it is possible to reduce the chances of them being damaged during production set-up. This document will go through the basics on how GlobalTop packages its modules to ensure they arrive at their destination without any damages and deterioration to performance quality, as well as some cautionary notes before going through the surface mount process.

⚠ **Please read the sections II to V carefully to avoid damages permanent damages due to moisture intake**

⚠ **GPS receiver modules contain highly sensitive electronic circuits and are electronic sensitive devices and improper handling without ESD protections may lead to permanent damages to the modules. Please read section VI for more details.**

5.1 Moisture Sensitivity

GlobalTop GPS modules are moisture sensitive, and must be pre-baked before going through the solder reflow process. It is important to know that:

GlobalTop GPS modules must complete solder reflow process in 72 hours after pre-baking.

This maximum time is otherwise known as “Floor Life”

If the waiting time has exceeded 72 hours, it is possible for the module to suffer damages during the solder reflow process such as cracks and delamination of the SMD pads due to excess moisture pressure.

5.2 Packing

GlobalTop GPS modules are packed in such a way to ensure the product arrives to SMD factory floor without any damages.

GPS modules are placed individually on to the packaging tray. The trays will then be stacked and packaged together.

Included are:

1. Two packs of desiccant for moisture absorption
2. One moisture level color coded card for relative humidity percentage.

Each package is then placed inside an antistatic bag (or PE bag) that prevents the modules from being damaged by electrostatic discharge.

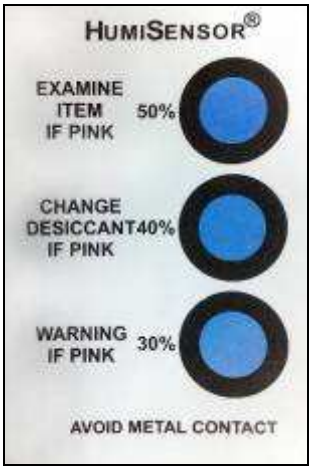
Figure 1: One pack of GPS modules

Each bag is then carefully placed inside two levels of cardboard carton boxes for maximum protection.

Figure 2: Box protection

The moisture color coded card provides an insight to the relative humidity percentage (RH). When the GPS modules are taken out, it should be around or lower than 30% RH level.

Outside each electrostatic bag is a caution label for moisture sensitive device.



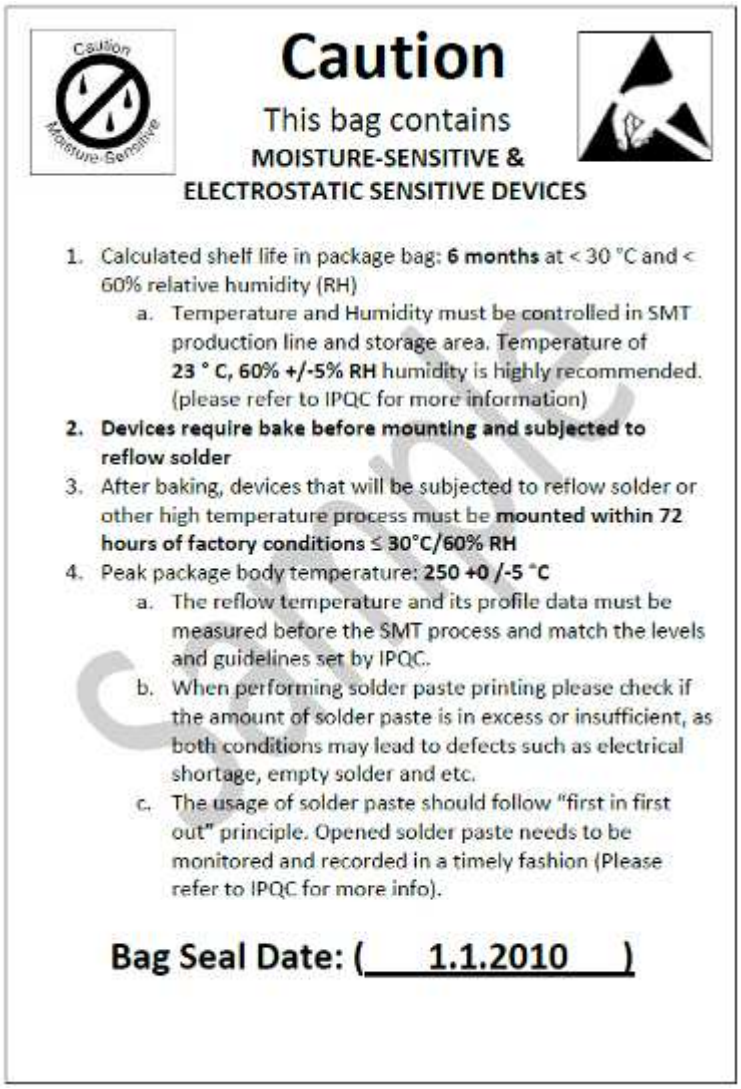


Figure 3: Example of moisture color coded card and caution label

5.3 Storage and Floor Life Guideline

Since GlobalTop modules must undergo solder-reflow process in 72 hours after it has gone through pre-baking procedure, therefore if it is not used by then, it is recommended to store the GPS modules in dry places such as dry cabinet.

The approximate shelf life for GlobalTop GPS modules packages is 6 months from the bag seal date, when store in a non-condensing storage environment (<30°C/60% RH)

⚠ **It is important to note that it is a required process for GlobalTop GPS modules to undergo pre-baking procedures, regardless of the storage condition.**

5.4 Drying

Because the vapor pressures of moisture inside the GPS modules increase greatly when it is exposed to high temperature of solder reflow, in order to prevent internal delaminating, cracking of the devices, or the "popcorn" phenomenon, it is a **necessary requirement** for GlobalTop GPS module to undergo pre-baking procedure before any high temperature or solder reflow process.

The recommendation baking time for GlobalTop GPS module is as follows:

- ✓ **60°C for 8 to 12 hours**

Once baked, the module's floor life will be "reset", and has additional 72 hours in normal factory condition to undergo solder reflow process.

⚠ **Please limit the number of times the GPS modules undergoes baking processes as repeated baking process has an effect of reducing the wetting effectiveness of the SMD pad contacts. This applies to all SMT devices.**

⚠ **Oxidation Risk: Baking SMD packages may cause oxidation and/or intermetallic growth of the terminations, which if excessive can result in solderability problems during board assembly. The temperature and time for baking SMD packages are therefore limited by solderability considerations. The cumulative bake time at a temperature greater than 90°C and up to 125°C shall not exceed 96 hours. Bake temperatures higher than 125°C are now allowed.**

5.5 ESD Handling

Please carefully follow the following precautions to prevent severe damage to GPS modules.

GlobalTop GPS modules are sensitive to electrostatic discharges, and thus are Electrostatic Sensitive Devices (ESD). Careful handling of the GPS modules and in particular to its patch antenna (if included) and RF_IN pin, must follow the standard ESD safety practices:

- ✓ Unless there is a galvanic coupling between the local GND and the PCB GND, then the first point of contact when handling the PCB shall always be between the local GND and PCB GND.
- ✓ Before working with RF_IN pin, please make sure the GND is connected
- ✓ When working with RF_IN pin, do not contact any charges capacitors or materials that can easily develop or store charges such as patch antenna, coax cable, soldering iron.
- ✓ Please do not touch the mounted patch antenna to prevent electrostatic discharge from the RF input
- ✓ When soldering RF_IN pin, please make sure to use an ESD safe soldering iron (tip).

6. Reflow Soldering Temperature Profile

The following reflow temperature profile was evaluated by GlobalTop and has been proven to be reliable qualitatively. Please contact us beforehand if you plan to solder this component using a deviated temperature profile as it may cause significant damage to our module and your device.

All the information in this sheet can only be used only for Pb-free manufacturing process.

6.1 SMT Reflow Soldering Temperature Profile
(Reference Only)

Average ramp-up rate (25 ~ 150°C): 3°C/sec. max.

Average ramp-up rate (270°C to peak): 3°C/sec. max.

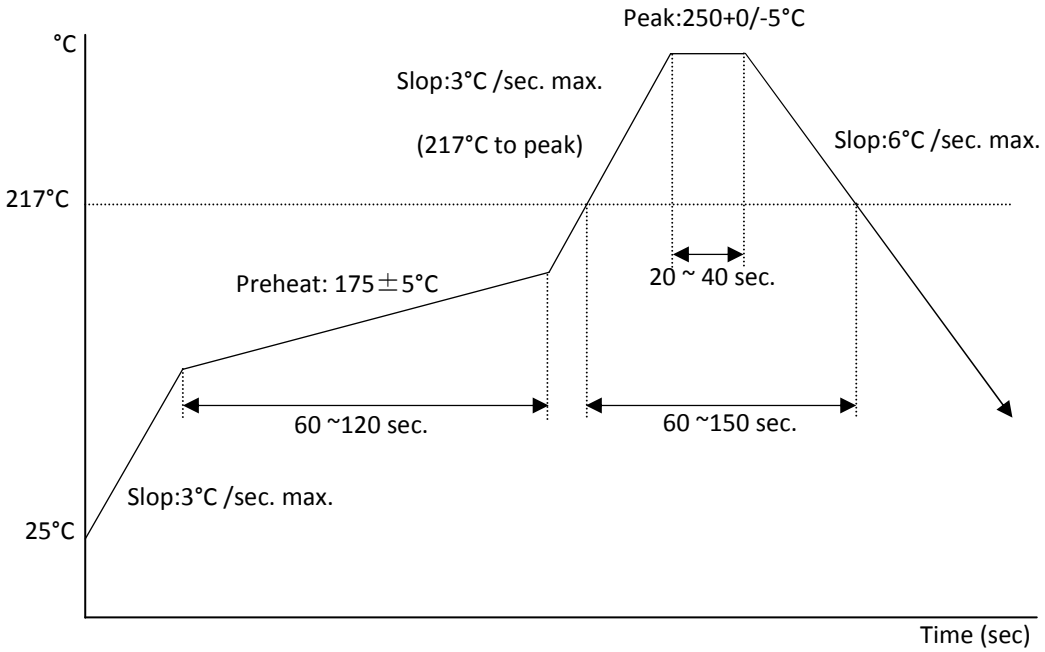
Preheat: 175 ± 25°C, 60 ~ 120 seconds

Temperature maintained above 217°C: 60~150 seconds

Peak temperature: 250 +0/-5°C, 20~40 seconds

Ramp-down rate: 6°C/sec. max.

Time 25°C to peak temperature: 8 minutes max.



	Details	Suggestions	Notes
1	Before proceeding with the reflow-soldering process, the GPS module must be pre-baked.	Pre-bake Time: 6 Hours @ 60°±5°C or 4 Hours @ 70°±5°C	**The maximum tolerated temperature for the tray is 100°C.** **After the pre-baking process, please make sure the temperature is sufficiently cooled down to 35°C or below in order to prevent any tray deformation.**
2	Because PCBA (along with the patch antenna) is highly endothermic during the reflow-soldering process, extra care must be paid to the GPS module's solder joint to see if there are any signs of cold weld(ing) or false welding.	The parameters of the reflow temperature must be set accordingly to module's reflow-soldering temperature profile.	**Double check to see if the surrounding components around the GPS module are displaying symptoms of cold weld(ing) or false welding.**
3	Special attentions are needed for PCBA board during reflow-soldering to see if there are any symptoms of bending or deformation to the PCBA board, possibility due to the weight of the module. If so, this will cause concerns at the latter half of the production process.	A loading carrier fixture must be used with PCBA if the reflow soldering process is using rail conveyors for the production.	**If there is any bending or deformation to the PCBA board, this might causes the PCBA to collide into one another during the unloading process.**
4	Before the PCBA is going through the reflow-soldering process, the production operators must check by eyesight to see if there are positional offset to the module, because it will be difficult to readjust after the module has gone through reflow-soldering process.	The operators must check by eyesight and readjust the position before reflow-soldering process.	**If the operator is planning to readjust the module position, please do not touch the patch antenna while the module is hot in order to prevent rotational offset between the patch antenna and module**

Note: References to patch antenna is referred to GPS modules with integrated Patch-on-top antennas (PA/Gms Module Series), and may not be applicable to all GPS modules.

	Details	Suggestions	Notes
5	Before handling the PCBA, they must be cooled to 35°C or below after they have gone through the reflow-soldering process, in order to prevent positional shift that might occur when the module is still hot.	1. Can use electric fans behind the Reflow machine to cool them down. 2. Cooling the PCBA can prevent the module from shifting due to fluid effect.	**It is very easy to cause positional offset to the module and its patch antenna when handling the PCBA under high temperature.**
6	1. When separating the PCBA panel into individual pieces using the V-Cut process, special attentions are needed to ensure there are sufficient gap between patch antennas so the patch antennas are not in contact with one another. 2. If V-Cut process is not available and the pieces must be separated manually, please make sure the operators are not using excess force which may cause rotational offset to the patch antennas.	1. The blade and the patch antenna must have a distance gap greater than 0.6mm. 2. Do not use patch antenna as the leverage point when separating the panels by hand.	**1. Test must be performed first to determine if V-Cut process is going to be used. There must be enough space to ensure the blade and patch antenna do not touch one another.** **2. An uneven amount of manual force applied to the separation will likely to cause positional shift in patch antenna and module.**
7	When separating panel into individual pieces during latter half of the production process, special attentions are needed to ensure the patch antennas do not come in contact with one another in order to prevent chipped corners or positional shifts.	Use tray to separate individual pieces.	**It is possible to chip corner and/or cause a shift in position if patch antennas come in contact with each other.**

Note: References to patch antenna is referred to GPS modules with integrated Patch-on-top antennas (PA/Gms Module Series), and may not be applicable to all GPS modules.

Other Cautionary Notes on Reflow-Soldering Process:

1. Module must be pre-baked **before** going through SMT solder reflow process.
2. The usage of solder paste should follow “first in first out” principle. Opened solder paste needs to be monitored and recorded in a timely fashion (can refer to IPQC for related documentation and examples).
3. Temperature and humidity must be controlled in SMT production line and storage area. Temperature of 23°C, 60±5% RH humidity is recommended. (please refer to IPQC for related documentation and examples)
4. When performing solder paste printing, please notice if the amount of solder paste is in excess or insufficient, as both conditions may lead to defects such as electrical shortage, empty solder and etc.
5. Make sure the vacuum mouthpiece is able to bear the weight of the GPS module to prevent positional shift during the loading process.
6. Before the PCBA is going through the reflow-soldering process, the operators should check by eyesight to see if there are positional offset to the module.
7. The reflow temperature and its profile data must be measured before the SMT process and match the levels and guidelines set by IPQC.
8. If SMT protection line is running a double-sided process for PCBA, please process GPS module during the second pass only to avoid repeated reflow exposures of the GPS module. Please contact GlobalTop beforehand if you must process GPS module during the 1st pass of double-side process.

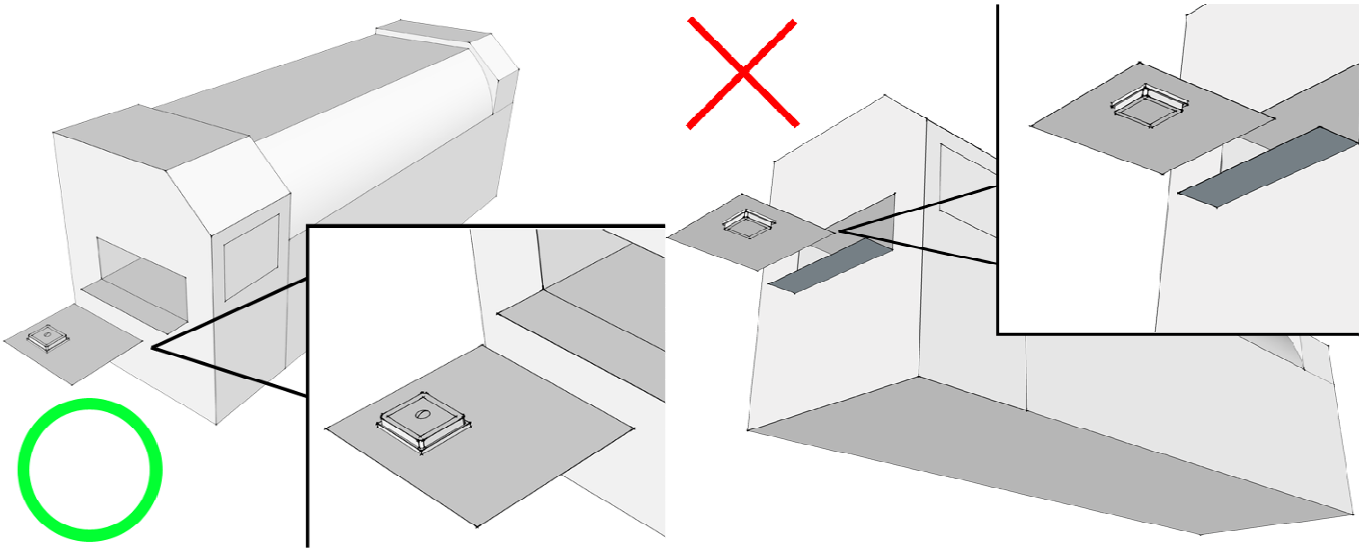


Figure 6.2: Place GPS module right-side up when running reflow-solder process, do not invert.

9. Module must be pre-baked **before** going through SMT solder reflow process.
10. The usage of solder paste should follow “first in first out” principle. Opened solder paste needs to be monitored and recorded in a timely fashion (can refer to IPQC for related documentation and examples).
11. Temperature and humidity must be controlled in SMT production line and storage area. Temperature of 23°C, 60±5% RH humidity is recommended. (please refer to IPQC for related documentation and examples)
12. When performing solder paste printing, please notice if the amount of solder paste is in excess or insufficient, as both conditions may lead to defects such as electrical shortage, empty solder and etc.
13. The reflow temperature and its profile data must be measured before the SMT process and match the levels and guidelines set by IPQC.

6.2 Manual Soldering

Soldering iron:

Bit Temperature: Under 380°C Time: Under 3 sec.

Notes:

1. Please do not directly touch the soldering pads on the surface of the PCB board, in order to prevent further oxidation
2. The solder paste must be defrosted to room temperature before use so it can return to its optimal working temperature. The time required for this procedure is unique and dependent on the properties of the solder paste used.
3. The steel plate must be properly assessed before and after use, so its measurement stays strictly within the specification set by SOP.
4. Please watch out for the spacing between soldering joint, as excess solder may cause electrical shortage
5. Please exercise with caution and do not use extensive amount of flux due to possible siphon effects on neighboring components, which may lead to electrical shortage.
6. Please do not use the heat gun for long periods of time when removing the shielding or inner components of the GPS module, as it is very likely to cause a shift to the inner components and will leads to electrical shortage.

7. Contact Information

GlobalTop Technology Inc.

Address: No.16 Nan-ke 9rd Road Science-based Industrial Park, Tainan 741, Taiwan

Tel: +886-6-5051268

Fax: +886-6-5053381

Website: www.gtop-tech.com

Email: sales@gtop-tech.com