



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Final de Grado
Grado en Ingeniería Aeroespacial

DESARROLLO DE MIDDLEWARES PARA PILOTOS AUTOMÁTICOS DE BAJO COSTE

MEMORIA

Curso académico 2017-2018

Alumno: ANDRÉS MASIP, Miguel

Tutor: GARCÍA-NIETO RODRÍGUEZ, Sergio

Agradecimientos

A mi familia por el apoyo constante tanto a lo largo de estos 4 años como de las 21 primaveras que tengo. En especial, a mis abuelos, por haber sido referentes y modelos de vida.

A los amigos que han estado ahí toda la vida y a los que han llegado más tarde, pero que han llegado para quedarse.

A toda la gente relacionada con la *terreta*, a los *K14* y a los *tetes* por todos los momentos y fiestas juntos.

A todas las personas que he conocido durante mi etapa universitaria y que han hecho de la misma, un periodo único e irrepetible. Al equipo de futsal bicampeón, al *Report Team*, a los *Tibus*, a Sara, a Miriam,... por haber constituido el núcleo de esta última etapa de mi vida y haberme aguantado todo este tiempo.

Finalmente, quiero dedicarle unas líneas a mi tutor, Sergio, por haberme apoyado y soportado a lo largo de estos meses de proyecto.

Resumen

El trabajo a realizar se basará en la adaptación de uno de los principales pilotos automáticos disponibles en el mercado a un nuevo hardware de control, la plataforma BeagleBone Blue. En ambos casos, son proyectos de carácter abierto, radicando ahí el interés de desarrollar y colaborar en estas iniciativas de futuro. Por otra parte, la BeagleBone Blue reúne las principales características necesarias para ser una apuesta de futuro en el campo de UAV, debido a su integración de potencia computacional, multitud de puertos y conexiones, así como sensores integrados. Esta es la principal novedad frente a otras versiones anteriores o otras opciones de mercado: una unidad inercial de 9DOF y un barómetro, permitirá el *tracking* y la navegación inercial de la plataforma.

Abstract

This project is focused on allowing the implementation of one of the most important autopilot over a new hardware proposal, the BeagleBone Blue. Both are open-source projects and that is the main purpose of this work, the fact of collaborate and develop looking-forward initiatives. Furthermore, the BeagleBone Blue has the main specifications for being a good UAV platform: computational power, many connections, ports and integrated sensors. This is the main difference between it and previous models or other market alternatives: a 9DOF IMU and a barometer, which will allow inertial navigation or tracking capabilities.

Resum

Aquest projecte es fonamenta en la adaptació de un important autopilot a una nova proposta de hardware disponible al mercat, la BeagleBone Blue. Ambdós casos, constitueixen projectes de codi obert, permetint la col·laboració i el desenvolupament d'aquestes iniciatives de futur. Per altra part, la BeagleBone Blue està capacitada per a ser un referent en el món dels UAV, juntant una gran potència computacional, connexions o ports, així com sensors integrats. Aquesta es la novetat fonamental sobre versions anteriors o altres alternatives del mercat: un baròmetre i una IMU de 9DOF, que permetran la navegació inercial i possibilitat de *tracking*.

Índice general

1	Introducción	7
1.1	Motivación	7
1.2	Planteamiento inicial	8
1.3	Estado del arte	8
2	Alcance del proyecto y objetivos	13
2.1	Alcance	13
2.2	Objetivos	13
3	Descripción del sistema hardware	15
3.1	Procesador	19
3.2	Conexiones e interacciones	19
3.3	Sensores	20
3.4	Configuración y utilización	21
4	Sistemas Operativos: Conceptos básicos	22
4.1	Hardware	23
4.2	Software	24
4.3	Firmware	26
4.4	Stack o aplicación	26
4.5	GNU/Linux	27
4.6	Compilador	29
4.7	Protocolos de comunicación	29
5	Descripción del sistema software	32
5.1	¿Qué es?	32
5.2	DroneCode	32
5.3	Miembros	35
5.4	BSD	35
5.5	Hardware compatible	37
5.6	Aeronaves controlables	38
5.7	Modos de vuelo	39
5.8	Desarrolladores	42
5.9	Arquitectura	43
5.10	Flight stack	45
5.11	Posibilidades	46
5.12	QGroundControl	47

5.13 MAVlink	48
6 Solución adoptada	49
6.1 Estructura genérica del código	49
6.2 Intervenciones adoptadas	51
7 Resultados y test	67
7.1 Compilado	67
7.2 Exportar PX4	70
7.3 Resultados	71
7.4 QGroundControl y calibración	72
8 Conclusiones	76
Bibliografía	77

Índice de figuras

3.1	Detalle de la BeagleBone Blue destacando sus principales componentes [25].	15
3.2	Detalle resaltado de la conexión I2C desde el SIPB de OSD3358 para la BeagleBone Blue [25].	20
3.3	Diagrama en detalle de las conexiones del barómetro BMP280 en la BeagleBone Blue [25].	20
3.4	Diagrama en detalle de las conexiones de la IMU MPU9250 en la BeagleBone Blue [25].	21
4.1	Estructura de capas básica de un sistema informático.	22
4.2	Comparación de los principales protocolos de comunicación.	29
5.1	Detalle de la arquitectura DroneCode y las principales componentes [3].	34
5.2	Cuadro de decisión para los modos de vuelo de PX4.	41
5.3	Repositorio PX4 sobre la plataforma GitHub.	42
5.4	Diagrama de arquitectura PX4 [3].	45
5.5	Control para multi-cópteros con nomenclatura PX4 [3].	46
5.6	Control para aeronaves de ala fija con nomenclatura PX4 [3].	46
5.7	Pantalla principal de previsualización de QGroundControl.	47
5.8	Estructura de envío de datos para el protocolo MAVLink [8]	48
6.1	Detalle de los principales directorios y las funciones de cada archivo.	50
7.1	Compilación de PX4 realizada sin errores.	69
7.2	Detalle sobre la subida y exportación de archivos mediante el IDE Cloud9.	70
7.3	Resultado de ejecución del PX4 sobre la BeagleBone Blue.	71
7.4	Pantallas de calibración del <i>compass</i>	72
7.5	Pantalla de calibración del <i>giróscopo</i>	73
7.6	Pantalla de calibración del <i>acelerómetro</i>	73
7.7	Resultado del comando <i>sensors status</i>	74
7.8	Resultados del comando <i>listener</i> sobre los principales sensores.	74

Introducción

En el presente documento, “*Desarrollo de Middlewares para pilotos automáticos de bajo coste*” se introducirán las modificaciones necesarias para ser capaz de ejecutar el piloto automático PX4 sobre la BeagleBone Blue. Por otra parte, se revisarán los principales aspectos de este novedoso hardware, del autopiloto, del emergente sector de los UAVS (*Unmanned Aerial Vehicles*), así como algunos conceptos básicos y necesarios sobre fundamentos de computadores.

1.1. Motivación

Hoy en día existen numerosos proyectos relacionados con UAVs tanto comerciales como propuestas académicas o de investigación. La importancia de profundizar en el funcionamiento de estos dispositivos es vital en un sector aeronáutico cada vez más complejo, saturado y globalizado. Fundamentar y entender los conceptos relativos a este tipo de aeronaves permitirá un conocimiento más profundo de este campo emergente y contribuirá al desarrollo del mismo.

Por otra parte, como se comentará más adelante el código utilizado será un piloto automático muy funcional y versátil, denominado PX4. Para el desarrollo propuesto será vital entender el funcionamiento del mismo y con las interfaces generadas se contribuirá a elevar el número de plataformas disponibles para este piloto. Por tanto, el trabajo se postula como un intento de profundizar en uno de los pilotos más importantes del mercado, para poder desarrollar y ampliar las utilidades del mismo. El desarrollo de *Middlewares* también constituye uno de los retos más interesantes que se pueden plantear ya que necesita un profundo conocimiento sobre sistemas operativos, programación y fundamentos computacionales. Será necesario remarcar que el desarrollo de estas interfaces permite la popularización y fomenta el concepto de multiplataforma, es decir, la capacidad para operar sobre multitud de dispositivos.

1.2. Planteamiento inicial

En cuanto a la organización del trabajo, en primer lugar se dispondrá una intensa labor de recopilación, búsqueda y procesamiento de información. El desarrollo íntegro de los elementos que integran la HAL (*Hardware Abstraction Layer*) o el *Middleware*¹, quedará completamente fuera del alcance de este proyecto.

El código empleado es una iniciativa fundamentada en los principios del software libre, que plantea el problema de la diversificación y en algunos casos, falta de documentación clara. Este problema, bastante común en este tipo de iniciativas, dificulta sobre todo los inicios del proyecto, aunque como se expondrá, la licencia de este código permite controlar el desarrollo del mismo y documentar la mayoría de la información contenida.

En segundo lugar, lo novedoso del hardware propuesto² infunde una complejidad extra. La falta de documentación y de proyectos o libros consolidados sobre esta plataforma es el principal problema que se presenta. Sin embargo, las similitudes con otras plataformas o versiones anteriores, así como el foro y la comunidad BeagleBone [13] permiten mitigar este problema, exigiendo un esfuerzo extra de revisión y fomentando el fenómeno conocido como *troubleshooting*³.

Finalmente, la última dificultad se relaciona con las arquitecturas compatibles con el piloto automático estudiado. Existen plataformas con arquitectura Linux, aunque el código no está implementado sobre ninguna versión de BeagleBone previa a la disponible. A lo largo de este proyecto se expondrán algunas arquitecturas similares que podrán servir como ayuda en los pasos iniciales, así como pauta para guiar y dividir el problema propuesto haciéndolo más abarcable.

1.3. Estado del arte

El mercado actual se ha visto fuertemente influido por un crecimiento exponencial del sector de los UAVs en los últimos años. Dicha consolidación se puede fundamentar en una serie de factores a nivel tecnológico, social y económico que pueden explicar dicho crecimiento y popularización de las aeronaves no tripuladas. Por otra parte, introducir el concepto de piloto automático y la evolución de aviones teledirigidos al concepto actual de UAV será clave en esta etapa.

¹Ambos conceptos se tratarán en profundidad más adelante

²Se fundamenta en una arquitectura Linux y será similar a la combinación BeagleBone Black-BeagleBone Mini, pero su fecha de lanzamiento se sitúa en 2016.

³Término referido a la necesidad de navegar y buscar ayuda en foros para configuraciones o cambios en arquitecturas muy permisivas con el usuario, como por ejemplo Linux.

Autopilotos

Se deberá introducir el concepto de piloto automático como un conjunto de elementos hardware-software destinados a generar un vuelo seguro y controlado, así como aportar una serie de funciones complementarias que dependerán del sistema en cuestión.

Además del concepto de vehículo autónomo controlado, mediante la incorporación de estos pilotos se pueden implementar otras funciones como la programación de tareas repetitivas, rutas preconfiguradas o modos de control que simplifiquen el pilotaje de la misma. Es decir, generalmente estos pilotos automáticos, en función de sus prestaciones, buscan dar valor añadido a la aeronave permitiendo cada vez más funcionalidades. Aquí radicará la principal diferencia que existe entre el concepto de avión tele dirigido y el vehículo autónomo no tripulado o dron.

Aplicaciones

En este apartado se expondrán las principales aplicaciones actuales y estudiadas para este tipo de aeronaves no tripuladas, exponiendo así los principales nichos de mercado en los que se fundamenta su crecimiento actual y futuro:

Civiles

- Tareas de reconocimiento en zonas aisladas o de carácter peligroso.
- Colaboración en tareas de salvamento.
- Vigilancia forestal.
- Ayuda en zonas catastróficas.
- Internet en zonas aisladas.
- Cualquier uso particular como fotografía o grabación de deportes extremos⁴.
- Alternativa al reparto de paquetes o mensajería.
- Exploración e investigación científica

⁴Uso comercial debido a las empresas que venden estas aeronaves, sin embargo, se podría considerar la división alternativa como aplicaciones fundamentadas en “hobbies”

Militares

- Mayor seguridad para aviones y bombarderos de ataque.
- Mayor versatilidad en el frente.
- Reconocimiento preventivo de zonas conflictivas.
- Minimizar riesgos para los soldados.

Se observa que todas las aplicaciones se fundamentan en las tres principales características que aporta un vehículo aéreo no tripulado:

- **Reducción de riesgos:** en determinadas situaciones o conflictos se evita el posible daño personal, asumiendo sólo aquel material.
- **Versatilidad:** son por lo general diseños escalables y con mentalidad modular. Además los pilotos automáticos disponibles permiten controlar casi cualquier morfología de aeronave imaginable con la configuración adecuada.
- **Repetición y coordinación:** clave en tareas de carácter rutinario como vigilancias o repartos. Funcionalidades como ser capaces de planificar rutas o el vuelo autónomo coordinado con otras aeronaves brindarán un gran mercado a estas aeronaves.

Análisis del mercado

El sector de los vehículos aéreos no tripulados está en claro auge y posicionamiento como un gran mercado de futuro. Es vital comentar los números y predicciones que se manejan para los próximos años y explicar las principales características de este mercado. Se postularán las siguientes categorías:

Creciente: en 2020 se espera un crecimiento hasta los 11.2 billones de dolares, casi duplicando los ingresos de 2017 cifrados en 6 billones de dolares. [18]

Cambiante y volátil: desde 2016 hasta 2018 han desaparecido el 10% de las principales compañías de ese año (711). Por otra parte, casi 360 se han añadido, mostrando la gran variedad, cambio y la dificultad de mantenerse entre las grandes compañías, reforzándose además la idea de crecimiento constante. [31]

Diversificación: sector fundamentado en grandes compañías y pequeñas *startups* que servirán y nutrirán de I+D a estas empresas ya que serán absorbidas. Esta tendencia genera un mercado diversificado, con grandes disparidades pero que fomentará las pequeñas *startups* con financiación de mayores compañías.

DJI: se ha comentado la gran fragmentación y la existencia de grandes compañías junto a pequeñas empresas. Sin embargo, el dominio de DJI ha alejado a sus principales competidores donde compañías top 10 mundial se han visto incapaces de competir, como Parrot o 3RS. [31], [39] y [40]

Capital nuevo: la inversión de gigantes empresariales de otros ámbitos (Amazon, Google o Facebook) puede invertir determinadas tendencias, potenciar y fortificar este sector. [17]

Investigación: mayor automatización, algoritmos de inteligencia artificial, minería de datos o alianzas⁵ entre empresas de diversos ámbitos generan sinergias fomentadas en un aumento de la competitividad y una fuerte inversión en I+D. [31]

Factores condicionantes

Justificado el crecimiento, se deberán exponer las causas que acontecen a este fenómeno. Se desarrollarán grandes bloques que condicionan, guían y marcan el desarrollo del sector estos últimos años, así como el devenir del mismo a corto-medio plazo.

Factores tecnológicos

Se engloban en este bloque todas aquellas condiciones tecnológicas que han contribuido al desarrollo de pequeños UAV para aplicaciones tipo “hobby”:

- Capacidad de computo sustancialmente mayor con menores recursos.
- Miniaturización y abaratamiento de componentes en controladores, motores, chips, circuitos integrados, etc.
- Mejora de los sensores, algoritmos y hardware que permiten tanto un mejor control como una mayor cantidad de aplicaciones.
- Paradigmas multiplataforma e integración de controles con *smartphones* por ejemplo.

⁵Komatsu y NVIDIA, IBM y Hitachi o Qualcomm y Mitsui

Factores económicos

En lo relativo a incentivos o factores puramente económicos que permiten el gran desarrollo de la industria han sido expuestos en el análisis del mercado. Evidentemente los factores para potenciar este campo serán la inversiones de compañías de otros sectores o la pujanza de compañías como DJI, con el posible perjuicio⁶ que pueda ocasionar una situación de monopolio (copa un 70 %) por parte de la multinacional china.

Factores sociales

La concepción social será determinante en el crecimiento del sector ya que la venta recreativa de estas aeronaves constituye una base fundamental de este mercado⁷. Una mejora en la concepción social de estas aeronaves⁸, permitirá un entorno económico y de inversión favorable.

Factores legislativos

El marco normativo siempre ha constituido un gran condicionante para el desarrollo del sector aeronáutico. Generalmente muy restrictivo en lo relativo a seguridad, se necesitará siempre un marco normativo coherente con las necesidades actuales del sector para no frenar un futuro desarrollo. El marco legislativo dependerá de cada región y se verá condicionado a restricciones locales, siendo difícil exponer un análisis global del mercado en este marco.

El mercado español estará actualmente restringido bajo las disposiciones del Real Decreto 1036/2017 de 15 de diciembre de 2017 [16]. En este documento se establecerán todas las definiciones y regulaciones relativas a las sanciones, categorías y licencias para el vuelo de vehículos no tripulados. Todos los artículos recogidos en este decreto son de pleno derecho en todo el territorio nacional y han sido emitidos por el Ministerio de la Presidencia para las Administraciones Territoriales, regulando el espacio aéreo de aeronaves no tripuladas y modificando los siguientes decretos:

- Reglamento del aire y disposiciones operativas comunes para los servicios y procedimientos de navegación aérea. Real Decreto de 552/2014
- Reglamento de Circulación Aérea. Real Decreto de 57/2002.

⁶La incapacidad de competir en el sector fabricación y diseño, ha obligado a algunas empresas a trabajar únicamente en el mercado del desarrollo de software [31].

⁷DJI: el mayor fabricante únicamente construye y diseña drones para fotografía y grabación de videos

⁸Actualmente existen algunos conflictos en materia de seguridad aérea o violación de privacidad [32]

Alcance del proyecto y objetivos

2.1. Alcance

La herramienta PX4 que posteriormente se comentará en detalle, será un código versátil, multiplataforma, maduro y completo. Será una aplicación o *stack*¹ que necesitará de una serie de herramientas para interactuar y ser capaz de ejecutarse sobre un hardware: los Middlewares. El desarrollo de estos elementos, no buscará mejorar ni implementar ningún aspecto extra o permitir el completo manejo del piloto automático.

Se propone un desarrollo para los dos sensores básicos y necesarios en la obtención de orientación y navegación en un UAV: barómetro y unidad inercial. Cualquier diseño integro, completo y funcional de todo el código en un *all-in-one computer* o desarrollo integro de drivers queda completamente fuera del alcance de este trabajo.

2.2. Objetivos

Se ha evidenciado y justificado la elaboración del presente documento. Sin embargo, se expondrán primero los objetivos principales, los cuales constituyen el núcleo o matriz del proyecto. Por otra parte, habrán otros secundarios o complementarios que se fundamentarán en un conocimiento paralelo de carácter transversal adquirido por el propio desarrollo del proyecto y documentación consultada.

¹Definición contenida en el capítulo 4 de este documento

Objetivos principales

- Entender y utilizar uno de los principales pilotos automáticos del mercado.
- Documentar y generar código para la BeagleBone Blue.
- Elaboración del Middleware, drivers y HAL.

Objetivos secundarios

- Aprendizaje sobre plataformas *open-source*.
- Documentación sobre hardware y software novedoso.
- Colaborar, profundizar y usar herramientas de colaboración para proyectos de esta índole como GitHub.
- Verificación de código con los consiguientes problemas y conflictos originados entre el software-hardware.
- Recopilación de información sobre diferentes propuestas del sector y sus principales características.
- Profundizar en el conocimiento de las capas, organización y la arquitectura interna de un sistema informático.
- Interpretación, mejora y adaptación de códigos ya publicados a nuevos paradigmas, como otras plataformas o otros protocolos de comunicación.

Descripción del sistema hardware

BeagleBone Blue es un SBC¹ (*Single Board Computer*), *open-source*, de bajo consumo producido por Texas Instrument en asociación con DigiKey y Newark Element14.

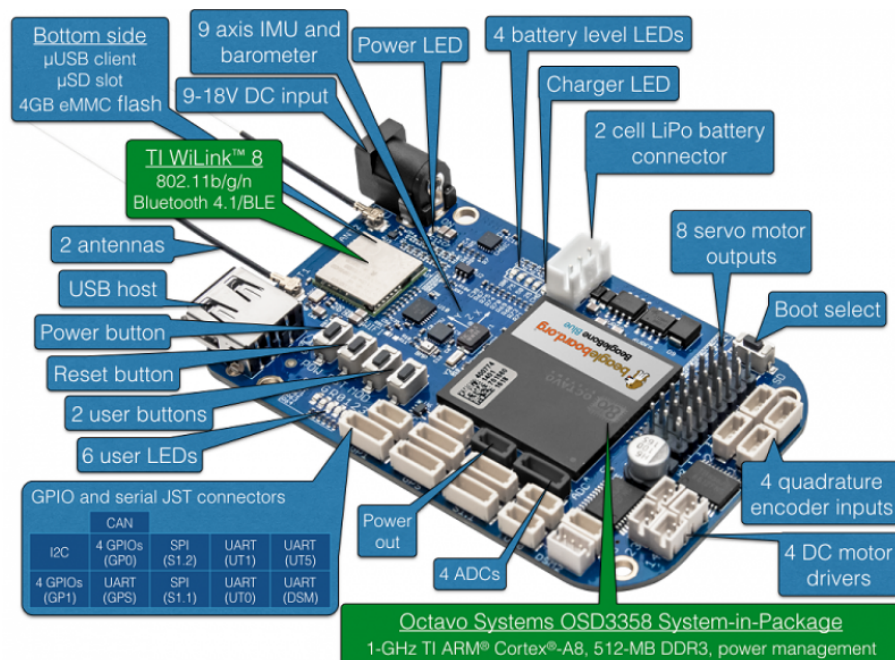


Figura 3.1: Detalle de la BeagleBone Blue destacando sus principales componentes [25].

Posibilidades

Constituye el último modelo de la serie BeagleBone, siendo el más avanzado y con características orientadas hacia el campo de la robótica. Dispone de uno de los hardwares más potentes del mercado en relación a almacenamiento, capacidad de

¹Siglas que describen un ordenador integrado en una única unidad, donde se localizarán todos los componentes necesarios que tiene un computador funcional. Suele presentar un tamaño y consumo reducido, siendo posible la denominación *all-in-one computer* como equivalente

cómputo y conexiones.

Las principales herramientas que constituyen esta compatibilidad y la hacen ideal para el campo de la robótica o la incursión en el campo de los UAVs son:

- Sensores de posicionamiento plenamente integrados y funcionales. Presenta una unidad inercial y un barómetro, brindando la posibilidad de seguimiento y *tracking*.
- Basada en Debian Linux y compatible con ROS (*Robotics Operative System*).
- Microprocesador con unidades programables en tiempo real PRUs integradas.
- Funcionalidades wifi y compatibilidad con Cloud9.
- Gran comunidad de desarrolladores.
- Memoria expandible mediante micro-SD.
- Multitud de puertos y conexiones: ADC, UART, SPI, I2C, GPIO, etc.
- 11 LEDs y 2 botones programables.
- Almacenamiento de estado sólido.
- Alimentación y carga de baterías LIPO.

Especificaciones

Processor	
Processor	AM335x 1 GHz ARM Cortex-A8
Graphics accelerator	SGX530
Acelerador	NEON floating-point
Microcontroladores	2x PRU 32-bit 200 MHz
Memory	
RAM	512 MB DDR3 800 MHz (Integradas en OSD3358)
ROM	4GB 8-bit eMMC
SD	Expandible
Software	
Debian Ardupilot ROS	

Cloud9
IDE on Node.js w/BoneScript

Conectividad	
Puerto	High speed USB 2.0 Client port: Access to USB0 Client mode via microUSB
Puerto	High speed USB 2.0 Host port: Access to USB1 Type A Socket, 500mA LS/FS/HS
Wifi	WiLink 1835 WiFi 802.11 b/g/n 2.4GHz. Supports the following modes 2x2 MIMO,AP,SmartConfig, STA, Wi-Fi Direct, Mesh over Wi-Fi based on 802.11s
Wilink	WiLink 1835 Bluetooth 4.1 with BLE
Puertos Serie	UART0, UART1, UART5 available via 4 pin JST-SH connectors UART2 available via 6 pin JST-SH connector (EM-506 GPS style connector) UART4 RX available via 3 pin DSM2 (JST-ZH) connector
I2C	I2C1 available via 4 pin JST-SH connector
SPI	SPI1 CS0 (S1.1) and SPI1 CS1 (S1.2) available via 6 pin JST-SH connectors
CAN	CAN available via 4 pin JST-SH connector (includes TCAN1051 CAN transceiver)
GPIO	8 GPIOs (GP0 and GPI1) available via 6 pin JST-SH connectors
ADC	ADC inputs 0 to 3 available via 6 pin JST-SH connectoar
Voltaje	3.3VDC and 5VDC power output via 4 pin JST-SH connector
Power Management	
PMIC ² Cargadores	TPS65217C PMIC is used along with a separate LDO to provide power to the system (Integrated in the OSD3358) 2 cell (2S) LiPo battery charger (powered by 9 – 18VDC DC Jack)

²Power Management integrated circuit

Regulador	6VDC 4A regulator to drive servo motor outputs
	Debug support
	JTAG test points
	Power source
	microUSB USB cell (2S) LiPo battery JST-XH connector 9 - 18VDC DC Jack
	User Input / Output
	Power Button Reset Button Boot Button user configurable buttons (MOD, PAU)
	11 user configurable LEDs (USR0-3, Red, Green, WIFI, Battery 0-3) Charger LED Power LED
	Motor Control (requires either DC Jack or 2S battery)
	4 DC motor drivers 4 Quadrature encoder inputs Servo motor outputs
	Sensors
	9 axis IMU Barometer

Cuadro 3.1: Especificaciones de la BeagleBone Blue [25]

3.1. Procesador

Arquitectura ARM Cortex A-8, con soporte para Linux. Complementado con un motor gráfico y dos microcontroladores auxiliares. Estos permiten la generación de pulsos PWM³ en tiempo real, sin los errores y retardos propios de las arquitecturas con microprocesador.

La combinación de ambos dispositivos permite una capacidad de cómputo superior con una óptima gestión de las ordenes de más bajo nivel y los pulsos necesarios para los motores y otros dispositivos electrónicos. Además de esta característica interesante desde el punto de vista del control de dispositivos, dispone de un acelerador gráfico y de coma flotante, optimizando su rendimiento en estos campos donde los microprocesadores pueden tener problemas⁴.

3.2. Conexiones e interacciones

La funcionalidad wireless, así como la ranura para micro-SD permitirán varias combinaciones para interactuar y/o gestionar los archivos internos de nuestro hardware:

- Cloud9: Constituye el IDE (*Integrated Development Environment*) nativo ofrecido por la BeagleBone Blue para la gestión, compilación y ejecución de archivos. Ejecutado sobre la librería Javascript/Node.js permitirá una gestión y visualización óptima de aquellos resultados obtenidos. Solo compatible con los navegadores Firefox o Chrome.
- Protocolo SSH: interacción mediante la consola o terminal nativo de Linux. Necesitará algunos ajustes del protocolo para iniciar la conexión según la versión disponible en el ordenador y la BeagleBone Blue. Proporciona una gestión menos intuitiva.
- Conexión wifi: alimentando la BeagleBone Blue de electricidad es posible el acceso a la red generada por la misma. Intercambio de información similar al protocolo SSH y gestión mediante terminal de Linux.
- Archivos vía microSD: comunicación viable para aumentar la capacidad o para *flashear* una nueva imagen de Linux en la BeagleBone Blue.

³*Pulse Width Modulation*: la modulación por ancho de pulso constituye uno de los principales elementos de control de dispositivos electrónicos. En una onda rectangular con periodo o frecuencia fija, el tiempo a mayor nivel se le llama pulso, teniendo asociado un ancho. Modificando este ratio alto/bajo nivel se pueden controlar señales y ciclos de trabajo, siendo esta precisión solo viable en μC o sistemas en tiempo real.

⁴Gestión específica de los recursos gráficos o operaciones en coma flotante optimizan el rendimiento y liberan al procesador principal de funciones para las cuales no ha sido especialmente diseñado.

3.3. Sensores

Para el caso de la BeagleBone Blue, sólo será accesible el I2C-2. En la figura 3.2 se observa el detalle de la salida del procesador principal y en las siguientes imágenes se apreciará la conexión a cada uno de los sensores integrados.

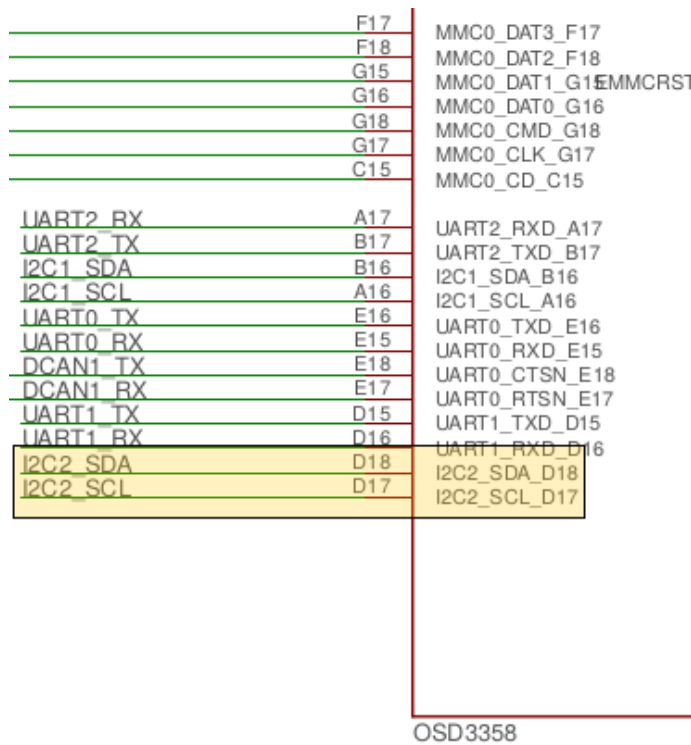


Figura 3.2: Detalle resaltado de la conexión I2C desde el SIPB de OSD3358 para la BeagleBone Blue [25].

Barómetro

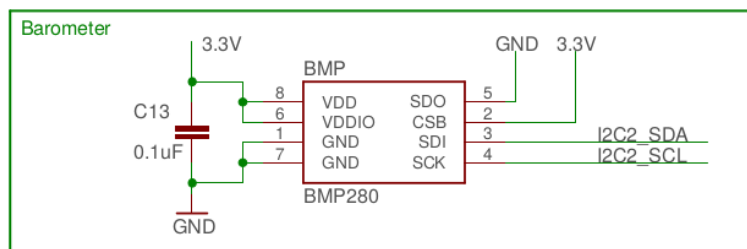


Figura 3.3: Diagrama en detalle de las conexiones del barómetro BMP280 en la BeagleBone Blue [25].

BMP280: sensor de presión barométrica absoluta fabricado por Bosch. Sus características (tamaño reducido y bajo consumo) y fabricación están orientadas a tareas o aplicaciones en sistemas móviles [23]. En este caso, como se aprecia en 3.3 estará conectado I2C, en concreto el canal I2C_2.

Unidad inercial

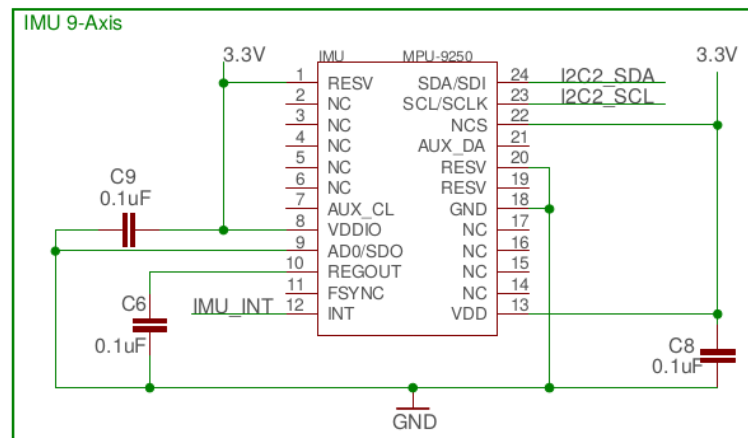


Figura 3.4: Diagrama en detalle de las conexiones de la IMU MPU9250 en la BeagleBone Blue [25].

MPU9250: unidad inercial de 9DOF. Fabricada por Invensense, presenta 3 acelerómetros, 3 giróscopos y 3 magnetómetros. Constituye una versión mejorada y más completa de la serie MPU6050 (esta no incluye magnetómetros), compartiendo muchas características básicas y registros internos con esta familia [22].

Conectada mediante el protocolo I2C, en concreto por el canal I2C_2 como se puede apreciar en la figura 3.4. Necesitará una configuración y gestión interna de los magnetómetros diferente, ya que estos sensores estarán conectados por I2C al chip principal de forma interna.

3.4. Configuración y utilización

En este documento no se expondrán las configuraciones iniciales o comandos necesarios para actualizar y gestionar los recursos de la BeagleBone. El proceso completo se expondrá y documentará en el manual de usuario adjunto a este documento.

Por otra parte, mas adelante⁵ se comentarán algunas funcionalidades necesarias para ser capaz de exportar y ejecutar el código sobre este hardware objetivo.

⁵Apartado de Resultados y tests.

Sistemas Operativos: Conceptos básicos

Se deberá comentar de forma genérica la organización y jerarquía de cualquier sistema informático o de control para poder entender el desarrollo propuesto a lo largo del documento. Por otra parte, se podrá abreviar en algunos aspectos una vez la componente sistémica y técnica se haya introducido a lo largo de esta sección.

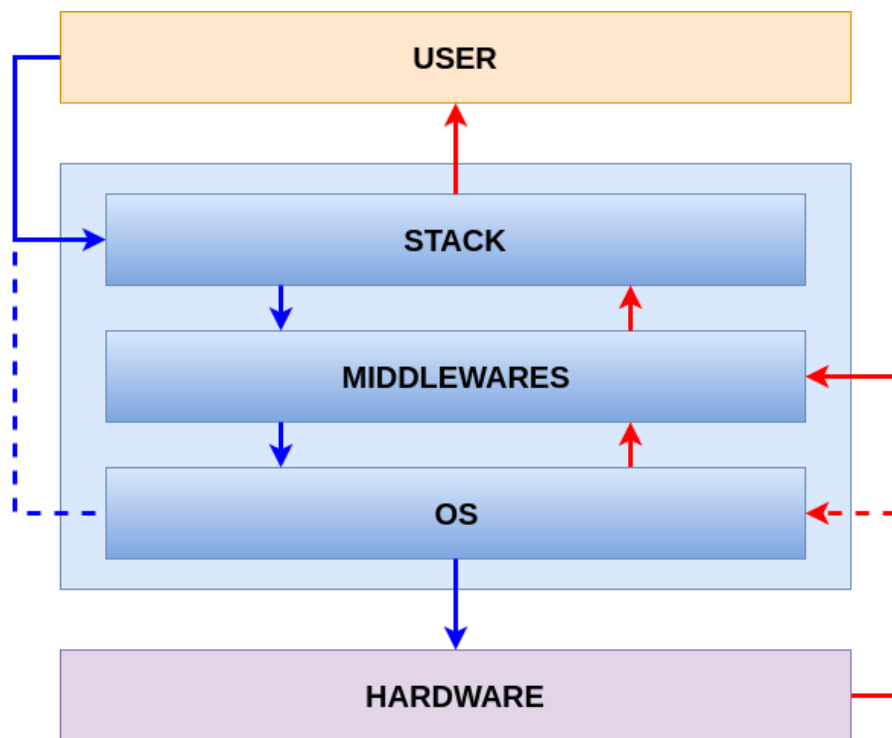


Figura 4.1: Estructura de capas básica de un sistema informático.

4.1. Hardware

Hardware es un término genérico empleado para describir cualquier elemento electrónico que permite el cómputo, almacenamiento o intercambio de información. Como se puede ver en 4.1 constituye la primera capa de cualquier sistema industrial o informático y es todo elemento físico o tangible capaz de procesar, almacenar o regular las entradas y salidas de información del sistema.

Esta primera capa será fácil de definir y reconocer, ya que constituye la parte visible del sistema. El interés en conocer la misma se fundamentará en la necesidad de entender la estructura y organización interna de los elementos del hardware para así poder interactuar con los mismos. Por tanto, se deberá siempre tener información sobre la capacidad de cómputo de un procesador, de transmisión de un bus, el protocolo utilizado, la conexión de los diferentes sensores, etc. En definitiva, para el desarrollo, diseño y programación de cualquier interfaz *Middleware* es vital conocer las características físicas de un diseño.

Para el caso propuesto, podremos diferenciar diversos tipos de hardware según su finalidad o disposición:

- Microprocesador: abarca toda la capacidad de cómputo del sistema. Coordina todas las acciones y elementos. Contiene la ALU¹ y la CU², aunque su arquitectura interna queda fuera de las competencias expuestas en este documento. [10]
- Memoria: almacena la información necesaria para hacer funcionar al sistema. En general se albergan diversos tipos de memorias según capacidad, accesos y rapidez.
- Buses de datos: conectan internamente registros del procesador o el mismo con otros elementos del sistema electrónico. Son los encargados de distribuir la información en cualquier sistema informático, pudiendo interactuar con elementos internos o sirviendo como enlace para conectar periféricos.
- Sensores: recopila datos sobre los estados externos al sistema. Pueden ser *I/O Devices* conectados por buses de datos (puertos serie) o estar integrados en la arquitectura.

¹*Arithmetic Logic Unit*: es un circuito digital encargado de calcular operaciones aritméticas y lógicas entre argumentos. Constituyen la base de cómputo de un procesador (podría tener más de una ALU) y utilizará la representación en complemento a dos en contraposición a los *Floating point Units* que están sobrepasando a estas unidades actualmente debido a su cómputo en coma flotante.

²*Control Unit*: constituye uno de los 3 bloques principales de una CPU (unidad central de procesamiento). Su función es buscar las instrucciones en memoria, decodificar y ejecutar usando otra unidad, denominada unidad de procesos.

Todo sistema físico necesita de instrucciones correctamente introducidas para cumplir con su finalidad y diseño inicial. Este hecho, conducirá a la capa inmediatamente superior en la organización interna de un sistema electrónico: el software.

4.2. Software

El software será el conjunto de capas que darán y ordenarán las instrucciones para que puedan ser ejecutadas a más bajo nivel por el hardware. Será todo aquel elemento no visible y que permite el correcto funcionamiento del sistema. En lo relativo al software, su organización y funciones serán más difusas que las del hardware.

En el diagrama de la figura 4.1 se establece el flujo de información caracterizado con líneas rojas, aquellas que se elevan hacia el usuario. En contraposición, se establecen las líneas azules que serán aquellas instrucciones a realizar que evidentemente tendrán carácter descendente, comandadas por el usuario, gestionadas por el software y ejecutadas por el hardware.

Por otra parte, las líneas discontinuas se fundamentan en aquellos flujos posibles pero no constantes, es decir, el usuario puede modificar y ordenar instrucciones al SO pero suele hacerlo a través de terceras aplicaciones o el software puede recibir información del hardware para comprobaciones aunque lo lógico es el flujo y procesamiento en capas superiores a través del Middleware.

Sistema Operativo

Capa inmediatamente superior al hardware que constituye el software principal de cualquier sistema informático o industrial. Es el encargado de controlar las principales características del sistema pudiendo ejecutar sobre ella otras tareas más específicas.

Su estructura y organización suele ser bastante compleja y diversa, ya que son muchos sus usos, funciones y variantes. Fundamentalmente es el encargado de gestionar recursos, actividades y procesos de la forma más conveniente o eficiente en base a las especificaciones del hardware que esté controlando.

No se profundizará en conceptos más extensos sobre arquitectura interna de sistemas operativos ya que el trabajo no lo requiere. Sólo se deberá saber lo introducido hasta ahora: la funcionalidad principal, el hecho de ser la capa más cercana al hardware y servir de base para otras aplicaciones más específicas. Asimismo, más adelante se evaluará e introducirá el sistema operativo Linux, el presente en la BeagleBone Blue, destacando alguna característica más concreta y que se deberá conocer para entender las actuaciones propuestas en el desarrollo del trabajo.

Middleware

El Middleware o *interlogical* es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, programas, redes o hardwares. Es una capa situada encima del sistema operativo y por debajo de las aplicaciones en ejecución que sirve como intérprete de instrucciones. Simplifica y abstrae la complejidad y heterogeneidad de las conexiones de hardware subyacentes, generando una interfaz de programación que simplifica el manejo y construcción de aplicaciones.

Se puede asumir que existen muchas variantes y nombres para todo el conjunto del *Middleware* que necesita una aplicación para funcionar sobre un hardware específico. Se podrán diferenciar por ejemplo, los denominados:

- *Hardware Abstraction Layer* (HAL): es la capa intérprete, que coordina y extrae información de los elementos de más bajo nivel.
- Drivers: software primario que interactúa con un elemento (sensor, periférico, puerto) y que permite la extracción o control de la información contenida en el mismo.

En definitiva, el Middleware permite el desarrollo, gestión y programación para que los elementos de más bajo nivel se integren con la aplicación y el Sistema Operativo, brindando un entorno para el desarrollo simplificado, programación y manejo de aplicaciones (API³) [11]. La HAL proporciona un ecosistema que evita la complejidad del nivel más bajo cuando las aplicaciones se ven obligadas a acceder a información de esta capa. Finalmente, dentro de esta estructura existen unos elementos que procesan la información directamente desde los componentes físicos del sistema.

Nota: en algunos casos los drivers, HAL, middleware o algunos elementos del sistema operativo se pueden confundir o difuminar dentro de la organización interna del sistema. Será interesante entender y diferenciar la finalidad de cada uno de los mismos, sobre todo para generar interfaces compatibles con una arquitectura específica.

³*Application Programming Interface*: es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como capa de abstracción.

4.3. Firmware

“La combinación de instrucciones de un dispositivo de hardware e instrucciones y datos de computadora que residen como software de solo lectura en ese dispositivo”. (IEEE, Std 610.12-1990).

En función de la definición aportada, el Firmware será aquel elemento de software de sólo lectura que especializará el sistema de forma definitiva. El ejemplo más evidente de Firmware será la BIOS, un estándar de facto para definir la interfaz de Firmware en ordenadores compatibles con este conjunto de normas. Su principal función sera iniciar y probar el hardware, siendo el primer programa que se ejecutará, cargando el gestor de arranque del sistema operativo.

La denominación de Firmware presente en los repositorios de GitHub para el código que se estudiará no es Firmware como tal, pero este nombre se refiere a la posibilidad de poder compilar y ejecutar el código directamente una vez descarga del sitio web sin ninguna consideración especial.

4.4. Stack o aplicación

El *stack* o aplicación será cualquier código funcional capaz de realizar una o diversas tareas específicas. Constituye la capa superior de una arquitectura básica de computadores (por debajo del propio usuario), y deberá ser compatible con el sistema operativo y hardware empleado.

Como se puede observar en el diagrama de la figura 4.1 recibe las instrucciones directamente del usuario del sistema. En el desarrollo de sus funciones será el *Middleware* el encargado de suministrar y procesar la información proveniente de las capa inferior (hardware), para así poder desempeñar su función. Será la capa que más interactuará con el usuario, tanto a nivel de órdenes como de información procesada y elevada a este último nivel de la jerarquía expuesta.

Para la propuesta de este documento, se definirá el *flight stack* como todos los elementos de control, comunicación y registro que permiten el vuelo autónomo y controlado de la aeronave. La arquitectura general de un piloto automático y las particularidades del código PX4 se comentarán en los sucesivos apartados del trabajo.

4.5. GNU/Linux

Iniciativa de código abierto⁴, desarrollado a mitad de los años 80 sobre un núcleo o kernel⁵ UNIX⁶ para la creación un sistema operativo completamente funcional y libre, auspiciado por la *Free Software Foundation*.

En concreto, a pesar de que la jerga popular se conoce al sistema completo como Linux, esta arquitectura solo constituye el núcleo o kernel del mismo. Para referirse al sistema operativo completo se debería utilizar el nombre GNU/Linux, aunque esta confusión existe incluso en algún documento bibliográfico [12].

Entre las características más destacadas de este sistema operativo se resaltan:

- Software libre y gratuito.
- Comunidad de desarrollo y actualización constante⁷.
- Gran capacidad para ser configurado y editado⁸.
- Multitarea robusta y multiusuario. Tratamiento individual de procesos, permite cierta redundancia y capacidad de maniobra al sistema ante fallos críticos.
- Multiplataforma.

Por otra parte, también es interesante introducir determinadas características interesantes desde el punto de vista de este proyecto, ya que la BeagleBone se basa en una distribución Debian. Estas características en su mayoría se remiten a la arquitectura interna de este OS:

⁴Licencia GNU General Public License: permite la utilización del código respetando la autoría original y con el compromiso de liberar a su vez los cambios bajo la misma licencia.[33]

⁵Núcleo o kernel: parte del sistema operativo encargado de distribuir los recursos a los programas ejecutados. Inicia y para las diferentes tareas a ejecutar, en contraposición a SO que constituye todo el conjunto de programas que son suficientes para realizar una variedad de trabajos. Este propósito general hace que sea necesario un manejo de todos los trabajos que cualquier usuario pueda querer realizar.[12]

⁶Sistema operativo portable, multitarea y multiusuario desarrollado en 1969

⁷Estabilidad o posibilidad relativas a la configuración son características dependientes de la distribución y versión utilizada.

⁸Algunas distribuciones y características pueden ser completamente configurables pero no ser demasiado “amigables” desde el punto de vista usuario.

Soporte para POSIX

El *Portable Operating System Interface* es un estándar especificado por la IEEE⁹ para mantener la compatibilidad entre sistemas operativos. Estar sometido a estos estándares o normas de programación para las interfaces de sistema asegura una portabilidad a cualquier arquitectura coherente con estas normas.

Sistemas de archivos de linux.

Constituye una de las principales características del sistema operativo. Todos los recursos como documentos, directorios, unidades, periféricos, buses, comunicaciones o procesos son flujos de bits expuestos a través del sistema de archivo. La gestión interna de los archivos, rutas de direccionamiento o descriptores de archivo son algunos de los términos generados y empleados por el sistema para la comunicación interna. Un adecuado control y gestión de estos documentos permite reconfigurar totalmente el sistema mediante la edición de archivos de texto.

Sin embargo, se destacarán diversos directorios debido al futuro interés que pueden suscitar en el desarrollo de las interfaces Middleware:

/dev/: Archivos de dispositivos. Serán importantes a la hora de acceder a puertos serie, buses o cualquier otro dispositivo del sistema.

/sys/: Archivos referentes al sistema. Estructura o gestor virtual proporcionado por el kernel del SO. Permitirá acceder a las principales características y configurar aspectos básicos del mismo.

/lib/: Librerías.

Esta organización de archivos directamente extraídos del kernel, simplifica en gran medida el trabajo con drivers. Esta referencia a los archivos que conectan con los puertos o buses, permite acceder directamente a las rutas, sin entrar en registros internos del hardware. Esta ventaja aparente, permite una mayor versatilidad del sistema operativo, además sirve como capa de abstracción de los niveles más bajos de un sistema informático.

⁹Institute of Electrical and Electronic Engineers

4.6. Compilador

Un compilador es un programa informático que traduce un lenguaje de programación a un lenguaje de máquina capaz de ser ejecutado. Es el primer proceso en la ejecución de un código funcional, aunque la compilación no asegura la exención de errores del mismo al ser ejecutado sobre la plataforma.

No será necesario complementar la información expuesta con las partes y el funcionamiento interno de un compilador, simplemente será necesario saber de su existencia. La forma de compilar un archivo es variada según el programa o compilador escogido, para este proyecto se realizará mediante el comando `make`. El archivo raíz o origen cargará definiciones y documentos a incluir, iniciándose así el proceso.

A lo largo de los documentos relacionados se establecerán una serie de archivos *CMakeLists* nativos a cada directorio, que constituirán las instrucciones para enlazar y seguir el proceso. Estos generarán y compartirán de forma adecuada los directorios generados y anclados¹⁰ al programa principal, siendo estos *make files* los que controlen el proceso y generen sucesivamente nuevas invocaciones hasta concluir el procedimiento.

4.7. Protocolos de comunicación

Además de la estructura interna de los ordenadores, se deberá comentar los intercambios de comunicación entre los diferentes componentes del hardware. La interacción mediante buses de datos es vital ya que garantizará un óptimo intercambio de información y será crítico en determinadas aplicaciones (p.e. UAVs, debido a que se debe mantener unos niveles de transmisión de datos óptimos e iguales en todos los componentes). Sin embargo, los buses y elementos físicos son propios del diseño, por tanto, solo se introducirán los principales protocolos de conexión y envío de datos, sin remitirse a velocidades de envío o características concretas para diseños particulares:

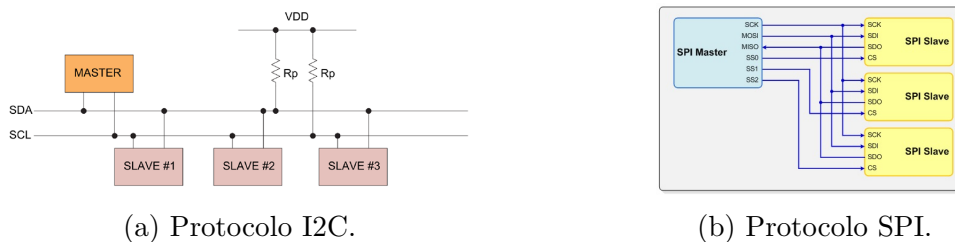


Figura 4.2: Comparación de los principales protocolos de comunicación.

¹⁰Como se comentará más adelante, los archivos generados serán compartidos, no es posible la ejecución en estático del archivo binario principal.

I2C

Circuito interintegrado: constituye un protocolo de transmisión desarrollado en la década de los 80. Utilizado en la comunicación entre controladores, circuitos integrados y periféricos. Su funcionamiento se basa en un esquema maestro-esclavo y utiliza dos cables, bus de datos (SDA) y un reloj (SCL) como se puede apreciar en la figura 4.2a. Esta configuración generará una simplicidad a nivel de hardware y complejidad a nivel de controladores de dispositivos ya que la información se debe gestionar y sincronizar de forma correcta.

SPI

Bus de comunicación periférico: constituye otro protocolo muy extendido en la industria. Necesitará 3 puertos y un cable por cada uno de los elementos conectados al bus de datos (figura 4.2b). Esta característica generará dificultades asociadas al montaje y reconfiguración, aunque aportará robustez y simplicidad en la gestión de la información enviada.

Comparación

En las siguientes categorías se comparan las posibilidades y ventajas que ofrece cada una de las alternativas de comunicación. En general la utilización de un protocolo u otro se ve condicionado al diseño propuesto. [43]

Ventajas I2C

Nº de conexiones: Su configuración permite que sea operable con 2 cables (Datos y Reloj), siendo necesarios $3+n$ conexiones para SPI (donde n es el número de dispositivos conectados).

Voltaje máximo: Permite diversos voltajes. Por contra, en conexiones SPI se vuelve más restrictivo aquel elemento que tenga menor voltaje máximo.

Cambios de configuración: Solo son posibles mediante el protocolo I2C que también permite modo multi-maestro. SPI limitado por diseño del cableado.

Añadir dispositivos: posible con I2C añadiendo nuevos registros de dirección. Necesidad de cablear y modificar el diseño para el caso SPI.

Ventajas SPI

Simplicidad: no necesita ningún driver ni software, ya que son todas conexiones directas. I2C necesitará de algún elemento para controlar la información del canal.

Rapidez: mayores velocidades de transmisión para la conexión SPI. La ventaja del colector abierto, limita y limita las velocidades de transmisión a la capacitancia de las redes.

Redundancia: el protocolo SPI se encuentra totalmente duplicado, separando lectura y envío. I2C solo estará medio duplicado, un único bus.

Conflictos: permite más posibilidades y registros de accesos sin estar preconfigurados. I2C podría llevar a más conflictos en el acceso a registros, siendo necesario que estos sean especificados en diseño.

Descripción del sistema software

Ademas del hardware fundamental que se empleará como base en el proyecto, se deberán comentar las principales características, partes y funcionalidades del código a utilizar.

5.1. ¿Qué es?

“PX4 es un proyecto de código y hardware libre que tiene como objetivo convertirse en uno de los mejores y mas populares pilotos automáticos del mercado. Esta centrado en la industria, el ámbito académico y comunidades de entusiastas, siendo parte de la Linux Foundation DroneCode y se rige por una licencia BSD.”(*PX4 Development Team*, 2014)

Se observa en base a la definición propiamente dada en su pagina web [2], que se trata de un ambicioso proyecto para la unificación y creación de un piloto automático que pueda competir en todos los ámbitos del mercado, académico o industrial. Entre sus fundamentos y características destacan una potente comunidad que sirve de base y desarrollo desde que se fundara la iniciativa.

5.2. DroneCode

DroneCode es un sistema multiplataforma, multidispositivo, completo y versátil fundamentado en los principios del código libre y el desarrollo por parte de una comunidad de usuarios. Por tanto, se trata del marco global fundacional donde encaja el código de piloto automático abordado. En esta iniciativa se incluirán y constituirán los principales complementos necesarios para controlar un UAV plenamente operativo y funcional.

A continuación, se exponen los principales componentes que constituirán el proyecto y serán responsables de cada una de las partes básicas de un UAV plenamente funcional serán:

- MAVlink: módulo de comunicaciones que permite la comunicación tierra-vehículo.
- Ground Control Systems: sistemas encargados de gestionar la información recibida por la aeronave y controlar los principales aspectos del mismo para un vuelo controlado y planificado.
- Hardware: previamente definido, constituye todo elemento físico y tangible capaz de almacenar, procesar o transmitir información. Base de ejecución para cualquier aplicación o código que deberá interactuar con el hardware a través del Middleware.
- Simulation: posibilidad de ejecutarse sobre un ordenador sin la necesidad de controlar ninguna aeronave o elemento físico.
- Developer APIS: marco general para introducir aplicaciones secundarias o añadir nuevas características y funcionalidades.
- PX4 o código de vuelo. Constituirá el núcleo principal de este trabajo, siendo el piloto automático responsable del control.

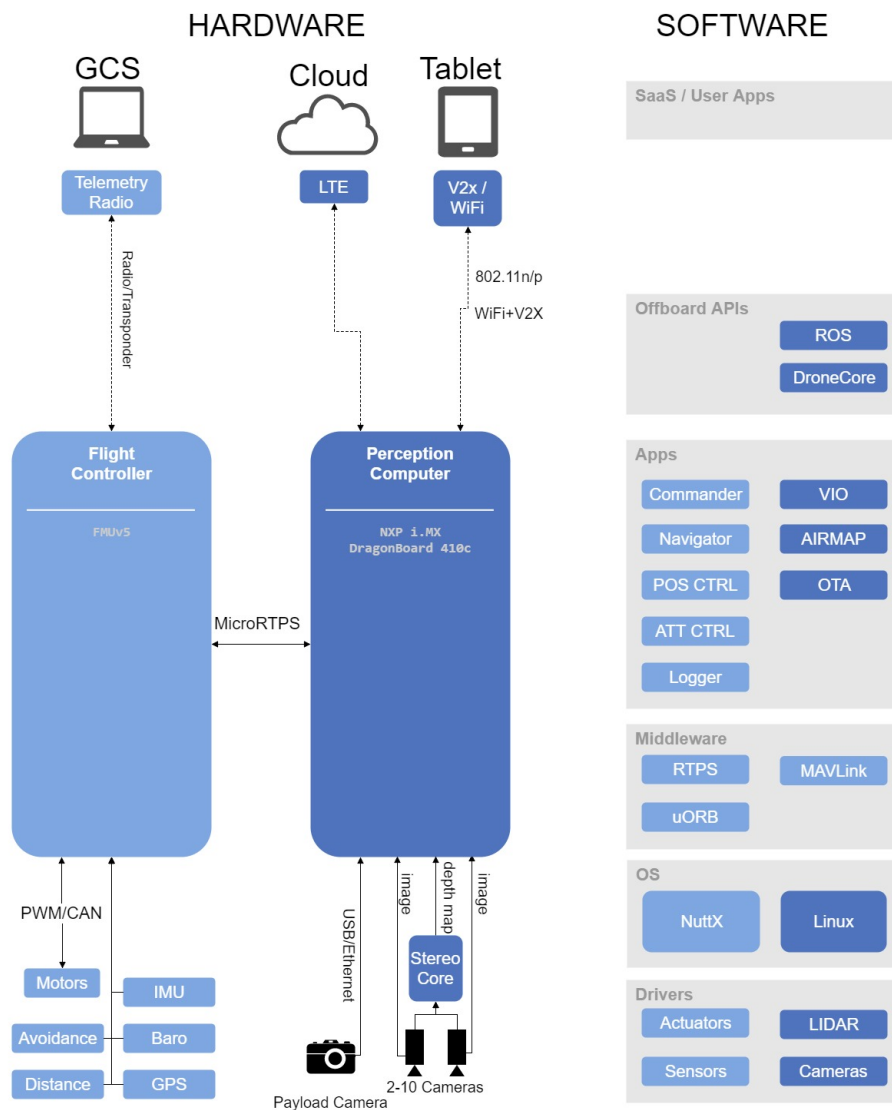


Figura 5.1: Detalle de la arquitectura DroneCode y las principales componentes [3].

La finalidad última de este trabajo es elaborar las interfaces necesarias para que el código o algoritmo de control se comunique con las capas de hardware a más bajo nivel. Por tanto, sólo se alterarán archivos correspondientes al último punto aunque, como se comentará más adelante, se utilizará la comunicación MAVLink, el QGroundControl o APIs para ser capaces de ejecutar el código. Por tanto, a nivel individual la separación completa de los elementos es inviable, ya que las comunicaciones, por ejemplo, se integran dentro del código operativo de PX4.

5.3. Miembros

Por otra parte, destacan el número de empresas que respaldan o forman parte de la iniciativa, donde se podría remarcar:

Miembros platino

- 3DR: empresa pionera en la creación de UAV, que actualmente dedica sus esfuerzos a la creación de software de escaneo, posicionamiento y reconocimiento de imágenes para UAVs.
- Qualcomm: uno de los mayores fabricantes de chipsets móviles del mundo.
- Intel: el mayor fabricante de circuitos integrados del mundo.

Otros miembros

- Miembro oro: Yuneec Aviation Technology
- Miembros plata: Aerotenna, Airmap, DroneDeploy, DronesLab, Natilus, Vertical Technologies, Wingtra,etc.
- Patrocinadores: UAVIATORS, Open Source Robotics Foundation, Open TX, Aerospace Design Lab, UAVCAN,etc.

Ha quedado evidenciado que además de una poderosa y numerosa comunidad de desarrolladores, se exhibe un fuerte respaldo empresarial. Algunas de los mayores fabricantes de productos electrónicos, chipsets, desarrolladores de software para UAV o fabricantes de drones se engloban dentro de la iniciativa. Por otra parte, la organización interna y forma en que se regula el proyecto no serán necesarias en este trabajo.

5.4. BSD

La licencia BSD se trata de una distribución de software libre, similar a la licencia MIT, que autoriza la distribución y copia de los contenidos del código, aunque se reconoce la autoría original del software. La licencia permite no liberar el código por parte de terceros desarrolladores, siendo esta la principal diferencia con la otra licencia de código abierta más utilizada en el mercado, la de tipo GPL(*General Public License*).

Todos los documentos del código, contienen el siguiente encabezado, que remiten a la autoría original y las condiciones específicas de todos los documentos dispuestos en la plataforma:

```

/*****
*
* Copyright (C) 2016 Julian Oes. All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
* notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in
* the documentation and/or other materials provided with the
* distribution.
* 3. Neither the name PX4 nor the names of its contributors may be
* used to endorse or promote products derived from this software
* without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
* FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
* COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
* BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
* OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
* AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
* ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
*
*****/

```

Como se puede ver, se exige la presencia de este encabezado en todos los archivos generados y la comunidad es responsable y supervisora de las actividades de desarrollo del entorno DroneCode. Este encabezado reconoce la autoría y marca las posibilidades que el usuario dispone en cuanto a la utilización y distribución del contenido de la plataforma.

5.5. Hardware compatible

Existen numerosos modelos en el mercado compatibles con el código propuesto, sin embargo, se establecerá una pequeña diferenciación:

Serie Pixhawk

Iniciativa independiente y de carácter abierto que intenta producir hardware funcional y de bajo coste para proyectos de índole académico, lúdico o industrial. Basados en NuttX OS, un sistema operativo en tiempo real, escalable para microcontroladores de 8 a 32 bit y que se basa en la aplicación de los estándares POSIX y ANSI.

- HKPilot32
- DroPix
- Pixracer
- Pixhawk. Modelos: 2,v3,3 Pro, 4, v5.

Para tareas intensivas de computación

En este caso, la arquitectura objeto es la de los microprocesadores. No disponen de sistema operativo en tiempo real, ya que en su mayoría utilizan software libre Linux y serán bastante mas potentes que las alternativas basadas en microcontroladores.

- Raspberry pi 2 con Navio 2
- Qualcomm Snapdragon Flight
- Intel Aero Ready to Fly Drone

UAV comerciales

Constituyen los modelos comerciales plenamente funcionales que han sido adaptados para ser capaces de ejecutar el *PX4 flight stack*:

- Crazyflie 2.0
- Parrot Bebop

5.6. Aeronaves controlables

Evidentemente, se entiende que la versatilidad y usabilidad de un piloto automático son características prioritarias para su acogida en el mercado. En general, las iniciativas de código libre y bajo coste, buscan el mayor número de soportes y misiones alcanzables por los usuarios.

PX4 no plantea una filosofía diferente, su componente fundamental y su estructura modular permite que una vez procesada la información se obtenga la respuesta en función de la plataforma o modo de vuelo escogido. Se pueden clasificar y englobar los diferentes tipos de aeronaves y opciones de código según la siguiente disposición:

- Aviones
 - Aviones de ala fija
 - Alas volantes
 - Alas en V invertidas
- Multicópteros
 - Helicópteros
 - Tricópteros
 - Quadricópteros
 - Hexarotores
 - Dodecarotores
- VTOL Airframes
 - *Tailsitters*
 - *Tiltrotors*
 - *QuadPlanes*
- UGVs/Rovers

5.7. Modos de vuelo

La versatilidad no solo es una característica básica, sino que la posibilidad de regular el nivel de asistencia y el modo de vuelo, se puede demostrar como imprescindible también en esta clase de códigos. El hecho de disponer de todas estas funcionalidades brindan un gran valor añadido al software y permitirán diferenciarse de otros códigos equivalentes.

Un modo de vuelo será como el piloto automático responde a los movimientos demandados y controla el movimiento del vehículo. Estarán descritos y agrupados según la siguiente descripción:

MANUALES

Serán aquellos modos en los que el usuario tiene control directo del vehículo a través del control RC (joystick). A pesar de este movimiento de seguimiento de las instrucciones el nivel de actuación dependerá del modo, debiendo ser este acorde con la experiencia y nivel del piloto.

- Ala fija, rovers o barcos
 - MANUAL
 - ESTABILIZADO
 - ACRO
- Multirotores
 - MANUAL/ESTABILIZADO:
 - ACRO
 - RATTITUDE

ASISTIDOS

Constituyen el conjunto de modos de pilotaje controlado por el usuario pero utilizan un nivel de asistencia automática. Pueden ayudar a controlar y recuperar el vuelo controlado.

- ALTCTL: control de altitud
 - Aviones de ala fija
 - Multirrotores
- POSCTL: control de posición
 - Aviones de ala fija
 - Multirrotores

AUTOMÁTICOS

Aquellos modos que apenas requieren controles del usuario.

- AUTO_LOITER
 - Aviones de ala fija
 - Multirrotores
- AUTO_RTL (vuelta a tierra)
 - Aviones de ala fija
 - Multirrotores
- AUTO_MISSION
 - Todo tipo de sistemas
 - *Off-board*

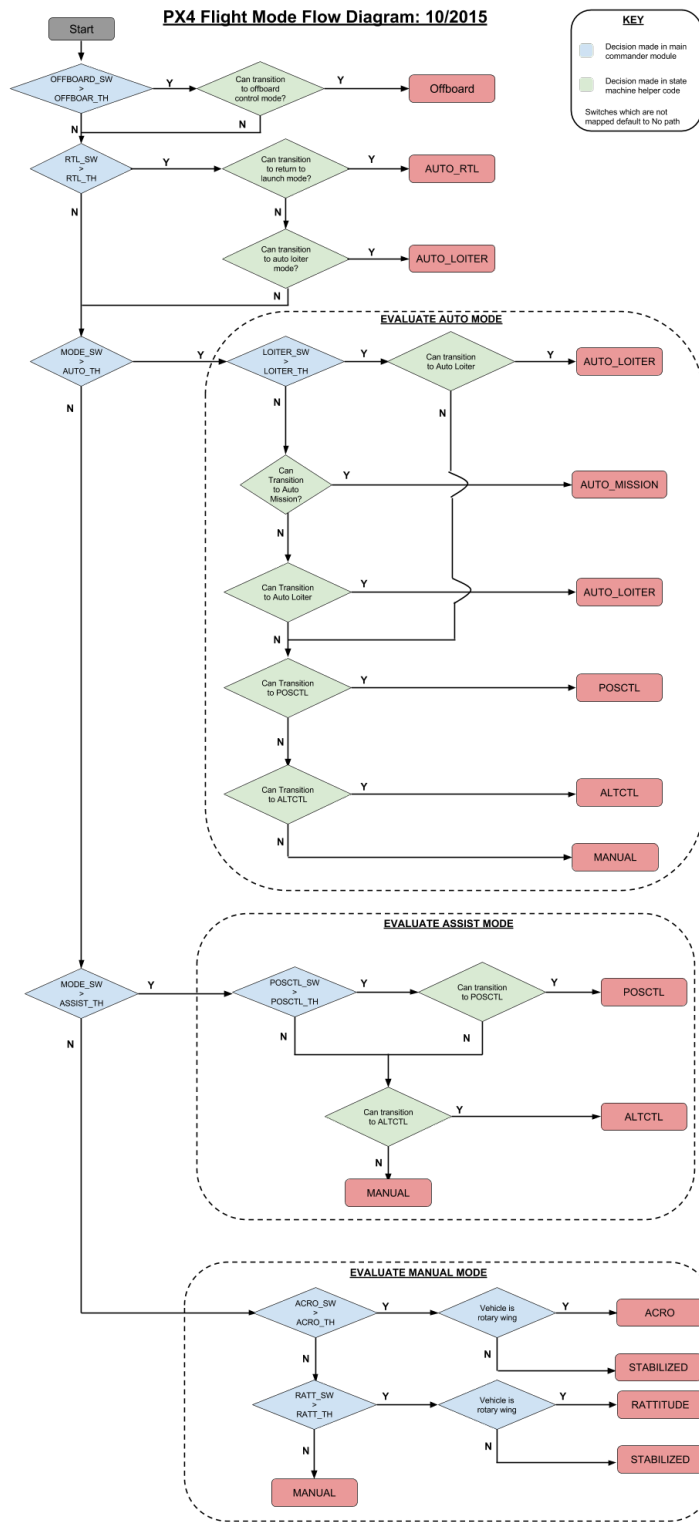


Figura 5.2: Cuadro de decisión para los modos de vuelo de PX4.

SIMULACIÓN

Finalmente, se deberá introducir la posibilidad de la simulación sin ejecutar el código *on-board*. Esta última opción, a nivel práctico, genera otro hardware compatible, el ordenador o terminal sobre el que se ejecuta la simulación. Generalmente, si se quisiera profundizar más, la arquitectura sería Linux, Mac o Windows sobre 64 bit, ya que esta combinación abarca casi cualquier ordenador comercial actual.

5.8. Desarrolladores

Github: herramienta para compartir y desarrollar archivos en línea de forma intuitiva. Junto con la extensión `.md` y las opciones de clonado permite una modificación intuitiva y cómoda de archivos. Su funcionamiento se basa en alojar proyectos y repositorios según el control de versiones Git¹.

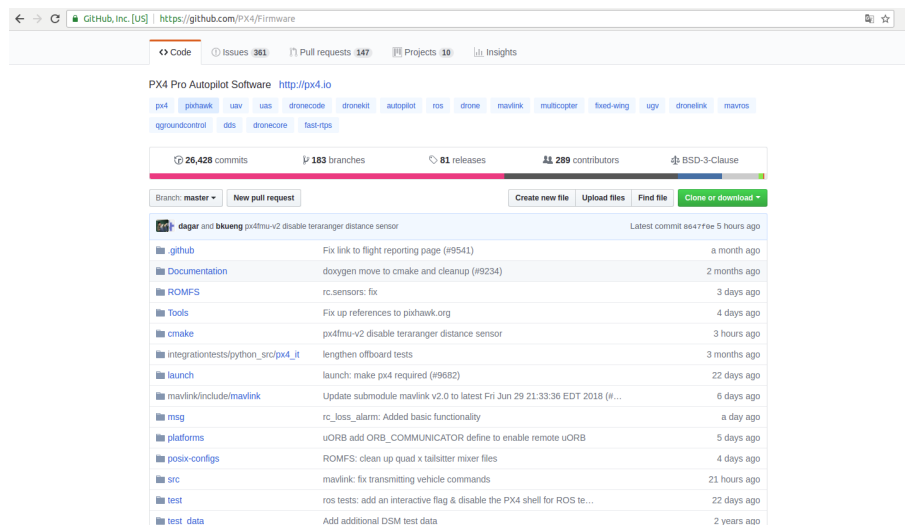


Figura 5.3: Repositorio PX4 sobre la plataforma GitHub.

Se incluye en toda la documentación disponible los siguientes elementos fundamentales en el desarrollo y aplicación del código:

¹Diseñado por Linus Torvalds, orientada a la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando tienen un gran número de archivos de código fuente. Por tanto, se basa en registrar los cambios en archivos y coordinar el trabajo sobre documentos compartidos.

- *User guide* o guía de usuario. Constituye todas las iniciaciones y tópicos interesantes para un usuario del código. [4]
- *Developer guide* o guía del desarrollador: constituye una herramienta fundamental para aquel desarrollador o miembro de la comunidad que desee modificar cualquier parte del código. También puede ser interesante desde el punto de vista académico para entender la estructura funcional de bloques, mediante el código propio del piloto automático. [3]
- *Firmware* o código operativo. Constituye la parte funcional previamente introducida. se engloban todas las librerías, códigos y configuraciones necesarias para compilar el código en cualquiera de los hardwares propuestos. [5]

La posibilidad de copiar y clonar directorios, así como subir resultados o cambio a los repositorios de GitHub de forma intuitiva, contribuye a este desarrollo de plataforma *open-source*. Sin embargo, como punto negativo o desventaja a considerar de este tipo de desarrollo se debería comentar la desorganización que puede generar. La gran comunidad que respalda y construye el código podría constituir un hándicap en el desarrollo del mismo sino se documentan todos los progresos debidamente. La organización interna de DroneCode, su supervisión, su respaldo empresarial y su tipo de licencia favorecen su control, frenan su desarrollo y permiten una documentación consistente con el código generado.

5.9. Arquitectura

En primer lugar, se estudia la versatilidad del código fundamentada en una capa *stack*, que se posiciona como la capa de más alto nivel. Esta última capa será la responsable del guiado y control de la aeronave, nutriendo los cálculos necesarios de las variables fundamentales de un UAV (velocidad, posición, orientación, rpm, etc). La capa de configuración, el middleware y los drivers o capas internas de los sensores serán aquellos intérpretes que transforman la información recopilada a las variables necesarias para el cálculo y control de las respuestas.

La arquitectura básica del código (figura 5.4) se ejemplifica en el siguiente esquema, donde se podrán inferir 5 grandes bloques:

- Almacenamiento: encargado de registrar toda la información relativa al vuelo. Bloque superior del diagrama estructural. La parte asociada a este bloque será en el caso propuesto la micro-SD o el MAVLink (envío externo de información).
- Comunicaciones: necesarias para comunicarse con el piloto de la aeronave así como para recopilar remotamente los datos relativos al vuelo. Bloque izquierdo del diagrama. Además de MAVlink, pueden haber otros tipos de protocolos de comunicación según el hardware disponible.

- Drivers: Parte constitutiva del middleware, se encargan de hacer posible la ejecución de una serie de instrucciones de alto nivel por parte del hardware del dispositivo a controlar. Bloque derecho del diagrama. Constituidos por todos los periféricos, sensores, actuadores o elementos que sean capaces de modificar, alterar o interactuar con los estados del sistema².
- Control: incluye algoritmos de guiado, control y procesamiento de los sensores instalados en los controladores. Constituye el *flight stack* propiamente dicho. Bloque inferior. Serán los archivos contenidos en src y que procesarán la información recogida de los drivers para alterar el estado del sistema y emitir señales de control.
- Bus³: envían y distribuyen las ordenes y la información. Bloque central. La parte del código asociada a los buses serán los protocolos internos de comunicación y los *uORB message*.

Este diagrama ejemplifica la interconexión de todos los elementos de la plataforma. En secciones anteriores se ha comentado la separación MAVLink de PX4, sin embargo, se observa que todas las iniciativas de piloto automático funcional están relacionadas.

Como es evidente, a pesar de que se adaptará el *flight stack* o el sistema responsable del control de la aeronave, todos los archivos a modificar no constituirán parte de este bloque según lo introducido. Formarán parte de la capa de Middleware, la conexión e interpretación del software sobre el hardware. Además de drivers sobre los sensores, también habrán modificaciones en el protocolo de comunicación y en los accesos a puerto series.

²En este diagrama, los drivers señalados son de carácter entrada. La drivers de salida o actuadores se sitúan en la región más baja del diagrama, una vez la información se ha procesado.

³Punto crítico en sistemas reactivos. Estos sistemas serán los que pueden trabajar con carga de trabajo variable, comunicación asíncrona y toda funcionalidad es dividida en elementos intercambiables y reutilizables. [3] y [9]

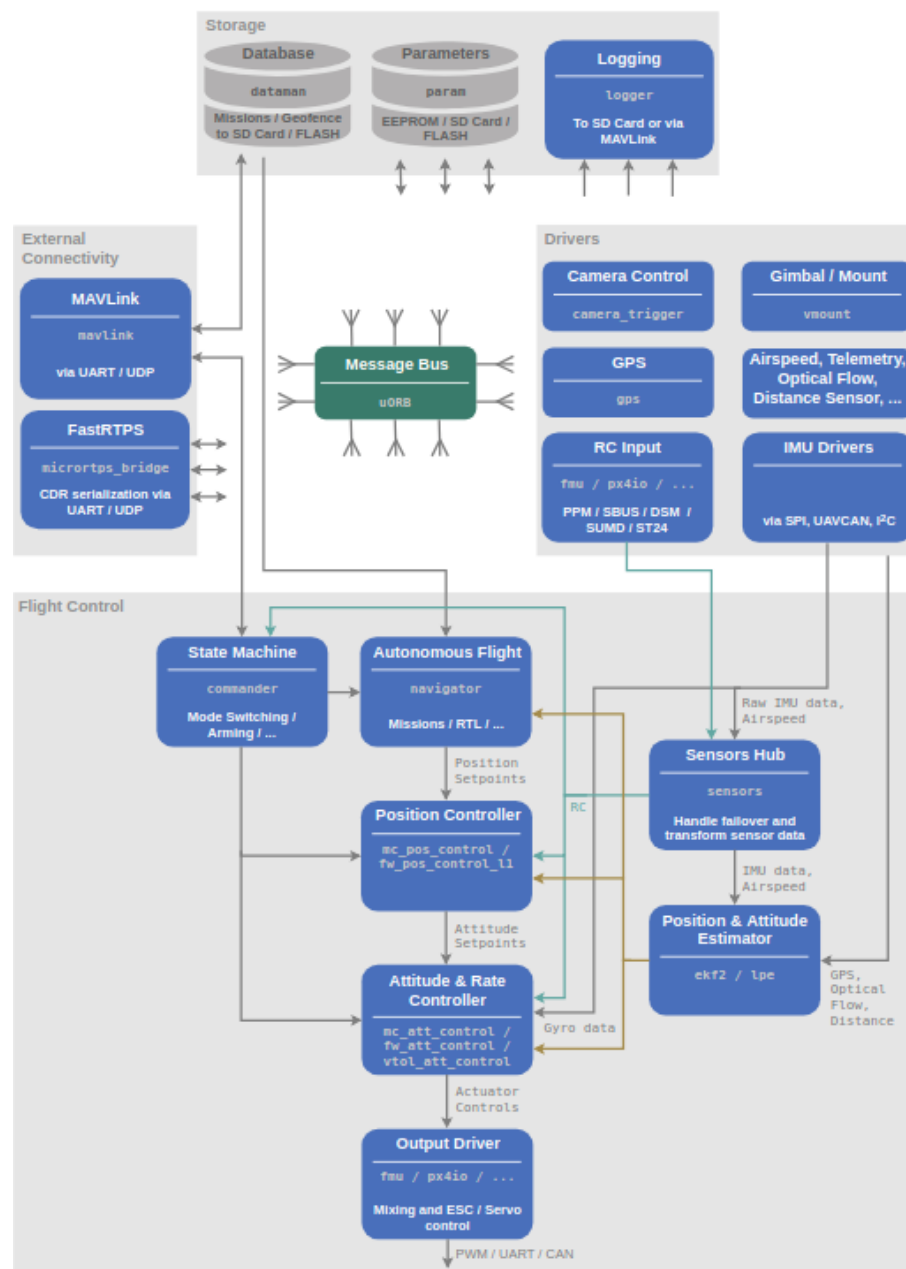


Figura 5.4: Diagrama de arquitectura PX4 [3].

5.10. Flight stack

En este apartado, se introducirá el procedimiento y algoritmo de control empleado por el código de PX4. En ambos casos, constituyen diagramas básicos de control y corrección de sistemas dinámicos. La profundización en estos conceptos de carácter aeronáutico no es objetivo fundamental de este documento, por tanto, solo se introducirán ambos diagramas.

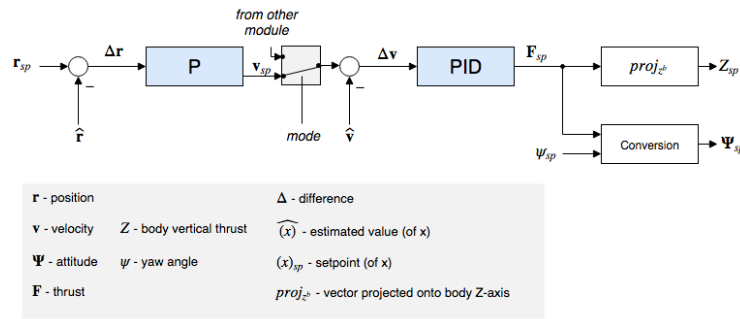


Figura 5.5: Control para multi-cópteros con nomenclatura PX4 [3].

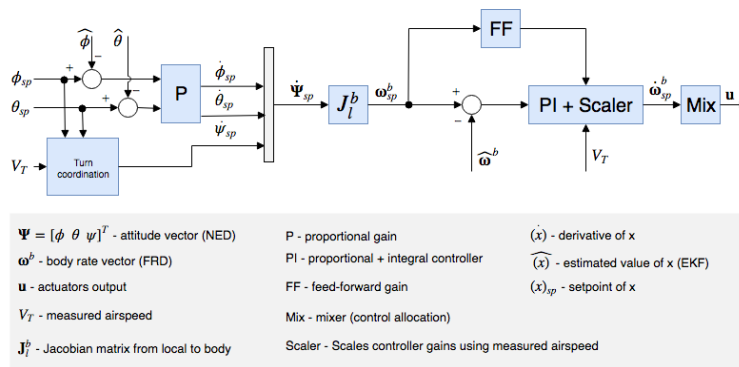


Figura 5.6: Control para aeronaves de ala fija con nomenclatura PX4 [3].

5.11. Posibilidades

Se han expuesto las principales características y arquitectura del código estudiado a lo largo de este documento. Los argumentos para utilizar este piloto automático son diversos: compatibilidad con gran variedad de SO, sensores disponibles del mercado, licencia libre y una gran comunidad de desarrolladores.

Además, un concepto no mencionado hasta el momento permitirá el desarrollo futuro y contribuirá en gran medida a la realización de este trabajo: el *DriverFramework* junto a la arquitectura modular⁴.

El **DriverFramework** constituye un marco global para generar drivers compatibles bajo las directrices POSIX. Desarrollado mediante una serie de archivos de publicación, configuración y gestión de la información para generar e intercambiar resultados de forma genérica. Esto permitirá que con cambios mínimos y adecuación de rutas, protocolos o conexiones locales, se pueda generar un driver plenamente operativo o adaptar un driver existente a otro tipo de protocolo por ejemplo.

⁴Más adelante se comentará la relación de los módulos con los archivos de código ya que aún no se ha establecido este paso como necesario

5.12. QGroundControl

Es una herramienta multisoporte, multiplataforma y multivehicular que permitirá el seguimiento, control y configuración de cualquier tipo de aeronave compatible. Constituirá la plataforma de control, guiado y tratamiento de información a niveles de navegación y conducción del vuelo.

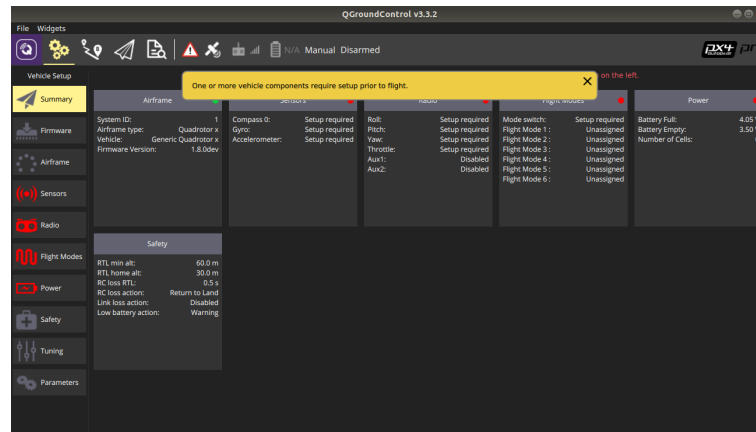


Figura 5.7: Pantalla principal de previsualización de QGroundControl.

Sus funcionalidades son diversas y destacarán:

- Configuración total prevuelo de aeronaves ejecutando PX4 Pro o ArduPilot.
- Soporte en vuelo para UAVs a través del protocolo MAVLink.
- Permite planear rutas para vuelo autónomo.
- Mapa interactivo con la información de posicionamiento, rastreo, puntos de referencia e instrumentación de vuelo.
- Vídeo en directo con los instrumentos sobrepuestos.
- Coordinación multivehicular y soporte multiplataforma.

Similar a otras opciones del mercado como *MissionPlanner* se enlazará al canal abierto por el código en ejecución a través del protocolo previamente comentado. Ofrece un guiado de las configuraciones previas y diversas ventanas de configuración (Setup, Settings), seguimiento (Fly) y planificación (Plan).

Plenamente integrado dentro de la iniciativa DroneCode constituyendo el programa responsable del control de vuelo. Está en constante desarrollo y crecimiento, documentando los progresos en su guía de usuario [6] y de desarrollo [7]

5.13. MAVlink

MAVLink o *Micro Air Vehicle Link* es un protocolo de comunicación para pequeños vehículos no tripulados (UAVs). Se utiliza fundamentalmente en conexiones entre estaciones de tierra y esta clase de aeronaves, siendo ampliamente utilizado por muchos proyectos para el envío y transmisión de coordenadas, velocidades y datos del vuelo.

Su funcionamiento básico se definirá según lo expuesto en la figura 5.8. Todos los paquetes de cabecera, pueden tomar valores de 0 a 255 y su función se detallará a continuación [8]:

- Inicio: marca el comienzo del paquete.
- Longitud: determina el número de bytes enviados como payload.
- Número de secuencia: muestra la numeración del paquete enviado para corroborar que no existen errores de conexión y para ordenar paquetes en envíos divididos.
- Identificación del sistema: permite diferenciar e identificar el sistema que ha enviado el paquete en una red, asociándole un número.
- Identificación del componente: marcará el componente concreto que ha enviado el mensaje dentro de un sistema.
- Identificación: indicará mediante un número que tipo de payload es, marcando cómo se debe leer y traducir la carga útil.

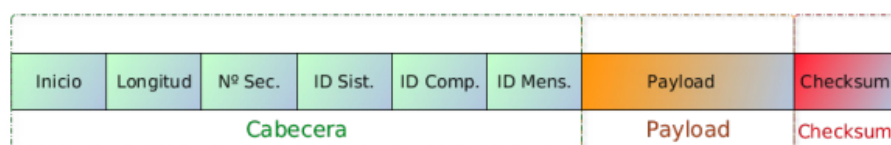


Figura 5.8: Estructura de envío de datos para el protocolo MAVLink [8]

La carga útil será la información a transmitir, habiendo sido marcado su tamaño por el segundo byte de la cabecera. Los mensajes tendrán formato xml y la información enviada se encontrará entre las etiquetas `<message>`/`</message>`. El envío concluirá con el control de checksum para detectar errores de envío.

Nota: el punto crítico es la vulnerabilidad estructural del protocolo que no ofrece ninguna confidencialidad, mecanismos de autenticación o seguridad.

Solución adoptada

Además de las características genéricas de código y hardware empleado, se deberá generar una analogía archivos-funcionalidad que permitirá al desarrollador trabajar en los archivos de interés.

6.1. Estructura genérica del código

Una vez descargado, clonado o consultado¹ el repositorio con el Firmware de PX4 se pueden destacar las siguientes carpetas y directorios de interés:

build: albergará los archivos generados una vez compilados. Como vemos en la figura 6.1 sus componentes principales serán el ejecutable PX4 y las librerías compartidas y asociadas a este archivo binario².

cmake: contendrá el archivo de generación que permitirá e iniciará la compilación de todo el código. Por otra parte, definirá variables internas y cargará todas las librerías y drivers asociados al hardware planteado. En este directorio se albergarán también algunos archivos o *toolchains* internos que se necesitan invocar para la compilación del código.

posix-configs: contendrá el archivo px4.config que abrirá, iniciará y configurará todos los sensores, puertos y drivers. No interviene en la compilación, pero debe ser coherente con las definiciones y elementos cargados en la compilación del software.

src: contendrá todos los archivos y librerías necesarias. Sus diferentes documentos se emplearán en función de las instrucciones de compilación, se buscará una compatibilidad con estándares como POSIX y condicionaran las opciones y archivos del código PX4 que se compilará.

¹Revisar manual de usuario

²El archivo necesita de estas librerías puesto que no se puede ejecutar en estático, es un archivo dinámico enlazado a estos archivos del directorio.

El resto de partes o directorios³ como són: **Documentation**, **launch**, **mavlink**, **msg**, **platforms**, **Test** y **Tools** contendrán archivos complementarios que no serán de interés ni necesario modificar para el correcto desarrollo de la interfaz *Middlewares*.

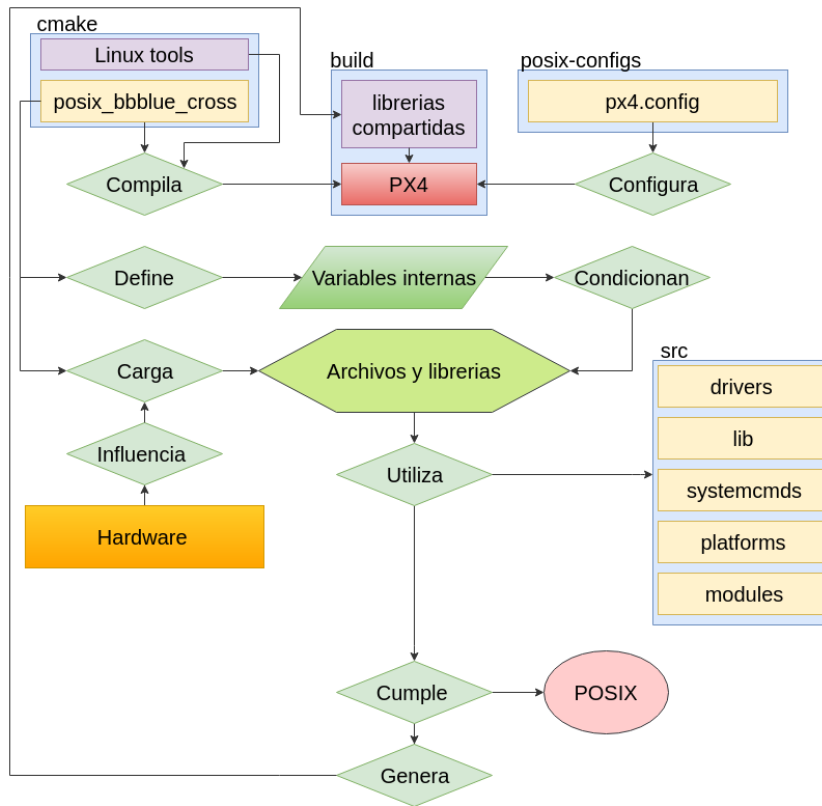


Figura 6.1: Detalle de los principales directorios y las funciones de cada archivo.

Una vez se ha comentado el diagrama y su importancia en la analogía archivos-funcionalidad, se deberá introducir un ejemplo de trabajo sencillo para poder trabajar y manejar esta estructura y jerarquía. Los elementos de configuración (posix-configs) y compilado (cmake) serán los archivos de control y carga de todos los elementos funcionales del código. Así, en estos documentos se deberá introducir correctamente los elementos disponibles en la plataforma a adaptar y todos los módulos o funcionalidades que requiera el usuario para el código funcional.

Dispuesto este lanzamiento, se deberá corroborar la existencia de los archivos objetivo, es decir, que dentro del abanico de elementos de hardware compatible existe aquel impuesto. En caso negativo, se deberá diseñar integralmente el mismo para operar y obtener la información que necesitará el código funcional. Si este existe, se deberá modificar para el caso tratado utilizando las variables y definiciones internas como elementos de control y decisión.

³La estructura puede estar ligeramente modificada en función de la versión descargada

6.2. Intervenciones adoptadas

Por tanto, se deberán resaltar todos las intervenciones aproximadas en cada uno de los directorios de interés:

- `cmake file`: cambios mínimos ya que sólo se necesitará particularizar al barómetro y unidad inercial propuesta, siendo los *systemcmds* o *modules* incluidos genéricos. Estos archivos sólo generarán funcionalidades y permitirán al código el cálculo o estimación de la posición por ejemplo, no están vinculados con el hardware del dispositivo y recibirán la información ya procesada.
- `posix-configs`: se expondrá la secuencia de inicio, rutas de acceso, orden y configuración de puertos. Tampoco sufrirá grandes modificaciones, una adaptación puntual a las características concreta de la plataforma.
- `src`: exigirá diversos cambios en una serie de librerías, tanto a nivel de accesos, protocolos de comunicación, rutas o validación de datos. Como aquello expuesto en el primer punto, sólo se modificarán los archivos relacionados con el hardware, ya que el resto de archivos quedarán intactos.

Llegados a este punto, ha quedado evidenciada la necesidad de una correcta y extensa revisión bibliográfica y documental previa. Se han revisado todos los conceptos genéricos, disposiciones de software y hardware necesarios para entender y justificar las modificaciones tanto en alcance como en localización. Por otra parte, el concepto introducido de `DriverFramework` ayudará a las intervenciones oportunas debido a sus características, desarrollo y archivos disponibles.

A continuación, se expondrán una a una las intervenciones realizadas en orden cronológico más que por importancia, nivel o jerarquía.

Mapeado de la BeagleBone Blue

Archivo: board_config.h

Ruta: Firmware/src/drivers/boards/bbblue/

```

/**
 * @file board_config.h
 *
 * BBBLUE internal definitions
 */

#pragma once

#define BOARD_OVERRIDE_UUID "BBBLUE " // must be of length 12
    (PX4_CPU_UUID_BYTE_LENGTH)
#define BOARD_OVERRIDE_MFGUID BOARD_OVERRIDE_UUID

#define BOARD_NAME "BBBLUE"

#define ADC_BATTERY_VOLTAGE_CHANNEL 2
#define ADC_BATTERY_CURRENT_CHANNEL 3

#define BOARD_BATTERY1_V_DIV (10.177939394f)
#define BOARD_BATTERY1_A_PER_V (15.391030303f)

#define BOARD_MAX_LEDS 1 // Number external of LED's this board has

/*
 * I2C busses
 */
#define PX4_I2C_BUS_EXPANSION 2
#define PX4_NUMBER_I2C_BUSES 1

#include <system_config.h>
#include "../common/board_common.h"

```

En este caso, se redefine la línea de alimentación, nombre genérico y se mapea el bus de tipo I2C. Configuración bastante genérica la que aporta este archivo, similar a otras arquitecturas Linux como la Raspberry Pi 2 con Navio 2.

Archivo de compilación

Archivo principal de compilación, que incluirá y enlazará todo lo necesario para generar un código funcional y operativo.

Archivo: posix_bbblue_common

Ruta: Firmware/cmake/common/

En este caso, la función de este archivo y de las modificaciones oportunas serán las siguientes:

- Añadir las definiciones propias de Linux, del DriverFramework y de la BeagleBone Blue, que se utilizarán para jerarquizar e incluir archivos o variables según convenga.

```
add_definitions(  
-D__PX4_POSIX_BBBLUE  
-D__DF_LINUX # For DriverFramework  
-D__DF_BBBLUE # For DriverFramework  
)
```

- Incluir los módulos y plataformas genéricas.

```
set(config_module_list  
# Board support modules  
#  
#drivers/barometer  
drivers/batt_smbus  
drivers/differential_pressure  
drivers/distance_sensor  
#drivers/telemetry  
  
modules/sensors  
  
#platforms/posix/drivers/df_hmc5883_wrapper  
#platforms/posix/drivers/df_isl29501_wrapper  
#platforms/posix/drivers/df_lsm9ds1_wrapper  
platforms/posix/drivers/df_bmp280_wrapper  
platforms/posix/drivers/df_mpu9250_wrapper  
#platforms/posix/drivers/df_trone_wrapper  
  
# System commands  
#  
systemcmds/param  
systemcmds/led_control  
systemcmds/mixer
```

```
systemcmds/ver
systemcmds/esc_calib
systemcmds/reboot
systemcmds/topic_listener
systemcmds/tune_control
systemcmds/perf

# Estimation modules
#
modules/attitude_estimator_q
modules/position_estimator_inav
modules/local_position_estimator
modules/landing_target_estimator
modules/ekf2

# Vehicle Control
#
modules/fw_att_control
modules/fw_pos_control_l1
modules/gnd_att_control
modules/gnd_pos_control
modules/mc_att_control
modules/mc_pos_control
modules/vtol_att_control

# Library modules
#
modules/sdlog2
modules/logger
modules/commander
modules/dataman
modules/land_detector
modules/navigator
modules/mavlink

# PX4 drivers
#
drivers/linux_sbus
drivers/gps
#drivers/bbblue_adc
#drivers/navio_sysfs_rc_in
drivers/linux_gpio
drivers/linux_pwm_out
#drivers/navio_rgblcd
drivers/pwm_out_sim
#drivers/rpi_rc_in
)
```

- Añadir los drivers específicos de la BeagleBone Blue.

```
#  
# DriverFramework driver  
#  
set(config_df_driver_list  
  bmp280  
  mpu9250  
)
```

El archivo denominado “common” será compartido por dos compilados distintos como serían el *cross* y el *native*. Para la BeagleBone Blue solo se ha contemplado la posibilidad de compilar en un ordenador base y exportar el código, es decir, el compilado denominado *cross* según la nomenclatura de PX4.

Por tanto, se tendrá un último archivo responsable de lanzar y llamar al pre-vio cuando se produzca la compilación, dispone las siguientes líneas y no se verá modificado respecto de otras arquitecturas Linux:

Archivo: posix_bbblue_cross

Ruta: Firmware/cmake/common/

```
include(configs/posix_bbblue_common)  
SET(CMAKE_TOOLCHAIN_FILE  
  ${PX4_SOURCE_DIR}/cmake/toolchains/Toolchain-arm-linux-gnueabihf.cmake)
```

Configuración

Archivo:px4.config

Ruta: Firmware/posix-configs/bbblue

En este caso, la configuración del código apenas se ve modificada. Se incluyen e inician los drivers de forma coherente con los incluidos en el archivo del directorio cmake. Por otra parte, se definen los puertos de comunicación del canal mavlink. Se utilizan dos de los puertos serie presentes en la BeagleBone Blue, aunque se podría utilizar cualquiera de los disponibles.

```
# bbblue config for a quad
uorb start
param load
param set SYS_AUTOSTART 4001
param set MAV_BROADCAST 1
param set MAV_TYPE 2
param set SYS_MC_EST_GROUP 2
param set BAT_CNT_V_VOLT 0.001
param set BAT_V_DIV 10.9176300578
param set BAT_CNT_V_CURR 0.001
param set BAT_A_PER_V 15.391030303
dataman start
#df_lsm9ds1_wrapper start -R 4
df_bmp280_wrapper start
df_mpu9250_wrapper start -R 10
#df_hmc5883_wrapper start
#df_ms5611_wrapper start
#gps start -d /dev/spidev0.0 -i spi -p ubx
sensors start
commander start
navigator start
ekf2 start
land_detector start multicopter
mc_pos_control start
mc_att_control start
mavlink start -x -u 14556 -r 1000000
mavlink stream -u 14556 -s HIGHRES_IMU -r 50
mavlink stream -u 14556 -s ATTITUDE -r 50
mavlink start -x -d /dev/ttyS3
mavlink stream -d /dev/ttyS3 -s HIGHRES_IMU -r 50
mavlink start -x -d /dev/ttyS2
mavlink stream -d /dev/ttyS2 -s ATTITUDE -r 50
logger start -t -b 200
mavlink boot_complete
```

Drivers

La modificación de los drivers exige pequeños cambios en un amplio número de archivos. Por otra parte, se deberá comentar en primer lugar ciertas particularidades sobre la forma de trabajar del DriverFramework.

El Driver Framework aporta ciertos modelos y estructuras de trabajo, para iniciar y gestionar objetos e instancias coherentes con el marco de instrucciones POSIX. Algunos ejemplos serán los archivos I2CDevObj, SPIDevObj o DevMgr.

Dichos archivos no se verán alterados, sin embargo se deberá conocer su arquitectura interna y la forma de generar objetos para poder interactuar con los mismos. Este marco genérico donde incluir los drivers se ha podido comprobar que es muy útil debido a:

- Inclusión de muchos elementos comunes y genéricos del mercado.
- Utilización de los protocolos de comunicación mas utilizados.
- Sometido a las directrices POSIX.
- Proceso secuencial de carga de los elementos. Generación de errores y advertencias secuenciales que permiten lidiar con los problemas.

Como se podría intuir con algunos ajustes y cambios, se puede desarrollar un driver plenamente funcional, siendo cuidadoso con la interacción de los registros, las conexiones y las rutas.

BMP280

Para este caso, el driver presente en PX4 conectará el barómetro según el protocolo I2C. Sin embargo, el mapeado según los archivos presentes en /dev/ hará que se necesite una modificación de la ruta.

```
#if defined(__DF_BBBLUE)
#define BARO_DEVICE_PATH "/dev/i2c-2"
#else
#define BARO_DEVICE_PATH "/dev/iic-3"
#endif
```

Este cambio contempla la nueva interacción con el *device* según esta mapeado en el sistema operativo presente en la BeagleBone. El proceso quedará secuenciado y especificado en el archivo mencionado a continuación.

Archivo: df_bmp280_wrapper

Ruta: Firmware/src/platforms/posix/drivers/

Finalmente, aunque no se modificará ningún aspecto de este código se comprobarán y ratificarán los registros de configuración con la datasheet, ArduPilot([15]) o Robotics Cape⁴ para asegurar su correcto funcionamiento. Estos registros se alojan en dos ficheros, los cuales disponen el siguiente contenido:

Archivo: bmp280.cpp

Ruta: Firmware/src/lib/DriverFramework/drivers/bmp280

```
#define BMP280_REG_ID 0xD0
#define BMP280_REG_CTRL_MEAS 0xF4
#define BMP280_REG_CONFIG 0xF5
#define BMP280_REG_PRESS_MSB 0xF7
#define BMP280_ID 0x58

#define BMP280_BITS_CTRL_MEAS_OVERSAMPLING_TEMP2X 0b01000000
#define BMP280_BITS_CTRL_MEAS_OVERSAMPLING_PRESSURE8X 0b00010000
#define BMP280_BITS_CTRL_MEAS_POWER_MODE_NORMAL 0b00000011

#define BMP280_BITS_CONFIG_STANDBY_OMS5 0b00000000
#define BMP280_BITS_CONFIG_FILTER_OFF 0b00000000
#define BMP280_BITS_CONFIG_SPI_OFF 0b00000000
```

Archivo: bmp280.hpp

Ruta: Firmware/src/lib/DriverFramework/drivers/bmp280

```
// update frequency is 50 Hz (44.4-51.3Hz ) at 8x oversampling
#define BMP280_MEASURE_INTERVAL_US 20000

#define BMP280_BUS_FREQUENCY_IN_KHZ 400
#define BMP280_TRANSFER_TIMEOUT_IN_USECS 9000
#define BMP280_MAX_LEN_SENSOR_DATA_BUFFER_IN_BYTES 6
#define BMP280_MAX_LEN_CALIB_VALUES 26

// TODO: include some common header file (currently in drv_sensor.h).
#define DRV_DF_DEVTYPE_BMP280 0x42

#define BMP280_SLAVE_ADDRESS 0b1110110 /* 7-bit slave address */
```

⁴Capa de software desarrollada por Strawson Design con librerías, ejemplos y tests para el proyecto Robot Control. Ha crecido para incluir librerías matemáticas para respuestas en tiempo discreto o funciones (compatibles con POSIX) para tiempos, hilos, flujo de programas y mucho más, todo desarrollado para el control robótica en sistemas embarcados. [14]

IMU MPU9250

Los cambios a implementar en la IMU serán de un carácter mayor. En primer lugar, la MPU9250 esta presente en el listado de elementos compatibles con el *PX4 flight stack*. Sin embargo, la conexión realizada en el repositorio de GitHub será utilizando el protocolo SPI. En el hardware propuesto la conexión sera de tipo circuito interintegrado (I2C), exigiendo una serie de modificaciones.

SPIDevObj vs I2CDevObj

La ventaja propuesta se fundamenta en las posibilidades del DriverFramework, donde ambos protocolos de comunicación se contemplan como viables. Las diferencias fundamentales entre los dos archivos que generan los objetos para interaccionar con cada protocolo se resumirán en la siguiente lista:

- Función escritura: cambio apreciable del número de variables. En el caso de I2C se incluye una variable longitud, necesario para controlar tiempos y longitudes de envío.
 - **I2C**: `_writeReg(uint8_t address, uint8_t *in_buffer, size_t length)`
 - **SPI** `_writeReg(uint8_t address, uint8_t val)`
- Función lectura: cambios similares a la función de escritura
 - **I2C**: `_readReg(uint8_t address, uint8_t *out_buffer, size_t length)`
 - **SPI** `_readReg(uint8_t address, uint8_t &val)`
- Función modificar: no se realizará la comparación puesto que solo existe para el caso SPI.
- Configuración: mayor diferencia, para SPI solo se especifica una frecuencia, sin embargo, la configuración esclavo de I2C exige un registro, una frecuencia y un tiempo de *timeout*.
 - **I2C**: `_setSlaveConfig(uint32_t slave_address, uint32_t bus_frequency_khz, uint32_t transfer_timeout_usec)`
 - **SPI**: `_setBusFrequency(uint32_t freq_hz)`

Definición de rutas

En el código se contempla la posibilidad de enlazar una unidad inercial por I2C como vemos en el siguiente extracto:

Archivo: `ImuSensor.hpp`

Ruta: `Firmware/src/lib/DriverFramework/framework/include`

```

    ImuSensor(const char *device_path, unsigned int sample_interval_usec,
              bool mag_enabled = false) :
#if defined(__IMU_USE_I2C)
I2CDevObj("ImuSensor", device_path, IMU_CLASS_PATH, sample_interval_usec),
#else
SPIDevObj("ImuSensor", device_path, IMU_CLASS_PATH, sample_interval_usec),
#endif

```

Se puede observar, que se deberá definir una variable que controla el tipo de protocolo utilizado `__IMU_USE_I2C`. Se empleará el siguiente código para definir dicha variable de control, así como una serie de rutas e incluir los archivos necesarios:

```

#if defined(__DF_BBBLUE)
#define __IMU_USE_I2C
#include "I2CDevObj.hpp"
#endif

#if defined(__IMU_USE_I2C)
#include "I2CDevObj.hpp"
#else
#include "SPIDevObj.hpp"
#endif

#if defined(__DF_QURT)
#include "dev_fs_lib_spi.h"
#define IMU_DEVICE_PATH "/dev/spi-1"
#elif defined(__DF_BEBOP)
#define IMU_DEVICE_PATH "/dev/i2c-mpu6050"
#elif defined(__DF_RPI)
#define IMU_DEVICE_PATH "/dev/spidev0.1"
#elif defined(__DF_EDISON)
#define IMU_DEVICE_PATH "/dev/spidev5.1"
#elif defined(__DF_OCPOC)
#define IMU_DEVICE_PATH "/dev/spidev1.0"
#elif defined(__DF_BBBLUE)
#define IMU_DEVICE_PATH "/dev/i2c-2"
#else
#define IMU_DEVICE_PATH "/dev/spidev0.0"
#endif

```

```
#if defined(__DF_RPI)
#include <linux/spi/spidev.h>
#define IMU_DEVICE_ACC_GYRO "/dev/spidev0.3"
#define IMU_DEVICE_MAG "/dev/spidev0.2"
#elif defined(__DF_RPI_SINGLE)
#define IMU_DEVICE_ACC_GYRO "/dev/spidev0.1"
#define IMU_DEVICE_MAG "/dev/spidev0.1"
#else
#define IMU_DEVICE_ACC_GYRO ""
#define IMU_DEVICE_MAG ""
#endif

#define IMU_CLASS_PATH "/dev/imu"
```

Archivos principales

Se ha definido una conexión I2C y se ha generado dicho objeto para la unidad inercial. Sin embargo, la conexión y escritura del modelo de MPU propuesto esta remitida al protocolo SPI. Las modificaciones se fundamentarán en adaptar un intercambio de información SPI a I2C de forma consistente con los archivos introducidos previamente.

Archivo: mpu9250.cpp

Ruta: Firmware/src/lib/DriverFramework/drivers/mpu9250

En el archivo referente a la conexión del magnetómetro no se realizarán grandes intervenciones ya que por defecto esta conexión si que esta realizada mediante I2C.

Archivo: mpu9250_mag.cpp

Ruta: Firmware/src/lib/DriverFramework/drivers/mpu9250

Nota: Algunas pequeñas modificaciones se han realizado debido a errores de “violación de acceso” o *smash stacked*^a que han requerido modificar opciones de compilación (ver sección siguiente o manual de programación) o recombinar funciones secundarias en aquella principal para evitar errores en memoria.

^aError debido a un posible desbordamiento del buffer. Esta protección que aborta la ejecución del código cuando se detecta un posible *overflow* es una opción por defecto en compiladores actuales. Necesita un cambio en las banderas de compilación o una configuración del compilador nativo diferente para subsanarse.

Este hecho ha llevado a suprimir la función de `detect()` del magnetómetro e integrarla directamente en el código genérico y principal por ejemplo. Este cambio se debe a los conflictos que generaba en la memoria el retorno de algunas variables de funciones secundarias. Además en este archivo se habrán modificado los registros conforme a las funciones definidas en el archivo:

Archivo: `mpu9250.hpp`

Ruta: `Firmware/src/lib/DriverFramework/drivers/mpu9250`

Nota: debido a su longitud este archivo se encuentra contenido en el manual de programación anexo a esta memoria principal.

Por otra parte, como se ha comentado en la conexión del barómetro, estos archivos modificados son fundamentalmente aquellos que inician, cargan y construyen la comunicación. Las directrices POSIX y el intercambio posterior de información vienen regulados según el siguiente documento, del cual no se incluirá el código, puesto que no ha sufrido modificaciones:

Archivo: `df_mpu9250_wrapper`

Ruta: `Firmware/src/platforms/posix/drivers/`

Data Validation

Las modificaciones finales no constituyen ningún cambio en rutas, accesos o protocolos. Llegados a este punto, se obtienen errores en la consistencia de los datos. Evidentemente, estos se solucionarán en la etapa de programación aunque no serán visibles hasta la etapa de compilación y tests.

Accelerómetro

[sensors] **Accel #0: TIMEOUT**: error debido al tiempo de actualización y lectura del acelerómetro. Debido al conflicto en los datos se generará un error⁵ denominado TIMEOUT. El siguiente fragmento del archivo expuesto recoge esta funcionalidad:

Archivo: voted_sensors_update.cpp

Ruta: Firmware/src/modules/sensors

```
if (failover_index != -1) {
mavlink_log_emergency(&_mavlink_log_pub, "%s #i fail: %s%s%s%s!",
sensor_name,
failover_index,
((flags & DataValidator::ERROR_FLAG_NO_DATA) ? " OFF" : ""),
((flags & DataValidator::ERROR_FLAG_STALE_DATA) ? " STALE" : ""),
((flags & DataValidator::ERROR_FLAG_TIMEOUT) ? " TIMEOUT" : ""),
((flags & DataValidator::ERROR_FLAG_HIGH_ERRCOUNT) ? " ERR CNT" : ""),
((flags & DataValidator::ERROR_FLAG_HIGH_ERRDENSITY) ? " ERR DNST" : ""));
```

La activación de las banderas y en consecuencia el error en consola de los archivos relativos a “sensors”, se activarán en el siguiente fragmento:

Archivo: data_validator.cpp

Ruta: Firmware/src/lib/ecl/validation

⁵Se hará saltar una bandera de aviso

```
float
DataValidator::confidence(uint64_t timestamp)
{
float ret = 1.0f;

/* check if we have any data */
if (_time_last == 0) {
_error_mask |= ERROR_FLAG_NO_DATA;
ret = 0.0f;

} else if (timestamp - _time_last > _timeout_interval) {
/* timed out - that's it */
_error_mask |= ERROR_FLAG_TIMEOUT;
ret = 0.0f;

} else if (_value_equal_count > _value_equal_count_threshold) {
/* we got the exact same sensor value N times in a row */
_error_mask |= ERROR_FLAG_STALE_DATA;
ret = 0.0f;

} else if (_error_count > NORETURN_ERRCOUNT) {
/* check error count limit */
_error_mask |= ERROR_FLAG_HIGH_ERRCOUNT;
ret = 0.0f;

} else if (_error_density > ERROR_DENSITY_WINDOW) {
/* cap error density counter at window size */
_error_mask |= ERROR_FLAG_HIGH_ERRDENSITY;
_error_density = ERROR_DENSITY_WINDOW;

}

/* no critical errors */
if (ret > 0.0f) {
/* return local error density for last N measurements */
ret = 1.0f - (_error_density / ERROR_DENSITY_WINDOW);

if (ret > 0.0f) {
_error_mask = ERROR_FLAG_NO_ERROR;
}
}

return ret;
}
```

La solución a este error se realizará realizando dos ajustes:

Elevar la velocidad de procesamiento: siguiendo las instrucciones recogidas en <https://github.com/mirkix/ardupilotblue>.

1. Actualización de software: `sudo apt update && sudo apt upgrade -y`
2. Instalar software: `sudo apt install -y bb-cape-overlays cpufrequtils`
3. Fijar frecuencia a 1GHz: `sudo sed -i 's/GOVERNOR=.ndemand/GOVERNOR=performance/g' /etc/init.d/cpufrequtils`
4. Actualizar archivos: `cd /opt/scripts && sudo git pull`
5. Maximizar la partición de la tarjeta micro-SD:
`sudo /opt/scripts/tools/grow_partition.sh`
6. Instalar RT Kernel 4.9: `sudo /opt/scripts/tools/update_kernel.sh -ti-rt-channel -lts-4_9`
7. Especificar el directorio binario usado en arranque:
`sudo sed -i 's/#dtb=/dtb=am335x-boneblue.dtb/g' /boot/uEnv.txt`
8. Reiniciar el sistema: `sudo reboot`

Modificaciones en el validador de datos: debido a que se trata de un código reactivo, aumentar la frecuencia de procesamiento puede no ser suficiente para evitar el conflicto en la validación de datos. Se deberán modificar los *thresholds* o límites de activación, para este sensor y error concreto, elevando el `_timeout_interval` de 20000 a 30000.

Archivo: `voted_sensors_update.cpp`

Ruta: `Firmware/src/modules/sensors`

```
private:
uint32_t _error_mask{ERROR_FLAG_NO_ERROR}; /**< sensor error state */

uint32_t _timeout_interval{30000}; /**< interval in which the datastream
    times out in us*/

uint64_t _time_last{0};          /**< last timestamp */
uint64_t _event_count{0};       /**< total data counter */
uint64_t _error_count{0};       /**< error count */

int _error_density{0};          /**< ratio between successful reads and
    errors */

int _priority{0};               /**< sensor nominal priority */
```

```
float _mean[dimensions] {};      /**< mean of value */
float _lp[dimensions] {};      /**< low pass value */
float _M2[dimensions] {};      /**< RMS component value */
float _rms[dimensions] {};     /**< root mean square error */
float _value[dimensions] {};   /**< last value */
float _vibe[dimensions] {};    /**< vibration level, in sensor unit */

unsigned _value_equal_count{0};  /**< equal values in a row */
unsigned _value_equal_count_threshold{VALUE_EQUAL_COUNT_DEFAULT}; /**<
    when to consider an equal count as a problem */

DataValidator *_sibling{nullptr}; /**< sibling in the group */

static const constexpr unsigned NORETURN_ERRRCOUNT = 100000; /**< if the
    error count reaches this value, return sensor as invalid 10000*/
static const constexpr float ERROR_DENSITY_WINDOW = 100.0f; /**< window
    in measurement counts for errors */
static const constexpr unsigned VALUE_EQUAL_COUNT_DEFAULT = 50000; /**<
    if the sensor value is the same (accumulated also between axes) this
    many times, flag it 100*/

/* we don't want this class to be copied */
DataValidator(const DataValidator &) = delete;
DataValidator operator=(const DataValidator &) = delete;
};
```

Resultados y test

En primer lugar, una vez modificado los archivos necesarios para poder ejecutar el código PX4 sobre el hardware propuesto, se deberá compilar y exportar hacia la BeagleBone Blue.

7.1. Compilado

Como se ha comentado anteriormente, el archivo principal de ejecución y cargado de librerías será `posix_bbblue_cross`. La secuencia de compilación sera la siguiente:

```
cd <Directorio PX4>/Firmware
make posix_bbblue_cross
```

En algún caso será necesario incluir la siguiente opción de compilación `CXXFLAGS='-fno-stack-protector'`. Este comando desactiva la protección de *overflow del buffer* presente en algunas configuraciones y versiones del compilador nativo de Linux (GCC). Esta protección activa puede generar el error **stack smashed detected** previamente comentado y abortar la ejecución del código.

En este caso los comandos de compilación serian:

```
cd <Directorio PX4>/Firmware
make CXXFLAGS='-fno-stack-protector' posix_bbblue_cross
```

La secuencia de compilación se iniciará con las siguientes líneas:

```
-- PX4 VERSION: v1.8.0-99-g5cb675357
-- CONFIG: posix_bbblue_cross
-- Build Type: RelWithDebInfo
-- The CXX compiler identification is GNU 7.2.0
-- The C compiler identification is GNU 7.2.0
-- The ASM compiler identification is GNU
-- Found assembler: /usr/bin/arm-linux-gnueabi-gcc
-- Check for working CXX compiler: /usr/bin/arm-linux-gnueabi-g++
-- Check for working CXX compiler: /usr/bin/arm-linux-gnueabi-g++ --
  works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Check for working C compiler: /usr/bin/arm-linux-gnueabi-gcc
-- Check for working C compiler: /usr/bin/arm-linux-gnueabi-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- ccache enabled (export CCACHE_DISABLE=1 to disable)
-- Found PythonInterp: /usr/bin/python (found version "2.7.14")
-- Found PY_jinja2: /usr/lib/python2.7/dist-packages/jinja2
-- C compiler: arm-linux-gnueabi-gcc (Ubuntu/Linaro 7.2.0-6ubuntu1)
  7.2.0
-- C++ compiler: arm-linux-gnueabi-g++ (Ubuntu/Linaro 7.2.0-6ubuntu1)
  7.2.0
DF Drivers: bmp280;mpu9250
Adding DF driver: bmp280
Adding DF driver: mpu9250
-- PX4 ECL: Very lightweight Estimation & Control Library
  v0.9.0-561-gb26c2d6
-- Configuring done
-- Generating done
-- Build files have been written to:
  /home/miguel/Escritorio/Firmware/build/posix_bbblue_cross
```

7.2. Exportar PX4

La versatilidad y posibilidades de la BeagleBone Blue permiten el envío y subida de archivos de diversas formas. No obstante, se empleará el IDE nativo denominado Cloud9 para la gestión, envío y ejecución de archivos.

Fundamentalmente, se trata de un IDE en línea de carácter libre. Soporta los lenguajes de programación más populares del mercado y será accesible de forma nativa desde la BeagleBone Blue. Una vez cableada, se lanzará el entorno accediendo a la dirección 192.168.7.2:3000 del navegador web¹. Se trata de un entorno de desarrollo estándar y en este documento sólo se introducirán los pasos imprescindibles para exportar los archivos necesarios. Otras funcionalidades y opciones se pueden consultar en el manual de usuario anexo a este documento.

La característica introducida en el capítulo anterior de *cross compiling* genera la necesidad de exportar desde el ordenador base los archivos hasta el hardware objetivo².

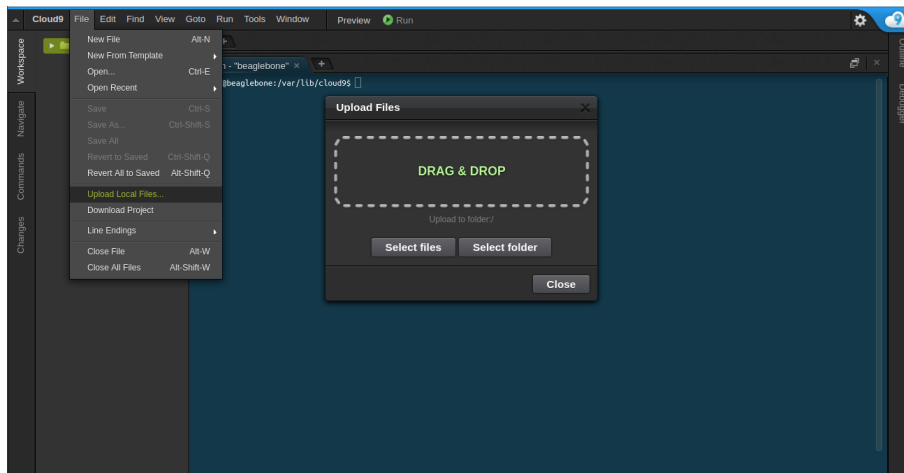


Figura 7.2: Detalle sobre la subida y exportación de archivos mediante el IDE Cloud9.

La subida de archivos se realiza como podemos ver en la figura 7.2 mediante el comando *Upload local files* ubicado en el menú *File*. Será necesario exportar los siguientes documentos:

- Carpeta `posix_bbblue_cross`. Contiene el ejecutable y todas las librerías compartidas para ejecutarse de forma dinámica .
- Archivo de configuración del código: `posix-configs/bbblue/px4.config`. Configura e inicializa los distintos drivers y módulos del código.

¹Google Chrome o Mozilla Firefox.

²La posibilidad de compilar de forma nativa en la BeagleBone Blue no se ha contemplado, aunque es una alternativa viable en otros hardwares compatibles con PX4.

7.3. Resultados

Exportados todos los archivos necesarios al hardware de la BeagleBone, se deberá ejecutar el código. Primero, se cambiarán los privilegios del archivo para evitar conflictos de lectura o escritura. Se utilizará la secuencia:

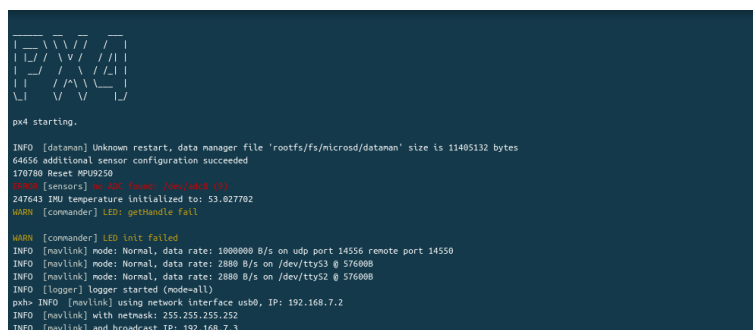
```
cd posix_bbblue_cross/  
sudo chmod +x px4
```

Nota: En una imagen de Debian por defecto, la contraseña para el comando `sudo` será `tempus`, acrónimo abreviado de *temporal password*.

Finalmente, la ejecución del archivo binario se realizará mediante la siguiente instrucción:

```
sudo ./px4 bbblue/px4.config
```

Al ejecutar el código se obtiene el siguiente resultado por consola:



```
px4 starting.  
INFO [dataman] Unknown restart, data manager file 'rootfs/fs/microsd/dataman' size is 11405132 bytes  
44656 additional sensor configuration succeeded  
170780 Reset MPU9250  
WARN [sensors] IMU: (warn: exceeded ts)  
247643 IMU temperature initialized to: 53.027702  
WARN [commander] LED: gethandle fail  
  
WARN [commander] LED init failed  
INFO [mavlink] mode: Normal, data rate: 1000000 B/s on udp port 14556 remote port 14550  
INFO [mavlink] mode: Normal, data rate: 2888 B/s on /dev/ttyS3 @ 576000  
INFO [mavlink] mode: Normal, data rate: 2888 B/s on /dev/ttyS2 @ 576000  
INFO [logger] logger started (mode=11)  
px4b INFO [mavlink] using network interface usb0, IP: 192.168.7.2  
INFO [mavlink] with netmask: 255.255.255.252  
INFO [mavlink] and broadcast IP: 192.168.7.3
```

Figura 7.3: Resultado de ejecución del PX4 sobre la BeagleBone Blue.

Como se observa en la figura 7.3, se generan errores debido al ADC o al driver responsable de los LEDs. En este caso, no era objetivo fundamental del trabajo el desarrollo del Middleware completo con gestión de información a través de los LEDs o la realización de un driver específico para el control del ADC.

Por otra parte, se evidencian los puertos y conexiones generadas a través de los cuales se podrá interactuar con el vehículo, para enviar instrucciones o recibir datos sobre navegación y posicionamiento. Finalmente, el error relativo al magnetómetro es aquel introducido anteriormente que no aborta o impide la ejecución ni el inicio del magnetómetro, originándose al validar los datos.

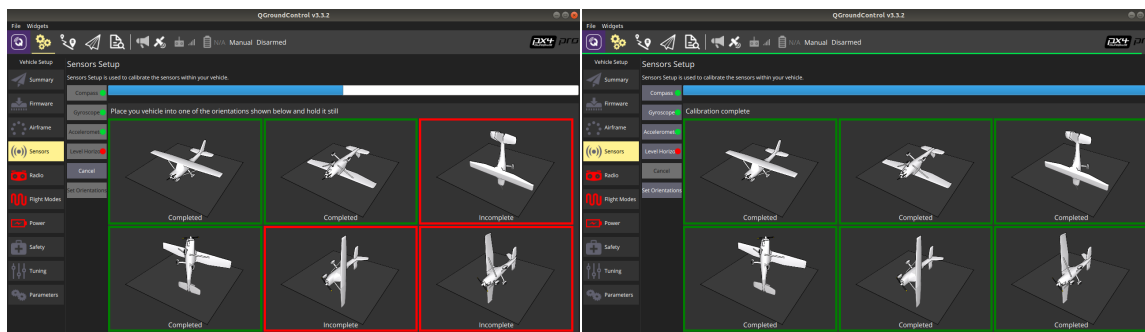
7.4. QGroundControl y calibración

Llegados a este punto, con el código ejecutándose y habiendo generado canales de comunicación podemos abrir el Control de Tierra. Este software previamente introducido servirá para calibrar, controlar y planear los principales aspectos del vuelo de la aeronave.

Calibración de compass

En la ventana de calibración se podrá calibrar este sensor. Esto se realizará siguiendo las instrucciones que se mostrarán gráficamente en el QGroundControl.

Tras unos sencillos pasos, se observa la calibración correcta en función de dos parámetros: todas las posiciones se han completado correctamente y el sensor status ejecutado en la línea de comandos, devuelve la información del calibrado.

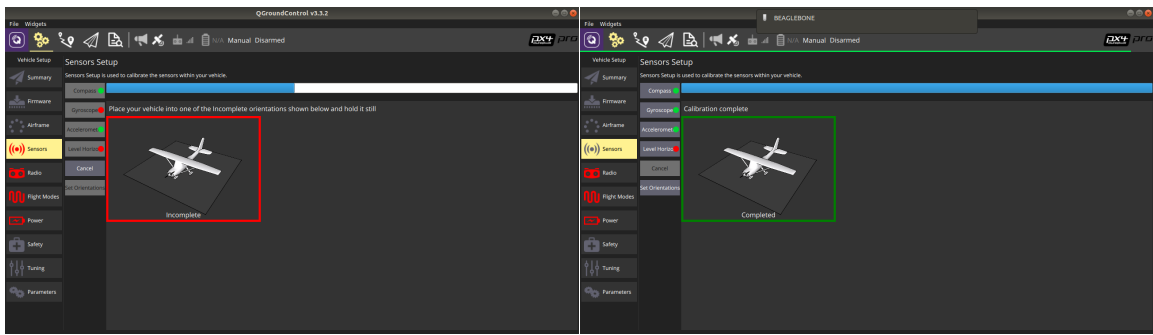


(a) Proceso de calibración del *compass*. (b) Calibración completada del *compass*.

Figura 7.4: Pantallas de calibración del *compass*.

Calibración de giroscopo

De forma similar, se podrá generar la calibración correcta del giróscopo. El procedimiento en este caso será algo mas largo, pudiéndose observar los progresos en la parte superior del QGroundControl (barra de estado) o mediante la línea de comandos en Cloud9.

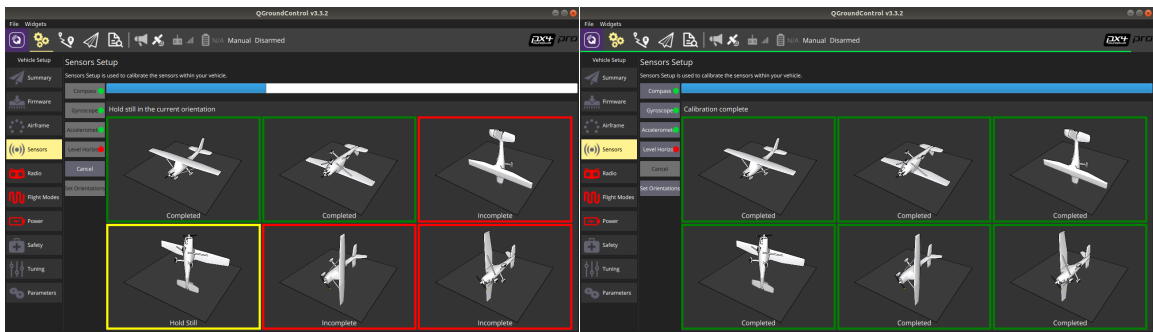


(a) Proceso de calibración del *giróscopo*. (b) Calibración completada del *giróscopo*.

Figura 7.5: Pantalla de calibración del *giróscopo*.

Calibración de acelerómetro

No se profundizará de forma intrínseca en las funcionalidades del QGroundControl o se ensamblará el UAV plenamente operativo con batería, GPS, Mando, etc. Como se especificó en el apartado de alcance del proyecto el desarrollo fundamental del mismo es la generación del *Middleware* necesario para los principales sensores necesarios en un UAV (barómetro y unidad inercial). En las siguientes imágenes se observa la calibración final del acelerómetro.



(a) Proceso de calibración del *acelerómetro*. (b) Calibración completada del *acelerómetro*.

Figura 7.6: Pantalla de calibración del *acelerómetro*.

Resultados de la calibración

Además de los evidentes resultados de la calibración y lectura correcta de información por parte de la unidad inercial, se puede comprobar el estado de los sensores de diversas formas. Por otra parte, se podría profundizar en los comandos disponibles por el código PX4 una vez se ejecuta de forma satisfactoria. Sin embargo, no es el objetivo fundamental del apartado de resultados ya que sólo se busca la ratificación de una conexión de la IMU9250 y el BMP280.

Sensors status: mediante este comando se puede visualizar el estado y valor concreto de los sensores cuando se llama. Los términos que aparecen serán conceptos estadísticos de valor, media o confianza. Por otra parte, se diferencian los errores del magnetómetro con su estado de STALE que anula la validez de las medidas y del barómetro que para el caso de ECNT permite y muestra valores, pero insinúa su gran error y dispersión.

```
INFO [sensors] gyro status:
INFO [ecl/validation] validator: best: 0, prev best: 0, failsafe: NO (0 events)
INFO [ecl/validation] sensor #0, prio: 75, state: OK
INFO [ecl/validation] val: -0.0016, lp: 0.0100 mean dev: -0.0837 RMS: 1.0884 conf: 1.0000
INFO [ecl/validation] val: 0.0004, lp: -0.0004 mean dev: 0.0758 RMS: 0.7811 conf: 1.0000
INFO [ecl/validation] val: -0.0026, lp: 0.0269 mean dev: 0.1878 RMS: 0.9849 conf: 1.0000
INFO [sensors] accel status:
INFO [ecl/validation] validator: best: 0, prev best: 0, failsafe: NO (0 events)
INFO [ecl/validation] sensor #0, prio: 75, state: OK
INFO [ecl/validation] val: -0.0433, lp: -0.0463 mean dev: -0.4577 RMS: 1.9668 conf: 1.0000
INFO [ecl/validation] val: -0.0942, lp: -0.2003 mean dev: -1.7777 RMS: 2.1417 conf: 1.0000
INFO [ecl/validation] val: -9.7980, lp: -9.7615 mean dev: -0.9319 RMS: 1.8337 conf: 1.0000
INFO [sensors] mag status:
INFO [ecl/validation] validator: best: -1, prev best: 0, failsafe: YES (1 events)
INFO [ecl/validation] sensor #0, prio: 1, state: STALE
INFO [ecl/validation] val: 0.0000, lp: 0.0000 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
INFO [ecl/validation] val: 0.0000, lp: 0.0000 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
INFO [ecl/validation] val: 0.0000, lp: 0.0000 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
INFO [sensors] baro status:
INFO [ecl/validation] validator: best: -1, prev best: -1, failsafe: NO (0 events)
INFO [ecl/validation] sensor #0, prio: 75, state: ECNT
INFO [ecl/validation] val: 1019.0941, lp: 1019.0908 mean dev: 0.0083 RMS: 0.0180 conf: 0.0000
INFO [ecl/validation] val: 42.7400, lp: 42.7177 mean dev: -0.0093 RMS: 0.0167 conf: 0.0000
INFO [ecl/validation] val: 0.0000, lp: 0.0000 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
```

Figura 7.7: Resultado del comando *sensors status*.

listener: este comando servirá para listar la información relativa a cada sensor. Su invocación se producirá particularizando para el elemento del cuál se desea recibir la información.

```
pxh> listener sensor_accel
TOPIC: sensor_accel instance 0 #1
sensor_accel_s
timestamp: 66394317 (0.009960 seconds ago)
integral_dt: 0
error_count: 0
x: 0.028
y: -0.246
z: -9.819
x_integral: 0.000
y_integral: 0.000
z_integral: 0.000
temperature: 0.000
range_00: -1.000
scaling: -1.000
device_id: 4288776
x_raw: 277
y_raw: -349
z_raw: -9619
temperature_raw: 0
pxh>
```

(a) Detalle del acelerómetro.

```
pxh> listener sensor_gyro
TOPIC: sensor_gyro instance 0 #1
sensor_gyro_s
timestamp: 87372139 (0.014115 seconds ago)
integral_dt: 0
error_count: 0
x: -0.004
y: 0.000
z: -0.004
x_integral: 0.000
y_integral: 0.000
z_integral: 0.000
temperature: 0.000
range_00: -1.000
scaling: -1.000
device_id: 4288776
x_raw: -6
y_raw: -8
z_raw: 2
temperature_raw: 0
pxh>
```

(b) Detalle del giróscopo.

```
pxh> listener sensor_baro
TOPIC: sensor_baro instance 0 #1
sensor_baro_s
timestamp: 100529109 (0.001055 seconds ago)
error_count: 24389139753999012
pressure: 1019.113
temperature: 43.650
device_id: 3056600184
pxh>
```

(c) Detalle del barómetro.

```
pxh> listener sensor_mag
TOPIC: sensor_mag instance 0 #1
sensor_mag_s
timestamp: 49437663 (0.005298 seconds ago)
error_count: 0
x: 0.000
y: 0.000
z: 0.000
range_00: -1.000
scaling: -1.000
temperature: 0.000
device_id: 4288776
x_raw: 0
y_raw: 0
z_raw: 0
is_external: 0
pxh>
```

(d) Detalle del magnetómetro.

Figura 7.8: Resultados del comando *listener* sobre los principales sensores.

Nuevamente, según las figuras 7.8c y 7.8d se corrobora la existencia y conexión de los mismos, pero se identifica una anomalía en los vectores de datos obtenidos que generan un *error count* en el primer caso y una imposibilidad de lectura en el segundo.

Conclusiones

La mayor dificultad de este proyecto ha quedado evidenciada en la verificación del código. Esta faceta ya se había introducido como compleja siendo la integración hardware-software uno de los principales escollos del proyecto. Ser un diseño novedoso y no conocer previamente ninguna de la estructura del software o el hardware ha agravado este problema.

Por otra parte, a pesar de que no se haya abarcado el desarrollo integro de toda la capa Middleware se ha evidenciado los posibles problemas y la complejidad que entrañan estas propuestas. Exigen un enorme conocimiento sobre fundamentos de ordenadores, código a implementar y hardware objetivo, que en muchos casos derivan en problemas de compilación o ejecución¹.

En lo relativo al análisis de resultados, se han conseguido desarrollar interfaces viables para iniciar y leer valores desde el barómetro y la unidad inercial. Por otra parte, se ha conseguido ejecutar el código de forma satisfactoria sobre la BeagleBone Blue, concluyendo que todas las particularidades relacionadas con el mapeo de la BeagleBone se han elaborado de forma satisfactoria.

Evidentemente, no se han conseguido elaborar drivers para la gestión de los LEDs ni el ADC, puesto que no se establecieron dichos objetivos, siendo no necesarios para la funcionalidad de PX4. El inicio y calibración de los sensores de la unidad inercial refuerza la tesis sobre la ejecución satisfactoria del código, sin embargo, no se ha solventado un problema asociado a lecturas incorrectas incapaces de ser procesadas por el sistema.

El hecho de limitar el alcance del proyecto y de disponer de herramientas como el DriverFramework ha permitido hacer abordable y asumible la empresa a realizar, ya que siempre se ha rechazado la idea de desarrollar cualquier parte del código íntegramente desde el principio. Se ha tratado de un trabajo de adaptación, búsqueda, revisión y resolución de conflictos (como se postuló en sus orígenes) para ser capaz de realizarse en tiempo y bajo las condiciones propuestas, obteniéndose un resultado satisfactorio.

¹Problemas como *smash stacked detected* o *segmentation fault*

Bibliografía

- [1] PX4 Development Team (2014). *DroneCode discussion forum, PX4, QGroundControl, SDK, MAVLink* [foro]. Consultado: 7 Julio 2018
- [2] PX4 Development Team (2014). *PX4 Autopilot Pro: About us*. Accesible en: <http://px4.io/about-us/> [en línea]. Consultado: 7 Julio 2018.
- [3] PX4 Development Team (2014). *PX4 Autopilot Pro: Developer guide*. Accesible en: <https://dev.px4.io/en/> [en línea]. Consultado: 7 Julio 2018.
- [4] PX4 Development Team (2014). *PX4 Autopilot Pro: User guide*. Accesible en: <https://docs.px4.io/en/> [en línea]. Consultado: 6 Julio 2018.
- [5] “PX4 source code”, GitHub, 2018. [en línea]. Disponible en: <https://github.com/PX4/Firmware>. Consultado: 06 Julio 2018.
- [6] QGroundControl Development Team (2014). *QGroundControl: User guide*. Accesible en <https://docs.qgroundcontrol.com/en/>. Consultado: 7 Junio de 2018.
- [7] QGroundControl Development Team (2014). *QGroundControl: Developer guide*. Accesible en: <https://dev.qgroundcontrol.com/en/>. Consultado: 7 Junio de 2018.
- [8] “MAVLink: protocolo de comunicación para drones”, *El mundo de los drones en tus manos.*, 2017. [en línea]. Disponible en: <http://www.xdrones.es/mavlink/>. Consultado: 09 Julio 2018.
- [9] “The Reactive Manifesto”, *Reactivemanifesto.org*, 2014. [en línea]. Disponible en: <https://www.reactivemanifesto.org/>. Consultado: 08 Julio 2018.
- [10] “Computer Organization and Architecture, Seventh Edition”, *Williamstallings.com*, 2009. [en línea]. Disponible en: <http://williamstallings.com/COA/COA7e.html>. Consultado: 08 Julio 2018.
- [11] Bakken, David, ”MIDDLEWARE”, Whashington State University Pullman, Washington: Washington State University, 2003. Disponible en: <http://www.dia.uniroma3.it/cabibbo/ids/altrui/middleware-bakken.pdf>
- [12] “Preguntas frecuentes sobre GNU/Linux - Proyecto GNU - Free SoftwareFoun-

- ation”, *Gnu.org*, 2018. [en línea]. Disponible en: <https://www.gnu.org/gnu/gnu-linux-faq.es.html#osvskernel>. Consultado: 08 Julio 2018.
- [13] “BeagleBoard.org - discuss”, *Beagleboard.org*, 2018. [en línea]. Disponible en: <http://beagleboard.org/discuss>. Consultado: 08 Julio 2018.
- [14] “librobotcontrol”, GitHub, 2018. [en línea]. Disponible en: <https://github.com/StrawsonDesign/librobotcontrol>. Consultado: 06 Julio 2018.
- [15] “ArduPilot Open Source Autopilot”, *Ardupilot.org*, 2018. [en línea]. Disponible en: <http://ardupilot.org/>. Consultado: 08 Julio 2018.
- [16] Real Decreto 1036/2017. para la utilización civil de las aeronaves pilotadas por control remoto. Decreto de 15 de diciembre de 2017. BOE-A-2017-15721
- [17] “Amazon, Google y Facebook apuestan fuerte por los drones — Dronair Dronair”, *Dronair.es*, 2017. [en línea]. Disponible en: <https://www.dronair.es/amazon-google-y-facebook-apuestan-fuerte-por-los-drones>. Consultado: 07 Julio 2018.
- [18] “Enterprise Drone Market To Grow And For Good Reasons - CXOtoday.com”, *Cxotoday.com*, 2017. [en línea]. Disponible en: <http://www.cxotoday.com/story/drones-have-a-huge-potential-for-the-commercial-market/>. Consultado: 07 Julio 2018.
- [19] “Windows 10 - Microsoft Store España”, *Microsoft.com*, 2018. [en línea]. Disponible en: <https://www.microsoft.com/es-es/store/b/windows>. Consultado: 07 Julio 2018.
- [20] “Amazon.com: Books”, *Amazon.com*, 2018. [en línea]. Disponible en: <https://www.amazon.com/books-used-books-textbooks/b?ie=UTF8&node=283155>. Consultado: 07 Julio 2018.
- [21] “Creative Commons — Reconocimiento-CompartirIgual 2.5 España — CC BY-SA 2.5 ES”, *Creativecommons.org*, 2018. [en línea]. Disponible en: <https://creativecommons.org/licenses/by-sa/2.5/es/>. Consultado: 07 Julio 2018.
- [22] MPU-9250 Datasheet [en línea].San Jose (California, USA): Invensense Inc, 2003. Actualizado el 20/6/2016. Citado: 7 Junio 2018. Revisión 1.1. Referencia: PS-MPU-9250A-01.
- [23] BST BMP280 Digital Pressure Sensor Datasheet [en línea]. Gerlingen (Baden-Württemberg, Germany): Robert Bosch GmbH, 1886. Actualizado 15/5/2015. Citado 7 Junio 2018. Revisión 1.14. Referencia: 0273 300 416.
- [24] Procesador Intel Core i5-3337U datasheet [en línea]. Santa Clara (California,USA): Intel Corporation, 1968. Actualizado en Marzo de 2013. [citado 7 Junio 2018]. Disponible en: <https://ark.intel.com/es-es/products/72055/Intel-Core-i5->

3337U-Processor-3M-Cache-up-to-2_70-GHz

- [25] “beagleboard/beaglebone-blue”, GitHub, 2017. [en línea]. Disponible en: https://github.com/beagleboard/beaglebone-blue/blob/master/docs/BeagleBone_Blue_ShortSpec.pdf. Consultado: 06 Julio 2018.
- [26] “BeagleBoard.org - community supported open hardware computers for making”, Beagleboard.org, 2018. [en línea]. Disponible en: <https://beagleboard.org/>. Consultado: 07 Julio 2018.
- [27] A. Silberschatz, P. Galvin and G. Gagne, *Operating system concepts*. Hoboken, NJ: John Wiley & Sons, 2013.
- [28] I. Englander, *The architecture of computer hardware and systems software: An information technology approach*. Hoboken, NJ: Wiley, 2010.
- [29]]D. Molloy, *Exploring BeagleBone*. Indianapolis, IN :Wiley, 2015.
- [30] R. Grimmett, *BeagleBone robotic projects*. Birmingham: Packt Pub., 2013.
- [31] TEAM DRONEII, “Drone Market Environment Map 2018 - Drone Industry Insights”, *Drone Industry Insights*, 2018. [en línea]. Disponible en: <https://www.droneii.com/drone-market-environment-map-2018>. [Consultado: 07- Julio 2018].
- [32] E. Martínez, “Los drones y la privacidad”, *Herrero & Asociados*, 2017. .
- [33] “Dos clases de libertad: GPL y BSD”, *Linux Adictos*, 2018. [en línea]. Disponible en: <https://www.linuxadictos.com/dos-clases-de-libertad-gpl-y-bsd.html>. Consultado: 06 Julio 2018.
- [34] “Beagle Bone Black GPIO LED control”, *Linuxkernel51.blogspot.com*, 2018. [en línea]. Disponible en: <http://linuxkernel51.blogspot.com/2015/08/beagle-bone-black-gpio-led-control.html>. Consultado: 06 Julio 2018.
- [35] “BeagleBone Black Booting Process”, *slideshare.net*, 2018. [en línea]. Disponible en: <https://es.slideshare.net/sysplay/beaglebone-black-booting-process-78730088>. Consultado: 06 Julio 2018.
- [36] D. Molloy, “Writing a Linux Loadable Kernel Module (LKM) - Interfacing to GPIOs — derekmolloy.ie”, *derekmolloy.ie*, 2018. [en línea]. Disponible en: <http://derekmolloy.ie/kernel-gpio-programming-buttons-and-leds/>. Consultado: 07 Julio 2018.
- [37] “Navio Documentation”, *Docs.emlid.com*, 2018. [en línea]. Disponible en: <https://docs.emlid.com/navio/>. Consultado: 07 Julio 2018.
- [38] D. Molloy, “Beaglebone: An I2C Tutorial — derekmolloy.ie”, *derekmolloy.ie*, 2013. [en línea]. Disponible en: <http://derekmolloy.ie/beaglebone/beaglebone->

- an-i2c-tutorial-interfacing-to-a-bma180-accelerometer/. Consultado: 07 Julio 2018.
- PWM* “Linux Core PWM User’s Guide - Texas Instruments Wiki”, *Processors.wiki.ti.com*, 2018. [en línea]. Disponible en: http://processors.wiki.ti.com/index.php/Linux_Core_PWM_User%27s_Guide. Consultado: 07 Julio 2018.
- thales* “Unmanned Aerial Vehicles: Scenarios, Solutions, Perspectives — Thales Group”, *Thalesgroup.com*, 2016. [en línea]. Disponible en: <https://www.thalesgroup.com/en/worldwide/magazine/unmanned-aerial-vehicles-scenarios-solutions-perspectives>. Consultado: 07 Julio 2018.
- [39] “Here are the world’s largest drone companies and manufacturers to watch and invest in”, *Business Insider*, 2017. [en línea]. Disponible en: <http://www.businessinsider.com/top-drone-manufacturers-companies-invest-stocks-2017-07?IR=T>. Consultado: 07 Julio 2018.
- [40] “Drone market shows positive outlook with strong industry growth and trends”, *Business Insider*, 2017. [en línea]. Disponible en: <http://www.businessinsider.com/drone-industry-analysis-market-trends-growth-forecasts-2017-7?IR=T>. Consultado: 07 Julio 2018.
- [41] “Commercial Unmanned Aerial Vehicle (UAV) Market Analysis – Industry trends, companies and what you should know”, *Business Insider*, 2017. [en línea]. Disponible en: <http://www.businessinsider.com/commercial-uav-market-analysis-2017-8?IR=T>. Consultado: 07 Julio 2018.
- [42] “Drones Startups”, *AngelList*, 2018. [en línea]. Disponible en: <https://angel.co/drones-2>. Consultado: 07- Julio 2018.
- [43] “What are the pros and cons of an I2C versus an SPI interface? - Quora”, *Quora.com*, 2014. [en línea]. Disponible en: <https://www.quora.com/What-are-the-pros-and-cons-of-an-I2C-versus-an-SPI-interface>. Consultado: 07 Julio 2018.
- [44] “Linux Core ADC Users Guide - Texas Instruments Wiki”, *Processors.wiki.ti.com*, 2018. [en línea]. Disponible en: http://processors.wiki.ti.com/index.php/Linux_Core_ADC_Users_Guide#. Consultado: 07 Julio 2018.
- [45] “IIO device files”, *Wiki.analog.com*, 2017. [en línea]. Disponible en: https://wiki.analog.com/software/linux/docs/iio/iio_snippets. Consultado: 07 Julio 2018.
- [46] “Linux Industrial I/O Subsystem [Analog Devices Wiki]”, *Wiki.analog.com*, 2017. [en línea]. Disponible en: <https://wiki.analog.com/software/linux/docs/iio/iio>. Consultado: 07 Julio

2018.

- [47] A. Naushad, “Basic Linux Commands for Beginners — Linux”, *Maker Pro*, 2018. [en línea]. Disponible en: <https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners>. Consultado: 07 Julio 2018.
- [48] “The Linux Kernel Archives”, *Kernel.org*, 2018. [en línea]. Disponible en: <https://www.kernel.org/>. Consultado: 07 Julio 2018.