Capítulo 11
Conclusiones y líneas futuras

En este capítulo:

En este capítulo se presentan las conclusiones obtenidas del trabajo realizado en este PFC, y que permiten proyectar las líneas futuras de trabajo.

Conclusiones

A la vista de los resultados se puede concluir, de forma global, que es posible realizar un sistema embebido con componentes de bajo coste que sea capaz de controlar un vehículo aéreo no tripulado en vuelo.

Gracias al gran avance de la industria de los *Smart Phones* y del automóvil, los costes de fabricación de los sensores necesarios para realizar este proyecto han bajado enormemente. Estos sensores tienen sus limitaciones y aunque no son adecuados para aplicaciones que requieran mucha precisión, son adecuados para los fines de este proyecto. Son altamente dependientes de la temperatura y normalmente vienen acompañados de mucho ruido. Para compensar estos inconvenientes, se han aplicado filtros digitales IIR y técnicas de *machine learning* para calibrar estos sensores y mejorar la estimación.

El prototipo hardware del proyecto se ha dividido en varios módulos por cuestiones prácticas. Se ha separado en placas distintas la electrónica de radiocomunicaciones y la electrónica de control. Inicialmente se diseñó un transceptor de 16W de potencia en 900Mhz, pero para las pruebas realizadas no se requirió tanta potencia. Las comunicaciones de telemetría se realizaron finalmente con 20dBm y se alcanzaron distancias de aproximadamente 1km.

La limitación de espacio físico fue un inconveniente desde un comienzo. Se disponía de una aeronave que no estaba pensada para incluir más peso del sus propios actuadores, por lo que incluía muy poco espacio físico para alojar componentes externos. Esta limitación supuso un reto a la hora de diseñar la placa de control que incluye dos procesadores ARM Cortex-M3, triple redundancia en sensores inerciales, sensores de presión diferencial y absoluta para medir la altura y velocidad, GPS, control de hasta 20 servomotores y lectura de 12, entradas salidas de propósito general para control de la carga de pago, entradas analógicas, comunicaciones por USB, I2C, bus CAN, RS-485 y RS-232. También incluye una memoria FLASH *onboard* de 16MB y un conector de tarjeta uSDCard. Un procesador se encarga de realizar la fusión sensorial y los algoritmos del piloto automático y navegación, mientras que el otro se encarga exclusivamente del control de la aeronave. La separación de estas tareas en dos procesadores permite añadir un grado de seguridad al sistema. Esto permite al operador tomar el control manual de la aeronave en cualquier momento, aunque aparezca algún

problema en el procesador principal. La placa es el corazón del proyecto, y se puede decir que aproximadamente el 90% del esfuerzo del proyecto se dedicó a la programación de los dos microcontroladores.

Gracias a la arquitectura software implementada, se pudo cumplir con holgura las restricciones de *hard real-time* que acompañan a todo sistema de piloto automático. Se evitó el uso de librerías externas y se implementó todo el software, desde la programación a bajo nivel trabajando con los registros de microcontrolador hasta los complejos algoritmos de los sistemas de aviónica.

El estudio de la arquitectura del procesador ARM Cortex-M3 permitió escribir códigos que se compilen de forma eficiente y poder utilizar las potencia del juego de instrucciones que ofrece esta arquitectura. Con el fin de utilizar al máximo los recursos hardware disponibles en el microcontrolador, se hizo un estudio en profundidad de los periféricos que éste incluye.

La depuración por hardware fue fundamental para el desarrollo de los algoritmos. Éste permite ver en tiempo real el estado del microcontrolador, la memoria y lo que está haciendo en la realidad el código. Esto permitió detectar y corregir errores de forma muy rápida durante el proceso de desarrollo del proyecto.

En la realización del proyecto fue fundamental seguir el procedimiento de diseño de una aplicación en tiempo real, dividiendo el trabajo en tareas, cada una de ellas responsable de una porción del trabajo. Cada tarea (también llamada *thread*) es un programa simple que piensa que tiene por completo la CPU para ella sola. En un sistema con una única CPU, sólo se puede ejecutar una tarea en un instante dado.

El kernel es responsable de la gestión de tareas. Multitarea es el proceso de programación y de conmutación de la CPU entre varias tareas. La CPU conmuta su atención entre varias tareas secuenciales. La multitarea da una ilusión de tener múltiples CPUs y en realidad maximiza el uso de la CPU. Uno de los aspectos más importantes de la multitarea es que ayuda en la creación de aplicaciones modulares. Los programas de aplicación son más fáciles de diseñar y mantener cuando se utiliza multitarea. El kernel también es responsable de gestionar las comunicaciones entre tareas y gestionar los recursos del sistema.

Lo que diferencia a los sistemas *hard* y *soft real-tim*e, es su tolerancia a los plazos de *deadline* y las consecuencias de no cumplirlos. Los valores calculados correctamente después de que haya pasado el *deadline* a menudo son inútiles.

Para los sistemas *hard real-time*, saltarse el *deadline* no es una opción. De hecho, en muchos casos, saltarse el *deadline* a menudo resulta en una catástrofe (como es el caso de este proyecto), y que puede implicar vidas humanas. Sin embargo, para los sistemas *soft real-time*, pasarse del *deadline* normalmente no es crítico.

La arquitectura software implementada está montada sobre el kernel de un RTOS y puede verse desde dos puntos de vista:

- En capas, donde cada capa ofrece servicios a la capa superior.
- Siguiendo la aproximación *Ouside-In*, donde la aplicación se descompone en tareas, y se siguen patrones de diseño software para sistemas embebidos en tiempo real.

Se diseñó una arquitectura llamada SupersonicOS que está formada por las capas del kernel, el Chip *Support Package* (CSP) y el *Board Support Package* (BSP). Ésta ofrece servicios a la capa de aplicación en tiempo real cumpliendo con las restricciones de *hard real-time*. Ejemplos de servicios que se incluyen son las comunicaciones, lectura automática de sensores y calibración, información de la posición en 3D, información de la *attitude*, control y lectura de los servos, monitorización de la batería y la temperatura, información de altura barométrica y la velocidad, registro automático de todos los valores calculados y las señales de los sensores en la uSDCard y manejo del USB.

El diseño de la capa de aplicación y del BSP tuvo como base la consideración de los problemas de antibloqueo (*deadlock*) y de inversión de prioridades.

La aproximación *Outside-In* permitió descomponer la aplicación en más de 30 tareas concurrentes. Se siguieron patrones de diseño software para sistemas embebidos en tiempo real para poder cumplir con las restricciones de *hard real-time* impuestas por la naturaleza del piloto automático. También se hizo uso constante de los servicios del sistema operativo (semáforos, mutex, mailboxes, etc) con el fin de optimizar la utilización de la CPU por las distintas tareas. El uso frecuente del DMA permitió paralelizar las funciones de transferencias de datos y así evitar cargar al procesador con tareas innecesarias. Ninguna de las tareas ocupa la CPU para realizar retardos, esperas, u operaciones similares que son innecesarias y malgastan el uso de la CPU. Con la arquitectura SupersonicOS y los algoritmos de los distintos subsistemas de aviónica funcionando, con todos los canales de comunicación funcionando de forma continua y en paralelo, el uso de la CPU es de un 9%. Cuando se activa la función de *Flight Recorder*, el sistema almacena con hora y fecha cada variable calculada, la lectura de cada uno de los sensores y la posición de los servos. Con esta función activada el uso de la CPU sube a un 40% ya que la comunicación la uSDCard es a través de SPI y es mucha información a almacenar en tiempo real.

Los resultados obtenidos en el algoritmo implementado para el AHRS son altamente satisfactorios. En la Tabla 1 se repiten los resultados obtenidos y una comparativa con otros productos existentes en el mercado. Como se puede observar, los resultados de desviación estándar obtenidos en la estimación del *yaw, pitch* y *roll* son hasta cuatro veces inferiores al de los otros productos.

Tabla 1 – Comparativa del AHRS con otros productos del mercado

Parámetro	EPF AHRS (1σ)	UAV Navigation VECTOR	PhD Sebastian Madgwick	XSense MTi-G-700
Roll	**0.05º**	0.5º	0.594º	0.2º
Pitch	**0.05º**	0.5º	0.497º	0.2º
Yaw	**0.1º**	1.0º	1.0º	1.0º

Finalmente, los algoritmos de navegación y del piloto automático implementados fueron testeados en simulaciones con *hardware-in-the-loop* con excelentes resultados.

El desarrollo de este proyecto permitió fusionar los conocimientos de diferentes áreas adquiridos durante la carrera así como ponerlos en práctica, tales como:

- Fabricación. Diseño avanzado de PCBs. Procesos de fabricación de PCBs mediante línea química y fresado por CNC. Montaje y soldadura de PCB con componentes de inserción, SMD estándar, SMD de paso fino y QFN. Fabricación de máscaras para dosificación de pasta de soldadura. Diseño mecánico con SolidWorks y corte de piezas con una cortadora láser.
- Circuitos de RF y microondas. Transmisión de datos de telemetría en la banda de 900Mhz, diseño de amplificadores de potencia para la banda de 900MHz y diseño de antenas con array de parches para la banda de 5.8GHz.
- Circuitos electrónicos analógicos. Diseño de conversores DC-DC, filtros pasivos analógicos y electrónica de potencia aplicada a motores.
- Sistemas embebidos en tiempo real y RTOS. Conceptos y patrones de tiempo real para sistemas embebidos. Sincronización y comunicación entre tareas. Modularización de una aplicación para convertirla en concurrente. Diseño de drivers de bajo nivel para RTOS. Programación de sistemas basados en microcontroladores de 32bits. Utilización de buses como: RS232, RS485, I2C, SPI,

CAN, USB y Ethernet. Manejo de interrupciones y periféricos como: Timers, PWM, Motor Control, RTC, ADC, DAC, SDCard y DMA. Depuración en hardware utilizando JTAG y SWD.

- Procesado digital de señales. Diseño y programación de filtros digitales en C, filtros Kalman y técnicas de *machine learning*.
- Comunicaciones. Diseño y programación en C de tramas y protocolos de comunicaciones.
- Sistemas de aviónica. Sistemas de datos de aire, sistemas de control de vuelo *fly-by-wire*, sensores inerciales, sistemas de navegación, pilotos automáticos y *flight management systems*.
- Aerodinámica. Cálculos básicos de aerodinámica como sustentación, resistencia al viento y estabilidad longitudinal.
- Otros. Sistemas de radio control, transmisión de vídeo analógica, robots microcontrolados, acondicionamiento de sensores y fusión sensorial.

Líneas futuras de investigación

En este apartado se indican algunas posibles mejoras que se podrían añadir al diseño propuesto y que podrían ser utilizados en futuros proyectos que traten de diseñar cualquier sistema de aviónica de bajo coste para UAVs.

Posibles mejoras en el hardware

En los siguientes puntos, se enumeran una serie de propuestas que pueden mejorar las prestaciones del trabajo realizado.

- Cambiar la arquitectura hardware a un solo procesador. Inicialmente no se tenía idea de la carga computacional que iba a requerir los algoritmos de aviónica, por lo que se diseñó el sistema con doble procesador por dos motivos: añadir seguridad y repartir la carga computacional. Durante el desarrollo del proyecto, se ha comprobado que un solo núcleo ARM es más que suficiente para afrontar la carga computacional de todo el sistema, al mismo tiempo que reduce la complejidad y el consumo de potencia.
- Cambiar la distribución de la alimentación. El estabilizador lineal que alimenta la electrónica digital a 3.3V a partir de 5V consume 1W de potencia en calor que es la mitad del consumo total de toda la electrónica de control. Habría que sustituir este estabilizador lineal por un conversor DC-DC, y sustituir todos los conversores DC-DC más modernos de más alta frecuencia y mayor eficiencia. Esto permite reducir el área del *footprint* de la electrónica de alimentación.
- Actualizar los sensores por los sensores más modernos y precisos que han aparecido éste último año.

Posibles mejoras en el Software

A raíz de los conocimientos adquiridos durante el desarrollo del proyecto, se plantean las siguientes mejoras en el software.

- El algoritmo del AHRS implementado obtiene excelentes resultados, pero a costa de que cada sistema requiera una calibración personalizada. Esto dificulta la producción en masa. Por este motivo, se plantea diseñar otros algoritmos que permita que el sistema se vaya auto calibrando en tiempo real.

- Mejorar el *Ground Control Station*. Incluir un mapa en el software para no tener que usar el Google Earth como software aparte. También hay que trabajar mucho en la interfaz gráfica.

Posibles líneas futuras de desarrollo

Como futura línea de investigación, se piensa trabajar más en el FMS. La idea principal sería pasar las funciones del FMS a una PCB con más capacidad de procesamiento, donde se le pueda conectar una cámara para hacer procesado de vídeo, o un láser scanner, o una Kinect. El objetivo principal es incluir un mecanismo para poder hacer navegación en interior sin necesidad de GPS a partir de percepción visual y realizar algoritmos complejos de robótica e inteligencia artificial para trabajar con sistemas cooperativos.

Otro punto donde se desea centrar la atención es en el sistema de navegación inercial. Estudiar los modelos de los sistemas de navegación inercial que se puedan fusionar con el GPS para poder mejorar la estimación de la posición cuando haya pérdidas temporales de la señal del GPS.

En la Figura 1 se muestra la arquitectura del nuevo sistema. Se aprovecharía el trabajo ya hecho en la parte de control, y se trabajaría más en la inteligencia del sistema.

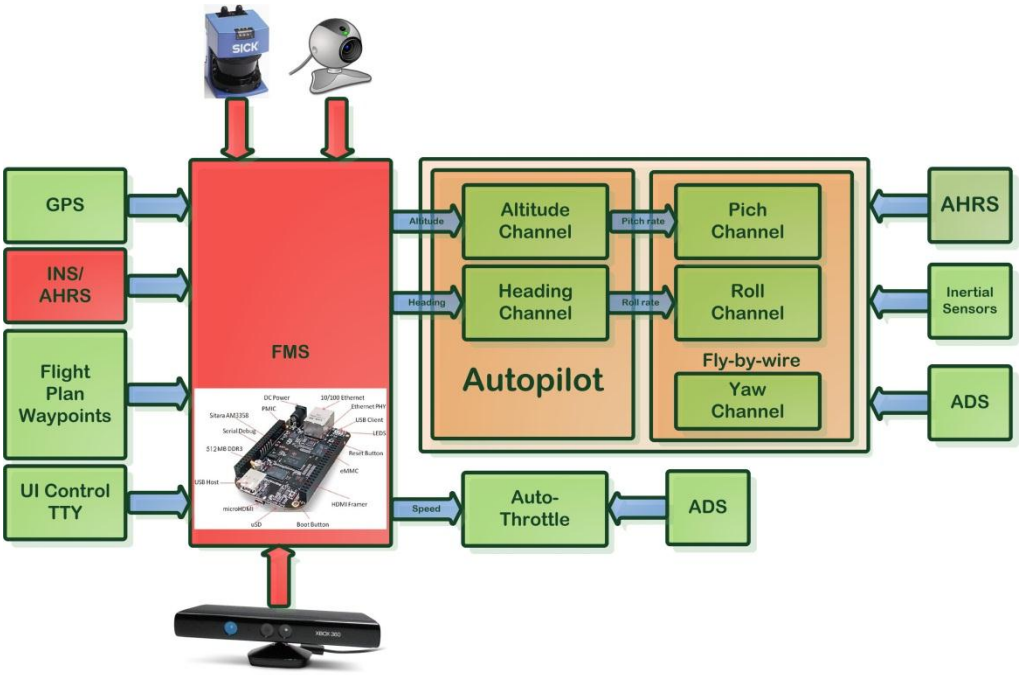

Figura 1 – Futura línea de investigación.