ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Acceso a datos internos de telemetría del dron MKQuad MikroKopter para su uso en aplicaciones avanzadas

(Internal telemetry data access of MKQuad MiKroKopter drone for advanced aplications)

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Sergio Fernandez Lorenzo

Octubre – 2020

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Sergio Fernandez Lorenzo
Director de TFG: Hector Posadas Cobos
Título: "Acceso a datos internos de telemetría del dron MKQuad MikroKopter para su uso en aplicaciones avanzadas"
Title: "Internal Telemetry data access of drone MKQuad MiKroKopter using for advanced aplications"

Presentado a examen el día:

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Eugenio Villar Bonet
Secretario (Apellidos, Nombre): Iñigo Ugarte Olano
Vocal (Apellidos, Nombre): Héctor Posadas Cobo

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente	Fdo.: El Secretario
Fdo.: El Vocal	Fdo.: El Director del TFG (sólo si es distinto del Secretario)
Vº Bº del Subdirector	Trabajo Fin de Grado N.º (a asignar por Secretaría

Índice

1. Introducción

En los últimos tiempos, los drones han empezado a jugar un papel cada vez más importante en la economía y los procesos productivos. Las compañías han comenzado a introducirlos en sus áreas de negocio, conscientes de las oportunidades que abren en múltiples industrias como la agricultura, las infraestructuras, la seguridad o la logística. Esta gran variedad de áreas ha hecho que el número de empresas dedicadas a operar estas aeronaves no tripuladas no deje de crecer, siendo en un gran número de casos su tamaño reducido, especialmente en España [1].

Además, con el desarrollo de otros campos como la inteligencia artificial, es posible aumentar la complejidad funcional y la capacidad de movimiento autónomo de los drones. Esto permite el desarrollo de drones con amplias capacidades para las actividades concretas a las que son destinados. Esta expansión de las capacidades de los drones está dando lugar a un amplio interés en el desarrollo de tecnologías asociadas a los drones, tanto desde el punto de vista de la empresa como en el área investigador [2][3]. Sin embargo, el desarrollo de estos elementos necesita también de tecnologías más básicas, como sistemas de comunicaciones, baterías, sistemas ligeros con bastante capacidad de cómputo, y manejo de la sensórica del dron.

Además, en el manejo de drones hay que considerar con otras características físicas, de entre las cuales el peso es un elemento fundamental. El peso de un dron, especialmente en sistemas basados rotores, debe ser soportado directamente por la fuerza de las hélices, ya que no pueden ayudarse de otras soluciones como las alas de los aviones o el cuerpo de los zepelines. Eso significa que el mantenimiento del peso del dron en el aire requiere de una gran cantidad de energía. Pero, además, esa energía está almacenada en una batería, en la que cualquier aumento de capacidad también implica un importante aumento de peso, haciendo que el problema del peso añadido al dron sea aún mayor. En consecuencia, podemos decir que cada gramo cuenta.

En este trabajo, esta problemática del peso se ha centrado en el manejo de la unidad de medida de movimiento (IMU). En muchas ocasiones, las plataformas base de los drones traen integrados unas UMUs de fábrica que no siempre son fáciles de acceder. En muchas ocasiones, el código o la información de cómo se puede acceder a los datos internos obtenidos por la IMU es cerrado y no se puede acceder a él, con lo que se necesita un trabajo de estudio de una documentación escasa y algo de ingeniería inversa para poder acceder a la información de los sensores. En consecuencia, si una empresa o unidad de investigación quiere utilizar esa plataforma para desarrollar aplicaciones más complejas sobre ella, esa dificultad de acceso se convierte en un problema serio a resolver. Una alternativa es incluir otra IMU en el dron, que sea fácilmente manejada por el sistema adicional añadido a la plataforma base. Sin embargo, eso implica un aumento del peso y de la energía consumida por el dron. Además, las IMUs necesitan procesos adicionales, como la calibración de los sensores, que ya vienen resueltos en la IMU de la plataforma base del dron, pero que habría que desarrollar adicionalmente en caso de poner otra IMU adicional.

Este trabajo se ha centrado en explorar la posibilidad de acceder a la información de una IMU compleja de usar, preparando una infraestructura básica que permita desarrollar funcionalidades complejas de forma sencilla. Para ello se ha utilizado el dron MKQuad

MikroKopter que existía previamente en posesión del grupo de Ingeniería Microelectrónica de la Universidad de Cantabria.

En consecuencia, en este documento se describe el método de comunicación de los datos de telemetría del dron MKQuad MikroKopter[4], la codificación interna que utiliza, así como el proceso por el cual se ha desarrollado el sistema de acceso a dicha información de forma automática utilizando una placa FPGA ZedBoard.

1.1 Motivación

En nuestro caso, contamos con una versión básica del dron MKQuad MikroKopter configurado como un cuadricóptero controlado por control remoto de forma manual. También contamos con un código VHDL capaz de generar señales ppm desarrollados en el departamento de TEISA para su uso con este dron.

Se desea ampliar la utilidad del dron para ser capaz de realizar futuras tareas de experimentación. Para ello, el principal requisito es que ha de ser posible controlar el dron mediante un programa de vuelo de forma automática. Dicho programa de vuelo se habrá de ejecutar en una plataforma programable que sea capaz de ejecutar el código y contar con suficiente capacidad para implementar cualquier otra funcionalidad para la que se desee emplear el dron.

La razón principal de estas condiciones es el minimizar el peso y consumo total del dron, ya que al tratarse de un dron multi-rotor su principal limitación es el peso que es capaz de manejar, que se traduce en estrés en los motores, y el tiempo de vuelo total, que depende de la batería disponible y de lo rápido que los motores agoten dicha batería.

1.2 Objetivos

Como primer objetivo y punto de partida, se desea entender el funcionamiento y control del dron y asegurar que es capaz de realizar un vuelo controlado.

Como segundo objetivo, se desea establecer una comunicación bidireccional entre la placa de control interna del dron y una placa FPGA externa, basada en la plataforma Zedboard[5]. Para ello será necesario averiguar cómo acceder a los datos de telemetría y comandos internos del dron, así como desarrollar un programa que permita a la placa FPGA recibir, codificar y enviar el mismo tipo de comandos y simular la señal PPM de control que originalmente se recibiría desde el mando radio control.

2. Tecnología de Drones

El término vehículo aéreo no tripulado se hizo común en los años noventa para describir las aeronaves robóticas o drones. Se trata de un vehículo aéreo sin ocupantes, pilotado por control remoto. Actualmente, este tipo de vehículos se utilizan principalmente para operaciones de ocio y entretenimiento, reconocimiento y control, toma de fotografías y video y, ocasionalmente, operaciones de transporte.

Existe una gran multitud de distintas configuraciones de drones en un gran rango de complejidad y coste variables, desde plataformas básicas y relativamente baratas diseñadas para ocio y educación a diseños profesionales para aplicaciones específicas, industriales o aplicaciones militares.

Actualmente la tecnología de drones está cada vez mas presente en campos cada vez más diversos y sigue desarrollándose a gran velocidad.

2.1 Aplicaciones

Actualmente, la tecnología de drones tiene una gran cantidad de aplicaciones, especialmente en la industria audiovisual gracias a la facilidad con la que son capaces de proporcionar fotografías y videos aéreos.

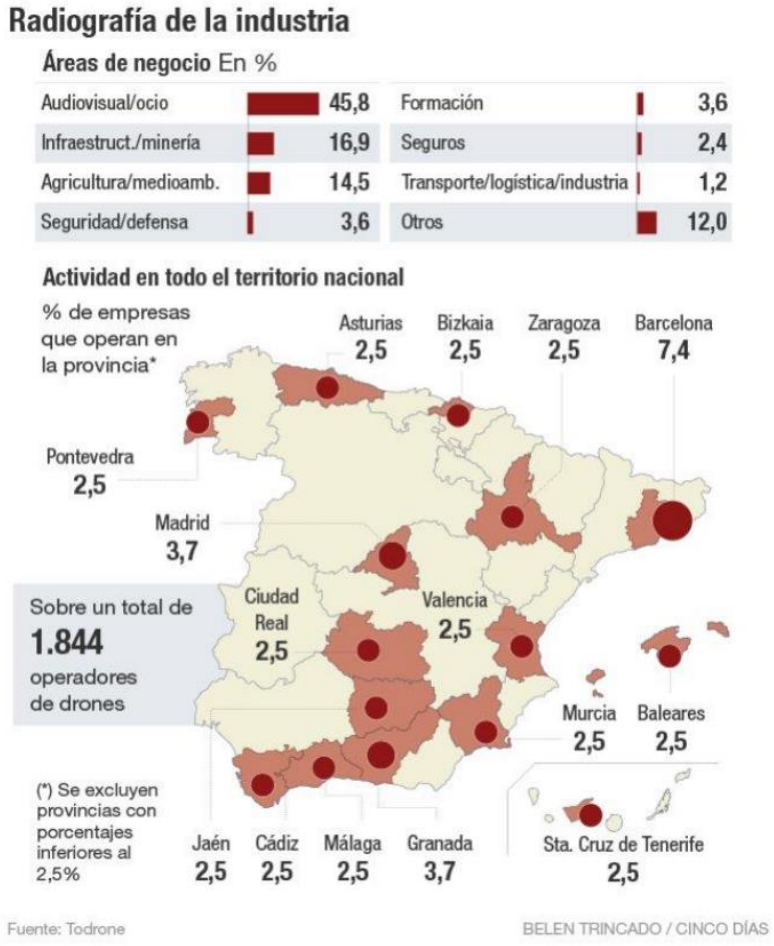

Figura 1: Uso industrial de los Drones en España [1]

Si bien esta es la aplicación principal, actualmente se han desarrollado tecnologías y usos mucho más especializados tanto en el sector civil como el militar.

Dentro de las aplicaciones militares destacan la vigilancia y reconocimiento, detección y clasificación de amenazas adquisición de blancos e incluso formas de combate directo.

En el ámbito civil y comercial su aplicación principal es indiscutiblemente en el ámbito audiovisual seguida por aplicaciones recreativas, pero también se emplean para otras situaciones como mapeado topográfico, escáner 3D de terrenos, objetos u otros entornos, o inspecciones.

En el caso de la industria fotovoltaica, el uso de drones equipados con cámaras térmicas permite monitorizar el rendimiento de sistemas de paneles solares de forma rápida y barata.

En otras industrias, el uso de drones permite la inspección de equipamiento u operaciones minimizando el riesgo humano, especialmente en trabajos de altura o con riesgo de exposición a gases perjudiciales.

también se benefician del uso de drones otras industrias que se ven en la necesidad de controlar grandes cantidades de terreno, como la industria de agricultura o en operaciones de prospección y minería de recursos.

Sin embargo, el sector que más ha crecido en lo que se refiere al uso de drones es en el ámbito de ocio. Desde la toma de video y fotografías aéreas, videos grabados desde drones con seguimiento automático de un objetivo o espectáculos y exhibiciones de vuelo.

Por último, pero no menos importante, en el caso de aplicaciones militares, cuentan con la ventaja de que estos vehículos no necesitan de un piloto, por lo que son más baratos y cuentan con una capacidad de carga efectiva más alta. Esta es la razón por la que el uso de drones esta tan integrada en operaciones militares en la actualidad. Así, dentro de las aplicaciones militares destacan la vigilancia y reconocimiento, detección y clasificación de amenazas adquisición de blancos e incluso formas de combate directo.

Esta variedad de aplicaciones se ve reflejado también en la diversidad de tipos de dones existentes en el mercado.

2.2 Tipos de drones

Existen dos clasificaciones básicas según la tecnología de vuelo: Ala fija o rotatoria, que se subdivide en helicópteros y multirrotores.

Ala Fija

Similar a un avión convencional, cuentan con la mayor autonomía gracias a no necesita mantener los motores en funcionamiento constante. Tiene menos agilidad de vuelo que los otros tipos y es incapaz de mantener una posición estacionaria, pero al ser capaz de

planear sin apenas usar energía es capaz de realizar aterrizajes de emergencia e incluso mantener el vuelo en caso de interferencias o interrupciones de la señal de control.

Helicópteros

Los helicópteros poseen la mayor capacidad de carga y una autonomía considerable gracias a que sólo poseen un motor funcionando a velocidad constante. Su desventaja es su complejidad mecánica y la necesidad de mantenimiento y ajustes constantes, además de la dificultad que presentan a la hora de ser pilotados.

Multirrotores

La tecnología más extendida, con una gran versatilidad y fácil mantenimiento y control de vuelo. Constituyen una Plataforma naturalmente estable, ya que los motores se encuentran a la misma distancia del centro de gravedad, lo que les proporciona una gran maniobrabilidad y control.

Según la cantidad de motores se clasifican en:

- Tricópteros
- Cuadricópteros
- Hexacópteros
- Octocópteros

según la configuración de los brazos los clasificamos en:

- Y griega
- Y invertida
- X
- Cruz

También existen los multirrotores coaxiales donde se montan dos motores por cada brazo.

La mayor desventaja de esta tecnología es la baja autonomía que tienen este tipo de drones comparados a los otros dos tipos.

3. MKQuad MikroKopter

El dron MKQuad MikroKopter es un dron multirrotor radiocontrol fabricado por la empresa alemana MikroKopter.

El modelo que se utiliza específicamente para este proyecto se encuentra actualmente descontinuado y sustituido por nuevas versiones, pero algunos de los componentes de este dron son los mismos que utilizan algunas de las nuevas iteraciones de la plataforma, como la placa de control de vuelo o el controlador de los motores.

3.1 Electrónica

Flight Control

El componente principal que contiene el procesador y los sensores principales de vuelo necesarios para un vuelo estable.

Como procesador utiliza un [AtMega] 644 y como sensores utiliza un set de giroscopios, que miden cambios en los ángulos x, y y z en grados por segundo, y acelerómetros, que miden la aceleración lineal sobre cada uno de los tres ejes.

También existe la opción de añadir un módulo adicional capaz de medir la altitud, aunque en el modelo utilizado en este caso dicho componente no se encuentra presente.

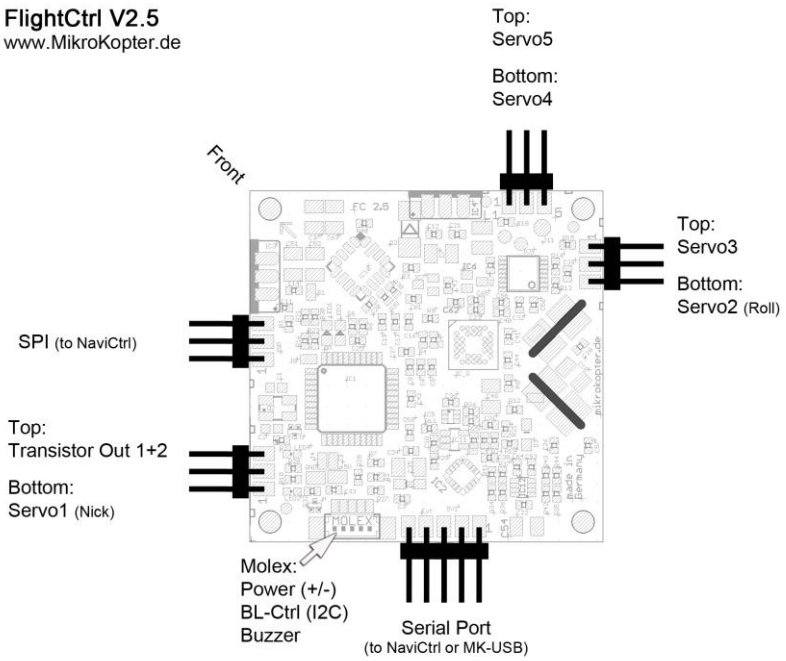

Figura 2.- Electrónica del dron

Como conexiones cuenta con una línea de I2C, la señal del buzzer para alarmas acústicas y el voltaje de entrada de la batería (12V) en el puerto molex. En la parte inferior hay varios pines que ofrecen conexiones directas a las mismas señales que el molex, así como a todas las señales de control de los motores de vuelo y servomotores opcionales para

montaje de cámaras, un jumper de configuración para el protocolo de telemetría y receptor de señales PPM Sum.

Receptor PPM Sum

El tipo de receptor que se utiliza depende de que mando se use para el control radiocontrol del dron. En este caso, se usa un receptor HoTT capaz de recibir señales analógicas de control desde el mando, así como enviar al display del mando datos de telemetría.

PPM - Summensignal + Telemetrieanschluss		
Pad	Funktion	cable collor
GN	GND/Minus	black or brown
+5	Plus 5V	red
PPM	Datenleitung	orange
RX	Telemetrieanschluss	arbitrarily
JET	Solder bridge for Telemetry	-

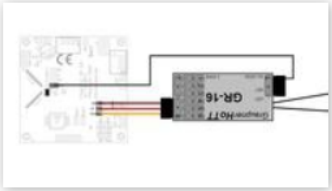

Figura 3.- Receptor PPM

Brushless Controller

MikroKopter utiliza cuatro controladores AVR ATMEGA8 de Atmel para motores sin escobillas, o brushless, conectados mediante I2C al procesador de FlightController, cada uno utilizando una dirección única.

Es por este tipo de comunicación entre los controladores y el procesador que un controlador brushless estándar no funcionaría en este sistema.

Motores

MikroKopter utiliza cuatro motores Motoren equipados con 2 hélices CW o Clock Wise y 2 CCW o Counter Clock Wise

3.2 Operación del Dron

Vuelo Manual

Para el vuelo manual de prueba se utiliza el mando Graupner MC-20 HoTT. Para conectar el mando al dron solo hay que, primero, comprobar que el firmware del mando esta actualizado a la última versión, y segundo, instalar el perfil MK-Easy que se puede descargar directamente desde la wiki de MikroKopter.

Por último, con el dron conectado a Windows, desde el software MikroKopter Tool, cargar la configuración de control MK-Easy para ajustar la asignación de canales de forma que coincide con el mando.

Es importante realizar un vuelo de prueba utilizando la misma secuencia de comandos que luego se va a utilizar de forma automática desde la FPGA para comprobar cómo responde el dron en un entorno controlado.

MikroKopter Tool

MikroKopter Tool está diseñado para poder actualizar, configurar y calibrar el dron directamente desde Windows. Aunque muchas de las funciones que ofrece no pueden usarse sin algunos de los módulos extra como el GPS, control de cámaras y módulos bluetooth, la herramienta ha resultado imprescindible para el desarrollo de este proyecto.

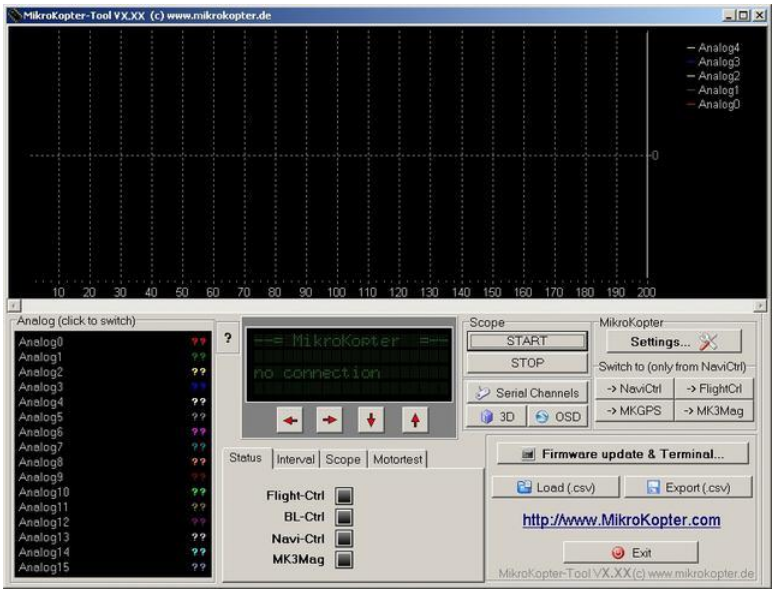

Figura 4.- Herramienta MicroKopter Tool

El display de los sensores internos del dron ha sido imprescindible para realizar comprobar la precisión de distintas lecturas, así como para identificar las señales de telemetría por el puerto serie. Este ha sido también el método más fiable para determinar si las señales ppm generadas desde la FPGA Zedboard envían los valores deseados.

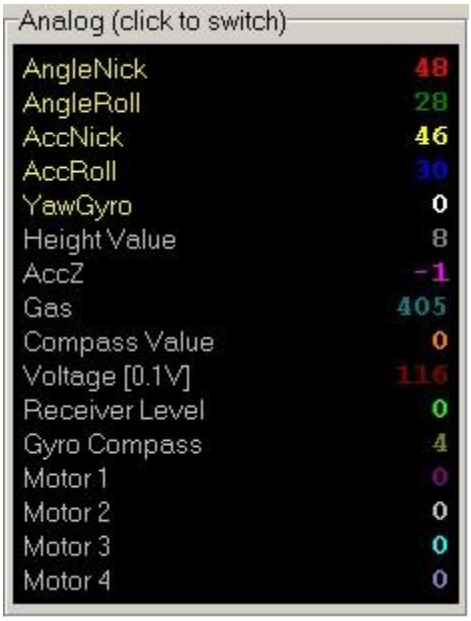

Figura 5.- Información general del Dron

También dentro de la propia herramienta se puede consultar la información de telemetría del display de FlightControl. Esta información es la que se enviaría al mando radiocontrol, en caso de ser un modelo compatible, pero en este caso se ha utilizado para obtener las lecturas de los sensores de vuelo.


Figura 6.- Telemetría del dron

La ventaja principal de obtener las lecturas de telemetría de esta forma es que las lecturas que se obtienen han sido ya procesadas directamente por el procesador del dron, lo que ahorra cálculos extra que se deberían hacer para llegar al mismo resultado si se leyesen directamente los valores a la salida de los sensores.

3.3 Calibración

Es imprescindible realizar al menos una calibración básica antes de iniciar cada vuelo, y recomendable realizar una calibración completa de todos los sistemas.

Durante el proceso de calibración el dron debe encontrarse en una superficie lo más plana y horizontal posible.

El proceso de calibración, encendido y apagado de motores se indican enviando distintas combinaciones de pitch, roll, yaw y throttle:

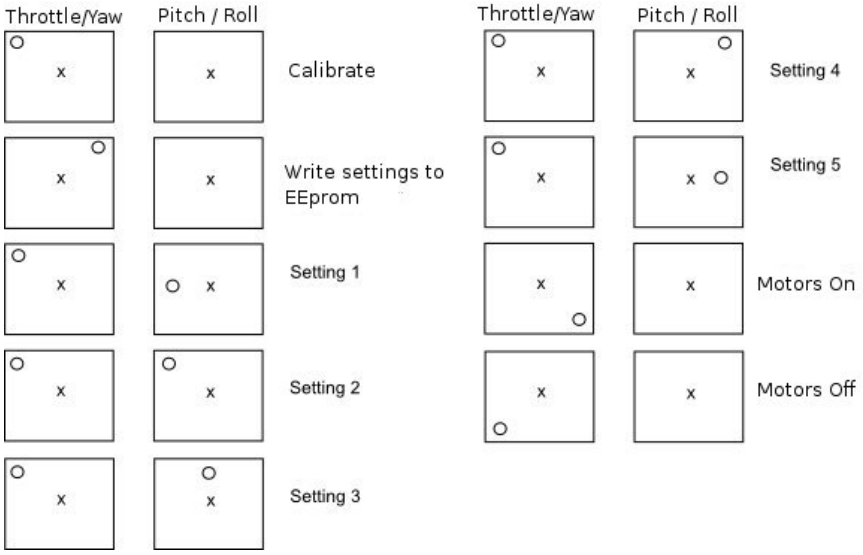

Figura 7.- Calibración del dron

Pre-Flight Check

Es recomendable realizar una comprobación visual de que no hay problemas con el dron antes de realizar cualquier operación de vuelo:

- El led verde de FlightControl este encendido y el buzzer no hace ruido.
- El led verde de cada uno de los cuatro controladores de motores este encendido.
- Durante el test de encendido de los motores, cada uno funciona, están firmemente fijados al dron y rotan en la dirección adecuada.
- Pasados unos segundos, el buzzer comenzara a hacer ruido cada vez que se pierde la señal RC transmitida desde el mando o generada desde la FPGA.

Calibración ACC

Los acelerómetros se han de calibrar al menos una vez, aunque es recomendable realizar una calibración completa de forma periódica. Una vez se han calibrado, los valores guardados se cargan cada vez que el dron se inicia.

Para realizar la calibración, con el dron en posición totalmente plana y horizontal, colocar el joystick izquierdo del mando RC en la posición 'Write Settings to EEprom' y esperar a la confirmación acústica.

Calibración Gyro

Los giroscopios han de calibrarse nuevamente siempre que se inicia el dron antes de realizar ninguna operación de vuelo, ya que los sensores son sensibles a variaciones de temperatura.

Para realizar la calibración basta con colocar los joysticks del mando RC en la posición de 'Calibrate' y esperar a que el buzzer suene confirmando que la operación se ha realizado con éxito.

Es posible elegir entre 5 distintos niveles de precisión dependiendo de la posición del joystick derecho. El número de pitidos del buzzer indica el nivel de precisión. De no seleccionar ninguno se utiliza el nivel 3 por defecto.

4. Desarrollo

4.1 Acceso externo a datos internos de telemetría

La obtención de la información del protocolo para poder acceder a la placa de control del dron, a efectos de poder mandar los comandos necesarios para su configuración y para obtener la información de telemetría ha supuesto quizá el mayor esfuerzo de este trabajo. La razón para esto ha sido la escasa, casi nula información disponible en muchos de los casos.

Para obtener dicha información, se han usado dos mecanismos fundamentales. En primer lugar, se ha partido de la información disponible en la wiki de Microkopter [6]. Esta información es muy reducida, pero da una primera idea sobre la que realizar la exploración del sistema de comunicaciones.

La segunda etapa ha consistido en una decodificación de la comunicación existente entre la herramienta MikroKopter Tool y el dron. Se ha utilizado la herramienta como forma de generar los comandos necesarios, escuchando la línea serie para decodificar los valores de los bits de los paquetes enviados y recibidos. Una vez detectada la configuración de los comandos a enviar, también se ha utilizado la herramienta para decodificar la respuesta a los comandos enviados, comprobando si los comandos enviados eran correctos o no. Para ello, se ha utilizado un osciloscopio y una placa Raspberry Pi [8].

Se pueden obtener lecturas de los datos de telemetría y valores de los sensores internos del dron desde la herramienta MikroKopter Tool, pero el objetivo es poder leer esos mismos datos directamente desde una placa externa. Para ello se ha de averiguar, primero, que conexión física nos permite el acceso, y segundo, como decodificar y obtener una lectura correcta.

Se utiliza una placa Raspberry Pi [8] para este proceso por lo fácil que es de programar y realizar ajustes mientras se realizan las pruebas necesarias para averiguar el funcionamiento del Código.

En primer lugar, se decide utilizar como puerto físico el mismo puerto USB que el dron utiliza para comunicarse con la herramienta MikroKopter tool. La ventaja de esto es que utiliza una conexión USB serial, por lo que es fácil hacer un proceso de depurado y comparar los datos que se están leyendo desde la raspberry con los que se reciben desde windows, o directamente en la herramienta MikroKopter tool.

En teoría, los datos de telemetría deberían ser también accesibles desde el receptor HoTT que envía y recibe señales PPM al mando RC. Sin embargo, el acceso por esta ruta es mucho más complicado y no da acceso a la lectura de todos los sensores internos del dron.

En segundo lugar, se procede a conectar por USB el dron con la raspberry y se configura para leer a la misma frecuencia que transmite el dron.

En este punto podemos empezar a escuchar los datos que envía el dron directamente sin codificar. En base a esto, y siguiendo la estructura que se describe en el punto 3.3 sobre la codificación que se utiliza, se puede primero decodificar a mano algunos de los datos,

comprobar comparando con MikroKopter Tool, y comenzar a escribir el código necesario para decodificar todo de forma automática.

Este código será el que luego se utilice en la fpga para solicitar e interpretar los datos necesarios para evaluar el estado del dron.

4.2 Conexión con la placa ZedBoard

Hardware

La conexión entre la placa ZedBoard y el módulo de control de la plataforma MikroKopter se basa en el periférico desarrollado por el departamento de TEISA capaz de generar la señal PPM SUM necesaria para enviar y recibir los datos de telemetría y control [9].
El objetivo es generar una señal similar a la que el dron recibiría del mando radiocontrol durante un vuelo manual que incluya la información de raw, pitch y roll, así como otras señales necesarias para el control del vuelo.
Para generar tal señal, el periférico cuenta de 14 registros, que contienen la información siguiente:

- Rg 0: Ciclos de reloj que debe estar la señal a '0' antes de pasar al '1' siguiente.
- Rg 13:
  - Bit 0: '1': indica que se deben cargar los datos que hay en los registros.
  - Bits 8-12: número de canales activos (sin contar el de espera)
- Rg 1-12: número de ciclos que cada canal debe estar a '1'.

Utilizando vivado, se realiza el siguiente diseño:

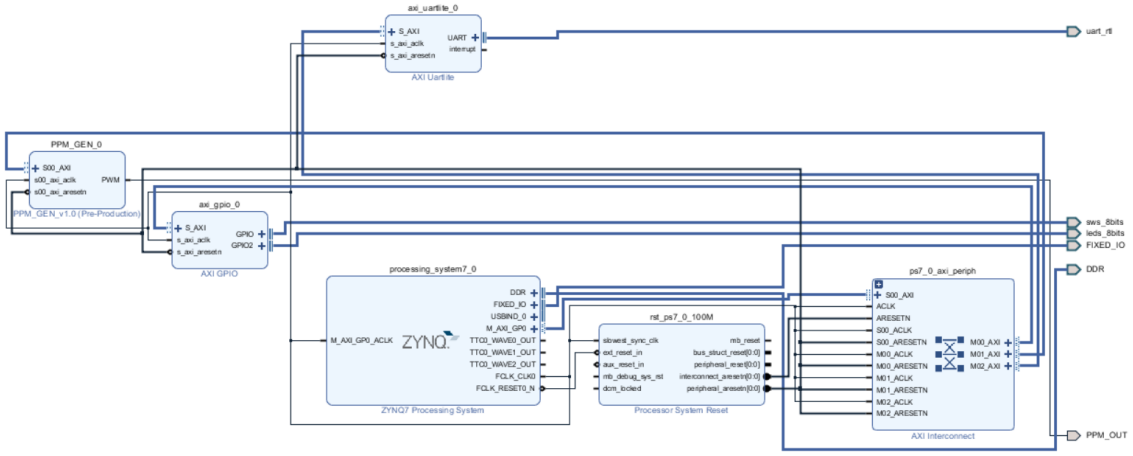

Figura 8.- Arquitectura del bloque de diseño en FPGA

- Procesador Zync7
  - Con la conexione UART activada y asignada
- periférico GPIO de entrada y salida estándar
  - Proporciona conexiones a los switches y leds de la placa ZedBoard
- periférico de generación de señales PPM
  - Con su salida conectada a un puerto de salida PPM_OUT

- Modulo UART LITE
  - Configurada a 57600 Baud Rate, 8 bits de datos y sin paridad

Esta configuración hardware permitirá configurar la señal PPM que se necesita para controlar el dron, así como enviar dicha señal y recibir los datos de telemetría del dron mediante la conexión UART.

Tras sintetizar el diseño y asignar los pines de entrada y salida se pasa a la configuración software, conforme a la información en [6].

Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref
All ports (149)										
DDR_41999 (71)	INOUT					✓	502	(Multiple)*	1.500	(Multiple)
FIXED_IO_41999 (59)	INOUT					✓	(Multiple)	(Multiple)*	(Multiple)	(Multiple)
leds_8bits_41999 (8)	OUT					✓	33	LVCMOS33*	3.300	
sws_8bits_41999 (8)	IN					✓	(Multiple)	LVCMOS25*	2.500	
uart_rtl_41999 (2)	(Multiple)					✓	13	LVCMOS18	1.800	
Scalar ports (2)										
uart_rtl_rxd	IN				AA11	✓	13	LVCMOS18	1.800	
uart_rtl_txd	OUT				Y10	✓	13	LVCMOS18	1.800	
Scalar ports (1)										
PPM_OUT	OUT				Y11	✓	13	LVCMOS18	1.800	

Figura 9.- Lista de pines de la FPGA

Desde el SDK, se ha de configurar la forma de la señal ppm a generar, el puerto UART y la transmisión y lectura de datos.

Señal PPM

La función de generación de la señal ppm toma un puntero a data y lo transforma una señal ppm interpretable por el procesador del dron. Al inicio, se definen los datos de la señal que se desea generar y que se van luego a cargar en los registros correspondientes.

```
/* PPM Channels */
#define NChannels           8
#define DataChannelsOffset  1
#define SyncChannel         (NChannels + 1)
#define ChannelsLogic       (SyncChannel << 8)
/* PPM Time */
#define Tcycle 100      //Escala la duracion de los ciclos de reloj
#define Tsignal 20000   //Tiempo total de cada iteracion de la senal
#define Tpause 400      //Tiempo a 0 entre pulsos
#define Tmed 1090       //Duracion media de pulso de datos (0%)
#define Tvar 410        //Rango de variacion del pulso de datos
/* PPM Data Range */
#define UpperRange 1
#define LowerRange -1
```

1. Indica en el registro 0 el tiempo que ha de estar la señal a 0 entre cada dato. Esto es lo que evita que una señal 0xFF aparezca como una línea constante.

```
//REG 0 -> Tpause
PPM[0] = Tpause * Tcycle;
```

2. Para cada canal de datos:
   a. Restringe el valor de la señal al máximo o mínimo permitido.
   b. Almacena el valor a transmitir escalado y centrado entre los valores máximo y mínimo permitidos. La duración del pulso para cada dato se

compone de una componente variable en función de su valor (Tvar) y una componente constante (Tmed).

```
//REG 1:NChannels -> DATA
for(int i = 0; i < NChannels; i++){
    //Data Range Check
    if(Data[i] > UpperRange){
        Data[i] = UpperRange;
    }else if(Data[i] < LowerRange){
        Data[i] = LowerRange;
    }

    Tchannels += (int) (Data[i] * Tvar + Tmed);
    PPM[i+DataChannelsOffset] = (int) ((Data[i] * Tvar + Tmed) * Tcycle);
}
```

3. La duración total de la señal ha de permanecer constante en cada ciclo un pese a que la duración de cada canal varíe en función de su valor, por lo que se utiliza un canal de sincronismo que extiende la señal hasta dicho valor a partir del tiempo total de todos los pulsos de datos y las pausas entre pulso y pulso transmitidos en cada ciclo.

```
int Tsync = Tsignal - Tchannels - (Tpause * SyncChannel);

//REG SyncChannel -> Sync
PPM[SyncChannel] = Tsync * Tcycle;
```

4. Se carga el bit de carga del registro de control manteniendo constante el indicador del número de canales activos.

```
//REG13 -> Load Registers
PPM[13] = ChannelsLogic + 1;
PPM[13] = ChannelsLogic;
```

Comunicación UART

```
#define UART_CR_RESET       0x03
#define UART_SR_TX_FULL     0x08
#define UART_SR_RX_EMPTY    0x01
```

```c
void uart_init(int BaseAddress){
    volatile int *ctrl = (int *) (BaseAddress + UART_CR);

    ctrl[0] = UART_CR_RESET;
    ctrl[0] = 0;
}
char RxByte(int BaseAddress)
{
    char Byte;
    volatile int *SR = (int *) (BaseAddress + UART_SR);
    volatile char *RX = (char *) (BaseAddress + UART_RX);

    while(!(SR[0] & UART_SR_RX_EMPTY)) { continue; }
    Byte = *RX;
    return (char) Byte;
}
void TxByte(int BaseAddress, char Byte)
{
    volatile int *SR = (int *) (BaseAddress + UART_SR);
    volatile char *TX = (char *) (BaseAddress + UART_TX);

    while(SR[0] & UART_SR_TX_FULL) { continue; }
    TX[0] = Byte;
}
```

La gestión de la comunicación UART se realiza mediante tres funciones:

1. Uart_init: Realiza un reset del módulo completo y deja el registro de control con valor 0.
2. RxByte: Espera hasta que el registro de estado detecta datos sin leer y los pasa a la salida como un byte tipo char.
3. TxByte: Espera a que el registro de salida este vacío para poder cargar y enviar un nuevo byte de datos.

4.3 Acceso a datos de Telemetría

Codificación Base64

Los datos que se reciben desde el puerto UART una vez que se conecta la placa ZedBoard al módulo de control del dron MikroKopter han de ser decodificados antes de poder ser leídos e interpretados.
Para ello, se utiliza la siguiente función:

```c
void Decode64(unsigned char *ptrOut, unsigned char len, unsigned char *RxdBuffer, unsigned char ptrIn, unsigned char max) {
    unsigned char a,b,c,d;
    unsigned char ptr = 0;
    unsigned char x,y,z;

    while(len){
      a = RxdBuffer[ptrIn++] - '=';
      b = RxdBuffer[ptrIn++] - '=';
      c = RxdBuffer[ptrIn++] - '=';
      d = RxdBuffer[ptrIn++] - '=';

      if(ptrIn > max - 2) break;

      x = (a << 2) | (b >> 4);
      y = ((b & 0x0f) << 4) | (c >> 2);
      z = ((c & 0x03) << 6) | d;

      if(len--) ptrOut[ptr++] = x; else break;
      if(len--) ptrOut[ptr++] = y; else break;
      if(len--) ptrOut[ptr++] = z; else break;
   }
}
```

El dron utiliza una variación de Base64 para codificar los datos de entrada y salida. La función que se utiliza aquí es una adaptación de la función original que utiliza el programa del propio dron que, para cada byte:

1. Anula el símbolo de padding.
2. Realiza las operaciones de decodificación
3. Acumula los bytes decodificados en la cadena de salida.

La ventaja de este método es que permite transmitir cada byte en menos de 8 bits por byte, pero la desventaja es que para decodificar los datos es necesario contar con la cadena de entrada completa ya que la decodificación de los bytes posteriores depende de los anteriores. Esto imposibilita la decodificación byte a byte según se reciben e introduce un pequeño retraso en la recepción de los comandos y datos de telemetría.

Comandos MikroKopter

El dron MikroKopter cuenta con un sistema de comandos específico para enviar y recibir información que siguen la siguiente estructura:

Start-Byte	Address Byte	ID-Byte	n Data-Bytes coded	CRC-Byte1	CRC-Byte2	Stop-Byte
'#'	'a'+ Addr	'V','D' etc	"modified-base64"	variable	variable	'\r'

Existen varios tipos de comandos para transmitir los datos de telemetría, pero debido a inconsistencias y errores en varios de los comandos diseñados para ese propósito, la mejor opción para obtener lecturas fiables es el comando Display:

	Received			*Sent*			
Description	*ID*	*Address*	*Data*	*ID*	*Address*	*Data*	*remark*
Request display	'h'	AnyAddr	u8 ~RemoteKey, u8 AutoSendInterval	'H'	SlaveAddr	char[80] DisplayText	
Request display	'l'	AnyAddr	u8 MenuItem	'L'	SlaveAddr	u8 MenuItem, u8 MaxMenuItem, char[80] Display Text	

Este comando manda desde el dron a la FPGA una cadena de datos codificados en base64 que, tras decodificar, resulta en una cadena de texto. Originalmente, el propósito de este comando es enviar información al mando radiocontrol para mostrar de forma rápida algunas de las lecturas de los sensores y configuración de vuelo.

La ventaja de utilizar el display para obtener las lecturas de telemetría es que proporciona acceso tanto a los valores directos de los sensores del giroscopio y acelerómetro del dron como los valores después de pasar por el módulo de control de vuelo, mucho más útiles para monitorizar la situación actual del dron.

La desventaja es que el display cuenta con varias “ventanas” de información, y para acceder a cada set de datos es necesario utilizar comandos de siguiente, anterior y enviar

periódicamente una petición de refresco para seguir recibiendo los datos. Asimismo, hace falta trabajar con cadenas de texto para extraer las lecturas de los sensores, lo cual introduce un pequeño retraso y hace este método especialmente vulnerable a futuras actualizaciones del firmware del dron.

Implementación

Para recibir y guardar los datos de telemetría como variables numéricas a partir de la cadena de texto recibida desde el comando Display hay que seguir los siguientes pasos:

1. Enviar una solicitud al MikroKopter para que inicie la transmisión indicando la frecuencia de actualización y la duración total. Para mantener una comunicación ininterrumpida, esta solicitud deberá seguir enviándose continuamente.
2. Ajustar dicha pantalla enviando al dron los comandos para siguiente y anterior hasta que se comience a recibir la pantalla con la información deseada.
3. Obtener a partir de la cadena de texto la información numérica de los sensores.

Ya que el envío de información al MikroKopter ha de ser constante e ininterrumpido, se realiza un fork() del programa y se dedica un hijo exclusivamente a controlar la comunicación entre la FPGA y el dron:

```
/* Kopter */
#define Kopter_display_request  "#ah|na=Fq"
#define Kopter_display_next     "#ah|NA=FQ"
#define Kopter_display_last     "#ah|^a=Fa"
#define Kopter_display_target   2   //ToDo: Document Acc Info Screen
 pid_t pid = fork();

 if(pid == -1){
     printf("[ERROR] Fork failed\n");
 }else if (pid == 0){
     unsigned char request[9] = Kopter_display_request;
     unsigned char requestNextScreen[9] = Kopter_display_next;
     int i = 0;
     int screen = 0;
     while(1){
         /* Ask to start data stream */
         if(i >= (sizeof(request)+1)){
             TxByte((int)UART, 0x0D); i = 0;
             sleep(1);
         }else TxByte((int)UART, request[i]);
         i++;
         /* Gets desired display screen:
          * 0: General Info
          * 1: Altitude Telemetry
          * 2: Nick, Roll, Compass Telemetry
          * 5: Raw Gyro, Nick, Roll, Yaw Sensor Data
          * 6: Acc Nick, Roll, Z */
         while(screen<Kopter_display_target){
             for(int j=0; j<=sizeof(requestNextScreen);j++){
                 if( j >= (sizeof(requestNextScreen)+1)){
                     TxByte((int)UART, 0x0D);
                 }else TxByte((int)UART, requestNextScreen[j]);
             } usleep(50); screen++;
         }
     }
```

Se aprovecha que al inicio el dron siempre empieza en la misma pantalla para mantener la cuenta de la pantalla mostrada utilizando una variable local en vez de esperar a decodificar los datos, lo que introduciría un retraso extra en el refresco de los datos de telemetría.

Una vez el dron comienza a enviar los datos deseados, el proceso para obtener las lecturas de los sensores consiste simplemente en pasar la información a la función de decodificación y analizar la cadena de texto obtenida para guardar la información necesaria para gestionar el vuelo del dron de forma automática:

```
/* Telemetry Data from display sceen [2] */
for(int i=0; i<=sizeof(ptr);i++){
    if(ptr[i-6] == 'N' && ptr[i-3] == 'k'){
        char data[5] = {0};
        for(int t=0; ;t++){
            if(ptr[i+t] == 0x20){
                data[t] = 0x0D; i+=t; break;
            } data[t] = ptr[i+t];
        } TelemetryData[0] = atof(data);
    }else if(ptr[i-6] == 'R' && ptr[i-3] == 'l'){
        char data[5] = {0};
        for(int t=0; ;t++){
            if(ptr[i+t] == 0x20){
                data[t] = 0x0D; i+=t; break;
            } data[t] = ptr[i+t];
        } TelemetryData[1] = atof(data);
    }else if(ptr[i-8] == 'C' && ptr[i-3] == 's'){
        char data[5] = {0};
        for(int t=0; ;t++){
            if(ptr[i+t] == 0x20){
                data[t] = 0x0D; i+=t; break;
            } data[t] = ptr[i+t];
        } TelemetryData[2] = atof(data);
    }break;
}
```

4.4 Control desde ZedBoard

Realizar un control automático del dron MikroKopter desde ZedBoard se encuentra con dos obstáculos:

1. Ser autosuficiente y capaz de iniciar la ejecución programada automáticamente.
2. Realizar la configuración y calibrado del dron previa al inicio del vuelo.

Bootloader

La FPGA ZedBoard utiliza una imagen de boot generada desde el SDK de Xilinx que le permite cargar Linux e iniciar la ejecución del programa de control directamente desde la tarjeta SD.

Desde el SDK se genera Boot.bin utilizando un proyecto default tipo bitloader e incluyendo la configuración hardware con el generador PPM y el puerto UART, y un archivo u-boot.elf que indica como se han de cargar el sistema operativo Linux desde los siguientes archivos:

1. devicetree.dtb
2. uImage
3. uramdisk.image.gz

Estos tres archivos, el boot.bin generado, y el bitstream con el programa principal a ejecutar han de estar cargados en la tarjeta SD al inicio de la FPGA.

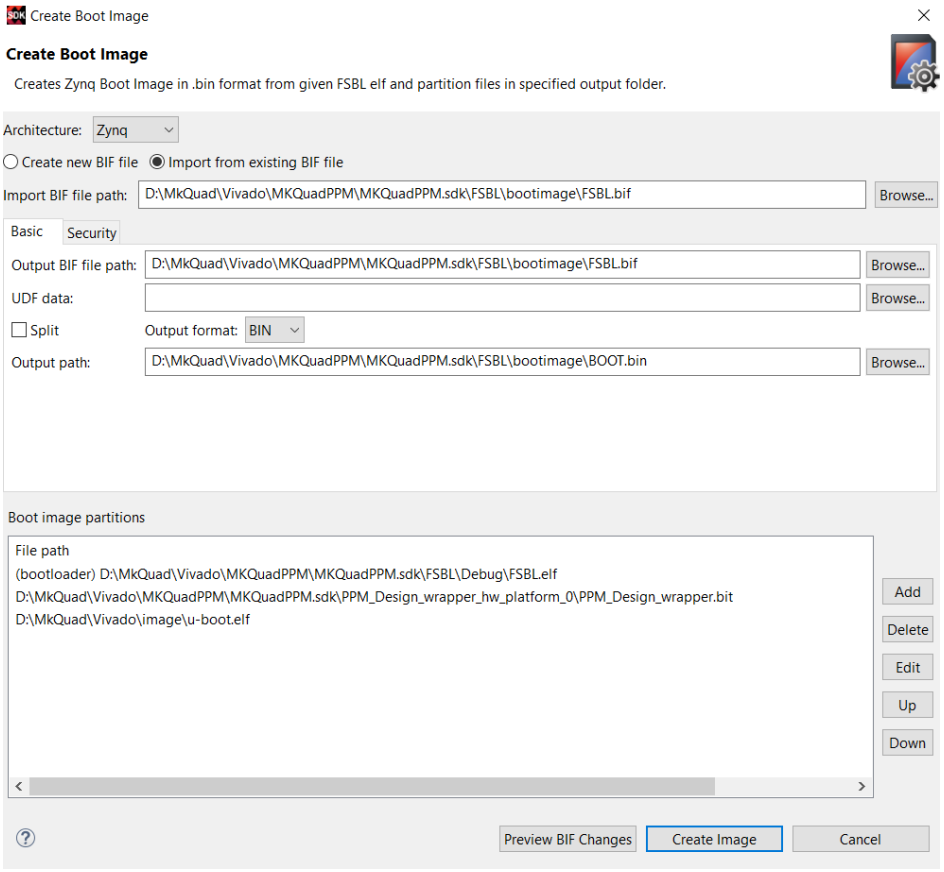

Figura 10.- Generación de la imagen

Calibrado y Ajustes pre-vuelo

Antes de realizar ninguna operación es necesario mapear las direcciones de memoria físicas de los periféricos a memoria virtual para poder operar desde Linux:

```
int mem_fd;
void *PPM;
void *UART;

if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC)) < 0){
    printf("Failed to open /dev/mem (check for permissions).\n");
    return -1;
}
PPM = mmap( NULL, BLOCK_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, mem_fd, PPM_ADDR);
if (PPM == MAP_FAILED) { perror("mmap"); return -1; }
UART = mmap( NULL, BLOCK_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, mem_fd, UART_LITE);
if (UART == MAP_FAILED) { perror("mmap"); return -1; }
```

Tras mapear los periféricos a memoria virtual, se pueden enviar las señales ppm necesarias para preparar al dron antes de iniciar el vuelo. Se realizan las mismas operaciones que para realizar un vuelo manual:

1. Calibración de los acelerómetros y sensores de posición.
2. Activar el control de altitud.
3. Activar el modo carefree.
4. Activar y desactivar los motores manteniendo un throttle negativo.

Durante todo este proceso, el dron deberá permanecer estacionario y en una superficie estable y plana. El buzzer incorporado en la placa FlightControl del MikroKopter deberá pitar repetidamente durante la calibración de los sensores. Por último, el dron ha de permanecer estacionario por sí mismo aun durante la activación de los motores.

En caso contrario, se deberá abortar inmediatamente.

```
/* PPM Channels */
#define Throttle        0
#define Roll            1
#define Nick            2
#define Yaw             3
#define Carefree        6
#define AltitudeControl 5


/* CalibrateACC */
printf("Calibrating ACC..."); usleep(50);
PPM_Data[Throttle] = 1; PPM_Data[Yaw] = 1; PPM_GEN(PPM, PPM_Data);
sleep(10);
PPM_Data[Throttle] = 0; PPM_Data[Yaw] = 0; PPM_GEN(PPM, PPM_Data);
printf(" done\n"); usleep(50);

/* Altitude Control*/
printf("Activating Altitude Control..."); usleep(50);
PPM_Data[AltitudeControl] = 1; PPM_GEN(PPM, PPM_Data);
printf(" done\n"); usleep(50);

/* Carefree Mode */
printf("Carefree On..."); usleep(50);
PPM_Data[Carefree] = 1; PPM_GEN(PPM, PPM_Data);
printf(" done\n"); usleep(50);

/* Start Motors */
printf("Starting engine..."); usleep(50);
PPM_Data[Throttle] = -1; PPM_Data[Yaw] = 1; PPM_Data[Nick] = -1; PPM_Data[Roll] = 1; PPM_GEN(PPM, PPM_Data);
sleep(5);
PPM_Data[Throttle] = 0; PPM_Data[Yaw] = 0; PPM_Data[Nick] = 0; PPM_Data[Roll] = 0; PPM_GEN(PPM, PPM_Data);
printf(" done\n"); usleep(50);

/* Stop Motors */
printf("Stopping engine..."); usleep(50);
PPM_Data[Throttle] = -1; PPM_Data[Yaw] = -1; PPM_Data[Nick] = -1; PPM_Data[Roll] = -1; PPM_GEN(PPM, PPM_Data);
sleep(5);
PPM_Data[Throttle] = 0; PPM_Data[Yaw] = 0; PPM_Data[Nick] = 0; PPM_Data[Roll] = 0; PPM_GEN(PPM, PPM_Data);
printf(" done\n"); usleep(50);
```

Vuelo Automático

Una vez realizada la calibración del dron, se inicia la comunicación y transmisión de los datos de telemetría que permiten monitorizar el estado del dron durante el vuelo:

```
/* Get Telemetry Data */
/* Input String */
for(i=0; ;i++){
    buffer[i] = RxByte((int)UART);
    if(buffer[i] == 0x0D){
        N = i;
        break;
    }else if(buffer[i] == '#') i=0;
}
/* Telemetry Data */
if(buffer[0] == '#'){
    addr = buffer[1];
    id = buffer[2];
    unsigned char data[N-5];
        for(i = 0; i < (N - 5); i++){
            data[i] = buffer[i+3];
        }data[i] = 0x0D;
    //crc[0] = buffer[N-2]; crc[1] = buffer[N-1]; crc[2] = buffer[N];
    printf("Rx: %s\n", buffer);
    printf("\tAddr: %c, ID: %c, Data: (%i) %s\n",addr, id, sizeof(data), data);

    Decode64(telemetry, sizeof(data), buffer, 3, N);
```

Desde el puerto UART se reciben continuamente cadenas de texto. Durante el boot del dron, la placa de control interna envía texto ASCII con información sobre la inicialización de los sistemas instalados. Seguidamente, comienza a enviar comandos periódicos por defecto.

Todos los comandos tienen la misma estructura: Un byte de inicio ‘#’, dos bytes de identificación, los datos codificados en base64 y dos bytes de crc.

El único comando de interés para la ejecución en vuelo es el de display, identificado por un byte de id ‘H’. En cuanto se recibe este comando, se extraen los bytes de datos y se decodifican para luego extraer los datos de los sensores.

```
    switch(id)
    {
    case 'T': if(addr == 'b'){ printf("\tEngine Test\n"); } break;
    case 'V': printf("\tVersion Request\n"); break;
    case 'H': if(printDisplay(telemetry) != Kopter_display_target)
                { printf("[Warning] Telemetry out of sync..."); state[2] = 0; }else{ state[2] = 1; } break;
    case 'D': printDebug(telemetry); break;
    case 'k': printWinkel(telemetry); break;
    case 'C': print3D(telemetry); break;
    default: printf("\tUnknown Command: %s\n", buffer);
    }
}else printf("%s\n", buffer); //Boot Text
```

5. Conclusiones

En este trabajo se ha conseguido acceder a la información de telemetría interna del dron MKQuad MikroKopter, solventando los problemas relacionados con la falta de documentación e información encontrados. Como resultado, se ha desarrollado una infraestructura HW/SW basada en una placa Zedboard, incluyendo el código software necesario para poder acceder a dicha información y controlar el movimiento del dron, como de los comandos necesarios para su manejo, la estabilización o el calibrado.

En lo que se refiere al acceso de los datos internos y lecturas de los sensores incorporados en el dron, el trabajo desarrollado demuestra que es posible recibir los datos deseados durante el vuelo del dron, aunque la velocidad de refresco de los datos puede no hacerlos válidos para todo tipo de aplicaciones.

Por ejemplo, la velocidad de transmisión es demasiado lenta para poder realizar correcciones instantáneas durante el vuelo. No obstante, sería posible implementar una rutina de vuelo automático, siempre y cuando el tipo de vuelo sea muy estable con cambios lentos y controlados ya que el programa sería incapaz de reaccionar a cambios repentinos simplemente porque la información que recibe del dron sobre el estado actual y posición espacial de la plataforma no se actualiza con la suficiente rapidez.

La principal causa de este problema es la velocidad del puerto UART programada en la placa interna del dron, así como el sistema de codificación y comandos que utiliza el dron para transmitir la información.

Es posible que este problema pueda resolverse reprogramando la placa base del dron, pero en ese caso sería más fácil directamente conectar los sensores y motores del dron a una placa FPGA y desarrollar un nuevo programa de control de vuelo directamente en la placa. Otra posible solución sería tratar de adquirir los datos necesarios para controlar las operaciones de vuelo directamente de los sensores en vez de a través de la placa de control. Esto significa encontrar una forma de acceder a las señales de los sensores internos sin alterar su conexión actual con la placa de control de vuelo, e implementar un código extra que se encargue de calibrar y procesar la información analógica y traducirla a información digital que se pueda usar para controlar el dron.

Un segundo problema que se ha detectado, es cierta falta de precisión en los datos recibidos. Eso significa que sería necesario un buen sistema de filtrado y corrección de los datos para permitir su utilización.

En el caso de realizar un vuelo manual, esto no es un problema porque un piloto humano es capaz de ajustar y reaccionar a los cambios en vuelo producidos por las pequeñas variaciones en la lectura de los sensores.

En el caso de un vuelo automático es más problemático, porque sin un sistema de realimentación, es imposible que la plataforma de control se dé cuenta de cuando empiezan a aparecer errores en los valores de la posición y orientación del dron.

En la práctica, quizás este problema se podría controlar si existiese otra forma de verificar la información. Por ejemplo, una cámara de video que permitiese observar cuando existen

movimientos inesperados y, más importante, que sea capaz de confirmar de forma periódica la orientación y posición actual del dron.

El mayor problema de las lecturas de posición es el error que introducen en el tiempo, lo que significa que cuanto mayor sea la duración del vuelo, mayor será el error total. Una entrada de video con su correspondiente análisis o una fuente de información adicional como un GPS que permitiese confirmar la orientación y posición del dron de forma periódica permitiria efectivamente resetear el valor de referencia de los sensores de posición y evitar que el error introducido crezca excesivamente.

Referencias

[1] Belén Trincado, “Drones, una industria creciente pero muy fragmentada y volátil”, Cinco Dias, 5 de Diciembre de 2016, https://cincodias.elpais.com/cincodias/2016/12/05/ tecnologia/1480965319_898663.html

[2] Jihun Park;Dae Hoe Kim;Young Sook Shin;Sang-ho Lee, “A comparison of convolutional object detectors for real-time drone tracking using a PTZ camera”, 17th International Conference on Control, Automation and Systems (ICCAS), 2017

[3] Arne Devos;Emad Ebeid;Poramate Manoonpong, "Development of Autonomous Drones for Adaptive Obstacle Avoidance in Real World Environments” , 21st Euromicro Conference on Digital System Design (DSD), 2018

[4] Microkopter, https://www.mikrokopter.de/en/home

[5] Xilinx Zynq 7000 Overview http://www.xilinx.com/support/documentation/ data_sheets/ds190-Zynq-7000-

[6] Microkopter Xiki, http://wiki.mikrokopter.de/en

[7] Overview.pdf xi Guía HW de la ZedBoard http://zedboard.org/sites/default/files/ documentations/ ZedBoard_HW_UG_v2_2.pdf

[8] Raspberry Pi Foundation, www.raspberrypi.org

[9] Ramón González Ruiz, “Plataforma para realización de rutinas de vuelo autónomo sobre cuadricóptero”, Trabajo Fin de Grado, Ingeniería de Tecnologías de Comunicación, Universidad de Cantabria, 2016