UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TRABAJO DE FINAL DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

DESARROLLO DE UN DRON DIRIGIBLE A TRAVÉS DE APLICACIÓN MÓVIL DE ANDROID CON CAPACIDAD DE GRABACIÓN DE RECORRIDOS

Memoria y Anexos

Autor/a: Abad Pérez, Joan
Director/a: Velasco Quesada, Guillermo

Convocatoria: Mayo 2021

Resumen

En este proyecto se realiza el diseño y montaje de un dron capaz de guardar el recorrido realizado, y se desarrolla una aplicación móvil para el control remoto del dron mediante comunicación Wifi. Se ha construido un dron de ala rotatoria de 4 motores con GPS incorporado, usando el entorno de desarrollo Arduino IDE. Los componentes más relevantes utilizados para su montaje son: un módulo de unidad inercial, un módulo GPS, una placa de distribución de energía, cuatro motores sin escobilla con sus cuatro variadores electrónicos y batería LiPo. La aplicación móvil se ha desarrollado con el entorno de desarrollo oficial de Android Studio. La interfaz de la aplicación simula las palancas de mando que encontramos en un mando de radiocontrol convencional, permite el ajuste remoto de los parámetros PIDs y la lectura del recorrido hecho por el dron. Para la comunicación entre el dron y la aplicación se ha utilizado el protocolo de comunicación TCP/IP. Integrar el módulo GPS permite la grabación del recorrido y podría servir en un futuro para conseguir un vuelo autónomo, pudiéndole dar una utilidad más práctica. Este proyecto consigue plasmar en profundidad todo el proceso de fabricación de un dron desde cero, el desarrollo de una aplicación móvil y de un sistema de comunicación vía Wifi.

Resum

En aquest projecte es realitza el disseny i muntatge d'un dron capaç de guardar el recorregut realitzat, i es desenvolupa una aplicació mòbil per al control remot del dron mitjançant comunicació Wifi. S'ha construït un dron d'ala rotatòria de 4 motors amb GPS incorporat, usant l'entorn de desenvolupament Arduino IDE. Els components més rellevants utilitzats per al seu muntatge són: un mòdul d'unitat inercial, un mòdul GPS, una placa de distribució d'energia, quatre motors sense escombreta amb els seus quatre variadors electrònics i bateria LiPo. L'aplicació mòbil s'ha desenvolupat amb l'entorn de desenvolupament oficial d'Android Studio. La pantalla de l'aplicació simula les palanques de comandament que trobem en un comandament de radiocontrol convencional, permet l'ajust remot dels paràmetres PIDs i la lectura del recorregut fet pel dron. Per a la comunicació entre el dron i l'aplicació s'ha utilitzat el protocol de comunicació TCP / IP. Integrar el mòdul GPS permet l'enregistrament del recorregut i podria servir en un futur per aconseguir un vol autònom, podent donar una utilitat més pràctica. Aquest projecte aconsegueix plasmar en profunditat tot el procés de fabricació d'un dron des de cero, el desenvolupament d'una aplicació mòbil i d'un sistema de comunicació via Wifi.

Abstract

In this project we design and assemble a drone capable of saving the route taken, and we develop a mobile application for remote control of the drone via Wifi communication. A 4-motor rotary wing drone with GPS has been built using the Arduino IDE development environment. The most relevant components used for its assembly are: an inertial measurement unit, a GPS module, a power distribution board, four brushless motors with their four electronic speed control and LiPo battery. The mobile application has been developed with the official Android Studio development environment. The application interface simulates the joysticks found in a conventional radio control, allows remote adjustment of the PIDs parameters and the reading of the path made by the drone. For the communication between the drone and the application, the TCP/IP communication protocol has been used. Integrating the GPS module allows the recording of the route and could be used in the future to achieve an autonomous flight, giving it a more practical use. This project manages to capture the whole process of manufacturing a drone from scratch, the development of a mobile application and a communication system via Wifi.

Índice

1 Introducción

En los últimos años ha habido un interés creciente en relación a los vehículos aéreos no tripulados, conocidos como drones, se está promoviendo su uso tanto en el ámbito profesional para solucionar problemas de manera más eficientes, como en el ámbito recreativo para tomar fotografías o simplemente como vehículo teledirigido. Habitualmente hay concursos o actividades que incentivan a ingenieros a diseñar un dron para cumplir algún objetivo en particular. Gracias a los avances tecnológicos y a los bajos costes de la electrónica básica, es posible, con los conocimientos adecuados, llevar a cabo esta tarea y crear un dron propio con las características específicas que se quiera incorporar.

Otro mundo, en relación con la electrónica, que también ha impactado en el siglo XXI, son los teléfonos móviles inteligentes, ya adoptando el nombre anglosajón: smartphones. En estos, existen gran variedad de aplicaciones con múltiples funciones, que incluso permiten el control de dispositivos del hogar o del coche, a través del concepto Internet of Things (IoT) que se basa en interconectar objetos a través de internet.

En este trabajo se va aplicar el concepto IoT para interconectar el dron con el Smartphone. Se realizará, por un lado, el diseño y montaje del dron, y por otro, se desarrollará una aplicación móvil a través de Android Studio que permita manejar el dron y obtener el recorrido grabado por este.

1.1 Objetivos del trabajo

Los objetivos de este proyecto son los siguientes:

- Montaje y desarrollo de un dron a través de Arduino IDE.
- Obtención de un vuelo estable del dron.
- Obtención de las coordenadas geográficas del recorrido realizado por el dron.
- Grabación en memoria flash de coordenadas recibidas por GPS.
- Programación del dron para recibir parámetros y consignas a través del Wifi.
- Desarrollo de una aplicación Android para el control remoto del dron.
- Envío de consignas y parámetros desde la aplicación al dron.
- Recepción, a la aplicación móvil, de datos guardados en el dron.

2 Marco teórico

2.1 Vehículo aéreo no tripulado

Los vehículos aéreos no tripulados, también conocidos como dron [1], son aeronaves que vuelan sin tripulación y que pueden mantener un vuelo controlado y sostenido de manera autónoma. El control del vuelo lo puede llevar a cabo un usuario de forma remota o puede llevar programado una ruta de vuelo autónoma sin control humano durante el vuelo. Los drones se propulsan mediante motores ya sean eléctricos, de reacción o de explosión. [2]

Originariamente los drones fueron desarrollados para un uso militar. Con el paso del tiempo y el desarrollo tecnológico han bajado notablemente los costes de obtención de la tecnología mínima necesaria para el montaje y desarrollo de un dron, y de la misma manera se ha reducido el tamaño de esta tecnología. Eso ha propiciado el aumento de usos de los drones y el aumento de producción de estos ya que se vuelven asequibles para el ciudadano medio.

Hoy en día podemos encontrar una gran variedad de usos de los drones que no son militares:

- Adquisición de fotos y videos.
- Realización de carreras.
- Distribución de internet.
- Transporte de mercancías.
- Servicios forestales.
- Servicios agrícolas.
- Mapear grandes áreas.
- Otros.

Dentro de la familia de los drones podemos distinguir dos grandes tipos de dron, los de ala fija y los de ala rotatoria.

Los de ala fija, como indica su nombre, son drones con alas fijas que, al igual que los aviones, aprovechan su forma aerodinámica para generar sustentación gracias a la velocidad del vuelo, eso les da más autonomía de vuelo con respecto a un dron de ala rotatoria. El dron de ala fija tiene dos grandes inconvenientes, no pueden mantener el vuelo sobre un punto determinado ya que necesitan estar en constante movimiento, y no pueden realizar el despegue de forma autónoma ya que necesitan una velocidad elevada para empezar el vuelo.

Los drones de ala rotatoria, o hélices, son aquellos en el que el motor transmite el movimiento al ala y esta gira para crear un flujo de aire en dirección perpendicular al ala. Estos drones despegan y aterrizan de forma vertical y tienen una gran estabilidad en el aire pudiendo mantenerse fijos en un punto. El gran inconveniente es la poca autonomía de vuelo que tienen. A este tipo de dron se le sub-clasifica por el número de motores que tiene. Comúnmente tienen dos, tres, cuatro, seis u ocho motores. Cuantos más motores tenga el dron más estabilidad y potencia puede ofrecer, pero, a la vez, menos autonomía de vuelo. [3]

Para el control de cualquier vehículo es importante tener definida una referencia para definir la orientación de este. Para definir la orientación de aeronaves, embarcaciones y satélites se suelen usar los ángulos de navegación, que son los que se usan en este proyecto. Son tres estos ángulos de navegación: Yaw, Pitch y Roll. [4]

A los drones de ala rotatoria de 4 motores o más, al tener dos ejes de simetría perpendiculares, no se les pueden determinar los ángulos de navegación en función de la estructura. Los ángulos de navegación se determinan en función de la posición de la Unidad de Medida Inercial (IMU, en inglés).

2.2 Unidad de Medida Inercial. IMU (Inertial Measurement Unit)

Una IMU es un dispositivo electrónico que contiene dos sensores, un acelerómetro y un giroscopio. Con los datos de estos dos sensores junto a unos cálculos se puede obtener la posición actual del dron.

Los módulos con IMU suelen indica la dirección de dos de los ejes, el tercero es el eje z y este va perpendicular al módulo. La IMU se sitúa se manera que sus ejes coincidan con los ejes de la estructura donde va situada. [5]

Figura 2.1. Fotografía de un módulo electrónico GY-521 (IMU)

2.3 Movimientos de un dron de ala rotatoria y cuatro motores

Una vez se han establecido los ángulos de navegación del dron, dados por la IMU, ya se pueden asignar los movimientos de los motores para el control de estos ángulos de navegación. Siguiendo la enumeración de los motores de la imagen superior izquierda de la figura 2.2:

Los motores 1 y 3 giran siempre en sentido contrario a las agujas del reloj y los motores 2 y 4 en el sentido de las agujas del reloj, para poder controlar la rotación del eje Yaw. Para girar sobre el eje Yaw hay que aumentar la velocidad de aquellos motores que giren en el sentido deseado de giro.
Para ascender se aumenta la velocidad de todos los motores. A la consigna de ascensión se le llama acelerador, más comúnmente conocido en inglés como Throttle.
El Pitch y el Roll tienen la misma forma de funcionar. Para hacer un giro sobre uno de estos dos ejes se tiene que aumentar la velocidad de los drones del sentido contrario al que se quiere girar.
La figura 2.2 esquematiza muy bien cómo deben girar los motores para el giro de cada eje.

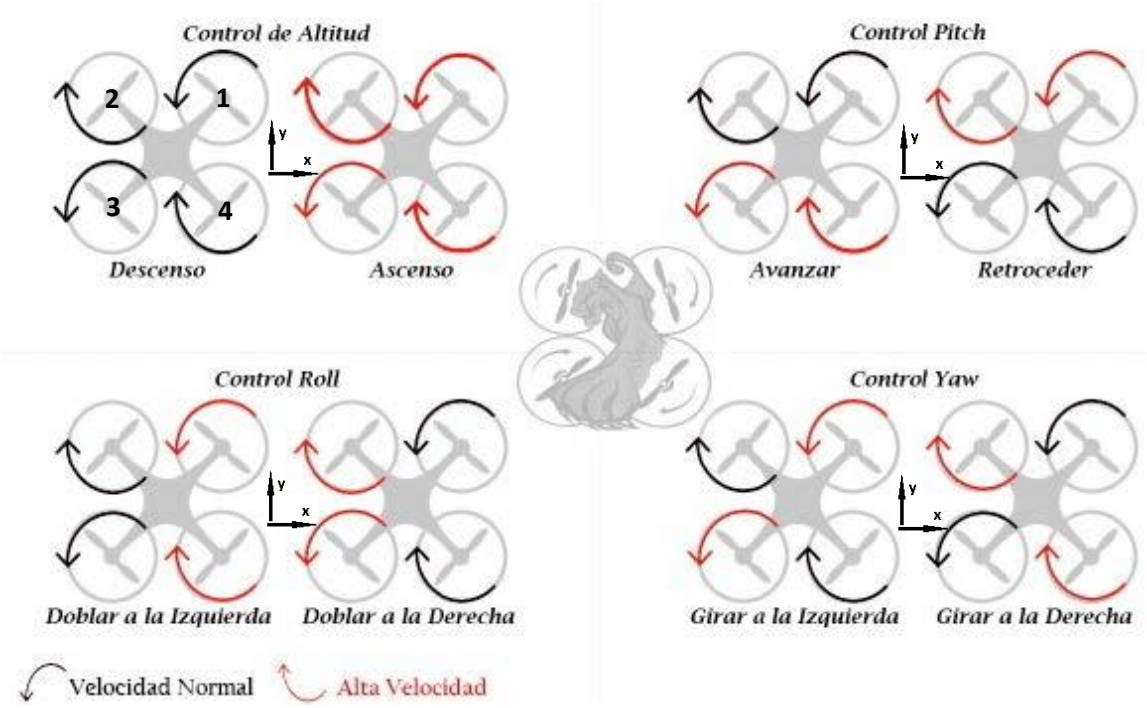

Figura 2.2. Esquema de velocidad de motores para el diferente movimiento sobre los ángulos de navegación [6]

2.4 Motores, hélices y controladores electrónicos de velocidad

Los motores eléctricos son los más baratos y los más ligeros.

Hay dos tipos: los motores con escobillas (brushed) y los motores sin escobillas (brushless).
La diferencia entre ambos motores es funcional y estructural. Para inducir o modificar la velocidad en un motor eléctrico hay que cambiar la polaridad del rotor, para ello el motor con escobillas utiliza escobillas conectadas a un colector. Estas escobillas son las encargadas de conmutar mecánicamente la corriente de las bobinas del motor. La conmutación mecánica de la corriente de las bobinas genera rozamiento que disminuye el rendimiento, genera calor, genera ruido y requiere de un mantenimiento continuo. [7]

Por otro lado, los motores sin escobillas no incorporan ni colector ni escobillas. Un controlador electrónico de velocidad realiza electrónicamente la conmutación de las bobinas.

Los datos técnicos más relevantes de los motores brushless son el valor KV y el tamaño.

KV (rpm/V) se refiere a revoluciones por minuto por voltio suministrado. Se recomienda que cuanto mayor sea el valor KV más pequeñas sean las hélices. Para valores del orden de 2.300KV se recomiendan hélices de 5 a 7 pulgadas. Los drones con estos motores y hélices suelen ser utilizados para carreras. Por otro lado, para valores del orden de 900KV se recomiendan hélices por encima de 10 pulgadas. Los drones con esta configuración suelen ser grandes y no muy rápidos, pero sí muy estables.

El tamaño de los motores se suele indicar en un valor de 4 cifras. Las dos primeras cifras determinan el diámetro (en mm) del estator. Las dos últimas cifras indican la altura (en mm) de los polos electromagnéticos.

Las hélices tienen tres parámetros básicos. El primero es un número que indica la longitud de punta a punta de la hélice, el segundo es el ángulo de torsión que tiene la misma y el tercero es una letra, que puede no estar, para indicar el sentido de giro de las hélices. [8]

Un controlador electrónico de velocidad, de ahora en adelante ESC (Electronic Speed Controller), es un dispositivo electrónico que permite variar la velocidad de giro de un motor eléctrico. Hace de transformador pasando la corriente continua de la batería a señal trifásica para alimentar a los motores. Para controlar la velocidad de giro el ESC recibe las señales PWM y con ellas modula la señal trifásica destinada a los motores. Los ESCs generalmente aceptan una señal PWM cuya amplitud de pulso varía de 1 ms a 2 ms. Cuando el pulso es de 1 ms, el ESC responde apagando el motor conectado a su salida. Una amplitud de pulso de 1.5 ms hará que el ciclo de trabajo sea del 50% lo que moverá el motor a media velocidad. Para que el motor trabaje a su máxima velocidad, el pulso tendrá que ser de 2 ms. Es importante que la frecuencia a la que se envía la señal PWM sea constante sino el ESC no admite la entrada. [9]

De todas las características de los Brushless ESC se pueden resumir las más importantes en:

- Constant current: corriente constante de trabajo máximo.
- Burst current: corriente de pico máximo (la puesta en marcha o durante un breve instante de tiempo).
- BEC: Battery Eliminator Circuit. Provee de una salida rectificada para alimentar un microcontrolador.

- LiPo Cells: número de células de una batería LiPo que soporta (equivale a la tensión máxima: 3,3V / celda).
- Programmable: capacidad de ser programado (vía tarjeta SD o vía microcontrolador) o no
- Weight: peso total de la ESC.

2.5 Batería

Debido al alto consumo de energía de los motores brushless se suelen alimentar de baterías Litio-Polímero (LiPo a partir de ahora) capaces de suministrar corrientes del orden de la decena de amperios. Estas baterías están formadas por celdas de 3,7V y un punto a tener en cuenta a la hora de elegir batería es el número de celdas de esta. Una batería LiPo S3 tiene 3 celdas y una tensión de 3x3,7V = 11,1V. Otro punto importante es la capacidad eléctrica que se mide en tiempo de carga o descarga en amperios/hora de la batería (Ah). La intensidad que puede suministrar se indica con la letra C y está directamente relacionado con la carga eléctrica. Una batería de 20C y 1500mAh puede suministrar una intensidad de 20x1500mAh = 30.000mA. [10]

Hay que tener mucha precaución con estas baterías para que no se descarguen por debajo del 90% de la tensión nominal, porque si no la vida útil de la batería se reduce drásticamente.

2.6 Arduino y placa de desarrollo NodeMCU

Una placa de desarrollo es un dispositivo que cuenta con un microcontrolador reprogramable construido sobre una placa de circuito único, con microprocesador(es), memoria, entrada/salida (E/S) y las características requeridas en un ordenador funcional.

La placa NodeMCU es una placa de desarrollo basado en el ESP8266, un chip Wifi de bajo coste que permite la conexión WiFi y es compatible con el entorno de desarrollo integrado de Arduino [11]. Este entorno de desarrollo se utiliza para escribir y cargar programas en placas compatibles con Arduino.

La estructura básica de cualquier programa Arduino es la siguiente:

```
// Declaración de librerías y variables generales:
.
.
.

void setup() {
  // Código de configuración que se ejecuta una vez:
  .
```

```
  .
  .
}

void loop() {
  // Código principal que se ejecuta repetidamente:
  .
  .
  .
}
```

El código empieza declarando las librerías a usar junto con las variables, aunque las variables se podrían declarar junto al código de configuración. Estas librerías son trozos de código, hechos por usuarios de Arduino, que facilitan mucho la programación. En la parte del *'setup'* se introduce el código de configuración o preparación del programa, el cual solo se ejecutará una sola vez al encender la placa de desarrollo. El código principal del programa se introduce en la parte del *'loop'* y este se ejecuta después del código de configuración repetidamente e indefinidamente mientras la placa de desarrollo esté alimentada. [12]

Los pines digitales, analógicos y de alimentación que nos aporta la placa NodeMCU y que se pueden usar son los siguiente:

- 9 pines digitales numerados del D0 al D8.
- 1 pin analógico numerado A0 que admite un rango de valores de 0 a 3,3V con una resolución de 10-bit.
- 3 pines de salida de 3,3V.
- 1 pin de alimentación de 5V.
- 5 pines de tierra GND.

El resto de pines de los que dispone la placa son para uso interno de esta.

La placa NodeMCU tiene dos pares de salidas con comunicación serial(UART), aunque un par no se puede usar por estar conectado a la memoria Flash y el otro par no se recomienda porque es el que usa la NodeMCU al conectar el USB.

La modulación de ancho de pulso, de ahora en adelante PWM, consiste en activar una salida digital durante un tiempo y mantenerla apagada durante el resto. En esta modulación se mantiene constante la frecuencia, mientras que se hace variar la anchura del pulso. La proporción de tiempo que está encendida la señal, respecto al total del ciclo, se denomina "Duty cycle", y generalmente se expresa en tanto por ciento.

Una conexión I2C es un protocolo de comunicación serial. Es un bus con múltiples maestros, lo que significa que se pueden conectar varios chips al mismo bus y que todos ellos pueden actuar como maestro. La metodología de comunicación de datos del bus I2C es en serie y sincrónica. Una de las

señales del bus marca el tiempo (SCL: pulsos de reloj) y la otra se utiliza para intercambiar datos (SDA). Se necesita conectar los puertos SCL de los distintos dispositivos entre sí para sincronizarse y los puertos SDA para el intercambio de datos.

La comunicación en serie UART toma bytes de datos y transmite los bits individuales de forma secuencial. En el destino, un segundo UART re-ensambla los bits en bytes completos.

Por otro lado, la placa NodeMCU dispone de dos *watchdogs*, uno por software y otro por hardware. Un *watchdog* es un temporizador interno de la unidad de control (UC), que provoca un reset del sistema en caso de que este se haya bloqueado. El *watchdog* de software tiene un temporizador de 2,3s y se puede desactivar, sin embargo, el *watchdog* hardware (de 8s) no se puede desactivar. [14]

La placa de desarrollo también dispone de una memoria Flash donde poder guardar información. Para ello se usa el sistema SPIFFS (SPI Flash File System) que es un sistema de archivos diseñado para funcionar en memorias flash conectadas por SPI en dispositivos embebidos y con escasa cantidad de RAM, como el ESP8266. El IDE de Arduino puede dar soporte al sistema SPIFFS del ESP8266 de forma sencilla mediante la instalación del plugin ESP8266 FS para Arduino. [15]

2.7 Controlador PID

Para la estabilidad del dron un controlador PID es fundamental, sin el controlador no se puede automatizar la estabilidad de vuelo. Un controlador PID es un mecanismo de control lazo cerrado o realimentado en el que su objetivo es ajustar la salida del sistema en el que pertenece hasta conseguir el valor de salida deseado. Para ello el controlador PID calcula la diferencia entre la salida del sistema y la salida deseada, y con esa diferencia o error suma las diferentes acciones del actuador para la corrección del error. La salida del controlador PID se manda a los actuadores como motores, resistencias, LEDs, electro válvulas, etc. Estas acciones son las que dan nombre al controlador, la acción proporcional, la acción integral y la acción derivativa. El valor proporcional depende del error actual, el integral depende de los errores pasados y el derivativo es una predicción de los errores futuros. [16]

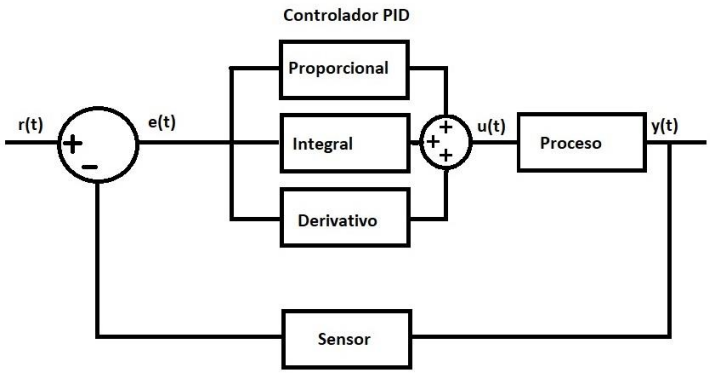

Figura 2.3. Diagrama de bloques de un controlador PID

La parte proporcional consiste en el producto entre el error y una constante proporcional Kp:

$$\boldsymbol{P = e(t) * K_p} \qquad \textbf{(Eq. 2.1)}$$

La parte integral calcula la integral de la señal de error e(t). La integral se puede ver como la suma o acumulación de la señal de error en el tiempo.

$$\boldsymbol{I = K_i \int_0^t e(t) * dt} \qquad \textbf{(Eq. 2.2)}$$

La desventaja del uso de la parte integral en un controlador PID o un controlador PI viene dado por las limitaciones de los actuadores. Si en algún momento durante el proceso los actuadores se encuentran limitados hará que durante ese tiempo tengamos un error constante de salida. Ese error constante dispararía el valor de la parte integral hacía valores excesivos, que deteriorarán la respuesta transitoria del sistema. A este efecto se le denomina comúnmente 'Wind-up'. [17]

La parte derivativa es proporcional a la derivada de la señal de error e(t). La parte derivativa es usada para evitar sobre-pulso y oscilaciones en torno a la referencia. Estas se producen cuando el sistema está corrigiendo el error de salida a gran velocidad hacia el punto de referencia. Cuando llegue al punto de referencia de salida el actuador tendrá una inercia que generará sobre-pulsos. Para evitar este problema, el controlador debe reconocer la velocidad a la que el sistema se acerca a la referencia para poder frenarle con antelación a medida que se acerque a la referencia deseada y evitar que la sobrepase.

$$\boldsymbol{D = K_d * \frac{de}{dt}} \qquad \textbf{(Eq. 2.3)}$$

Hay varios métodos de ajuste de los parámetros Kp, Ki y Kd. El método ideal es con el conocimiento de las ecuaciones de la planta o del sistema controlado. Con este conocimiento se puede calcular el valor exacto de las constantes del PID.

El resto de métodos de ajustes son empíricos. El más extendido es el método Ziegler-Nichols. Este método permite definir las ganancias proporcional, integral y derivativa a partir de la respuesta del sistema en lazo abierto o a partir de la respuesta del sistema en lazo cerrado. El sistema en lazo abierto es método de sintonización que se adapta bien a los sistemas que son estables en lazo abierto y que presentan un tiempo de retardo desde que reciben la señal de control hasta que comienzan a actuar. El sistema en lazo cerrado no requiere retirar el controlador PID del lazo cerrado. En este caso solo hay que reducir al mínimo la acción derivativa y la acción integral del regulador PID. Como se ha podido observar para el dron el mejor sistema es el de lazo cerrado. El ensayo en lazo cerrado consiste en

aumentar poco a poco la ganancia proporcional hasta que el sistema oscile de forma mantenida ante cualquier perturbación. Esta oscilación debe ser lineal, sin saturaciones. En este momento hay que medir la ganancia proporcional, llamada ganancia crítica o Kc, y el periodo de oscilación Tc en segundos. Una vez medidos estos dos valores, se pueden calcular los parámetros del controlador PID. [18]

El último método es el método manual. Se varían los valores de los parámetros PID hasta que la respuesta del sistema es suficientemente buena para el usuario. [19]

2.8 Receptor GPS

El Sistema de Posicionamiento Global (GPS) utiliza las señales enviadas por los satélites para determinar con precisión su posición en la Tierra. Para determinar la posición tridimensional, el módulo GPS localiza automáticamente como mínimo cuatro satélites de la red, de los que recibe unas señales indicando la identificación y hora del reloj de cada uno de ellos. En base a estas señales, el aparato sincroniza su propio reloj con el tiempo del sistema GPS y calcula el tiempo que tardan en llegar las señales al equipo, de tal modo mide la distancia al satélite. Mediante triangulación calcula su propia posición. [20]

Un módulo GPS recibe información como la latitud, longitud, altitud, hora UTC, etc. de los satélites en forma de cadena NMEA. Esta cadena necesita ser analizada para extraer la información que se quiera usar.

2.9 Comunicación Wifi y protocolo de comunicación

El Wifi es una forma de comunicación inalámbrica mediante modulación de ondas electromagnéticas u ondas de radiofrecuencia de baja potencia. El Wifi utiliza generalmente el espectro de ondas de 2,4 GHz. Los dispositivos habilitados con Wifi pueden conectarse entre sí o a Internet a través de un punto de acceso de red inalámbrica. El estándar Wifi está diseñado para interconectar dispositivos a la red a distancias reducidas, y cualquier uso de mayor alcance está expuesto a un excesivo riesgo de interferencias. Para proteger la comunicación entre dispositivos se utilizan protocolos de cifrado de datos tales como el WEP, el WPA o el WPA2, que se encargan de codificar la información transmitida para proteger su confidencialidad. [21]

TCP/IP es un conjunto de protocolos que permiten la comunicación entre los dispositivos pertenecientes a una red.

Para la comunicación en redes se usan los protocolos más extendidos como son el protocolo TCP/IP. El protocolo TCP define la manera en que los datos son fragmentados en secciones de información manejable o en paquetes, que luego son enviados individualmente a través de la red de Internet. Esto permite establecer una conexión y el intercambio de datos entre dos anfitriones. Mientras que el protocolo IP controla el recorrido de los paquetes hasta su destino, como si fuera una especie de sistemas de direcciones basados en los números IP periódicos. Este protocolo lleva los datos a otras máquinas de la red. [22]

Un servidor Web es un programa que utiliza el protocolo HTTP (Hypertext Transfer Protocol) generando una respuesta y sirviendo los archivos que forman páginas Web a los usuarios, en respuesta a sus solicitudes, que son enviados por los clientes HTTP desde sus dispositivos. Siguiendo las capas de protocolo primero va el protocolo IP, luego el TCP y por encima el HTTP. [23]

2.10 Android

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de apps para Android. Para su uso es necesario tener conocimientos previos en lenguaje de programación java, o en Kotlin, y en lenguaje xml.

Cada módulo de la app contiene las siguientes carpetas: manifests: contiene el archivo AndroidManifest.xml, java: contiene los archivos de código fuente Java, incluido el código de prueba de JUnit y res: contiene todos los recursos sin código, como diseños XML, strings de IU e imágenes de mapa de bits. [24]

Al empezar un proyecto nuevo en Android Studio el asistente de creación de proyectos guía al usuario por las distintas opciones de configuración del nuevo proyecto Android Studio.
Primero propone una serie de plantillas para la primera actividad. Seguidamente se introduce el nombre del proyecto, el paquete java para nuestras clases (se rellena automáticamente aunque se puede modificar), la ruta del ordenador donde guardar el proyecto, el lenguaje de programación (java o Kotlin) y la versión mínima SDK. El sistema operativo Android se va actualizando con el tiempo y cada dispositivo tiene una versión de este sistema operativo. Cuanto más nuevo sea el dispositivo más actual será la versión Android que admita. La versión mínima SDK que da a escoger el asistente de configuración de nuevos proyectos Android se refiere a la versión menos actualizada de Android que debe tener un dispositivo para poder hacer uso de la aplicación.

Al finalizar las opciones de configuración iniciales Android Studio crea de forma autónoma toda la estructura del proyecto y los elementos necesarios para el funcionamiento del proyecto. Destaca el archivo AndroidManifest.xml que describe información esencial de la aplicación, los archivos .xml que definen la interfaz de las diferentes pantallas y las clases java creadas para el desarrollo de la

funcionalidad de la aplicación. Cada actividad, o pantalla, que se muestra en la aplicación tiene mínimo un archivo .xml y una clase definida.

3 Métodos

3.1 Desarrollo aplicación móvil

Para el desarrollo de la aplicación móvil se ha usado el entorno de desarrollo gratuito Android Studio. En Android Studio se puede desarrollar en los lenguajes de programación Java o Kotlin, junto con el diseño de la interfaz de usuario en .xml. Al crear un nuevo proyecto de Android Studio se elige usar el lenguaje Java para la programación, partir de una actividad vacía y se elige, para la aplicación, la versión SDK API 25.

A la aplicación se le ha dado el nombre de DroneControl y se han creado 3 pantallas en la aplicación: la pantalla de inicio, la pantalla de las palancas de mando y la pantalla de petición de archivos al dron.

3.1.1 Pantalla de inicio

Figura 3.1. Pantalla de inicio de la aplicación DroneControl

La comunicación desde la aplicación móvil a la placa de desarrollo NodeMCU, el microcontrolador del dron, se hace a través de peticiones URL configurando la placa como un servidor. La primera pantalla se utiliza para introducir la IP que genera la placa como servidor y 4 parámetros de vuelo. Esos 4 parámetros son las consignas máximas de movimiento de los ejes y la consigna máxima del Throttle.

Se establece la consigna máxima del Throttle en valor de porcentaje. La consigna Throttle que se puede enviar va desde 0 hasta 100, este valor, por defecto, se establece en 100. Este valor representa la fuerza, en porcentaje, que puede hacer el dron para empujar el aire

La consigna máxima de Yaw se establece desde 5 hasta 20. Este valor, por defecto, se pone en 10. El valor que se envía de Yaw representa la velocidad angular que se quiere para el eje Yaw, expresado en º/s.

La consigna máxima de Pitch y de Roll se establece desde 5 hasta 30. Este valor, por defecto, se pone en 5. El valor que se envía de Pitch y de Roll representa la posición angular que se quiere para cada uno de los ejes, expresado en º.

Una vez introducidos todos los valores se clica el botón 'Vuelo' para ir a la pantalla desde donde se mandarán las consignas al dron. En caso de que los valores de consignas máximas salgan de los límites establecidos o que se deje en blanco el campo IP, aparece un texto encima del botón 'Vuelo' informando de que los campos se han de rellenar con los intervalos señalados.

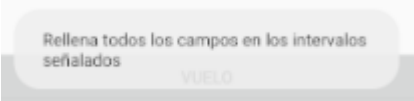

Figura 3.2. Mensaje de la pantalla de inicio de la aplicación DroneControl al salirse de los intervalos indicados

3.1.2 Pantalla de palancas de mandos

Para un buen control del dron, esta pantalla simula dos palancas de mandos como las de los mandos de radiocontrol que se usan habitualmente en el manejo de drons.

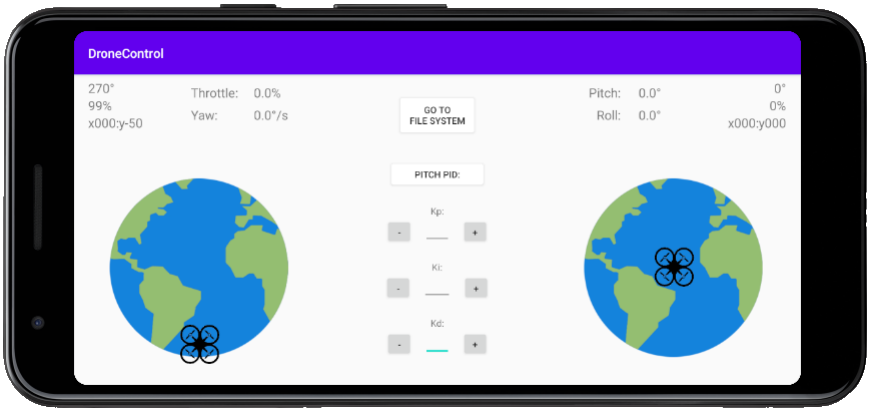

Figura 3.3. Pantalla de palancas de mandos de la aplicación DroneControl

Las esferas con el dibujo de la tierra son los límites de la palanca de mandos y las siluetas de los drones representan las posiciones de las palancas de mandos. Para el desarrollo de las palancas de mandos se ha usado una librería llamada *'virtual-joystick-android'* [25]. Con la ayuda de la librería se puede mover la palanca de mandos por dentro del círculo del mundo sin que salga de él y obtener la posición de la palanca con respecto el centro del círculo, tomando el centro como origen de coordenadas. Se muestran en los extremos superiores, tanto izquierdo como derecho, la posición cartesiana de la palanca, el ángulo de la palanca y la distancia de la palanca al centro de la esfera expresada en porcentaje, tomando como límite los bordes de las esferas.

La palanca izquierda es para el control del Throttle y del Yaw, el eje X para el Yaw y el eje Y para el Throttle. Las consignas de Throttle se envían en porcentaje de fuerza escogida para que los motores empujen el aire. Se configura para que la palanca, en el punto inferior del eje Y de la esfera, envíe una consigna de 0, y para el eje superior Y, el valor indicado en la pantalla anterior como consigna máxima. Todos los valores intermedios se escalan según la consigna máxima de Throttle indicada en la pantalla de inicio. Para el Yaw se toma el centro de la esfera como el origen del eje. Si se desplaza la palanca hacia la izquierda respecto el origen se envían consignas negativas, y si se desplaza hacia la derecha se

envían consignas positivas. Las consignas parten de 0 en el origen escalando hasta el valor máximo, establecido en la pantalla de inicio, al borde de la esfera.

La palanca derecha se utiliza para el control del Pitch y del Roll, el eje X para el Roll y el eje Y para el Pitch. Tanto para el Pitch como para el Roll se toma el centro de la esfera como el origen de coordenadas, e igual que en Yaw, las consignas parten de 0 en el origen escalando hasta el valor máximo, establecido en la pantalla de inicio, al borde de la esfera.

Un problema que existe al usar el móvil, en lugar de un mando radio control, es la sensibilidad de los dedos con las palancas, es decir, sin mirar la pantalla el usuario no puede saber si se ha salido de los márgenes de la palanca de mandos o si su dedo se ha desplazado hacía una dirección que no quiere. Por ello, para el eje X de la palanca derecha (Yaw) y para el eje X e Y de la palanca de la izquierda (Pitch y Roll), se ha establecido un margen de un 10% de seguridad respecto al origen del eje. Por ejemplo, si solo se quiere accionar el Throttle se tiene un margen del 10% para mover la palanca en referencia al eje central de Throttle sin que afecte al valor de Yaw. En la parte superior de cada esfera se ha añadido el valor de los dos parámetros, de cada esfera, a enviar.

La actualización de la nueva posición de la palanca la hace cada 20 milisegundos, mientras el usuario las esté tocando. Cada vez que actualiza la posición de las palancas calcula el valor de Throttle, Yaw, Pitch y Roll, y se envía una nueva petición URL a la IP introducida en la pantalla de inicio con los parámetros actualizados.

Desde la pantalla actual también se pueden actualizar las constantes Kp, Ki y Kd de los PIDs, de Pitch, de Roll y de Yaw. Como se puede observar en la Figura 3.3 solo se puede modificar una Kp, una Ki y una Kd. Para saber a cuál de los tres PIDs se están cambiando las constantes se debe observar el botón que está justo encima de Kp. Este botón informa a que PID pertenecen esas constantes y a que PID se están modificando. Pulsando el botón, que informa a que PID pertenecen las constantes, cambiamos de PID e informa de las constantes del PID actual. Las constantes se pueden modificar de dos formas distintas. Una opción es, si se clica encima de la constante a modificar aparece un teclado desde donde se puede introducir el nuevo valor de la constante seleccionada. La otra, a cada lado de cada constante se encuentran dos botones, uno para sumar valor y el otro para restar valor. Para que los valores de las constantes PID seleccionada se actualicen hay que mover alguno de las dos palancas. Solo se actualizarán los valores del PID seleccionado al mover la palanca.

Las peticiones URL tienen la siguiente forma:
http://<Dirección>/<camino>?<parametro1>=<valor1>&<parametro2>=<valor2>...

Como ya se ha explicado, cada 20 milisegundos, si el usuario está tocando alguna de las palancas, se envía una nueva petición URL a la IP introducida en la pantalla de inicio con los parámetros actualizados y las constantes indicadas. Esta es la petición URL que se genera:

```
"http://"+urlip+"/dronePar?Pitch="+Pitch+"&Roll="+Roll+"&Throttle="+Throttle+"&
Yaw="+Yaw+"&Kp="+Kp+"&Ki="+Ki+"&Kd="+Kd+"&PIDSelected="+PIDSelected
```

Donde *urlip* es el valor IP dado en la pantalla principal, *PIDSelected* indica el PID seleccionado para actualizar las constantes y el resto son los valores de las consignas y de las constantes.

Encima del botón de selección del PID hay otro botón llamado 'Go to file system'. Al clicar en este botón se abrirá otra pantalla de la aplicación.

3.1.3 Pantalla para hacer peticiones de lectura del sistema de archivos de la NodeMCU

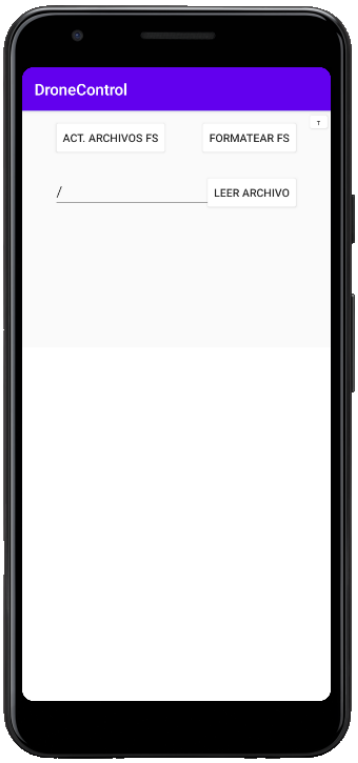

Figura 3.4. Pantalla File system de la aplicación DroneControl

Esta pantalla es usada para hacer peticiones URL sobre el sistema de archivos de la NodeMCU y para la lectura de las respuestas.

En la parte superior de la pantalla se encuentran los botones para las peticiones al dron y la parte inferior es una extensión WebView que sirve para mostrar páginas web, en nuestro caso para visualizar la respuesta del servidor a las peticiones URL. En esta pantalla todas las peticiones se hacen a *urlip* que es el valor IP dado en la pantalla principal.

El botón de la parte superior izquierda envía la petición URL `http://"+urlip+"/directoryFS`. La respuesta que se espera del servidor es una lista del directorio de archivos existentes en la NodeMCU, es decir, el nombre de todos los archivos guardados en la memoria flash, y esta lista aparece en la mitad inferior de la pantalla.

En la parte superior derecha encontramos el botón llamado 'Formatear FS'. Este envía la petición URL `http://"+urlip+"/formatFS`. El dron cuando recibe esta petición borra todos los datos encontrados en el sistema de archivos de la memoria flash.

Debajo de los dos botones anteriores se encuentra un editor de texto, que empieza con el carácter '/', y un botón llamado 'Leer archivo'. La funcionalidad de estos dos elementos es para visualizar el contenido de uno de los archivos del directorio de archivos de la NodeMCU. Se escribe el nombre del

archivo deseado en el editor de texto y se clica el botón 'Leer archivo'. Al presionarlo, envía la petición URL `http://"+urlip+"/readFile?file="+urlArchivo`, donde *urlArchivo* es el texto introducido en el editor de texto. Si el archivo existe aparece, en la mitad inferior de la pantalla, el contenido del archivo indicado.

En la esquina superior derecha hay un botón muy pequeño con una T escrita. Este botón sirve solo a modo de pruebas, para comprobar que la aplicación tiene los permisos para conexión a internet y comprobar que la extensión WebView funciona correctamente. Si se clica el botón aparece en la parte inferior de la pantalla la página principal de google.

3.1.4 Configuración general de la aplicación

Una parte muy importante a la hora de desarrollar una aplicación en Android Studio es el archivo AndroidManifest.xml. En este archivo, entre otras cosas, se da permiso de acceso a internet a la aplicación, se da el nombre de la aplicación (en este proyecto es DroneControl), se indica que imagen tiene el icono de acceso a la aplicación, cual es la pantalla principal de la aplicación y se configura la pantalla de palancas de mandos para que siempre este en horizontal.

El código principal se puede encontrar en el Anexo A.

3.2 Montaje del dron

3.2.1 Materiales para montaje del dron

Se escoge una estructura (o Frame) para 4 motores que sea robusta pero no demasiado pesada, que sea lo suficientemente grande para albergar todos los componentes y de bajo coste. Un F450 Frame de 4 ejes con una distancia entre ejes de 450mm.

Como IMU, se usa el módulo electrónico GY-521. Este módulo se basa en el dispositivo MPU-6050 que contiene un acelerómetro y un giroscopio, cada sensor con 3 ejes. Se usa el eje Z como eje Yaw, el eje Y como eje Roll y el eje X como Pitch.

El estudio de la correcta elección de los motores, las hélices y los ESCs podría suponer todo un proyecto debido a la gran variedad de motores, hélices y ESCs que hay. Al no centrarse este proyecto en este estudio se seguirán las recomendaciones del distribuidor de la estructura del dron elegida [26].

Por lo que los componentes escogidos son los siguientes:

- Motores: Motor sin escobillas XXD A2212 1000KV, eficiencia de corriente: 4 - 10 A, voltaje de entrada: 2-4 células de batería de Litio-Polímero.

- ESCs: Brushless ESC 30A, corriente de pico 40A hasta 10 segundos, voltaje de entrada: 2-4 células de batería de Litio-Polímero. BEC de 5V.
- Hélices: Hélices 1045 L / R. 10 pulgadas de diámetro total (254mm) y 4,5 pulgadas de paso (114mm).

La batería escogida es una batería de LiPo recargable, 3S (11,1V), 4500mAh, 45C.

Para facilitar la alimentación de todos los motores y de la placa de desarrollo se decide añadir al dron una placa de distribución de energía. Esta placa tiene una entrada de alimentación, 6 salidas con la misma tensión que en la entrada y dos salidas rectificadas de 5V y 12V. Se usa la salida de 5V rectificada para alimentar la placa de desarrollo.

La placa de distribución de energía utilizada es la PDB-XT60, de ahora en adelante PDB.

Para controlar el nivel de la batería se configura un divisor de tensión en la entrada analógica de la placa de desarrollo.

Los materiales utilizados para el montaje del hardware del dron son los siguientes:

- Placa de desarrollo: NodeMCU Lua Lolin V3.
- IMU: Modulo GY-521 con el sensor MPU-6050.
- Receptor GPS: Modulo NEO-6M con receptor GPS.
- Motores: Motor sin escobillas A2212 1000KV
- Transformador y variador para motor: Brushless ESC 30A
- Placa de distribución de energía: PDB-XT60
- Pantalla: Pantalla LCD I2C 1602
- Resistencias: una resistencia de 51kΩ y una resistencia de 20kΩ
- Batería: Batería de LiPo recargable, 3S (11,1V), 4500mAh, 45C
- Interruptor: Palanca basculante de conmutación de 12V y 20A con led integrado

El resto de materiales utilizados para el montaje del dron son los siguientes:

- Hélices: Hélices 1045. 10 pulgadas de diámetro total (254mm) y 4,5 pulgadas de paso (114mm)
- Estructura: F450 Frame de 4 ejes. Distancia entre ejes de 450mm
- Conectores: Conectores macho - hembra para PCB
- Tornillos amortiguadores: Tornillo M3 8*8 con goma amortiguadora en el centro
- PCB: Placas de circuito impreso de doble cara. Una de 8cm*12cm*0,16cm y otra de 7cm*9cm*0,16cm y otra de 5cm*7cm*0,16cm. Paso de orificios de 2,54mm
- Conectores: Conectores de pala con terminal de crimpado, macho - hembra

- Tornillos y tuercas: Juego de tuercas M3 de nailon macho – hembra con separador hexagonal
- Cables: Pares de cables rojo y negro con cabezal para enchufe macho - hembra de 1,4mm de diámetro externo. Y otros cables rojos y negros de 3mm de diámetro externo
- Alambre para soldar: Alambre de Soldadura sin Plomo de 0,8mm de diámetro
- Bridas de nylon de 2,5mm*100mm y de 2,5mm*120mm

Materiales usados para la estructura de pruebas, usada en el ajuste de las constantes PID:

- Varilla metálica de 6mm de diámetro.
- 4 tornillos de 4,5mm de rosca y 40mm de longitud
- 1 madera 29cm*23cm*2,5cm
- 2 maderas de 6,5cm*17cm*3cm
- 1 conglomerado de madera de 8cm*9cm*1,5cm con agujero en el centro de 4cm*5cm
- 2 tornillos de gancho cerrado M3, de 6mm diámetro de agujero
- 4 alcayatas roscadas M2 20mm*10mm

3.2.2 Esquema de conexionado del hardware

El esquema de conexión del hardware con el material mencionado en el apartado anterior:

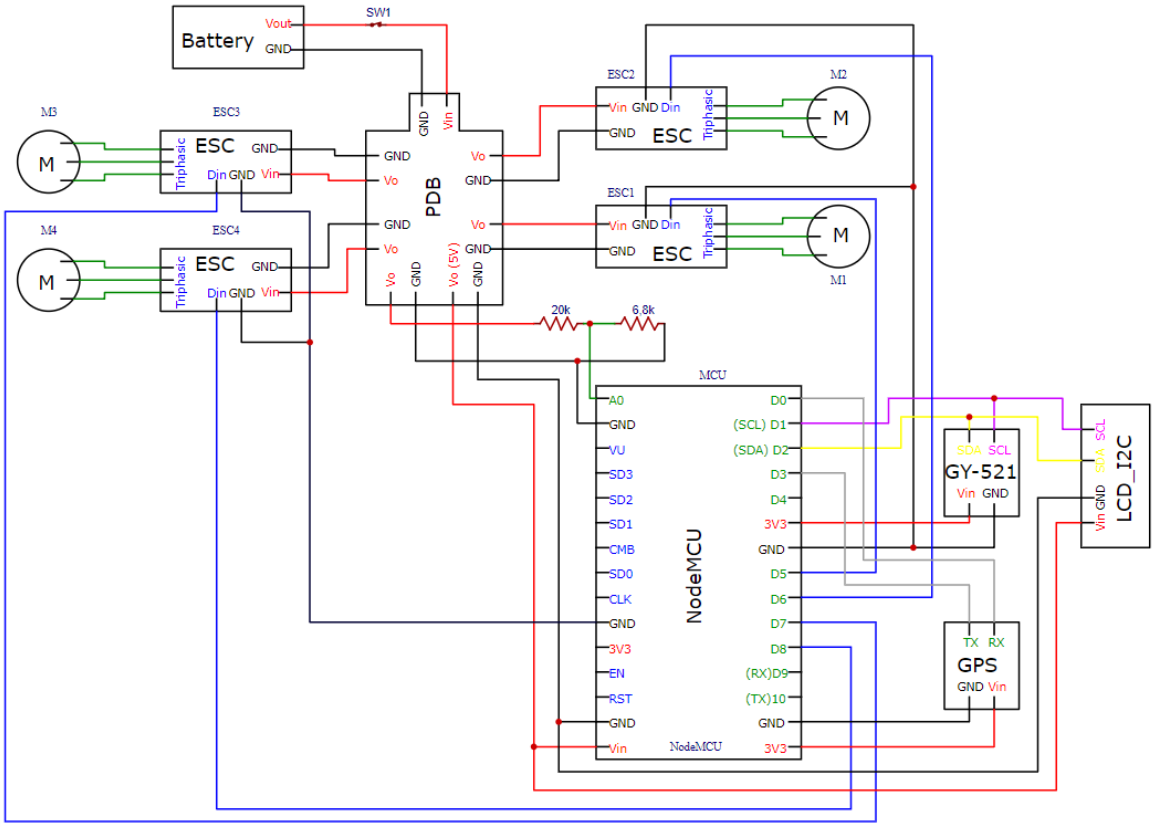

Figura 3.5. Esquema de conexionado del hardware

3.2.3 Proceso de montaje

El montaje se ha hecho de manera que, en caso de rotura de alguna parte, se pueda desmontar y remplazar la parte rota.

3.2.3.1 Preparación del hardware:

En la PDB se suelda 4 cables rojos y 4 cables negros, de 3mm de diámetro, en cuatro salidas no moduladas para alimentar los cuatro ESCs. Se hace lo mismo para la entrada de tensión. En la otra punta de cada uno de estos cables se adhiere un conector de pala macho. A la PDB se sueldan un par de cables, rojo y negro, con cabezal para enchufe hembra, de 1,4mm de diámetro, el rojo en la salida modulada de 5V y el negro en la masa de al lado de esta. Se hace lo mismo con la salida marcada como Vcc. Una vez está soldada la entrada y las salidas necesarias, se atornilla la PDB a la PCB de 5cm*7cm*0,16cm. La PDB tiene 4 orificios en sus 4 esquinas. Se coloca la PDB cruzada encima del centro la PCB, de forma que sobresalga la entrada de tensión a la PDB. Se marcan las 4 esquinas de la PDB encima de la PCB para realizar 4 agujeros que fijan, mediante tornillería, los dos elementos.

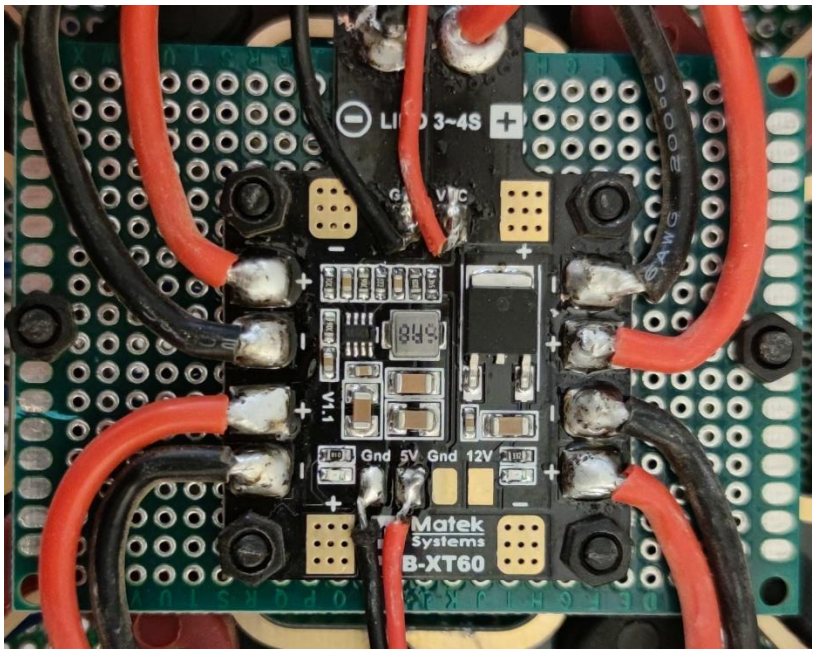

Figura 3.6. Montaje de la placa PDB

En los extremos de los cables de los motores se adhieren conectores de pala macho. En los dos cables de alimentación de cada ESC y en los tres de salida a los motores se adhieren conectores de pala hembra.

Todas las conexiones entre la NodeMCU y los módulos se hacen siguiendo el esquema de conexionado del hardware (figura 3.5) y por la parte posterior de la PCB a la que se conectan. Se sitúa la PCB más

grande (8cm*12cm*0,16cm) en posición vertical y en la parte inferior, posicionada de forma horizontal, encima de esta, se coloca la placa NodeMCU con los conectores hembra. Se suelda a la PCB los conectores hembra, de esta forma se tiene conectada la NodeMCU a la PCB con los conectores de por medio. Al lado de los pines, de la NodeMCU, Vin y GND se sueldan un par de cables, rojo y negro, con cabezal para enchufe macho, de 1,4mm de diámetro. Junto a la NodeMCU se sueldan las resistencias del divisor de tensión, se conecta a la NodeMCU, y se sueldan un par de cables, rojo y negro, con cabezal para enchufe macho, de 1,4mm de diámetro, conectados al divisor de tensión. En esta PCB se sueldan los conectores hembra, para la conexión de la IMU, de tal forma que el centro de la IMU queda en el centro de la PCB. El eje Y de la IMU tiene que quedar paralelo al lado largo de la PCB y el eje X de la IMU tiene que quedar paralelo al lado corto de la PCB. Con un tornillo y goma antivibración se fija la IMU a la PCB. Mediante cables, de 1,4mm de diámetro, conectaremos la IMU a la NodeMCU. En la esquina opuesta a la NodeMCU se sueldan los conectores hembra para PCB donde va conectado el modulo GPS. Se fija el otro extremo del GPS a la PCB mediante un tornillo y se conecta el GPS a la NodeMCU. Cerca del módulo GPS se sueldan, en la PCB, 4 tríos de conectores macho, para PCB, donde van conectados los ESCs. Se conectan, estos 4 tríos de conectores, a la NodeMCU teniendo en cuenta que estos van conectados a los ESCs. Junto a la IMU se sueldan 4 conectores macho, para PCB, que van conectados a la pantalla LCD.

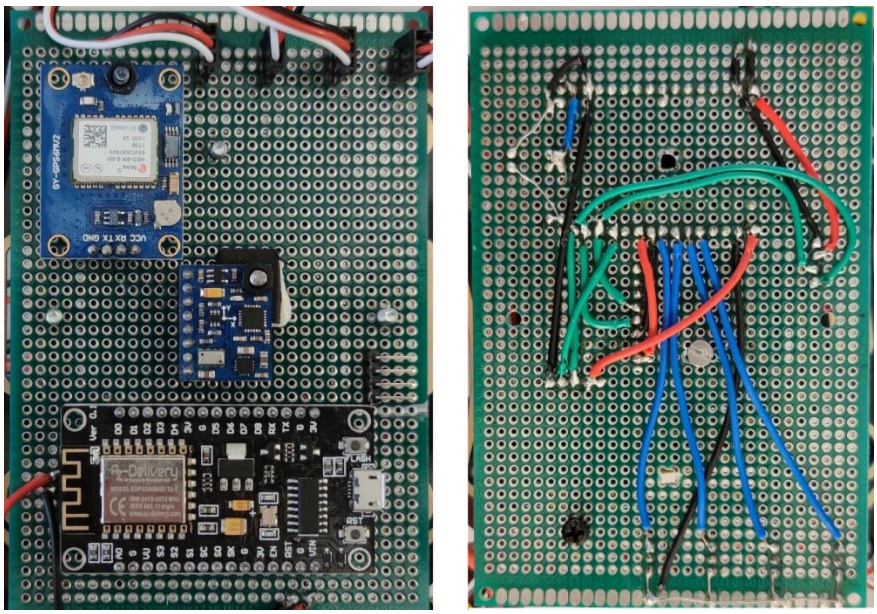

Figura 3.7. Montaje de la PCB grande. Izquierda parte anterior. Derecha parte posterior

La PCB mediana de 7cm*9cm*0,16cm se sitúa encima de la PCB grande y se fija a esta mediante tornillos. En esta PCB se pega la antena del módulo GPS. Esta se situará justo encima del módulo GPS ya que el cable que tiene la antena es muy corto. Al lado de la antena se atornilla la pantalla LCD. Este se conecta a la NodeMCU mediante cables removibles que van conectados a los 4 pines machos soldados en la PCB grande. En esta PCB también se coloca el interruptor del sistema. Se sueldan dos

cables de 3mm de diámetro, uno rojo al positivo y uno negro a masa, al conector macho de la batería. Las otras puntas de estos dos cables van conectados al interruptor. Se suelda otro cable rojo de 3mm de diámetro a la salida de alimentación del interruptor y otro cable negro de 3mm de diámetro. En las otras puntas de estos dos últimos cables se adhieren conectores de pala hembra.

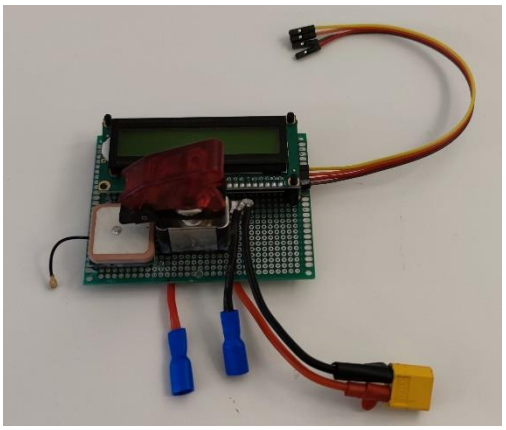

Figura 3.8. Montaje de la PCB mediana

3.2.3.2 Montaje del dron

La estructura F450 Frame es una estructura desmontable. Tiene dos bases, una superior y una inferior, 4 ejes, en los extremos de estos van situados los motores, y 4 patas para suavizar el aterrizaje. La base superior se sitúa encima de los extremos de los ejes y se fija a ellos con tornillos, y la parte inferior se sitúa por debajo de los ejes que también se fija a ellos con tornillos.

Se fija la base superior a los 4 ejes. La base superior tiene 4 orificios. Se aprovechan estos 4 orificios para atornillar la PCB grande a esta base. Para ello se hacen 4 orificios en la PCB, procurando que coincidan estos orificios con los orificios de la base superior del Frame. La IMU de la PCB debe quedar centrada en la base, y el eje Y y el eje X de la IMU deben quedar paralelos a los lados de la base, dejando una configuración del dron de tipo X. Para atornillar los dos elementos se usan los 4 tornillo con la goma amortiguadora, quedando la goma entre la base superior del Frame y la PCB grande. En la parte inferior de la base superior del Frame quedan, sobresaliendo, la parte inferior de los tornillos con goma amortiguadora. Se aprovecha esto para atornillar a ellos la PCB pequeña, haciendo antes los orificios pertinentes en la PCB pequeña. Posteriormente se atornillan a la base inferior del Frame las 4 patas.

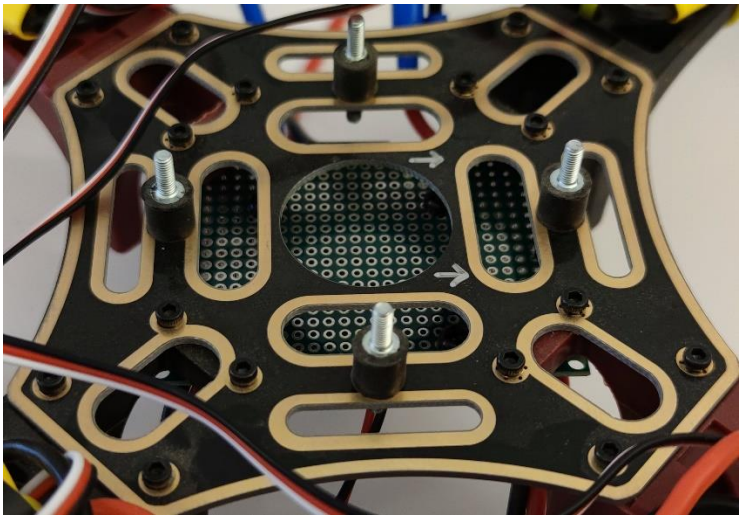

Figura 3.9. Montaje de la base superior con los tornillos con goma amortiguadora y la PCB pequeña

Los motores se atornillan en los extremos de los ejes de Frame donde ya hay unos orificios habilitados para ello. Los ESCs se fijan con unas bridas en los ejes y se conectan a los motores mediante los conectores de pala macho – hembra. Para que los cables de conexión no puedan tocar las hélices durante el vuelo, se fijan estos al eje con bridas. A los motores se les colocan las hélices junto con sus adaptadores.

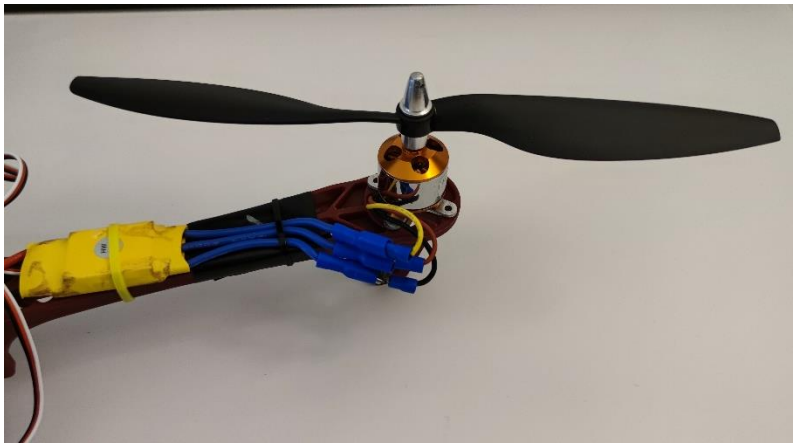

Figura 3.10. Montaje del motor, con la hélice, y el ESC en un eje del Frame

Se enumeran los motores, o los ESCs, según la posición en la que se encuentran en referencia a la IMU.

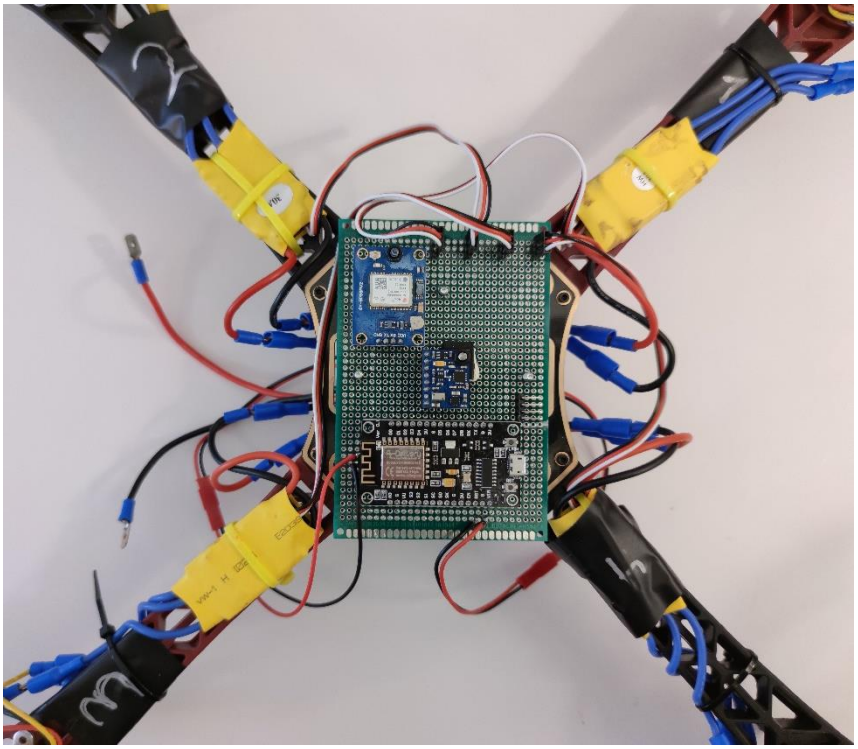

Figura 3.11. Montaje de la PCB grande con los ejes y la base del Frame y los ESCs enumerados

La PCB mediana se coloca encima de la PCB grande, se separa para que no se toquen y se fija con tornillos a la PCB grande. Para ello se utilizan los tornillos de nailon macho – hembra con separador hexagonal. Se conecta la antena con el módulo GPS, se hacen el resto de conexiones pertinentes siguiendo el esquema de conexión del hardware (figura 3.5). La batería va situada encima de la base del Frame inferior.

Figura 3.12. Montaje del dron completado

3.3 Desarrollo del dron

La placa NodeMCU no tiene ninguna salida de modulación de ancho de pulso (PWM) por hardware y en su lugar tiene que hacerlo mediante el software, por lo que se pueden usar las salidas que se quieran con salida de modulación de ancho de pulso. La NodeMCU tampoco tiene salida hardware I2C por lo cual tiene que emularlo por software. Eso significa que podemos emplear I2C con casi cualquier pin de salida. Estas salidas emuladas por software suponen una carga para el procesador.

3.3.1 Preparación entrono de desarrollo (Arduino IDE)

El entorno de desarrollo desde donde se ha programado la placa de desarrollo es el Arduino IDE. Para poder programar el NodeMCU primero hay que descargar el entorno de desarrollo de Arduino desde la página web https://www.arduino.cc/en/software. Una vez descargado e instalado el entorno se tiene que preparar el IDE para poder utilizarlo con la placa de desarrollo NodeMCU. El software para poder utilizar esta placa no forma parte del repositorio estándar del IDE de Arduino, por lo que antes se debe configurar el entorno para poder programarla. Para ello se han seguido los pasos que marca el documento descargado de la página web https://www.az-delivery.de/, desde donde se ha comprado la NodeMCU [27].

En este entorno de desarrollo se importan varías librerías necesarias para el proyecto.

La primera librería que se instala, llamada *'Wire'*, es una librería que permite la comunicación entre la NodeMCU y módulos externos con bus I2C.

```
#include <Wire.h>                    //Declarar la librería Wire

void setup() {
  Wire.begin();                      //Inicializar la librería Wire
  Serial.begin(115200);              //Establece la velocidad, en baudios
                                (bits por segundo), para la comunicación
                                de datos en serie.
}
```

Por otra parte, también se establece la velocidad a la que se comunicara con el puerto serie (USB o pines RX y TX).

3.3.2 Configuración pantalla LCD

En la placa de desarrollo se conecta una pantalla LCD para que el usuario pueda visualizar y seguir el estado del código y del dron. Se emula por software una conexión I2C.

```
#include <LiquidCrystal_I2C.h>

//Se instancia un objeto de la clase LiquidCrystal_I2C (lcd)
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

```
void setup() {
  lcd.init();          //Activar la pantalla
  lcd.backlight();     //Activar la luz de fondo
}
```

Se declara primero la librería llamada *'LiquidCrystal_I2C'*. Seguidamente se da nombre a la pantalla LCD conectada junto con la dirección I2C y las dimensiones de la pantalla. Una vez hecho esto, para visualizar algo en la pantalla, se usa una estructura similar a la siguiente:

```
  lcd.clear();               //Borrar el contenido anterior de la pantalla
  lcd.setCursor(0, 0);       //Indicar la posición del cursor: 0 horizontal y 0 vertical
  lcd.print("Esperando a ");  //visualizar en pantalla desde la posición marcada
  lcd.setCursor(0, 1);
  lcd.print("conectarse...");
```

3.3.3 Conexión de la NodeMCU al Wifi y recepción de parámetros enviados desde el móvil

Para que la placa pueda recibir los parámetros enviados desde el móvil se conecta la placa a la red Wifi creada con el móvil.

```
#include <ESP8266WiFi.h>


//======Subrutina de configuración y conexión Wifi ======//
void init_WifiConnection() {
  WiFi.mode(WIFI_STA);        //Establece la placa como cliente Wifi
  WiFi.begin(ssid, password); //Conectar al Wifi con el ssid y el password indicados

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Esperando a ");
  lcd.setCursor(0, 1);
  lcd.print("conectarse...");

  while (WiFi.status() != WL_CONNECTED) {    //Esperar hasta estar conectado al Wifi
    delay(500);
  }

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(WiFi.localIP());  //Mostrar en pantalla la IP local generada
}
```

La placa, en caso de no poder conectarse al Wifi configurado, se queda esperando en bucle. La librería, llamada *'ESP8266WiFi'*, da las herramientas necesarias para la conexión a internet mediante Wifi. Se muestra en pantalla la IP local generada.

Para recibir las consignas desde el móvil se configura la placa NodeMCU como un servidor web. Para ello se utiliza la librería *'ESP8266WebServer'* y el siguiente código. [28]

```
#include <ESP8266WebServer.h>

ESP8266WebServer server(80);
/*
Se instancia un objeto de la clase ESP8266WebServer (server) y se le da valor al puerto(80)
El servidor responde a los clientes en el puerto 80, que es un puerto estándar para que los
navegadores web hablen con los servidores web.
*/
```

```
//Declaración de variables
.
.
.
//

void setup() {
  server.on("/dronePar", handleParemeters);//La subrutina a ejecutar al solicitar /dronePar
  server.on("/formatFS", formatFS);
  server.on("/directoryFS", directoryFS);
  server.on("/readFile", readFile);
  server.begin();                     //Inicializa el servidor web
  while (consignaThrottle != 1) {     //Se espera hasta recibir el valor 1 en Throttle
    server.handleClient();            //Se "escuchan" las peticiones URL entrantes
    consignaThrottle = argumentThrottle.toFloat();
  }
}

//====== Subrutina de conversión datos de lectura URL ======//
void WifiConnection_GetData()
{
  server.handleClient();
  consignaPitch = argumentPitch.toFloat();   //Se convierte la variable de String a float
  consignaRoll = argumentRoll.toFloat();
  consignaThrottle = argumentThrottle.toFloat();
  consignaYaw = argumentYaw.toFloat();
  PIDSelected = PIDSelectedString.toInt();

  if (PIDSelected == 0) {
    KpPitch = KpString.toFloat();
    KiPitch = KiString.toFloat();
    KdPitch = KdString.toFloat();
  }
  else if (PIDSelected == 1) {
    KpRoll = KpString.toFloat();
    KiRoll = KiString.toFloat();
    KdRoll = KdString.toFloat();
  }
  else if (PIDSelected == 1) {
    Kp_yaw_w = KpString.toFloat();
    Ki_yaw_w = KiString.toFloat();
    Kd_yaw_w = KdString.toFloat();
  }
}

void handleParemeters()
{
  argumentPitch = server.arg("Pitch");       //Se lee el valor enviado del argumento Pitch
  argumentRoll = server.arg("Roll");
  argumentThrottle = server.arg("Throttle");
  argumentYaw = server.arg("Yaw");
  KpString = server.arg("Kp");
  KiString = server.arg("Ki");
  KdString = server.arg("Kd");
  PIDSelectedString = server.arg("PIDSelected");

  server.send(200, "text/plain", "");        //Respuesta del servidor a la petición URL
}
```

La placa de desarrollo, tal y como ha sido configurada, estará esperando recibir la información a través de la dirección URL http://*'la-ip-local-generada'*/dronepar. Los parámetros que se le enviarán desde la aplicación móvil y que la placa leerá, a través de la subrutina handleParemeters, son *Pitch*, *Roll*, *Throttle*, *Yaw*, *Kp*, *Ki*, *Kd* y *PIDSelected*. Dependiendo del valor del parámetro PIDSelected se actualizarán unas constantes para un PID o para otro.

La respuesta del servidor a la petición URL está en blanco. Este paso no se puedo omitir, porque si no se aumentaría hasta en 2 segundos el tiempo de comunicación entre el dron y la aplicación, ya que por defecto espera ese tiempo a obtener una respuesta, lo que dificultaría conseguir una buena estabilidad.

Las siguientes subrutinas a las que se llama en la recepción de peticiones URL se tratarán en el apartado Guardar datos en memoria de la placa.

3.3.4 Obtención de datos del sensor MPU-6050

La configuración en Arduino del módulo GY-521 se ha hecho siguiendo la hoja de datos del MPU-6050. Se emula por software una conexión I2C. Se configura la sensibilidad del giroscopio, la sensibilidad del acelerómetro y el filtro pasa bajos (LPF). Hay que tener en cuenta que cuanto mayor sea la sensibilidad del sensor mayor será la precisión de las lecturas, pero también será mayor la sensibilidad al ruido y mayor puede ser el error. Debido a esto se han escogido unos parámetros intermedios para la sensibilidad. Para el filtro pasa bajos cuanto más se filtre la salida mejor para evitar errores en las medidas, pero cuanto más se filtre más retraso genera en la salida y más retraso en la estabilidad del dron. [29]

```
//======Subrutina de configuración del MPU-6050======//
void conf_MPU() {
  Wire.beginTransmission(MPU6050_address);   //Iniciar comunicación con el MPU-6050
  Wire.write(0x6B);                          //Entrar en la configuración
  Wire.write(0x00);                          //Reiniciar el sensor para configurarlo
  Wire.endTransmission();                    //Cerrar comunicación con el MPU-6050
  //Por cada parámetro a configurar abrimos y cerramos la comunicación
  Wire.beginTransmission(MPU6050_address);
  Wire.write(0x1B);                          //Se abre configuración del giroscopio
  Wire.write(0x08);                          //Sensibilidad giroscopio: 0x08 => +/- 500°/s
  Wire.endTransmission();

  Wire.beginTransmission(MPU6050_address);
  Wire.write(0x1C);                         //Se abre configuración del acelerómetro
  Wire.write(0x10);                         //Sensibilidad acelerómetro: 0x10 => +/- 8g
  Wire.endTransmission();

  Wire.beginTransmission(MPU6050_address);
  Wire.write(0x1A);                       //Se abre configuración del filtro pasa bajo(LFP)
  Wire.write(0x03);                       //LFP: 0x03 => 42Hz(con retraso de 4.9ms)
  Wire.endTransmission();
}
```

Una vez configurado ya se pueden obtener las lecturas del sensor MPU-6050 leyendo los bytes indicados en la hoja de datos. [30]

```
//====== Subrutina de obtención de lecturas del MPU-6050======//
void MPU_6050() {
  Wire.beginTransmission(MPU6050_address);
  Wire.write(0x3B);                           //Se indica el byte de inicio de la lectura
  Wire.endTransmission();
  Wire.requestFrom(MPU6050_address, 14);      //Petición de 14 bytes desde el byte indicado
  while (Wire.available() < 14);              //Se espera hasta recibir los 14 bytes

  //Se guarda, en variables int de 16 bits, cada dos bytes leídos del sensor
```

```
  AccelX = Wire.read() << 8 | Wire.read();      //0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AccelY = Wire.read() << 8 | Wire.read();      //0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AccelZ = Wire.read() << 8 | Wire.read();      //0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Temperature = Wire.read() << 8 | Wire.read(); //0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyroX = Wire.read() << 8 | Wire.read();       //0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyroY = Wire.read() << 8 | Wire.read();       //0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyroZ = Wire.read() << 8 | Wire.read();       //0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)

  //Las variables anteriores se convierten a tipo float para facilitar el cálculo posterior
  ax = (float)AccelX;
  ay = (float)AccelY;
  az = (float)AccelZ;
  temperature = (float)Temperature;
  gx = (float)GyroX;
  gy = (float)GyroY;
  gz = (float)GyroZ;
}
```

Al hacer lecturas del sensor MPU, en estado de reposo, se observan unas desviaciones de las lecturas a las que cabía esperar. A esta desviación de las lecturas se le llama offset. Debido a este offset medimos la desviación de las lecturas y tomamos como referencia a cero esa desviación y no la referencia a cero que nos indica el sensor.

Se mide el offset del sensor antes de empezar a calcular la inclinación del sensor. Para calcular el offset se toman 3000 lecturas de cada uno de los seis grados de libertad y se hace una media aritmética de cada uno de estos. Pero antes de ello se desactiva el *watchdog* de software sino no da tiempo a tomar las 3000 lecturas. Cuantas más lecturas se tomen para el offset más preciso será este, por ello se toman tantas muestras como se pueden antes de que salte el *watchdog*.

```
//====== Subrutina de obtención del offset del MPU-6050======//
void init_MPU6050() {
  lcd.setCursor(0, 1);        //Se posiciona el cursor en vertical 1 para no borrar la IP
  lcd.print("...Offset...");
  ESP.wdtDisable();     //Se desactiva el watchdog de software

  for (count = 0; count < 3000 ; count ++) {
    MPU_6050();             //Salta a la subrutina llamada MPU_6050 (Obtención de lecturas)
    offset_pitch_gyro += gx;   //Suma en offset_pitch_gyro todos los valores de gx
    offset_roll_gyro += gy;
    offset_yaw_gyro += gz;
    offset_pitch_acce += ax;
    offset_roll_acce += ay;
    offset_yaw_acce += az;
    delayMicroseconds(1000);  //Espera 1 milisegundo a hacer las siguientes lecturas
  }

  offset_pitch_gyro = offset_pitch_gyro / 3000; //Media aritmética de todas las lecturas gx
  offset_roll_gyro = offset_roll_gyro / 3000;
  offset_yaw_gyro = offset_yaw_gyro / 3000;
  offset_pitch_acce = offset_pitch_acce / 3000;
  offset_roll_acce = offset_roll_acce / 3000;
  offset_yaw_acce = offset_yaw_acce / 3000;

  ESP.wdtEnable(0);         //Se vuelve a activar el watchdog de software
  lcd.setCursor(0, 1);
  lcd.print("Offset obtenido");
}
```

Se tiene que tener en cuenta que para el offset del acelerómetro en el grado az se eliminará la aceleración de la gravedad. El acelerómetro se ha configurado con una sensibilidad de ±8g (1g =

9,8m/s^2) y, según la hoja de datos del sensor MPU-6050, cada 4096 valores de lectura equivalen a una aceleración de 9,8m/s^2. Por lo que, cuando se haga la corrección del offset en el grado az, se sumarán 4096 para devolver a la medida la aceleración por gravedad.

Con el offset definido ya se puede calcular la velocidad angular y la inclinación del sensor. Se parte de un ángulo 0 antes del inicio del ascenso y se van sumando todas las variaciones de ángulo calculadas. En cada ciclo de ejecución del programa se obtiene la velocidad de rotación en ese ciclo y el tiempo que ha estado rotando a esa velocidad, se multiplican estos dos parámetros y el resultado es la variación de inclinación en ese tiempo. El giroscopio se ha configurado a una sensibilidad de ±500º/s por lo que, según la hoja de datos del sensor MPU-6050, cada 65,5 valores de lectura equivalen a una velocidad de giro de 1º/s.

$$\boldsymbol{Pitch_n = Pitch_{n-1} + \frac{g_y}{65,5} * dt} \quad \textbf{(Eq. 3.1)}$$

$$\boldsymbol{Roll_n = Roll_{n-1} + \frac{g_x}{65,5} * dt} \quad \textbf{(Eq. 3.2)}$$

Al usar el giroscopio como única herramienta para el cálculo de inclinación a medio y largo plazo aparece el error de deriva (drift). Este error deriva de la acumulación de los pequeños errores de cálculo de la variación de ángulo y al ruido del giroscopio.

Por otro lado, el cálculo de la inclinación a través del acelerómetro no presenta error de deriva a medio o largo plazo ya que el cálculo se hace de forma directa a través del ángulo que forma el sensor con la dirección de la gravedad. El problema del cálculo de la inclinación con el acelerómetro aparece a corto plazo, ya que es muy sensible al ruido y al movimiento del sensor por lo que la lectura no es fiable. Para el cálculo de la inclinación con el acelerómetro, sabiendo el valor de la gravedad y la medida de los tres grados del acelerómetro, con trigonometría podemos calcular el ángulo de inclinación:

$$\boldsymbol{Pitch = tan^{-1} * \left(\frac{a_y}{\sqrt[2]{a_x^2 + a_z^2}}\right)} \quad \textbf{(Eq. 3.3)}$$

$$\boldsymbol{Roll = tan^{-1} * \left(\frac{a_x}{\sqrt[2]{a_y^2 + a_z^2}}\right)} \quad \textbf{(Eq. 3.4)}$$

Entonces, para el cálculo de la inclinación, usaremos los dos sensores, el giroscopio a corto plazo y el acelerómetro a medio y largo plazo. Esto es conocido como filtro complementario, que se comporta como un filtro de paso alto para el cálculo del ángulo con el giroscopio y un filtro de paso bajo para el cálculo del ángulo con el acelerómetro. Para ello se multiplica el ángulo calculado con el giroscopio por un factor casi 1 y se suma con el ángulo calculado con el acelerómetro multiplicado por 1 menos el factor anterior:

$$\boldsymbol{Pitch = Pitch_{giro} * x + Pitch_{acce} * (1 - x)} \qquad \textbf{(Eq. 3.5)}$$

$$\boldsymbol{Roll = Roll_{giro} * x + Roll_{acce} * (1 - x)} \qquad \textbf{(Eq. 3.6)}$$

Siendo x <1

De forma empírica se han probado varios valores de x dando una mejor respuesta con x = 0,98.

```
//====== Subrutina de cálculo de ángulos MPU-6050======//
void MPU6050_GetData() {

  //Tiempo transcurrido desde la última lectura, en segundos
  execution_time = (micros() - loop_timer) / 1000000;  //  /1M => Conversión a segundos
  loop_timer = micros();

  MPU_6050();    //Salta a la subrutina llamada MPU_6050 (Obtención de lecturas)

  //Correción del offset
  ax -= offset_pitch_acce;   //Ajustamos la lectura con el offset
  ay -= offset_roll_acce;
  az -= offset_yaw_acce;
  az = az + 4096;            //La suma de 4096 corresponde a la aceleración de la gravedad,
                               borrada en el offset según la sensibilidad configurada
  gx -= offset_pitch_gyro;
  gy -= offset_roll_gyro;
  gz -= offset_yaw_gyro;

  //Cálculo de la velocidad de rotación
  w_pitch = gx / 65.5; // 65.5: si leo 65.5 en gx, significa que gira a 1°/s
  w_roll = gy / 65.5;
  w_yaw = gz / 65.5;

  //Calcular pitch y roll con el giroscopio
  angle_pitch += w_pitch * execution_time; //Velocidad de rotación * tiempo rotando
  angle_roll += w_roll * execution_time;

  //Calcular pitch y roll con el acelerómetro. 57,2958 = 180/pi (de radianes a grados)
  accel_ang_pitch = atan(ay / sqrt(pow(ax, 2) + pow(az, 2)))*57.2958;
  accel_ang_roll = atan(-ax / sqrt(pow(ay, 2) + pow(az, 2)))*57.2958;

  //Filtro complementario para corrección del drift
  angle_pitch = angle_pitch * 0.98 + accel_ang_pitch * 0.02;
  angle_roll = angle_roll * 0.98 + accel_ang_roll * 0.02;
}
```

Para el eje Yaw nos basta saber la velocidad de rotación del eje dada por el giroscopio como explica el siguiente apartado.

3.3.5 Configuración PID

Una vez ya se tiene el valor consigna de la inclinación (enviado desde el móvil) y el valor real de la inclinación (calculado del MPU-6050) ya se puede hacer el PID que da estabilidad al dron. Para la programación del PID se ha seguido la guía de Brett Beauregard publicada en su blog http://brettbeauregard.com/blog/ [31].

Una de las recomendaciones de Brett Beauregard es hacer el control PID de forma regular. El periodo en que se ejecuta el código varía de 5ms a 12ms, según si recibe una nueva petición URL o no. Para tener una ejecución regular de control PID se establece un periodo mínimo de 12 ms para la ejecución de la subrutina de cálculo de los PID.

Como se explica en el apartado 4.1.7 Configuración PWM las salidas se configuran a valores desde 1000 a 2000, por lo que quedan 1000 valores posibles de salida. Se limita a los PID a poder variar el valor de salida en un 80%, es decir, como mucho sumar o restar 400 a la salida. A su vez también se limita el termino I con un margen de actuación de los otros términos respecto al límite PID.

El dron puede tener dos modos de conducción o estabilización, modo acrobático o modo estable. La diferencia entre ambos son los diferentes parámetros que se toman para la obtención del error para el cálculo del PID. Para el modo acrobático basta saber las velocidades de rotación de los diferentes ejes y la consigna es velocidad angular. Para el modo estable se calcula el ángulo de cada eje y la consigna es posición angular. El método de cálculo del PID del modo estable necesita de más cálculos y de gasto de más recursos que el método de modo acrobático. En este proyecto se usa el método de modo estable con los ejes Pitch y Roll, y el método de modo acrobático para el eje Yaw. De esta forma se obtiene, de forma autónoma, una gran estabilidad en el aire y no se pierde estabilidad al usar el método de modo acrobático en el eje Yaw, y se reducen recursos al no usar el método de modo estable para este eje.

Los parámetros Kp, Ki y Kd se ajustan una vez el dron está listo para volar.

El código, adaptado a este proyecto, es el siguiente:

```
// Parámetros limite
int I_max = 300;  // Limitar parte integral (anti-wind-up)
int I_min = -300;
int PID_max = 400;  //Limitar salida del PID (anti-wind-up)
int PID_min = -400;


//====== Subrutina de control PID ======//
void PID_ang() {
  //Esperar a tiempo transcurrido desde la última llamada a subrutina > 12ms
  PIDTimer = (micros() - PIDTimer_last);
  if (PIDTimer < 12000){
    delayMicroseconds(12000 - PIDTimer);
  }
```

```
  PIDTimer_last = micros();

  //Obtención tiempo transcurrido desde el último control PID
  PIDTime = (micros() - PIDTime_last)/10000;  // Calculo en decenas de milisegundos
  PIDTime_last = micros();


  //Cálculo del error en cada eje de navegación
  pitch_angle_error = consignaPitch - angle_pitch;
  roll_angle_error = consignaRoll - angle_roll;
  yaw_w_error = consignaYaw - w_yaw;

  //Cálculo del termino I
  if (mappedThrottle > 1550) {    //No aplicar termino I hasta cierta consigna Throttle
    I_pitch_angle += (KiPitch * pitch_angle_error*PIDTime);
    I_pitch_angle = constrain(I_pitch_angle, I_min, I_max);     //Limitar valores termino I

    I_roll_angle += (KiRoll * roll_angle_error*PIDTime);
    I_roll_angle = constrain(I_roll_angle, I_min, I_max);

    I_yaw_w += (Ki_yaw_w * yaw_w_error*PIDTime);
    I_yaw_w = constrain(I_yaw_w, I_min, I_max);
  }
  else {
    I_pitch_angle = 0;
    I_roll_angle = 0;
    I_yaw_w = 0;
  }

  //Cálculo del termino D
  D_pitch_angle = KdPitch * (angle_pitch - prev_angle_pitch)/PIDTime;
  D_roll_angle = KdRoll * (angle_roll - prev_angle_roll)/PIDTime;
  D_yaw_w = Kd_yaw_w * (w_yaw - prev_w_yaw)/PIDTime;

  //Cálculo PID
  PID_pitch_ang = KpPitch * pitch_angle_error + I_pitch_angle - D_pitch_angle;
  PID_pitch_ang = constrain(PID_pitch_ang, PID_min, PID_max);  //Limitar valores salida PID

  PID_roll_ang = KpRoll * roll_angle_error + I_roll_angle - D_roll_angle;
  PID_roll_ang = constrain(PID_roll_ang, PID_min, PID_max);

  PID_yaw_w = Kp_yaw_w * yaw_w_error + I_yaw_w - D_yaw_w;
  PID_yaw_w = constrain(PID_yaw_w, PID_min, PID_max);

  //Variables para el siguiente ciclo
  prev_angle_pitch = angle_pitch;
  prev_angle_roll = angle_roll;
  prev_w_yaw = w_yaw;
}
```

3.3.6 Configuración PWM

La placa NodeMCU no dispone de salidas PWM por hardware por que se harán por software. Lo primero es configurar las salidas para indicar como debe ser la modulación de ancho de pulso e indicar que salidas de la placa se usarán como PWM. La frecuencia máxima teórica para el uso que se quiere dar es de 500Hz (periodo de 2ms), ya que la señal máxima del ESC de 2ms de ciclo de trabajo. Pero si se pone a esa frecuencia los ESCs no captan bien la señal PWM, por lo que se pone una frecuencia de 400Hz (periodo de 2,5ms).

Seguidamente se puede indicar el rango de valores que acepta la salida. Al tener un periodo configurado de 2,5ms se indica un rango de 2500 valores ya que así cada 1 valor de salida equivale a 1µs de ciclo de trabajo. Como valor mínimo los ESCs aceptan 1ms de ciclo de trabajo y como máximo

2ms, lo que equivale, con esta configuración, a valores de salida de la NodeMCU para los ESCs de 1000 a 2000.

Para que todos los motores se inicialicen a la vez, nada más empezar el programa, se inicializan los motores enviando señales PWM a 1ms de ciclo de trabajo.

```
//MOTORES
uint8_t M1 = D5;
uint8_t M2 = D6;
uint8_t M3 = D7;
uint8_t M4 = D8;

//====== Subrutina de inicialización PWM y motores ======//
void init_motors() {
  analogWriteRange(2500);      //Rango de 2500 valores
  analogWriteFreq(400);        //ciclos de 400Hz (periodo de 2,5ms)
  analogWrite(M1, 1000);
  analogWrite(M2, 1000);
  analogWrite(M3, 1000);
  analogWrite(M4, 1000);
}
```

Para modular la salida de cada motor se tiene en cuenta la consigna Throttle y los tres PIDs. La consigna que llega de la aplicación móvil es un valor numérico del 0 al 100, este hace referencia al porcentaje de potencia que se quiere en los motores. Antes de pasar esta consigna a los motores, es necesario normalizar esta consigna a los valores de salida para los ESCs. Para ello se utiliza la función map() de Arduino que es lo mismo que aplicar la ecuación de una recta que pasa por dos puntos:

$$y = \frac{y_2 - y_1}{x_2 - x_1} * (x - x_1) + y_1 \qquad \textbf{(Eq. 3.7)}$$

Una vez se ha normalizado el Throttle, por la posición de los motores, respecto al sensor MPU-6050 y el sentido de giro asignado (Figura 2.2), queda el código de modulación y de salida a los ESCs de la siguiente manera:

```
//====== Subrutina de modulación salida ======//
void Modulador () {

  mappedThrottle = map(consignaThrottle, 0, 100, 1000, 1800);
  // Limitar throttle a 1800 para dejar margen a los PID

  if (mappedThrottle <= 1300) {
    esc1 = 1000;
    esc2 = 1000;
    esc3 = 1000;
    esc4 = 1000;
  }

  // Si el throttle es mayor a 1300us, el control de estabilidad se activa.
  if (mappedThrottle > 1300) {
    esc1 = mappedThrottle + PID_pitch_ang - PID_roll_ang - PID_yaw_w; // Motor 1
    esc2 = mappedThrottle + PID_pitch_ang + PID_roll_ang + PID_yaw_w; // Motor 2
    esc3 = mappedThrottle - PID_pitch_ang + PID_roll_ang - PID_yaw_w; // Motor 3
    esc4 = mappedThrottle - PID_pitch_ang - PID_roll_ang + PID_yaw_w; // Motor 4
    if (esc1 < 1100) esc1 = 1100; //Se evita que alguno de los motores de detenga
    if (esc2 < 1100) esc2 = 1100;
    if (esc3 < 1100) esc3 = 1100;
```

```
    if (esc4 < 1100) esc4 = 1100;
    if (esc1 > 2000) esc1 = 2000; //Se evita mandar consignas mayores de 2000us
    if (esc2 > 2000) esc2 = 2000;
    if (esc3 > 2000) esc3 = 2000;
    if (esc4 > 2000) esc4 = 2000;
  }

  //Salidas PWM
  analogWrite(M1, esc1);
  analogWrite(M2, esc2);
  analogWrite(M3, esc3);
  analogWrite(M4, esc4);
}
```

La misma enumeración de los motores de la figura 2.2 se utiliza para las variables de las salidas de los ESCs. A todas las salidas se suma la consigna Throttle y se les suma o resta la salida PID en función de la posición del motor respecto a esos ejes.

Teniendo las lecturas reales de posición, las consignas de actuación, los PIDs y las salidas moduladas, ya se dispone del código mínimo para que el dron pueda volar con estabilidad a falta de predeterminar las constantes de los PIDs.

3.3.7 Lectura batería

Antes de determinar y testear las constantes de los PIDs se harán lecturas constantes del voltaje de la batería para no dañarla y que no disminuya demasiado su carga. El control del estado de la batería se hace leyendo, desde la NodeMCU, el voltaje de salida de la batería. Para esta lectura se necesita la entrada analógica de la placa. El voltaje máximo que admite la entrada analógica de la placa es de 3,3V, por lo que, como se observa en la Figura 4.1 del apartado Materiales y esquema de conexionado del hardware, para poder leer lecturas de hasta 12,6V que puede dar la batería, se configura un divisor resistivo de tensión para adaptar la señal.

Para el cálculo del divisor de tensión se usa la ecuación 3.8. En este caso V_{in} es la tensión máxima de la batería y V_{out} la tensión máxima de entrada de la entrada analógica. La relación de las resistencias tiene que ser $R_2 \leq R_1 \times 0{,}355$. Se elige $R_1 = 20k\Omega$ y $R_2 = 6{,}8k\Omega$.

$$v_{in} = \frac{R_1+R_2}{R_2} * v_{out} \rightarrow v_{in} = 3{,}94 * v_{out} \qquad \textbf{(Eq. 3.8)}$$

El factor de la relación de las resistencias, al no ser resistencias ideales, se calibra luego con la ayuda de un multímetro. La entrada analogía de la NodeMCU está conectada a un ADC de una resolución de 10 bits, por lo que los valores de entrada van desde 0 hasta 1023. Al leer la entrada valores de 0 a 1023 se escalan estos a valores deseados.

```
//====== Subrutina de cálculo voltaje batería======//
void batteryMonitoring() {
  batt_read = analogRead(A0); // Leer entrada analógica
  battery_voltage = 3.73 * (batt_read * 3.3 / 1023);
```

```
  // Si la tensión es inferior a 10.8V, se activa la señal de batería baja
  if (battery_voltage < 10.8) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("¡¡Batería baja!!");
    lcd.setCursor(0, 1);
    lcd.print("¡Apagar el dron!");

    BatTime = (millis() - BatTime_last);
    if (BatTime < 500)
      digitalWrite(LED_BUILTIN, LOW);
    else if (BatTime < 1000)
      digitalWrite(LED_BUILTIN, HIGH);
    else if (BatTime >= 1000)
      BatTime_last = millis();
  }
}
```

Se ha calibrado el valor de la relación de las resistencias del divisor con la ayuda de un multímetro. Si la placa detecta que la batería está por debajo de 10,8V aparecerá un aviso en la pantalla y parpadeará el led que tiene incorporado la placa.

3.3.8 Configuración GPS

Para la configuración del GPS se han usado dos librerías. La primera es *'TinyGPS.h'*, la cual, con los datos que recibe el GPS, proporciona las coordenadas del GPS, la fecha y la hora. La segunda es *'SoftwareSerial.h'*, esta librería permite hacer una comunicación en serie mediante software ya que los puertos hardware para esta comunicación no se recomienda usarlos debido a que el puerto USB también va conectado a estos puertos.

```
#include <SoftwareSerial.h>
#include <TinyGPS.h>

TinyGPS gps;   //Declaramos el objeto GPS
SoftwareSerial serialgps(0,16);//Declaramos el pin D3 como Rx y D0 como Tx

//====== Subrutina de iniciación puerto serie GPS ======//
void init_GPS() {
  serialgps.begin(9600);      //Iniciamos el puerto serie del GPS
}

//====== Subrutina de procesamiento y obtención de los datos GPS ======//
void GPS_GetData() {
  while(serialgps.available())
  {
    int c = serialgps.read();
    if(gps.encode(c))
    {
    gps.f_get_position(&latitude, &longitude); //Obtención de posición
    gps.crack_datetime(&year,&month,&day,&hour,&minute,&second,&hundredths);
    gps.stats(&chars, &sentences, &failed_checksum);
    }
  }
}
```

El guardado de coordenadas geográficas se hace cada vez que el dron recorre más de 2 metros. Para saber cuándo se han recorrido más de 2 metros se calcula la distancia entre la primera coordenada recibida y las siguientes entrantes, hasta que se calcula una distancia entre coordenadas de más de 2

metros. Cuando eso sucede se compara esta última coordenada con las siguientes, y así sucesivamente. Cada vez que se detecte un desplazamiento de más de 2 metros se guarda la nueva coordenada.

Para el cálculo de distancia más corta entre dos coordenadas geográficas se ha seguido el ejemplo dado por la web https://www.movable-type.co.uk/scripts/latlong.html [32], que trabaja con la fórmula de haversine.

$$\boldsymbol{a} = sin^2\left(\Delta\varphi/_2\right) + cos\varphi_1 * cos\varphi_2 * sin^2\left(\Delta\lambda/_2\right)$$

$$c = 2 * atan2\left(\sqrt{a}, \sqrt{1-a}\right) \qquad \textbf{(Eq. 3.9)}$$

$$d = R * c$$

Donde φ es la latitud, λ es la longitud y R es el radio de la Tierra (6.371.000m)

```
//====== Subrutina de cálculo de distancia entre dos coordenadas geográficas ======//
void Compare_GPSData() {

  //Se convierte la última coordenada geográfica recibida de angular a radial
  longitudeR = longitude*(PI/180);
  latitudeR = latitude*(PI/180);

  //Se convierte la última coordenada geográfica guardada de angular a radial
  longitudeLR = longitudeL*(PI/180);
  latitudeLR = latitudeL*(PI/180);

  //Se restan las coordenadas
  dlongitude = longitudeLR - longitudeR;
  dlatitude = latitudeLR - latitudeR;

  //Formula Haversine para distancia entre dos puntos. Distancia dada en metros
  a = (sq(sin(dlatitude/2))) + cos(latitudeR) * cos(latitudeLR) * (sq(sin(dlongitude/2)));
  c = 2 * atan2(sqrt(a), sqrt(1-a)) ;
  d = R * c;   //R es el radio de la tierra (6371000m)

  if (d>2){
    FS_SaveData();        //Se guarda la nueva coordenada
    longitudeL = longitude; //Se actualiza la última coordenada guardada
    latitudeL = latitude;
  }
}
```

3.3.9 Guardar datos en memoria flash de la placa

La placa NodeMCU tiene conectada una memoria Flash de 4Mb. Se aprovecha esta memoria para guardar datos capturados durante la ejecución del programa. Para guardar estos datos se usa SPIFFS (SPI Flash File System) que es un sistema de archivos para funcionar en la memoria flash conectada por SPI. La librería que se usa y facilita esta funcionalidad se llama *'FS.h'*.

```
#include <FS.h>

//====== Subrutina de iniciación de SPIFFS ======//
```

```
void init_FS() {
  //Crear un archivo nuevo, le damos nombre y escribir en él
  File f=SPIFFS.open(filename,"w");

  f.println("Latitud;Longitud");  //Escribe el nombre de las columnas
  f.close();    //Cerramos el archivo creado
}
```

Para guardar los datos primero se declara la librería. Seguidamente se crea el archivo donde se va a escribir. En el caso de este proyecto se quiere guardar el recorrido seguido por el dron por lo que, para mejorar la lectura de los datos, se guardará siguiendo el formato csv. La primera fila marcará el nombre de las columnas, el resto de filas son los datos guardados en cada columna. Cada registro se separará en columnas mediante el carácter ';'.

```
//====== Subrutina de guardar datos ======//
void FS_SaveData() {
  File f=SPIFFS.open(filename,"a"); //Abrimos el archivo para escribir al final de este

  f.print(latitude);    //Escribimos los datos deseados
  f.print(";");
  f.println(longitude);
  f.close();
}
```

Crear el archivo y escribir en él se inicia una vez ha empezado el vuelo. Cuando no se está volando hay la posibilidad de formatear los archivos creados, leer el directorio de archivos y abrir un archivo desde las peticiones URL recibidas, como se ha visto en el apartado recepción de parámetros enviados desde el móvil. Con el comando server.send(), que facilita la librería *'ESP8266WebServer'*, se "sube" al servidor lo que se encuentre dentro del paréntesis.

```
//====== Subrutina de envío del directorio existente ======//
void directoryFS()
{
  Dir dir = SPIFFS.openDir("");
  while (dir.next()) {
    directory = directory + "\n"+ dir.fileName();
  }
  server.send(200, "text/plain", directory); //Responde a la petición con el directorio
  lcd.setCursor(0, 1);
  lcd.print("Direct. enviado");
  directory = "";
}

//====== Subrutina de lectura del archivo pedido ======//
void readFile()
{
  String str = "";
  File f = SPIFFS.open(server.arg(0), "r");
  if (!f) {
    server.send(200, "Can't open SPIFFS file !\r\n");
  }
  else { //Se guarda en la variable str el contenido del archivo para enviarlo
    char buf[1024];
    int siz = f.size();
    while(siz > 0) {
      size_t len = std::min((int)(sizeof(buf) - 1), siz);
      f.read((uint8_t *)buf, len);
      buf[len] = 0;
      str += buf;
      siz -= sizeof(buf) - 1;
    }
```

```
    f.close();
    server.send(200, "text/plain", str); //Responde a la petición con str
    lcd.setCursor(0, 1);
    lcd.print("Archivo enviado");
  }
}

//====== Subrutina de formateo de SPIFFS ======//
void formatFS()
{
  lcd.setCursor(0, 1);
  lcd.print("Formateando...");
  SPIFFS.format();
  lcd.setCursor(0, 1);
  lcd.print("¡¡¡Formateado!!!");
  server.send(200, "text/plain", "SPIFFS formated");
}
```

3.3.10 Programa principal de Arduino

La rutina principal del programa, desde donde se llaman a todas las otras subrutinas, queda de la siguiente manera:

```
//Declaración de librerías a usar
.
.
.
//

//Declaración de variables a usar
.
.
.
//

void setup() {
  init_motors();
  init_GPS();
  Wire.begin();           //Inicializar la librería Wire
  Serial.begin(115200);
  conf_MPU();
  pinMode(LED_BUILTIN, OUTPUT);       //Se inicializa el led de la placa como salida
  digitalWrite(LED_BUILTIN, HIGH);           //Se apaga el led
  lcd.init();       //Activar la pantalla
  lcd.backlight();  //Activar la luz de fondo
  init_WifiConnection();
  init_MPU6050();
  lcd.setCursor(0, 1);
  lcd.print("Buscan satelites");
  while (latitude == 0.00000) {   //Se espera hasta recibir coordenadas reales al GPS
    GPS_GetData();
    Serial.print(".");
    delay(250);
  }
  //Se da nombre y formato al archivo de grabación de coordenadas
  filename = "/"+String(day)+"/"+String(month)+"/"+String(year)+"_"+
                              String(hour)+":"+String(minute)+":"+String(second)+".txt";
  SPIFFS.begin();       //Se Inicializa el sistema de archivos
  server.on("/dronePar", handleParemeters);//La subrutina a ejecutar al solicitar /dronePar
  server.on("/formatFS", formatFS);
  server.on("/directoryFS", directoryFS);
  server.on("/readFile", readFile);
  server.begin();                                 //Inicializa el servidor web
  digitalWrite(LED_BUILTIN, LOW);   //Indicador de que están hechas todas las calibraciones
  lcd.setCursor(0, 1);
  lcd.print("Throttle a 1");
  while (consignaThrottle != 1) {  //Se espera hasta recibir el valor 1 en Throttle
    server.handleClient();         //Se "escuchan" las peticiones URL entrantes
    consignaThrottle = argumentThrottle.toFloat();
```

```
  }
  digitalWrite(LED_BUILTIN, HIGH); //Se apaga el led conforme el Throttle ha llegado a 1
  init_FS();
}


void loop() {
  WifiConnection_GetData();
  MPU6050_GetData();
  PID_ang();
  Modulador();
  GPS_GetData();
  batteryMonitoring();
}
```

3.3.11 Iniciación de los ESCs

El fabricante recomienda calibrar los ESCs antes de usarlos de forma normal. Para ello se siguen las indicaciones del fabricante, que son las siguientes:

1) Conecte el ESC con el motor, el otro extremo conecte con el receptor.
2) Encienda el control remoto y presione el acelerador hasta el punto más alto
3) Enchufe el ESC en las baterías
4) Hay sonido de doble clic (el punto más alto de la confirmación del acelerador)
5) Después de obtener el sonido de clic, lleve el acelerador al punto más bajo
6) Confirme el sonido de tres clics que significa OK.

El código que se usa para la calibración de los ESCs es el siguiente:

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <Wire.h>

ESP8266WebServer server(80);

//Variables
const char* ssid = <Dirección Wifi>;
const char* password = <contraseña Wifi>;
String argumentThrottle;
float floatThrottle = 0;
int consignaThrottle = 0;

uint8_t M1 = D5;
uint8_t M2 = D6;
uint8_t M3 = D7;
uint8_t M4 = D8;


void handleParemeters()
{
  argumentThrottle = server.arg("Throttle");
  server.send(200,"text/plain", "");
}


//======// SETUP //======//
void setup() {
  analogWriteRange(2500);
  analogWriteFreq(400);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);
  Wire.begin();
```

```
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println ("Esperando a conectarse");  //Escribimos en el puerto serial
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  Serial.print("Dirección IP:  ");
  Serial.println(WiFi.localIP());
  server.on("/dronePar", handleParemeters);
  server.begin();
  digitalWrite(LED_BUILTIN, LOW);
  Serial.println("Poner Throttle a 1");
  while (floatThrottle != 1) {
    server.handleClient();
    floatThrottle = argumentThrottle.toFloat();
  }
  digitalWrite(LED_BUILTIN, HIGH);
  Serial.println("Antes de encender los motores poner el Throttle a 100");
}

//======// LOOP //======//
void loop() {
  server.handleClient();
  floatThrottle = argumentThrottle.toFloat();
  consignaThrottle = map(floatThrottle, 0, 100, 1000, 2000);
  analogWrite(M1, consignaThrottle);
  analogWrite(M2, consignaThrottle);
  analogWrite(M3, consignaThrottle);
  analogWrite(M4, consignaThrottle);
}
```

Todo lo que aparece en el código se ha visto en los apartados anteriores. En este caso no se necesita ni los PIDs, ni la IMU, ni el GPS, ni guardar datos en memoria, y tampoco se hace el control de batería. La placa NodeMCU se conecta al Wifi indicado, esta se establece como servidor, se espera a recibir consignas de la aplicación móvil. Según las consignas recibidas se modula la salida a los motores.

Para la calibración se desconecta el cable de alimentación de la placa con la PDB y se quitan las hélices, por seguridad. Se conecta el cable USB a la placa y se carga el código. Con el cable conectado se corre el código siguiendo los pasos para recibir las consignas desde el móvil. Se mandan consignas 100 a la placa y se cierra el interruptor para alimentar los motores. Si todo funciona se emite un doble pitido. Una vez se oye el pitido, se mandan a la placa consignas de 0. Si no se producen se emite un triple pitido confirmando que se han calibrado correctamente los ESCs.

3.3.12 Ajuste del sentido de giro de los motores

Por otro lado, para el correcto funcionamiento del dron es importante que los motores giren en el sentido deseado. Para conocer el sentido actual de giro se utiliza el código anterior, pero sin el USB conectado al ordenador y con el cable de alimentación de la placa conectado a la PDB.

Se cierra el interruptor para que empiece a ejecutar el programa. Después del punto del programa donde se baja a 1 el Throttle, desde la aplicación móvil incrementamos poco a poco la consigna Throttle

hasta que empiecen a moverse los motores. Los motores tienen que girar según el sentido que se indica en la figura 3.13.

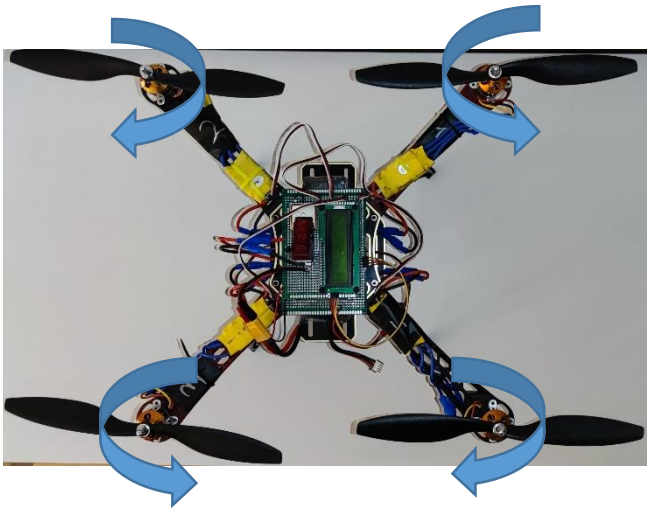

Figura 3.13. Fotografía del dron desde arriba con los sentidos deseados de giro

En caso de que alguno de los motores gire en el sentido contrario al deseado, se cambian dos de los tres conectores que conectan el motor con el ESC y el motor girará en el sentido contrario que anteriormente.

3.3.13 Ajuste parámetros Kp, Ki y Kd

Para el funcionamiento deseado del dron es importante ajustar bien el valor de las constantes Kp, Ki y Kd del PID. Si estas constantes no están bien ajustadas el dron no responderá a las consignas dadas desde el móvil y el gobierno del dron es imposible.

La obtención de estos tres parámetros a través de los métodos matemáticos más famosos supondría un proyecto en sí mismo. Al no ser un objetivo principal el cálculo exacto de estos parámetros se usa el método de ensayo y error, es decir, se modifican sucesivamente los parámetros de control hasta obtener el resultado deseado.

El método que se ha usado es el propuesto por la empresa Technische Beratung & Ingenieurbüro [33]. A diferencia de lo propuesto por la empresa, en este proyecto se usa una plataforma para la obtención, de forma aislada, de las constantes de uno de los PIDs. Se hace esta diferenciación debido a que la estructura utilizada en este proyecto no es igual en el eje Pitch que en el eje Roll.

Figura 3.14. Fotografía de la plataforma de pruebas para el control PID

Esta plataforma se usa para el establecer los mejores parámetros para el PID del eje Pitch y para el PID del eje Roll. Una vez establecidos los parámetros de los PIDs de esos dos ejes, se quita el dron de la plataforma y se ajusta el valor de los parámetros PID del eje Yaw en vuelo. Una vez finalizado el ajuste de los ejes se consigue un vuelo estable.

Figura 3.15. Fotografía de usuario volando el dron y ajustando el eje Yaw

4 Discusión

Para un buen funcionamiento de un vehículo no tripulado es muy importante la capacidad de corrección del error de la salida frente a la consigna. Para la elección de la comunicación entre el dron y el móvil se necesita una comunicación rápida, porque cuanto más rápida sea la ejecución del programa del dron más rápida será la capacidad de corrección del error y más estabilidad tendrá este en vuelo.

Para la comunicación Wifi, en este proyecto, se usa el protocolo de comunicación TCP/IP, que es el más comúnmente utilizado, y ofrece mayor seguridad en la transmisión de datos. Es posible que hubiera sido más adecuado usar el protocolo UDP/IP que se caracteriza por transmitir más rápido los paquetes de datos y podría haber mejorado la estabilidad del dron. Sin embargo, al ser la información enviada de muy poco tamaño los tiempos de comunicación con el protocolo usado se consideran suficientemente rápidos, por lo que probablemente no hubiera habido diferencias en cuanto a estabilidad. Una de las imitaciones de haber utilizado comunicación Wifi, es que es una línea de comunicación de corto alcance.

Por otro lado, en la elección de los módulos y componentes electrónicos se busca bajo coste, de facilidad adquisición y que contenga información para el desarrollo de estos. La búsqueda de componentes electrónicos se ha limitado a aquellos que son compatibles con IDE Arduino.

Por ello la placa de desarrollo que se ha escogido es la placa NodeMCU Lua Lolin V3. Esta placa de desarrollo tiene como microcontrolador el ESP8266, que es un chip integrado con conexión Wifi. El ESP8266 es compatible con Arduino y este entorno incluye librerías para comunicar con WiFi utilizando TCP/IP. Esta placa, frente a otras con wifi integrado, es de muy bajo coste.

A la hora de escoger las hélices se debe tener en cuenta su forma y tamaño, y estas están relacionadas con las características del motor elegido. Obviamente, cuanto mayor sea la hélice, más aire mueve y por tanto más fuerza (empuje) será capaz de generar el motor. Pero, si esta es demasiado grande, el motor probablemente no tenga suficiente potencia para hacer ascender el dron.

En caso de que en este proyecto no fuera necesaria una comunicación tan rápida, la mejor solución para guardar las coordenadas del recorrido del dron sería guardarlas directamente a una hoja de cálculos de Google Drive. El problema, como ya se ha mencionado, es que en este caso el tiempo de comunicación se alargaría demasiado y se perdería la estabilidad del dron.

Otra opción al guardado del recorrido hecho por el dron sería guardar los datos en una tarjeta micro SD acoplada en un módulo para Arduino. Con esta opción se aumentaría el tamaño de memoria, pero aumentaría el coste del proyecto y no quedan suficientes pines disponibles en la NodeMCU.

Al final se decide guardar los datos en la memoria flash ya que ya viene ensamblado y no añade coste al proyecto.

Buscando gran estabilidad y bajo coste se elige desarrollar un dron de ala rotatoria de 4 motores. Debido al corto alcance que ofrece la cobertura Wifi del móvil es más conveniente el uso de un dron capaz de mantener el vuelo sobre un punto, por eso el dron es de ala rotatoria. Los drones de menos de 4 motores tienen mucha menos estabilidad ya que solo tienen un eje de simetría frente los dos ejes de simetría perpendiculares que tiene el dron de 4 motores. Los drones de más de 4 motores son más estables y tienen más potencia, pero eso implica mayor coste y mayor gasto energético. El fabricante de la estructura obtenida para el montaje del dron aconseja unos motores, unas hélices y unos ESCs específicos para el uso de dicha estructura. Se han seguido estos consejos ya que el estudio de búsqueda de los componentes ideales se escapa del objetivo del proyecto.

Uno de los grandes problemas a la hora de desarrollar y montar un dron es la posterior estabilidad de este en el aire. El acelerómetro de la IMU es muy sensible a ruidos y la vibración de las hélices al girar es una gran fuente de ruido. El módulo GY-521 viene con un filtro pasa bajos para filtrar parte del ruido. Aparte de utilizar el filtro de la IMU también se ponen tornillos con gomas amortiguadoras entre la estructura, a la que van atornillados los motores, y la PCB a la que va sujetado la IMU, disminuyendo aún más el ruido.

La configuración de los PIDs es otro factor clave para la estabilidad aérea del dron. El método ideal sería obtener el modelo matemático del dron y con él calcular las constantes PID. Sin embargo, le nivel de complejidad es demasiado elevado como para poder hacerlo. Por otro lado, el método de Ziegler-Nichols es uno de los métodos de sintonización más ampliamente difundidos y utilizados. Este método sería muy laborioso ya que habría que montar un sistema por el cual poder hacer pruebas con el movimiento libre del dron sin que el dron o el usuario resulten dañados. Además, esto no aseguraría una correcta estabilidad del dron que podría conllevar igualmente tener que hacer un ajuste manual. Por lo que para la configuración de los PIDs y la obtención de constantes validas se ha seguido el método manual de ensayo y error, qué es un método adecuado y menos laborioso, pero menos preciso.

Conclusiones

La elección de Arduino como entorno de desarrollo ha sido acertada ya que su uso es muy intuitivo y al estar tan generalizado se puede encontrar información fácilmente en la comunidad.

Una de las principales dificultades en el desarrollo del dron es conseguir una buena estabilidad en el vuelo. Esta gran dificultad se debe tener en cuenta desde un inicio ya que condiciona la elección de componentes electrónicos y estructurales. Además, implica muchas horas de trabajo tanto en la fase de montaje y de desarrollo, como en las pruebas de vuelo para el ajuste de las constantes de los PIDs. Si no se planifica bien puede conllevar el fracaso del proyecto.

La grabación de las coordenadas geográficas del recorrido del dron puede parecer que no es de gran utilidad sin una vía de comunicación de más largo alcance, sin embargo, es de gran interés la incorporación de esta funcionalidad para poder conseguir un vuelo autónomo en un futuro.

La comunicación vía Wifi permite enviar varios parámetros a la vez, mientras que otros tipos de comunicación habitualmente solo permiten el envío de consignas. Gracias a esta particularidad el ajuste de las constantes de los PIDs es más sencillo. Aunque el vuelo del dron se ve limitado por el área de cobertura que proporciona el Wifi del móvil. Esto se podría solucionar cambiando la comunicación vía Wifi por un módulo con tarjeta SIM.

Finalmente, aunque el desarrollo de una aplicación móvil es complejo, a la larga es una ventaja controlar el dron desde el propio dispositivo móvil por la posibilidad de añadir nuevas funcionalidades, el uso extendido de dispositivos móviles, facilidad de instalación, capacidad de personalizar el diseño y un manejo comparable a los mandos de radiocontrol convencionales.

Futuras líneas de desarrollo

Las posibilidades que da un dron son muchísimas. Cada día se quiere implementar más la tecnología del dron para tareas profesionales. Por ello las posibles futuras líneas de desarrollo son muy numerosas.

Siguiendo el proyecto descrito faltaría la funcionalidad de dar al dron la capacidad de reproducir, de forma autónoma, un recorrido grabado con anterioridad.

Para desarrollar dicha funcionalidad haría falta, aparte del material actual, un magnetómetro y un barómetro de precisión. El magnetómetro sirve para saber en qué dirección está el polo magnético terrestre respecto a nuestro dron. El barómetro sirve para medir la presión atmosférica y con eso saber

la altitud a la que está el dron respecto el nivel del mar. El GPS también nos informa de la altitud del dron, pero con poca exactitud.

Se usaría el barómetro para guardar la altitud junto con las coordenadas geográficas para luego poder reproducir bien el recorrido hecho por el dron. También se usaría para que el dron mantuviera una altitud concreta de forma autónoma. Esto se lograría añadiendo un PID para la altitud, añadiendo un botón a la aplicación que indicara que se quiere mantener la altitud y cambiando las consignas de Throttle, de porcentaje de potencia a velocidad lineal de subida o descenso. Y el magnetómetro se usaría para apuntar con el dron hacía donde está la coordenada a la que se quiere ir.

Por otro lado, se necesitaría cambiar el protocolo de comunicación al protocolo UDP/IP, se necesitaría disminuir la capacidad del filtro pasa bajos del MPU-6050 y reducir todos los tiempos posibles ya que las entradas de las lecturas del magnetómetro y del barómetro harían aumentar considerablemente los tiempos de ejecución del programa.

Una de las mejoras que se podría hacer es poner sensores de proximidad de forma estratégica en el dron y programar un sistema anticolisión. Una forma sencilla y barata de hacerlo sería usando módulos HC-SR04, que son sensores de distancia que funcionan por ultrasonidos. Se harían lecturas constantes con estos módulos y al detectar algún objeto a cierta distancia se podría forzar una consigna para que el dron se moviera en dirección contraria a la de este objeto.

Otras funcionalidades podrían ser:

- Poner cámara al dron y enviar el video a tiempo real a la aplicación móvil.
- Poner cámara con visión térmica para detectar posibles incendios en bosques.
- Poner un módulo para tarjeta SIM y poder controlar el dron sin necesidad de estar cerca.
- Desde el móvil enviar las coordenadas al dron y hacer que este siga al móvil.

Análisis económico del coste de construcción

En este apartado se hace un cálculo del coste económico de la construcción del dron. Para este cálculo se tienen en cuenta los costes materiales, las horas de desarrollo, las horas de montaje y las horas de pruebas.

Coste de desarrollo de la aplicación móvil DroneControl

Desarrollo DroneControl	**Horas**	**Coste (€)/Hora**	**Coste(€)**
Creación de app y pantalla de inicio	20	15	300
Palancas de mandos	180	15	2.700
Parámetros PID	30	15	450
Peticiones para sistema de archivos	10	15	150
Total	240		3.600

Coste de desarrollo del programa, en Arduino, para el dron

Desarrollo programa para el dron	**Horas**	**Coste (€)/Hora**	**Coste(€)**
Programa principal y pantalla LCD	10	15	150
Wifi y configurador del servidor	30	15	450
MPU-6050	60	15	900
PIDs	20	15	300
Salida PWM	10	15	150
Tensión de la batería	15	15	225
GPS	10	15	150
Sistema de archivos NodeMCU	25	15	375
Iniciación ESCs	10	15	150
Total	190		2850

Coste de los materiales

Nombre del material	Coste del material (€)
NodeMCU Lua Lolin V3	7,79
Modulo GY-521	2,99
Modulo NEO-6M	4,99
4 x (Motor sin escobillas A2212 1000KV + ESC 30)	42,88
PDB-XT60	9,99
Pantalla LCD I2C 1602	3,69
Batería de LiPo, 3S (11,1V), 4500mAh, 45C	25,75
Cargador de balance para baterías de LiPo	21,99
Adaptador de potencia para cargador de balance	12,99
Interruptor conmutación de 12V y 20A con led	4,01
5 pares de hélices 1045	4,69
F450 Frame de 4 ejes	18,39
4 x Tornillo M3 8*8 con goma amortiguadora	7,99
100 x Conectores de pala macho - hembra	8,99
Juego de 180 tuercas M3 con separador hexagonal	7,65
5m de cables rojos y negros de 3mm de diámetro (16AWG)	7,29
Alambre de Soldadura sin Plomo de 0,8mm. 100g	9,99
Bridas de nylon	1
Varilla metálica de 6mm de diámetro	2,65
Cajita con 10 tornillos de 4,5mm de rosca y 40mm de longitud	1,85
Cajita con 15 tornillos de gancho cerrado	1,79
Cajita con 12 alcayatas roscadas	1,59
Pack: 600 resistencias, 26 piezas de 40 conectores macho - hembra para PCB, 15 PCBs, 10 pares de cables de 1,4mm con cabezal	19,99
Total:	**230,93**

Coste del montaje

Montaje del dron	**Horas**	**Coste (€)/Hora**	**Coste(€)**
Alimentación de componentes (PDB + interruptor + cableado)	7	15	105
Montaje Frame con motores y ESCs	4	15	60
PCB grande con los componentes	8	15	120
PCB mediana con los componentes	6	15	90
Ensamblaje de todos los elementos	10	15	150
Total	35		525

Coste de pruebas

Pruebas	**Horas**	**Coste (€)/Hora**	**Coste(€)**
Comunicación móvil - dron	7	15	105
Posición mediante datos MPU-6050	14	15	210
Sentido de lo motores	1	15	15
Ajustes de parámetros PIDs	35	15	525
Recepción coordenadas GPS	3	15	45
Total	60		900

Coste total

	Coste(€)
Desarrollo de la aplicación móvil DroneControl	3.600
Desarrollo del dron	2.850
Materiales	230,93
Montaje	525
Pruebas	900
Total:	8.105,93

Las horas totales dedicadas a la construcción del dron son 525h. Se ha puesto un precio hora de 15€ ya que, por las horas dedicadas, se deduce que se trata de un ingeniero junior. Económicamente este proyecto no es rentable, pero, al ser un proyecto de uso académico, no tratamos con rentabilidad económica ya que el coste se eleva desmesuradamente por el coste humano, coste invertido en el aprendizaje.

Bibliografía

[1] Real Academia Española. Diccionario de la lengua española [en línea]. 23ªedición Madrid: 2021. [Consulta: 20 de febrero del 2021]. Disponible en: https://dle.rae.es/

[2] Colaboradores de Wikipedia. Vehículo aéreo no tripulado [en línea]. Wikipedia, La enciclopedia libre, 2021 [Consulta: 20 de febrero del 2021]. Disponible en: https://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado

[3] Zima Robotics. Tipos de drones aéreos [en línea]. En: Powered by Zima Robotics. [Consulta: 20 de febrero del 2021]. Disponible en: https://dronespain.pro/tipos-de-drones-aereos/

[4] Colaboradores de Wikipedia. Ángulos de navegación [en línea]. Wikipedia, La enciclopedia libre, 2021 [Consulta: 20 de febrero del 2021]. Disponible en: https://es.wikipedia.org/wiki/%C3%81ngulos_de_navegaci%C3%B3n

[5] Colaboradores de Wikipedia. Unidad de medición inercial [en línea]. Wikipedia, La enciclopedia libre, 2021 [Consulta: 20 de febrero del 2021]. Disponible en: https://es.wikipedia.org/wiki/Unidad_de_medici%C3%B3n_inercial

[6] Villegas Cortez, J. Diseño y construcción de un dron para adquisición de datos del clima[en línea]. Celaya: 2016. ISSN 1405-1249 [Consulta: 20 de febrero del 2021]. Disponible en: https://www.researchgate.net/figure/Angulos-de-navegacion-para-los-4-motores-de-nuestro-dron-propuesto_fig5_309286252

[7] Macho, JC. LO QUE HAY QUE SABER PARA ELEGIR LOS MOTORES PARA UN CUADRACÓPTERO [en línea]. En: PROMETEC [en línea]. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.prometec.net/elegir-motores-cuadracoptero/

[8] Macho, JC. LO QUE HAY QUE SABER PARA ELEGIR LAS HÉLICES PARA UN CUADRACÓPTERO [en línea]. En: PROMETEC [en línea]. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.prometec.net/elegir-helices-dron/

[9] Macho, JC. LO QUE HAY QUE SABER PARA ELEGIR LOS ESCS PARA UN CUADRACÓPTERO [en línea]. En: PROMETEC [en línea]. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.prometec.net/esc-para-drones/

[10] Macho, JC. LO QUE HAY QUE SABER PARA ELEGIR UNA BATERÍA LIPO. GUÍA PARA PRINCIPIANTES [en línea]. En: PROMETEC [en línea]. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.prometec.net/elegir-bateria-lipo/

[11]Llamas, L. “NODEMCU, LA POPULAR PLACA DE DESARROLLO CON ESP8266” [blog].En: Luis Llamas [en línea]. 1 de junio del 2018. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.luisllamas.es/esp8266-nodemcu/

[12]Crespo, E. Aprendiendo Arduino [en línea]. España. [Consulta: 20 de febrero del 2021]. Disponible en: https://aprendiendoarduino.wordpress.com/2016/12/11/ide-arduino/

[13]Arduino. Arduino IDE 1.8.13 [Entorno de desarrollo integrado]. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.arduino.cc/en/software

[14]Santos, N. “ESP8266: Watchdog functions” [blog].En: techtutorialsx [en línea]. 21 de enero del 2017. [Consulta: 20 de febrero del 2021]. Disponible en: https://techtutorialsx.com/2017/01/21/esp8266-watchdog-functions/

[15]Llamas, L. “CÓMO USAR EL SPIFFS DEL ESP8266 CON EL ARDUINO IDE” [blog]. En: Luis Llamas [en línea]. 7 de agosto del 2019. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.luisllamas.es/como-usar-el-spiffs-del-esp8266-con-el-arduino-ide/

[16]Pardo, C. Controlador PID [en línea]. PICUINO. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.picuino.com/es/arduprog/control-pid.html

[17]Colaboradores de Wikipedia. Integral windup [en línea]. Wikipedia, La enciclopedia libre, 2021 [Consulta: 20 de febrero del 2021]. Disponible en https://en.wikipedia.org/wiki/Integral_windup

[18]Pardo, C. Método de Ziegler-Nichols [en línea]. PICUINO. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.picuino.com/es/arduprog/control-ziegler-nichols.html

[19]Colaboradores de Wikipedia. Controlador PID [en línea]. Wikipedia, La enciclopedia libre, 2021 [Consulta: 20 de febrero del 2021]. Disponible en https://es.wikipedia.org/wiki/Controlador_PID

[20]Colaboradores de Wikipedia. GPS [en línea]. Wikipedia, La enciclopedia libre, 2021 [Consulta: 20 de febrero del 2021]. Disponible en https://es.wikipedia.org/wiki/GPS

[21]Redacción. “Qué es el WiFi y cómo funciona para conectar todo a Internet” [en línea]. AdslZone. 1 de marzo del 2021. [Consulta: 20 de abril del 2021]. Disponible en: https://www.adslzone.net/reportajes/tecnologia/que-es-wifi-como-funciona/

[22]Castillo, JA. “Protocolo TCP/IP – Qué es y cómo funciona” [blog]. En: profesionalreview [en línea]. 21 de marzo del 2020. [Consulta: 20 de abril del 2021]. Disponible en: https://www.profesionalreview.com/2020/03/21/protocolo-tcp-ip/

[23] MDN contributors. "Generalidades del protocolo HTTP" [blog]. En: developer mozilla [en línea]. 1 de mayo del 2021 [Consulta: 1 de mayo del 2021]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/Overview

[24] Google developers. Introducción a Android Studio [en línea]. developers [Consulta: 20 de febrero del 2021]. Disponible en https://developer.android.com/studio/intro?hl=es-419

[25] Brun, D. virtual-joystick-android. [Software para Android Studio]. v1.10.1. [En línea] 28 de noviembre de 2018. [Consulta: 20 de abril del 2021]. Disponible en: https://github.com/controlwear/virtual-joystick-android

[26] Amazon, S.L. Ellenbogenorthese-LQ Drone F450 Drone con Marco de cámara 450 para RC MK MWC 4 Ejes RC Multicopter Quadcopter Heli Multi-Rotor con Land Gear Drone Shell [en línea]. Amazon. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.amazon.es/Ellenbogenorthese-LQ-c%C3%A1mara-Multicopter-Quadcopter-Multi Rotor/dp/B08ZNK3ZVQ/ref=sr_1_56?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=2NGZRFLGX2KXT&dchild=1&keywords=f450+frame&qid=1619911626&sprefix=f450+fr%2Caps%2C211&sr=8-56

[27] AZ-Delivery. NodeMCU Lua Lolin V3 Modul mit ESP8266 12F - Español. [en línea]. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.az-delivery.de/es/a/downloads/-/cf420676b0e0d6f3/ef7e659e54c3a81d

[28] Llamas, L. "CÓMO EMPLEAR EL ESP8266 O ESP32 COMO SERVIDOR HTTP" [blog].En: Luis Llamas [en línea]. 5 de mayo del 2019. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.luisllamas.es/como-emplear-el-esp8266-como-servidor/

[29] Naylamp Mechatronics SAC. "TUTORIAL MPU6050, ACELERÓMETRO Y GIROSCOPIO" [blog]. En: Naylamp Mechatronics SAC [en línea]. 21 de abril del 2018. [Consulta: 20 de febrero del 2021]. Disponible en: https://naylampmechatronics.com/blog/45_tutorial-mpu6050-acelerometro-y-giroscopio.html

[30] InvenSense Inc. MPU-6000 and MPU-6050 Product Specification. Revision 3.4. Sunnyvale: 2013. PS-MPU-6000A-00 [Consulta: 20 de abril del 2021]. Disponible en: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[31] Beauregard, B. "Improving the Beginner's PID" [blog]. En: brettbeauregard [en línea]. 15 de abril del 2011. [Consulta: 20 de febrero del 2021]. Disponible en: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

[32] Movale Type Ltd. Calculate distance, bearing and more between Latitude/Longitude points [en linea]. Cambridge (Reino Unido): Movable Type Ltd. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.movable-type.co.uk/scripts/latlong.html

[33] Brunner, D. Setting the PID controller of a drone properly [en linea].Technische Beratung & Ingenieurbüro. [Consulta: 20 de febrero del 2021]. Disponible en: https://www.technik-consulting.eu/en/optimizing/drone_PID-optimizing.html

[34] AZ_Delivery. NodeMCU Lua Lolin V3 Modul mit ESP8266 12E Datenblatt. [En línea]. Disponible en: https://cdn.shopify.com/s/files/1/1509/1638/files/NodeMCU_Lua_Lolin_V3_Modul_mit_ESP8266_12E_Datenblatt.pdf?342081239282763366

[35] u-blox. NEO-6 u-blox 6 GPS Modules. E. Thalwil: 2011. GPS.G6-HW-09005-E. [Consulta: 20 de abril del 2021]. Disponible en: https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf

A. Anexo - Código Android Studio

El código que se ha desarrollado, partiendo de la estructura básica de un nuevo proyecto Android Studio, es el siguiente:

B.1 Manifiesto Android

Código del archivo AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.dronecontrol">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/person_dron"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/person_dron"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        android:usesCleartextTraffic="true">
        <activity
            android:name=".FileSystem"
            android:windowSoftInputMode="adjustNothing"></activity>
        <activity
            android:name=".PantallaJoystick"
            android:screenOrientation="landscape" />
        <activity
            android:name=".MainActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

B.2 Pantalla inicio

La pantalla de inicio tiene dos archivos que la definen:

A.2.1 MainActivity.java

```
package com.example.dronecontrol;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;


public class MainActivity extends AppCompatActivity {

    private EditText ipxx;
    private EditText MaxThrottle;
    private EditText MaxHoldThrottle;
    private EditText MaxYaw;
    private EditText MaxPitch;
    private EditText MaxRoll;
    private static Button led;

    public static String urlip;
    public static String urlmaxpitch;
    public static String urlmaxthrottle;
    public static String urlmaxholdthrottle;
    public static String urlmaxroll;
    public static String urlmaxyaw;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        led=(Button) findViewById(R.id.Click);
        ipxx=(EditText) findViewById(R.id.droneIp);
        MaxThrottle=(EditText) findViewById(R.id.IntroThrottle);
        MaxHoldThrottle=(EditText)   findViewById(R.id.IntroHoldThrottle);
        MaxYaw=(EditText) findViewById(R.id.IntroYaw);
        MaxPitch=(EditText) findViewById(R.id.IntroPitch);
        MaxRoll=(EditText) findViewById(R.id.IntroRoll);

        led.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                urlip=ipxx.getText().toString();
                urlmaxthrottle=MaxThrottle.getText().toString();
                urlmaxyaw=MaxYaw.getText().toString();
                urlmaxpitch=MaxPitch.getText().toString();
                urlmaxroll=MaxRoll.getText().toString();
                urlmaxholdthrottle=MaxHoldThrottle.getText().toString();

                if(       MaxThrottle.getText().toString().matches("") ||
                          Integer.parseInt(MaxThrottle.getText().toString())< 70 ||
                          Integer.parseInt(MaxThrottle.getText().toString())> 100 ||
                          MaxYaw.getText().toString().matches("") ||
                          Integer.parseInt(MaxYaw.getText().toString())< 5 ||
                          Integer.parseInt(MaxYaw.getText().toString())> 20 ||
                          MaxPitch.getText().toString().matches("") ||
                          Integer.parseInt(MaxPitch.getText().toString())< 5 ||
                          Integer.parseInt(MaxPitch.getText().toString())> 30 ||
                          MaxRoll.getText().toString().matches("") ||
                          Integer.parseInt(MaxRoll.getText().toString())< 5 ||
                          Integer.parseInt(MaxRoll.getText().toString())> 30 ||
                          ipxx.getText().toString().matches("")){

                    Toast.makeText(MainActivity.this,
                    "Rellena todos los campos en los intervalos señalados",
                    Toast.LENGTH_LONG).show();

                }else {
                    Intent ht1 = new Intent(MainActivity.this,PantallaJoystick.class);
                    startActivity(ht1);
                }
            }
        });
    }
}
```

A.2.2 activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/droneIp"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_marginStart="10dp"
        android:layout_marginTop="40dp"
        android:ems="10"
        android:hint="Introducir el Ip del drone"
        android:inputType="textPersonName"
        android:text="192.168.43.31"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/TextMaxThrottle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="30dp"
        android:layout_marginLeft="30dp"
        android:layout_marginTop="180dp"
        android:text="Max pwr:"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/IntroYaw"
        android:layout_width="40dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="110dp"
        android:layout_marginLeft="110dp"
        android:layout_marginTop="280dp"
        android:ems="10"
        android:inputType="number"
        android:maxLength="2"
        android:text="10"
        android:textSize="14sp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/IntroRoll"
        android:layout_width="40dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="110dp"
        android:layout_marginLeft="110dp"
        android:layout_marginTop="420dp"
        android:ems="10"
        android:inputType="number"
        android:maxLength="2"
        android:text="5"
        android:textSize="14sp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/IntroPitch"
        android:layout_width="40dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="110dp"
```

```
        android:layout_marginLeft="110dp"
        android:layout_marginTop="350dp"
        android:ems="10"
        android:inputType="number"
        android:maxLength="2"
        android:text="5"
        android:textSize="14sp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/Click"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="10dp"
        android:text="Vuelo"
        android:visibility="visible"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/TextIp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_marginStart="10dp"
        android:layout_marginTop="5dp"
        android:text="Ip del drone:"
        android:textSize="18sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/TextCaracteristicas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_marginStart="10dp"
        android:layout_marginTop="120dp"
        android:text="Características de vuelo:"
        android:textSize="18sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/TextThrottle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="30dp"
        android:layout_marginLeft="30dp"
        android:layout_marginTop="150dp"
        android:text="Throttle:"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/TextYaw"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="30dp"
        android:layout_marginLeft="30dp"
        android:layout_marginTop="260dp"
        android:text="Yaw:"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
```

```
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/TextPitch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="30dp"
        android:layout_marginLeft="30dp"
        android:layout_marginTop="330dp"
        android:text="Pitch:"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/TextRoll"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="30dp"
        android:layout_marginLeft="30dp"
        android:layout_marginTop="400dp"
        android:text="Roll:"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/TextMaxYaw"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="30dp"
        android:layout_marginLeft="30dp"
        android:layout_marginTop="290dp"
        android:text="Max angle:"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/TextMaxRoll"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="30dp"
        android:layout_marginLeft="30dp"
        android:layout_marginTop="430dp"
        android:text="Max angle:"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/TextMaxPitch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="30dp"
        android:layout_marginLeft="30dp"
        android:layout_marginTop="360dp"
        android:text="Max angle:"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/IntroThrottle"
        android:layout_width="40dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="110dp"
        android:layout_marginLeft="110dp"
        android:layout_marginTop="170dp"
        android:ems="10"
        android:inputType="number"
        android:maxLength="3"
        android:text="100"
        android:textSize="14sp"
        app:layout_constraintStart_toStartOf="parent"
```

```
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/ThrottleInterval"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="160dp"
            android:layout_marginLeft="160dp"
            android:layout_marginTop="180dp"
            android:text="50% - 100%"
            android:textSize="10sp"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/YawInterval"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="160dp"
            android:layout_marginLeft="160dp"
            android:layout_marginTop="290dp"
            android:text="5°/s - 20°/s"
            android:textSize="10sp"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/throttleInterval2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="160dp"
            android:layout_marginLeft="160dp"
            android:layout_marginTop="360dp"
            android:text="5° - 30°"
            android:textSize="10sp"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/RollInterval"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="160dp"
            android:layout_marginLeft="160dp"
            android:layout_marginTop="430dp"
            android:text="5° - 30°"
            android:textSize="10sp"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

    </RelativeLayout>
```

B.3 Pantalla de palancas de mandos

La pantalla de palancas de mandos tiene tres archivos que la definen. El archivo .xml y dos clases java, la que define la funcionalidad de la pantalla y la otra se ejecuta mientras toquemos alguno de las palancas de mandos:

A.3.1 PantallaJoystick

```
package com.example.dronecontrol;

import androidx.appcompat.app.AppCompatActivity;
```

```
import android.annotation.SuppressLint;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.Switch;
import android.widget.TextView;

import io.github.controlwear.virtual.joystick.android.JoystickView;
import static com.example.dronecontrol.MainActivity.urlmaxpitch;
import static com.example.dronecontrol.MainActivity.urlmaxroll;
import static com.example.dronecontrol.MainActivity.urlmaxthrottle;
import static com.example.dronecontrol.MainActivity.urlmaxholdthrottle;
import static com.example.dronecontrol.MainActivity.urlmaxyaw;

public class PantallaJoystick extends AppCompatActivity {

    private TextView mTextViewAngleLeft;
    private TextView mTextViewStrengthLeft;
    private TextView mTextViewCoordinateLeft;
    private TextView mTextViewThrottle;
    private TextView mTextViewYaw;

    private TextView mTextViewAngleRight;
    private TextView mTextViewStrengthRight;
    private TextView mTextViewCoordinateRight;
    private TextView mTextViewPitch;
    private TextView mTextViewRoll;

    private EditText ActualKp;
    private EditText ActualKi;
    private EditText ActualKd;

    private static Button ButtonFS;
    private static Switch HoldSwitch;
    private static Button SumarKp;
    private static Button RestarKp;
    private static Button SumarKi;
    private static Button RestarKi;
    private static Button SumarKd;
    private static Button RestarKd;
    private static Button SwitchPID;

    public static double Kp;
    public static double Ki;
    public static double Kd;
    public static double Pitch;
    public static double Roll;
    public static double Throttle;
    public static double Yaw;
    public static int HoldA = 0;
    public static int PIDSelected = 0;

    public static String KpPitch = "4.0";
    public static String KiPitch = "0.2";
    public static String KdPitch = "75";
    public static String KpRoll = "4.0";
    public static String KiRoll = "0.2";
    public static String KdRoll = "75";
    public static String KpYaw = "2.6";
    public static String KiYaw = "0.05";
    public static String KdYaw = "0";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pantalla_joystick);

        ButtonFS=(Button) findViewById(R.id.buttonFS);
        HoldSwitch=(Switch) findViewById(R.id.altitudeSwitch);
```

```
        SumarKp=(Button) findViewById(R.id.sumador_Kp);
        RestarKp=(Button) findViewById(R.id.Restador_Kp);
        SumarKi=(Button) findViewById(R.id.sumador_Ki);
        RestarKi=(Button) findViewById(R.id.Restador_Ki);
        SumarKd=(Button) findViewById(R.id.Sumador_Kd);
        RestarKd=(Button) findViewById(R.id.Restador_Kd);
        SwitchPID=(Button) findViewById(R.id.SwitchParam);

        ActualKp=(EditText) findViewById(R.id.KpInput);
        ActualKi=(EditText) findViewById(R.id.KiInput);
        ActualKd=(EditText) findViewById(R.id.KdInput);

        mTextViewAngleLeft = (TextView) findViewById(R.id.textView_angle_left);
        mTextViewStrengthLeft = (TextView) findViewById(R.id.textView_strength_left);
        mTextViewCoordinateLeft = (TextView) findViewById(R.id.textView_coordinate_left);
        mTextViewThrottle = (TextView) findViewById(R.id.JThrottleOutput);
        mTextViewYaw = (TextView) findViewById(R.id.JYawOutput);

        final JoystickView joystickLeft =
                (JoystickView) findViewById(R.id.joystickView_left);
        joystickLeft.setAutoReCenterButton(false);

        HoldSwitch.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener()
{
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                if(isChecked){
                    joystickLeft.setAutoReCenterButton(true);
                    HoldA = 1;
                }
                else {
                    joystickLeft.setAutoReCenterButton(false);
                    HoldA = 0;
                }
            }
        });

        joystickLeft.setFixedCenter(true);
        joystickLeft.setOnMoveListener(new JoystickView.OnMoveListener() {
            @SuppressLint("DefaultLocale")
            @Override
            public void onMove(int angle, int strength) {
                mTextViewAngleLeft.setText(angle + "°");
                mTextViewStrengthLeft.setText(strength + "%");
                mTextViewCoordinateLeft.setText(
                        String.format("x%03d:y%03d",
                                (joystickLeft.getNormalizedX()-50),
                                (joystickLeft.getNormalizedY()-50)*-1)
                ); //urlmaxholdthrottle
                if (HoldA == 1) {
                    if (joystickLeft.getNormalizedY() > 40 &&
                                        joystickLeft.getNormalizedY() < 60)
                        Throttle = 0;
                    else {
                        if (joystickLeft.getNormalizedY() <= 40)
                            Throttle = Math.round((joystickLeft.getNormalizedY() - 40)
                              * -0.025 * Integer.parseInt(urlmaxholdthrottle));
                        else if (joystickLeft.getNormalizedY() >= 60)
                            Throttle = Math.round((joystickLeft.getNormalizedY() - 60)
                              * -0.025 * Integer.parseInt(urlmaxholdthrottle));
                    }
                    mTextViewThrottle.setText(Throttle + "cm/s");
                }
                else {
                    Throttle = Math.round((joystickLeft.getNormalizedY() - 100)
                                    * -0.01 * Integer.parseInt(urlmaxthrottle));
                    mTextViewThrottle.setText(Throttle + "%");
                }
                if (joystickLeft.getNormalizedX() > 35 &&
                                    joystickLeft.getNormalizedX() < 65) Yaw = 0;
                else {
                    if (joystickLeft.getNormalizedX() <= 35)
                        Yaw = Math.round((joystickLeft.getNormalizedX()-35)*(0.02857)
```

```
                                        *Integer.parseInt(urlmaxyaw));
                else if (joystickLeft.getNormalizedX() >= 65)
                    Yaw = Math.round((joystickLeft.getNormalizedX()-65)
                                        *(0.02857)*Integer.parseInt(urlmaxyaw));
            }
            mTextViewYaw.setText(Yaw + "°/s");
            SendParameters process1 = new SendParameters();
            process1.execute();
        }
    },20); //lee cada X (20) milisegundos los valores

    mTextViewAngleRight = (TextView) findViewById(R.id.textView_angle_right);
    mTextViewStrengthRight = (TextView) findViewById(R.id.textView_strength_right);
    mTextViewCoordinateRight = (TextView) findViewById(R.id.textView_coordinate_right);
    mTextViewPitch = (TextView) findViewById(R.id.JPitchOutput);
    mTextViewRoll = (TextView) findViewById(R.id.JRollOutput);

    Kp = Math.round(Double.valueOf(ActualKp.getText().toString())*100d)/100d;
    Ki = Math.round(Double.valueOf(ActualKi.getText().toString())*1000d)/1000d;
    Kd = Math.round(Double.valueOf(ActualKd.getText().toString())*100d)/100d;

    final JoystickView joystickRight =
                    (JoystickView) findViewById(R.id.joystickView_right);
    joystickRight.setFixedCenter(true);
    joystickRight.setOnMoveListener(new JoystickView.OnMoveListener() {
        @SuppressLint("DefaultLocale")
        @Override

        public void onMove(int angle, int strength) {
            mTextViewAngleRight.setText(angle + "°");
            mTextViewStrengthRight.setText(strength + "%");
            mTextViewCoordinateRight.setText(
                    String.format("x%03d:y%03d",
                            (joystickRight.getNormalizedX()-50),
                            (joystickRight.getNormalizedY()-50)*-1)
            );
            if (joystickRight.getNormalizedY() > 40 &&
                    joystickRight.getNormalizedY() < 60) Pitch = 0;
            else {
                if (joystickRight.getNormalizedY() <= 40)
                    Pitch = Math.round((joystickRight.getNormalizedY()-40)*-0.025*
                            Integer.parseInt(urlmaxpitch));
                else if (joystickRight.getNormalizedY() >= 60)
                    Pitch = Math.round((joystickRight.getNormalizedY()-60)*-0.025*
                            Integer.parseInt(urlmaxpitch));
            }
            mTextViewPitch.setText(Pitch + "°");
            if (joystickRight.getNormalizedX() > 40 &&
                    joystickRight.getNormalizedX() < 60) Roll = 0;
            else {
                if (joystickRight.getNormalizedX() <= 40)
                    Roll = Math.round((joystickRight.getNormalizedX()-40)*0.025*
                            Integer.parseInt(urlmaxroll));
                else if (joystickRight.getNormalizedX() >= 60)
                    Roll = Math.round((joystickRight.getNormalizedX()-60)*0.025*
                            Integer.parseInt(urlmaxroll));
            }
            mTextViewRoll.setText(Roll + "°");
            SendParameters process1 = new SendParameters();
            process1.execute();
        }
    },20); //lee cada X (20) milisegundos los valores

    ButtonFS.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent ht1 = new Intent(PantallaJoystick.this,FileSystem.class);
            startActivity(ht1);
        }
    });

    SumarKp.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
    public void onClick(View v) {
        Kp = Math.round((Double.valueOf(ActualKp.getText().toString()) +
              0.2)*100d)/100d;
        ActualKp.setText(""+Kp);
    }
});

RestarKp.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Kp = Double.valueOf(ActualKp.getText().toString());
        if (Kp >0) {
            if (Kp == 0.2) {
                Kp = 0;
                ActualKp.setText("0.0");
            }
            else {
                Kp = Math.round((Kp - 0.2)*100d)/100d;
                ActualKp.setText(""+Kp);
            }
        }
    }
});

SumarKi.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Ki = Math.round((Double.valueOf(ActualKi.getText().toString()) +
                                0.05)*1000d)/1000d;
        ActualKi.setText(""+Ki);
    }
});

RestarKi.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Ki = Double.valueOf(ActualKi.getText().toString());
        if (Ki >0) {
            if (Ki == 0.05) {
                Ki = 0;
                ActualKi.setText("0.00");
            }
            else {
                Ki = Math.round((Ki - 0.05)*1000d)/1000d;
                ActualKi.setText(""+Ki);
            }
        }
    }
});

SumarKd.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Kd = Math.round(Double.valueOf(ActualKd.getText().toString()) + 25);
        ActualKd.setText(""+Kd);
    }
});

RestarKd.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Kd = Double.valueOf(ActualKd.getText().toString());
        if (Kd >0) {
            if (Kd == 25) {
                Kd = 0;
                ActualKd.setText("0");
            }
            else {
                Kd = Math.round(Kd - 25);
                ActualKd.setText(""+Kd);
            }
```

```
                }
            }
        });

        SwitchPID.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (PIDSelected == 0) {     //Pitch = 0, Roll = 1, Yaw = 2
                    PIDSelected = 1;
                    SwitchPID.setText("Roll PID:");

                    KpPitch = ActualKp.getText().toString();
                    KiPitch = ActualKi.getText().toString();
                    KdPitch = ActualKd.getText().toString();

                    Kp = Double.valueOf(KpRoll);
                    Ki = Double.valueOf(KiRoll);
                    Kd = Double.valueOf(KdRoll);

                    ActualKp.setText(KpRoll);
                    ActualKi.setText(KiRoll);
                    ActualKd.setText(KdRoll);
                }
                else if (PIDSelected == 1) {
                    PIDSelected = 2;
                    SwitchPID.setText("Yaw PID:");
                    KpRoll = ActualKp.getText().toString();
                    KiRoll = ActualKi.getText().toString();
                    KdRoll = ActualKd.getText().toString();

                    Kp = Double.valueOf(KpYaw);
                    Ki = Double.valueOf(KiYaw);
                    Kd = Double.valueOf(KdYaw);

                    ActualKp.setText(KpYaw);
                    ActualKi.setText(KiYaw);
                    ActualKd.setText(KdYaw);
                }
                else if (PIDSelected == 2) {
                    PIDSelected = 0;
                    SwitchPID.setText("Pitch PID:");
                    KpYaw = ActualKp.getText().toString();
                    KiYaw = ActualKi.getText().toString();
                    KdYaw = ActualKd.getText().toString();

                    Kp = Double.valueOf(KpPitch);
                    Ki = Double.valueOf(KiPitch);
                    Kd = Double.valueOf(KdPitch);

                    ActualKp.setText(KpPitch);
                    ActualKi.setText(KiPitch);
                    ActualKd.setText(KdPitch);
                }
            }
        });
    }
}
```

A.3.2 SendParameters.java

```
package com.example.dronecontrol;

import android.os.AsyncTask;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
```

```
import java.net.URL;

import static com.example.dronecontrol.MainActivity.urlip;
import static com.example.dronecontrol.PantallaJoystick.Pitch;
import static com.example.dronecontrol.PantallaJoystick.Roll;
import static com.example.dronecontrol.PantallaJoystick.Throttle;
import static com.example.dronecontrol.PantallaJoystick.Yaw;
import static com.example.dronecontrol.PantallaJoystick.HoldA;
import static com.example.dronecontrol.PantallaJoystick.Kp;
import static com.example.dronecontrol.PantallaJoystick.Ki;
import static com.example.dronecontrol.PantallaJoystick.Kd;
import static com.example.dronecontrol.PantallaJoystick.PIDSelected;

public class SendParameters extends AsyncTask<Void,Void,String> {
    @Override
    protected String doInBackground(Void... voids) {
        try {
            URL url = new
URL("http://"+urlip+"/dronePar?Pitch="+Pitch+"&Roll="+Roll+"&Throttle="+Throttle+"&Yaw="+Ya
w+"&Hold="+HoldA+"&Kp="+Kp+"&Ki="+Ki+"&Kd="+Kd+"&PIDSelected="+PIDSelected);
            HttpURLConnection httpURLConnection = (HttpURLConnection) url.openConnection();
            InputStream inputStream = httpURLConnection.getInputStream();
            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(inputStream));
        }
        catch (MalformedURLException e)
        {

        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        return null;
    }
    @Override
    protected void onPostExecute(String aVoid) {
        super.onPostExecute(aVoid);
    }
}
```

A.3.3 activity_pantalla_joystick.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".PantallaJoystick">

    <EditText
        android:id="@+id/KiInput"
        android:layout_width="34dp"
        android:layout_height="32dp"
        android:layout_marginBottom="30dp"
        android:ems="10"
        android:inputType="numberDecimal"
        android:text="0.2"
        android:textAlignment="center"
        android:textSize="10sp"
        app:layout_constraintBottom_toBottomOf="@+id/textViewKd"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <Button
        android:id="@+id/sumador_Ki"
        android:layout_width="32dp"
        android:layout_height="32dp"
```

```
        android:layout_marginStart="10dp"
        android:layout_marginBottom="30dp"
        android:text="+"
        android:textAlignment="center"
        android:textSize="10sp"
        app:layout_constraintBottom_toTopOf="@+id/Sumador_Kd"
        app:layout_constraintStart_toEndOf="@+id/KiInput" />

    <Button
        android:id="@+id/Restador_Ki"
        android:layout_width="32dp"
        android:layout_height="32dp"
        android:layout_marginEnd="10dp"
        android:layout_marginBottom="30dp"
        android:text="-"
        android:textSize="10sp"
        app:layout_constraintBottom_toTopOf="@+id/Restador_Kd"
        app:layout_constraintEnd_toStartOf="@+id/KiInput" />

    <TextView
        android:id="@+id/textViewKi"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Ki:"
        android:textSize="10sp"
        app:layout_constraintBottom_toTopOf="@+id/KiInput"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <EditText
        android:id="@+id/KpInput"
        android:layout_width="32dp"
        android:layout_height="32dp"
        android:layout_marginBottom="30dp"
        android:ems="10"
        android:inputType="numberDecimal"
        android:text="4.0"
        android:textAlignment="center"
        android:textSize="10sp"
        app:layout_constraintBottom_toTopOf="@+id/KiInput"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/textViewKp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Kp:"
        android:textSize="10sp"
        app:layout_constraintBottom_toTopOf="@+id/KpInput"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <Button
        android:id="@+id/Restador_Kp"
        android:layout_width="32dp"
        android:layout_height="32dp"
        android:layout_marginEnd="10dp"
        android:layout_marginBottom="30dp"
        android:text="-"
        android:textSize="10sp"
        app:layout_constraintBottom_toTopOf="@+id/Restador_Ki"
        app:layout_constraintEnd_toStartOf="@+id/KpInput" />

    <Button
        android:id="@+id/sumador_Kp"
        android:layout_width="32dp"
        android:layout_height="32dp"
        android:layout_marginStart="10dp"
        android:layout_marginBottom="30dp"
        android:text="+"
        android:textAlignment="center"
```

```
        android:textSize="10sp"
        app:layout_constraintBottom_toTopOf="@+id/sumador_Ki"
        app:layout_constraintStart_toEndOf="@+id/KpInput" />

    <TextView
        android:id="@+id/textViewKd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Kd:"
        android:textSize="10sp"
        app:layout_constraintBottom_toTopOf="@+id/KdInput"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/textView_angle_left"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="5dp"
        android:text="0°"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView_strength_left"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView_angle_left"
        android:layout_marginStart="16dp"
        android:layout_marginTop="0dp"
        android:text="0%"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView_angle_left" />

    <io.github.controlwear.virtual.joystick.android.JoystickView
        android:id="@+id/joystickView_left"
        android:layout_width="230dp"
        android:layout_height="230dp"
        android:layout_alignParentBottom="true"
        android:layout_marginStart="25dp"
        android:layout_marginBottom="16dp"
        android:background="@drawable/globe_icon"
        app:JV_buttonImage="@drawable/dron_cercle"
        app:JV_fixedCenter="false"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent" />


    <TextView
        android:id="@+id/textView_angle_right"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:layout_marginTop="5dp"
        android:layout_marginEnd="16dp"
        android:text="0°"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />


    <TextView
        android:id="@+id/textView_strength_right"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView_angle_right"
        android:layout_alignParentRight="true"
        android:layout_marginTop="0dp"
        android:layout_marginEnd="16dp"
        android:text="0%"
        app:layout_constraintEnd_toEndOf="parent"
```

```
        app:layout_constraintTop_toBottomOf="@+id/textView_angle_right" />

    <TextView
        android:id="@+id/textView_coordinate_right"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView_strength_right"
        android:layout_alignParentRight="true"
        android:layout_marginTop="0dp"
        android:layout_marginEnd="16dp"
        android:text="x050:x050"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView_strength_right" />

    <TextView
        android:id="@+id/textView_coordinate_left"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView_strength_left"
        android:layout_alignParentRight="true"
        android:layout_marginStart="16dp"
        android:text="x050:x050"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView_strength_left" />

    <io.github.controlwear.virtual.joystick.android.JoystickView
        android:id="@+id/joystickView_right"
        android:layout_width="230dp"
        android:layout_height="230dp"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true"
        android:layout_marginEnd="25dp"
        android:layout_marginBottom="16dp"
        android:background="@drawable/globe_icon"
        app:JV_buttonImage="@drawable/dron_cercle"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <TextView
        android:id="@+id/JThrottleText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="130dp"
        android:layout_marginTop="10dp"
        android:text="Throttle:"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/JYawText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="130dp"
        android:layout_marginTop="35dp"
        android:text="Yaw:"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/JPitchText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:layout_marginEnd="200dp"
        android:text="Pitch:"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/JRollText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_marginTop="35dp"
        android:layout_marginEnd="200dp"
        android:text="Roll:"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/JThrottleOutput"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="200dp"
        android:layout_marginTop="10dp"
        android:text="0%"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/JYawOutput"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="200dp"
        android:layout_marginTop="35dp"
        android:text="0°"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/JPitchOutput"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:layout_marginTop="10dp"
        android:layout_marginEnd="155dp"
        android:text="0°"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/JRollOutput"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:layout_marginTop="35dp"
        android:layout_marginEnd="155dp"
        android:text="0°"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Switch
        android:id="@+id/altitudeSwitch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:visibility="invisible"/>

    <TextView
        android:id="@+id/switchtext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hold altitude"
        app:layout_constraintBottom_toTopOf="@+id/altitudeSwitch"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.499"
        app:layout_constraintStart_toStartOf="parent"
        android:visibility="invisible"/>

    <EditText
```

```
        android:id="@+id/KdInput"
        android:layout_width="32dp"
        android:layout_height="32dp"
        android:layout_marginBottom="30dp"
        android:ems="10"
        android:inputType="numberDecimal"
        android:text="75"
        android:textAlignment="center"
        android:textSize="10sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <Button
        android:id="@+id/Sumador_Kd"
        android:layout_width="32dp"
        android:layout_height="32dp"
        android:layout_marginStart="10dp"
        android:layout_marginBottom="30dp"
        android:text="+"
        android:textAlignment="center"
        android:textSize="10sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toEndOf="@+id/KdInput" />

    <Button
        android:id="@+id/Restador_Kd"
        android:layout_width="32dp"
        android:layout_height="32dp"
        android:layout_marginEnd="10dp"
        android:layout_marginBottom="30dp"
        android:text="-"
        android:textSize="10sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/KdInput" />

    <Button
        android:id="@+id/SwitchParam"
        android:layout_width="110dp"
        android:layout_height="35dp"
        android:layout_marginBottom="30dp"
        android:backgroundTint="#FFFFFF"
        android:text="Pitch PID:"
        android:textSize="10sp"
        app:layout_constraintBottom_toTopOf="@+id/KpInput"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <Button
        android:id="@+id/buttonFS"
        android:layout_width="90dp"
        android:layout_height="50dp"
        android:layout_marginTop="20dp"
        android:backgroundTint="#FFFFFF"
        android:text="Go to
        File system"
        android:textSize="10sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

B.4 Pantalla de peticiones para el sistema de archivos NodeMCU

La pantalla de peticiones para lectura del sistema de archivos de la NodeMCU tiene dos archivos que la definen:

A.4.1 FileSystem.java

```
package com.example.dronecontrol;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.webkit.WebResourceRequest;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.EditText;

import static com.example.dronecontrol.MainActivity.urlip;

public class FileSystem extends AppCompatActivity {

    String url = "";
    private EditText archivoFs;
    WebView web;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_file_system);

        Button listarArchivos =(Button) findViewById(R.id.archivosFs);
        Button formatAllFS =(Button) findViewById(R.id.formatFS);
        Button verArchivoFs =(Button) findViewById(R.id.buttonLeerFS);
        Button botonTestFS =(Button) findViewById(R.id.testbuttonFS);
        archivoFs=(EditText) findViewById(R.id.leerFs);

        web = (WebView) findViewById(R.id.miVisor);
        web.setWebViewClient(new MyWebViewClient());
        WebSettings settings = web.getSettings();
        settings.setJavaScriptEnabled(true);
        web.loadUrl(url);

        listarArchivos.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                url = "http://"+urlip+"/directoryFS";
                web.loadUrl(url);
            }
        });

        formatAllFS.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                url = "http://"+urlip+"/formatFS";
                web.loadUrl(url);
            }
        });

        verArchivoFs.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String urlArchivo=archivoFs.getText().toString();
                url = "http://"+urlip+"/readFile?file="+urlArchivo;
                web.loadUrl(url);
            }
        });

        botonTestFS.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                url = "https://www.google.es";
                web.loadUrl(url);
            }
```

```
        });
    }

    private class MyWebViewClient extends WebViewClient{
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String request) {
                view.loadUrl(url);
                return true;
        }
    }
}
```

A.4.2 activity_file_system.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FileSystem">

    <Button
        android:id="@+id/formatFS"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:layout_marginEnd="40dp"
        android:backgroundTint="#FFFFFF"
        android:text="Formatear FS"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/testbuttonFS"
        android:layout_width="30dp"
        android:layout_height="28dp"
        android:backgroundTint="#FFFFFF"
        android:text="test"
        android:textSize="8sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/leerFs"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="40dp"
        android:layout_marginTop="80dp"
        android:ems="10"
        android:text="/"
        android:inputType="textPersonName"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/archivosFs"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="40dp"
        android:layout_marginTop="10dp"
        android:backgroundTint="#FFFFFF"
        android:text="Act. Archivos FS"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <WebView
        android:id="@+id/miVisor"
        android:layout_width="match_parent"
```

```
        android:layout_height="450dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent" />

    <Button
        android:id="@+id/buttonLeerFS"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="80dp"
        android:layout_marginEnd="40dp"
        android:backgroundTint="#FFFFFF"
        android:text="Leer archivo"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

B. Anexo - Código Arduino

B.1 Codigo principal

```
//Declaración de librerías a usar
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <Wire.h>
#include <FS.h>
#include <SoftwareSerial.h>
#include <TinyGPS.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2); //Se instancia un objeto de la clase LiquidCrystal_I2C
(lcd)


//GPS
TinyGPS gps;  //Declaramos el objeto GPS
SoftwareSerial serialgps(0,16);//Declaramos el pin D3 conectado a Tx y D0 conectado Rx
int year;
byte month, day, hour, minute, second, hundredths;
unsigned long chars;
unsigned short sentences, failed_checksum;
float latitude = 0.00000, longitude = 0.00000, latitudeL = 0.00000, longitudeL = 0.00000, R
= 6371000;
float longitudeR, latitudeR, longitudeLR, latitudeLR, dlongitude, dlatitude;
float a, c, d;


//File system
//const char* filename;
String filename;
String directory;


//MobileSetPoint_Variables
/*
Se instancia un objeto de la clase ESP8266WebServer (server) y se le da valor al puerto(80)
El servidor responde a los clientes en el puerto 80, que es un puerto estándar para que los
navegadores web hablen con los servidores web.
*/
ESP8266WebServer server(80); // 80 is the port number
const char* ssid = "JAP Mi 10T Pro";
const char* password = "caracarton";
String argumentPitch, argumentRoll, argumentThrottle, argumentYaw;
String KpString, KiString, KdString, PIDSelectedString;
float consignaPitch = 0, consignaRoll = 0, consignaThrottle = 0, consignaYaw = 0;
float KpPitch = 4.0, KiPitch = 0.2, KdPitch = 75;
float Kp_yaw_w = 2.6, Ki_yaw_w = 0.05, Kd_yaw_w = 0;
float KpRoll = 4.0, KiRoll = 0.2, KdRoll = 75;;
int PIDSelected = 0;


// MPU6050 Variables
#define MPU6050 address 0x68
float angle_pitch, angle_roll, w_yaw, w_pitch, w_roll;
float accel_ang_pitch, accel_ang_roll;
float offset_pitch_gyro, offset_roll_gyro, offset_yaw_gyro;
int count;
float ax, ay, az, temperature, gx, gy, gz;
bool set_gyro_angles;
float offset_yaw_acce, offset_roll_acce, offset_pitch_acce;
int16_t AccelX, AccelY, AccelZ, Temperature, GyroX, GyroY, GyroZ;
float execution_time, loop_timer;


// PID
```

```
float PIDwInYaw, yaw_w_giroscopio, yaw_w_error, I_yaw_w, prev_yaw_w_error, PID_yaw_w,
PID_yaw_w_ant, D_yaw_w;
float pitch_ang_giroscopio, pitch_angle_error, I_pitch_angle, prev_pitch_angle_error,
PID_pitch_ang, D_pitch_angle;
float roll_ang_giroscopio, roll_angle_error, I_roll_angle, prev_roll_angle_error,
PID_roll_ang, D_roll_angle;
float prev_angle_pitch, prev_angle_roll, prev_w_yaw;
float Altitud_error, I_Altitud, D_Altitud, PID_Altitud, prev_Speed_Altitud;
float PIDTime = 0, PIDTime_last = 0, PIDTimer = 0, PIDTimer_last = 0;


// BATTERY
float battery_voltage, batt_read = 0.00, BatTime, BatTime_last = 0;



//PWM
float esc1, esc2, esc3, esc4, mappedThrottle;

//MOTORES
uint8_t M1 = D5;
uint8_t M2 = D6;
uint8_t M3 = D7;
uint8_t M4 = D8;

void setup() {
  init_motors();
  init_GPS();
  Wire.begin();           //Inicializar la librería Wire
  Serial.begin(115200);   //Establece la velocidad, en baudios (bits por segundo), para la
comunicación de datos en serie
  conf_MPU();
  pinMode(LED_BUILTIN, OUTPUT);  //Se inicializa el led de la placa como salida
  digitalWrite(LED_BUILTIN, HIGH);    //Se apaga el led
  lcd.init();       //Activar la pantalla
  lcd.backlight();  //Activar la luz de fondo
  init_WifiConnection();
  init_MPU6050();
  lcd.setCursor(0, 1);
  lcd.print("Buscan satelites");
  Serial.print("Buscando satelites");
  while (latitude == 0.00000) {   //Se espera hasta recibir coordenadas reales al GPS
    GPS_GetData();
    Serial.print(".");
    delay(250);
  }
  //Se da nombre y formato al archivo de grabación de coordenadas
  filename =
"/"+String(day)+"/"+String(month)+"/"+String(year)+"_"+String(hour)+":"+String(minute)+":"+
String(second)+".txt";
  Serial.println("");
  SPIFFS.begin();       //Se Inicializa el sistema de archivos
  server.on("/dronePar", handleParemeters);     //La subrutina a ejecutar al solicitar
/dronePar
  server.on("/formatFS", formatFS);
  server.on("/directoryFS", directoryFS);
  server.on("/readFile", readFile);
  server.begin();                               //Inicializa el servidor web
  digitalWrite(LED_BUILTIN, LOW);   //Se activa led de la placa indicando que ya se han
hecho todas las calibraciones
  Serial.println("Mover el Throttle a 1");
  lcd.setCursor(0, 1);
  lcd.print("Throttle a 1");
  while (consignaThrottle != 1) {               //Se espera hasta recibir el valor 1 en
Throttle por seguridad
    server.handleClient();                      //Se “escuchan” las peticiones URL
entrantes
    consignaThrottle = argumentThrottle.toFloat();
  }
  digitalWrite(LED_BUILTIN, HIGH);        //Se apaga el led conforme el Throttle ha llegado
a 1
  init_FS();
```

```
}

void loop() {

  WifiConnection_GetData();
  MPU6050_GetData();
  PID_ang();
  Modulador();
  GPS_GetData();
  batteryMonitoring();
}
```

B.2 Codigo Wifi y recepción de parametros

```
//====== Subrutina de configuración y conexión Wifi ======//
void init_WifiConnection() {

  WiFi.mode(WIFI_STA);          //Establece la placa como cliente Wifi
  WiFi.begin(ssid, password);   //Conectar al Wifi con el ssid y el password indicados

  lcd.clear();                 //Borrar el contenido anterior de la pantalla
  lcd.setCursor(0, 0);         //Indicar la posición del cursor: 0 horizontal y 0 vertical
  lcd.print("Esperando a ");   //visualizar en pantalla desde la posición marcada
  lcd.setCursor(0, 1);
  lcd.print("conectarse...");

  while (WiFi.status() != WL_CONNECTED) {   //Esperar hasta estar conectado al Wifi
    delay(500);
    Serial.println("Esperando a conectarse");
  }
  Serial.print("Dirección IP:  ");
  Serial.println(WiFi.localIP());

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(WiFi.localIP());            //Mostrar en pantalla la IP local generada
}

//====== Subrutina de conversión datos de lectura URL======//
void WifiConnection_GetData()
{
  server.handleClient();
  consignaPitch = argumentPitch.toFloat();  //Se convierte la variable de String a float
  consignaRoll = argumentRoll.toFloat();
  consignaThrottle = argumentThrottle.toFloat();
  consignaYaw = argumentYaw.toFloat();

  PIDSelected = PIDSelectedString.toInt();
  if (PIDSelected == 0) {
    KpPitch = KpString.toFloat();
    KiPitch = KiString.toFloat();
    KdPitch = KdString.toFloat();
  }
  else if (PIDSelected == 1) {
    Kp_yaw_w = KpString.toFloat();
    Ki_yaw_w = KiString.toFloat();
    Kd_yaw_w = KdString.toFloat();
  }
  else if (PIDSelected == 2) {
    KpRoll = KpString.toFloat();
    KiRoll = KiString.toFloat();
    KdRoll = KdString.toFloat();
  }
}

//====== Subrutina de lectura de petición URL======//
void handleParemeters()
{
```

```
  argumentPitch = server.arg("Pitch");    //Se lee el valor enviado del argumento Pitch
  argumentRoll = server.arg("Roll");
  argumentThrottle = server.arg("Throttle");
  argumentYaw = server.arg("Yaw");
  KpString = server.arg("Kp");
  KiString = server.arg("Ki");
  KdString = server.arg("Kd");
  PIDSelectedString = server.arg("PIDSelected");
  server.send(204);     //Respuesta del servidor a la petición URL   , "text/plain", ""
}
```

B.3 Codigo IMU

```
//====== Subrutina de configuración del MPU-6050======//
void conf_MPU() {
  Wire.beginTransmission(MPU6050_address);    //Iniciar comunicación con el MPU-6050
  Wire.write(0x6B);                           //Entrar en la configuración
  Wire.write(0x00);                           //Reiniciar el sensor para configurarlo
  Wire.endTransmission();                     //Cerrar comunicación con el MPU-6050
  //Por cada parámetro a configurar abrimos y cerramos la comunicación
  Wire.beginTransmission(MPU6050_address);
  Wire.write(0x1B);                           //Se abre configuración del giroscopio
  Wire.write(0x08);                           //Sensibilidad giroscopio: 0x08 => +/- 500°/s
  Wire.endTransmission();

  Wire.beginTransmission(MPU6050_address);
  Wire.write(0x1C);                          //Se abre configuración del acelerómetro
  Wire.write(0x10);                          //Sensibilidad acelerómetro: 0x10 => +/- 8g
  Wire.endTransmission();

  Wire.beginTransmission(MPU6050_address);
  Wire.write(0x1A);                        //Se abre configuración del filtro pasa
bajo(LFP)
  Wire.write(0x03);                        //LFP: 0x03 => 42Hz( con retraso de 4.9ms)
  Wire.endTransmission();
}

//====== Subrutina de obtención del offset del MPU-6050======//
void init_MPU6050() {
  Serial.println("Calibrando gyro MPU6050...");

  lcd.setCursor(0, 1);        //Se posiciona el cursor en vertical 1 para no borrar la IP
  lcd.print("Midiendo Offset");

  ESP.wdtDisable();     //Se desactiva el watchdog de software.

  for (count = 0; count < 3000 ; count ++) {

    MPU_6050();             //Salta a la subrutina llamada MPU_6050 (Obtención de lecturas)

    offset_pitch_gyro += gx;   //Suma en offset_pitch_gyro todos los valores de gx
    offset_roll_gyro += gy;
    offset_yaw_gyro += gz;

    offset_pitch_acce += ax;
    offset_roll_acce += ay;
    offset_yaw_acce += az;
    delayMicroseconds(1000);  //Espera 1 milisegundo a hacer las siguientes lecturas
  }

  offset_pitch_gyro = offset_pitch_gyro / 3000; //Media aritmética de todas las lecturas gx
  offset_roll_gyro = offset_roll_gyro / 3000;
  offset_yaw_gyro = offset_yaw_gyro / 3000;

  offset_pitch_acce = offset_pitch_acce / 3000;
  offset_roll_acce = offset_roll_acce / 3000;
  offset_yaw_acce = offset_yaw_acce / 3000;
```

```
  ESP.wdtEnable(0);          //Se vuelve a activar el watchdog de software

  lcd.setCursor(0, 1);
  lcd.print("Offset medido");
}

//====== Subrutina de cálculo de ángulos MPU-6050======//
void MPU6050_GetData() {

  //Tiempo transcurrido desde la última lectura
  execution_time = (micros() - loop_timer) / 1000;
  loop_timer = micros();

  MPU_6050();    //Salta a la subrutina llamada MPU_6050 (Obtención de lecturas)

  //Correción del offset
  ax -= offset_pitch_acce;  //Ajustamos la lectura con el offset
  ay -= offset_roll_acce;
  az -= offset_yaw_acce;
  az = az + 4096;

  gx -= offset_pitch_gyro;
  gy -= offset_roll_gyro;
  gz -= offset_yaw_gyro;

  //Cálculo de la velocidad de rotación
  w_pitch = gx / 65.5; // 65.5: si leo 65.5 en gx, significa que gira a 1°/s
  w_roll = gy / 65.5;
  w_yaw = gz / 65.5;

  //Calcular pitch y roll con el giroscopio
  angle_pitch += w_pitch * execution_time/1000;
  angle_roll += w_roll * execution_time/1000;

  //Calcular pitch y roll con el acelerómetro. 57,2958 = 180/pi (de radianes a grados)
  accel_ang_pitch = atan(ay / sqrt(pow(ax, 2) + pow(az, 2)))*57.2958;
  accel_ang_roll = atan(-ax / sqrt(pow(ay, 2) + pow(az, 2)))*57.2958;

  //Filtro complementario para corrección del drift
  angle_pitch = angle_pitch * 0.98 + accel_ang_pitch * 0.02;
  angle_roll = angle_roll * 0.98 + accel_ang_roll * 0.02;
}


//====== Subrutina de obtención de lecturas del MPU-6050======//
void MPU_6050() {
  Wire.beginTransmission(MPU6050_address);
  Wire.write(0x3B);                          //Se indica el byte de inicio de la lectura
  Wire.endTransmission();
  Wire.requestFrom(MPU6050_address, 14);     //Petición de 14 bytes desde el byte indicado
  while (Wire.available() < 14);             //Se espera hasta recibir los 14 bytes
  //Se guarda, en variables int de 16 bits, cada dos bytes leídos del sensor
  AccelX = Wire.read() << 8 | Wire.read();      //0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AccelY = Wire.read() << 8 | Wire.read();      //0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AccelZ = Wire.read() << 8 | Wire.read();      //0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Temperature = Wire.read() << 8 | Wire.read(); //0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyroX = Wire.read() << 8 | Wire.read();       //0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyroY = Wire.read() << 8 | Wire.read();       //0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyroZ = Wire.read() << 8 | Wire.read();       //0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)

  //Las variables anteriores se convierten a tipo float para facilitar el cálculo posterior
  ax = (float)AccelX;
  ay = (float)AccelY;
  az = (float)AccelZ;
  temperature = (float)Temperature;
  gx = (float)GyroX;
  gy = (float)GyroY;
  gz = (float)GyroZ;
}
```

B.4 Codigo PID

```
// Parámetros limite
int I_max = 300;  // Limitar parte integral (anti-wind-up)
int I_min = -300;
int PID_max = 400;  // Limitar salida del PID (anti-wind-up)
int PID_min = -400;


//====== Subrutina de control PID ======//
void PID_ang() {
  //Esperar a tiempo transcurrido desde la última llamada a subrutina > 12ms
  PIDTimer = (micros() - PIDTimer_last);
  if (PIDTimer < 12000){
    delayMicroseconds(12000 - PIDTimer);
  }
  PIDTimer_last = micros();

  //Obtención tiempo transcurrido desde el último control PID
  PIDTime = (micros() - PIDTime_last)/10000;  // Calculo en decenas de milisegundos
  PIDTime_last = micros();


  //Cálculo del error en cada eje de navegación
  pitch_angle_error = -consignaPitch - angle_pitch;
  roll_angle_error = consignaRoll - angle_roll;
  yaw_w_error = consignaYaw - w_yaw;

  //Cálculo del termino I
  if (mappedThrottle > 1550) {    //No aplicar termino I hasta cierta consigna Throttle
    I_pitch_angle += (KiPitch * pitch_angle_error*PIDTime);
    I_pitch_angle = constrain(I_pitch_angle, I_min, I_max);     //Limitar valores termino I

    I_roll_angle += (KiRoll * roll_angle_error*PIDTime);
    I_roll_angle = constrain(I_roll_angle, I_min, I_max);

    I_yaw_w += (Ki_yaw_w * yaw_w_error*PIDTime);
    I_yaw_w = constrain(I_yaw_w, I_min, I_max);
  }
  else {
    I_pitch_angle = 0;
    I_roll_angle = 0;
    I_yaw_w = 0;
  }

  //Cálculo del termino D
  D_pitch_angle = KdPitch * (angle_pitch - prev_angle_pitch)/PIDTime;
  D_roll_angle = KdRoll * (angle_roll - prev_angle_roll)/PIDTime;
  D_yaw_w = Kd_yaw_w * (w_yaw - prev_w_yaw)/PIDTime;

  //Cálculo PID
  PID_pitch_ang = KpPitch * pitch_angle_error + I_pitch_angle - D_pitch_angle;
  PID_pitch_ang = constrain(PID_pitch_ang, PID_min, PID_max);  //Limitar valores salida PID

  PID_roll_ang = KpRoll * roll_angle_error + I_roll_angle - D_roll_angle;
  PID_roll_ang = constrain(PID_roll_ang, PID_min, PID_max);

  PID_yaw_w = Kp_yaw_w * yaw_w_error + I_yaw_w - D_yaw_w;
  PID_yaw_w = constrain(PID_yaw_w, PID_min, PID_max);

  //Variables para el siguiente ciclo
  prev_angle_pitch = angle_pitch;
  prev_angle_roll = angle_roll;
  prev_w_yaw = w_yaw;
}
```

B.5 Codigo modulación de la señal para el actuador

```
//====== Subrutina de modulación salida ======//
void Modulador () {

  mappedThrottle = map(consignaThrottle, 0, 100, 1500, 1800); //(in value, min in value,
max in value, min out value, max out value) (consignaThrottle, 0, 100, 1000, 1800)
  // Limitar throttle a 1800 para dejar margen a los PID

  if (mappedThrottle <= 1300) {    //if (mappedThrottle <= 1550 && HoldStatus == 0)
    esc1 = 1000;
    esc2 = 1000;
    esc3 = 1000;
    esc4 = 1000;
  }

  // Si el throttle es mayor a 1300us, el control de estabilidad se activa.
  if (mappedThrottle > 1300) {  //(mappedThrottle > 1550 && HoldStatus == 0)
    if (mappedThrottle > 1800)mappedThrottle = 1800; // Limitar throttle a 1800 para dejar
manrgen a los PID
    esc1 = mappedThrottle + PID_pitch_ang - PID_roll_ang - PID_yaw_w; // Motor 1
    esc2 = mappedThrottle + PID_pitch_ang + PID_roll_ang + PID_yaw_w; // Motor 2
    esc3 = mappedThrottle - PID_pitch_ang + PID_roll_ang - PID_yaw_w; // Motor 3
    esc4 = mappedThrottle - PID_pitch_ang - PID_roll_ang + PID_yaw_w; // Motor 4
    if (esc1 < 1100) esc1 = 1100; //Se evita que alguno de los motores de detenga
    if (esc2 < 1100) esc2 = 1100;
    if (esc3 < 1100) esc3 = 1100;
    if (esc4 < 1100) esc4 = 1100;
    if (esc1 > 2000) esc1 = 2000; //Se evita mandar consignas mayores a 2000us
    if (esc2 > 2000) esc2 = 2000;
    if (esc3 > 2000) esc3 = 2000;
    if (esc4 > 2000) esc4 = 2000;
  }

  //Salidas PWM
  analogWrite(M1, esc1);
  analogWrite(M2, esc2);
  analogWrite(M3, esc3);
  analogWrite(M4, esc4);
}


//====== Subrutina de inicialización PWM y motores ======//
void init_motors() {
  analogWriteRange(2500); //Rango de 2500 valores
  analogWriteFreq(400); //ciclos de 400Hz (periodo de 2,5ms)
  analogWrite(M1, 1000);
  analogWrite(M2, 1000);
  analogWrite(M3, 1000);
  analogWrite(M4, 1000);
}
```

B.6 Codigo control de nivel de batería

```
//====== Subrutina de cálculo voltaje batería======//
void batteryMonitoring() {
  batt_read = analogRead(A0); // Leer entrada analógica
  battery_voltage = 3.73 * (batt_read * 3.3 / 1023);

  // Si la tensión es inferior a 10.8V, se activa la señal de batería baja
  if (battery_voltage < 10.8) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("¡¡Batería baja!!");
    lcd.setCursor(0, 1);
    lcd.print("¡Apagar el dron!");
```

```
    BatTime = (millis() - BatTime_last);
    if (BatTime < 500)
      digitalWrite(LED_BUILTIN, LOW);
    else if (BatTime < 1000)
      digitalWrite(LED_BUILTIN, HIGH);
    else if (BatTime >= 1000)
      BatTime_last = millis();
  }
}
```

B.7 Codigo para GPS

```
//====== Subrutina de iniciación puerto serie GPS ======//
void init_GPS() {
  serialgps.begin(9600);//Iniciamos el puerto serie del gps
  //Imprimimos:
}

//====== Subrutina de procesamiento y obtención de los datos GPS ======//
void GPS_GetData() {
  while(serialgps.available())
  {
    int c = serialgps.read();

    if(gps.encode(c))
    {
    gps.f_get_position(&latitude, &longitude); //Obtención de posición
    gps.crack_datetime(&year,&month,&day,&hour,&minute,&second,&hundredths); //Obtención de
fecha y hora
    gps.stats(&chars, &sentences, &failed_checksum);
    }
  }
}

//====== Subrutina de cálculo de distancia entre dos coordenadas geográficas ======//
void Compare_GPSData() {

  //Se convierte la última coordenada geográfica recibida de angular a radial
  longitudeR = longitude*(PI/180);
  latitudeR = latitude*(PI/180);

  //Se convierte la última coordenada geográfica guardada de angular a radial
  longitudeLR = longitudeL*(PI/180);
  latitudeLR = latitudeL*(PI/180);

  //Se restan las coordenadas
  dlongitude = longitudeLR - longitudeR;
  dlatitude = latitudeLR - latitudeR;

  //Formula Haversine para distancia entre dos puntos. Distancia dada en metros
  a = (sq(sin(dlatitude/2))) + cos(latitudeR) * cos(latitudeLR) * (sq(sin(dlongitude/2)));
  c = 2 * atan2(sqrt(a), sqrt(1-a)) ;
  d = R * c;   //R es el radio de la tierra (6371000m)

  if (d>2){
    FS_SaveData();        //Se guarda la nueva coordenada
    longitudeL = longitude; //Se actualiza la última coordenada guardada
    latitudeL = latitude;
  }
}
```

B.8 Codigo SPIFFS

```
//====== Subrutina de iniciación de SPIFFS ======//
void init_FS() {
  //Crear un archivo nuevo, le damos nombre y escribir en él
  File f=SPIFFS.open(filename,"w");
  if(!f) {
    Serial.println("file open failed");
  }
  else {
    f.println("Latitud;Longitud");  //Escribe el nombre de las columnas
    f.close();    //Cerramos el archivo creado
  }
}

//====== Subrutina de guardar datos ======//
void FS_SaveData() {
  File f=SPIFFS.open(filename,"a"); //Abrimos el archivo para escribir al final de este
  if(!f) {
    Serial.println("file open failed");
  }
  else {
    Serial.println("writing Data to File");  //Escribimos
    f.print(latitude);    //Escribimos los datos deseados
    f.print(";");
    f.println(longitude);
  }
  f.close();
}

//====== Subrutina de formateo de SPIFFS ======//
void formatFS()
{
  lcd.setCursor(0, 1);
  lcd.print("Formateando...");
  Serial.println("Wait around 30 seconds for format");
  SPIFFS.format();
  Serial.println("SPIFFS formated");
  lcd.setCursor(0, 1);
  lcd.print("¡¡¡Formateado!!!");
  server.send(200, "text/plain", "SPIFFS formated");
}

//====== Subrutina de envío del directorio existente ======//
void directoryFS()
{
  Dir dir = SPIFFS.openDir("");
  while (dir.next()) {
    directory = directory + "\n"+ dir.fileName();
  }
  server.send(200, "text/plain", directory);
  lcd.setCursor(0, 1);
  lcd.print("Direct. enviado");
  directory = "";
}

//====== Subrutina de lectura del archivo pedido ======//
void readFile()
{
if (!SPIFFS.begin()) {
    Serial.println("SPIFFS failed to mount !\r\n");
  }
  else {
    String str = "";
    File f = SPIFFS.open(server.arg(0), "r");
    if (!f) {
      server.send(200, "Can't open SPIFFS file !\r\n");
    }
    else { //Se guarda en la variable str el contenido del archivo para enviarlo
      char buf[1024];
```

```
      int siz = f.size();
      while(siz > 0) {
        size_t len = std::min((int)(sizeof(buf) - 1), siz);
        f.read((uint8_t *)buf, len);
        buf[len] = 0;
        str += buf;
        siz -= sizeof(buf) - 1;
      }
      f.close();
      server.send(200, "text/plain", str); //Responde a la petición con str
      lcd.setCursor(0, 1);
      lcd.print("Archivo enviado");
    }
  }
}
```