Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Disseny i implementació d'un nou protipus de dron quadrirotor controlat mitjançant STM32 usant l'entorn Arduino.

TITULACIÓN: Grado en Ingeniería de Aeronavegación.

AUTOR: León Enrique Prieto Bailo.

DIRECTOR: Ramon Casanella Alonso.

FECHA: 7 de julio del 2023.

Título: Diseño e implementación de un nuevo prototipo de dron cuadricóptero controlado mediante STM32 utilizando el entorno de Arduino.

Autor: León Enrique Prieto Bailo.

Director: Ramon Casanella Alonso.

Fecha: 7 de julio del 2023.

Resumen

La familia de microcontroladores STM32 de 32 bits es la más ampliamente utilizada para el diseño de controladores de drones en la actualidad. Por otro lado, Arduino es una plataforma de programación de microcontroladores de código abierto y costo asequible. En este trabajo se propone el diseño e implementación de un dron cuadricóptero controlado mediante un microcontrolador de la familia STM32 programado usando el entorno Arduino. En él se han desarrollado y validado los diferentes elementos que componen un sistema cuadricóptero y se han integrado con el fin de obtener un prototipo final plenamente funcional que permita su reconfiguración y mejora en función de necesidades futuras. Finalmente, sobre la configuración base desarrollada, se ha diseñado e integrado un sistema adicional de control de altitud basado en la información de un sensor barométrico.

Title: Design and implementation of a new quadcopter drone prototype controlled through STM32 using the Arduino environment.

Author: León Enrique Prieto Bailo.

Director: Ramon Casanella Alonso.

Date: July 7th, 2023.

Overview

The 32-bit STM32 microcontroller family is the most widely used for drone controller design today. On the other hand, Arduino is an open-source and affordable microcontroller programming platform. This work proposes the design and implementation of a quadcopter drone controlled by an STM32 family microcontroller programmed using the Arduino environment. Different components of a quadcopter system have been developed and validated, and they have been integrated to obtain a fully functional final prototype that allows reconfiguration and improvement according to future needs. Finally, based on the developed base configuration, an additional altitude control system has been designed and integrated using information from a barometric sensor.

ÍNDICE

Agradecimientos

Quiero empezar agradeciendo al director de este TFG. Hace casi dos años que tuvimos un primer contacto para hacer este trabajo y desde ese mismo momento, me has ayudado con todas mis necesidades y siempre has tenido tiempo cuando lo he necesitado. El resultado de este proyecto hubiera sido imposible sin tu dedicación y ambición. Gracias, Ramon.

También quiero expresar mi profundo agradecimiento hacia mi familia. A mi padre, quien despertó mi pasión por la aeronáutica a través de sus cautivadoras historias y emocionantes aventuras. A mi madre, quien siempre ha estado a mi lado, brindándome apoyo incondicional y animándome a seguir adelante incluso en los momentos más difíciles. Y a mi hermano, quien ha sido mi fiel compañero a lo largo de toda mi vida, caminando a mi lado en cada paso del camino.

Finalment, vull fer una especial menció al meu professor de matemàtiques de batxillerat, en Jordi Caelles Abillar. Vull agrair-te en nom de tots els estudiants als quals sempre has motivat, ajudat, i ens has despertat l'interès de les matemàtiques. Potser no ho saps, però me'n recordo moltíssim de tu i sé que sense el teu esforç, no hauria arribat on soc. Aquest èxit, també és teu. Gràcies.

CAPÍTULO 1. INTRODUCCIÓN

En los últimos años, la industria de los drones ha experimentado un crecimiento importante, revolucionando diversos sectores y abriendo nuevas posibilidades en la forma en que nos desplazamos, realizamos tareas y experimentamos el mundo desde el aire. Este auge se ha visto impulsado por avances tecnológicos clave, como el desarrollo de sistemas sofisticados para evitar impactos, regular la altitud, habilitar el vuelo automático, entre otros.

Fig. 1.1. DJI Matrice 600, equipado con una camara.

Gracias a estos avances tecnológicos los cuales incrementan la seguridad de la operación de drones, aparecen conceptos nuevos como la creación del U-Space [1]. El U-Space representa un conjunto de servicios y procedimientos basados en un alto nivel de automatización de funciones con el objetivo de proporcionar, de forma regulada y coordinada, un acceso seguro y eficiente al espacio aéreo a un gran número de UAS. Por ello es cada vez más necesario que los drones incorporen funciones avanzadas de detección de impactos, control de automático de altitud o vuelo totalmente automático mediante GPS.

Uno de los elementos fundamentales que ha contribuido al éxito y la eficiencia de los drones son los avances tecnológicos en la industria de los microcontroladores. Estos dispositivos electrónicos son el elemento central de la controladora de vuelo y son responsables de controlar y coordinar las diferentes funciones del dron, permitiendo un vuelo seguro y preciso. Si bien existen diversas familias de microcontroladores en el mercado, uno de los más populares y ampliamente utilizados es la familia STM32 [2].

La familia STM32 ha destacado por su rendimiento confiable y su capacidad para adaptarse a una amplia gama de aplicaciones. Estos microcontroladores ofrecen

un equilibrio entre potencia de procesamiento, consumo de energía y costo, lo que los convierte en una elección preferida para muchos fabricantes de drones. Su versatilidad y robustez los hacen ideales para implementar funciones críticas, como el control de motores, la comunicación inalámbrica y la gestión de sensores.

A medida que la industria de los drones sigue creciendo y evolucionando, también han surgido diferentes entornos de configuración para los microcontroladores de los drones. Dos de los más comunes son Betaflight [3] y ArduPilot [4]. Betaflight se utiliza principalmente en drones más pequeños y orientados a vuelos de alta velocidad y acrobacias, conocidos como FPV. Los drones FPV (First Person View) son drones que, mediante el uso de unas gafas, proporcionan experiencias de vuelo inmersivas y en tiempo real, en primera persona. Por otro lado, ArduPilot se utiliza en drones de mayor tamaño y en aplicaciones más profesionales, ofreciendo opciones de personalización.

La ventaja principal de emplear este tipo de software es que permiten programar una controladora de vuelo compatible mediante una interfaz gráfica, lo que facilita enormemente las tareas de configuración al usuario final. Sin embargo, es este mismo hecho el que limita las posibilidades de realizar implementaciones customizadas, ya que los marcos de trabajo son limitados. Esto ha llevado a la búsqueda de alternativas más personalizables que permitan un control más preciso y adaptable para satisfacer las necesidades específicas de cada usuario.

Con el objetivo de superar las limitaciones y ofrecer un entorno más flexible para los desarrolladores con experiencia en programación, se ha propuesto una solución que se basa en experiencias previas exitosas. Estos proyectos corresponden al dron con un Arduino Nano de 8 bits de Arduproject [5] y al dron con un STM32F103 de Joop Brokking [6]. Esta propuesta consiste en desarrollar el código necesario en el entorno Arduino [7] para el microcontrolador STM32F405, que actualmente se utiliza en controladoras de vuelo comerciales de drones.

Al utilizar el entorno Arduino, se proporciona a los desarrolladores una plataforma familiar y altamente versátil para programar y configurar sus drones. Esto permite aprovechar el amplio ecosistema de librerías y herramientas disponibles en Arduino, así como la experiencia previa en programación de microcontroladores STM32.

Además de la configuración necesaria mínima para tener un dron funcional, esta propuesta destaca por su flexibilidad al permitir implementar características adicionales. Un ejemplo concreto es la incorporación de un sistema de control de altitud utilizando un sensor barométrico. Esta funcionalidad, que no sería fácilmente desarrollada en entornos como Betaflight o Ardupilot, demuestra la capacidad de adaptación y personalización que ofrece el entorno Arduino en combinación con el microcontrolador STM32F405.

La implementación de un control de altitud más preciso y sofisticado, basado en lecturas de presión, brinda nuevas oportunidades en términos de estabilidad y rendimiento durante el vuelo. Esto puede ser especialmente relevante en aplicaciones que requieren un vuelo controlado y preciso, como la captura de imágenes aéreas o la inspección de infraestructuras.

1.1. Objetivos

El objetivo principal de este trabajo es el diseño e implementación de un dron cuadricóptero controlado mediante un microcontrolador de la familia STM32, el STM32F405, programado usando el entorno Arduino. Para ello, se deben conseguir los siguientes objetivos parciales:

- Poder ensamblar, a partir de una estructura de hardware previa definida por la universidad, los distintos elementos de funcionales que componen un dron. Se deberá también verificar que elementos de propulsión usados y el sistema alimentación implementados son suficientes para conseguir una maniobrabilidad y tiempo de vuelo que estén dentro de los parámetros habituales.

- Usando el entorno Arduino, desarrollar un software de control de vuelo que permita el vuelo funcional y estable del dron. La programación realizada deberá ser fácilmente reconfigurable para permitir incorporar futuras funcionalidades al dron.

Finalmente, sobre la configuración base desarrollada, se plantea el objetivo adicional de desarrollar e implementar alguna funcionalidad extra, como la de un control de altitud y aterrizaje automático basado en la información de un sensor barométrico o un sistema de ultrasonidos.

1.2. Metodología

Para realizar un desarrollo apropiado del software, se ha optado por una estrategia basada en implementar funcionalidades de carácter modular y fragmentado e ir probando los resultados de manera exhaustiva. Este procedimiento consiste en realizar implementaciones lo más pequeñas posibles y llevar a cabo las pruebas necesarias para verificar el correcto funcionamiento. Emplear esta estrategia ha sido realmente conveniente ya que ha sido muy útil para hallar las problemáticas que podían aparecer durante el desarrollo. Además, emplear una estructura modular permite implementar funcionalidades adicionales como un modo de vuelo “Altitude Hold”, basado en las lecturas de presión de un barómetro, a medida que se va desarrollando el proyecto

1.3. Estructura de la memoria

La memoria se estructura en tres capítulos principales, comenzando con el diseño de hardware del dron. En este primer capítulo, se presentan las características técnicas de los componentes que conforman el hardware del dron, brindando una explicación detallada de su funcionamiento y justificando su viabilidad tanto como elementos individuales como en conjunto como sistema integrado. Además, se incluye al final del capítulo un esquema eléctrico que muestra la interconexión entre los diversos elementos de hardware del sistema, proporcionando una visión completa de su configuración.

El segundo capítulo se centra en el diseño de software, en él se profundiza en la integración detallada de los componentes de hardware dentro del programa del microcontrolador. La organización de este capítulo se basa en la estructura del algoritmo de control implementado en el software. En la descripción de la arquitectura y el desarrollo, se presenta la estructura utilizada para la generación de las señales y se explica cómo contribuyen al algoritmo de control. Posteriormente, se hace un análisis más profundo del funcionamiento de cada uno de los módulos, se analizan las subrutinas principales y se realiza una explicación detallada de la implementación.

En el tercer capítulo de este proyecto, se recopila información acerca del prototipo final y se detalla la realización de la operación de vuelo. Esta sección contiene imágenes del prototipo y los costes de los componentes que confirman el dron. Adicionalmente, se abordan distintos aspectos clave como los controles de vuelo, la puesta en marcha del dron, el calibrado de los PID, las instrucciones de vuelo y finalmente, se describe el proceso y los resultados obtenidos durante la operación del cuadricóptero.

Finalmente, la memoria incluye las conclusiones que exponen los resultados generales del trabajo, las impresiones obtenidas y se marcan las posibles direcciones a seguir para futuras ampliaciones del proyecto.

CAPÍTULO 2. DISEÑO DE HARDWARE

2.1. Descripción de los componentes del dron.

El hardware de un dron responde a la necesidad de implementar un sistema de alimentación y control de los motores de la aeronave para que pueda ser pilotada mediante radiocontrol a la vez que es capaz de mantenerse estable frente a perturbaciones externas. En la Fig. 2.1 se muestra la estructura general de un dron, cuyos elementos principales se explican a continuación.

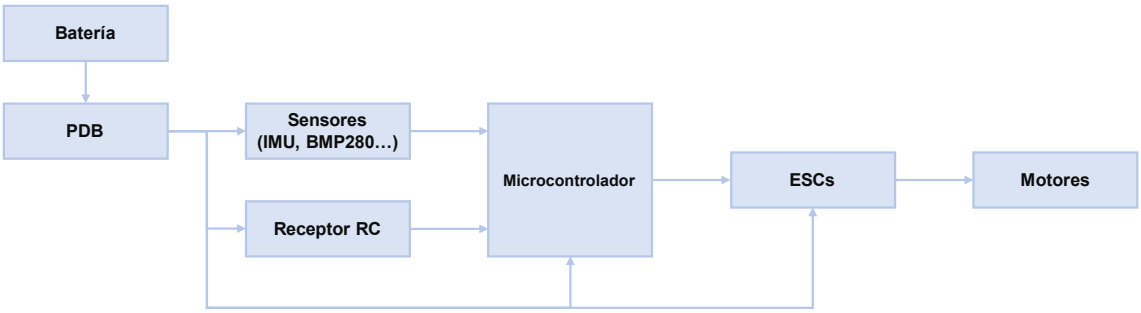

Fig. 2.1. Esquema general del sistema que conforma el dron.

- El microcontrolador: Es posiblemente la pieza más importante de un dron debido a su función central de controlar y coordinar todas las operaciones del vehículo aéreo no tripulado. Es responsable del control de vuelo, procesamiento de datos, comunicación y seguridad del dron. Controla los motores y actuadores, procesa la información de los sensores, toma decisiones en tiempo real, facilita la comunicación entre componentes y con dispositivos externos, y garantiza la seguridad y protección del dron.

- Los sensores: Los elementos sensoriales de un dron desempeñan un papel fundamental en su funcionamiento. Dentro de estos elementos podemos encontrar la IMU o el barómetro, entre otros. La IMU garantiza la estabilidad y orientación proveyendo al microcontrolador de lecturas de aceleración y velocidad angular. El barómetro se emplea para proveer lecturas de altura basadas en la presión lo que permite realizar controles de altitud.

- Receptor RC: El receptor de control remoto es el componente que actúa como el enlace de comunicación entre el operador y el dron. El receptor RC recibe las señales de control enviadas por el control remoto y las transmite al sistema de control del dron.

- La batería: El sistema de alimentación es el componente encargado del suministro de energía necesario para su funcionamiento. Proporciona la

electricidad requerida para alimentar los motores, los sistemas electrónicos y los componentes del dron, asegurando una fuente confiable y constante de energía durante el vuelo. Se conecta al resto de los sistemas del dron mediante la PDB.

- La PDB: La Power Distribution Board (PDB) es un componente que distribuye de manera segura la energía eléctrica de un sistema. Actúa como un centro de distribución conectando los cables de alimentación y ramificándolos hacia diferentes dispositivos.

- Las ESCs: Los controladores electrónicos de velocidad (ESC, por sus siglas en inglés) se encargan de regular y controlar la velocidad de los motores. Los ESC reciben las señales de control provenientes del sistema de control del dron y las utilizan para ajustar la velocidad de los motores de forma individual. Esto permite al dron realizar maniobras precisas, cambios de dirección y ajustes de velocidad según las instrucciones del piloto.

- Los motores: Los motores son los componentes encargados de la actuación de la fuerza propulsora necesaria para el vuelo. Los motores convierten la energía eléctrica suministrada por la batería en energía mecánica, que se utiliza para hacer girar las hélices del dron. Los motores permiten al dron desplazarse verticalmente y cambiar de dirección mediante el control de su velocidad de rotación.

La elección de los componentes de hardware ha sido directamente proporcionada por la universidad por lo que no se han realizado tareas de optimización en lo que se refiere a la selección de componentes electrónicos y estructurales del dron.

A continuación, se explican de forma detallada los componentes específicos que han sido utilizados en la implementación del dron desarrollado en este proyecto.

2.1.1. FRAME DJI-F450

El frame DJI-F450 [8] es un frame diseñado por la empresa DJI, muy reconocida dentro de la industria de los drones. Este frame tiene infinidad de aplicaciones como la fotografía y videografía aérea, el mapeo o la vigilancia entre otros. El marco está fabricado con la aleación PA66+30GF, la cual es una mezcla de poliamida 66 (nylon 66) con un 30% en peso de fibra de vidrio como refuerzo. La poliamida 66 es un polímero termoplástico que posee una alta resistencia mecánica, rigidez y resistencia al calor. La fibra de vidrio, por otro lado, se utiliza como material de refuerzo para mejorar las propiedades mecánicas del polímero.

El DJI-F450 está diseñado para ser fácilmente ensamblado y desmontado, lo que permite un fácil mantenimiento y reparación en caso de ser necesario además de ser económico y fácil de adquirir. Su tamaño compacto lo hace fácil de transportar y almacenar.

También es compatible con una amplia gama de componentes electrónicos, incluyendo controladores de vuelo, motores, ESC, baterías y equipos de radiocontrol. La compatibilidad con diferentes componentes hace que sea fácil personalizar y actualizar el dron según las necesidades específicas del usuario.

Además de las características mencionadas anteriormente, es importante destacar que el DJI-F450 es también un marco espacioso que permite añadir una multitud de componentes electrónicos. Este espacio adicional ofrece la flexibilidad necesaria para personalizar el dron según las necesidades del usuario, permitiendo la integración de sistemas de posicionamiento global (GPS), sensores de distancia, iluminación LED y otros componentes que pueden mejorar la funcionalidad del dron.

Otra ventaja del Frame DJI-F450 es su diseño integrado de PCB, que permite la conexión segura y ordenada de los componentes electrónicos. Este diseño optimizado no solo simplifica el cableado de los ESC y la batería, sino que también proporciona un aspecto más limpio y profesional al dron.

En la Fig. 2.2 se puede ver el frame con ESCs, motores y propellers.

Fig. 2.2. Frame DJI-F450.

2.1.2. Adafruit Feather STM32F405

La Adafruit STM32F405 [9] es una placa de desarrollo que se basa en el microcontrolador STM32F405RG de STMicroelectronics. Este microcontrolador pertenece a la familia STM32FX, la cual es típicamente seleccionada como controladora de vuelo dentro del mundo de los drones debido a la potencia de procesamiento, eficiencia energética y versatilidad

La placa de desarrollo de Adafruit se caracteriza por su facilidad de uso y su amplia gama de características, lo que la hace adecuada para una gran variedad de proyectos. En su núcleo, el STM32F405RG es un microcontrolador ARM Cortex-M4 de 32 bits, que opera a una frecuencia de 168 MHz y tiene una memoria RAM total de 192KB. Además de su capacidad de punto flotante, también incluye numerosos periféricos integrados.

La conectividad es una de las fortalezas del Adafruit STM32F405. La placa cuenta con puertos USB-C, UART, SPI, I2C y GPIO, que permiten una fácil comunicación con otros dispositivos. Los pines operan a una tensión de 3,3 V aunque la mayoría también admiten hasta tensiones de 5 V. También incluye un conector microSD para almacenamiento externo, brindando opciones adicionales para la expansión de memoria.

Una de las ventajas de utilizar el Adafruit STM32F405 es su compatibilidad con el entorno de desarrollo Arduino. Adafruit ha desarrollado una librería que permite programar el microcontrolador utilizando el lenguaje de programación y las herramientas familiares de Arduino. Esto simplifica el proceso de desarrollo y programación, especialmente para aquellos que ya están familiarizados con el ecosistema de Arduino.

Además de las características principales, la placa Adafruit STM32F405 incluye componentes adicionales como un chip SPI Flash de 2MB y un led RGB. Estos componentes ofrecen funcionalidades adicionales y expanden las posibilidades del proyecto.

Con una amplia variedad de pines de entrada/salida (ver Fig. 2.3), el Adafruit STM32F405 brinda una gran flexibilidad para conectar y controlar dispositivos y periféricos externos. Esto permite la integración de una amplia gama de sensores, actuadores y otros dispositivos en los proyectos.

Fig. 2.3. Microcontrolador Adafruit Feather STM32F405

2.1.3. MPU6050

El MPU6050 [10] es un módulo de sensor de movimiento utilizado para medir la aceleración y la velocidad angular. Este módulo es especialmente popular en aplicaciones de robótica, drones, control de movimiento y realidad virtual.

El MPU6050 integra un acelerómetro y giroscopio de tres ejes en un solo chip. Esto permite medir la aceleración lineal y la velocidad angular en tres direcciones diferentes: X, Y y Z. El acelerómetro mide la aceleración lineal en cada uno de los ejes, mientras que el giroscopio mide la velocidad angular o la tasa de cambio del ángulo en cada eje.

Una de las características destacadas del MPU6050 es su capacidad para proporcionar mediciones en tiempo real con alta precisión y sensibilidad. Esto permite detectar movimientos y cambios de orientación con gran precisión, lo que resulta útil en aplicaciones como la estabilización de vuelo de drones, la detección de movimientos en juegos de realidad virtual y la navegación inercial en robótica.

Además de sus capacidades de medición, el MPU6050 también incluye características adicionales como un sensor de temperatura incorporado o la capacidad de ajustar diferentes rangos de medición y tasas de muestreo según las necesidades del proyecto.

Para establecer la conexión entre el MPU6050 y el microcontrolador, se utilizará el puerto I2C (Inter-Integrated Circuit). El puerto I2C permite una comunicación sencilla y eficiente entre dispositivos, y es compatible con el MPU6050. Mediante la programación adecuada, el microcontrolador podrá recibir datos del MPU6050 y realizar las acciones necesarias en función de las mediciones de aceleración y velocidad angular proporcionadas por el sensor. Esto permite aprovechar las capacidades del MPU6050 en aplicaciones como estabilización de vuelo, control de movimiento y detección de orientación precisa.

En la Fig. 2.4 se puede ver el chip con el MPU6050 seleccionado para este proyecto.

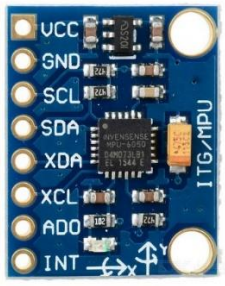

Fig. 2.4. IMU MPU6050

2.1.4. BMP280

El sensor BMP280 [11] es un sensor de presión y temperatura de alta precisión fabricado por Bosch Sensortec. Está diseñado para medir la presión barométrica y la temperatura en una amplia gama de aplicaciones, incluyendo la navegación, los drones y la meteorología. El BMP280 utiliza un principio de medición piezo-resistivo para medir la presión atmosférica con una precisión de hasta ±1 hPa y una resolución de 0.16 Pa, lo que lo hace ideal para aplicaciones en las que se necesita una medición precisa de la presión.

En cuanto a su aplicación en drones, el BMP280 se utiliza para medir la altura y la altitud del dron. Al medir la presión atmosférica, el BMP280 puede calcular la altitud del dron con una precisión razonable. Esta información complementada con un algoritmo de control de altitud se puede emplear para desarrollar modos de vuelo como el “Altitude Hold”, el cual permite maniobrar el dron a una altura constante, o el control de vuelo por GPS el cual permite mantener una posición en el espacio de manera constante.

Para los drones contemporáneos es prácticamente obligatorio que dispongan de lecturas de presión para aplicar este tipo de control sobre la altitud del dron. Todo esto tiene aplicaciones para cuadricópteros que realizan misiones de vigilancia, mapeo y fotografía aérea, ya que les permite mantener una altitud constante y controlar su posición con mayor precisión.

En este proyecto en cuestión el sensor BMP280 se utiliza en combinación con la MPU6050, con el propósito de implementar, de manera viable, un modo de vuelo “Altitude Hold” que permita al cuadricóptero mantenerse a una altura constante y moverse, en un plano horizontal, utilizando el joystick que controla roll, pitch y yaw. Para hacerlo, el controlador de vuelo dispondrá, en su algoritmo de control, de diferentes controladores PID que reaccionaran de manera independiente a las perturbaciones obtenidas por los sensores con el fin de realizar las correcciones pertinentes.

En la Fig. 2.5 se puede ver el chip con el BMP280.

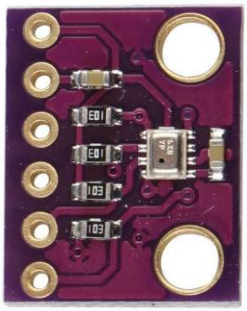

Fig. 2.5. Barómetro BMP280.

2.1.5. HC-SR04

El sensor de ultrasonidos HC-SR04 [12] es un dispositivo popular utilizado para medir distancias utilizando ondas ultrasónicas. Es ampliamente utilizado en proyectos de robótica, sistemas de seguridad y automatización, y en aplicaciones donde se requiere detección de objetos.

Como se puede ver en la Fig. 2.6, el HC-SR04 consta de dos componentes principales: un transmisor y un receptor. El transmisor emite pulsos ultrasónicos a una frecuencia de alrededor de 40 kHz los cuales son inaudibles para los humanos debido a su alta frecuencia. El receptor captura los pulsos ultrasónicos reflejados por un objeto y los convierte en señales eléctricas.

El funcionamiento del HC-SR04 se basa en el principio del tiempo de vuelo. Cuando se emite un pulso ultrasónico, se calcula el tiempo que tarda en rebotar en un objeto y regresar al sensor. Utilizando la velocidad del sonido en el aire, que es aproximadamente 343 m/s a una temperatura de 20 grados Celsius, se puede determinar la distancia al objeto.

Es importante destacar que el HC-SR04 tiene un rango de medición limitado y puede funcionar de manera confiable en distancias de 2 cm a 4 metros, aproximadamente. Además, el sensor puede verse afectado por el ángulo y la forma del objeto que se está detectando, así como por las condiciones ambientales, como la temperatura y la humedad.

Fig. 2.6. Sensor de ultrasonidos HC-SR04.

2.1.6. Flysky i6

La Flysky i6 [13] es una emisora de 6 canales diseñada para el pilotaje de modelos de radiocontrol, como aviones, helicópteros, drones y otros dispositivos RC. Es una radio muy popular entre los entusiastas de los vuelos de radiocontrol debido a la simplicidad de su manejo, versatilidad y precio asequible.

La Flysky i6 cuenta con una pantalla LCD retroiluminada y un diseño ergonómico que la hace cómoda de sostener y utilizar durante largos períodos de tiempo. Ofrece una interfaz intuitiva con botones de fácil acceso y un menú de

navegación sencillo, lo que facilita la configuración y personalización de los ajustes.

Una de las principales características de la Flysky i6 es su capacidad de operar usando el protocolo AFHDS 2A, que proporciona una comunicación de radio con frecuencia estable y confiable. Esto garantiza una respuesta precisa y rápida entre la emisora y el receptor, lo que resulta crucial para un control preciso del UAS.

El radiocontrol Flysky i6 ofrece una amplia gama de funciones y ajustes, como la asignación y mezcla de canales, la configuración de puntos de ajuste y límites, la selección de modos de vuelo (modo “Stable”, modo “Altitude Hold”, etc.) y la programación de diferentes modelos.

Adicionalmente, la Flysky i6 es compatible con varios receptores Flysky, lo que brinda flexibilidad para adaptarse a diferentes modelos y configuraciones. También ofrece la posibilidad de actualización de firmware, lo que permite agregar nuevas funciones y mejoras a medida que estén disponibles.

Respecto al procesado de la señal desde el microcontrolador, la Flysky es una radio que utiliza la modulación PPM (Pulse Position Modulation) para transmitir la información de control desde la emisora al receptor. La modulación PPM es un método eficiente que permite enviar múltiples señales de control a través de un solo cable, lo que ahorra el uso de pines del microcontrolador en el receptor. En lugar de tener un cable separado para cada canal de control, la Flysky utiliza una única señal PPM que lleva consigo la información de todos los canales de control en forma de pulsos de diferentes longitudes. Esto simplifica la conexión entre la emisora y el receptor, liberando pines en el microcontrolador para otros usos. Además, la modulación PPM permite una transmisión rápida y precisa de la información de control, lo que contribuye a una mayor eficiencia y capacidad de respuesta en el sistema de radiocontrol.

En la Fig. 2.7 se puede ver el radiocontrol seleccionado.

Fig. 2.7. Flysky Fs-i6

2.1.7. LiPo

Las baterías LiPo (Lithium Polymer) son una fuente de energía comúnmente utilizada en los drones y otros dispositivos electrónicos portátiles. Se caracterizan por tener una alta densidad de energía y de dimensiones y peso reducido, lo que significa que pueden proporcionar una gran cantidad de energía en un paquete pequeño y ligero.

En el caso específico del proyecto de dron que estamos desarrollando, utiliza una batería LiPo de la marca SUNPADOW 3s (ver Fig. 2.8), lo que significa que contiene tres celdas de batería en serie. Cada celda proporciona una tensión nominal de 3.7V, lo que significa que la batería completa tiene una tensión nominal de 11.1V. Esta batería tiene una capacidad de 2250 mAh, lo cual es una cantidad de energía suficiente para operar el dron.

La tasa máxima de descarga de la batería es de "60C", este valor hace referencia al factor máximo de corriente que podemos llegar a pedir a la batería, esto que significa que puede descargar energía a una tasa de hasta 60 veces su capacidad nominal (en este caso, 60 x 2.25 A = 135 A) sin sufrir daños significativos. Esto es realmente importante y es uno de los motivos por los cuales este tipo de baterías se utilizan tanto en el sector de los drones ya que la batería debe ser capaz de suministrar suficiente corriente a los motores para mantener el vuelo.

La batería es enchufada al dron utilizando el conector XT60. Este es un tipo de conector diseñado específicamente para baterías LiPo y son muy habituales junto con los conectores XT30, JST o XT90. Se caracteriza por ser fácil de conectar y desconectar, seguro y resistente a altas corrientes.

Además, es importante tener en cuenta ciertas precauciones cuando se opera con este tipo de baterías ya que una mala praxis puede inutilizar la batería o

producir lesiones al usuario. Una de las pautas a seguir es realizar las cargas con un cargador que tenga funciones de seguridad incorporadas como detección de voltaje máximo y capacidad de apagado automático y realizar una supervisión de la carga. También es conveniente realizar inspecciones regulares en busca de hinchazones, daños o ver si se calienta en exceso después de utilizarla en operación.

Fig. 2.8. Batería LiPo de 3 celdas.

2.1.8. PDB XT-60

La principal función de una PDB es recibir la energía proveniente de la batería del dron y distribuirla de manera adecuada a los motores, controladores de vuelo, sensores y otros dispositivos conectados. Esto permite un suministro de energía estable y confiable a cada componente, asegurando un funcionamiento óptimo del dron durante el vuelo.

La PDB cuenta con múltiples puntos de conexión para los cables de los motores y otros dispositivos, también incluye reguladores de voltaje y circuitos de protección. Estos reguladores aseguran que cada componente reciba la tensión adecuada para su correcto funcionamiento, evitando sobrecargas o fluctuaciones de voltaje que podrían dañar los dispositivos.

Además de la distribución de energía, la PDB también puede incluir características adicionales, como filtros para reducir el ruido eléctrico o la interferencia, y LED indicadores para mostrar el estado de alimentación del dron.

La Power Distribution Board utilizada en este proyecto es la XT-60 [14] de Matek Systems (ver Fig. 2.9). Esta PDB, está especialmente diseñada para su uso en drones y viene integrada con el conector XT-60, compatible con la batería seleccionada.

Fig. 2.9. PDB XT-60 de Matek Systems.

2.1.9. ESCs, motores y propellers.

Las ESCs (Electronic Speed Controllers) son dispositivos electrónicos que se utilizan para controlar la velocidad de los motores eléctricos de los drones. Las ESCs se conectan directamente a la PDB del dron y a los motores, y utilizan señales de control provenientes del controlador de vuelo para ajustar la velocidad de rotación y así controlar la altura, la velocidad y la dirección del dron. Las ESC, son esencialmente, conversores DC-AC regulables, estos se encargan de convertir la tensión continua de las baterías en tensión alterna, utilizando una señal que proviene de la controladora de vuelo. Esta señal recibida del controlador de vuelo afecta al ancho de pulso de la señal generada por las ESC, lo que regula la velocidad de los motores para realizar el control de vuelo.

En este proyecto se han escogido las Tmotor AIR20A, estas son ESCs de alta calidad diseñadas específicamente para vehículos multirrotores sin BEC (Battery Elimination Circuit). Estas ESCs son capaces de proporcionar una alta corriente de salida y están diseñadas para ser muy eficientes en términos de energía. Además, las AIR20A son compatibles con una gran variedad de controladores de vuelo y software de programación, lo que las hace muy versátiles y fáciles de integrar en cualquier proyecto de dron.

Los motores escogidos para este proyecto son los Tmotor AIR 2213. Estos motores eléctricos son ideales para cuadricópteros de 1200 g a 1500 g. Son motores de respuesta rápida y de poco ruido, son compatibles con el frame seleccionado y fáciles de instalar en conjunto con las ESC. Dos de ellos deben girar en sentido CW y los otros dos en sentido CCW. Estos motores son de 920 KV lo que significa que por cada voltio que apliquemos a los motores obtendremos 920 revoluciones por minuto, en combinación con la LiPo de 12,6 V de tensión máxima (4,2 V/celda · 3 celdas) los motores pueden llegar a girar a un ritmo de 11592 RPM.

Los propellers escogidos son los T9545, son motores de 9,5 pulgadas de diámetro y con un paso de 4,5 pulgadas. Estos propellers, vienen con una

cabecera de metal de autobloqueo la cual está hecha para asegurar que, durante el vuelo, no se aflojan los propellers. Esta cabecera de autobloqueo es simplemente el diseño de una rosca que se ajusta en la dirección contraria a la que giran los motores lo que prevé de la posibilidad de que la rosca se salga.
Los propellers escogidos son los T9545, que son propellers de 9,5 pulgadas de diámetro y tienen un paso de 4,5 pulgadas. Estos propellers vienen equipados con una cabecera de metal de autobloqueo, diseñada para garantizar que los propulsores no se aflojen durante el vuelo. La cabecera de autobloqueo consiste en una rosca que se ajusta en sentido contrario al giro de los motores, evitando así que la rosca se desenrosque.

Todos estos elementos se pueden obtener en forma de kit [15], como se puede ver en la Fig. 2.10, y son compatibles entre ellos además de serlo con los demás elementos del dron como el frame y la controladora de vuelo.

Fig. 2.10. T-Motor AIR GEAR 350.

El peso del cuadricóptero es de aproximadamente 1050 gramos, por lo que es crucial seleccionar motores y hélices que sean capaces de generar la fuerza de empuje necesaria para sostener y volar el dron. A continuación, en la Tabla 2.1, se presentan las especificaciones para esta combinación de motores y hélices:

Tabla 2.1. Especificaciones AIR 2213 KV920/T-9545.

Item No	Vots (V)	Prop	Throtte	Amps (A)	Watts (W)	Thrust (g)	RPM	Efficiency (g/W)
AIR 2213 KV920	11,1	T 9545	50%	2	22,2	240	4400	10,61
			65%	3,8	42,18	386	5900	9,15
			75%	5,5	61,05	490	6900	8,03
			85%	7,2	79,92	594	7800	7,43
			100%	9,8	108,78	722	8300	6,64
	12		50%	2,3	27,6	278	4800	10,00
			65%	4,4	52,8	445	6300	8,43
			75%	6,2	74,4	568	2200	7,63
			85%	8,1	97,2	679	8100	6,99
			100%	10,9	130,8	813	8900	6,22
	14,8		50%	13	45,54	403	5700	8,25
			65%	6,2	91,76	636	7600	6,93
			75%	8,4	124,32	786	8600	6,32
			85%	10,7	158,36	907	9500	5,73
			100%	14,3	211,64	1064	10200	5,12

Como se puede observar en la Tabla 2.1, el empuje (thrust) generado para un 65% de throttle a 11,1 V supera el peso total del dron, lo que indica que los motores y las hélices seleccionadas son más que adecuados para realizar el vuelo sin problemas. Esto garantiza una capacidad de empuje suficiente para mantener el dron en el aire y maniobrar de manera segura. El empuje máximo proporcionable por los motores es:

$$Thrust\ (11.1\ V, 100\%) = 722\,\frac{g}{motor} \cdot 4\ motor = 2888\ g \qquad \textbf{(2.1)}$$

Con un thrust-to-weight ratio de:

$$\frac{T}{W} = \frac{2888\ g}{1050\ g} = 2{,}75 \qquad \textbf{(2.2)}$$

Está relación empuje/peso es suficiente para garantizar una buena maniobrabilidad en exterior y una buena respuesta del dron.

Además, conociendo los consumos de los motores, se pueden calcular los tiempos de vuelo del dron, como el tiempo de vuelo máximo o el tiempo medio de vuelo, asumiendo un throttle para un vuelo convencional del 65%.

$$t_{max} = \frac{0{,}85 \cdot 2{,}25\ \frac{A}{h}}{9.8\ \frac{A}{motor} \cdot 4\ motor} = 0.048h \approx 3\ mins \quad \textbf{(2.3)}$$

$$t_{avg} = \frac{0{,}85 \cdot 2{,}25\ \frac{A}{h}}{3.8\ \frac{A}{motor} \cdot 4\ motor} = 0.126\ h \approx 7{,}5\ mins \quad \textbf{(2.4)}$$

También se puede calcular el tiempo de vuelo empleando un throttle de equilibrio. El valor del empuje de equilibrio tiene que ser igual al peso del dron, por lo que:

$$Thrust_{hover} = \frac{W}{4} = \frac{1050\ g}{4} = 262.5\ g \quad \textbf{(2.5)}$$

Aplicando una interpolación lineal, se puede hallar el consumo de corriente:

$$I_{hover} = 2.53\ A \quad \textbf{(2.6)}$$

Por lo que el tiempo de vuelo para el empuje de equilibrio es:

$$t_{hover} = \frac{0{,}85 \cdot 2{,}25\ \frac{A}{h}}{2{,}53\ \frac{A}{motor} \cdot 4\ motor} = 0.19\ h \approx 11{,}3\ minutos \quad \textbf{(2.7)}$$

Como se puede observar, los tiempos de vuelo obtenidos corresponden a los habituales de un dron de estas características.

2.2. Esquema eléctrico

El esquema eléctrico resultante después de realizar las conexiones de los diferentes componentes de hardware del sistema se puede hallar en la Fig. 2.11.

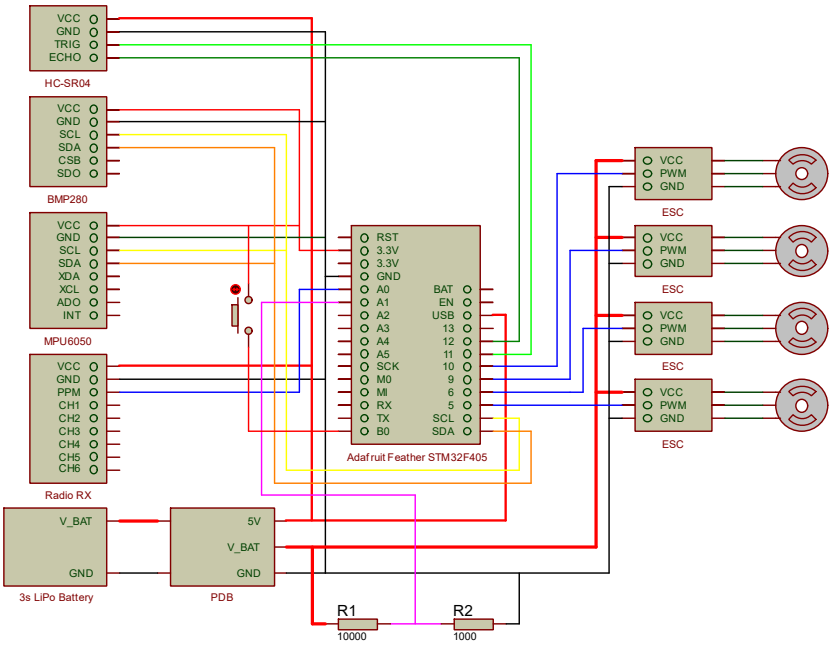

Fig. 2.11. Esquema eléctrico del dron.

Como se puede ver en el esquema, la PDB es alimentada por la batería y esta distribuye la energía al resto de los componentes. Existen tres fuentes de tensión que alimentan los componentes de hardware y se representan mediante cables de color rojo y diferentes grosores. Estas fuentes son: 3,3 V, 5 V y la tensión de la batería (9,6 V – 12,6 V).

Según el fabricante, la placa está hecha para ser alimentada principalmente empleando el conector USB o enchufando una LiPo 1s. Sin embargo, para alimentar la placa, se ha optado por una de las alternativas que presenta Adafruit la cual consiste en alimentar con 5 V el pin “USB”. De esta manera podemos alimentar el microcontrolador empleando directamente la PDB del dron sin añadir fuentes de alimentación externa. Es importante tener en cuenta que alimentar la placa de esta manera conlleva el riesgo de dañar el ordenador si lo conectamos al dron alimentado con la batería.

Para cargar el software al microcontrolador, es necesario poner el pin “B0” en “HIGH” durante el arranque. Al realizar esta operación, el microcontrolador entra en modo carga y admite la subida del programa. Para facilitar esta operación, se ha optado por implementar un pequeño botón el cual está directamente conectado a la fuente de 3.3 V y, si el usuario necesita cargar un código, simplemente presionando el botón cuando se enchufa el USB provocara el arranque del modo de carga.

CAPÍTULO 3. DISEÑO DE SOFTWARE

La parte de software en sistemas de control es crucial para el correcto funcionamiento de cualquier sistema, incluyendo drones. El algoritmo de control es la pieza central del software, ya que es el encargado de calcular la señal de control que se enviará a los actuadores para lograr que el dron actúe en la dirección deseada.

El algoritmo de control típicamente se compone de varias partes: generación de referencias, adquisición de datos, procesamiento de datos, cálculo de errores, cálculo de la señal de control, y envío de la señal de control a los actuadores. Cada una de estas partes es importante y debe estar diseñada para trabajar en conjunto de manera eficiente y precisa.

La generación de referencias se encarga de determinar la trayectoria deseada del dron, basándose en los objetivos del sistema. Esta trayectoria puede ser definida en términos de velocidad, aceleración, posición, orientación, entre otros. Es importante tener en cuenta que las referencias pueden cambiar constantemente en tiempo real, por lo que se requiere de un algoritmo que pueda actualizarlas de forma rápida y precisa.

La adquisición de datos es la parte del software que se encarga de leer los sensores, en el caso del dron, la MPU6050 y el BMP280. Estos sensores proporcionan información sobre la posición vertical, velocidad, orientación, aceleración, entre otros parámetros, que son necesarios para el cálculo de la señal de control. Es importante que los datos adquiridos sean precisos y se actualicen a una tasa adecuada para evitar errores en el cálculo del control.
El procesamiento de datos es la parte del software que se encarga de analizar los datos adquiridos y prepararlos para su uso para el cálculo del control. Esto puede incluir la conversión de unidades, filtrado de señales, eliminación de ruido, calibración de los sensores, entre otros.

El cálculo de errores es una parte fundamental del algoritmo de control. Esta parte se encarga de comparar las referencias con los datos adquiridos, y determinar la diferencia entre ellos. Esta diferencia se conoce como error, y es la base del cálculo de la señal de control. Es importante que el cálculo del error sea preciso y actualizado constantemente.

El cálculo de la señal de control es la parte del algoritmo que se encarga de determinar la señal que se enviará a los actuadores para lograr la trayectoria deseada. Este cálculo se realiza mediante algoritmos de control, como los controladores PID, que ajustan la señal de control en función del error calculado y de otros parámetros del sistema.

Finalmente, la señal de control es enviada a los actuadores, que son los encargados de mover el dron en la dirección deseada. En el caso del dron

mencionado, los actuadores son los motores que varían su velocidad en función de señal de control procesada por las ESCs.

El software de control es una parte fundamental para el correcto funcionamiento del sistema. Requiere de algoritmos de control precisos y eficientes, así como de una arquitectura adecuada para asegurar que todas las partes trabajen en conjunto de forma óptima.

3.1. Arquitectura y desarrollo del software

Es fundamental establecer una estructura bien definida para implementar el algoritmo de control del dron, asegurando que los cálculos internos se realicen de manera procedimental y ordenada. Esto resulta de gran utilidad a la hora de identificar posibles errores, fallos de implementación u otras situaciones similares, ya que facilita el proceso de diagnóstico gradual del código.

Desde la perspectiva del desarrollador, una arquitectura de software bien diseñada agiliza tanto la implementación inicial como la posterior modificación de funcionalidades en el código. En caso que otros desarrolladores muestren interés en el proyecto o deseen colaborar en él, una sólida arquitectura de software facilita considerablemente la comprensión del código y permite que desarrolladores externos se familiaricen rápidamente con él.

La arquitectura de software seguida para implementar el algoritmo de control se basa en una estructura modular. Trabajar con estructuras modulares presenta múltiples ventajas, ya que permite al desarrollador incorporar nuevas funcionalidades en el sistema de manera sencilla. Una estructura modular sigue el concepto de poder añadir y mover módulos con facilidad cuando sea necesario. En este caso, se realizará una llamada a los módulos principales, que consisten en funciones ubicadas en archivos separados. Estos módulos, a su vez, se dividen en subrutinas adicionales las cuales se llaman desde la ejecución interna del módulo principal.

Por ejemplo, existe un módulo que se encarga de recopilar toda la información de los sensores y realizar el procesado necesario de las señales para poder obtener información. El bucle principal llama a la función ubicada en el archivo correspondiente que a su vez llama a otras subrutinas ubicadas en este mismo archivo que se encargan de, por ejemplo, leer y procesar la IMU. Esto, como se puede ver, es realmente ventajoso a la hora de implementar nuevos sensores u otras funcionalidades ya que solo hace falta añadir las variables globales en la cabecera del archivo principal y añadir la función que se desee, sin afectar al resto del código.

Esta arquitectura no solo proporciona ventajas al implementar nuevas funcionalidades, sino que también facilita la comprensión del funcionamiento del

código en general y la segmentación de las operaciones realizadas en el algoritmo de control. En la Fig. 3.1 se presenta la arquitectura diseñada.

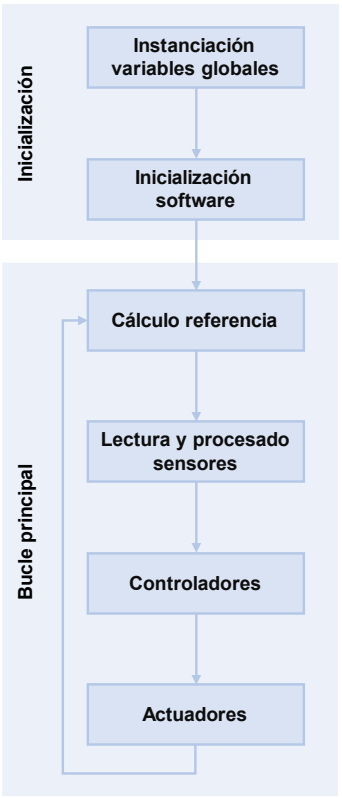

Fig. 3.1. Arquitectura del software.

A continuación, se detalla de manera procedimental y general, la estructura del código:

3.1.1. Tareas de inicialización

3.1.1.1. Declaración de variables globales utilizadas por el código.

En general, con proyectos basados en Arduino C, para este tipo de proyectos donde cada método requiere de bastantes entradas y salidas diferentes y se opta por una estructura modular, lo más sencillo es trabajar con variables globales donde se tiene acceso a las variables desde cualquier punto del código. Adicionalmente, el hecho de tener variables globales facilita la implementación de los módulos de código ya que estas son accesibles desde cualquier método. Por lo tanto, se ha decidido usar este tipo de instanciación para las variables del código.

3.1.1.2. Inicialización del software.

Lo primero que se hace al ejecutar cualquier software basado en Arduino es ejecutar la función `setup`. Esta función, principalmente, se encarga de todas las tareas que hay que ejecutar una sola vez al inicio. Aquí encontraríamos tareas como el calibrado de la IMU, la detección de la radio, la vinculación de interrupciones de hardware o la inicialización de conexión serial para realizar tareas de depuración.

Dentro de la función `setup`, hallamos la llamada a la subrutina `init_components`. Esta función se encarga de ejecutar los diferentes módulos necesarios para la inicialización de los componentes del dron como se muestra en la Fig. 3.2. Dentro de ella encontramos llamadas, de forma secuencial, a las siguientes funciones:

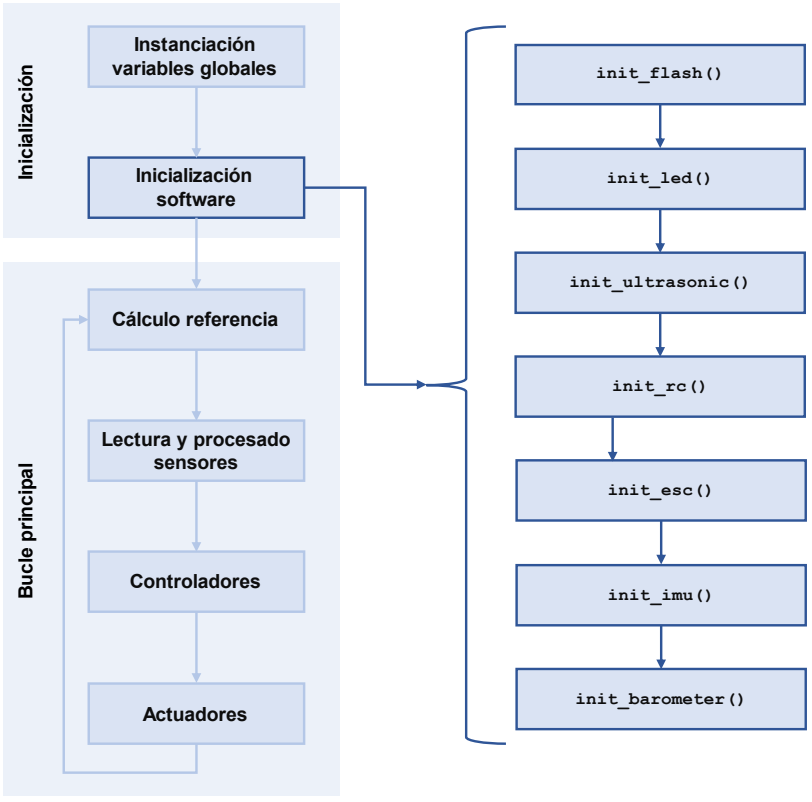

Fig. 3.2. Estructura procedimental de la inicialización.

`init_flash()`: Esta función, esencialmente, se encarga de iniciar la comunicación con la unidad SPI Flash, empleando una librería desarrollada por Adafruit y disponible en Github [16]. Este método no tiene ningún impacto directo sobre la funcionalidad del algoritmo de control y, durante el desarrollo del proyecto, se empleó para realizar tareas de diagnóstico / almacenaje de datos.

`init_led()`: Prepara el pin conectado al LED para poder controlarlo con el microcontrolador.

`init_ultrasonic()`: Inicializa el pin correspondiente a los ecos del sensor de ultrasonidos y vincula el método para asignar el comportamiento cuando se detecte una interrupción.

`init_rc()`: Este método se encarga de inicializar la conexión con el dispositivo receptor de la radio. Configura el pin de entrada de la señal PPM de la radio y le adjunta una interrupción de hardware que llama al método `read_PPM`, encargado de leer y procesar la señal de la radio.

`init_esc()`: Este método inicializa las señales PWM que reciben las ESC para realizar el control de velocidad de rotación de los motores. Para controlar adecuadamente los timers del microcontrolador, se utiliza la librería HardwareTimer de stm32duino, disponible en Github [17]. Es necesario el uso de los timers del microcontrolador ya que esto permite ahorrar el tiempo que supone generar las señales PWM de manera manual.

`init_imu()`: Este método, se encarga de realizar el calibrado necesario para preparar la MPU6050 para su uso. Para hacerlo, utiliza la librería Wire [18] para realizar la conexión I^2C con el chip. Las primeras líneas del método se encargan de iniciar la comunicación entre el microcontrolador y el dispositivo utilizando una dirección de 8 bits que identifica al dispositivo.

Una vez la conexión se ha demostrado que es satisfactoria, se configuran los valores de cuatro registros diferentes de 8 bits del sensor. Esta acción permite configurar ciertos parámetros.

Como se puede ver en la Tabla 3.1, el Registro 107 (0x6B) [19]: Nos permite activar y configurar el modo de operación del sensor.

Tabla 3.1. Registro 0x6B.

Registo (Hex)	Registo (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

En la Tabla 3.2 se puede ver la configuración asignada del registro:

Tabla 3.2. Asignación del registro 0x6B.

Bit	Descripción	Valor
7	Reinicia el dispositivo	0
6	Modo de suspensión de bajo consumo	0
5	Modo de ciclo desactivado	0
4	-	0
3	Termómetro deshabilitado	0
2	Oscilador de 8 MHz seleccionado.	0
1		0
0		0

Escribimos el registro mediante la librería wire:

```
Wire.beginTransmission(gyro_address);
Wire.write(0x6B);
Wire.write(0x00);
Wire.endTransmission();
```

Y repetimos el mismo proceso para el resto de los registros.

Una vez se ha configurado adecuadamente el sensor, es hora de calibrarlo. Para hacerlo, se obtienen 2000 lecturas del sensor y se hace un promedio para, posteriormente, extraer estos valores de las futuras lecturas. Esto se hace para fijar como valores de referencia los valores que nos provee el acelerómetro al principio de la ejecución del código.

`init_barometer()`: Este método, se encarga de realizar el calibrado necesario para preparar el barómetro BMP280 para su uso. Para hacerlo, utiliza la librería Wire para realizar la conexión I^2C con el chip, junto con la MPU6050. Las primeras líneas del método se encargan de iniciar la comunicación entre el microcontrolador y el dispositivo utilizando una dirección de 8 bits que identifica al dispositivo.

Una vez la conexión se ha demostrado que es satisfactoria, se configuran los valores de cuatro registros diferentes de 8 bits del sensor. Esta acción permite hacer lecturas de los parámetros de compensación y configurar el sensor.

Como se ha mencionado anteriormente, el sensor es tremendamente sensible a los cambios de temperatura y es por eso por lo que es necesario hacer correcciones para poder utilizar las lecturas como referencia para el algoritmo de control. Para realizar las debidas correcciones se emplean unos parámetros de compensación que vienen integrados en los registros de lectura del microcontrolador.

Estos parámetros de compensación son 12 valores de 16 bits, 3 se emplean para correcciones de temperatura y 9 para correcciones de presión. El datasheet [20] del BMP280 informa que estos valores se encuentran a partir del registro 136

(0x88) en adelante por lo que se hace una solicitud de 24 bytes y se almacenan en las variables para luego utilizarlos para aplicar las correcciones.

```
Wire.beginTransmission(BMP280_ADDRESS);
Wire.write(0x88);
Wire.endTransmission();
Wire.requestFrom(BMP280_ADDRESS, 24);
dig_T1 = Wire.read() | Wire.read() << 8;
dig_T2 = Wire.read() | Wire.read() << 8;
dig_T3 = Wire.read() | Wire.read() << 8;
dig_P1 = Wire.read() | Wire.read() << 8;
dig_P2 = Wire.read() | Wire.read() << 8;
dig_P3 = Wire.read() | Wire.read() << 8;
dig_P4 = Wire.read() | Wire.read() << 8;
dig_P5 = Wire.read() | Wire.read() << 8;
dig_P6 = Wire.read() | Wire.read() << 8;
dig_P7 = Wire.read() | Wire.read() << 8;
dig_P8 = Wire.read() | Wire.read() << 8;
dig_P9 = Wire.read() | Wire.read() << 8;
```

Finalmente, ajustamos los registros de configuración del sensor para prepararlo para el tipo de operación a realizar. Esta configuración se lleva a cabo de una manera muy similar a la que se hace con la IMU. Los registros modificados son el 244 y el 245, los cuales permiten ajustar los siguientes parámetros:

- Temperature oversampling: x2.
- Pressure oversampling: x2.
- Operational mode: Normal mode.
- Normal mode standby time duration: 0,5 ms.
- IIR filter time constant: 16.

3.1.2. Bucle principal

En el bucle principal encontraremos la ejecución reiterativa de las funciones principales que segmentan el código en sus respectivos módulos. En estas funciones se encontrarán las tareas de cálculo de referencias, obtención de medidas de los sensores, cálculo de la señal de error y posterior cálculo de los PID y finalmente la adaptación de la señal para los actuadores del cuadricóptero. Adicionalmente también hay un módulo encargado de realizar tareas de diagnóstico para depurar y un sistema de control de tiempo de ejecución de bucle.

3.1.2.1. Sistema de duración de control del bucle

Aunque la ejecución del bucle principal del código siempre involucre la repetición de los mismos módulos de código, no siempre se ejecuta con la misma rapidez. Esto puede deberse a situaciones específicas en las que se realizan cálculos más complejos o se requiere un tiempo adicional para acceder a los registros de alguno de los sensores.

No obstante, es crucial que los algoritmos de control se ejecuten a una frecuencia constante para garantizar la estabilidad del dron. Para abordar esta necesidad, al final del bucle principal se incluye un fragmento de código encargado de mantener la duración del bucle de manera uniforme.

Este fragmento de código se implementa con el propósito de ajustar el tiempo de espera o realizar acciones adicionales para compensar cualquier variación en la duración del bucle principal. Su objetivo es mantener una frecuencia de ejecución constante, independientemente de las circunstancias que puedan afectar la velocidad de ejecución del código.

Esta práctica asegura que el algoritmo de control se ejecute de manera consistente y predecible, lo que resulta fundamental para lograr una operación estable y segura del dron. En la Fig. 3.3 se puede ver la estrategia seguida para mantener una frecuencia de ejecución constante.

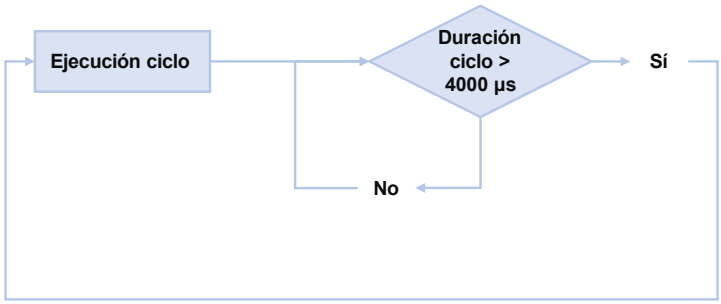

Fig. 3.3. Estrategia para harmonizar los tiempos de ejecución del ciclo.

3.1.2.2. Cálculo de referencia.

En este primer módulo, se encuentra toda la información relacionada con las referencias que se deben utilizar para llevar a cabo la ejecución adecuada del algoritmo de control. Esto incluye la definición de los modos de vuelo del cuadricóptero y la lógica necesaria para realizar las transiciones entre ellos.

El método que se encarga de realizar estas funciones es el denominado `reference_computation`. La idea general de este módulo es segmentar la generación de las referencias en función del modo de vuelo que se opere y que, en el resto de ejecución del algoritmo de control, no aparezcan prácticamente distinciones relacionadas con el modo de vuelo. Hacer esto permite que sea más

fácil añadir modos de vuelo a posteriori como por ejemplo modos de vuelo basados en GPS como “Loiter” o “RTH”.

Las subrutinas anidades en el módulo encargado de la generación de la referencia se pueden observar en la Fig. 3.4.

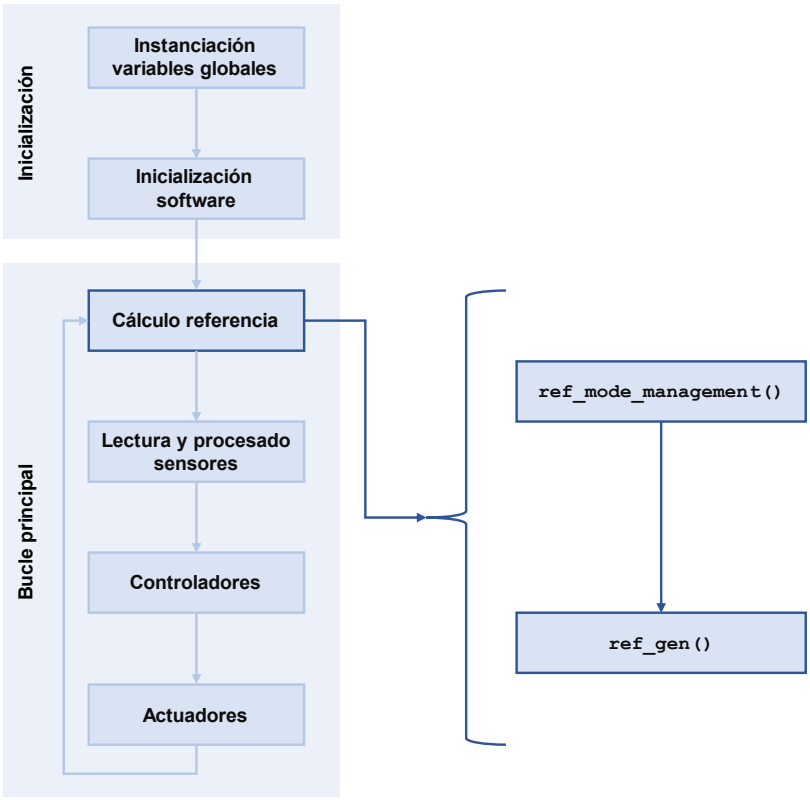

Fig. 3.4. Estructura procedimental de la generación de las referencias.

`ref_mode_management()`: Este primer método es el encargado de contener toda la lógica para discernir cual es el modo de vuelo en el que opera. La estructura de este método es bastante sencilla y, con unas pocas condiciones, permite implementar las transiciones necesarias entre los modos de vuelo del cuadricóptero.

El método en si trabaja con una enumeración de Arduino, la cual es muy beneficiosa para establecer asignaciones ordenadas y para atribuir etiquetas donde sea necesario.

`ref_gen()`: Este método, se encarga de generar la referencia en función del valor del modo de vuelo establecido en el anterior método.

El método consiste en generar las referencias y contener la parametrización de vinculada a cada uno de los modos de vuelo. A continuación, en la Tabla 3.3, se pueden ver los diferentes comportamientos en función del modo de vuelo.

Tabla 3.3. Definición de los modos de vuelo.

Flight Mode	Descripción
Disabled	El dron permanece quieto, sin capacidad para actuar los motores. Este modo de vuelo es en el que siempre se debe trabajar físicamente con el cuadricóptero alimentado ya que permite una manipulación segura y evita posibles lesiones. Por defecto este es el modo de vuelo por defecto al enchufar la batería y realizar los calibrados.
Mounting	Modo de transición entre “Disabled” y los modos de vuelo de operación del dron.
Stable	Modo de vuelo que permite volar el dron en modo estabilizado, donde el usuario controla la altura con el throttle, y el roll pitch y yaw.
Altitude Hold	Modo de vuelo que permite volar el dron en modo de altitud constante, donde la altura se regula automáticamente y el usuario controla roll pitch y yaw.

3.1.2.3. Lectura de los sensores y proceso de la señal.

Este módulo se encarga de leer las señales provenientes de los sensores y de los elementos de hardware externos para llevar a cabo el procesamiento necesario. Su función principal consiste en establecer las conexiones adecuadas con los sensores y obtener las lecturas necesarias para adaptar las señales recibidas a la ejecución del algoritmo de control.

La función `read_process_units` es la que tiene el propósito de realizar esta tarea. Esta función es esencial en la arquitectura de software del sistema de control de vuelo, ya que es el origen de todas las señales provenientes de los sensores en las cuales se basa el algoritmo de control. Por lo tanto, es fundamental que esta función sea desarrollada cuidadosamente y de forma robusta para asegurar que todas las señales de entrada sean adquiridas y procesadas de forma correcta.

Este método, de la misma manera que los anteriores, está compuesto por la ejecución de subrutinas anidadas visibles en la Fig. 3.5.

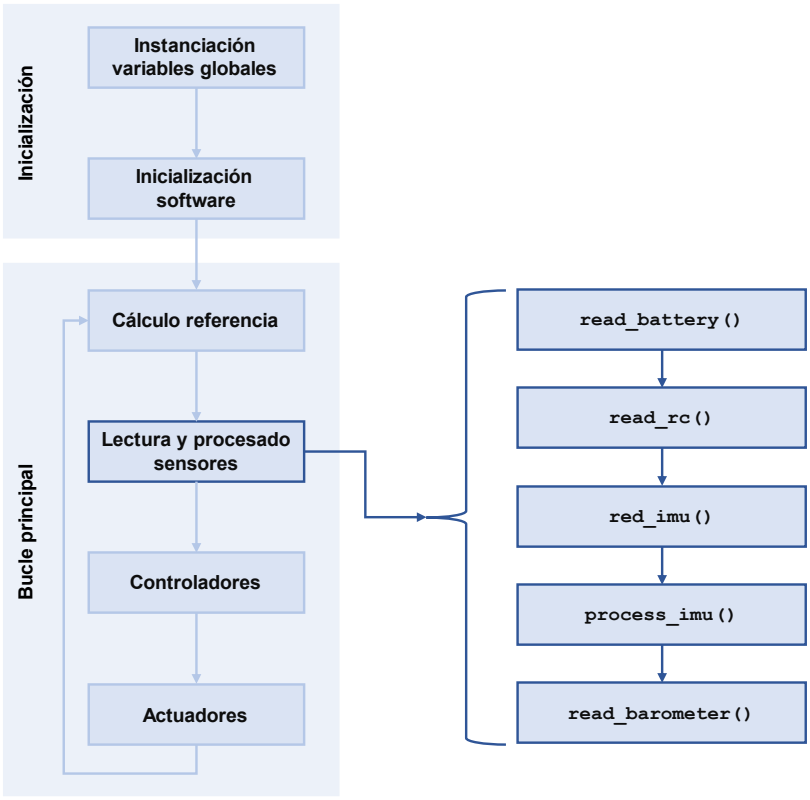

Fig. 3.5. Estructura procedimental de la lectura y procesado de los sensores.

`read_battery()`: Se encarga de medir la tensión de la batería. Es importante tener en cuenta que la batería utilizada en el dron es de 11,1V nominales y los pines del microcontrolador operan entre 3.3-5 V, lo que significa que, si intentáramos leer directamente la tensión de la batería con el microcontrolador, se produciría un cortocircuito y posiblemente resultaría en daños permanentes del microcontrolador.

Para evitar esto, se utiliza un divisor de tensión de dos resistencias en serie para escalar la tensión de la batería a un nivel seguro y legible por el microcontrolador. De esta manera, se puede medir la tensión de la batería con precisión y utilizar esta información para ajustar la potencia del dron y garantizar que no se agote la batería durante el vuelo.

La función `read_battery` se encarga de leer los valores de voltaje a través del divisor de tensión y realizar los cálculos necesarios para convertir esos valores en una lectura de tensión precisa de la batería. Conocer en todo momento la tensión de la batería es importante para diferentes aspectos, como asegurarse que no agotamos la batería completamente o regular adecuadamente la potencia de los motores.

Para procesar adecuadamente la señal de tensión de la batería, debemos entender cómo afecta el divisor de tensión a las lecturas desde el

microcontrolador. Implementando el divisor de tensión, sabemos que el voltaje que habrá en el borne del pin sigue la siguiente expresión:

$$V_{PIN} = V_{BAT} \cdot \frac{R_1}{R_1 + R_2} \quad \textbf{(3.1)}$$

La tensión que lee el PIN es un valor de 12 bits donde 0 equivale a 0 V y 4095 equivale a 3,3V. Por lo tanto, el valor de tensión convertido por el ADC del microcontrolador corresponderá a la siguiente expresión:

$$V_{AR} = \frac{4095}{3{,}3\ V} \cdot V_{PIN} \quad \textbf{(3.2)}$$

Una vez disponemos de este valor de tensión digitalizado, podemos recuperar la tensión de la batería combinando ambas expresiones:

$$V_{AR} = V_{BAT} \cdot \frac{4095}{3{,}3\ V} \cdot \frac{R_1}{R_1 + R_2} \rightarrow V_{BAT} = V_{AR} \cdot \frac{3{,}3V}{4095} \cdot \frac{R_1 + R_2}{R_1} \quad \textbf{(3.3)}$$

Sustituyendo $R_1 = 10000\ \Omega$ y $R_2 = 1000\ \Omega$:

$$V_{BAT} = \frac{V_{AR}}{112{,}8} \quad \textbf{(3.4)}$$

Adicionalmente, se aplica un filtro complementario para reducir contribuciones aleatorias generadas por el ruido.

$$V_{BAT}(z) = 0{,}92 \cdot V_{BAT}(z-1) + 0{,}08 \cdot V_{BAT}(z) \quad \textbf{(3.5)}$$

`read_ppm()`: Este método se encarga de demodular la señal recibida por la radio. Esta emplea la modulación por posición de pulso (PPM) donde la señal consiste en un tren de pulsos ubicados en instantes de tiempo variables los cuales permiten, a partir de medidas temporales, obtener el ancho de los canales de la radio. Este método se implementa mediante una interrupción de hardware desde la inicialización por lo que no contiene una llamada explicita desde `read_process_units`.

Para demodular correctamente esta señal, es necesario establecer un sincronismo entre microcontrolador y receptor RX. Para hacerlo, se emplea el denominado pulso de sincronismo, el cual se identifica por tener una duración destacablemente mayor al resto. Es por eso por lo que cada vez que el microcontrolador detecta la presencia de un pulso con una duración mayor, se reinicia la lectura de los pulsos:

```
if (micros() - pulse_instant[flank_count - 1] > 2500) flank_count = 0;
```

Estos instantes de tiempo se almacenan en un vector y se van actualizando a medida que el receptor RX envía nuevas señales al microcontrolador.

`read_rc()`: Este método se encarga de procesar los datos obtenidos por la función anterior. Se encarga de restar los valores almacenados en el vector para obtener las posiciones de los pulsos PPM las cuales oscilan entre 600 µs y 1600 µs. Posteriormente se realiza un mapeo de estos valores a una escala que oscile entre 1000 µs y 2000 µs, este mapeo no es obligatorio ni imprescindible pero facilita la comprensión y es más fácil operar con estos valores.

`read_imu()`: Este método, se encarga de realizar las lecturas de la MPU6050. El procedimiento es bastante similar que el empleado en la inicialización del sensor, mediante la librería Wire y el protocolo I2C, realizaremos conexiones de lectura y escritura a los diferentes registros de interés para obtener las medidas del sensor.

Los registros de interés son los que nos proporcionan las aceleraciones y velocidades angulares en los tres ejes. A continuación, en la Tabla 3.4, se puede ver el fragmento del mapa de registros de la MPU6050 que contiene los valores correspondientes al acelerómetro y giróscopo.

Tabla 3.4. Registros de aceleración, temperatura y velocidad angular.

Addr (Hex)	Addr (Dec)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT[7:0]							

Se puede apreciar que los valores de aceleración, temperatura y velocidad angular son de 16 bits y se proporcionan de manera consecutiva por lo que

podemos solicitarlos todos directamente para ahorrar tiempo en la consulta y almacenarlos en las variables correspondientes.

`process_imu()`: Este método se encarga de realizar el procesado de los datos obtenidos en la subrutina anterior, esto se encarga de transformar los bits en lecturas útiles de aceleración y velocidad angular.

Lo primero que se computa es la conversión de los valores de velocidad angular de 16 bits a º/s y a la aplicación de un filtro complementario que permite reducir posibles señales de ruido. Estas señales corresponden a las medidas que debemos controlar mediante el uso del algoritmo de control.

Posteriormente, se calculan las contribuciones de la velocidad angular al cambio en la orientación del dron, asumiendo comportamientos lineales para cada iteración del bucle.

`read_barometer()`: Este método, se encarga de realizar las lecturas del BMP280. La metodología seguida para obtener las medidas de este sensor a partir de los registros es la misma que la IMU, mediante el protocolo I2C y la librería Wire. Esta función provee de lecturas de presión al algoritmo de control las cuales serán usadas posteriormente para realizar el control de altitud mediante el modo de vuelo “Altitude Hold”.

Durante el desarrollo y la implementación de la función aparecieron una serie de problemáticas que imposibilitaban la lectura adecuada de los datos y la correcta ejecución del bucle principal. El primer problema que apareció fue que la frecuencia de actualización de registros del barómetro era inferior a la de la lectura de los mismos registros, lo cual conllevaba que la señal presión y temperatura obtenidas por el controlador de vuelo tenía valores repetidos lo que resultaba en una pérdida de rendimiento en la ejecución.

Para solucionar este problema, la solución a implementar fue realizar las solicitudes con una frecuencia similar a la frecuencia de actualización de registros del sensor lo cual eliminaba los valores repetidos de las señales por lo que la frecuencia de solicitud y lectura se redujo a una vez cada cuatro ciclos. A continuación, en la Fig. 3.6, se puede observar la estrategia implementada.

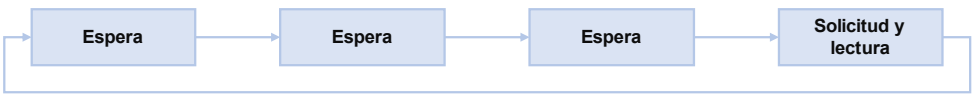

Fig. 3.6. Estrategia de solicitud y lectura cada 4 ciclos.

Otro problema que apareció fue que los ciclos del bucle principal en los cuales se hacia la consulta y lectura de los registros del sensor eran demasiado lentos.

Es decir, el solicitar los registros y proceder a su lectura conllevaba que el bucle principal del código superase el tiempo máximo de ejecución por ciclo. Es de vital importancia que la frecuencia de ejecución de los controladores sea constante ya que puede derivar en problemas graves de estabilidad del dron.

Para solucionar este problema, se decidió cambiar la manera en la que se hacia la consulta y la lectura al barómetro. Se optó por, en vez de realizar la lectura y consulta del barómetro una vez cada cuatro ciclos, separar consulta y lectura en ciclos diferentes. En la Fig. 3.7 se muestra la implementación definitiva.

Fig. 3.7. Estrategia de solicitud y lectura separadas, cada 4 ciclos.

Como se ha mencionado anteriormente en la inicialización, el BMP280 es un sensor muy sensible a cambios de temperatura, tanto es que mover el sensor de presión de la sombra al sol, ya corresponde a un error en la lectura de presión equivalente a varios metros de altura. Es por ello por lo que es necesario realizar las correcciones necesarias para evitar este tipo de alteraciones en la señal a controlar. Dichas correcciones emplean los registros cargados durante la inicialización y se aplican directamente con unas funciones de código que provee el fabricante, las cuales han sido transcritas a Arduino C para su uso para este proyecto.

La señal resultante de presión la cual provee el sensor es una señal prácticamente inmune a cambios de temperatura razonables para el tipo de operación en cuestión, pero es una señal irregular y difícilmente útil para emplearla en un algoritmo de control. Para solucionar este inconveniente se han aplicado una serie de filtros los cuales ajustan la señal para un control más adecuado.

El primer filtro que se le aplica es un filtro complementario, los filtros complementarios son útiles para eliminar ruido sin embargo se ven penalizados cuando se requiere de una respuesta rápida del dron, aplicar filtros con coeficientes mayores incrementa la lentitud en la respuesta del dron.

Para solucionar este problema de cambios grandes de presión en un intervalo de tiempo pequeño, se ha optado por añadir un sistema que detecta dichos cambios para darle prioridad a la señal y pasar el filtro complementario a un segundo plano.

Con esto se obtiene un sistema que mitiga los ruidos de manera eficaz pero que también es capaz de reaccionar de manera rápida cuando existen cambios de presión repentinos.

`read_ultrasonic():` Este método, lanzado por interrupción de hardware, se encarga de procesar los pulsos recibidos en los "echo" del sensor de ultrasonidos. El funcionamiento del método consiste en registrar el tiempo en el que se detecta una interrupción de LOW a HIGH como tiempo de inicio del pulso, y registrar como tiempo de fin de pulso cuando pasa de HIGH a LOW.

Una vez se obtiene el ancho de pulso recibido por el sensor, se puede calcular la distancia calculando el tiempo de propagación de la onda en el medio, para hacerlo se asumen velocidades de transmisión del sonido para condiciones de atmosfera estándar.

$$\Delta x = \frac{v_{sonido} \cdot t}{2} \tag{3.6}$$

Con esta expresión, se puede obtener la altura en la cual se halla el dron. Una vez se obtiene este valor, se realiza un filtrado de rangos de interés. Esto es porque los rangos de uso del sensor de ultrasonido son bastante limitados y, señales que excedan estos rangos definidos son probablemente valores erróneos o que no son de utilidad.

$$\Delta x_{filt} \in [10, 200]\ cm \tag{3.7}$$

A este valor, se le aplica filtro complementario para mitigar las contribuciones generadas por el ruido.

Uno de los factores que se observó durante la implementación del sensor de ultrasonidos fue que, a veces, existían lecturas de pulsos que ya habían sido recibidos por el sensor. Esto, debido al relieve del medio, se producía porque había pulsos que rebotaban contra otras superficies y encontraban caminos alternativos a la dirección vertical del dron. Para corregir esto, se implementa un sistema que se encarga de discriminar los pulsos que no correspondan al primero.

Estas lecturas del sensor de ultrasonidos se empleaban para realizar un vuelo de control de altitud a baja altura, correspondiente a una primera versión del "Altitude Hold". Sin embargo, con la implementación del barómetro, esto fue discontinuado.

Durante el desarrollo del método, se vio conveniente obtener la señal velocidad a partir de la derivada de la señal posición, asumiendo cambios lineales en cada ciclo de ejecución del bucle. Esto se hizo para estudiar la posibilidad de implementar un algoritmo de control que permita hacer ascensos y descensos del dron a una velocidad controlada.

$$v = \frac{dx}{dt} \approx \frac{\Delta x(z) - \Delta x(z-1)}{t(z) - t(z-1)} \quad \textbf{(3.8)}$$

Aunque, esta idea de control vertical de la velocidad empleando el sensor de ultrasonidos fue posteriormente discontinuada, se ha dejado el cálculo de la señal velocidad por si algún desarrollador está interesado en aprovecharla.

Finalmente, el sensor de ultrasonidos se ha empleado como takeoff detector, este takeoff detector ha sido empleado para desactivar la parte integral del controlador cuando el dron no ha despegado, esto evita que el PID funcione en lazo abierto y el integrador no se vuelva inestable en el proceso de despegue. Esto sucede porque los integradores operados en lazo abierto tienden a acumular error cada ciclo y, al no tener la potencia necesaria para volar y estabilizarse, el error tiende a acumularse infinitamente. Si bien existen alternativas para que esto no suceda como limitar la salida del integrador, da mejor sensación al usuario emplear este takeoff detector, ya que la contribución del integrador es completamente nula durante el despegue.

3.1.2.4. Cálculo del error de control y PIDs.

Este módulo se encarga de comparar las señales correspondientes a la referencia y la medida, calcular el error de control y aplicar los algoritmos PID. Es el núcleo del algoritmo de control, ya que determina cómo responder a los errores en las magnitudes que deseamos controlar.

Estas tareas se realizan dentro de la subrutina principal denominada `controllers`. Aquí se encuentran todos los métodos que se encargan de aplicar los PID al algoritmo de control. Este módulo, se encarga de recuperar los datos recibidos de `reference_computation` y `read_units` para obtener las señales de referencia y las señales de los sensores y realizar los cálculos del error de control.

Todos los controladores que se han empleado en este proyecto son de tipo PID (Proportional-Integral-Derivative). Un PID es una función de transferencia empleada para controlar un sistema de lazo cerrado y es el controlador más popular para este tipo de aplicaciones.

El término "proporcional" se refiere a que la acción de control es proporcional al error entre el valor deseado y el valor medido del proceso. El componente proporcional responde de manera directamente proporcional al error, lo que significa que cuanto mayor sea el error, mayor será la acción correctiva.

El término "integral" se refiere a la acumulación del error a lo largo del tiempo. Este componente toma en cuenta la suma acumulada de los errores pasados y corrige el control en función de esta integral. Ayuda a eliminar el error acumulado y a reducir el error en estado estacionario.

El término "derivativo" se refiere a la tasa de cambio del error. Este componente ayuda a predecir cómo cambiará el error en el futuro y permite realizar correcciones anticipadas. Ayuda a reducir la velocidad de respuesta y en la estabilidad del sistema.

La ejecución de este módulo se ha dividido como se muestra en la Fig. 3.8.

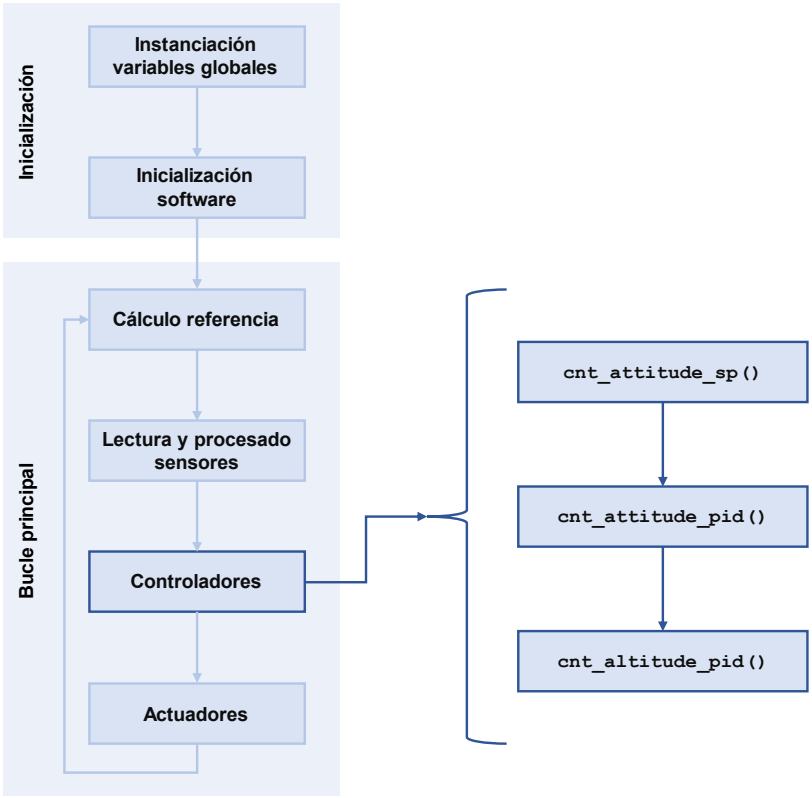

Fig. 3.8. Estructura procedimental de la ejecución de los controladores.

`cnt_attitude_sp()`: Este método se encarga de convertir las lecturas proporcionadas por la radio en referencias a seguir. Lo primero que hace es establecer una banda muerta para las lecturas de los sticks de la radio, esto es útil porque así evitamos derivas producidas por fallos en la calibración de la radio o por ruido en la señal recibida. Posteriormente, se centran en cero los valores

de manera que oscilan entre ±500 µs. Finalmente se obtiene el cálculo del error de control restando la señal referencia de la radio de la medida que entrega la IMU. El mismo proceso se repite con roll, pitch y yaw.

`cnt_attitude_pid()`: Esta función se encarga de realizar la ejecución de los PID para roll, pitch y yaw. Con la ejecución de esta subrutina se obtiene la salida del PID generada por la parte proporcional, integral y derivativa.

El algoritmo del PID se ejecuta convencionalmente, calculando la señal de error, almacenando en memoria el resultado del integrador, acoplando las tres componentes, aplicando limitaciones de seguridad y finalmente, almacenando el valor del error del ciclo para compararlo con el del siguiente ciclo para aplicar la parte derivativa.

`cnt_altitude_pid()`: Esta subrutina se encarga de la ejecución del PID que se encarga de aplicar el control por altitud del dron. La frecuencia de este controlador es cuatro veces más baja ya que el periodo de ejecución va en fase con la disponibilidad de las lecturas del barómetro por lo que la frecuencia de ejecución es de 50 Hz en vez de 200 Hz.

La estructura de este controlador es prácticamente igual a la de roll, pitch y yaw, con algunas pequeñas modificaciones. La modificación más relevante es que se le añade a la salida del PID un término que modifica la parte proporcional del controlador de manera lineal cuando el error en la lectura es grande. Esto se hace para disponer de una parte proporcional suave cuando el dron está bastante cerca del valor deseado y aumentarla de manera lineal cuando el error sea mayor y se requiera de una respuesta más contundente.

3.1.2.5. Generación de la señal para los actuadores.

En este último módulo del algoritmo de control se integran las salidas de los PIDs para lograr el control deseado. En esta etapa, se aplica la lógica que determina qué señales deben utilizarse en función del modo de vuelo del dron, y se generan las señales correspondientes para los actuadores.

El método principal que se encarga de realizar todas estas tareas es el denominado “actuators” y se encarga de llamar a todas las subrutinas relacionadas con la generación de la señal derivada para los actuadores, como se muestra en la Fig. 3.9.

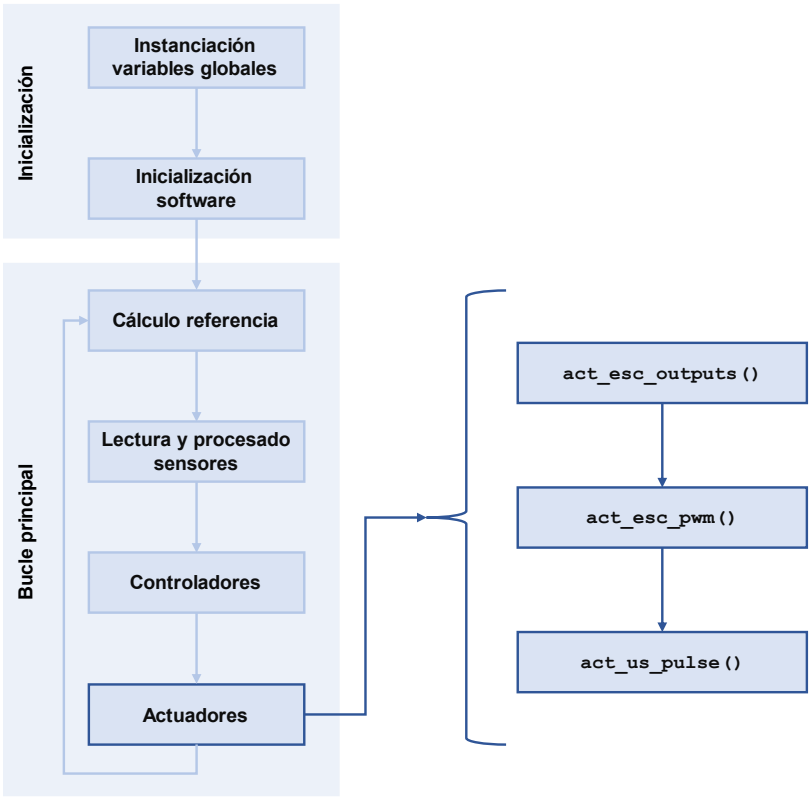

Fig. 3.9. Estructura procedimental de la generación de la actuación.

`act_esc_outputs()`: Este método se encarga, principalmente, de acoplar las salidas de los PID para posteriormente procesarlos para su envío a las ESCs.

Es importante saber que las señales de salida deben variar entre 1000 µs y 2000 µs, esto corresponde a el ancho de pulso de la señal PWM generada para las ESCs donde enviar un ancho de pulso de 1000 µs significa a tener los motores apagados y 2000 µs significa tener los motores operando a máxima potencia.

Las salidas de los PID que deben ser acopladas dependen del modo de vuelo en el que opere el cuadricóptero. Es por ello por lo que la función es esencialmente una condición que trabaja en función del modo de vuelo de la aeronave.

Para el modo de vuelo “Stable” el acople de las salidas de los PID es el siguiente:

$$ESC_1 = throttle - PID_\phi - PID_\theta - PID_\varphi \quad \textbf{(3.9)}$$
$$ESC_2 = throttle + PID_\phi - PID_\theta + PID_\varphi \quad \textbf{(3.10)}$$
$$ESC_3 = throttle + PID_\phi + PID_\theta - PID_\varphi \quad \textbf{(3.11)}$$
$$ESC_4 = throttle - PID_\phi + PID_\theta + PID_\varphi \quad \textbf{(3.12)}$$

Se ha seguido una estrategia para implementar el modo de “Altitude Hold” que consiste en encontrar un ajuste para el throttle a través de un controlador PID, con el objetivo de regular la altura del cuadricóptero. La premisa inicial es la siguiente:

$$throttle = throttle_{hover} + PID_{hover} \quad \textbf{(3.13)}$$

De esta manera, se puede controlar la altura del cuadricóptero empleando el PID de altitud y añadiendo la salida de este al throttle de equilibrio.

Sin embargo, uno de los problemas de esta solución es que la velocidad de los motores depende de la tensión que la batería puede proporcionar y este valor no se mantiene constante por lo que, el hover throttle que te permite lograr un equilibrio con la batería a 12,6 V no será el mismo que cuando la batería esté a 11,1 V.

$$throttle = throttle_{hover}(V) + PID_{hover} \quad \textbf{(3.14)}$$

Para abordar esta situación, se decidió llevar a cabo un vuelo completo en modo “Stable”, agotando por completo la batería. Durante este vuelo, se fue registrando el valor de la tensión de la batería y el valor del throttle cada vez que se detectaba un cambio de signo de la aceleración en el eje Z. Esto correspondía al thottle que generaba, aproximadamente, una fuerza nula en el dron.

$$a_z \approx 0 \quad \textbf{(3.15)}$$

Después del vuelo en modo “Stable”, se dispuso de una gran cantidad de lecturas las cuales fueron representadas gráficamente y se observó un comportamiento claramente lineal, es decir, los motores perdían potencia de manera lineal cuando la batería se iba agotando lo que conllevaba un aumento lineal del throttle.

Observando este comportamiento, se optó por obtener la recta de regresión de los datos obtenidos para estimar la dependencia del throttle de equilibrio con la tensión y usarlo como referencia para el “Altitude Hold”.

Una vez se concluye la búsqueda del throttle de equilibrio y su dependencia con la tensión de los motores, se pueden obtener las ecuaciones de las ESCs con el acople de las salidas de los PID para el modo “Altitude Hold”.

$$ESC_1 = throttle_{hover} + PID_{hover} - PID_{\phi} - PID_{\theta} - PID_{\varphi} \quad \textbf{(3.16)}$$
$$ESC_2 = throttle_{hover} + PID_{hover} + PID_{\phi} - PID_{\theta} + PID_{\varphi} \quad \textbf{(3.17)}$$
$$ESC_3 = throttle_{hover} + PID_{hover} + PID_{\phi} + PID_{\theta} - PID_{\varphi} \quad \textbf{(3.18)}$$
$$ESC_4 = throttle_{hover} + PID_{hover} - PID_{\phi} + PID_{\theta} + PID_{\varphi} \quad \textbf{(3.19)}$$

`act_esc_pwm()`: Este método se encarga de recuperar los valores de las ecuaciones implementadas en el método anterior y generar la señal PWM. Para hacerlo, se modifican los anchos de pulso de las señales generadas con por los timers de la siguiente manera:

```
TIM_M1_M2->setCaptureCompare(channel_motor1, esc_1, MICROSEC_COMPARE_FORMAT)
```

`act_us_pulse()`: Este método, ajeno a la actuación de los motores, consiste en generar el pulso para el sensor de ultrasónicos, este pulso es generado una vez cada 7,5 ms y se emplea para medir la distancia cuando el cuadricóptero se halla cerca del suelo.

CAPÍTULO 4. DISEÑO FINAL Y PRUEBAS DE VUELO

Este capítulo contiene información detallada acerca el resultado final del prototipo y la puesta en operación. Esto incluye información acerca del calibrado de los PID, la asignación de los controles de la radio, la puesta en marcha, el vuelo del cuadricóptero y los resultados de las pruebas de vuelo.

4.1. Prototipo

El resultado final del prototipo consiste en el resultado de la creación de una aeronave completamente funcional y preparada para cumplir sus objetivos. El prototipo de la Fig. 4.1 representa la versión final del dron, incorporando todas las características y funcionalidades descritas en esta memoria.

Fig. 4.1. Prototipo final del dron.

Los costes correspondientes a los elementos de hardware del prototipo del dron vienen detallados en la Tabla 4.2.

Tabla 4.1. Costes de los componentes principales.

Componente	Coste
DJI-F450	24,40 €
Adafruit Feather STM32F4	37,79 €
MPU6050	6,99 €
BMP280	8,88 €
FlySky i6	69,99 €
LiPo 3s	26,99 €
Kit Tmotor 2213	109,00 €
HC-SR04	3,09 €
TOTAL	**287,13 €**

4.2. Operación

Una vez se dispone del prototipo del dron en conjunto con el software desarrollado se puede empezar con la puesta en operación. En esta sección, se describen con detalle los controles y los procedimientos a seguir para poder volar el dron adecuadamente. Este apartado concluye con las pruebas de vuelo llevadas a cabo con el cuadricóptero.

4.2.1. Controles

Este apartado contiene la información acerca de la asignación de los diferentes canales de la radio y como repercuten en el pilotaje del dron. Existen una gran multitud de combinaciones y de maneras de controlar el dron con la radio, sin embargo, para este proyecto se ha optado por una de las configuraciones más clásicas e intuitivas. La asignación de los canales se puede ver detallada en la Tabla 4.2.

Tabla 4.2. Controles del dron.

Magnitud	Control	Recorrido	Canal
Throttle	Joystick Izquierdo	Vertical	Nº3
Roll	Joystick Derecho	Vertical	Nº1
Pitch	Joystick Derecho	Horizontal	Nº2
Yaw	Joystick Izquierdo	Horizontal	Nº4
Flight Mode	Interruptor SWD	-	Nº5

Se establece, para los controles del joystick con recorrido vertical e interruptores el mínimo cuando estos apuntan hacia abajo, y máximo cuando apuntan hacia arriba. Para los joysticks con recorrido horizontal, el mínimo corresponde a la

izquierda y el máximo a la derecha. Finalmente, para los interruptores el mínimo se establece cuando el interruptor apunta hacia abajo y el máximo hacia arriba.

Adicionalmente, se considera que el morro del dron corresponde a la bisectriz del ángulo formado por los brazos de color rojo. Respecto al yaw, se observa la dirección de giro desde un punto de vista por encima del dron. Aplicando este convenio, el impacto de los controles sobre el dron se puede ver en la Tabla 4.3.

Tabla 4.3. Respuesta a los controles del dron.

Control	Mínimo	Máximo
Throttle	Sin throttle	Máximo throttle
Roll	Izquierda	Derecha
Pitch	Adelante	Atrás
Yaw	Antihorario	Horario
Flight Mode	Altitude Hold	Stable

4.2.2. Puesta en marcha

Primero, es necesario conectar la batería LiPo 3s al cuadricóptero. Para lograrlo, debemos utilizar el conector XT60 que se encuentra en la placa electrónica integrada en el marco del dron. Una vez que los componentes del circuito estén alimentados, el microcontrolador iniciará la ejecución del programa.

Al ejecutar el código, se llevan a cabo las tareas de inicialización. El microcontrolador carga todas las variables globales en su memoria y luego procede a ejecutar la función `setup`. Esta función se encarga de realizar las calibraciones necesarias y llevar a cabo el montaje del dron.

Cuando el usuario conecte la batería al dron, notará que los ESCs comienzan a emitir pitidos. Estos pitidos indican que se están llevando a cabo los calibrados necesarios para los distintos elementos hardware del dron. Una vez que la calibración haya finalizado y la radio esté encendida, se escuchará un último pitido sincronizado, señalando una calibración exitosa. En este momento, el LED integrado en el microcontrolador se encenderá, como se muestra en la Fig. 4.2.

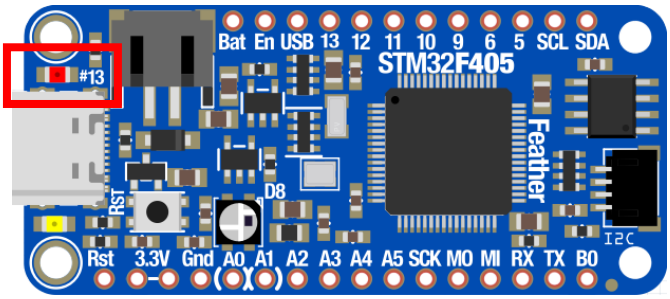

Fig. 4.2. Adafruit Feather STM32F405.

El LED nos indicará que el dron se encuentra en modo de vuelo "Disabled". Este modo implica que el dron está aislado del bucle principal del algoritmo de control, lo cual nos permite manipularlo físicamente con seguridad. Es importante destacar que siempre que interactuemos con el dron, este LED debe estar encendido, ya que indica la presencia de un sistema de seguridad en el bucle. Este sistema nos permite realizar acciones físicas en el dron, como desconectar la batería o acercarnos en caso de un accidente.

4.2.3. Calibrado de los PID

Uno de los aspectos más importantes es el calibrado de los PID. Como se ha mencionado anteriormente, los PID tienen tres componentes, la proporcional, integral y derivativa. Para que el algoritmo de control funcione adecuadamente es necesario calibrar los factores de los tres componentes del controlador de manera adecuada.

Si bien existen muchas técnicas para calibrar un PID, como laboratorios o simuladores, para un proyecto de estas magnitudes lo más habitual y asequible es realizar el calibrado de manera experimental. Este proceso experimental, para gente con conocimientos sobre la materia, puede resultar una labor sencilla sin mucho misterio. Sin embargo, desde la inexperiencia, llevar a cabo el calibrado ha sido una de las tareas más desafiantes del proyecto ya que requiere de muchas pruebas y algún que otro accidente para aprender a calibrarlos adecuadamente.

Para realizar el calibrado, se ha seguido una estrategia publicada por Joop Brokking en su canal de YouTube [21]:

1. Poner a 0 la parte proporcional, integral y derivativa.
2. Aumentar la parte derivativa.
3. Encender el dron, sujetarlo con la mano y augmentar throttle hasta que empiece a oscilar. Si no oscila, volver al paso 2.
4. Reducir parte derivativa un 25%.
5. Aumentar parte proporcional.

6. Encender el dron, volarlo de manera normal. Si el dron no oscila, volver al paso 4.
7. Reducir parte proporcional un 50%.
8. Augmentar parte integral.
9. Encender el dron, volarlo de manera normal. Si el dron no oscila, volver al paso 8.
10. Reducir parte integral un 50%.

4.2.4. Instrucciones de vuelo

Una vez que se hayan completado satisfactoriamente las tareas de inicialización del dron y se hayan calibrado los PID, es posible comenzar a volar con él. Para ello, es necesario realizar una transición hacia los modos de vuelo operativos del dron, que incluyen el modo “Stable” y “Altitude Hold”.

Para realizar la transición a los modos de vuelo operativos, debemos utilizar la combinación correspondiente del joystick de control de throttle-yaw. Moviendo el joystick hacia abajo a la derecha, como se muestra en la Fig. 4.3, el microcontrolador detectará esta orden de la radio y cambiará del modo de vuelo "Disabled" al modo de "Mounting".

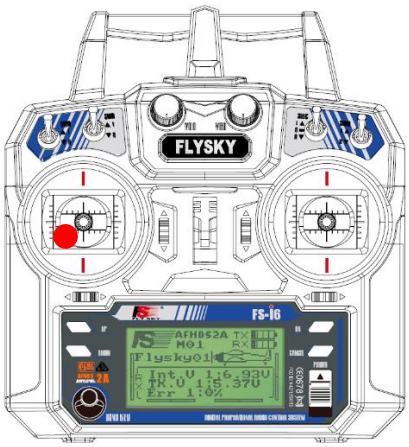

Fig. 4.3. Transición de “Disabled” a "Mounting".

Una vez que el cuadricóptero realice esta transición de modo de vuelo, entrará en un estado de espera y, aguardará a que el stick del yaw se centre para realizar el cambio al modo “Stable”, como se muestra en la Fig. 4.4.

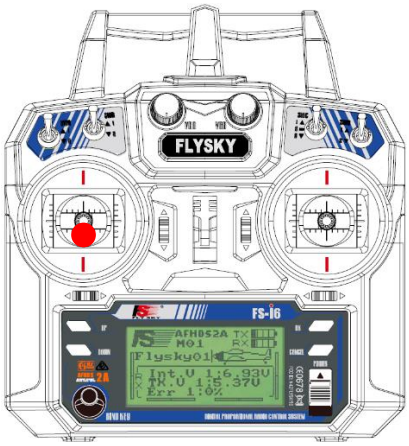

Fig. 4.4. Transición de "Mounting" a "Stable".

Una vez que el dron esté en modo “Stable”, responderá a las órdenes de la radio y, al proporcionarle el throttle necesario, se elevará para despegar.

Cuando se halle en este modo, se podrá realizar la transición al modo "Altitude Hold". Para hacerlo, simplemente se debe accionar hacia abajo el interruptor superior derecho de la radio, como se indica en la Fig. 4.5, lo cual hará que el dron pase de forma autónoma al modo de control de altitud basado en las lecturas de presión. Si el usuario decide volver a operar el dron en modo “Stable”, simplemente debe accionar nuevamente el interruptor para devolver el dron a su estado anterior.

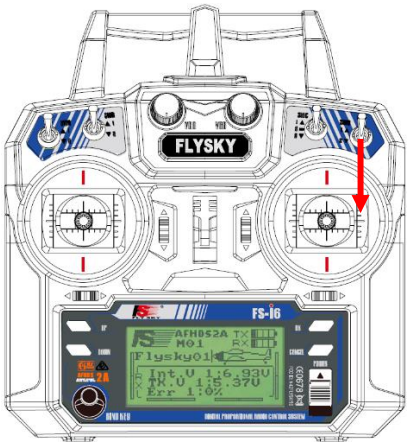

Fig. 4.5. Transición de "Stable" a "Altitude Hold".

El modo de vuelo "Disabled" también se utiliza como una función de seguridad para el cuadricóptero y está siempre disponible desde cualquier modo de vuelo en el que se encuentre el dron. Simplemente moviendo el joystick del throttle-yaw hacia abajo a la derecha, como se indica en la Fig. 4.6, el dron detendrá sus motores y realizará la transición al modo de vuelo "Disabled". Al hacer esta

transición, el LED de seguridad del dron se volverá a encender, indicando que puede ser manipulado físicamente por el operador de manera segura.

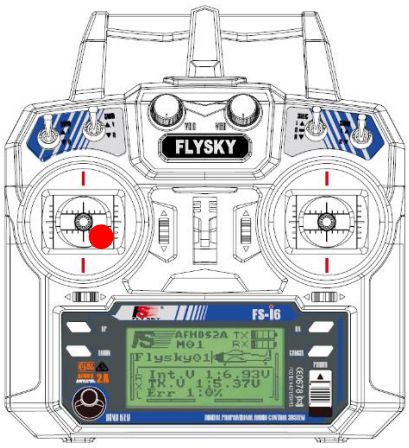

Fig. 4.6. Transición a "Disabled".

A continuación, en el diagrama de flujo de la Fig. 4.7, se pueden observar los diferentes modos de vuelo del dron y sus posibles transiciones.

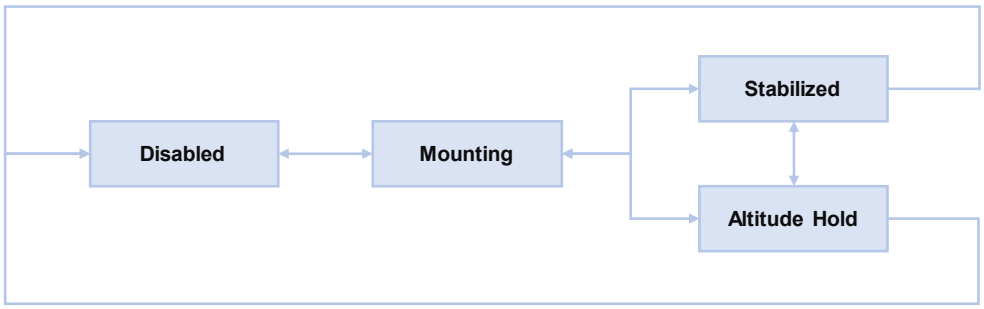

Fig. 4.7. Transiciones entre modos de vuelo.

4.2.5. Vuelo del cuadricóptero

Siguiendo las configuraciones descritas anteriormente y conociendo los controles del dron y las transiciones asociados a los modos de vuelo se han llevado a cabo las pruebas necesarias para verificar el correcto funcionamiento del sistema.

El primer modo de vuelo a implementar es el modo "Stable". Este modo se basa en las lecturas de la IMU para corregir la posición horizontal del dron, de manera que cuando se suelta el joystick, el dron se mantenga completamente estable. Se realizaron numerosas pruebas y ensayos para verificar el funcionamiento de este modo de vuelo en diversas condiciones atmosféricas, especialmente con

viento, y se llevaron a cabo diferentes tipos de operaciones. Se ha comprobado y confirmado el correcto funcionamiento y la viabilidad de operar el dron en este modo de vuelo. En la Fig. 4.8, se puede observar al dron volando en “Stable”.

Fig. 4.8. Dron volando en modo "Stable.

Posteriormente, con la llegada del modo de vuelo “Altitude Hold” fue necesario probar como el dron era capaz de controlar su propia altitud. Como se ha descrito en esta memoria, cuando se controla el dron empleando el barómetro, se utiliza un throttle de equilibrio basado en una recta de regresión derivada de una prueba del modo “Stable”. Por esta cuestión ha sido importante realizar pruebas en diferentes condiciones, sobre todo atmosféricas ya que el throttle de equilibrio puede variar en función de la densidad del aire (afectada por la temperatura). Las pruebas se han realizado satisfactoriamente con temperaturas desde los 0ºC hasta los 30ºC y con diferentes intensidades de viento, las cuales también afectan a la lectura del sensor. En la Fig. 4.9 se puede observar la radio en configuración “Altitude Hold” (véase el switch SWD accionado).

Fig. 4.9. Radio en modo "Altitude Hold”.

Adicionalmente, se implementó un control de altitud basado en el ultrasónico el cual fue discontinuado tras incorporar el barómetro. Este sensor era el que se empleaba para las medidas de altitud. El modo de vuelo implementado permitía realizar vuelos en “Altitude Hold” pero empleando referencias provenientes del sensor de ultrasonidos, a poca distancia del suelo. Se realizaron vuelos desde el Dronlab (ver Fig. 4.10) para verificar la funcionalidad de este modo de vuelo.

Fig. 4.10. Prototipo en el Dronlab de la UPC. A punto de realizar un vuelo basado en lecturas de altitud del HC-SR04.

CONCLUSIONES

Este Trabajo de Fin de Grado ha abordado el desarrollo y programación de un dron utilizando el STM32F405 como microcontrolador integrado en el frame DJI-F450. A lo largo del proyecto, se han explorado y aplicado una variedad de conocimientos y habilidades en el campo de la electrónica y la programación los cuales han confeccionado el prototipo final.

El objetivo principal de este trabajo fue diseñar y construir un dron capaz de realizar vuelos estables y controlados, utilizando el STM32F405 como controlador de vuelo. Para hacerlo, se ha seguido la metodología basada en implementaciones modulares y pruebas exhaustivas detallada en la introducción de esta memoria.

Durante el proceso de desarrollo, se ha utilizado el entorno de programación y desarrollo proporcionado por Adafruit para programar el STM32F405. Esto ha permitido aprovechar al máximo las características y capacidades del microcontrolador, así como la amplia gama de librerías y herramientas disponibles.

Uno de los aspectos más destacados de este proyecto ha sido el diseño e implementación del sistema de control de vuelo. Se han utilizado algoritmos de control PID (Proporcional, Integral y Derivativo) para mantener la estabilidad y controlar los movimientos del dron en tiempo real. Además, se ha integrado un sistema de control de altitud basado en un barómetro, mediante lecturas de presión.

Otro aspecto importante de este trabajo ha sido la integración y cohesión de diferentes sensores y actuadores en el dron. Se han utilizado sensores como acelerómetros y giroscopios para obtener datos precisos sobre la orientación y el movimiento del dron. Estos datos, mediante el algoritmo de control implementado, se han utilizado para llevar a cabo el control sobre los motores del dron, los cuales han servido para obtener un resultado operacional del cuadricóptero.

En términos de resultados, el dron desarrollado ha demostrado un rendimiento satisfactorio en las pruebas de vuelo realizadas. Ha sido capaz de mantener una estabilidad adecuada, responder correctamente a las entradas de control y cumplir con los objetivos de navegación establecidos, lo que ha permitido un funcionamiento coherente y eficiente del dron. Los objetivos de este proyecto final de grado se han satisfecho satisfactoriamente e incluso se han podido implementar funcionalidades adicionales a los objetivos originales.

En conclusión, este Trabajo de Fin de Grado ha sido un proyecto desafiante pero gratificante que ha permitido adquirir conocimientos profundos sobre el diseño, programación y control de drones utilizando el STM32F405. El resultado final es

un dron funcional y eficiente, capaz de llevar a cabo vuelos estables y reaccionar a los comandos provenientes del piloto. El trabajo realizado sienta las bases para futuras implementaciones y mejoras operacionales, abriendo así un amplio abanico de posibilidades para definir el camino a seguir.

BIBLIOGRAFÍA

[1] «What is U-space | EASA». https://www.easa.europa.eu/en/what-u-space (accedido 5 de julio de 2023).

[2] «Flight Controller Processors Explained: F1, F3, F4, G4, F7, H7». https://oscarliang.com/f1-f3-f4-flight-controller/#Understanding-the-Different-STM32-Processors-in-Flight-Controllers (accedido 5 de julio de 2023).

[3] «Betaflight - Pushing the Limits of UAV Performance». https://betaflight.com/ (accedido 5 de julio de 2023).

[4] «ArduPilot - Versatile, Trusted, Open». https://ardupilot.org/ (accedido 5 de julio de 2023).

[5] «Drone Arduino». https://arduproject.es/conceptos-generales-sobre-drones/ (accedido 5 de julio de 2023).

[6] «Project YMFC-32 autonomous - The STM32 Arduino autonomous quadcopter.» http://www.brokking.net/ymfc-32_auto_main.html (accedido 5 de julio de 2023).

[7] «Arduino». https://www.arduino.cc/ (accedido 5 de julio de 2023).

[8] «Flame Wheel ARF KIT - Features | DJI». https://www-v1.dji.com/flame-wheel-arf/feature.html (accedido 5 de julio de 2023).

[9] «Adafruit STM32F405 Feather Express». https://learn.adafruit.com/adafruit-stm32f405-feather-express/overview (accedido 5 de julio de 2023).

[10] «MPU-6050 | TDK InvenSense». https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/ (accedido 5 de julio de 2023).

[11] «Pressure Sensor BMP280». https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/bmp280/ (accedido 5 de julio de 2023).

[12] «Ultrasonic Ranging Module HC-SR04». www.Elecfreaks.com (accedido 7 de julio de 2023).

[13] «FS-i6». https://www.flysky-cn.com/fsi6 (accedido 5 de julio de 2023).

[14] «PDB-XT60 – Matek Systems». http://www.mateksys.com/?portfolio=pdb-xt60 (accedido 7 de julio de 2023).

[15] «T-Motor AIR GEAR 350 Set ». https://rc-innovations.es/shop/motores-T-motor-air-gear-350-pack-esc-20A-helices-phantom-F450#attr= (accedido 5 de julio de 2023).

[16] «GitHub - adafruit/Adafruit_SPIFlash: Arduino library for external (Q)SPI flash device». https://github.com/adafruit/Adafruit_SPIFlash (accedido 5 de julio de 2023).

[17] «HardwareTimer library · stm32duino/Arduino_Core_STM32». https://github.com/stm32duino/Arduino_Core_STM32/wiki/HardwareTimer-library (accedido 5 de julio de 2023).

[18] «Wire - Arduino Reference». https://www.arduino.cc/reference/en/language/functions/communication/wire/ (accedido 5 de julio de 2023).

[19] «MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2 MPU-6000/MPU-6050 Register Map and Descriptions», 2013.

[20] «BMP280 - Digital Pressure Sensor». https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf (accedido 5 de julio de 2023).

[21] «YMFC-3D – Quadcopter PID controller and PID tuning.» https://www.youtube.com/watch?v=JBvnB0279-Q (accedido 5 de julio de 2023).

ANEXOS

Anexo 1: Código implementado

El repositorio de código del proyecto se puede encontrar en línea en el siguiente enlace: https://github.com/leonprietobailo/DroneSTM32F405.

main.ino:

```
#include <Wire.h>
#include "LowPass.h"

// Flight Mode Enumeration and Initialization
enum FlightMode{
  FM_disabled,
  FM_mounting,
  FM_stable,
  FM_alt_hold
};

FlightMode fm = FM_disabled;

// BMP280: Adresses
#define BMP280_ADDRESS 0x76
#define BMP280_REGISTER_PRESSURE_MSB 0xF7
#define BMP280_REGISTER_PRESSURE_LSB 0xF8
#define BMP280_REGISTER_PRESSURE_XLSB 0xF9
#define BMP280_REGISTER_TEMPERATURE_MSB 0xFA
#define BMP280_REGISTER_TEMPERATURE_LSB 0xFB
#define BMP280_REGISTER_TEMPERATURE_XLSB 0xFC

// BMP280: Compensation Parameters
uint16_t dig_T1;
int16_t dig_T2;
int16_t dig_T3;
uint16_t dig_P1;
int16_t dig_P2;
int16_t dig_P3;
int16_t dig_P4;
int16_t dig_P5;
int16_t dig_P6;
int16_t dig_P7;
int16_t dig_P8;
int16_t dig_P9;

int64_t var1, var2, p;
int32_t p_32;
int32_t var1_32, var2_32;
int32_t t_fine;

// BMP280: Temperature and pressure variables
int32_t temperature_msb, temperature_lsb, temperature_xlsb, adc_T, t_temp;
float T, P, P_filt;
int32_t pressure_msb, pressure_lsb, pressure_xlsb, adc_P;

// BMP280: Pressure signal smoothened
float pressure_total_avarage, pressure_rotating_mem[12], actual_pressure_fast,
actual_pressure_slow = 101325, actual_pressure_diff, actual_pressure;
uint8_t pressure_rotating_mem_location, barometer_counter;

// BMP280: Parachute throttle:
int32_t parachute_buffer[20], parachute_throttle;
float pressure_parachute_previous;
uint8_t parachute_rotating_mem_location;



// HC-SR04
```

```
#define trigger_pin PC3  // Trigger Pin Ultrasonico
#define echo_pin PC2     // Echo Pin Ultrasonico
float distance;
float velocity;
LowPass<2> lp(3, 1e3, true);
float distance_filt;
float velocityFilt;
float velocity_raw;
float sent_last_pulse, duration, pulse_start, pulse_end, computed_distance;
bool pulse_sent;
long cAir = 331.3 + 20.0 * 0.606;
unsigned long timeLast;
float prev_distance, prev_distance_filt;

// LED
volatile long led_timer;

// BUZZER
int buzzer_timer;
bool tone_on;

// SPI Flash
#define pin_BUZZER PA6
#define length_str 20000
uint16_t voltage_str[length_str];
uint16_t throttle_str[length_str];
uint16_t n_str = 0;

// MOTORS
long time_motores_start, time_1, time_2, time_ON;

// FlightSky i6
#define pin_PPM PA4
#define number_channels 8
uint64_t pulse_instant[number_channels * 2 + 2];
uint16_t remote_channel[number_channels];
uint8_t flank_count = 1;
int16_t throttle;


// Battery
#define pin_BAT PA5
float battery_voltage;

// PID: Variables
float roll_level_adjust, pitch_level_adjust;
float pid_error_temp;
float pid_i_mem_roll, pid_roll_setpoint, gyro_roll_input, pid_output_roll,
pid_last_roll_d_error;
float pid_i_mem_pitch, pid_pitch_setpoint, gyro_pitch_input, pid_output_pitch,
pid_last_pitch_d_error;
float pid_i_mem_yaw, pid_yaw_setpoint, gyro_yaw_input, pid_output_yaw,
pid_last_yaw_d_error;
// Check:
float pid_i_mem_altitude, pid_i_mem_altitude_v2, pid_altitude_input,
pid_altitude_v2_input, pid_output_altitude, pid_output_altitude_v2,
pid_last_altitude_d_error, pid_last_altitude_v2_d_error;
float pid_altitude_setpoint, pid_error_gain_altitude;
uint16_t last_alt_hold_PID;
float pid_t_control_error, pid_t_output, pid_rate_control_error, pid_i_mem_rate,
pid_rate_error_prev, pid_output_rate;


// PID: Roll
float pid_p_gain_roll = 0.9;
float pid_i_gain_roll = 0.009;
float pid_d_gain_roll = 4;
int pid_max_roll = 400;

// PID: Pitch
```

```
123. float pid_p_gain_pitch = pid_p_gain_roll;
124. float pid_i_gain_pitch = pid_i_gain_roll;
125. float pid_d_gain_pitch = pid_d_gain_roll;
126. int pid_max_pitch = 400;
127.
128. // PID: Yaw
129. float pid_p_gain_yaw = 0.75;
130. float pid_i_gain_yaw = 0.01;
131. float pid_d_gain_yaw = 0;
132. int pid_max_yaw = 400;
133.
134. // PID: Altitude
135. float pid_p_gain_altitude = 1.4;
136. float pid_i_gain_altitude = 0.2; //0.2;
137. float pid_d_gain_altitude = 0.75; //0.75;
138. int pid_max_altitude = 30;
139.
140. // MPU6050: Address, setup and callibration
141.
142. #define MPU6050_ADDRESS 0x68
143. #define MPU6050_ACCEL_XOUT_H 0x3B
144. #define MPU6050_PWR_MGMT_1 0x6B
145. #define MPU6050_GYRO_CONFIG 0x1B
146. #define MPU6050_ACCEL_CONFIG 0x1C
147. #define MPU6050_CONFIG 0x1A
148.
149. boolean auto_level = true;
150. int16_t manual_acc_pitch_cal_value = 0;
151. int16_t manual_acc_roll_cal_value = 0;
152. uint8_t use_manual_calibration = false;
153. int16_t manual_gyro_pitch_cal_value = 0;
154. int16_t manual_gyro_roll_cal_value = 0;
155. int16_t manual_gyro_yaw_cal_value = 0;
156. int16_t manual_x_cal_value = 0;
157. int16_t manual_y_cal_value = 0;
158. int16_t manual_z_cal_value = 0;
159. int16_t cal_int;
160. int16_t temperature;
161. int16_t acc_x, acc_y, acc_z;
162. int16_t gyro_pitch, gyro_roll, gyro_yaw;
163. int32_t acc_total_vector;
164. int32_t gyro_roll_cal, gyro_pitch_cal, gyro_yaw_cal;
165. int32_t acc_x_cal, acc_y_cal, acc_z_cal;
166. float angle_roll_acc, angle_pitch_acc, angle_pitch, angle_roll;
167.
168. // ESC
169. #define pin_motor1 PC6  // Pin motor 1  GPIO 6
170. #define pin_motor2 PC7  // Pin motor 2  GPIO 5
171. #define pin_motor3 PB9  // Pin motor 3  GPIO 10
172. #define pin_motor4 PB8  // Pin motor 4  GPIO 9
173. int16_t esc_1, esc_2, esc_3, esc_4;
174.
175. TIM_TypeDef *TIM_DEF_M1_M2 = TIM3;
176. TIM_TypeDef *TIM_DEF_M3_M4 = TIM4;
177.
178. uint32_t channel_motor1 =
STM_PIN_CHANNEL(pinmap_function(digitalPinToPinName(pin_motor1), PinMap_PWM));
179. uint32_t channel_motor2 =
STM_PIN_CHANNEL(pinmap_function(digitalPinToPinName(pin_motor2), PinMap_PWM));
180. uint32_t channel_motor3 =
STM_PIN_CHANNEL(pinmap_function(digitalPinToPinName(pin_motor3), PinMap_PWM));
181. uint32_t channel_motor4 =
STM_PIN_CHANNEL(pinmap_function(digitalPinToPinName(pin_motor4), PinMap_PWM));
182.
183. HardwareTimer *TIM_M1_M2 = new HardwareTimer(TIM_DEF_M1_M2);
184. HardwareTimer *TIM_M3_M4 = new HardwareTimer(TIM_DEF_M3_M4);
185.
186. // Loop timer
187. uint32_t loop_timer;
188.
```

```
189. // Error signal
190. uint8_t error;
191.
192.
193. // Setup routine
194. void setup() {
195.   Serial.begin(57600);
196.   delay(5000);
197.
198.   init_components();
199.   led_off();
200.
201.   while (remote_channel[1] < 990 || remote_channel[2] < 990 || remote_channel[3] <
990 || remote_channel[4] < 990) {
202.     read_rc();
203.     delay(4);
204.   }
205.
206.   loop_timer = micros();
207.   led_on();
208.   Serial.println("Setup finished");
209. }
210.
211. // Main routine
212. void loop() {
213.
214.   reference_computation();
215.   read_process_units();
216.   controllers();
217.   actuators();
218.   diagnostics();
219.
220.   if (micros() - loop_timer > 4050) {
221.     Serial.println("LOOP SLOW");
222.   }
223.
224.   while (micros() - loop_timer < 4000);
225.   loop_timer = micros();
226. }
227.
```

init_components.ino:

```
 1. void init_components() {
 2.   init_flash();
 3.   init_led();
 4.   init_ultrasonic();
 5.   init_rc();
 6.   init_esc();
 7.   init_imu();
 8.   init_barometer();
 9. }
10.
11.
12. void init_flash(void) {
13.   // Init external flash
14.   flash.begin();
15.
16.   // Open file system on the flash
17.   if (!fatfs.begin(&flash)) {
18.     Serial.println("Error: filesystem is not existed. Please try SdFat_format example
to make one.");
19.     while (1) {
20.       yield();
21.       delay(1);
22.     }
23.   }
24.   myFile = fatfs.open("data.csv", FILE_WRITE);
25.   Serial.println("initialization done.");
26. }
27.
28. void init_led() {
29.   pinMode(PC1, OUTPUT);
30. }
31.
32. void init_ultrasonic() {
33.   pinMode(trigger_pin, OUTPUT);
34.   digitalWrite(trigger_pin, LOW);
35.   pinMode(echo_pin, INPUT);
36.   attachInterrupt(digitalPinToInterrupt(echo_pin), read_ultrasonic, CHANGE);
37. }
38.
39. void init_rc() {
40.   pinMode(pin_PPM, INPUT);  // YAW
41.   attachInterrupt(digitalPinToInterrupt(pin_PPM), read_PPM, CHANGE);
42. }
43.
44. void init_esc() {
45.   TIM_M1_M2->setPWM(channel_motor1, pin_motor1, 250, 0);
46.   TIM_M1_M2->setPWM(channel_motor2, pin_motor2, 250, 0);
47.   TIM_M3_M4->setPWM(channel_motor3, pin_motor3, 250, 0);
48.   TIM_M3_M4->setPWM(channel_motor4, pin_motor4, 250, 0);
49. }
50.
51.
52. void init_imu(void) {
53.   Wire.begin();
54.   Wire.beginTransmission(MPU6050_ADDRESS);
55.   error = Wire.endTransmission();
56.   while (error != 0) {
57.     delay(4);
58.   }
59.
60.   Wire.beginTransmission(MPU6050_ADDRESS);
61.   Wire.write(MPU6050_PWR_MGMT_1);
62.   Wire.write(0x00);
63.   Wire.endTransmission();
64.
65.   Wire.beginTransmission(MPU6050_ADDRESS);
66.   Wire.write(MPU6050_GYRO_CONFIG);
```

```
  Wire.write(0x08);
  Wire.endTransmission();

  Wire.beginTransmission(MPU6050_ADDRESS);
  Wire.write(MPU6050_ACCEL_CONFIG);
  Wire.write(0x10);
  Wire.endTransmission();

  Wire.beginTransmission(MPU6050_ADDRESS);
  Wire.write(MPU6050_CONFIG);
  Wire.write(0x03);
  Wire.endTransmission();

  if (use_manual_calibration) cal_int = 2000;
  else {
    cal_int = 0;
    manual_gyro_pitch_cal_value = 0;
    manual_gyro_roll_cal_value = 0;
    manual_gyro_yaw_cal_value = 0;
  }

  if (cal_int != 2000) {
    for (cal_int = 0; cal_int < 2000; cal_int++) {
      if (cal_int % 25 == 0) digitalWrite(PB4, !digitalRead(PB4));
      read_imu();
      gyro_roll_cal += gyro_roll;
      gyro_pitch_cal += gyro_pitch;
      gyro_yaw_cal += gyro_yaw;
      acc_x_cal += acc_x;
      acc_y_cal += acc_y;
      acc_z_cal += acc_z;
      delay(4);
    }
    gyro_roll_cal /= 2000;
    gyro_pitch_cal /= 2000;
    gyro_yaw_cal /= 2000;
    acc_x_cal /= 2000;
    acc_y_cal /= 2000;
    acc_z_cal /= 2000;
    manual_gyro_pitch_cal_value = gyro_pitch_cal;
    manual_gyro_roll_cal_value = gyro_roll_cal;
    manual_gyro_yaw_cal_value = gyro_yaw_cal;
    manual_x_cal_value = acc_x_cal;
    manual_y_cal_value = acc_y_cal;
    manual_z_cal_value = acc_z_cal - 4096;
  }
}


void init_barometer() {

  Wire.beginTransmission(BMP280_ADDRESS);
  Wire.write(0x88);
  Wire.endTransmission();
  Wire.requestFrom(BMP280_ADDRESS, 24);
  dig_T1 = Wire.read() | Wire.read() << 8;
  dig_T2 = Wire.read() | Wire.read() << 8;
  dig_T3 = Wire.read() | Wire.read() << 8;
  dig_P1 = Wire.read() | Wire.read() << 8;
  dig_P2 = Wire.read() | Wire.read() << 8;
  dig_P3 = Wire.read() | Wire.read() << 8;
  dig_P4 = Wire.read() | Wire.read() << 8;
  dig_P5 = Wire.read() | Wire.read() << 8;
  dig_P6 = Wire.read() | Wire.read() << 8;
  dig_P7 = Wire.read() | Wire.read() << 8;
  dig_P8 = Wire.read() | Wire.read() << 8;
  dig_P9 = Wire.read() | Wire.read() << 8;

  Wire.beginTransmission(BMP280_ADDRESS);
  Wire.write(0xF4);
```

```
137.   Wire.write(0x57);
138.   Wire.endTransmission();
139.
140.   Wire.beginTransmission(BMP280_ADDRESS);
141.   Wire.write(0xF5);
142.   Wire.write(0x08);
143.   Wire.endTransmission();
144. }
145.
```

reference_computation.ino:

```
 1. void reference_computation(){
 2.       ref_mode_management();
 3.       ref_gen();
 4. }
 5.
 6. void ref_mode_management(){
 7.       if (remote_channel[3] < 1100 && remote_channel[4] < 1100) fm = FM_mounting;
 8.       if (fm == FM_mounting && remote_channel[3] < 1100 && remote_channel[4] > 1450) fm
= FM_stable;
 9.       if (fm >=2 && remote_channel[3] < 1050 && remote_channel[4] > 1950) fm =
FM_disabled;
10.       if (fm >=2 && remote_channel[6] < 1500) fm = FM_stable;
11.       if (fm >=2 && remote_channel[6] >= 1500) fm = FM_alt_hold;
12. }
13.
14. void ref_gen(){
15.
16.       if(fm == FM_disabled) led_on();
17.
18.       if(fm == FM_mounting){
19.                 throttle = 950;
20.                 led_off();
21.                 angle_pitch = angle_pitch_acc;
22.           angle_roll = angle_roll_acc;
23.           pid_i_mem_roll = 0;
24.           pid_last_roll_d_error = 0;
25.           pid_i_mem_pitch = 0;
26.           pid_last_pitch_d_error = 0;
27.           pid_i_mem_yaw = 0;
28.           pid_last_yaw_d_error = 0;
29.       }
30.
31.       if(fm == FM_stable){
32.                 throttle = remote_channel[3];
33.           pid_i_mem_altitude = 0;
34.           pid_last_altitude_d_error = 0;
35.           pid_altitude_setpoint = actual_pressure;
36.       }
37.
38.       if(fm == FM_alt_hold){
39.       throttle = -63.4 * battery_voltage + 2203;
40.
41.       if(remote_channel[3] > 1750) pid_altitude_setpoint -= 1.0 / 250.0;
42.       else if(remote_channel[3] < 1250 && distance > 50) pid_altitude_setpoint += 1.0 /
250.0;
43.       }
44. }
45.
```

read_process_units.ino:

```
 1. void read_process_units() {
 2.   read_battery();
 3.   read_rc();
 4.   read_imu();
 5.   process_imu();
 6.   read_barometer();
 7. }
 8.
 9. void read_battery(void) {
10.   battery_voltage = battery_voltage * 0.92 + ((float)analogRead(PA7) / 352.27 *
0.9838);
11.   if (battery_voltage < 10.0 && error == 0) error = 1;
12. }
13.
14. void read_rc() {
15.   if (flank_count == 18) {
16.     for (int i = 1; i <= number_channels; i++) {
17.       remote_channel[i] = map(pulse_instant[2 * i] - pulse_instant[2 * i - 1], 600,
1600, 1000, 2000);
18.     }
19.   }
20. }
21.
22. void read_imu(void) {
23.   Wire.beginTransmission(MPU6050_ADDRESS);
24.   Wire.write(MPU6050_ACCEL_XOUT_H);
25.   Wire.endTransmission();
26.   Wire.requestFrom(MPU6050_ADDRESS, 14);
27.   acc_x = Wire.read() << 8 | Wire.read();
28.   acc_y = Wire.read() << 8 | Wire.read();
29.   acc_z = Wire.read() << 8 | Wire.read();
30.   temperature = Wire.read() << 8 | Wire.read();
31.   gyro_pitch = Wire.read() << 8 | Wire.read();
32.   gyro_roll = Wire.read() << 8 | Wire.read();
33.   gyro_yaw = Wire.read() << 8 | Wire.read();
34.   //gyro_pitch *= -1;
35.   gyro_roll *= -1;
36.   gyro_yaw *= -1;
37.   acc_x -= manual_x_cal_value;
38.   acc_y -= manual_y_cal_value;
39.   acc_z -= manual_z_cal_value;
40.   gyro_roll -= manual_gyro_roll_cal_value;
41.   gyro_pitch -= manual_gyro_pitch_cal_value;
42.   gyro_yaw -= manual_gyro_yaw_cal_value;
43. }
44.
45. void process_imu() {
46.
47.   gyro_roll_input = (gyro_roll_input * 0.7) + (((float)gyro_roll / 65.5) * 0.3);
48.   gyro_pitch_input = (gyro_pitch_input * 0.7) + (((float)gyro_pitch / 65.5) * 0.3);
49.   gyro_yaw_input = (gyro_yaw_input * 0.7) + (((float)gyro_yaw / 65.5) * 0.3);
50.
51.
52.   angle_pitch += (float)gyro_pitch * 0.0000611;
53.   angle_roll += (float)gyro_roll * 0.0000611;
54.
55.   angle_pitch -= angle_roll * sin((float)gyro_yaw * 0.000001066);
56.   angle_roll += angle_pitch * sin((float)gyro_yaw * 0.000001066);
57.
58.   acc_total_vector = sqrt((acc_x * acc_x) + (acc_y * acc_y) + (acc_z * acc_z));
59.
60.   if (abs(acc_y) < acc_total_vector) {
61.     angle_pitch_acc = asin((float)acc_y / acc_total_vector) * 57.296;
62.   }
63.   if (abs(acc_x) < acc_total_vector) {
64.     angle_roll_acc = asin((float)acc_x / acc_total_vector) * 57.296;
65.   }
```

```
66.
67.    angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;
68.    angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;
69.
70.    pitch_level_adjust = angle_pitch * 15;
71.    roll_level_adjust = angle_roll * 15;
72.
73.    if (!auto_level) {
74.      pitch_level_adjust = 0;
75.      roll_level_adjust = 0;
76.    }
77. }
78.
79. void read_barometer() {
80.    if (barometer_counter == 2) {
81.      Wire.beginTransmission(BMP280_ADDRESS);
82.      Wire.write(BMP280_REGISTER_PRESSURE_MSB);
83.      Wire.endTransmission();
84.    }
85.
86.    if (barometer_counter == 4) {
87.      Wire.requestFrom(BMP280_ADDRESS, 6);
88.
89.      pressure_msb = Wire.read();
90.      pressure_lsb = Wire.read();
91.      pressure_xlsb = Wire.read();
92.      temperature_msb = Wire.read();
93.      temperature_lsb = Wire.read();
94.      temperature_xlsb = Wire.read();
95.
96.      adc_P = pressure_msb << 12 | pressure_lsb << 4 | pressure_xlsb >> 4;
97.      adc_T = temperature_msb << 12 | temperature_lsb << 4 | temperature_xlsb >> 4;
98.
99.      bmp280_compensate_P_int64();
100.     bmp280_compensate_T_int32();
101.
102.     pressure_total_avarage -= pressure_rotating_mem[pressure_rotating_mem_location];
103.     pressure_rotating_mem[pressure_rotating_mem_location] = P;
104.     pressure_total_avarage += pressure_rotating_mem[pressure_rotating_mem_location];
105.     pressure_rotating_mem_location++;
106.     if (pressure_rotating_mem_location == 12) pressure_rotating_mem_location = 0;
107.     actual_pressure_fast = (float)pressure_total_avarage / 12.0;
108.     actual_pressure_slow = actual_pressure_slow * (float)0.985 + actual_pressure_fast
* (float)0.015;
109.
110.     actual_pressure_diff = actual_pressure_slow - actual_pressure_fast;
111.     if (actual_pressure_diff > 8) actual_pressure_diff = 8;
112.     if (actual_pressure_diff < -8) actual_pressure_diff = -8;
113.
114.     if (actual_pressure_diff > 1 || actual_pressure_diff < -1) actual_pressure_slow -
= actual_pressure_diff / 6.0;
115.     actual_pressure = actual_pressure_slow;
116.
117.     parachute_throttle -= parachute_buffer[parachute_rotating_mem_location];
118.     parachute_buffer[parachute_rotating_mem_location] = actual_pressure * 10 -
pressure_parachute_previous;
119.     parachute_throttle += parachute_buffer[parachute_rotating_mem_location];
120.     pressure_parachute_previous = actual_pressure * 10;
121.     parachute_rotating_mem_location++;
122.     if (parachute_rotating_mem_location == 20)parachute_rotating_mem_location = 0;
123.
124.     barometer_counter = 0;
125.   }
126.   barometer_counter++;
127. }
128.
129. void bmp280_compensate_T_int32() {
130.   var1_32 = ((((adc_T >> 3) - ((int32_t)dig_T1 << 1))) * ((int32_t)dig_T2)) >> 11;
131.   var2_32 = (((((adc_T >> 4) - ((int32_t)dig_T1)) * ((adc_T >> 4) -
((int32_t)dig_T1))) >> 12) * ((int32_t)dig_T3)) >> 14;
```

```
132.   t_fine = var1_32 + var2_32;
133.   t_temp = (t_fine * 5 + 128) >> 8;
134.   T = (float)t_temp / 100.0;
135. }
136.
137. void bmp280_compensate_P_int64() {
138.   var1 = ((int64_t)t_fine) - 128000;
139.   var2 = var1 * var1 * (int64_t)dig_P6;
140.   var2 = var2 + ((var1 * (int64_t)dig_P5) << 17);
141.   var2 = var2 + (((int64_t)dig_P4) << 35);
142.   var1 = ((var1 * var1 * (int64_t)dig_P3) >> 8) + ((var1 * (int64_t)dig_P2) << 12);
143.   var1 = (((((int64_t)1) << 47) + var1)) * ((int64_t)dig_P1) >> 33;
144.   if (var1 != 0) {
145.     p = 1048576 - adc_P;
146.     p = (((p << 31) - var2) * 3125) / var1;
147.     var1 = (((int64_t)dig_P9) * (p >> 13) * (p >> 13)) >> 25;
148.     var2 = (((int64_t)dig_P8) * p) >> 19;
149.     p = ((p + var1 + var2) >> 8) + (((int64_t)dig_P7) << 4);
150.     p_32 = (int32_t)p;
151.     P = float(p_32) / 256.0;
152.   }
153. }
154.
155.
156. ////////////////////////////////////////////////////////////////////
157. /// FUNCTIONS TRIGGERED BY INTERRUPTIONS                         ///
158. ////////////////////////////////////////////////////////////////////
159.
160. void read_ultrasonic() {
161.   if (pulse_sent) {
162.     if (digitalRead(echo_pin) == HIGH) {
163.       pulse_start = micros();
164.     } else {
165.       pulse_end = micros();
166.       duration = pulse_end - pulse_start;
167.       computed_distance = duration / 1e6 * cAir * 1e2 / 2.0;
168.
169.       if (computed_distance < 10) {
170.         computed_distance = 10;
171.       }
172.       if (computed_distance < 200) {
173.         prev_distance = distance;
174.         prev_distance_filt = distance_filt;
175.         distance = computed_distance;
176.         distance_filt = lp.filt(distance);
177.         velocity_raw = (distance - prev_distance) / (micros() - timeLast) * 1e6;
178.         velocityFilt = velocityFilt * 0.8 + 0.2 * (distance_filt -
prev_distance_filt) / (micros() - timeLast) * 1e6;
179.         velocity = 0.95 * velocity + 0.05 * (distance - prev_distance) / (micros() -
timeLast) * 1e6;
180.
181.         timeLast = micros();
182.       }
183.       pulse_sent = false;
184.     }
185.   }
186. }
187.
188. void read_PPM() {
189.   if (micros() - pulse_instant[flank_count - 1] > 2500) flank_count = 0;
190.   pulse_instant[flank_count] = micros();
191.   flank_count++;
192. }
193.
```

controllers.ino:

```
 1. void controllers() {
 2.   cnt_attitude_sp();
 3.   cnt_attitude_pid();
 4.   cnt_altitude_pid();
 5. }
 6.
 7. void cnt_attitude_sp() {
 8.   pid_roll_setpoint = 0;
 9.   if (remote_channel[1] > 1501) pid_roll_setpoint = remote_channel[1] - 1501;
10.   else if (remote_channel[1] < 1499) pid_roll_setpoint = remote_channel[1] - 1499;
11.
12.   pid_roll_setpoint -= roll_level_adjust;
13.   pid_roll_setpoint /= 3.0;
14.
15.   pid_pitch_setpoint = 0;
16.   if (remote_channel[2] > 1501) pid_pitch_setpoint = remote_channel[2] - 1501;
17.   else if (remote_channel[2] < 1499) pid_pitch_setpoint = remote_channel[2] - 1499;
18.
19.   pid_pitch_setpoint -= pitch_level_adjust;
20.   pid_pitch_setpoint /= 3.0;
21.
22.   pid_yaw_setpoint = 0;
23.   if (remote_channel[3] > 1050) {
24.     if (remote_channel[4] > 1550) pid_yaw_setpoint = (remote_channel[4] - 1550) /
3.0;
25.     else if (remote_channel[4] < 1450) pid_yaw_setpoint = (remote_channel[4] - 1450)
/ 3.0;
26.   }
27. }
28.
29. void cnt_attitude_pid() {
30.
31.   //Roll calculations
32.   float pid_i_gain_roll_in;
33.   if (distance > 25) {
34.     pid_i_gain_roll_in = pid_i_gain_roll;
35.   } else {
36.     pid_i_gain_roll_in = 0;
37.     pid_i_mem_roll = 0;
38.   }
39.   pid_error_temp = gyro_roll_input - pid_roll_setpoint;
40.   pid_i_mem_roll += pid_i_gain_roll_in * pid_error_temp;
41.   if (pid_i_mem_roll > pid_max_roll) pid_i_mem_roll = pid_max_roll;
42.   else if (pid_i_mem_roll < pid_max_roll * -1) pid_i_mem_roll = pid_max_roll * -1;
43.
44.   pid_output_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll +
pid_d_gain_roll * (pid_error_temp - pid_last_roll_d_error);
45.   if (pid_output_roll > pid_max_roll) pid_output_roll = pid_max_roll;
46.   else if (pid_output_roll < pid_max_roll * -1) pid_output_roll = pid_max_roll * -1;
47.
48.   pid_last_roll_d_error = pid_error_temp;
49.
50.   //Pitch calculations
51.   float pid_i_gain_pitch_in;
52.   if (distance > 25) {
53.     pid_i_gain_pitch_in = pid_i_gain_pitch;
54.   } else {
55.     pid_i_gain_pitch_in = 0;
56.     pid_i_mem_pitch = 0;
57.   }
58.   pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;
59.   pid_i_mem_pitch += pid_i_gain_pitch_in * pid_error_temp;
60.   if (pid_i_mem_pitch > pid_max_pitch) pid_i_mem_pitch = pid_max_pitch;
61.   else if (pid_i_mem_pitch < pid_max_pitch * -1) pid_i_mem_pitch = pid_max_pitch * -
1;
62.
```

```
 63.   pid_output_pitch = pid_p_gain_pitch * pid_error_temp + pid_i_mem_pitch +
pid_d_gain_pitch * (pid_error_temp - pid_last_pitch_d_error);
 64.   if (pid_output_pitch > pid_max_pitch) pid_output_pitch = pid_max_pitch;
 65.   else if (pid_output_pitch < pid_max_pitch * -1) pid_output_pitch = pid_max_pitch *
-1;
 66.
 67.   pid_last_pitch_d_error = pid_error_temp;
 68.
 69.   //Yaw calculations
 70.   float pid_i_gain_yaw_in;
 71.   if (distance > 25) {
 72.     pid_i_gain_yaw_in = pid_i_gain_yaw;
 73.   } else {
 74.     pid_i_gain_yaw_in = 0;
 75.     pid_i_mem_yaw = 0;
 76.   }
 77.   pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;
 78.   pid_i_mem_yaw += pid_i_gain_yaw_in * pid_error_temp;
 79.   if (pid_i_mem_yaw > pid_max_yaw) pid_i_mem_yaw = pid_max_yaw;
 80.   else if (pid_i_mem_yaw < pid_max_yaw * -1) pid_i_mem_yaw = pid_max_yaw * -1;
 81.
 82.   pid_output_yaw = pid_p_gain_yaw * pid_error_temp + pid_i_mem_yaw + pid_d_gain_yaw *
(pid_error_temp - pid_last_yaw_d_error);
 83.   if (pid_output_yaw > pid_max_yaw) pid_output_yaw = pid_max_yaw;
 84.   else if (pid_output_yaw < pid_max_yaw * -1) pid_output_yaw = pid_max_yaw * -1;
 85.
 86.   pid_last_yaw_d_error = pid_error_temp;
 87. }
 88.
 89. void cnt_altitude_pid() {
 90.
 91.   if (barometer_counter == 1) {
 92.   pid_altitude_input = actual_pressure;
 93.   pid_error_temp = pid_altitude_input - pid_altitude_setpoint;
 94.
 95.   pid_error_gain_altitude = 0;
 96.   if (pid_error_temp > 10 || pid_error_temp < -10) {
 97.     pid_error_gain_altitude = (abs(pid_error_temp) - 10) / 20.0;
 98.     if (pid_error_gain_altitude > 3) pid_error_gain_altitude = 3;
 99.   }
100.
101.   pid_i_mem_altitude += (pid_i_gain_altitude / 100.0) * pid_error_temp;
102.   if (pid_i_mem_altitude > pid_max_altitude) pid_i_mem_altitude = pid_max_altitude;
103.   else if (pid_i_mem_altitude < pid_max_altitude * -1) pid_i_mem_altitude =
pid_max_altitude * -1;
104.   pid_output_altitude = (pid_p_gain_altitude + pid_error_gain_altitude) *
pid_error_temp + pid_i_mem_altitude + pid_d_gain_altitude * parachute_throttle;
105.   if (pid_output_altitude > pid_max_altitude) pid_output_altitude = pid_max_altitude;
106.   else if (pid_output_altitude < pid_max_altitude * -1) pid_output_altitude =
pid_max_altitude * -1;
107.   }
108. }
109.
```

actuators.ino

```
 1. void actuators() {
 2.   act_esc_outputs();
 3.   act_esc_PWM();
 4.   act_us_pulse();
 5. }
 6.
 7. void act_esc_outputs() {
 8.   if (fm == FM_stable) {
 9.     if (throttle > 1800) throttle = 1800;
10.     esc_1 = throttle - pid_output_pitch - pid_output_roll - pid_output_yaw;
11.     esc_2 = throttle + pid_output_pitch - pid_output_roll + pid_output_yaw;
12.     esc_3 = throttle + pid_output_pitch + pid_output_roll - pid_output_yaw;
13.     esc_4 = throttle - pid_output_pitch + pid_output_roll + pid_output_yaw;
14.   }
15.
16.   else if (fm == FM_alt_hold){
17.
18.     if (throttle > 1800) throttle = 1800;
19.     esc_1 = throttle + pid_output_altitude - pid_output_pitch - pid_output_roll -
pid_output_yaw;
20.     esc_2 = throttle + pid_output_altitude + pid_output_pitch - pid_output_roll +
pid_output_yaw;
21.     esc_3 = throttle + pid_output_altitude + pid_output_pitch + pid_output_roll -
pid_output_yaw;
22.     esc_4 = throttle + pid_output_altitude - pid_output_pitch + pid_output_roll +
pid_output_yaw;
23.   }
24.
25.   else {
26.     esc_1 = 1000;
27.     esc_2 = 1000;
28.     esc_3 = 1000;
29.     esc_4 = 1000;
30.   }
31.
32.   if (esc_1 < 1000) esc_1 = 950;
33.   if (esc_2 < 1000) esc_2 = 950;
34.   if (esc_3 < 1000) esc_3 = 950;
35.   if (esc_4 < 1000) esc_4 = 950;
36.
37.   if (esc_1 > 2000) esc_1 = 2000;
38.   if (esc_2 > 2000) esc_2 = 2000;
39.   if (esc_3 > 2000) esc_3 = 2000;
40.   if (esc_4 > 2000) esc_4 = 2000;
41. }
42.
43. void act_esc_PWM(){
44.   TIM_M1_M2->setCaptureCompare(channel_motor1, esc_1, MICROSEC_COMPARE_FORMAT);
45.   TIM_M1_M2->setCaptureCompare(channel_motor2, esc_2, MICROSEC_COMPARE_FORMAT);
46.   TIM_M3_M4->setCaptureCompare(channel_motor3, esc_3, MICROSEC_COMPARE_FORMAT);
47.   TIM_M3_M4->setCaptureCompare(channel_motor4, esc_4, MICROSEC_COMPARE_FORMAT);
48. }
49.
50. void act_us_pulse() {
51.   if (micros() - sent_last_pulse > 7500) {
52.     sent_last_pulse = micros();
53.     digitalWrite(trigger_pin, HIGH);
54.     delayMicroseconds(10);
55.     digitalWrite(trigger_pin, LOW);
56.     pulse_sent = true;
57.   }
58. }
59.
```