Diseño e implementación de un planeador VTOL (*Vertical Take-Off and Landing*) autónomo para distribución de paquetería

Universidad de **Valladolid**

Escuela Técnica Superior de **Ingenieros** de **Telecomunicación**

Máster en **Ingeniería** de **Telecomunicación**

Desarrollado por:

Oscar González González

Bajo la tutela de:

Martín Jaraíz Maldonado

15 de junio de 2024

RESUMEN.

El presente documento recoge de forma práctica el enfoque, los procesos y la línea de desarrollo empleada frente a la propuesta de implementación de un VTOL, enfocado, como fin último, en el reparto de paquetería, presentando una configuración distinta en la estructura y gestión de la movilidad del dispositivo.

Para ello, se analizarán las necesidades principales de la aeronave, estableciendo y discutiendo los elementos *hardware*, exponiendo los algoritmos empleados para la obtención de datos y procesamiento del *software* y presentando distintas técnicas para la creación del diseño y la realización de pruebas reales.

Se expondrán los análisis y las validaciones prácticas necesarias para la implementación final de un prototipo viable y se enfocará su construcción de forma escalable para su utilización en el reparto de paquetería.

ABSTRACT.

This document presents in a practical way the approach, processes, and development line used for the proposed implementation of a VTOL focused, as an ultimate objective, on the delivery of parcels, employing a different configuration in the structure and management of the mobility of the device.

The main needs of the aircraft will be analyzed, establishing and discussing the hardware elements, exposing the algorithms used for data collection and software processing, and presenting different techniques for developing design patterns and the realization of real tests.

The practical analysis and validations necessary for the final implementation of a viable prototype will be presented, focusing its construction in a scalable way for its use in parcel delivery.

PALABRAS CLAVE.

VTOL, dispositivo, paquetes, electrónica, desarrollo.

KEY WORDS.

VTOL, device, parcels, electronics, develop.

ÍNDICE

1. INTRODUCCIÓN

1.1 Contextualización histórica

A lo largo de los años, el ser humano ha tratado de crear todo tipo de vehículos y dispositivos orientados al desplazamiento. Uno de los hitos más notables, ha sido la implementación de vehículos que pudieran desplazarse por el medio aéreo.

Las primeras ideas que podrían relacionarse de forma directa con la aeronáutica, y más concretamente con los VTOL (*Vertical Take-Off and Landing*), surgen entorno al siglo quince, de la mano de Leonardo da Vinci, que expuso los primeros bocetos de su "tornillo helicoidal", donde se mostraba un primer intento de un VTOL que, girando sobre sí mismo actuaría como un tornillo, desplazando el aire hacia abajo, consiguiendo así dicho despegue vertical. Años más tarde, se ha verificado cómo ese tipo de "hélice" era viable [1].

Posteriormente, otro de los avances más transcendentes en materia de VTOL surge en 1783 con la invención del globo por parte de los hermanos Montgolfier y Charles y Robert [2]. Con la utilización de los primeros globos, comenzó a surgir el término UAV (*Unmanned Aerial Vehicle*), cuyas percepciones iniciales se apreciaron en el entorno bélico, cuando en 1849 la artillería austriaca inventó el globo bomba con la finalidad de atacar Venecia [3], fue utilizado, a pesar de su fracaso.

La aparición de los primeros globos incentivó el crecimiento de los dirigibles, destacando el primer viaje de ida y vuelta, aterrizando en el mismo lugar de despegue con *La France*, un dirigible alimentado con un motor eléctrico a batería, encargado de mover las hélices, el primer vuelo de este dirigible se produjo el 9 de agosto de 1884 [4], la creación de estos dirigibles terminaría culminando con la invención del Zeppelin a finales del siglo diecinueve.

En 1898 comenzaron a forjarse los primeros sistemas de radiocontrol destacando la demostración pública de Nikola Tesla en Madison Square Garden, donde controlaba un pequeño barco mediante enlaces de radio [3], lo cual impulsó las pruebas de vehículos no tripulados.

En 1903 se produce el primer vuelo tripulado en un avión a motor, por parte de los hermanos Wright, tratándose de un pequeño vuelo de pruebas de unos pocos segundos. Para finales de 1905, su tercera versión de avión motorizado tripulado consiguió un vuelo

de 39 minutos, cubriendo 39,2 kilómetros de distancia [5]. Este último avance dio paso a desarrollos respecto a los aviones de una y dos plazas, que recibieron importantes mejoras como consecuencia de utilizarse en el ámbito militar durante la primera y la segunda guerra mundial.

Esta nueva invención, junto con las innovaciones militares, propiciaron que en 1935 se desarrollase el primer dron moderno a manos de las fuerzas aéreas británicas, bautizado como "Queen Bee", creado con la finalidad de realizar pruebas de control de vuelo baratas para pilotos en formación [3][6], se trataba de un blanco aéreo no tripulado, radio control.

Este último modelo de vehículo aéreo comienza a encajar en la definición actual del término dron, definido por la RAE (Real Academia Española) como "Aeronave no tripulada"[7] y como "Avión o barco sin tripulación guiado por control remoto o computadoras a bordo" en otros diccionarios angloparlantes [8].

En marzo de 1935 se hace efectiva la primera patente de lo que hoy día conocemos como helicóptero, definido en la patente como "Avión de elevación directa", conocido en la actualidad como un tipo de VTOL [9]. En septiembre de 1939 se realizó la primera prueba práctica del artefacto en Stratford, Connecticut [10]. Esta patente dio paso al diseño posterior de diversos tipos de VTOL, tanto en el ámbito empresarial como militar.

Como consecuencia de los avances en el desarrollo de transistores, comenzaron a surgir aviones de radiocontrol en 1960, destinados al uso recreativo [6]. En los años posteriores se siguieron sucediendo avances militares, pero fue en 2006 cuando dieron comienzo los primeros permisos legales para la utilización de drones en Estados Unidos en el ámbito civil [11][6]. A partir de esta fecha se dio paso a un auge elevado de drones radiocontrol y UAV orientados al ocio, así como destinados a labores de reconocimiento y rescate por parte de cuerpos estatales sanitarios, de bomberos y forestales.

En 2009 comienzan a hacerse notar los primeros proyectos exitosos de vehículos VTOL para el transporte de personas, de la mano de la *startup* china Terrafugia [12], afianzándose así su término en contraposición a la concepción popular de dron como una estructura formada por cuatro motores, aportando una definición más genérica de dispositivo aéreo.

En torno a 2013, algunas de las compañías más importantes de reparto, paquetería y tecnológicas, tales como FedEx, UPS, Amazon, Google o Uber comenzaron a interesarse por las ventajas de emplear drones, invirtiendo en investigación y desarrollo [3][13].

A lo largo de los siguientes años, estas inversiones en paquetería comenzaron a ponerse en práctica por parte de algunas empresas, obteniendo los permisos pertinentes [14]. También se observaron ejemplos de utilización en distribución durante la pandemia COVID-19, con la finalidad de repartir vacunas [15].

A pesar de que han surgido ocasiones puntuales de utilización, es a partir de 2022 cuando algunas de estas tecnologías de distribución de paquetería comenzarán a ponerse en práctica en grandes áreas, como es el caso de los anuncios de la empresa Amazon, que en junio de 2022 notificó que comenzaría a enviar paquetes mediante drones en Lockeford, California [16].

Teniendo en cuenta el contexto en el que nos encontramos actualmente, y habiendo aportado una pequeña perspectiva histórica, se observa como muchas empresas tratan de desarrollar todo tipo de drones y dispositivos VTOL.

Es este auge el que impulsa la iniciativa de implementación. En este documento se tratará de llevar a cabo un proceso de creación de prototipado, documentación y comprensión de los principios técnicos necesarios para la implementación. Para ello se buscarán nuevas configuraciones de dispositivos VTOL, con la finalidad de afrontar el desarrollo desde cero de uno de estos equipos, de forma similar a cómo podría planificarse en alguna de las anteriormente citadas empresas, en el ámbito de investigación y pruebas.

1.2 Objetivos

Este documento tiene como objetivo principal el desarrollo, implementación y estudio de la electrónica y el *software* que conforman los dispositivos de vuelo, en concreto los VTOL, orientados al área de distribución de paquetería. Para ello, se centrarán los esfuerzos en la creación física de un VTOL, cuyo desarrollo escalable permita desempeñar acciones asociadas a la recogida y entrega de paquetes de forma autónoma, al cual se le incorporarán múltiples características adicionales relacionadas con la eficiencia de cumplir su principal propósito.

Las principales áreas bajo estudio que permitirán alcanzar el objetivo fijado se enumeran a continuación:

- Comprender el funcionamiento de dispositivos que cuentan con microcontroladores embebidos, así como esclarecer sus procesos de puesta en marcha, tales que obtención de *drivers*, inclusión de bibliotecas compatibles o estudio de su núcleo central (microcontrolador).
- Obtener experiencia respecto a la utilización de distintos dipos de sensores en proyectos específicos, permitiendo una interconexión común que consiga aglomerar sus capacidades para centrarse en un fin único, aportando una mayor precisión.
- Estudiar y comprender algunos aspectos físico-técnicos relacionados con el campo aeronáutico y la electrónica asociada para el control de vuelo, empleando para ello mecanismos como PID (*Proportional Integral Derivative controller*) y su funcionamiento asociado con ESCs (*Electronic Speed Controllers*).
- Agrupar cada una de las características de los elementos individuales en un sistema realimentado que permita funcionar con suavidad y eficacia solucionando las condiciones externas.
- Adquirir nuevos conocimientos relacionados con el campo de las comunicaciones inalámbricas, pequeñas nociones acerca de la física de fluidos (vuelo) y la creación de sistemas que funcionen sin supervisión.
- Desarrollar nuevas habilidades en la creación de productos desde cero, empleando materiales y herramientas para la creación de estructuras, anclajes, soportes y piezas que permitan tomar ventaja en futuros proyectos de implementación física.
- Utilización de herramientas de diseño asistido por ordenador CAD (*Computer Aided Design*) para la generación de estructuras, soportes y piezas mecánicas que permitan integrar los componentes electrónicos en un único dispositivo funcional.
- Investigar las propiedades físicas necesarias para la obtención de datos, teniendo como objetivo la monitorización y actuación ante diversas situaciones físicas externas.
- Obtener una mayor comprensión frente a la brecha teórico-práctica, aplicando soluciones de ingeniería que permitan obtener condiciones óptimas de funcionamiento ante los diversos compromisos físico-electrónicos.

Cabe destacar que, a pesar de que las bases del proyecto implican una estrecha relación con el entorno aeronáutico, los objetivos principales se basan en las materias

relacionadas con el entorno de las telecomunicaciones, con lo cual, algunos de los problemas asociados al desarrollo del VTOL que traten sobre mecánica de fluidos, estructuras o distribución de pesos serán tratados someramente u obviados, resolviendo algunos aspectos básicos, pero sin detallar con profundidad los métodos matemáticos inmiscuidos en el proceso, en temas relacionados con la aerodinámica.

Continuando con la descripción de los objetivos, se relatará, a lo largo de este documento, los aspectos científico-técnicos asociados al proceso de implementación. Además de esto se desarrollará físicamente un prototipo del producto citado, explicando cada una de sus características.

Finalmente hay que indicar que, el proyecto será enfocado desde el punto de vista de desarrollo de un nuevo producto, similar a cómo actuaría un departamento de I+D, teniendo como fin último la implementación de un prototipo funcional que cumpla la mayor parte de requisitos. En el caso de no poderse llevar a la práctica bien por la limitación temporal del proyecto o por otros aspectos, se expondrán las pruebas llevadas a cabo, así como las conclusiones obtenidas, pero el desarrollo se ceñirá para conseguir una implementación escalable al fin último.

1.3 Condiciones de partida

Inicialmente deberán especificarse cuáles serán las premisas específicas que determinarán las características del VTOL.

Con la finalidad de añadir un nuevo grado de innovación, tratando de aportar características nuevas respecto a los VTOL más comunes, se definen a continuación una serie de requisitos que expondrán el marco de trabajo y las cualidades necesarias del dispositivo:

- El dispositivo deberá contar con menos de cuatro motores, alejándose así de la configuración común de cuadricóptero, aportando complejidad y permitiendo afrontar nuevos problemas surgidos como consecuencia de contar con un motor menos que en la configuración típica de la mayoría de los drones actuales.
- El VTOL deberá implementar dos servomotores para paliar algunos de los posibles problemas asociados a la falta de un motor. De este modo se aboga por una innovación añadida, frente a otros desarrollos en los que únicamente se

emplea un servo motor para equilibrar los giros producidos por contar con un número impar de motores con hélice, e.g. un tricóptero con un único servomotor.

- Los motores delanteros deberán poder situarse en el plano de la estructura, permitiendo en el desarrollo implementar la característica de planeo con la finalidad de poder prolongar el tiempo de vuelo, pudiendo así recorrer distancias mayores sin elevar significativamente el consumo de potencia de los motores.
- El marco de actuación estará orientado a la distribución de paquetería, siendo necesario motores que permitan elevar no solo la estructura del propio VTOL, sino también otros elementos que añadan más peso al dispositivo.

Una vez expuestas las restricciones iniciales se dará paso a la argumentación de la configuración elegida.

Se ha decidido implementar un tricóptero, como consecuencia de que la reducción de motores a dos o menos supondría incorporar estructuras mecánicas que aportaran una mayor aerodinámica al dispositivo. Dado que el proyecto se desarrolla en el ámbito de las telecomunicaciones, la utilización de tres motores permitirá un despegue vertical y un vuelo sin la incorporación de ningún tipo de alerones, flaps u otros elementos más propios del ámbito aeronáutico. Además, se agilizará el desarrollo de las primeras pruebas.

Los dos servos necesarios se incorporarán en la parte delantera con la finalidad de evitar la rotación del dispositivo, al no contar con cuatro motores y no ceñirse a la configuración de los conocidos helicópteros, donde se cuenta con un motor que evita dicho giro.

La incorporación de estos servos en la parte delantera permitirá que posteriormente se puedan girar los motores a 90 grados, permitiendo que entre en juego una posible sección estructural que permitirá el planeo.

Finalmente, se argumenta que se tratará de ceñirse lo máximo posible a la utilización de los materiales disponibles en el laboratorio, teniendo en cuenta que la incorporación de nuevos elementos supondría un coste añadido al desarrollo.

2. ESTUDIO GENERAL DEL SISTEMA

Esta sección del documento tendrá la finalidad de aportar una perspectiva global sobre cada uno de los módulos asociados a la implementación y desarrollo del prototipo físico, en lo relativo a parte lógico-electrónica. Para ello, se aporta, en la ***Figura 1***, un esquema de bloques que servirá para describir la lógica principal del dispositivo electrónico.

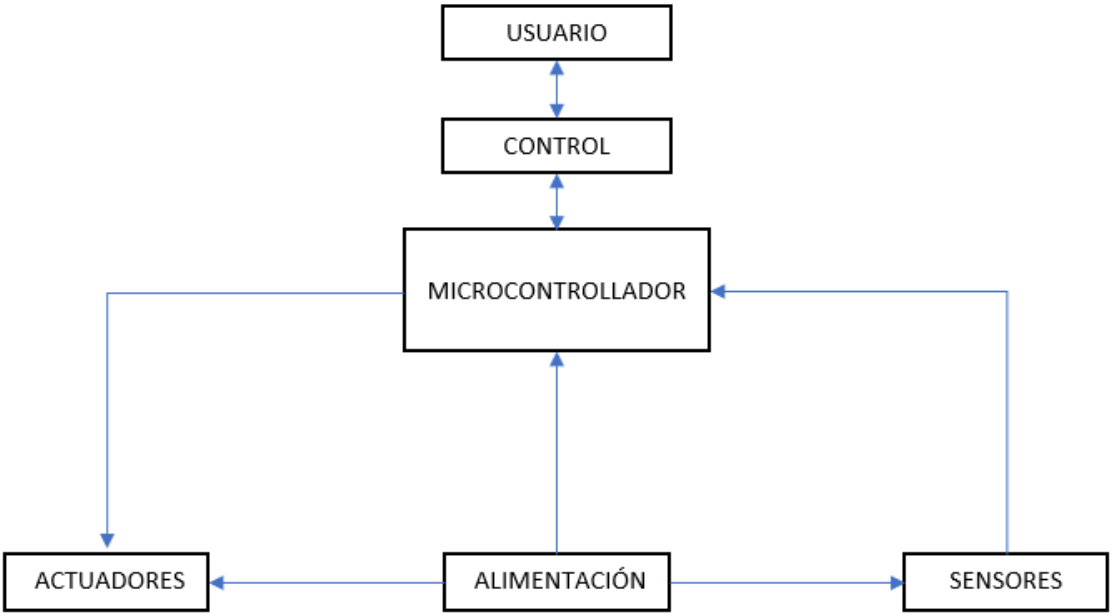


Figura 1. *Esquema de bloques que refleja la descripción principal lógico-electrónica del sistema.*

En la ***Figura 1*** pueden observarse la existencia de seis bloques lógico-electrónicos principales, en los que se fundamentará la implementación final.

A continuación, pasarán a detallarse las principales funcionalidades y cometidos de cada uno de los bloques que componen el esquema anterior.

2.1 Usuario

El sistema deberá contar con entradas de usuario que faciliten al conjunto de elementos conocer aspectos relacionados con la incorporación y recogida de paquetes. Si bien este cometido podría realizarse por medio de la incorporación de sensores, debido a las diferencias de dimensiones, peso y otras características relacionadas con la incorporación manual de paquetes al dispositivo de vuelo, se deja en manos del usuario el aportar las indicaciones necesarias para indicar que el paquete ha sido sustraído o añadido al dispositivo.

De esta forma, se descargará al sistema de la tarea relacionada con la detección de paquetes, confiando en el factor humano.

Por otro lado, tal y como nos indica el esquema de la ***Figura 1***, el usuario recibirá cierta realimentación, confirmando que el sistema conoce las indicaciones ofrecidas por el mismo con la finalidad de que, en el supuesto de que surja alguna situación de error o que se den excepciones no recogidas en el funcionamiento del programa, el mismo usuario pueda solventar por medio de entradas físicas tales inconvenientes, asegurando una comunicación óptima entre el sistema y el usuario.

El usuario técnico podrá realizar iteraciones con el módulo de control, de esta forma podrá incorporar instrucciones de movimiento en caso de establecerse un control manual durante las pruebas del prototipado, o en caso de fallo. Además, podrá recibir realimentación técnica sobre el estado de ciertos valores de vuelo.

Finalmente, cabe destacar que, en la interpretación del esquema anteriormente aportado, se considera usuario a cualquier persona que interactúe de manera directa con el prototipo implementado, ya sea el receptor, el emisor del paquete o personal técnico.

2.2 Control

El módulo de control será el encargado de ofrecer realimentación técnica sobre el estado de algunas de las condiciones de vuelo al usuario técnico.

Además, se añadirá en este elemento el control manual empleado durante el desarrollo de las pruebas, pudiendo así controlar el dispositivo VTOL de forma manual para controlar aspectos como el despegue, los cambios de dirección o la configuración de distintos modos de vuelo, así como las preconfiguraciones iniciales en caso del modo autónomo.

2.3 Microcontrolador

El microcontrolador será la piedra angular en la que se fundamente el sistema y tendrá el propósito de gestionar cada pedazo de información proveniente de las entradas de usuario y sensores, pudiendo aportar respuestas gracias a los datos recibidos, a través de sus diferentes actuadores.

Trabajará como centro neurálgico del sistema, interpretando las sensaciones recibidas del entorno, procesando los datos y actuando en consecuencia. Además, deberá de estar preparado para asegurar el funcionamiento, realimentando información al usuario en caso de ocurrir errores relacionados con cualquiera de los componentes que conforman en dispositivo.

Este componente deberá cumplir una serie de requisitos referentes con la gestión de la alimentación de los sensores, tener la capacidad de aportar diferentes tipos de señales, adaptarse a protocolos de comunicación digital, contar con una capacidad de cómputo suficiente como para poder aportar una rápida respuesta a los actuadores, así como permitir la ampliación de características del producto en el largo y medio plazo (escalabilidad).

2.4 Actuadores

En lo referente a los actuadores, tratarán de aportar respuestas, en función de las indicaciones del microcontrolador. Al tratarse de un dispositivo de vuelo, los principales actuadores estarán basados en motores que permitirán llevar a cabo la sustentación del dispositivo en el aire. Con la finalidad de cumplir esta premisa, dichos motores tendrán que cumplir una serie de características técnicas relacionadas con parámetros físicos tales que el torque, el consumo de potencia o la velocidad de giro. Debido a los requisitos de potencia de dichos motores, algunos de los actuadores, necesitarán poseer conexiones directas a la fuente de alimentación, asegurando en todo momento que cuentan con la capacidad eléctrica necesaria para aportar un buen funcionamiento.

Por otro lado, se contará con distintos actuadores que permitan aportar una realimentación al usuario, ya sea a través de pantallas, indicadores luminosos u otro tipo de implementaciones.

2.5 Alimentación

Al tratarse de un dispositivo que ha de precisar con la característica de movilidad, la principal fuente de alimentación tendrá que ser portátil, en el caso que ocupa, se trabajará con una batería.

Dicha batería deberá de estar en sintonía con los requisitos necesarios de funcionamiento en los actuadores que precisen de una mayor cantidad de potencia, tal y cómo se comentaba en la sección 2.4. Como consecuencia de la relativamente elevada potencia de algunos de los actuadores, la batería deberá poseer un voltaje de funcionamiento elevado, así como contar con una capacidad que permita realizar vuelos prolongados y finalmente disponer de la característica de aportar la suficiente potencia en un corto periodo de tiempo sin suponer un riesgo para la salud de esta.

En lo relativo a las tensiones, se impone de manera contundente la necesidad de incorporar elementos de adaptación de voltajes, permitiendo tanto que los actuadores funcionen con corriente alterna (motores de vuelo) como que el microcontrolador pueda alimentarse de manera correcta para después aportar la energía necesaria a los distintos sensores incorporados.

2.6 Sensores

Los sensores serán los encargados de comunicar al microcontrolador los datos relativos al medio, permitiendo equilibrar el prototipo, efectuar un vuelo suave y geolocalizarse.

Dichos sensores, por tanto, poseerán conexiones directas con el microcontrolador que permitan llevar a buen puerto su principal cometido.

Como se explicó en la sección relacionada con la alimentación, los sensores estarán alimentados por el microcontrolador a través de sus puertos de alimentación en la medida de lo posible. De este modo no será necesario incorporar dispositivos adicionales que adapten los valores de tensión y corriente desde la fuente de alimentación a los distintos tipos de sensores. Esta práctica resulta viable debido a que, en la mayoría de los casos, los valores de alimentación de los sensores suelen estar medianamente “estandarizados” comprendiendo voltajes de funcionamiento normalmente bajos.

La incorporación de sensores dependerá de la existencia de bibliotecas compatibles, facilitando un desarrollo ágil a nivel de *software*, que permita introducir de forma modular

nuevas mejoras, sin descartar la utilización de escritura directa en binario a los registros para obtener una mayor adaptación.

Por último, cabe destacar que el microcontrolador deberá contar con múltiples canales de comunicaciones, asegurando que en un futuro se podrán incorporar cierto número de nuevos sensores, sin que esto suponga un problema de cara a la cantidad de puertos disponibles en el centro neurálgico del dispositivo.

2.7 Otras consideraciones

Tras haber tratado de forma general el desarrollo lógico electrónico, resumido de forma visual en la ***Figura 1***, cabe destacar que el proyecto deberá contar con otras aportaciones mecánicas que determinarán de forma seria las configuraciones, distribución y localización de los distintos elementos.

Será necesario el desarrollo de una mecánica que permita ensamblar todos los componentes inmiscuidos en el sistema, además de tener que contar con una estructura o fuselaje que aporte cierta estabilidad durante las fases de vuelo.

Dicho esto, se recalca que este proyecto tiene como objetivo realizar una implementación física de un producto concebido como prototipo, es por esto por lo que, todas las aportaciones mecánicas se realizarán de forma artesanal, a pesar de que puedan llegar a diseñarse piezas específicas mediante herramientas CAD.

3. HARDWARE

Esta sección tratará de arrojar luz en los aspectos relativos a las decisiones tomadas de cara a la implementación física y elección de los componentes. Es por esto por lo que, se analizarán las características técnicas de los mismos, esclareciendo sus especificaciones más importantes y aportando aclaraciones matemáticas sobre los componentes, así como desgranando sus cualidades electrónicas principales en base a las necesidades requeridas.

También se realizarán explicaciones sobre los procesos inmiscuidos en la creación del prototipo, aportando detalles acerca de los pasos tomados durante la elaboración de la construcción electrónica, pruebas de funcionamiento realizadas, componentes descartados y otras aportaciones relacionadas con los procesos de prototipado electrónico.

Una vez esclarecido el funcionamiento general del sistema mediante diagramas de bloques conceptuales, se pasarán a explicar de forma técnica y detallada, el interior de cada uno de los bloques enunciados anteriormente.

3.1 Perspectiva de funcionamiento

Tal y como se expuso durante la sección de introducción de este documento, se tratará de estudiar una configuración de VTOL en concreto. Para ello, se han solicitado los siguientes requisitos de implementación:

- Utilización de una configuración distinta a los drones comunes de consumo, evitando para ello la utilización de cuatro motores.
- Incorporación de servo motores que permitan modificar el ángulo de sustentación de los motores que generarán el giro para la elevación.
- Utilización de un fuselaje escalable, orientado a la implementación de acciones de planeo, prolongando así el vuelo útil del dispositivo.
- Incorporación de acciones autónomas por parte de la aeronave, siendo capaz de realizar ciertas funcionalidades sin ningún tipo de supervisión.
- Capacidad para portar paquetes, mediante cualquier método.

Expuestos los requisitos solicitados, se ha tratado de implementar un VTOL que posea únicamente tres motores, de los cuales se podrá modificar el ángulo de sustentación en dos de ellos, permitiendo aportar nuevas funcionalidades y añadiendo estabilidad al sistema. En cuanto al resto de requisitos, serán debatidos a lo largo de la sección de

hardware de este documento, dejando la implementación de la estructura en manos de secciones posteriores.

3.2 Componentes

En lo referente a la elección de componentes cabe destacar que, las elecciones *hardware* se han visto acotadas por los materiales disponibles en el laboratorio de la Universidad de Valladolid, proporcionados por esta en su mayoría. Las valoraciones expuestas a continuación responden únicamente a las elecciones de componentes dentro de este marco. Si bien es cierto que los materiales proporcionados han sido válidos para el desarrollo del prototipo, podrían existir otros productos que permitieran optimizar aún más las características de la implementación física final.

Esta decisión ha venido marcada principalmente por un motivo económico desde el punto de vista del desarrollo. Al tratarse de un proyecto de ingeniería, el factor de costes supone un papel principal, motivo por el cual se ha tomado este camino, tratando de aportar una implementación funcional con los recursos disponibles. A pesar de esta premisa, en posteriores secciones se aportarán datos económicos relativos a los materiales para poder expresar una idea del precio total del prototipado.

A continuación, pasarán a explicarse en profundidad los componentes empleados y decisiones tomadas sobre los mismos durante el desarrollo del prototipado.

Algunos de los sensores y características expuestas, han sido objeto de estudio con vistas a futuras actualizaciones del proyecto, habiendo realizado pruebas reales con los mismos, sin embargo, no se han implementado de forma práctica en el prototipo final. Este hecho parte de la perspectiva de escalabilidad asociada a la filosofía del proyecto, debido al tiempo disponible y a la gestión de recursos humanos. Sin embargo, se ha realizado un desarrollo que permita la implementación de las citadas funcionalidades.

Por otro lado, algunos de los componentes aquí expuestos han sido probados de forma práctica incluyéndolos a la estructura, para después retirarlos tras visualizar que no cumplían con las cotas de calidad esperadas o debido a regulaciones externas, como se expondrá posteriormente.

3.2.1 Motores

A la hora de llevar a cabo la elección de los motores en base a los materiales disponibles, se ha dispuesto de dos motores diferentes, siendo el motor A2212/13T de 1000KV y el motor DX2205 de 2300KV de la marca Crazepony. La ***Figura 2*** muestra una imagen con los dos motores disponibles para la implementación del VTOL.

Figura 2. *Motores disponibles para el desarrollo del VTOL.*

Presentados ambos motores, se dará paso a una valoración teórica de sus respectivas características, aportando una decisión respecto a la elección de uno de estos componentes.

3.2.1.1 Estudio teórico:

A la hora de llevar a cabo una comprensión genérica de los motores, debemos desgranar sus características físicas asociadas. Los motores se basan principalmente en la ley de Ampère-Maxwell, que explica cómo una variación de corriente eléctrica que circula por un conductor genera un campo electromagnético, esta explicación se resume en la siguiente ecuación [17]:

$$\boldsymbol{\nabla} \times \boldsymbol{B} = \mu_0 * \boldsymbol{J} + \mu_0 * \varepsilon_0 * \frac{\partial \boldsymbol{E}}{\partial t}$$

Donde $\boldsymbol{B}$ representa el vector de inducción magnética, $\boldsymbol{J}$ el vector de densidad de corriente eléctrica, μ_0 la permeabilidad magnética en el vacío, ε_0 la permitividad eléctrica en el vacío y $\boldsymbol{E}$ el vector de intensidad de campo eléctrico. Mientras que el operador nabla indica la naturaleza rotacional de la ecuación.

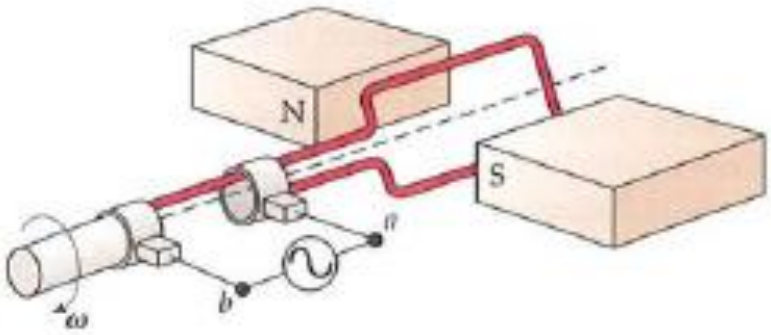


Figura 3. *Ejemplificación gráfica sobre el principio básico de funcionamiento de un motor.* [18]

La ***Figura 3*** aporta una ejemplificación gráfica sobre la ley de Ampère-Maxwell, en la que se muestra el principio básico de funcionamiento de un motor. Al introducir una corriente alterna en las bornas de la bobina, se genera un campo magnético como consecuencia de la anteriormente citada ley. Al encontrarse esta bobina en un campo magnético formado por los imanes, ambos campos magnéticos interactuarán alineándose en función de su polaridad. Si se invierne la polaridad de la corriente alterna en el momento en el que el campo magnético generado se alinea con el campo magnético de los imanes (espira colocada en posición vertical) se producirá una rotación continua, obteniendo el efecto deseado [18].

Conocidos los principios básicos en los que se fundamentan los motores, se pasará a desgranar la física de las principales características específicas de ambos.

En cuanto a las características comunes, nos encontramos con que ambos motores son motores sin escobillas, característica que les aportará mayor durabilidad, evitando así el desgaste de estas debido a la rotación del motor. Las escobillas son necesarias cuando el estátor del motor está formado por la parte imantada. Esta casuística surge de una configuración práctica, no es viable crear un motor cuyo rotor esté formado por la parte bobinada sin escobillas, dado que los cables se enrollarían, la corriente ha de pasarse a través de dichas escobillas haciendo contacto con la parte que rota, produciendo un rozamiento que, debido a la fricción, terminaría desgastando las escobillas y por tanto generando averías en el motor [19].

Por este motivo los motores sin escobillas son ampliamente utilizados, como consecuencia, el estátor estará formado por el bobinado, mientras que el rotor lo compondrán los imanes. Esta configuración está presente en ambos motores.

En la ***Figura 4*** pueden apreciarse las diferencias entre un motor con escobillas y sin escobillas típicos.

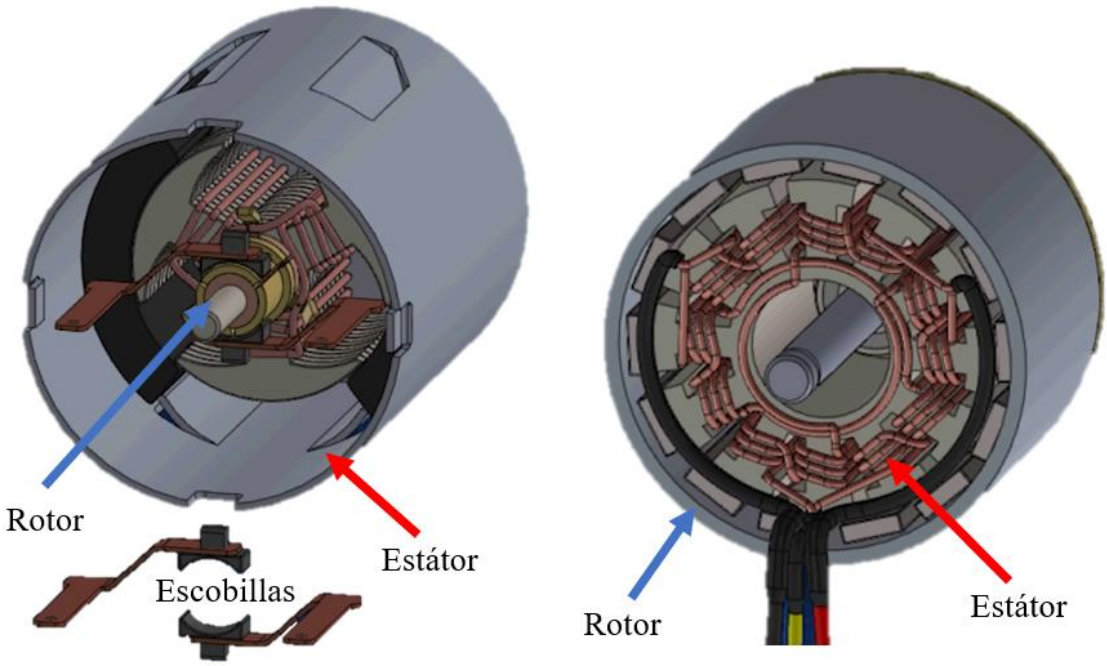


Figura 4. *Motor con escobillas y rotor interno (izquierda), motor sin escobillas y rotor externo (derecha).* [19]

Dado que el estátor está formado por el bobinado, una de las configuraciones más habituales, debido a la posibilidad de implementar un diseño más simple, es la de establecer el rotor en la parte externa.

Existen motores sin escobillas de rotor interno, cuya característica principal es que realizan giros más rápidos, sin embargo, son más utilizados los motores de rotor externo debido a que, aunque giran más lento, producen un torque mayor.

Continuando con las características principales de los motores, será necesario hacer hincapié en la relación entre torque y velocidad. El torque se define como fuerza de rotación, y viene definido por las siguientes ecuaciones [18]:

$$\boldsymbol{\tau} = \boldsymbol{r} \times \boldsymbol{F}$$

$$\tau = \|\boldsymbol{r}\| * \|\boldsymbol{F}\| * sen\theta$$

Donde τ representa el torque, $\boldsymbol{r}$ la posición del vector de fuerza, $\boldsymbol{F}$ el vector de fuerza y θ el ángulo formado entre el vector de fuerza y el radio o vector de posición de fuerza.

Expuesto el torque o momento de fuerza, se expondrá en las siguientes ecuaciones su relación con la velocidad [18]:

$$P = \frac{Trabajo}{Tiempo} = \frac{F * Distancia}{Tiempo} = F * Velocidad$$

$$P = \frac{\tau}{\|\boldsymbol{r}\| * sen\theta} * \omega \rightarrow Si\ \theta = 90º \rightarrow \tau = \frac{P * \|\boldsymbol{r}\|}{\omega}$$

Donde ω representa la velocidad angular. Si contamos con que la potencia y el radio de giro son constantes podremos observar como el torque y la velocidad angular tendrán una relación inversamente proporcional [20]. Esta última conclusión puede verse reflejada en la gráfica de la ***Figura 5.*** Extrapolando estas conclusiones al desarrollo de nuestro prototipo, se tratará de obtener un torque mayor en vez de optar por una mayor velocidad, dado que el PID tendrá que realizar cambios bruscos para equilibrar el dispositivo en el aire, tarea que requiere un mayor momento de fuerza angular.

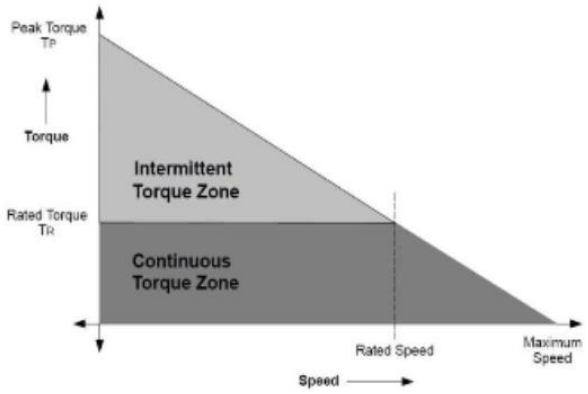


Figura 5. *Características de torque frente a velocidad en BLDC (BrushLess Direct Current).*[20]

A la hora de determinar el torque y la velocidad, influirán una gran cantidad de variables como la propia geometría del motor, las propiedades químicas del imán, la cantidad de ellos, la superficie de estos, su distribución física en el rotor, etc [21][22].

Continuando con las características, podemos apreciar en la ***Figura 2***, cómo ambos motores cuentan con orificios de ventilación en la parte superior. Estos orificios mejoran el flujo de aire del interior de los motores, evitando así que el calor generado por la corriente que atraviesa el bobinado derrita el recubrimiento de esmalte de los cables de cobre, esta relación entre potencia y calor se puede visualizar en las siguientes ecuaciones[18].

$$P(W) = I(A) * V(V) = I^2(A) * R(\Omega)$$

$$Q(J) = m(Kg) * c\left(\frac{J}{Kg * ºC}\right) * \Delta T(ºC)$$

$$P(W) = \frac{E(J)}{\Delta t(s)} \rightarrow I^2 * R = \frac{Q(J)}{\Delta t(s)} \rightarrow \Delta T(ºC) = \frac{I^2(A) * R(\Omega) * \Delta t(s)}{m(Kg) * c\left(\frac{J}{Kg * ºC}\right)}$$

Donde Q es la cantidad de calor en julios, m la masa del material y c la capacidad calorífica del material, dado que en un conductor la masa y la capacidad calorífica del material son constantes, se puede obtener una relación casi directa entre la intensidad y la cantidad de calor generada por el bobinado.

Si observamos nuevamente la ***Figura 2***, nos encontraremos con que los motores poseen tres cables cada uno, esto se debe a que cuentan con una configuración trifásica. Este tipo de configuración utiliza tres corrientes desfasadas de la misma frecuencia, permitiendo obtener una potencia más elevada y, en consecuencia, un rango de RPM (Revoluciones Por Minuto) de mayor magnitud. Estos motores son comúnmente empleados en la industria debido a su capacidad de operar con mayor potencia, además generan menores vibraciones y ser menos ruidosos [23]. La ***Figura 6*** muestra una gráfica en la que puede apreciarse la configuración de corrientes desfasadas en trifásica, siendo todas de la misma amplitud y frecuencia.

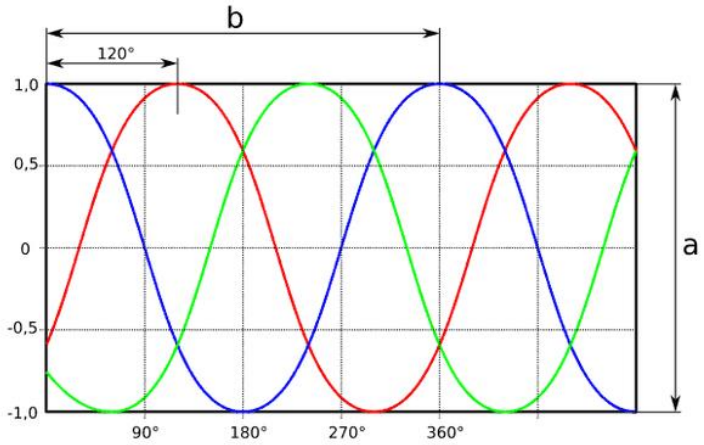


Figura 6. *Gráfica de corrientes empleadas en motores trifásicos.* [23]

La utilización de corrientes trifásicas permite distribuir la potencia de un modo más uniforme, aportando una mayor suavidad en la transición del motor. Además, la anterior

característica produce un par o torque más constante, contribuyendo a la obtención de una mayor eficiencia energética y, en consecuencia, a un menor calentamiento. Finalmente, otorga la capacidad de poseer un rango más alto de velocidades, dado que, al contar con tres fases distintas se mejora la granularidad de las transiciones.

Estos sistemas trifásicos contribuyen a facilitar la alternancia en el giro intercambiando dos de los cables, consiguiendo así invertir el orden de las fases, produciendo una rotación en sentido opuesto.

En última instancia, cabe destacar que, normalmente, la configuración del bobinado implementado suele ser *dLKR*, configuración que se muestra en la ***Figura 7***.

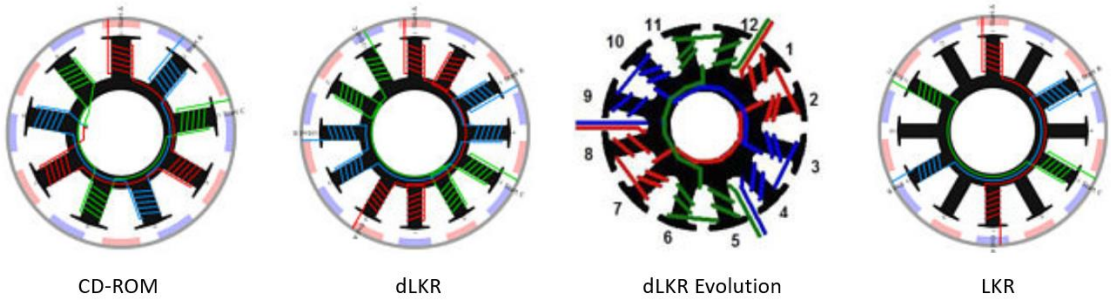


Figura 7. *Configuraciones típicas de esquemas de bobinado.*[24]

Estas configuraciones vienen fomentadas por una mejor organización del cableado, principalmente, y se emplean en función de la configuración de cables deseada o la construcción del motor. En ocasiones las distintas configuraciones son denominadas de diferentes formas por la industria a pesar de que se refieran a disposiciones idénticas en el bobinado.

3.2.1.2 Comparación técnica:

Una vez expuestas las principales características de los motores, encajadas en un marco físico-teórico global, se dará paso a la comparación de sus especificaciones técnicas concretas, con la finalidad de poder determinar sobre el papel qué motor sería el más idóneo para el proyecto.

Dado que ambos motores son ampliamente utilizados, resulta difícil encontrar el fabricante original, como consecuencia de la creación de réplicas e implementaciones similares por parte de competidores. Este hecho dificulta la obtención de hojas de datos

con alto nivel de detalle, para el caso que nos ocupa, se tendrán en cuenta las características expuestas por grandes vendedores, pudiendo obtener así obtener características físicas que ayudarán a realizar una comparación. A pesar de esto, se ha determinado durante las implementaciones prácticas que las características aportadas por los vendedores se ceñían a la realidad.

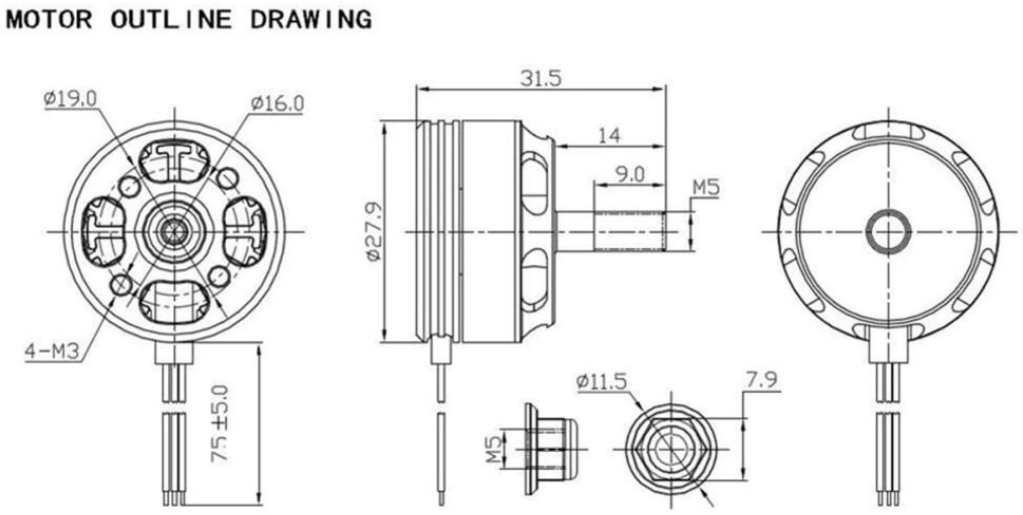
En lo referente al motor A2212/13T de 1000KV, las principales características aportadas por el vendedor son [25]:

- Motor Brushless KV1000, A2212
- Diametro del eje: 3.175mm.
- Corriente: 12A/60s.
- Controlador recomendado: ESC 30A.
- Propelas adecuadas: 8×4.5" 9×4.5" 10×4.5" 10×4.7"
- Ideal para baterías de LiPo de 2 a 3 celdas.
- RPM / V: 1200
- Máx. eficiencia: 82 %
- Máx. actual eficiencia: 7 – 16 A (> 75 %)
- Sin corriente de carga / 10 V: 0.9 A
- Resistencia interna: 150 MOhm
- Material: Aleación de aluminio
- Diámetro de salida del eje: 3.17 mm
- KV: 1000
- Bullet clip de la forma de hélice: 3.17 mm
- Para hélices de 6 mm de diámetro interior
- Dimensiones: 4 cm X 2.8 cm X 2.8 cm

En el caso del motor DX2205 de 2300KV se aporta una imagen más cercana a las hojas de datos convencionales, cuyas características pueden observarse en la ***Figura 8***.

Si nos centramos en las características electrónicas, la primera diferencia observable es que el motor A2212 consume menos corriente que el motor DX2205. Podemos apreciar como la corriente de carga, referente a la corriente que utiliza el motor cuando se está utilizando, es superior en el caso del DX2205. Este hecho, sumado a que las dimensiones de este segundo motor son inferiores, producirá que se genere una mayor cantidad de

calor en su interior debido al efecto Joule [18]. Tal y como se mostraba en las ecuaciones de la sección 3.2.1.1, suponiendo que la resistencia de los bobinados es similar en ambos casos, debido al aumento de corriente se elevará la temperatura del cableado.



Item	KV(rpm/v)	Voltage(v)	Prop	Load Current(A)	Pull(g)	Power(w)	Efficioney (g/w)	Lipo	Weight(g) Approx
DX2205	2300	11.1	5045	19.2	660	213	3.1	2-4S	28
		14.8		27.6	950	408	2.3		

Figura 8. *Características del motor DX2205 de 2300KV.* [26]

Continuando con las características principales, nos encontramos con los KV, esta variable se define como la constante de velocidad del motor y expresa las RPM por voltio, no ha de confundirse con kilovoltios. En el caso del DX2205 observamos que posee 2300KV, mientras que el A2212 cuenta con 1000KV. Esta característica nos muestra, sumado al voltaje máximo admitido cual será la velocidad máxima de rotación de los motores.

Si tenemos en cuenta que en las recomendaciones del motor A2212 se indica una recomendación 2-3S en baterías, esto determinará que su tensión estará en torno a 11.1V (esta afirmación se discutirá con mayor profundidad en la sección de alimentación) de entrada, en el caso del DX2205 se indica un voltaje 11.1-14.8V. Tomando que nuestra fuente de voltaje de entrada es común, fijada en 11.1V para ambos casos tendremos respectivamente unas potencias de:

$$A2212 \rightarrow 11.1V * 12A = 133.2W$$
$$DX2205 \rightarrow 11.1V * 19.2A = 213.12W$$

Si nos fijamos en las conclusiones extraídas durante el estudio teórico, en la ***Figura 5*** podemos observar cómo el torque y la velocidad suponen un compromiso, siendo inversamente proporcionales. Retomando las ecuaciones de torque y velocidad:

$$\tau = \frac{P * \|\boldsymbol{r}\|}{\omega}$$

$$A2212 \rightarrow \tau = \frac{133.2W * \|\boldsymbol{r}\|}{1000RPM/V * 11.1V * \frac{2\pi\ rad}{60s}} = 0.1145 * \|\boldsymbol{r}\|$$

$$DX2205 \rightarrow \tau = \frac{213.12W * \|\boldsymbol{r}\|}{2300RPM/V * 11.1V * \frac{2\pi\ rad}{60s}} = 0.0797 * \|\boldsymbol{r}\|$$

Observamos que, a pesar de que la potencia del motor DX2205 será mayor, su torque será inferior teniendo en cuenta que utilicemos propelas idénticas, verificando así que a mayor velocidad de rotación menor será el torque.

Otra característica principal relacionada con el torque es el *pull*, este valor se refiere a la fuerza de tracción o empuje que el motor puede generar, en el caso del motor DX2205 se indica que está entre 3.1 g/V y 2.1 g/V mostrando que a mayor potencia menor será la fuerza de tracción del motor. Si realizamos los cálculos de la eficiencia, teniendo en cuenta el *pull* calculado en función de las propelas, podemos realizar una pequeña estimación de la eficiencia [26][27][28]:

$$\eta = \frac{\eta_M}{\eta_E} = \frac{\tau * RPM}{V * I}$$

$$pull = \frac{3.1g_f}{V} \rightarrow F = \frac{3.1g_f}{V} * 11.1V = 34.41g_f \rightarrow 34.41g_f * \frac{0.009806\text{N}}{g_f} = 0.337N$$

$$\tau = F * \|\boldsymbol{r}\| = 0.337N * 0.127m = 0.042N * m$$

$$\eta = \frac{\tau * RPM}{V * I} = \frac{0.042N * m * \frac{\frac{2300RPM}{V} * 2\pi}{60s} * 11.1V}{213.12W} = 0.5268 \approx 53\%$$

Sabemos que la eficiencia de un motor se determina calculando la relación entre la potencia mecánica $\boldsymbol{\eta_M}$ y la potencia eléctrica $\boldsymbol{\eta_E}$. El *pull* que indica la fuerza de tracción, puede relacionarse con la fuerza física $\boldsymbol{F}$. Si tenemos en cuenta que en los datos de motor DX2205 se emplean unas propelas de 5045, es decir, 50 pulgadas (12,7cm) por 45 pulgadas de paso de la hélice (*pitch*). Con estos datos podremos calcular el torque $\boldsymbol{\tau}$ y en consecuencia realizar una estimación de la eficiencia.

Cabe destacar que este cálculo es una estimación, dado que se está asumiendo que la fuerza aplicada está posicionada en el extremo del brazo del propeler, además de no tener en cuenta el paso de la hélice. Sin embargo, nos permite hacernos una idea sobre la eficiencia del motor, viendo que probablemente sea inferior a la indicada en el motor A2212, donde dicha eficiencia es mayor al 75% para el voltaje de estudio, 11.1V.

En el caso de las hélices del motor A2212 se indica una mayor diversidad de dimensiones que podrían emplearse como óptimas. Como punto final de comprobación vemos que las dimensiones de las propelas indicadas para este último motor son más elevadas, yendo desde las 8 a las 10 pulgadas. Este es otro indicativo de que posee un mayor torque y una mejor eficiencia, puesto que además trabaja con valores de potencia eléctrica inferiores.

3.2.1.3 Implementación práctica

Tras la comparación técnica, hemos podido observar a grandes rasgos como el motor A2212 poseía unos valores inferiores de consumo, mejor eficiencia y gozaba de un mayor torque, soportando hélices de dimensiones superiores.

En la práctica se han probado ambos motores con la finalidad de cerciorarse sobre las características expuestas por los vendedores. En las etapas iniciales del desarrollo, durante la implementación de un controlador PID en un solo eje, se han podido verificar las conclusiones extraídas de forma teórica. La ***Figura 9*** representa una esquematización de la estructura empleada durante estas pruebas.

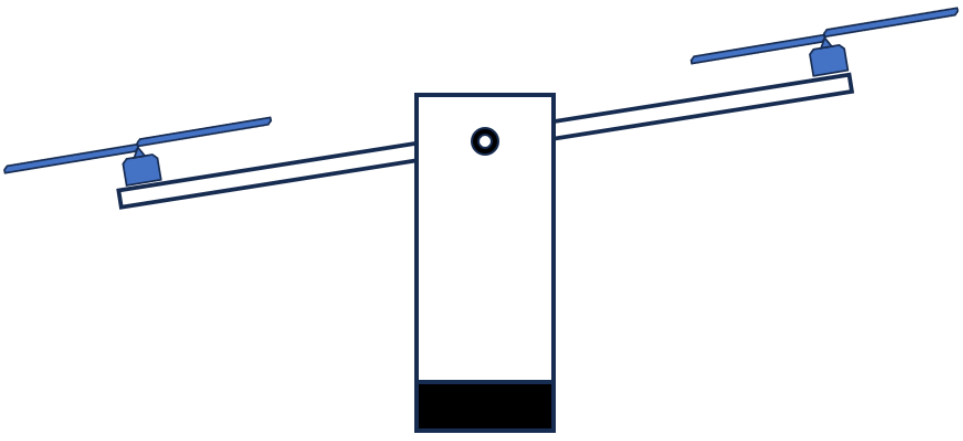


Figura 9. *Representación de la estructura empleada durante las pruebas de los motores.*

La utilización de los motores DX2205 permitían cambios bruscos en las revoluciones, facilitando una mayor granularidad en las velocidades mientras se realizaban los ajustes del PID, equilibrando rápidamente ante cualquier desajuste. Por contrapartida su calentamiento era excesivo, llegando en muchas ocasiones a producir quemaduras a la hora de colocarlo para las posiciones iniciales. A este hecho se le suma la duración inferior de las baterías.

En la práctica, el torque y eficiencia inferiores se manifestaban en grandes variaciones de la velocidad durante cortos periodos de tiempo para intentar equilibrar la estructura.

Respecto a los motores A2212 el calentamiento resultó ser notablemente inferior, pudiendo manipularlos tras haberlos utilizado por cortos periodos de tiempo sin peligro alguno, este calentamiento también se redujo en los ESC. En su parte práctica, empleando el mismo algoritmo para el PID se observaban variaciones mucho más suaves en las revoluciones del motor y permitían una mayor estabilidad en la práctica. Cabe destacar que era necesario aumentar el peso de la base de la estructura al utilizar estos motores para evitar que el giro de las hélices la elevara por completo.

Tras haber realizado el estudio teórico y las pruebas prácticas se ha decidido emplear el motor A2212, debido a sus características de eficiencia y consumo de potencia, las cuales alargarán el tiempo de vuelo y asegurarán la durabilidad del dispositivo evitando sobrecalentamientos que puedan dañar las piezas plásticas o cableado.

Otra de las características clave de este motor es su torque, dado que con muchas menos revoluciones conseguía elevar pesos de mayor magnitud, hecho que asegurará que pueda portar paquetes más grandes o pesados durante su vuelo.

Finalmente, el último factor determinante ha sido la estabilidad, el hecho de que tenga un rango de rpm inferior reduce el rango dinámico de posibles velocidades, provocando en la práctica que las transiciones sean de mayor suavidad. En nuestro caso no queremos realizar acrobacias ni giros bruscos, sino que los paquetes en el dispositivo puedan desplazarse con la mayor resiliencia posible.

3.2.2 ESCs

Siguiendo con los componentes, tras haber realizado un estudio sobre los motores, nos centraremos en la pieza que manejará la logística eléctrica de la actuación de dichos motores. En este caso, se ha tenido acceso a los ESC SIMONK30A de Lighting Hobby, cuya imagen puede apreciarse en la ***Figura 10***. En este caso, ha sido el único componente de este tipo disponible en el laboratorio.

Figura 10. *ESC SIMONK30A de Lighting Hobby.*

Se aportará a continuación un estudio sobre el componente, explicando también su implementación práctica.

3.2.2.1 Estudio teórico

Comenzando con el estudio de los ESC, el primer punto que debemos aclarar es la definición acerca de la funcionalidad principal de este elemento. En la sección 3.2.1 vimos como los motores BLDC empleados utilizaban una corriente trifásica. La ***Figura 11*** muestra la funcionalidad principal de un ESC.

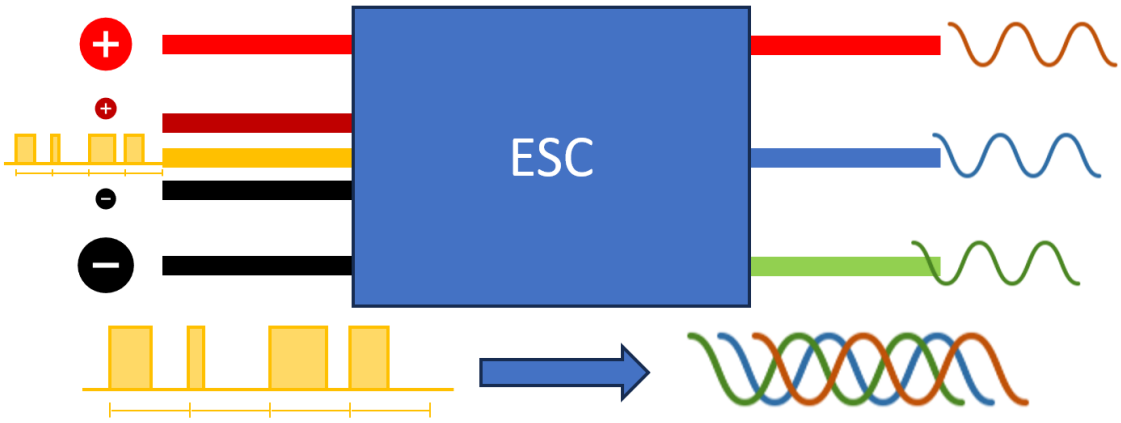


Figura 11. *Descripción esquemática de la funcionalidad de un ESC a nivel de señal.*

En la figura anterior podemos apreciar cómo su principal función es convertir una señal PWM (*Pulse Width Modulation*) en tres señales sinusoidales desfasadas que permitirán el funcionamiento del motor. Ante este funcionamiento surgen cuestiones técnicas relativas al desfase de la señal o la generación de una señal analógica. La ***Figura 12*** muestra el interior del ESC empleado, así como sus componentes principales.

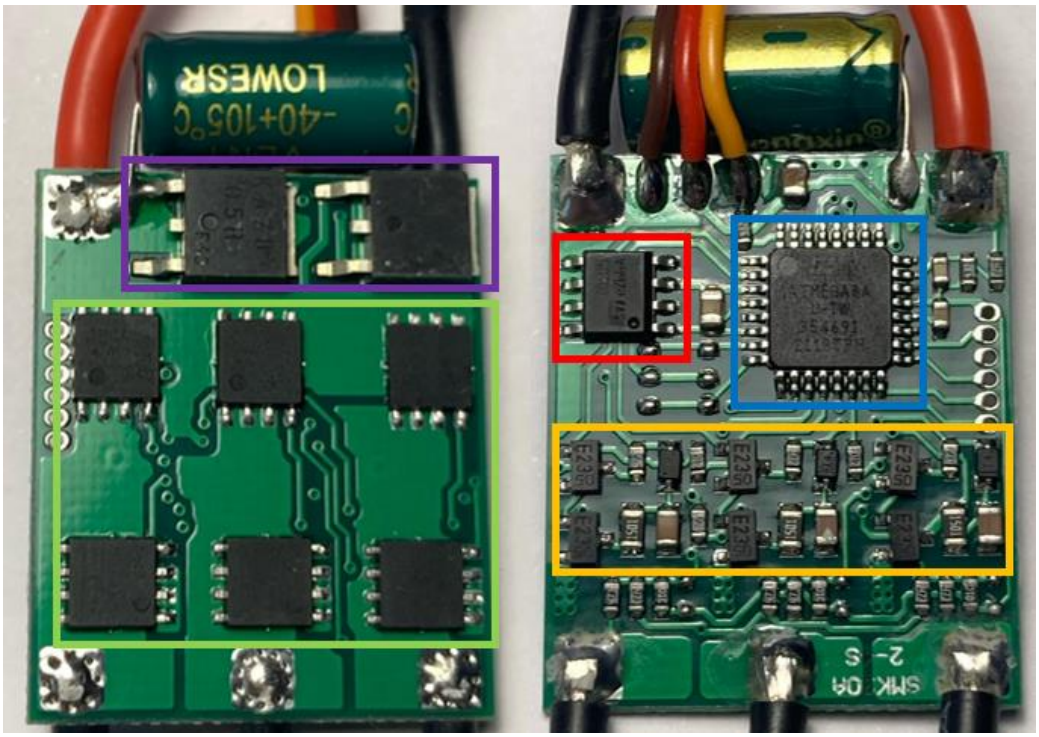


Figura 12. *ESC SimonK 30A, componentes internos tras la retirada de disipadores.*

En esta imagen, se observan sus principales componentes, mostrándose en azul el microcontrolador ATMEGA8A [29], en verde el puente que forman los MOSFET (*Metal Oxide Semiconductor Field-Effect Transistors*), TPCA8057-H [30], en naranja el grupo de transistores BJT E23 05 junto con resistencias, diodos y condensadores y finalmente dos grupos de reguladores de tensión, mostrándose en morado los reguladores KA7805R y en rojo el regulador 78L05.

El funcionamiento principal pasa por la recepción de la señal PWM de entrada en el microcontrolador ATMEGA-8A, este adaptará el ancho de los pulsos en función de la velocidad deseada, con la finalidad de generar señales adaptadas para los MOSFET, activándolos y desactivándolos, pudiendo así generar el desfase.

Gracias al puente se consigue la rectificación de la señal, obteniendo de los pulsos generados una parte negativa y positiva que conforman la señal sinusoidal. En cuanto al desfase generado es el microcontrolador el que se encargará de seleccionar que par de

MOSFET se activan en cada ciclo. Finalmente, son necesario los tres pares debido al bobinado delta del motor, la ***Figura 13*** muestra una imagen a modo de resumen, donde se expone el funcionamiento principal del ESC.

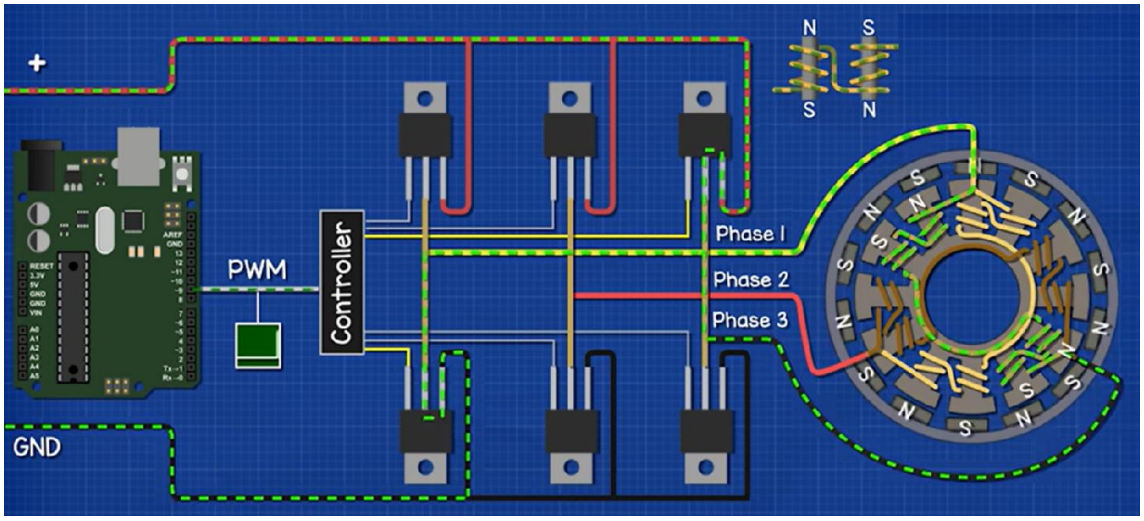


Figura 13. *Explicación gráfica del funcionamiento de un ESC* [19].

Centrándonos en el esquema electrónico se expone en la ***Figura 14*** la configuración de dicho puente. Este tipo de puente nos permitirá modificar el sentido de giro del motor simplemente intercambiando las conexiones de las fases 3 y 1 de salida del ESC. Esto sucede porque el microcontrolador mantiene el orden en el que activa los MOSFET, con lo que un simple intercambio de cables invertirá dicho orden.

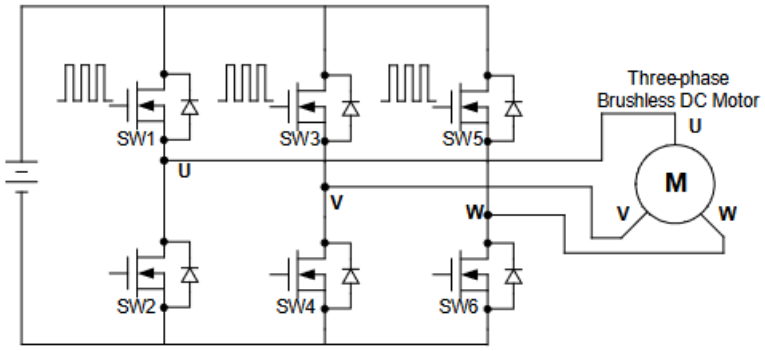


Figura 14. *Puente de tres fases, esquematización electrónica* [31].

Continuando con las distintas funcionalidades del ESC, debemos mencionar la necesidad del microcontrolador de conocer la velocidad de rotación de los motores. Algunos motores utilizan sensores de efecto Hall con la finalidad de llevar a cabo dicha tarea, en nuestro caso el controlador conocerá dicho sentido de giro a partir de la EMF (*Electromotive Force*) de retorno.

El ESC consigue llevar a cabo esta tarea debido a la dualidad del campo electromagnético. De igual modo que se utiliza la corriente con la finalidad de inducir un campo magnético en el bobinado que permita la iteración con los imanes del rotor, permitiendo así el movimiento, se sucede también el efecto contrario, donde los imanes del rotor generan una corriente de retorno en el bobinado [18]. Como consecuencia del desfase generado por el microcontrolador, existirán momentos en los cuales alguno de los bobinados no esté recibiendo corriente de salida del puente. En estas circunstancias, como consecuencia de la rotación de los imanes del rotor se genera una fuerza electromotriz inducida en la bobina. Esta corriente es medible y se utiliza para determinar la rotación o la velocidad, también conocida como EMF de retorno [19].

La siguiente ecuación aplica la Ley de Faraday para determinar el EMF de retorno [21]:

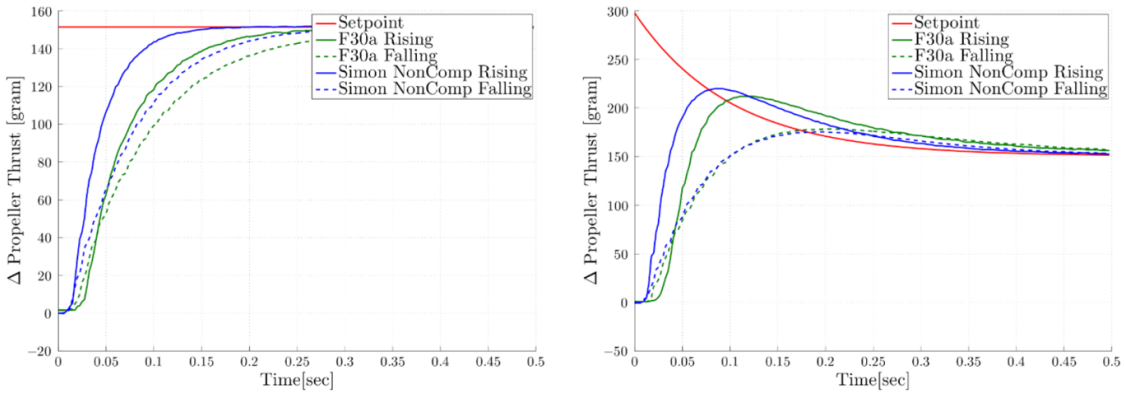
$$E = -N * \frac{d\varphi}{dt}$$

Donde E es la EMF de retorno, N el número de espiras del bobinado y la derivada muestra la variación temporal del flujo magnético φ. La cantidad de armónicos, así como la "idealidad" de la señal vendrán determinados por las condiciones geométricas de distribución de imanes y bobinado [21].

Respecto a la denominación de este tipo de ESC, visto que la electrónica, a grandes rasgos puede ser similar o basarse en los mismos principios, debemos hacer hincapié en su nombre. Este tipo de ESC se denominan SIMONK como referencia al *firmware* implementado por Simon Kirby [32].

El *firmware* hace referencia al código de programa compilado en el MCU (*Micro Controller Unit*), en este caso, nuestro ESC implementa un MCU con dicho firmware, por lo que tendrá características comunes en cuanto al código del programa, a pesar de que el *hardware* sea diferente o implementado con pequeñas variantes. Este *firmware* mejora los tiempos de respuesta del motor, en términos de empuje de las hélices frente a la señal PWM recibida. En la ***Figura 15*** se muestran algunas de las conclusiones principales en las que se compara el *firmware* de Simon Kirby. En dicha figura se representa una señal de entrada *Step1* que representa una estimación, frente a la señal *Step*

2 consistente en una señal de amplitud decreciente, pero con los mismos valores estacionarios.



ESC:	delay [ms]	τ_{up}[ms]	τ_{dn}[ms]	RMS
Standard	20	50	78	96
Simon	15	29	63	279
SimonCmp	15	30	35	11
From Parameters	N/A	32	32	0

Figura 15. *Comparación de los tiempos de respuesta empleando el firmware SimonK. Señal Step1 (izquierda), señal Step2 derecha y valores experimentales de un sistema de primer orden con retardo* [32].

En la figura anterior, se aprecia la mejora temporal en la respuesta del empuje del motor, donde se representa el tiempo el RMS (*Root Mean Square*) está determinado por la siguiente ecuación [32]:

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(R_i - F_i)^2}$$

Calculado como la diferencia de la subida R_i y la bajada F_i del empuje para un rango de 500ms y N el número de muestras.

Finalmente debemos hablar de los sonidos generados por el motor. De cara a la utilización práctica de los motores y a modo de realimentación el ESC generará señales analógicas para emitir diferentes sonidos en los motores, aportando realimentación sobre el estado del sistema.

A priori puede parecer una implementación poco intuitiva que los motores generen sonido, pero si nos centramos en el funcionamiento básico de un altavoz podemos observar que principalmente se encarga de convertir una corriente en una fluctuación de aire. Para ello un dispositivo emite una señal eléctrica que recorre una bobina, la cual

generará un campo electromagnético que transmitirá la señal a un imán acoplado a una membrana, generando la fluctuación de aire deseada y por tanto sonido.

Los motores BLDC poseen todos los componentes necesarios, el bobinado, los imanes y la señal generada por el microcontrolador del ESC. En este caso se emplea la carcasa del rotor como membrana, ya que solo se emitirán pitidos básicos a distintas frecuencias sin buscar una alta fidelidad sonora.

3.2.2.2 Implementación práctica

Tras haber explicado las características electrónicas asociadas, así como sus principales funciones nos centraremos en sus especificaciones prácticas. Nuevamente volvemos a encontrarnos frente a la falta de hojas de datos oficiales del fabricante, como consecuencia de la amplia utilización de este tipo de componentes electrónicos. Sin embargo, dado que la mayoría de ESC comerciales para motores BLDC emplean el *firmware* SimonK, incluso el propio Simon Kirby aporta una versión de código abierto en su repositorio de GitHub para controladores ATmega [33], podremos observar las características de componentes similares que implementen dicho *firmware*.

Inicialmente nos centraremos en su parte práctica o de utilización, para ello se ha tomado como ejemplo las instrucciones de usuario expuestas en [34], pertenecientes a una serie de ESC que emplean el *firmware* SimonK. Estas instrucciones aportan detalles acerca de los pasos para la configuración del ESC o su inicialización. Principalmente indican los tipos de sonidos para cada una de las casuísticas como armado de los ESC, control de frecuencia, o protección frente a ciertos valores de voltaje. Dado que los detalles específicos de dicha implementación se relacionan mayormente con el funcionamiento, serán tratados en la sección de *software* de este documento.

Centrándonos en los datos específicos que poseemos sobre el ESC nos remontaremos a la ***Figura 10*** donde se muestran sus principales características relacionadas con la corriente máxima que soportan y el tipo de baterías compatibles.

Respecto a su corriente máxima se aprecia que es de 30 A, superar dicha corriente podría generar que el ESC se calentara demasiado como consecuencia del efecto Joule.

En la práctica se han probado los ESC con ambos motores, observando que en el caso de la utilización del motor DX2205 llegaban a calentarse demasiado, llegando a quemarse durante usos prolongados a potencias máximas. Esto concuerda con lo expuesto durante la comparación técnica de los motores, donde se observaba que el motor A2212 consumía, para el voltaje dado, una menor cantidad de corriente.

Continuando con la implementación práctica se observa en la ***Figura 10*** sus compatibilidades con tipos de baterías, siendo compatible con baterías de NIMH (*Niquel*-Metalhidruro) de 5 a 12 celdas en serie y con Li-Po (Polímero de Litio) de 2 a 4 celdas en serie.

En nuestro caso se están empleando baterías Li-Po 3S, por lo que estamos dentro de los márgenes recomendados y no se ha apreciado ningún tipo de calentamiento excesivo o hinchazón en las mismas.

Finalmente, la ***Figura 16*** muestra el esquema de conexiones del ESC.

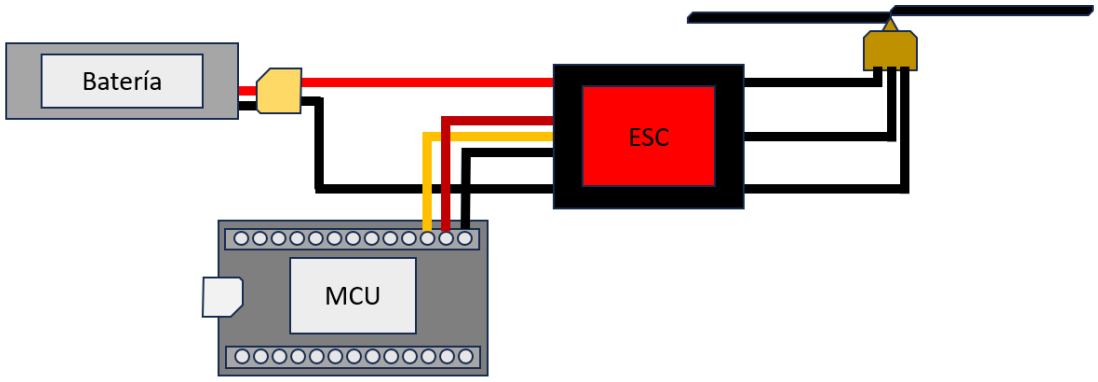


Figura 16. *Esquema de conexiones del ESC.*

3.2.3 Servomotores

Tras haber expuesto dos de los principales elementos que conformarán el sistema pasaremos a hablar de los servomotores. A la hora de seleccionar este dispositivo nos encontramos en el laboratorio con dos principales componentes, siendo estos el servo MG996R y el micro servo SG90, ambos pueden apreciarse en la ***Figura 17***.

Los servos nos permitirán implementar el movimiento de los motores, permitiendo así inclinarlos hacia delante o atrás, ya sea de forma simultánea o independiente.

Figura 17. *Servo MG996R (izquierda) y servo SG90 (derecha).*

Expuestos ambos servomotores, se dará paso a la exposición del estudio teórico.

3.2.3.1 Estudio teórico

El término servomotor se asocia con el concepto de conjunto electrónico en el que interviene un motor que, en adición con otros componentes electrónicos, permite regular una posición exacta en su rango de operación. Según lo define la RAE (Real Academia Española): "Sistema electromecánico que amplifica la potencia reguladora." [35].

A grandes rasgos, son conjuntos electrónicos que consiguen posicionar un motor con precisión en función de unas señales de entrada, lo más común es encontrar servomotores de corriente continua con escobillas, aunque existen todo tipo de variaciones [36].

Dado que el componente del motor que conforma el sistema cuenta con una base física similar a la expuesta durante la sección 3.2.1.1, nos centraremos en las peculiaridades que presentan.

En el caso que nos ocupa, la principal función del servomotor es poder determinar la posición de este, es decir, el ángulo de giro en el que se encuentra el motor. De este modo podremos posicionarlo exactamente en un ángulo o posición determinada dentro de su rango de actuación.

La ***Figura 18*** muestra de forma genérica el funcionamiento de un servomotor similar a los empleados para el desarrollo del VTOL. En ella se observan los principales componentes, destacando un sensor que permita determinar el posicionamiento del motor, el propio motor y un circuito de control que permitirá conocer la posición ofrecida por el sensor y recibir señales que permitan posicionar al motor en una posición determinada.

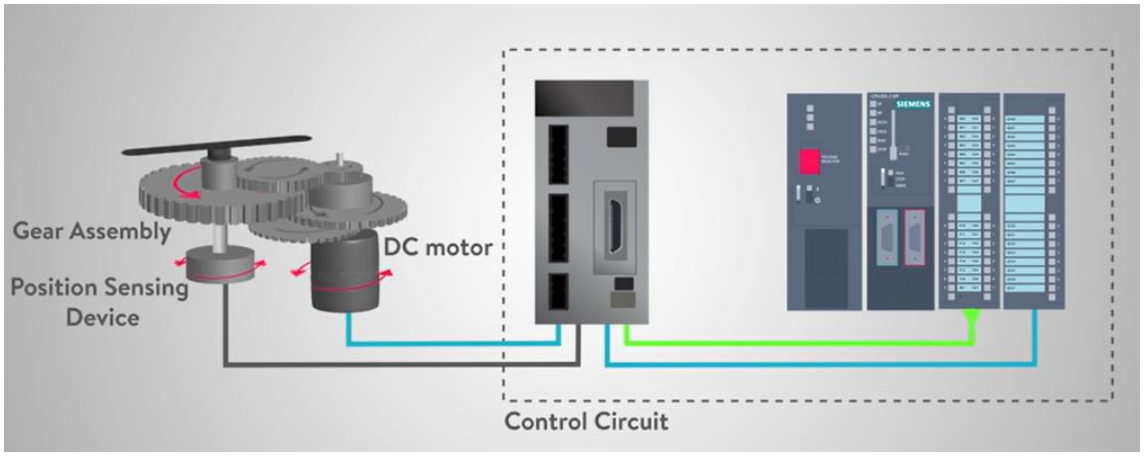


Figura 18. *Representación esquemática del funcionamiento de un servomotor* [36].

Con el objetivo de comprender en mayor profundidad dicho funcionamiento se aporta en la ***Figura 19*** una imagen del interior del servomotor MG996R. En ella podemos observar los engranajes que conformarán la rotación, destacando, en la esquina superior derecha de la primera sección, el tope que impide que el servomotor gire de forma infinita, ajustándose al rango permitido por el potenciómetro.

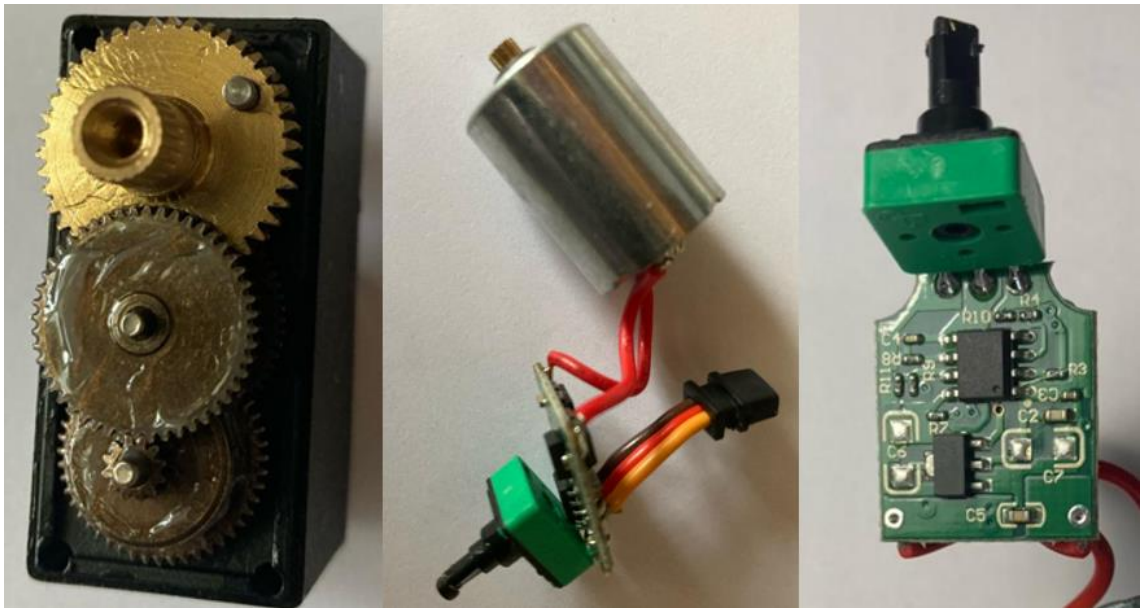


Figura 19. *Interior del servomotor MG996R.*

Respecto a su electrónica interna los servomotores de este tipo suelen contener un controlador de motor que actuará en función de un comparador, encargado de nivelar las señales del potenciómetro y de la señal PWM de entrada traducida a voltaje analógico.

La siguiente figura muestra el funcionamiento lógico de un servomotor de forma esquemática. En ella puede apreciarse cómo el comparador generará una salida que provocará la actuación del motor en el caso de que exista una diferencia entre el valor reportado por el potenciómetro, que indica la posición del motor como consecuencia de estar acoplado a su principal engranaje, y la señal de entrada PWM traducida a nivel de tensión.

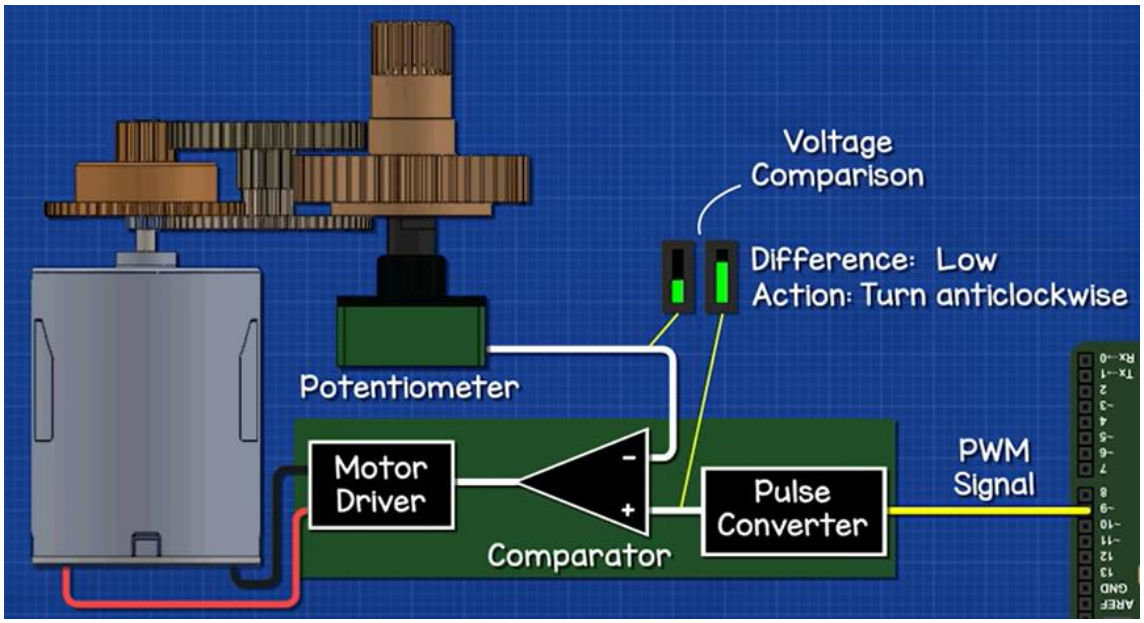


Figura 20. *Funcionamiento lógico de un servomotor* [37].

El comparador empleado podría estar formado por un amplificador diferencial, al que, si le añadimos la característica de amplificar la señal y una baja impedancia de salida, asegurando una intensidad suficiente que permita generar un campo electromagnético con potencia necesaria para mover el motor, estaremos hablando de un amplificador operacional.

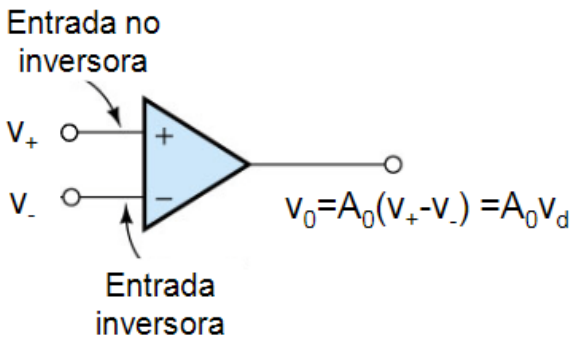


Figura 21. *Esquema Amplificador Operacional en lazo abierto* [38].

Tal y como se expone en la figura anterior estaremos generando una tensión con signo positivo o negativo, induciendo corrientes en el motor que permitirán girar en sentido horario o antihorario, ajustando rápidamente la posición de este en función de la señal suministrada. La señal PWM modulará su ancho de pulso para indicar, entre la posición máxima y mínima, el giro en el motor.

En lo referente al módulo conversor de PWM podría estar formado por un LPF (*Low Pass Filter*) conectado a un amplificador operacional con un *buffer* de salida para eliminar los efectos de la carga [39].

Conocemos que una señal PWM está compuesta por una componente de continua (*offset*) y la onda cuadrada que compone el tren de pulsos PWM, esta representación puede apreciarse en la ***Figura 22***.

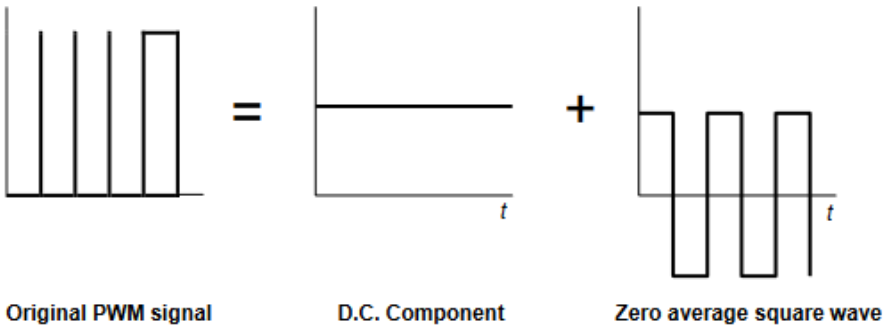


Figura 22. *Descomposición de una señal PWM* [40].

Además, una forma de onda PWM puede descomponerse mediante una suma de sinusoides con el *offset* anteriormente mencionado. Dicho *offset* puede filtrarse utilizando un LPF que posea una frecuencia central baja [41].

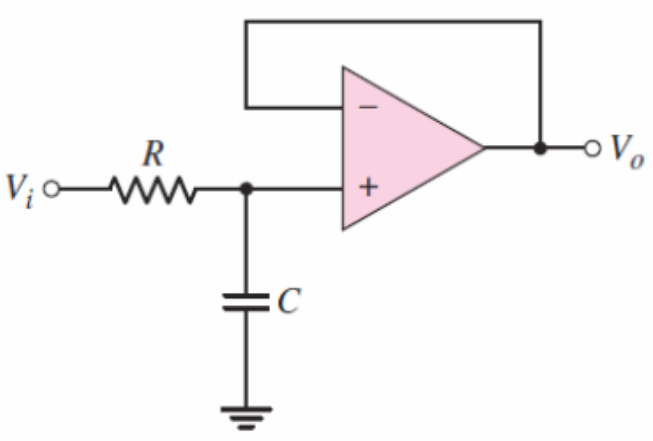


Figura 23. *Filtro paso bajo con buffer de salida* [39].

La serie de Fourier de una función $f(t)$ periódica y par similar al PWM puede expresarse según [40]:

$$f(t) = A_0 + \sum_{n=1}^{\infty} A_n \cos\left(\frac{2\pi nt}{T}\right) + \sum_{n=1}^{\infty} B_n \operatorname{sen}\left(\frac{2\pi nt}{T}\right)$$

Donde A_0 representa el *offset* de la función y sus coeficientes A_n y B_n, simplificando obtenemos los valores de los coeficientes [40]:

$$A_n = \frac{1}{T}\int_{-T}^{T} \cos\left(\frac{2\pi nt}{T}\right)\, dt$$

$$B_n = \frac{1}{T}\int_{-T}^{T} \operatorname{sen}\left(\frac{2\pi nt}{T}\right)\, dt$$

Calculando los n armónicos podemos observar cómo la mayor parte de la componente de *offset* se encuentra en la frecuencia más baja, motivo por el cual, utilizando un LPF estaremos acercándonos más al valor de dicho *offset*, teniendo en cuenta los armónicos más despreciables, armónicos de mayor n. La frecuencia central del filtro vendrá determinada por el pulso PWM que queremos filtrar y dependerá de los valores de la resistencia y el condensador [41]:

$$f_c = 1/2\pi RC$$

Si realizamos simulaciones con PSpice [42] podremos visualizar una salida similar a la mostrada en la ***Figura 24***.

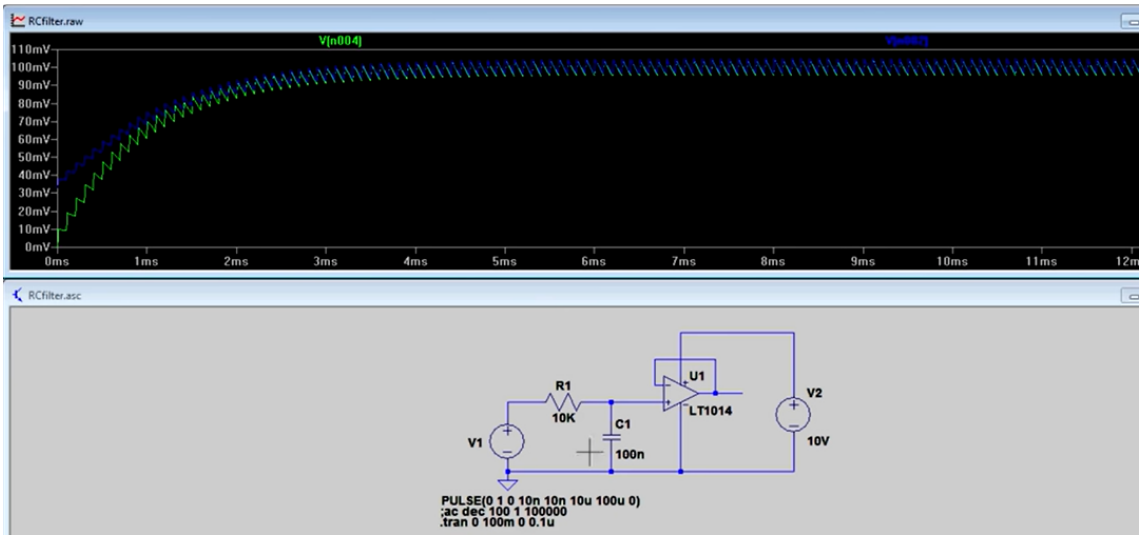


Figura 24. *Simulación de conversor PWM a DC PSpice.*

En esta figura los dientes de sierra pueden relacionarse con el *slew-rate* impuesto por el amplificador operacional [38]:

$$2\pi f V < SR \;\rightarrow\; f_{máx} = \frac{SR}{2\pi V}$$

Realizada una pequeña explicación acerca de la electrónica de los servomotores, se dará paso al estudio de sus características, empleando para ello las hojas de datos de ambos componentes [43][44].

Las características principales de corriente y voltaje son similares, encontrándose en unas condiciones de operación de aproximadamente 5V y 1A, pudiendo ser ligeramente superiores en el caso del MG996R. Nos centraremos en los principales parámetros, siendo estos el torque, la velocidad de movimiento, la velocidad mínima de movimiento y el peso.

Comenzando con el peso, podemos decir que el servomotor MG996R posee un peso de 55g frente a los 9g del SG90, sin embargo, esto no será un problema dado que los motores que poseemos tendrán potencia de sobra para poder levantarlos.

La característica más significativa que nos encontramos es el torque, según se define en las hojas de datos *stall torque*, este término hacer referencia al torque máximo que el motor puede ejercer desde una posición estática (*stall*), nos encontramos ante los valores de 9.4 kgf*cm (4.8V) 11kgf*cm (6V) y 1.8kgf*cm para los servomotores MG996R y

SG90 respectivamente. A simple vista podemos apreciar que el torque del motor de mayor tamaño será bastante superior, multiplicando casi en una magnitud la característica aportada por el motor SG90.

Los kilogramos de fuerza vienen descritos respecto a una longitud del eje de un centímetro, en este caso se optará por el de mayor torque, dado que si incrementamos la longitud desde el eje central de giro este torque disminuirá. En la sección 3.2.1 se exponía cómo la relación entre el torque y la fuerza era el producto vectorial del radio, con lo que, si aumentamos el radio manteniendo el par, menor será la fuerza aplicada.

De cara a la implementación práctica esto será importante, dado que los servomotores deberán poder mover la carga de los BLDC.

Otro de los factores a tener en cuenta será la velocidad del giro, lo que nos permitirá una rápida actuación a la hora de calibrar nuestros PID para el *yaw*. En este, es el servomotor SG90 el que saca ventaja con una velocidad de operación de 0.1s/60º frente a los 0.17s/60º (4.8V) 0.14 s/60º (6V) del MG996R. Sin embargo, esta cantidad no será lo suficientemente significativa como para despreciar ese torque potencialmente necesario.

Por último, nos encontramos con el ancho de banda "muerto" esta característica nos indica cual será el incremento mínimo necesario de ancho de pulso para que el motor se mueva.

Si tenemos en cuenta que ambos servomotores están preparados para trabajar a una frecuencia de 50Hz, se describen las siguientes ecuaciones respecto al ciclo de trabajo:

$$D = \frac{PW}{T} \rightarrow D = \frac{PW}{20ms}$$

En la anterior ecuación se define D como *Dutty Cycle* y PW como *Pulse Width*. Con lo que la característica de *death band width* nos está indicando cual es el mínimo ciclo de trabajo al que se moverá el motor. Se exponen a continuación las ecuaciones para ambos motores:

$$MG996R \rightarrow D = \frac{5\mu s}{20ms} \rightarrow D = 0.025\%$$

$$\text{SG90} \rightarrow D = \frac{10\mu s}{20ms} \rightarrow D = 0.05\%$$

Esta medida es similar a una tolerancia en el error, indica cómo el servomotor tendrá una mayor granularidad en sus pasos, pudiendo moverse con incrementos del 0.025% en el ciclo de trabajo, para el caso de MG996R, mientras que el SG90 necesitará un mayor incremento, teniendo una menor cantidad de pasos.

3.2.3.2 Implementación práctica

De cara a la prueba práctica de ambos motores, se ha probado su velocidad de reacción. En cuanto a esta característica se ha podido visualizar que no había distinción prácticamente perceptible en lo que a la velocidad de movimiento de los motores se refiere.

La parte más notable, tal y como se mostraba durante el estudio teórico ha sido la fuerza que los servomotores ejercían durante el giro. Para ello se han colocado estáticos, sujetos a una superficie plana, intentando mover objetos cotidianos de distintos pesos.

El motor SG90 no ha podido mover alguno de los objetos con la misma facilidad que el MG996R, quedando así descartado de cara a la implementación práctica. Este factor ha sido el que ha determinado principalmente la elección, dado que no solo necesitará mover el peso de los BLDC A2212, sino que también deberá poder mover la estructura, así como superar la fuerza de vientos en contra o el flujo de aire generado por las propias hélices.

3.2.4 Acelerómetro – Giróscopo

Tras haber expuesto los componentes relacionados con la elevación, pasarán a tratarse aquellos que se centran en la detección de realimentaciones externas. Uno de los encapsulados más importantes será el conjunto MEMS (*microelectromechanical systems*) que incorpora acelerómetro/giróscopo, disponible en el laboratorio de la UVA, en este caso nos encontramos ante el MPU-6050 [45].

Figura 25. *Conjunto de sensores MEMS MPU-6050.*

Este sensor es ampliamente conocido y utilizado en el mundo de la robótica y electrónica por sus prestaciones y su bajo precio, incluso se han generado nuevas versiones como el MPU-9250 [46], empleado durante desarrollos anteriores [47].

3.2.4.1 Estudio teórico

En lo referente a este MEMS incorpora principalmente varios ADC (*Analog to Digital Converter*) así como un DMP (*Digital Motion Processor*) y sus sensores acelerómetro y giróscopo. Todo ello dirigido por un pequeño microcontrolador que permitirá efectuar las transformaciones de datos y comunicación en diversos protocolos como SPI (*Serial Peripheral Interface*) o I2C (*Inter-Integrated Circuit*).

Respecto al funcionamiento electrónico de un acelerómetro nos encontramos frente a un sistema de segundo orden donde, a partir del movimiento de una masa, midiendo su distancia respecto a unos electrodos fijos, podremos calcular la aceleración. Se exponen las ecuaciones [48]:

$$v = \frac{\partial e}{\partial t}$$

$$a = \frac{\partial v}{\partial t} \rightarrow a = \frac{\partial^2 e}{\partial t^2}$$

$$m * a = k * e \rightarrow a = \frac{k * e}{m} \rightarrow m * \left(\frac{\partial^2 e_i}{\partial t^2} - \frac{\partial^2 e_o}{\partial t^2}\right) = k * e_o + B\frac{\partial e_o}{\partial t}$$

Donde k es la constante de recuperación del muelle, B la constante de amortiguamiento proporcional a la velocidad, m la masa, a la aceleración, e_o el desplazamiento de la masa respecto a la armadura y e_i la posición de la masa respecto a una referencia externa. Con

lo que, a partir de la masa, la constante de elongación y el desplazamiento podremos calcular la aceleración.

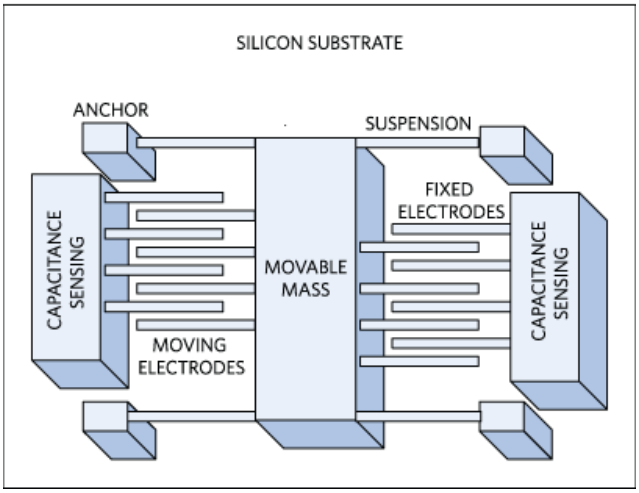


Figura 26. *Representación sobre un acelerómetro integrado.*[49]

La figura anterior representa un esquema funcional en el que pueden aplicarse las ecuaciones anteriormente expuestas para observar su funcionamiento. La magnitud eléctrica dependiente de la distancia, en el caso de la figura mostrada, es una capacidad [49].

$$C = (\varepsilon_0 * \varepsilon_r * A) * D$$

Donde $\varepsilon_0$es la permitividad eléctrica en el vacío, ε_r la permitividad eléctrica entre las placas, A la superficie de las placas superpuesta y D la separación entre los electrodos.

La ***Figura 27*** ilustra con mayor claridad el funcionamiento.

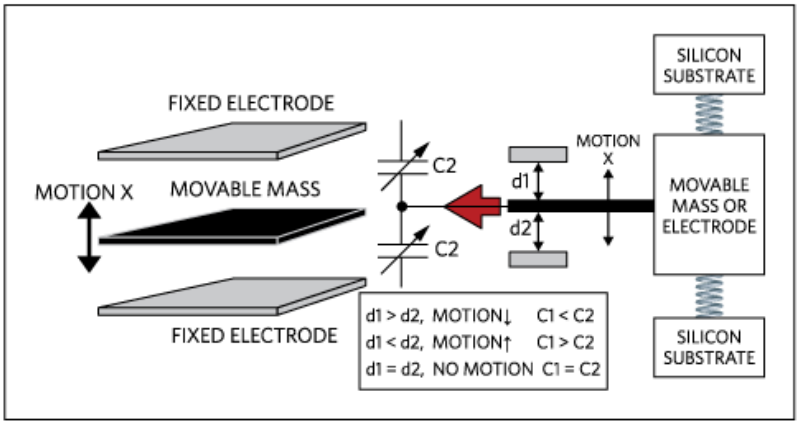


Figura 27. *Descripción del fenómeno capacitivo entre placas debido al movimiento de la masa.*[49]

En lo referente al giróscopo, el principio de funcionamiento es similar, basándonos en la capacidad que se genera cuando una masa se mueve, contenida por "muelles" u otro sistema de elongación.

La principal diferencia se basa en que, para este caso, se empleará el efecto Coriolis, que tiene en cuenta la velocidad de un elemento respecto a un objeto que lo contiene y se encuentra en movimiento. Sus ecuaciones se describen según:

$$\overrightarrow{F_c} = -2m(\vec{\omega} \times \vec{v})$$

Donde $\overrightarrow{F_c}$ es la fuerza Coriolis, m la masa, $\vec{\omega}$ la velocidad angular del sistema en rotación y $\vec{v}$ la velocidad del cuerpo en el sistema de rotación.

En este caso, dado que tendremos un cuerpo en movimiento, podremos observar cómo varían los valores de capacidad cuando el marco que contiene el cuerpo en movimiento rota, alterando así las direcciones de resonancia. La ***Figura 28*** ofrece un ejemplo de dicho sistema.

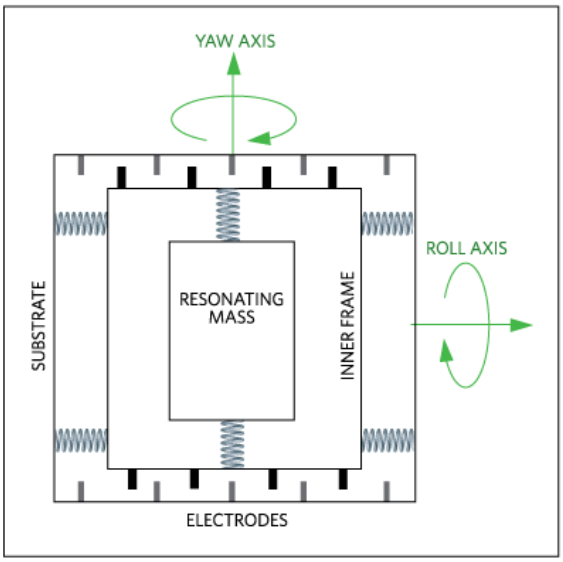


Figura 28. *Configuración espacial de un giróscopo integrado.*[49]

Tras haber visualizado las principales funcionalidades, en lo que a detección de parámetros físicos se refiere deberemos exponer sus características técnicas asociadas a dichas parametrizaciones.

Principalmente contaremos con un giróscopo de tres ejes con un rango dinámico programable de ±250, ±500, ±1000, y ±2000°/sec, esto supondrá un compromiso a la hora de elegir los rangos de operación, pudiendo obtener una granularidad mayor o decantarse

por rangos de operación más altos, ya que se utilizarán 16 bits para representar dichos rangos.

En el caso del acelerómetro nos encontramos con una situación similar teniendo un rango dinámico programable de ±2g, ±4g, ±8g y ±16g también con un ADC integrado de 16 bits para muestreo simultáneo.

En cuanto al integrado en el que viene insertado nuestro conjunto MEMS, podemos visualizar en la anteriormente mostrada ***Figura 25***. Que contamos con una entrada de 5V y poseemos la capacidad de utilizar los protocolos I2C (*twowire*) o SPI. Como características adicionales se incluye un regulador de tensión que permitirá convertir la señal de entrada de 5V a los 2.375-3.46V en los que opera el MPU6050 [45].

Si revisamos la configuración típica de funcionamiento expuesta en la hoja de datos, podremos apreciar que se corresponde con la configuración visualizada en la ***Figura 25***, en la que únicamente se ha incorporado un regulador de tensión, con sus resistencias asociadas y los condensadores de paso mostrados en la siguiente figura.

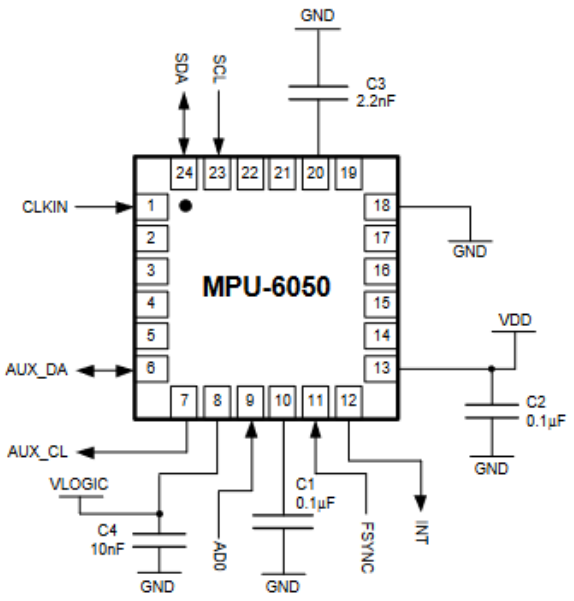


Figura 29. *Circuito de operación típico MPU-6050* [45].

El componente de color anaranjado es un condensador electrolítico de tántalo, que suele ser empleado, a pesar de su gran tamaño por su baja ESR (*Equivalent Series Resistance*), mejorando la respuesta transitoria del circuito y permitiendo un rápido amortiguamiento

de la señal, además de contar con una alta estabilidad en mayores rangos de temperatura [50].

3.2.4.2 Implementación práctica

Vistas sus principales características si discutirá sobre la implementación práctica. En el caso de uso que nos ocupa, nuestro MPU-6050 estará principalmente destinado a calcular la posición del VTOL, sus ángulos (*pitch*, *yaw* y *roll*) así como su velocidad angular.

Con todo esto vemos que este elemento, será imprescindible de cara a la implementación del PID, el núcleo central de cálculo de estabilización de nuestro VTOL.

Dado que los algoritmos de implementación de vuelo, así como su matemática asociada pueden englobarse dentro de una capa lógica, serán tratados durante la sección de *software* de este documento, discutiendo en mayor profundidad las entradas al conjunto MEMS, sus configuraciones o la matemática asociada para su funcionamiento.

3.2.5 Sensor ultrasonidos

Con el objetivo de optimizar algunos aspectos asociados al vuelo, mejorando la experiencia final, se ha empleado el sensor de ultrasonidos HC-SR04 disponible en el laboratorio de la UVA.

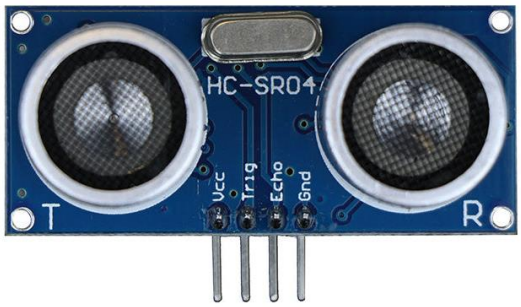


Figura 30. *Sensor de ultrasonido HC-SR04* [51].

Este sensor permitirá realizar cálculos de distancia precisos cuando la distancia no sea notablemente elevada, permitiendo ajustar algunas funcionalidades de nuestro VTOL.

3.2.5.1 Estudio teórico

Los sensores de ultrasonidos son ampliamente conocidos, sobre todo en el ámbito de la detección de distancias, destacan algunas aplicaciones como el SONAR (*Sound Navigation And Rangign*) cuando el uso de campos electromagnéticos no es viable, e.g: un medio como el acuático.

Su principio de operación es simple, se basan en que la velocidad del sonido en un medio concreto es constante, en el caso del aire esta velocidad es de 340m/s [18]. Cuentan con un emisor y un receptor, de modo que, midiendo el tiempo que ha tardado en volver la señal al origen, como consecuencia de haberse encontrado un medio no absorbente o parcialmente absorbente, puede calcularse la distancia recorrida por la onda.

En el caso de este sensor, empleará ultrasonidos, situados a frecuencias superiores del espectro audible del ser humano (~20kHz). Para la generación de estas ondas sonoras se emplean sensores piezoeléctricos, que permiten convertir las señales eléctricas en vibraciones debido a la contracción y expansión del material piezoeléctrico. Dada la reversibilidad de este tipo de sensores, podemos hablar de transductor, ya que será capaz de emitir y captar dichas vibraciones.

Además, el sensor estará operando en la zona de máxima sensibilidad entorno a la frecuencia central de operación, mejorando así la selectividad y respuesta [52].

Se expone a continuación la ecuación empleada para llevar a cabo el cálculo de la distancia:

$$d = \frac{c * t}{2}$$

Donde c representa la velocidad de propagación del sonido en el medio, d la distancia y t el tiempo total empleado por la onda desde su emisión hasta su recepción. La ***Figura 31*** muestra un esquema lógico-electrónico de un transductor ultrasónico piezoeléctrico.

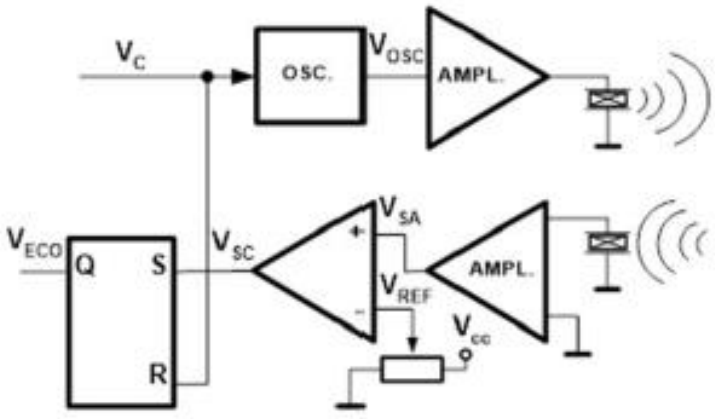


Figura 31. *Esquematización electrónica de un transductor piezoeléctrico sonoro* [52].

En el caso concreto de nuestro sensor HC-SR04, podemos observar su cristal oscilador en la parte central de la ***Figura 30***, vemos además como cuenta con 4 pines de entrada, 2 de ellos destinados a alimentación, un pin destinado al *trigger* y otro destinado a la recepción de la señal *echo.*

Al activar el pin *trigger* se activará una ráfaga de pulsos sónicos que se recibirán en el receptor activando la señal de *echo* y pudiendo así calcular la distancia recorrida por el pulso calculando el tiempo empleado [51].

Centrándonos en sus características específicas observamos en [51] que posee un consumo de corriente en reposo (*quiescent*) inferior a 2mA, un rango de distancia de operación máximo de 2 a 4 metros, un ángulo efectivo de detección de $\pm$15cm y un ancho de pulso necesario para el *trigger* de 10μs.

Estas características nos permiten entrever que la utilidad del sensor estará delimitada principalmente por su longitud máxima de alcance, destinado principalmente a detección de objetos cuasi perpendiculares, debido a su directivo ángulo de detección.

3.2.5.2 Implementación práctica

En la práctica el sensor será empleado para mejorar aspectos relacionados con la etapa de despegue, permitiendo ajustar valores que únicamente comenzarán a funcionar cuando el dispositivo se eleve una distancia pequeña del suelo, tales como el PID del *yaw*.

En lo relativo a otros usos, podrá servir para implementar un PID de altitud con mayor precisión cuando el dispositivo acabe de despegar o vaya a aterrizar, aumentando la fiabilidad y precisión del sistema.

3.2.6 Sensor de presión

En lo referente al sensor de presión, se hará necesaria su implementación de cara a la obtención de los parámetros físicos de altitud, permitiendo conocer la altura del dispositivo cuando este se encuentre a una longitud elevada durante el vuelo. En este caso se ha empleado el sensor de altitud BMP390 disponible en el laboratorio de la UVA.

Figura 32. *Implementación circuital sensor BMP390.*

Se trata de un sensor de la marca Bosch[53] , cuya implementación circuital se basa en un diseño de Adafruit [54], principal vendedor que ofrece el producto junto con sus bibliotecas.

3.2.6.1 Estudio teórico

En lo referente a los principios físicos nos encontramos con un sensor de temperatura-presión. Estas constantes físicas nos permitirán relacionar los parámetros obtenidos en las mediciones del sensor con la altitud de este. El fabricante expone una imagen gráfica sobre el funcionamiento de sus sensores de presión, dicho funcionamiento se expone en la ***Figura 33***.

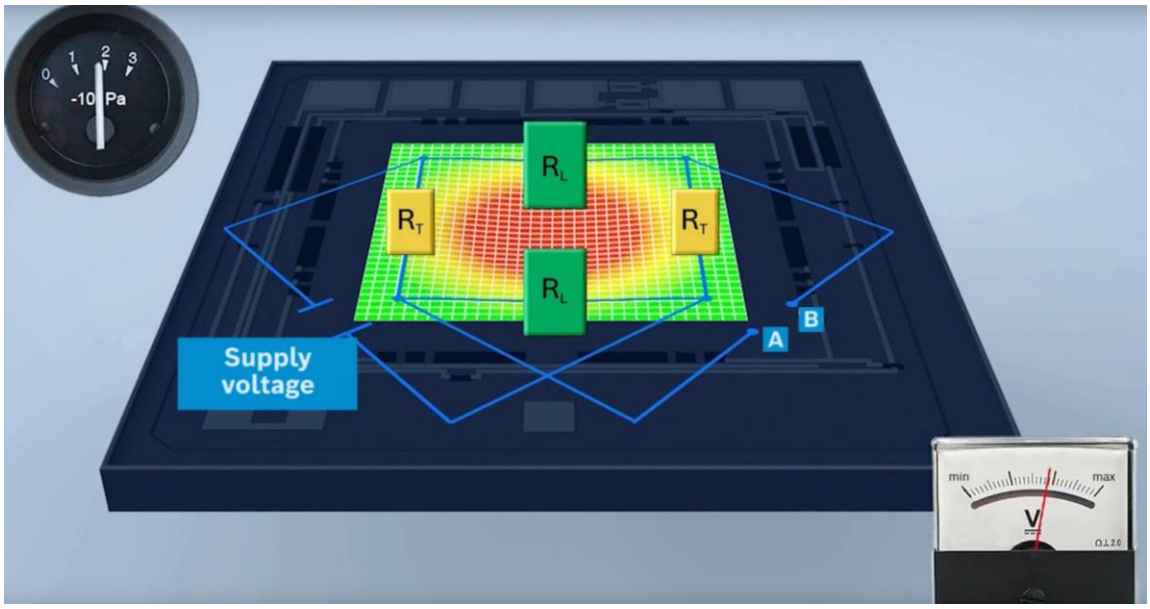


Figura 33. *Principio de funcionamiento resistivo en sensores de presión Bosh*[55].

Esta figura muestra cómo se implementa el circuito de acondicionamiento mediante un puente Weathstone, cuyas resistencias varían en función de la presión atmosférica que se ejerce sobre la placa metálica del sensor:

$$V_A = \frac{R_L}{R_T + R_L} * V_S$$

$$V_B = \frac{R_T}{R_L + R_T} * V_S$$

$$V_{out} = V_A - V_B = \left(\frac{R_L}{R_T + R_L} - \frac{R_T}{R_L + R_T}\right) * V_S = R_L \frac{1 - R_T}{1 + R_T} * V_S$$

Al tratarse de un puente de Weathstone balanceado, nos encontraremos con que, cuando no hay deformación $R_L = R_T \rightarrow V_{out} = 0$, mientras que cuando se produzca una deformación máxima $V_{out} \approx R_L * V_S$. Obteniendo así un valor de tensión proporcional a la presión atmosférica.

En su hoja de datos especifica de forma explícita que uno de los dispositivos objetivos para su uso es la implementación de drones. Destacando una precisión mínima relativa en la presión de $\pm$0.25m desde 700 a 1100hPa, para un rango de temperatura entre 25 y 40 grados celsius [53].

Para realizar el cálculo de la altitud, tanto en las bibliotecas de Adafruit para la familia de sensores 3XX [56], como en la documentación de Bosch para modelos de sensores de

presión anteriores (BMP180) [57], se emplea la siguiente ecuación, partiendo de la presión barométrica:

$$P = P_b \left[1 - \frac{L_{M,b}}{T_{M,b}}(h - h_b)\right]^{\frac{g_0 M_0}{R^* L_{M,b}}}$$

$$Altitud = 44330.3 * (1 - \frac{p}{p_0})^{\frac{1}{5.255}}$$

Donde P_b es la presión de referencia, $T_{M,b}$ la temperatura de referencia, $L_{M,b}$ la tasa de cambio de temperatura, h la altura a la que la presión es calculada, h_b la altura de referencia, g_0 la gravedad, M_0 la masa molar del aire en la tierra y R^* la constante universal del gas. Sustituyendo los valores $R^* = 8.3144598\,\mathrm{J/(mol \cdot K)}$, $g_0 = 9.8\,m/s^2$, $M_0 = 0.02896 Kg/mol$, $T_{M,b} = 25ºC + T = 273.15ºK + T$ ($Con\ T \sim 15$), y $L_{M,b} = -0.0065 K/m$, $h_b = 0$, llegamos finalmente a que p es la presión medida y p_0 la presión a nivel del mar o presión de referencia. Dicha presión puede obtenerse en la página de la AEMET (Agencia Estatal de Meteorología)[58].

Este es el motivo por el cual el sensor incorpora también un medidor de temperatura, permitiendo realizar la anterior ecuación con mayor exactitud y permitiendo que la temperatura, así como la tasa de cambio de la temperatura puedan ser ajustadas con mayor precisión.

3.2.6.2 Implementación práctica

En la práctica el sensor BMP390 nos permitirá medir la altitud a la que se encuentra nuestro VTOL, de este modo, utilizando una realimentación de los datos recogidos por el sensor, podremos implementar un PID que permita que nuestro dispositivo mantenga el vuelo a una altura constante, mejorando la estabilidad del dispositivo y permitiéndonos determinar la altitud a la que se quiere mantener el vuelo.

3.2.7 Fotosensor

De cara a la realización de pruebas de concepto, se empleará el fotosensor ADNS3080 [59]. Este sensor servirá para tratar de observar la viabilidad de reconocimiento de imágenes con la finalidad de determinar la posición. En la ***Figura 34*** se muestra el

módulo empleado durante las pruebas, que incorpora el sensor ADNS3080 junto con una lente.

Figura 34. *Módulo de flujo óptico para el sensor ADNS3080*

Expuesto el componente se dará paso a la exposición de su estudio teórico.

3.2.7.1 Estudio teórico

El sensor ADNS3080 es un sensor de flujo óptico, principalmente orientado a la fabricación de periféricos, concretamente está destinado a su ensamblaje en ratones de ordenador.

A priori se muestra que el propósito subyacente a la creación de dicho sensor nada tiene que ver con la navegación, sin embargo, debido a su bajo coste y a su característica de sensor de flujo se ha utilizado como ayuda de sistema de navegación de drones recreativos, una muestra de su utilización en esta área puede consultarse en la documentación del proyecto ArduPilot [60].

El sensor está diseñado para mejorar la utilización de ratones, empleando un fotosensor con una sensibilidad capaz de detectar 40 pulgadas por segundo y una aceleración de hasta 15g. Contiene un IAS (*Image Adquisition System*), un DSP (*Digital Signal Processor*) y cuatro puertos serie.

Respecto a su empleabilidad, es cierto que no nos permitirá obtener grandes imágenes de alta resolución, pero debido a su funcionalidad, en condiciones de alta luminosidad permitirá obtener imágenes de baja resolución en escala de grises, pudiendo así mejorar el posicionamiento.

El funcionamiento de este tipo de sensores se basa en la detección de diferencias de flujo óptico, de este modo, poseen la capacidad de determinar el movimiento relativo de posición.

La ***Figura 35*** muestra el esquema de ensamblado del sensor ADNS3080 en un ratón. En esta figura puede observarse cómo, mediante la implementación de un emisor de luz LED (*Light Emitting Diode*), se obtiene la luz reflejada en la superficie de forma directa en el sensor. Para ello, en el caso de ratones, se incorpora una lente que, mediante los mecanismos de refracción y reflexión de la luz permiten orientar el haz de luz reflejado directamente en el fotosensor.

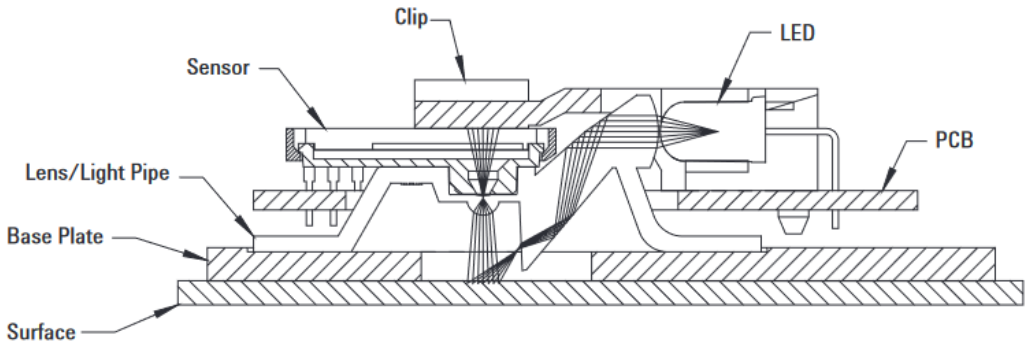


Figura 35. *Sección de ensamblado del sensor ADNS3080 en una PCB* [59].

El haz de luz reflejará de forma diferente en función de las irregularidades de la superficie en la que se encuentre, pudiendo así obtener un mapa topográfico en escala de grises mediante la matriz de fotorreceptores que incorpora el sensor. La ***Figura 36*** muestra los principales módulos del sensor ADNS3080.

En la imagen se muestran los módulos inmiscuidos en el proceso, destacando la matriz de fotosensores que se conectará directamente con procesador de imágenes incorporado. De esta forma, mediante el oscilador, se establecerá una frecuencia de muestreo para regular el tiempo de exposición de cada una de las imágenes tomadas y será el procesador de imágenes el que se encargue de calcular las diferencias entre las muestras topográficas de la superficie, con la finalidad de obtener un vector de dirección y obtener la posición relativa.

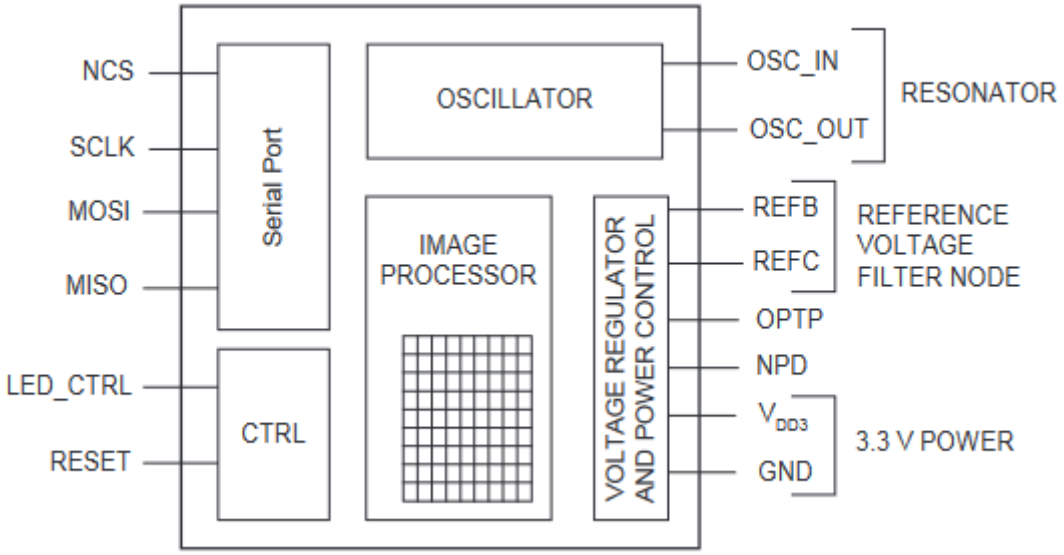


Figura 36. *Principales módulos del sensor óptico ADNS3080* [59].

Con el objetivo de mostrar de forma clara el proceso de obtención del desplazamiento, se incorpora la ***Figura 37*** en la que se ejemplifican de forma gráfica los pasos en el proceso de obtención de la imagen topográfica y el consecuente desplazamiento.

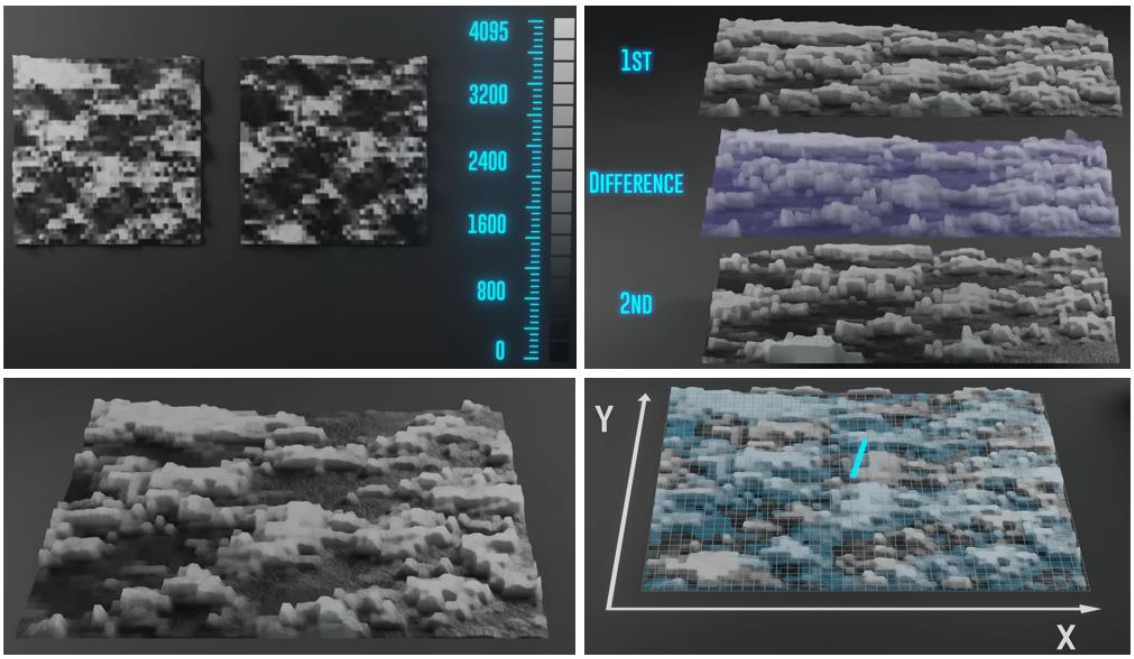


Figura 37. *Proceso de obtención del vector de dirección a través del sensor ADNS3080* [61].

En la figura puede observarse la imagen obtenida mediante la matriz de fotosensores en la parte superior izquierda, representando la profundidad topográfica en escala de grises.

Una vez obtenido un *frame* de dicha representación topográfica, se le resta el siguiente *frame*, y, a través del DSP, se realiza un cálculo matemático que trata de obtener la

diferencia mínima entre ambos, con la finalidad de conocer coincidencias, pudiendo conocer la posición inicial del *frame* 2 respecto al *frame* 1. Una vez encontrado el mínimo, y conocida la posición inicial del *frame* 2, se traza una correlación con el propósito de obtener el vector de desplazamiento. De este modo puede obtenerse tanto la dirección como la cantidad de desplazamiento.

Nuestro ADNS3080 cuenta con una matriz de 900 píxeles (0-899) y el rango de valores posibles de cada píxel abarca desde 0 hasta 221, según se expone en su *datasheet*. Además, se nos indica que la velocidad máxima que puede registrar dicho sensor es de 40 pulgadas/segundo (aproximadamente 1m/segundo [3.5km/h]) superando dicha velocidad no podrá computar los nuevos *frames*, ya que su tasa máxima de FPS (*Frames Per Second*) es de 6469 y no podrá encontrar el vector de dirección comparando con imágenes anteriores porque la imagen anterior será completamente distinta.

En esta situación vemos que únicamente será válido par aun vuelo a bajas velocidades.

Otro factor importante será la luminosidad mínima necesaria para que el sensor funcione correctamente, la tabla de la ***Figura 38*** indica los valores, en este caso la IRR (*Infrared Radiometer*) mínima incidente.

Irradiancia mínima (mW/m^2)	Longitud de onda (nm)
20	639
24	875
100	639
120	875

Figura 38. *Valores de irradiancia mínima por longitud de onda para el sensor ADNS3080* [59].

La luz emitida por el Sol está compuesta principalmente de frecuencias infrarrojas, teniendo su pico máximo entre 400 y 600 nm, a partir de las cuales comienza a descender la irradiancia solar [62], en nuestro caso observamos que la potencia mínima de detección del sensor se encuentra a la longitud de onda de 639nm.

Si consultamos cual es la irradiancia que llega a la atmósfera, se reporta que esta es de $1360W/m^2$. Al estar situados en España dicha potencia será un poco inferior, dado que no incide de forma perpendicular como consecuencia de la curvatura de la tierra, en la superficie de esta la irradiancia aproximada en un día completamente despejado podría concebirse en torno a un valor aproximado de $1000W/m^2$ [63].

A esta casuística deberá añadírsele la pérdida de irradiancia como consecuencia de la superficie de reflexión, sin embargo, vemos que las unidades son de varios ordenes de

magnitud mayores, con lo que no habría problema para su uso. Principalmente podríamos obtener dificultades para la obtención de datos durante días muy nublados.

La incorporación de una nueva lente en vez de la indicada en su hoja de datos permitirá conseguir que el foco lumínico se centre en el fotorreceptor del sensor para otras distancias superiores a las indicadas en la hoja de datos.

3.2.7.2 Implementación práctica

En la práctica se han realizado varias pruebas, empleando diferentes bibliotecas que serán tratadas durante la sección *software*. Se ha podido determinar la posición en condiciones de alta luminosidad, sin embargo, en días nublados o con baja luminosidad no se han conseguido datos fiables.

Además de esto, observando otras implementaciones prácticas de estos sensores, no se han obtenido los mismos resultados. Por último, hay que indicar que se ha descartado este sensor de la implementación, en parte por su baja precisión a elevadas alturas, dependiente además del color de la superficie, que reflejará distintas longitudes de onda. En el caso del ADNS3080 esto no es muy crítico, dado que el ratón está colocado con el sensor completamente tapado y únicamente se le emite luz de un led específico, sin embargo, a la hora de realizar pruebas, en función del incremento de la altitud y la superficie estas imágenes van perdiendo nitidez.

A esto se le suma que las bibliotecas empleadas utilizan interpolación para reducir el número de píxeles visibles, debido a la baja fiabilidad en distancias elevadas de una superficie, contribuyendo en una disminución de la precisión.

Como conclusión se indica que este sensor podría ser viable a bajas alturas en días soleados, por el contrario, no será una medida fiable en otras situaciones.

3.2.8 GPS

Continuando con el posicionamiento del VTOL, se ha utilizado durante las pruebas un GNSS (*Global Navigation Satellital System*) disponible en el laboratorio de la UVA, mostrado en la ***Figura 39***.

Figura 39. *Sensor GNSS NEO-M8N.*

Pasarán a comentarse las características técnicas de dicho sensor, así como las bases principales de su funcionamiento.

3.2.8.1 Estudio teórico

En lo referente a los GNSS existen varios tipos de implementaciones, que varían principalmente en función de los estados o instituciones que los hayan desplegado, debido a su relación directa con el ámbito militar y el campo de la seguridad. En el caso de nuestro sensor, será compatible con algunas de las bandas de los GNSS; GPS (*Global Positioning System*) [64], GLONASS (*Global'naya Navigatsionnaya Sputnikovaya Sistema*) [65], Galileo [66] y BeiDou [67] [68].

El principio de funcionamiento básico es común a la mayoría de ellos, se basa en la obtención de la posición relativa de los satélites respecto al objeto del que desea obtenerse su posición. Para ello, los satélites se encuentran continuamente enviando una señal que incorpora su posición con sus efemérides, junto con la marca temporal del instante en que se envió la señal. Esta marca temporal ha de ser extremadamente precisa, motivo por el cual, los satélites emplean relojes atómicos de alta precisión.

Con tres satélites podríamos obtener la longitud en la tierra, equivalente al cálculo de un único punto de corte en un sistema de dos dimensiones. El último es necesario para determinar la latitud a la que nos encontramos en nuestro sistema tridimensional. La ***Figura 39*** muestra de forma geométrica la anterior explicación.

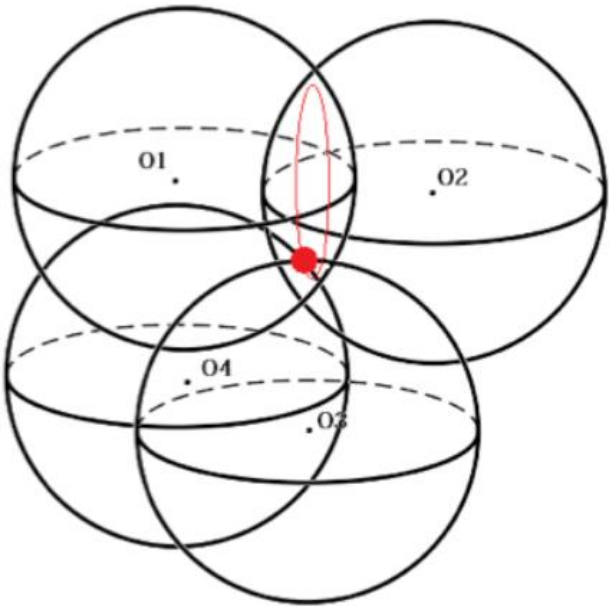


Figura 40. *Representación geométrica de la superposición de señales isotrópicas en un único punto* [69].

Debido a los efectos de la relatividad, dado que los satélites se encuentran orbitando a altas velocidades, se envían señales de corrección desde estaciones base situadas en tierra, corrigiendo así los desajustes temporales como consecuencia de los postulados de Einstein.

En lo referente a los cálculos matemáticos exactos, se calcula la posición del receptor teniendo en cuenta las coordenadas cartesianas de los satélites y la diferencia temporal entre el receptor y el reloj de los satélites, posteriormente se pasan a coordenadas esféricas y se extrapolan a varios satélites, si poseyésemos más información. Finalmente se aplican las ecuaciones normales del método de Gauss – Newton [70].

Nótese que la precisión temporal puede llegar a introducir errores en la posición muy notables, esto recalca la necesidad de poseer tiempos de alta exactitud, minimizando así los errores. Además, la incorporación de más de cuatro satélites en las ecuaciones provoca una disminución del error en la determinación de la geolocalización.

Continuando con las características específicas del sensor, se indica que la precisión horizontal, en condiciones idóneas, es de 2,5m para GPS y GLONASS combinados, de 2,5 metros para GLONASS, de 3 metros para BeiDou y de 3m para Galileo. Sin embargo, las mejores condiciones indican "*CEP, 50%, 24 hours static, -130dBm, >6SVs*".

Estos valores son bastante restrictivos, ya que nos indica que el CEP (*Circular Error Probability*), definido como el radio del círculo que engloba el 50% de las medidas certeras del sensor GNSS, además de estar 24 horas en una posición estática, recibiendo una señal con una potencia mínima de -130dBm y con la utilización de más de seis satélites [71]. Se indica además que pueden llegar a su posición mínima (2m, solo para GPS o GLONASS con GPS), en caso de uso de SBAS (*Satellite Based Augmentation System*), empleando para ello otras constelaciones de satélites como EGNOS [72].

Su hoja de datos incluye una amplia documentación sobre compatibilidades, codificaciones de señales o modulaciones que es capaz de interpretar. Con la finalidad de centrarnos en su utilización, se expondrá su conexión para transmisión de datos.

Este dispositivo viene con un conector USB (*Universal Serial Bus*), que ha sido sustituido para poder emplear su protocolo I2C, reduciendo así la cantidad de conexiones necesarias para su funcionamiento, la ***Figura 41*** muestra el conector con la modificación incorporada.

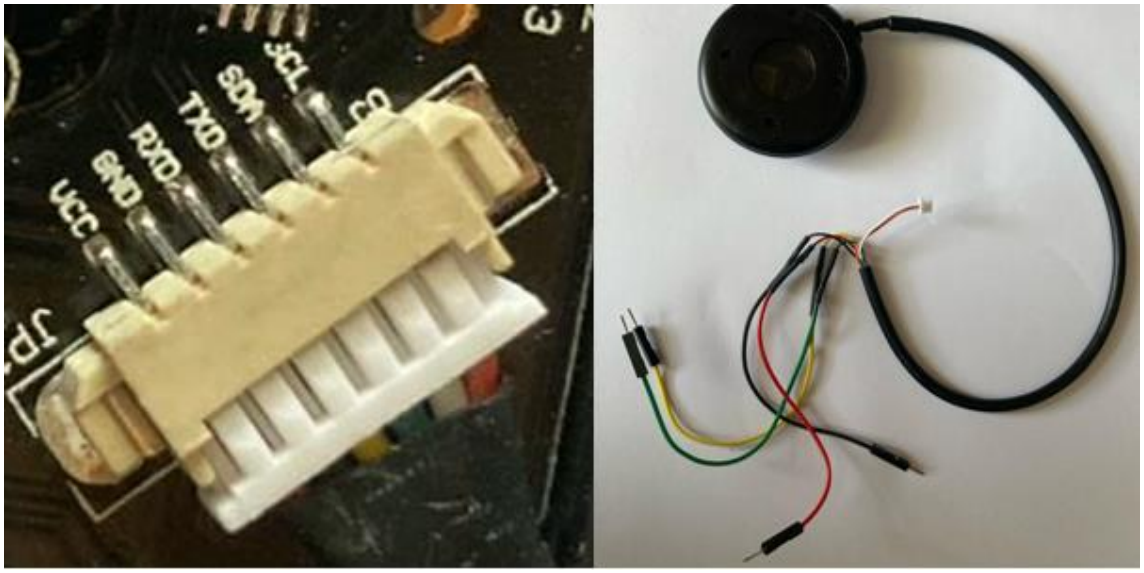


Figura 41. *Modificación de conector de USB a pines para su utilización mediante I2C.*

3.2.8.2 Implementación práctica

En cuanto a su implementación práctica, se ha conseguido obtener la posición con una precisión relativa de unos 7-8 metros. En la sección *software* se incluirán más detalles acerca de las bibliotecas empleadas y la obtención de datos.

La premisa de utilización de un sensor GNSS es claramente viable en el caso de viajes largos, donde podría aportar una gran precisión a la navegación, e incluso permitir la incorporación de algún tipo de algoritmo realimentado para la navegación autónoma. Sin embargo, probar esta característica en la práctica no es viable desde el punto de vista legal.

En la página web de Ministerio de Transportes y Movilidad Sostenible se aportan los requisitos mínimos legales para el manejo de drones [73], entre los que figuran requisitos como la necesidad de obtener un registro como operador, formación como piloto de drones o la necesidad de poseer un seguro de responsabilidad civil, para drones con un peso superior a los 250g, así como UAS (*Unmanned Aerial System*) [74]. El límite de aplicación de 250 no tiene validez si el dron puede ser capaz de capturar datos personales o no es concebido como juguete. En el caso que nos ocupa, no se han incorporado ningún tipo de cámaras que pudieran afectar a la privacidad.

Este hecho imposibilita la implementación práctica del sensor GNSS dado que las pruebas se han realizado en un entorno privado controlado y teniendo en cuenta que el GNSS nos ha permitido diferenciar distancias de unos 7-8 metros en la práctica, no ha sido posible realizar pruebas fiables en las que se incorpore el dispositivo GNSS en el VTOL.

Debido a la basta regulación respecto a esta área y los distintos organismos reguladores tanto a nivel comunitario como nacional [75], las pruebas efectuadas con el VTOL diseñado se han realizado en una propiedad privada controlada, estableciendo límites de potencia por seguridad, sin superar las delimitaciones arquitectónicas de la misma, con la finalidad de evitar posibles inconveniencias legales.

3.2.9 Microcontrolador

Una vez comentados los sensores barajados para la implementación de las diferentes características de nuestro VTOL, será necesario tratar cual será el dispositivo central que gestionará todos los datos e implementará la lógica principal del programa. En este caso se ha empleado el microcontrolador ESP-WROOM-32 [76], mostrado en la ***Figura 42***.

Figura 42. *ESP-WROOM-32 DevKit v1.*

Este microcontrolador actualmente está descatalogado debido al surgimiento de nuevas versiones por parte del fabricante, incluso se detalla en su hoja de datos "*Not Recommended For New Designs*", sin embargo, al ser el material disponible y contar con las capacidades necesarias para el desarrollo del VTOL se utilizará esta versión.

3.2.9.1 Estudio teórico

En lo referente a la decisión de utilización se han tenido en cuenta principalmente:

- Sus características de procesamiento, incorporando dos microprocesadores Xtensa®32-bit LX6 y poseyendo una frecuencia de reloj de CPU desde 80MHz a 240MHz [76], permitiendo ejecutar más instrucciones de código en un menor tiempo, frente a otros competidores como Arduino [77]. Este hecho puede visualizarse en algunas comparativas llevadas a cabo por la comunidad, donde se comparan consumos, CPU y ciclos de procesamiento, llevando a cabo experimentos en el establecimiento de salidas en alta y baja, así como en el procesamiento de instrucciones [78].

- Su conectividad y cantidad de puertos, que nos servirán para poder conectar múltiples sensores, soportando los protocolos UART (Universal Asynchronous Receiver/Transmitter), SPI o I2C (Two-Wire) [76].
- Sus tarjetas de comunicación inalámbrica, que soportarán Bluetooth v 4.2, permitiendo conectar todo tipo de dispositivos, así como WiFi (*Wireless Fidelity*) en su versión 802.11 b/g/n [76].
- Su capacidad para generar múltiples pulsos PWM frente a otros microcontroladores que tienen un número menos elevado de puertos, esto nos limitaría de cara a enviar las señales a los motores.

En lo referente a las conexiones implementadas, así como sus características para los puertos empleados serán discutidos durante la sección de los desarrollos esquemáticos y *software*, donde serán analizadas en detalle.

3.2.9.2 Implementación práctica

En la práctica se han utilizado todas las características mencionadas anteriormente, empleando pulsos PWM para el envío de señales de control a los ESC que gestionarán los motores, utilizando protocolos I2C(Two-Wire) para la comunicación con los sensores y UART para la comunicación con otros ESP-32.
En lo referente a las características de procesamiento, se ha observado su potencia a la hora de implementar los PID, obteniendo tiempos de respuesta cortos que permiten una mejor implementación de los sistemas de control.

Finalmente se han aprovechado sus conexiones WiFi para el envío de estadísticas de forma remota durante las pruebas de vuelo, así como su conexión Bluetooth para su conexión con el control manual del VTOL.

3.2.10 Alimentación

En lo referente a la alimentación se han empleado las baterías RoaRingTop 35c de alta descarga, esta batería se muestra en la ***Figura 43***. La capacidad de las baterías es de 2200mAh, formada por celdas de polímero de litio, que suman un total de 11.1V.

Figura 43. *Batería RoaRingTop 35c 2200mAh.*

3.2.10.1 Estudio teórico

Respecto a este tipo de baterías nos centraremos en sus principales características. Comenzando con la nomenclatura observamos que se indica 3S 11.1V, la S nos indica que la batería está compuesta por tres celdas Li-Po en serie [79], teniendo en cuenta que el voltaje nominal de una batería Li-Po, debido a la configuración química de las celdas de iones de litio, es de 3.7V tenemos $3.7V * 3 = 11.1\,V$.

Otro tipo de nomenclaturas podrían designar que se están colocando las baterías en paralelo utilizando una "P".

En lo referente a 35c indica la tasa de descarga máxima de la batería, describiendo cuanta corriente es capaz de entregar en un instante de tiempo sin sufrir daños estructurales o químicos [79]. En nuestro caso poseemos una batería de 2200mAh con una tasa máxima de descarga de 35c, lo que implica que nuestra batería será capaz de entregar $35C * 2.2Ah = 77Ah$, lo que significa que puede entregar toda su carga en $\frac{1h}{35C} * 60 \approx 1.7\,minutos$ o lo que es lo mismo $\frac{77Ah}{60min} \approx 1.7\,minutos$. Esta característica es

importante ya que será la que nos permitirá entregar la potencia necesaria a los ESC y finalmente a los motores.

Respecto al voltaje de 11.1V podemos indicar que se trata de un voltaje nominal, definido como el RMS (*Root Mean Square*) con lo que, dependiendo de la carga de la batería, la temperatura u otros factores, podremos obtener valores de tensión distintos.

Las baterías constan además de un conector JST-XH-4P que servirá para monitorizar la carga de cada una de las celdas de la batería. Este tipo de conector fue creado por la empresa J.S.T. Mfg.Co [80] con la finalidad de definir estándares en el proceso de carga de baterías.

A continuación, se expondrá el proceso de carga con la finalidad de comprender el desarrollo completo.

3.2.10.1.1 Proceso de carga

De cara a la realización del proceso de carga será necesario emplear un balanceador que facilite el reparto de la carga en cada una de las celdas de nuestra batería de tres celdas Li-Po. El balanceador empleado ha sido el IMAX-B6 mostrado en la ***Figura 44***.

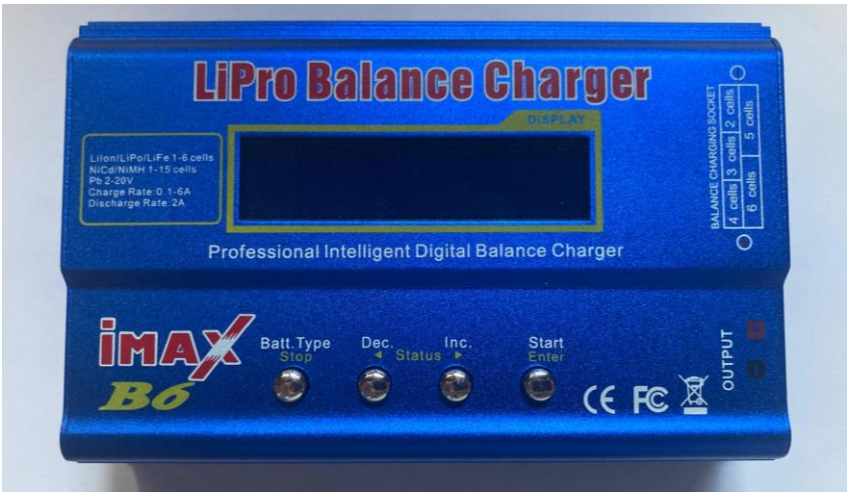


Figura 44. Balanceador de carga IMAX-B6.

Este balanceador es compatible con diferentes tipos de conectores JST-XH tal y como se muestra en la siguiente figura. Estos conectores servirán para poder monitorizar los voltajes de cada una de las celdas de la batería por separado, pudiendo balancear la carga para evitar que unas celdas se carguen de forma excesiva mientras que otras no están cargadas, lo que podría dañar la batería.

Tal y como se define en su manual de usuario, para la utilización de baterías cuya química está basada en Li-Po será necesario establecer un límite máximo de carga de 4.2V por celda y evitarse una descarga inferior a 3V por celda, para evitar el dañado de la batería.

Figura 45. *Conectores JST-XH en función de las celdas de la batería.*

Además de esto, se indica que la tasa de intensidad máxima debe ser inferior a 1C, en nuestro caso inferior a 2.2Ah [81].

Para realizar la carga será necesario seleccionar el tipo de batería Li-Po y establecer el voltaje y la carga máximos, en el caso que nos ocupa, se ha establecido en 1A para la carga y 11.1V. En cuanto a las conexiones deberemos conectar el positivo y negativo de la batería a los pines de salida OUTPUT del IMAX-B6, y el conector JST-XH-4P al conector *socket* de monitorización de balanceo de celdas.

Es importante destacar que el dispositivo de balanceo permite hacer una configuración totalmente libre en cuanto a tensiones, celdas y voltajes, por lo que será crítico conectar y configurar con precaución los valores de nuestras baterías.

3.2.10.2 Implementación práctica

En la práctica ha sido necesario modificar los conectores de las baterías y del balanceador con la finalidad de "estandarizar" las conexiones de nuestro VTOL. Aplicando las configuraciones indicadas en el manual para baterías Li-Po 3S de 11.1V no ha habido ningún tipo de problemas.

El balanceador cuenta además con protecciones frente a cortocircuitos, avisando con diferentes zumbadores en caso de establecer las corrientes en sentido inverso o en caso de que se superen los voltajes configurados en los ajustes durante el proceso de carga.

Durante vuelos intensivos se ha visualizado que la temperatura de las baterías ha aumentado significativamente al requerir entregar de golpe grandes cantidades de carga, sin embargo, no se han hinchado ni sufrido ningún tipo de daños.

3.3 Análisis de potencia

Una vez expuestos los componentes *hardware*, deberemos mencionar cuál será el consumo de potencia estimado, con la finalidad de determinar el tiempo de vuelo aproximado de nuestro VTOL.

Recopilando los datos de la sección *hardware* nos encontraremos con que el consumo máximo de nuestros motores BLDC será de 133.2W para los motores A2212 seleccionados. En lo referente a los ESC, dado que no conocemos de forma exacta el conjunto de componentes, ya que el vendedor no lo facilita de forma explícita, podremos estimar el consumo de este en un amperio, suponiendo que sus circuitos integrados trabajen a 5V tendríamos una adición de potencia de 5W.

Continuando con los servomotores, en su hoja de datos se indica que tienen un consumo de unos 500mA – 900mA para una tensión de 6V [43], en nuestro caso estarán alimentados con la salida de voltaje de los ESC, operando a 5V, además no estarán realizando cambios bruscos durante la mayor parte del tiempo, con lo que podemos estimar su consumo en unos $500mAh * 5V = 2.5W$.

En lo referente a los sensores, tenemos que el sensor de presión BMP390 estará operando a una tensión de alimentación de 3,3V, proporcionada por el ESP-32 principal, se especifica que su salida de 3V3 es capaz de aportar una tensión máxima de 3.6V y una carga máxima de 1.1A[76], sin embargo, el sensor BMP390, configurado con un *oversampling* a resolución estándar para la presión y a alta resolución para la temperatura tendrá un consumo típico de unos 13,8µA [53]. Será necesario incorporar los consumos aproximados del resto de componentes que permiten el funcionamiento del sensor, y se deberá tener en cuenta que operará a unos 50Hz. Con todo esto podemos estimar el consumo conjunto del encapsulado BMP390 en $25\mu A * 3{,}3V \approx 83\mu W$.

Continuando con el MPU6050 se indica que su consumo típico para uso único del acelerómetro y giróscopo con el DMP desactivado des de 3,8mA y operará a una tensión máxima de 3,46V [45]indicando que su consumo se encuentra entorno a los 13mW.

Finalmente toca hablar del HC-SR04 que operará a 5V y tendrá un consumo de unos 15mA cuando se es té usando, obteniendo 75mW de potencia.

Como hemos podido observar, el consumo de los sensores será prácticamente despreciable frente a la energía consumida por los motores, recopilando todos los datos:

$$133{,}2W + 5W + 2{,}5W + 83\mu W + 13mW + 75mW = 140{,}78W$$

Tomando estos datos y teniendo en cuenta que nuestra batería cuenta con 24,43Wh, la fuente de alimentación podrá suministrar 24,43W durante una hora, dado que tiene que proporcionar 140,78W, tendrá que aportar $\frac{140{,}78}{11{,}1V} = 12{,}68A$ de forma constante.

Haciendo una regla de tres obtenemos $\frac{12{,}68A*1h}{2{,}2A} = \frac{1h}{5{,}76} = 10{,}41min.$

El tiempo de vuelo estará reducido a 10,41 minutos teniendo en cuenta que los motores estén funcionando a su máxima potencia y los sensores y actuadores se empleen de forma constante.

En la práctica se ha podido determinar, que el tiempo de vuelo medio de cada batería completamente cargada es de unos 13-15 minutos, tiempo que será suficiente para la entrega de paquetes en cortas distancias.

Una de las posibles aplicaciones prácticas podría ser su implementación en entornos rurales, urbanizaciones o pequeños núcleos urbanos, donde la distancia de las viviendas a los centros logísticos de paquetería es notable para desplazamientos a pie, pero no excesivo empleando vehículos motorizados.

3.4 Desarrollo electrónico esquemático

Esta sección tiene como objetivo aclarar algunas de las características técnicas asociadas a la interconexión de los distintos elementos que componen el sistema, así como definir la implementación final tras la realización de las pruebas.

Se muestra en la siguiente página una figura relativa al diseño electrónico esquemático, la ***Figura 46*** aporta la historia de colores empleados en el esquemático.

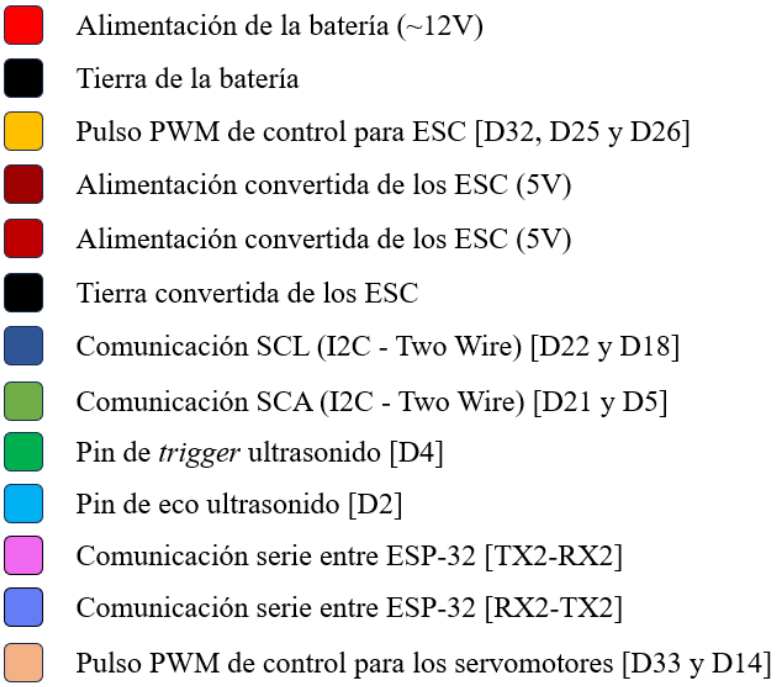


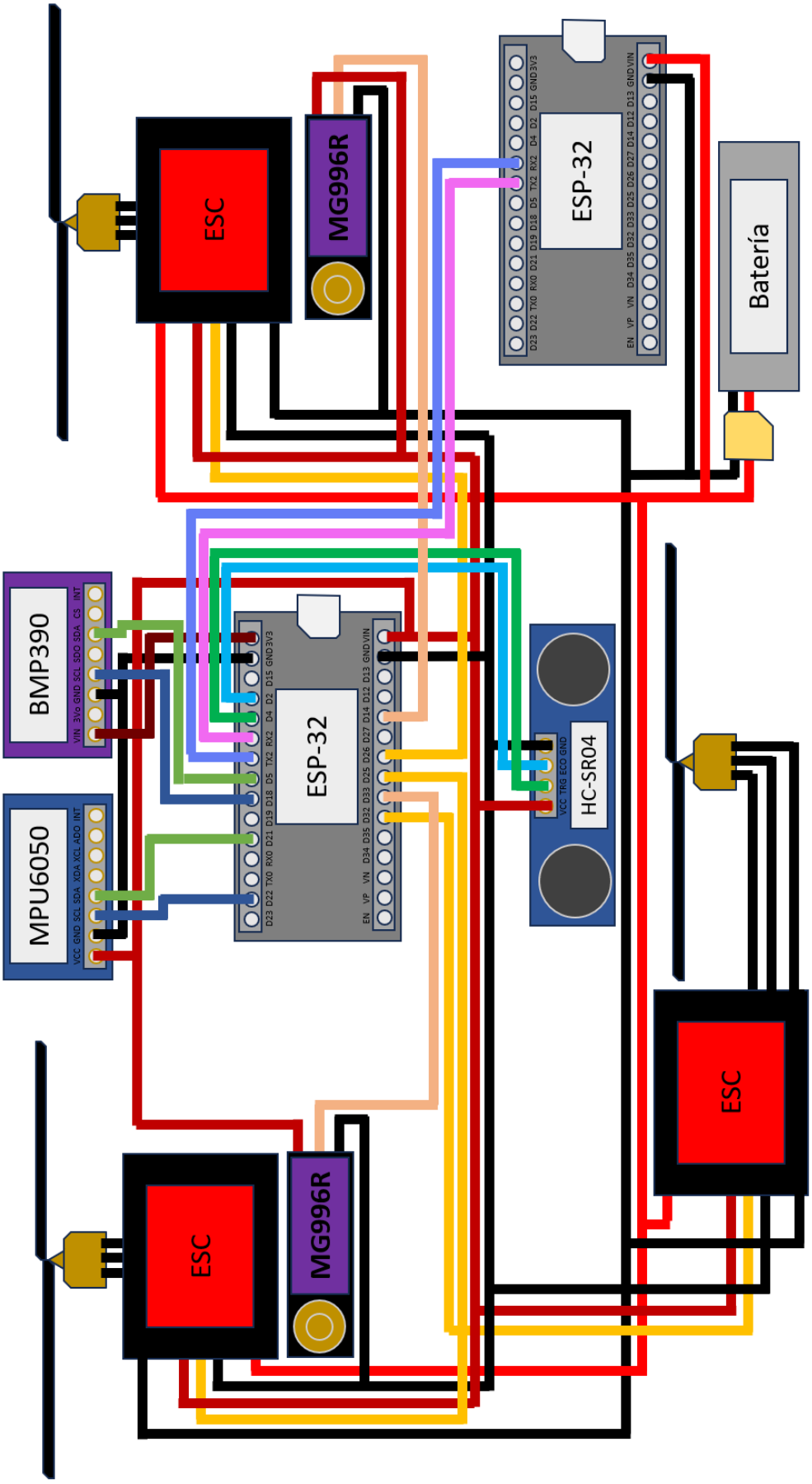
Figura 46. *Historia del diseño electrónico esquemático*

Como puede apreciarse, la alimentación principal corre a cargo de la batería, con una tensión aproximada de 12V cuando está totalmente cargada. Dicha tensión se convertirá en los ESC a 5V, tensión con la que operarán el resto de los componentes del circuito a excepción del sensor BMP390, que estará conectado al ESP32 a 3.3V.

En lo referente a la conexión de los sensores, en todos los casos utilizan dos pines, destacando la comunicación I2C (Two-Wire) para el MPU6050 y para el BMP390, dejando los pines de eco y gatillo en el sensor de ultrasonidos.

Los pulsos PWM se utilizan para el control de los ESC y de los servomotores, regulando tanto la velocidad de giro de los motores con hélices, así como la posición exacta de los servomotores.

Figura 47. *Diseño electrónico esquemático*



Como puede apreciarse en la ***Figura 47*** se ha incorporado un segundo ESP-32, la incorporación de este segundo microcontrolador se debe a la necesidad de espacio en la memoria del programa y a las capacidades incorporadas de envío de logs en remoto, aprovechando su conexión WiFi, los detalles serán discutidos durante la sección *software*.

Con la finalidad de simular la puesta en práctica se ha implementado el anterior diseño utilizando el programa Proteus, pudiendo así definir un diseño esquemático listo para su fabricación. El diseño de una posible PCB se muestra en la ***Figura 48***.

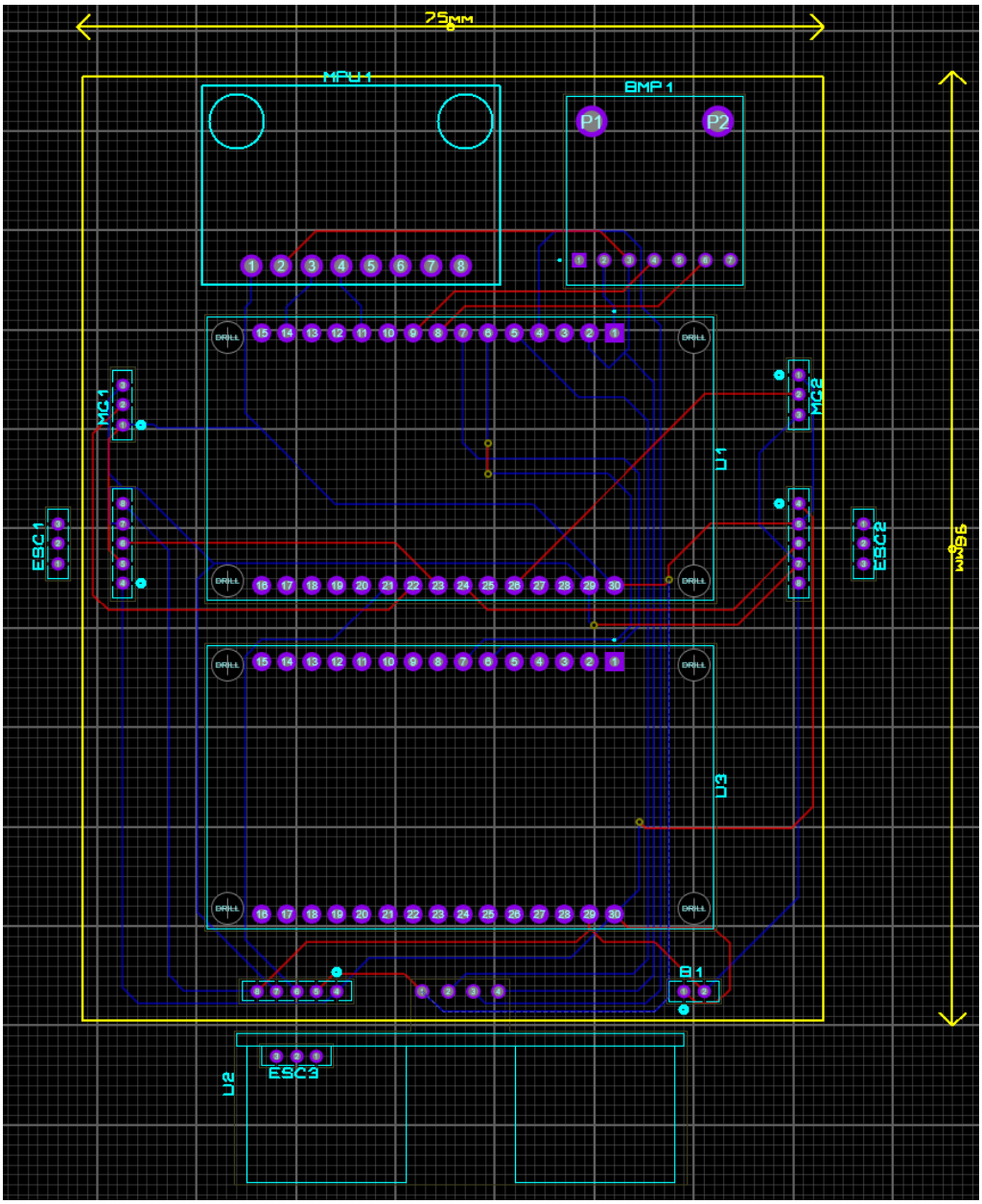


Figura 48. *Diseño PCB (layout) de la electrónica de nuestro VTOL.*

En el diseño puede apreciarse cómo las conexiones relativas a los motores se han dejado fuera de los esquemáticos, dado que, en la práctica, será más cómodo conectar los ESC a la PCB (*Printed Circuit Board*) debido a la disposición de sus cableados.

En lo referente a las conexiones se han concebido para poder incorporar de forma simple los distintos módulos electrónicos que conformarán el sistema. Este hecho produce, por un lado, que las dimensiones de la PCB aumenten significativamente, pero por otro, nos permitirán implementar pistas de mayor anchura para asegurar que no se dañan cuando trabajan con los altos consumos de potencia requeridos por los BLDC y los ESC.

Las dimensiones finales de la PCB son razonables, poseyendo unas medidas de 96X75mm, fácilmente implementables en la parte central de nuestro VTOL.

Obteniendo la visualización tridimensional, nuestra PCB se vería como se muestra en la ***Figura 49***, en esta imagen puede observarse la utilización de algunos de los troquelados con la finalidad de poder asegurar la PCB a la estructura del VTOL.

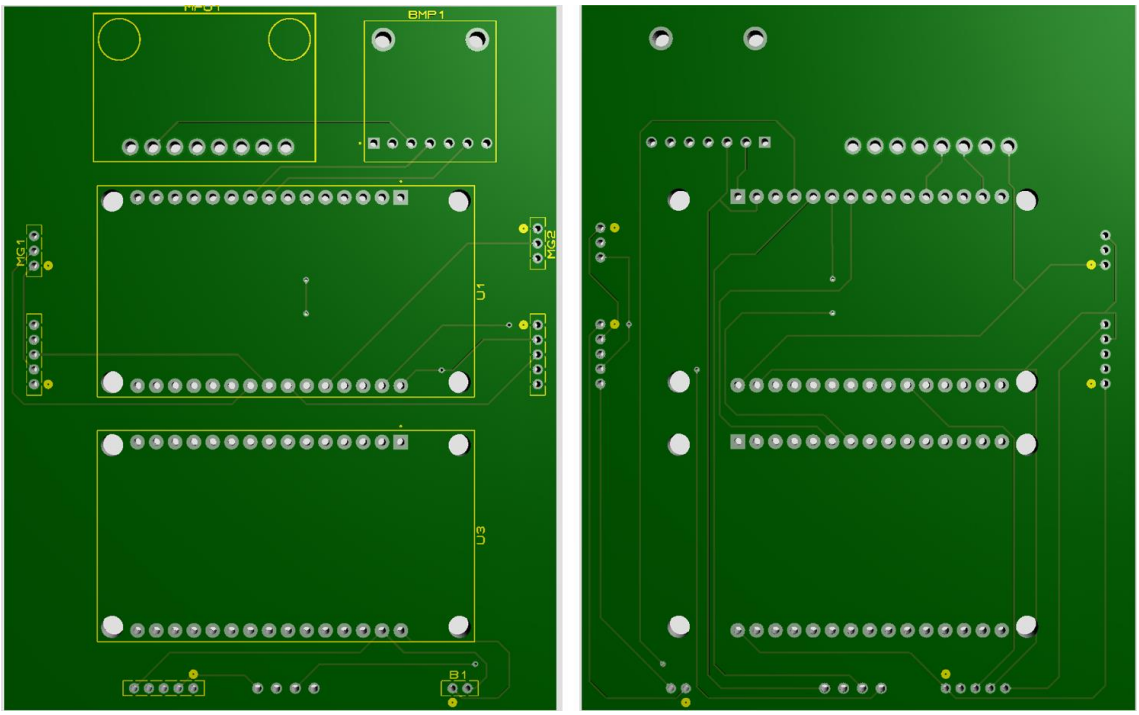


Figura 49. *Visualización tridimensional de la PCB.*

Finalmente se aporta en la ***Figura 50*** el diseño esquemático implementado con la herramienta Proteus.

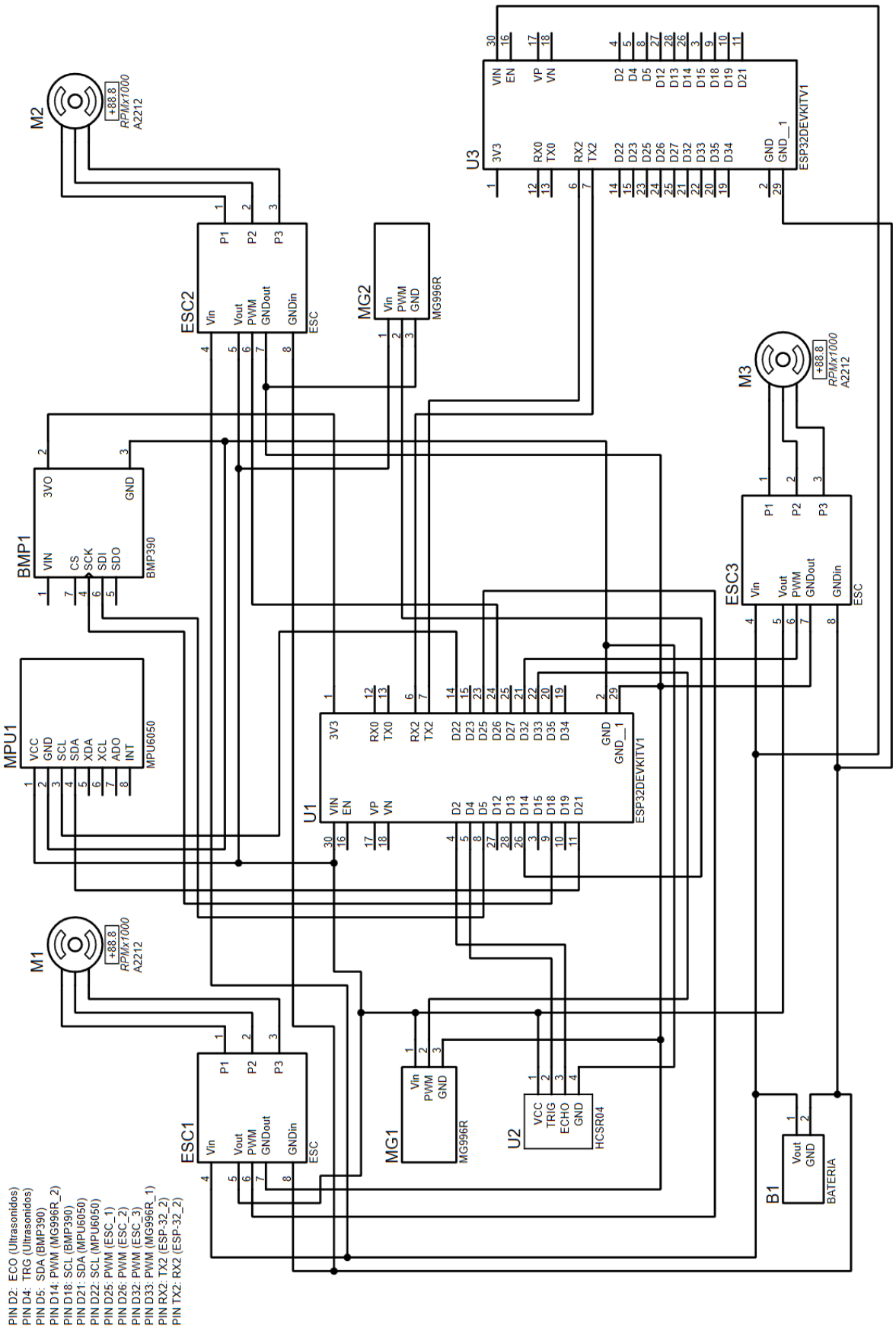


Figura 50. *Diseño esquemático implementado con Proteus.*

4. SOFTWARE

Esta sección tratará de exponer las secciones relacionadas con la configuración de entornos IDE (*Integrated Development Environment*), código fuente que expondrá la lógica del programa, bibliotecas empleadas y lenguajes de programación empleados.

Así mismo se expondrán los algoritmos utilizados, detallando su funcionamiento y los resultados prácticos obtenidos, también se tratarán de tender puentes entre la lógica del programa y la escritura y lectura de los sensores.

4.1 Instalaciones iniciales y compatibilidad

De cara a los primeros pasos, se ha tomado una decisión respecto a la elección de un IDE y lenguaje de programación. En el caso que nos ocupa, al operar con un ESP-32, se ha optado por emplear el IDE de Arduino [77], expuesto en su sección *software*. Si bien es cierto que la compañía que fabrica las PCB de ESP-32, Espressif, cuenta con su propio IDF (*IoT Development Framework*) [82], no cuenta con la extensa comunidad y soporte de bibliotecas con las que goza la plataforma Arduino. A esto se le suma el enfoque de muy bajo nivel que se toma en dicho IDF y la necesidad de controladores.

Por estas cuestiones se ha decidido utilizar el entorno Arduino en el desarrollo. Se expondrán a continuación los pasos iniciales llevados a cabo para poder establecer las primeras configuraciones del proyecto.

Estas configuraciones se han realizado siguiendo la guía de Espressif expuesta en [83]. A modo de resumen, la guía expone el repositorio de código estable en el que se incorporan las bibliotecas genéricas y los controladores de varias de las PCB ESP-32. Añadiendo dicha URL (https://espressif.github.io/arduino-esp32/package_esp32_index.json) en el administrador de placas de IDE Arduino y descargando los paquetes será suficiente para contar con la compatibilidad a nivel de controladores y bibliotecas. Dicha URL no es más que un fichero JSON que incorpora las URL en las que se encuentran los repositorios de cada una de las PCB disponibles.

En nuestro caso se ha instalado la versión 1.0.2 del controlador por compatibilidad con las bibliotecas empleadas y por necesidad ante la utilización de una PCB que actualmente se encuentra descatalogada.

4. 2 Obtención de datos y configuraciones

Una vez instalado el IDE y configuradas las bibliotecas y controladores principales de nuestro ESP-32 (Placa DOIT ESP32 DEVKIT V1) podremos empezar a escribir el código de nuestro programa principal.

Uno de los pasos iniciales es la comunicación con los sensores con la finalidad de poder obtener los datos recogidos. Ante esta casuística se han tomado dos estrategias diferentes:

- Obtener los datos directamente escribiendo en los registros para los casos en los que no existieran bibliotecas lo suficientemente óptimas o que se ajustaran a los requerimientos prácticos.
- Emplear bibliotecas desarrolladas por Arduino o la comunidad en los casos en los que la obtención de datos y comunicación no fueran críticas o las implementaciones adquiridas fueran suficientes, agilizando la implementación del código.

4.2.1 Bibliotecas empleadas

De cara al desarrollo se han empleado las siguientes bibliotecas:

- **ESP32Servo.h** [84] (versión 0.11.0): esta biblioteca nos permitirá generar pulsos PWM desde el ESP32 de forma cómoda con la finalidad de interactuar con los motores BLDC y con los servomotores.
- **Wire.h** [85] (última versión): empleada para comunicaciones directas con los sensores en crudo.
- **Ps3Controller.h** [86] (versión 1.0.0): utilizada para interactuar con un mando Bluetooth de Playstation 3, que servirá como mando de control para dirigir el vuelo del VTOL y establecer sus configuraciones de despegue.
- **Adafruit_Sensor.h** [87] (versión 1.1.6): Permitirá poder utilizar bibliotecas destinadas a la recopilación de datos del sensor BMP390.
- **Adafruit_BMP3XX.h** [56] (versión 2.1.2): Biblioteca principal para comunicación con el sensor BMP390. Requerirá la utilización de la biblioteca Adafruit BusIO [88].

4.2.2 Interacción manual con sensores

En el caso del sensor MPU6050 inicialmente se probaron distintas bibliotecas tales como [89], sin embargo, al probarlas de forma práctica en el ESP32 no funcionaban correctamente en cuanto a tiempos de procesamiento, haciendo inviable su utilización. Tras probar otras bibliotecas con resultados similares se decidió implementar las comunicaciones con el sensor de forma manual, escribiendo directamente en los registros del sensor.

Para ello se ha empleado la biblioteca Wire expuesta anteriormente, esta biblioteca se utiliza comúnmente para establecer comunicaciones I2C entre sensores desde un microcontrolador [85], además se aporta su documentación relativa a I2C [90].

Inicialmente se define e inicializa el constructor de la clase con TwoWire, llamando al objeto instanciado I2CIMU.

```
TwoWire I2CIMU = TwoWire(0);
```

Posteriormente se inicia la comunicación indicando los pines utilizados para la transmisión I2C, en nuestro caso los pines D21 y D22 con ***begin***. Tras esto deberemos iniciar la comunicación con el sensor. En su documentación se indica que la dirección I2C para su configuración en baja "*the address of the one of the devices should be b1101000 (pin AD0 is logic low)"* , pasando la dirección a hexadecimal tenemos:

```
I2CIMU.beginTransmission(0x68);
```

Posteriormente se fijarán algunos de los valores configurables del sensor como la administración de potencia, y los FSR (*Full Scale Range*) del giroscopio y el acelerómetro, así como la configuración de su LPF. Para ello tendremos que revisar la configuración de los registros del sensor, en su mapa de registros [91].

Inicialmente estableceremos el registro de potencia con todos sus valores a cero

```
  I2CIMU.beginTransmission(0x68);           //Envío la dirección del
esclavo 0x68
  I2CIMU.write(0x6B);                       //Selecciono el registro
de administración de potencia (0x6B)
  I2CIMU.write(0x00);                       //Establezco todos los
valores a cero
  I2CIMU.endTransmission(true);
```

La ***Figura 51*** muestra los bits del registro de administración de potencia, pudiendo observar que al establecer todos a cero, se ha desactivado el modo *sleep* que permite al

dispositivo entrar en modo de bajo consumo, sin desactivar el sensor de temperatura, por escalabilidad en la implementación, y seleccionando el oscilador interno del sensor.

4.28 Register 107 – Power Management 1
PWR_MGMT_1

Type: Read/Write

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

CLKSEL	Clock Source
0	Internal 8MHz oscillator
1	PLL with X axis gyroscope reference
2	PLL with Y axis gyroscope reference
3	PLL with Z axis gyroscope reference
4	PLL with external 32.768kHz reference
5	PLL with external 19.2MHz reference
6	Reserved
7	Stops the clock and keeps the timing generator in reset

Figura 51. Registro de administración de potencia MPU6050. [91]

A continuación, se ha configurado el FSR del giroscopio con una sensibilidad de ±500º/s.

4.4 Register 27 – Gyroscope Configuration
GYRO_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

FS_SEL selects the full scale range of the gyroscope outputs according to the following table.

FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s

Figura 52. *Registro de configuración del giroscopio MPU6050.* [91]

```
  I2CIMU.beginTransmission(0x68);           //Envío la dirección del
esclavo 0x68
  I2CIMU.write(0x1B);                       //Modificaré el registro
del giroscopio(0x1B)
  I2CIMU.write(0x08);                       //Escribo en el registro 0
0 0 01 0 0 0 (+-500°/s) FSR
  I2CIMU.endTransmission(true);             //Finalizo la comunicación
```

No se prescindirá de ninguno de los ejes.

Siguiendo con la configuración de los FSR, nos centraremos en el del acelerómetro, seleccionado en ±8g y nuevamente sin prescindir de ninguno de los ejes.

4.5 Register 28 – Accelerometer Configuration
ACCEL_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

AFS_SEL selects the full scale range of the accelerometer outputs according to the following table.

AFS_SEL	Full Scale Range
0	± 2*g*
1	± 4*g*
2	± 8*g*
3	± 16*g*

Figura 53. *Registro de configuración del acelerómetro MPU6050.* [91]

```
  I2CIMU.beginTransmission(0x68);           //Envío la dirección del
esclavo 0x68
  I2CIMU.write(0x1C);                       //Modifico el registro del
acelerómetro (0x1C)
  I2CIMU.write(0x10);                       //Escribo en el registro 0
0 0 10 0 0 0 (+- 8g) FSR
  I2CIMU.endTransmission(true);
```

Finalmente se seleccionarán los filtros de frecuencia para los datos. Seleccionando un LPF en 44Hz para el acelerómetro y en 42Hz para el giroscopio.

4.3 Register 26 – Configuration
CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1A	26	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		

The sampled value will be reported in place of the least significant bit in a sensor data register determined by the value of *EXT_SYNC_SET* according to the following table.

EXT_SYNC_SET	FSYNC Bit Location
0	Input disabled
1	TEMP_OUT_L[0]
2	GYRO_XOUT_L[0]
3	GYRO_YOUT_L[0]
4	GYRO_ZOUT_L[0]
5	ACCEL_XOUT_L[0]
6	ACCEL_YOUT_L[0]
7	ACCEL_ZOUT_L[0]

The DLPF is configured by *DLPF_CFG*. The accelerometer and gyroscope are filtered according to the value of *DLPF_CFG* as shown in the table below.

DLPF_CFG	Accelerometer (F_s = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

Figura 54. *Registro de configuración de LPF para los datos del MPU6050.*[91]

```
  I2CIMU.beginTransmission(0x68);           //Envío la dirección del
esclavo 0x68
  I2CIMU.write(0x1A);                       //Escribo en el registro
de configuración (0x1A)
  I2CIMU.write(0x06);                       //Introduzco 00 000 011
(Seleccionando el LPF en 44Hz para acelerómetro y 42Hz para
giroscopio)
  I2CIMU.endTransmission(true);
```

En este punto habremos establecido la configuración de nuestro sensor y nos quedará la parte relacionada con la obtención de las medidas. Para ello emplearemos los métodos de ***requestFrom*** y ***read***. Permitiéndonos obtener 6 registros de 8 bit para los datos del acelerómetro y el giróscopo. Cabe mencionar que cada uno de los valores en los ejes X, Y, Z poseen una precisión de 16 bit, por lo que para combinar los datos de los registros se ha aplicado un desplazamiento, guardando el valor completo en una variable en C.

```
  //Se solicita al sensor los datos del acelerómetro
  I2CIMU.beginTransmission(0x68);
  I2CIMU.write(0x3B); //Se solicita el registro 0x3B que corresponde
con el valor de aceleración del eje X
  I2CIMU.endTransmission(false);
  I2CIMU.requestFrom(0x68,6,true);//Se solicitan 6 registros de 8 bit,
correspondientes con los valores de aceleración de los tres ejes

  //A continuación se guarda la lectura de datos de aceleración del
IMU en m/(s^2)
  Acc_X=I2CIMU.read()<<8|I2CIMU.read(); //Son datos de 16 bit, así que
se desplazan 8 bit y se aplica la operación OR
  Acc_Y=I2CIMU.read()<<8|I2CIMU.read();
  Acc_Z=I2CIMU.read()<<8|I2CIMU.read();

  //Se obtienen los datos del giroscopio del IMU en radianes por
segundo
   I2CIMU.beginTransmission(0x68);
   I2CIMU.write(0x43); //Datos del giroscopio en el eje X
   I2CIMU.endTransmission(false);
   I2CIMU.requestFrom(0x68,6,true); //Solo se solicitan los datos de
los ejes X,Y y Z

   Gyr_X=I2CIMU.read()<<8|I2CIMU.read(); //Se vuelven a guardar los
valores desplazando y aplicando OR
   Gyr_Y=I2CIMU.read()<<8|I2CIMU.read();
   Gyr_Z=I2CIMU.read()<<8|I2CIMU.read();
```

Como puede apreciarse en el código anterior estaremos empleando los registros 0x3B y 0x43 que comienzan con los datos en el eje X respectivamente para el acelerómetro y el giróscopo [91], el resto de los datos se encuentran en registros adyacentes, con lo que iterando con ***requestFrom*** podemos obtener todos los datos.

4.3 Algoritmos y cálculos teóricos

Tras haber expuesto la configuración y obtención de datos en crudo, así como las bibliotecas empleadas durante el desarrollo, nos centraremos en la parte más técnica de procesamiento de datos, en la que se expondrán los algoritmos principales que componen la base de funcionamiento del VTOL.

4.3.1 Obtención de ángulos

El primer paso para poder determinar la posición del VTOL será obtener los ángulos a los que se encuentra este dispositivo, pudiendo obtener las inclinaciones de cada uno de sus ejes.

Frente a esta casuística nos encontramos con un problema principal, el giroscopio a medida que pasa el tiempo deja de aportar datos precisos como consecuencia del *drift*. Este *drift* se debe principalmente a la propia construcción mecánica del giroscopio, donde tenemos un "muelle" oscilante que se mueve en dos dimensiones en función de las excitaciones en un eje y debido al efecto Coriolis. Esto supone que cada uno de los movimientos como consecuencia de los rozamientos, no permitan volver a establecer la posición inicial de referencia de forma exacta. O que movimientos lentos no produzcan variaciones internas en el sensor como consecuencia de dicho rozamiento.

Matemáticamente esto puede observarse en que los datos que obtenemos del sensor son relativos a la velocidad angular, para conseguir la posición deberíamos efectuar una integral.

$$r(t) = r_0 + \int_0^t v\, dt$$

Esto produce que los términos constantes de la velocidad, al integrarlos generen un error que aumenta linealmente con el tiempo, podríamos concebirlo en nuestro sensor como un *offset* de continua [92]. Si incorporamos el concepto de frecuencia, podríamos obtener una integral similar a la siguiente:

$$\int \cos(2\pi f t) = \frac{sen(2\pi f t)}{2\pi f}$$

Cuando la frecuencia es elevada el error asociado al ruido desaparecerá totalmente con la integración, sin embargo, si la frecuencia es baja (movimientos muy lentos) el ruido se amplificará, aumentando el *drift* [93].

En la práctica, estos desajustes los obtendríamos si, al recoger los datos del sensor en radianes por segundo, multiplicamos por el tiempo en el que se obtuvieron los datos, dado que no contamos con unos valores de tiempo infinitamente precisos.

Como medida paliativa podríamos utilizar los datos del acelerómetro para contrarrestar dicho *drift*, dado que, pese a que sus mediciones debido a las vibraciones pueden ser muy inexactas, si utilizamos sus valores medios o sus valores altamente filtrados en frecuencia podremos obtener una situación viable [94].

En este caso, dado que la aceleración es un valor vectorial, podremos aplicar rotaciones matriciales con la finalidad de calcular los ángulos de Euler [95], pudiendo así obtener las componentes angulares de la aceleración que paliarán el problema del drift.

En nuestro código inicialmente se realizará una calibración de los sensores inicial, para tratar de minimizar el error ($AnguloAcc_{Xerror}$), posteriormente se obtendrán los datos del acelerómetro y se calcularán sus ángulos de Euler.

$$AnguloAcc_X = \left[atan\left(\frac{\frac{Acc_Y}{Acc_{scale}}}{\sqrt{\left(\frac{Acc_X}{Acc_{scale}}\right)^2 + \left(\frac{Acc_Z}{Acc_{scale}}\right)^2}}\right) * \frac{180}{\pi} \right] - AnguloAcc_{Xerror}$$

$$AnguloAcc_Y = \left[atan2\left(\frac{-Acc_X}{Acc_{scale}}, \frac{Acc_Z}{Acc_{scale}}\right) * \frac{180}{\pi} \right] - AnguloAcc_{Yerror}$$

Donde Acc representa los datos recogidos por el sensor y Acc_{scale} indica la escala aplicada en la configuración del sensor, en nuestro caso ±8g como se indicó en la sección anterior. Finalmente pasaremos de radianes a grados (nótese que el error ya fue transformado a grados durante la calibración).

A continuación, se obtendrán los datos del giroscopio, ajustando también su escala y se adaptará el traspaso de los ángulos *pitch* (eje Y) y *roll* (eje X) en los casos en los que haya rotación diagonal.

$$Gyro_X = Gyro_X + \left[Gyro_Y * elapsedTime * sen\left(Gyro_Z * elapsedTime * \frac{\pi}{180}\right) \right]$$

$$Gyro_Y = Gyro_Y + \left[Gyro_X * elapsedTime * sen(Gyro_Z * elapsedTime * \frac{\pi}{180})\right]$$

Donde $Gyro$ representa los datos del sensor normalizados con su escala y $elapsedTime$ el tiempo transcurrido tras la lectura de los datos del sensor, consiguiendo pasar de radianes por segundo a grados.

Finalmente se aplicará un filtro complementario donde se tendrán en cuenta principalmente los datos del giroscopio y se utilizarán los ángulos de Euler del acelerómetro para corregir dicha posición.

$$roll = 0.995 * (roll + Gyro_X) + 0.005 * AnguloAcc_X$$

$$pitch = 0.995 * (pitch + Gyro_Y) + 0.005 * AnguloAcc_Y$$

$$yaw = yaw + (Gyro_Z)$$

La utilización de este tipo de filtro lo que nos permite es tener muy en cuenta los nuevos valores del giroscopio, pero incorporando leves modificaciones en los ángulos del acelerómetro, pudiendo así evitar las vibraciones del dispositivo. En la práctica se intercambiaron los valores de *roll* y *pitch* como consecuencia de la orientación de los ejes del sensor en la implementación real.

4.3.2 PID

Una vez hemos calculado los ángulos a los que se encuentra nuestro VTOL deberemos tratar de realimentar esos datos con el objetivo de que los motores actúen ajustando el equilibrio del dispositivo. Será aquí donde entre en juego el PID.

Los PID serán los encargados de efectuar de forma práctica dicha estabilización de vuelo. De cara a su implementación, inicialmente se probó a introducir de forma directa los ángulos en el PID, tratando de ajustar sus coeficientes. Aplicando las siguientes ecuaciones genéricas para cada uno de los ángulos:

$$Ang_{error} = Ang_{actual} - Ang_{deseado}$$

$$P = K_p * Ang_{error}$$

$$I = I + (K_i * Ang_{error})$$

$$D = K_d + (\frac{Ang_{error} - Ang_{error\_previo}}{elapsedTime})$$

$$PID = P + I + D$$

Teniendo así que la parte proporcional depende directamente del error, la parte integral va acumulando los errores y la parte derivativa da cuenta de las diferencias de errores respecto al error anterior. Las constantes o coeficientes son valores que se han ido ajustando de forma práctica [96].

Esta primera implementación, en la práctica, arrojaba unos resultados válidos, pero nada estables, pudiendo mantener un vuelo, pero a duras penas, con cambios bruscos. Por este motivo se decidió emplear en el PID la velocidad angular en vez del ángulo total.

Al utilizar la velocidad angular, dado que se emplean tasas de cambio en vez de valores absolutos de ángulos las respuestas se visualizaron más rápidas, desembocando en una mayor estabilidad.

El problema de esta implementación era que, si el dispositivo al inicio del vuelo no estaba perfectamente calibrado, se recordaba esa posición como buena, volando con un plano de referencia un poco inclinado. Como decisión final se decidió implementar un procesamiento en cascada con un P y un PID. El primer proceso de estabilización trabajaría con los ángulos absolutos, mientras que el segundo ajustaría, una vez estabilizado el vuelo los valores relativos.

De este modo el primer PID pasará los valores al segundo cómo ángulo deseado con signo negativo para cada ángulo:

$$TasaAngularDeseada = -PID_{total}$$

Finalmente, el segundo PID realizará unos cálculos análogos, pero culminando con la modificación de los valores del *throttle* para cada uno de los motores:

```
pwmEscIzq = throttle - PID_total_roll - PID_total_pitch;
pwmEscDer = throttle + PID_total_roll - PID_total_pitch;
pwmEscCent = throttle + PID_total_pitch;
```

Esta implementación permitía calibrar el VTOL en una superficie plana, pero posteriormente facilitaba el despegue desde superficies inclinadas, adecuando el plano al plano de calibración gracias al primer PID absoluto. Como parte añadida, esta implementación nos permitirá inclinar de forma sencilla nuestro VTOL pudiendo conseguir que se desplace a izquierda, derecha, adelante o atrás.

4.3.2.1 Rasgos específicos del *yaw*

Dado que el VTOL en desarrollo únicamente contará con tres motores y además los dos motores de la parte delantera irán acoplados a unos servomotores se hace necesario definir algunas implementaciones clave de su desarrollo.

Al poseer un número impar de motores, se producirá de forma intrínseca un giro en el sentido opuesto al sentido de giro de dos de los motores. Para evitar dicho giro los servomotores generarán un pequeño desajuste en la alineación de los BLDC. Dicho desajuste, junto con el funcionamiento esquemático se representa en la ***Figura 55***.

La figura nos muestra el movimiento de rotación que se llevará a cabo para regular este giro. Además, estos giros nos permitirán desplazar el VTOL adelante y atrás sin necesidad de llevar a cabo ajustes adicionales en los parámetros de entrada de los PID.

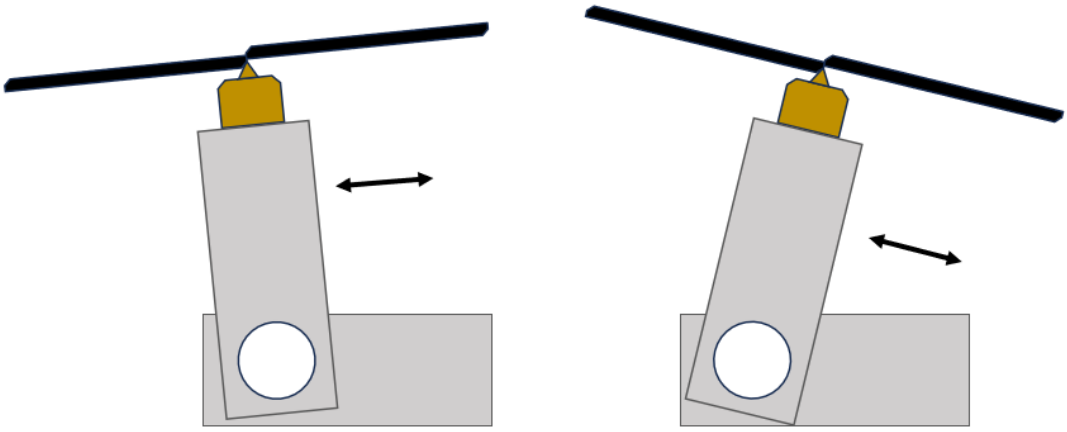


Figura 55. *Representación esquemática del movimiento de los BLDC acoplados a servomotores.*

De cara a su implementación, se ha calculado de forma práctica el desajuste necesario en el posicionamiento de los servomotores para evitar la rotación producida por tener un número de motores de propulsión impares.

En lo referente al PID se ha aplicado una equivalencia de valores que permita establecer unos límites de rango de operación, aplicando así el ajuste deseado en función de los valores de *yaw* realimentados por los sensores.

```
  //Mapeo el valor del yaw en función de la diferencia máxima de los
servos
  PID_total_yaw=map(PID_total_yaw, 0,pid_max_value_roll_pitch_yaw , 0,
1000);//Es como aplicar map de 0 a 10 pero con decimales, de 0 a 1000
/100
  PID_total_yaw/=100;
```

Finalmente ha sido necesario emplear el sensor de ultrasonidos para detectar si el vuelo ha comenzado. Para ello, cuando el VTOL se separa unos 10cm del suelo, se activa el PID del *yaw*. Es necesario realizar esta distinción, dado que, si tras el calibrado movemos el dispositivo a otra zona para el despegue, girándolo durante el proceso, se estaría incrementando la parte integral del *yaw* como consecuencia del rozamiento de la superficie, derivando en una rotación descontrolada del VTOL.

4.3.2.2 PID de altitud

Respecto al control de altura, se ha establecido un PID de forma análoga al expuesto al inicio de la sección 4.3.2, pero empleando un único PID sin conexiones en cascada.

Como principal diferencia, se han establecido constantes proporcionales no lineales, dado que, en la práctica, el VTOL no precisa de la misma potencia para ascender o descender, con lo que se han establecido restricciones en función de si el error es debido a superar la altitud deseada o si se necesita incrementar la altura.

```
    if (error_alt<10 && error_alt>-10){//Margen más estable
     kp_alt=kp_alt_ref;
    }else if(error_alt<=0){
     kp_alt=kp_alt_ref+((abs(error_alt)-10)/20);//Se genera un kp no
lineal, dado que no le cuesta lo mismo subir o bajar si tiene un
throttle más alto o bajo
                                                //Se endurece la condición;
a medida que aumenta el error también aumenta el KP.
    }else{
     kp_alt=kp_alt_ref+((abs(error_alt)-10)/15);//Además hago que se
endurezcan aún más las condiciones cuando se está por encima de la
altura deseada
    }
```

En el caso del valor aplicado al *throttle*, se realiza de forma uniforme para los tres motores.

4.3.3 Otras pruebas

Con la finalidad de arrojar luz acerca de otras alternativas probadas, se exponen algunos de los casos puestos en práctica durante el desarrollo del dispositivo. Estas pruebas en su mayoría se asocian directamente con el posicionamiento. Aunque como se ha comentado durante la sección *hardware* las pruebas fueron descartadas por viabilidad o por tecnología se aportan unas pequeñas nociones del *software* empleado.

4.3.3.1 Pruebas con GNSS

Las pruebas para determinar las distancias se llevaron a cabo realizando pruebas con dos principales bibliotecas: NeoGPS [97] y TinyGPS [98].

La primera de ellas no resultó ser funcional, debido a incompatibilidades con ESP-32 y sus versiones antiguas, por lo que se empleó la segunda, más antigua pero funcional. Se determinó que en la zona de pruebas no se podrían distinguir prácticamente más de 2-3 puntos sin mucha precisión, teniendo que mantener la posición durante un tiempo y en zonas muy extremas del área disponible.

También se llevaron a cabo pruebas sin la utilización de bibliotecas empleando un código similar al expuesto en el módulo de GPS del proyecto [96].

4.3.3.2 Pruebas con fotosensor

De cara a la utilización del ADNS3080 se empleó la biblioteca [99] con la que se pudo monitorizar movimientos hasta a 10cm de una superficie en condiciones normales y de hasta unos 50cm en condiciones de alta luminosidad. Esta biblioteca presenta un desarrollo escrito en Python [100] que permite visualizar imágenes en escala de grises tomadas por el fotosensor.

En la práctica se probó con varios sensores, incluso incorporando módulos Arduino, pero no se alcanzaba una resolución de imágenes viable para trabajar a largas distancias, a esto se le sumaba el problema de la homogeneidad de la superficie en la que se probó, que dificultaba aún más que esta alternativa fuera viable.

4.3.3.3 Posicionamiento con MPU6050

Se trataron de implementar algoritmos basados en la aceleración y el giróscopo del MPU6050 para poder determinar la posición con precisión. Para ello se implementó un algoritmo que eliminaba la componente de gravedad empleando matrices de rotación, de forma similar a como se expone en [101].

Básicamente se emplearon las matrices de rotación utilizando un vector que representaba la gravedad en cada componente $v[0,0,9.8]$ [102].

$$R_x = ((sin(\alpha) * cos(\beta)) * v[0]) + (((sin(\alpha) * sin(\beta) * sin(\gamma)) + (cos(\alpha) * cos(\gamma))) * v[1]) + (((sin(\alpha) * sin(\beta) * cos(\gamma)) - (cos(\alpha) * sin(\gamma))) * v[2])$$

$$R_y = ((cos(\alpha) * cos(\beta)) * v[0]) + (((cos(\alpha) * sin(\beta) * sin(\gamma)) - (sin(\alpha) * cos(\gamma))) * v[1]) + (((cos(\alpha) * sin(\beta) * cos(\gamma)) + (sin(\alpha) * sin(\gamma))) * v[2])$$

$$R_z = ((-sin(\beta)) * v[0]) + ((cos(\beta) * sin(\gamma)) * v[1]) + ((cos(\beta) * cos(\gamma)) * v[2])$$

A continuación, se ajustaban las componentes de aceleración en función de la rotación, restando a la aceleración obtenida del sensor la magnitud obtenida tras la rotación:

$$Acc_{X\_total} = Acc_x - R_x$$

$$Acc_{Y\_total} = Acc_y - R_y$$

Finalmente se aplicaban las ecuaciones de movimiento:

$$V = V_0 + Acc_{total} * elapsedTime$$

$$P = P_0 + \left[V * elapsedTime + \frac{aceleracion\_x * elapsedTime^2}{2} \right]$$

Debido a los problemas de realizar una doble integración, relacionados con los problemas de *drift* expuestos con anterioridad esta prueba no se pudo llevar a la puesta en marcha. Inicialmente detectaba los movimientos en la posición, pero a medida que se incrementaba el tiempo de ejecución los errores eran más elevados, añadiendo fallos al realizar cambios bruscos de sentido. Además, la matriz de rotación no lograba compensar totalmente el efecto de la gravedad, fallando solo en unas centésimas, que con el tiempo terminaban añadiendo más carga a los errores.

4.4 Explicación genérica del programa

Habiendo explicado los principales algoritmos utilizados, se dará paso a exponer de forma genérica el funcionamiento del programa.

4.1 Inicio (*setup*)

Durante el inicio del programa se ejecutará la parte estática. Antes de esta sección se declaran e inicializan las variables, así como los pines utilizados o la declaración de recursos de bibliotecas y métodos empleados.

Dentro de la sección *setup* se iniciarán las comunicaciones serie, tanto para establecer la configuración de los sensores como para exponer los datos del *log*.

Tras inicializar las comunicaciones serie principales, se establecerá la conexión con el control manual (mando) vía bluetooth.

A continuación, se iniciarán las calibraciones de los sensores MPU6050 y BMP390, estableciendo para ello varias iteraciones de cálculo de valores y realizando una media de los mismos. En el caso del BMP390 es necesario esperar para comenzar a tener en cuenta los valores, ya que tardan en estabilizarse.

Cuando finalicen las calibraciones se modificarán las luces de "jugador" del control manual y se iniciarán los motores a su mínima potencia. El dispositivo estará listo para comenzar el vuelo.

4.2 Ejecución principal (*loop*)

Después de haber realizado las configuraciones iniciales podremos comenzar a preparar el despegue. Para ello se seleccionará en el mando el modo despegue, que habilitará la detección de altitud vía ultrasonido, permitiendo detectar cuando se ha despegado de forma automática.

Se irá incrementando el *throttle* con el *joystick* izquierdo del control manual. Este control es incremental y va añadiendo valores al *throttle* por cada ciclo de ejecución que se mantiene en una dirección. Una vez se alcance el límite en el cual el dispositivo comienza a "despegarse" del suelo se activará una señal para activar los PID, pero sin activar el PID del *yaw*. Cuando se supere la altura mínima, entorno a 20cm, se activará el PID del *yaw* y comenzaremos el vuelo con todos los controles activados, menos el control de altura.

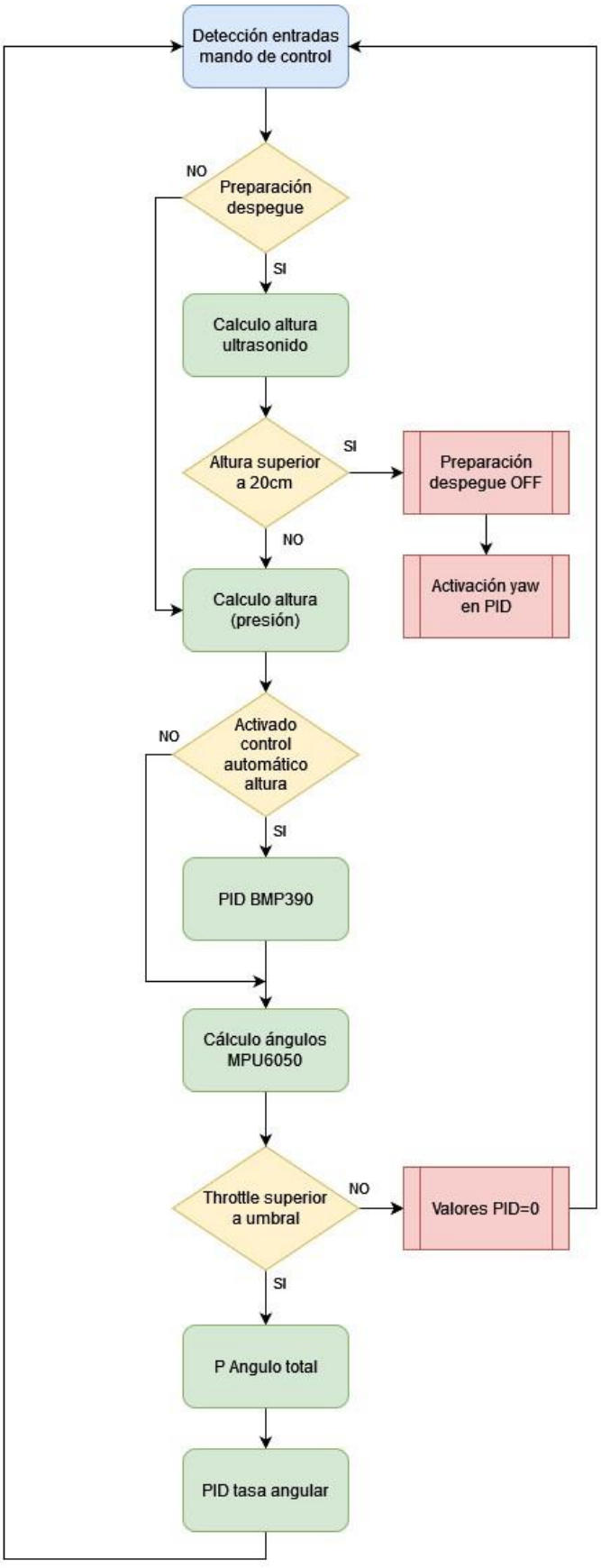


Figura 56. *Diagrama funcionamiento loop del programa VTOL*

La activación del control de altura por medio de los cálculos de presión se podrá activar desde el mando presionando el botón *X*, que cogerá la altitud de ese instante y se la insertará al PID como valor deseado. El botón *X* permitirá nuevamente desactivar el PID de altura.

En cuanto a los movimientos disponibles, se empleará el *joystick* derecho para realizar los virajes, pudiendo inclinar el VTOL para desplazarnos hacia derecha o izquierda. Este control es analógico e incrementará el giro en función de la utilización del *joystick*.

Continuando con el desplazamiento del *joystick* derecho adelante o atrás activará los servomotores delanteros, permitiendo inclinar los BLDC en una dirección u otra, también controlados de forma analógica.

Por otro lado, podremos realizar rotaciones sobre el eje central (vertical) del VTOL, empleando para ello los botones *L1* (izquierda) y *R1* derecha.

Finalmente, cuando aterricemos el dispositivo, se reiniciarán los valores de los PID y se apagarán los motores si el valor del *joystick* izquierdo llega a cero. Cabe destacar que por motivos de seguridad el control del *joystick* izquierdo es incremental en la parte máxima superior y en las partes medias inferiores, sin embargo, si se baja completamente se apagarán los motores de inmediato.

Otros mecanismos de seguridad son la disminución del *throttle* de los motores en caso de que no se detecte la conexión *bluetooth* con el mando.

La ***Figura 56*** muestra un resumen esquemático de los principales bloques del programa.

4.5 Recopilación remota de datos

De cara a la depuración durante el vuelo se ha implementado una recopilación remota de datos. Este punto es en el que se ha hecho necesaria la utilización de un segundo ESP-32.

De modo que, el ESP-32 principal se encargará de ejecutar el programa principal y se comunicará a través de la interfaz serie con un segundo ESP-32 que actuará como un AP (*AccessPoint*) emitiendo una señal WiFi e implementando un servicio REST (*Representational State Transfer*) que responderá a peticiones HTTP (*Hyper Text Transfer Protocol*) de clientes que se conecten a su misma red WiFi.

Esto permitirá que cualquier dispositivo realice peticiones HTTP al dispositivo emisor, que reportará los datos de vuelo del instante en que se realizó la petición.

Por simplicidad se ha desarrollado el código de cliente en un tercer ESP-32 que enviará las peticiones, recibirá los datos y los mostrará por la consola de un ordenador vía puerto serie, empleando el USB.

La implementación llevada a cabo se basa en dos premisas principales:

- Descargar lo máximo posible al ESP-32 que realizará los cálculos, con la finalidad de que el tiempo de ejecución del bucle sea lo más rápido posible, obteniendo así datos de mayor granularidad que realimenten los actuadores.
- Limitación en la memoria ROM del programa para el ESP-32 principal. Al tratar de enviar todo el código del programa, incluyendo el AP y el servicio REST junto con el resto del programa del VTOL se han reportado errores debido a la limitación de memoria desde el *burner* del IDE Arduino.

El código de la sección *software* se aportará de forma completa en el anexo.

5. DISEÑO

Tras haber definido las características *hardware* y *software* toca exponer cuáles serán las piezas sobre las que se ensamblará el sistema.

Al tratarse de un VTOL personalizado que únicamente cuenta con 3 motores, además dos de ellos móviles mediante servomotores, no se han encontrado estructuras prefabricadas para desarrollar el proyecto, con lo que se han implementado de forma manual, ajustadas a las necesidades del desarrollo.

En esta sección se mostrarán los materiales empleados, métodos de fabricación de la estructura y su montaje, así como los diseños personalizados implementados.

5.1 Estructura y materiales

A la hora de organizar la estructura, con la finalidad de distribuir lo máximo posible los pesos del VTOL de forma equitativa, se ha decidido implementar una configuración con ejes equiespaciados a 120º. De este modo, la distribución de los elementos partirá de un diseño simple y uniforme.

Para la implementación de los ejes se han utilizado varillas cúbicas de aluminio, huecas en su interior, debido a su relación peso-resistencia, aportando solidez a la estructura, pero sin aumentar el peso de manera significativa. Este tipo de varillas, a pesar de no ser la mejor solución aerodinámica, facilitarán la construcción, pudiendo realizar orificios de forma más sencilla al tener caras planas.

Uno de los principales problemas del diseño es que los servomotores MG996R cuentan con un peso elevado respecto a los BLDC y deberán ir en la parte delantera. Para compensar dicho desajuste se incorporó la batería en el eje trasero, determinando su posición de equilibrio.

Además, se modificó ligeramente la altura del motor trasero, situándose en un plano inferior al de los motores delanteros, para evitar desviaciones de peso cuando los servomotores situaran a los BLDC en posiciones distintas a la central.

5.2 Diseño de piezas

Tras determinar de forma genérica la estructura del VTOL, se observó la necesidad de incorporar piezas que permitieran sujetar los motores a las varillas, mantener unidas las varillas ente sí, sujetar la batería, mantener la electrónica adherida al dispositivo e implementar un mecanismo móvil que facilitara que los servomotores movieran los BLDC.

Al ser piezas tan específicas, se decidió utilizar un desarrollo tridimensional, haciendo uso de una impresora 3D que generaría las piezas empleando un plástico compuesto por PLA, que posee una rigidez suficiente para asegurar los componentes.

El diseño se ha llevado a cabo con la herramienta online ThinkerCAD[103] , a pesar de ser una herramienta simple basada en formas poliédricas básicas, permite realizar diseños de gran complejidad. Para la impresión se ha utilizado la herramienta Ultimaker Cura [104], que permitirá convertir los ficheros STL a ficheros GCODE basados en posiciones que pueda interpretar nuestra impresora 3D. Además, nos ofrecerá la posibilidad realizar multitud de configuraciones de impresión, desde las capas de adhesión hasta soportes y configuraciones geométricas de relleno de piezas.

La fabricación de las piezas se ha realizado con una impresora BIQU-B1, vendidas por el fabricante BIQU [105].

La pieza principal para comenzar la construcción será la que una las tres varillas de aluminio, situándolas en ejes de 120º. Esta pieza se muestra en la ***Figura 57***. La pieza se ha diseñado de forma simétrica y contiene orificios, que permitirán, mediante perforaciones en las varillas, introducir tornillos con tuerca para sujetar la estructura.

Sobre esta pieza se colocará el soporte para la electrónica, concretamente la PCB principal en la que se alojará la mayor parte del *hardware*, puede visualizarse en la ***Figura 58***.

La ***Figura 59*** muestra los soportes donde se acoplarán los servomotores delanteros en las varillas, estos soportes, además permitirán incorporar una pieza móvil que sujete a los BLDC en su parte superior, esta segunda pieza se muestra en la ***Figura 60***. Para el desarrollo de estas dos últimas piezas se empleó un diseño ya creado del motor MG559R en la plataforma Thingiverse [106]. La ***Figura 61*** muestra el funcionamiento de la pieza móvil, que será movida por el servo.

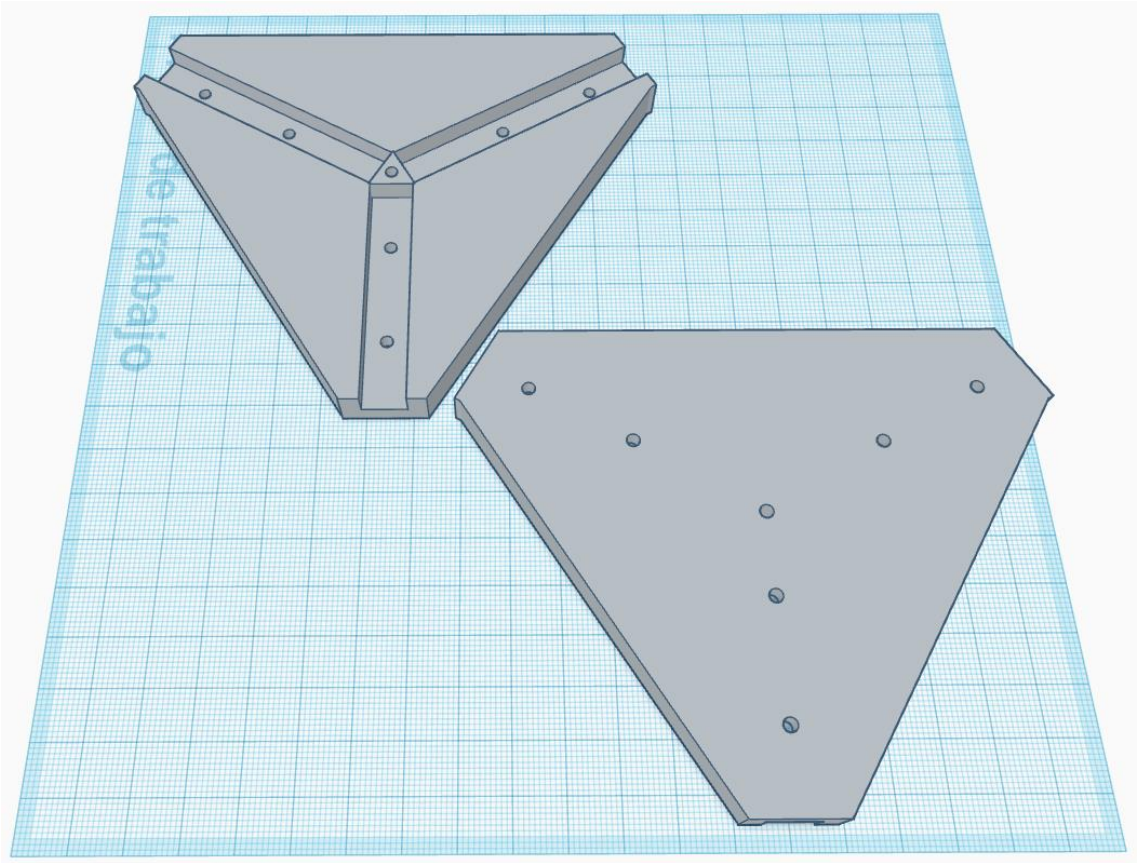


Figura 57. *Pieza central de unión de las varillas metálicas.*

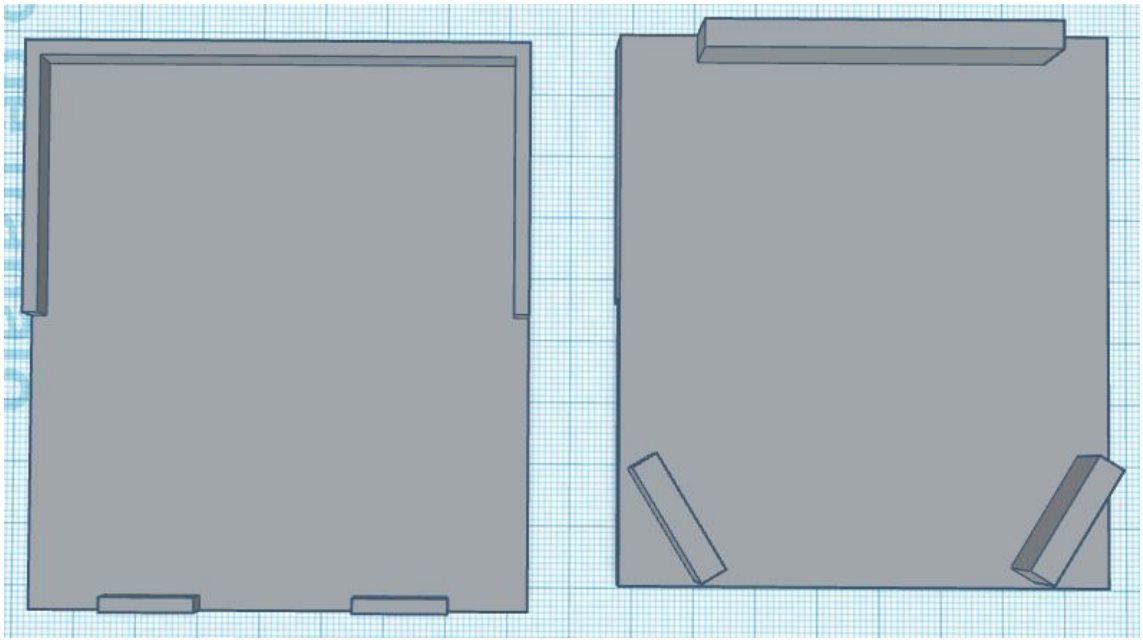


Figura 58. *Vista superior(izquierda) e inferior(derecha) se la sujeción de la PCB genérica.*

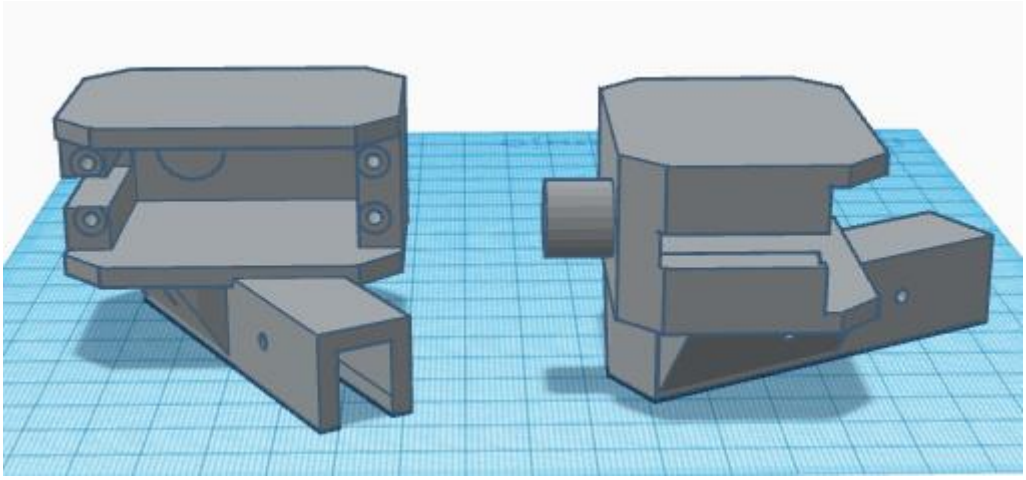


Figura 59. *Soporte para incorporar los servomotores en las varillas delanteras.*

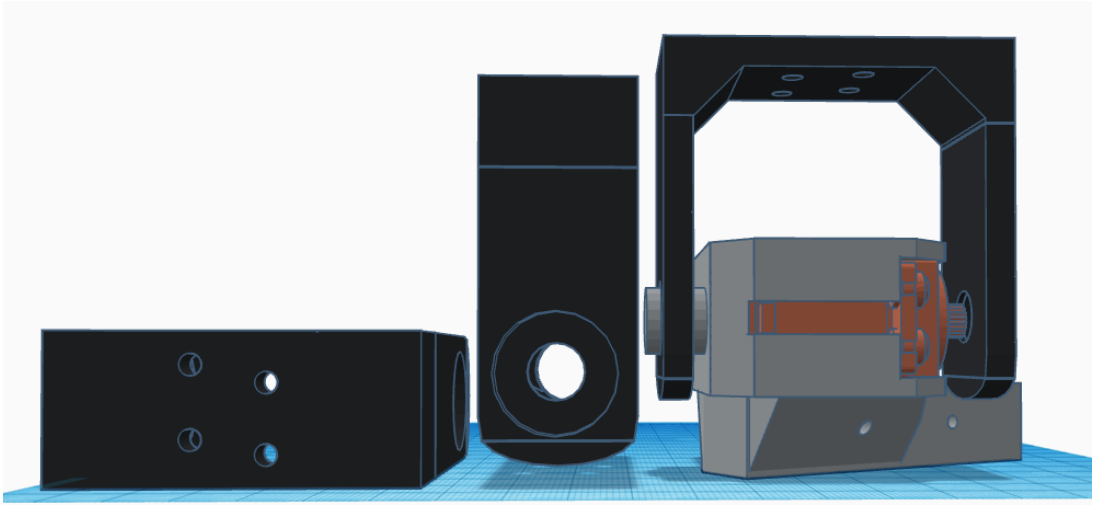


Figura 60. *Pieza móvil (izquierda) acoplada a los servomotores, para incorporar los BLDC y muestra de montaje (derecha).*

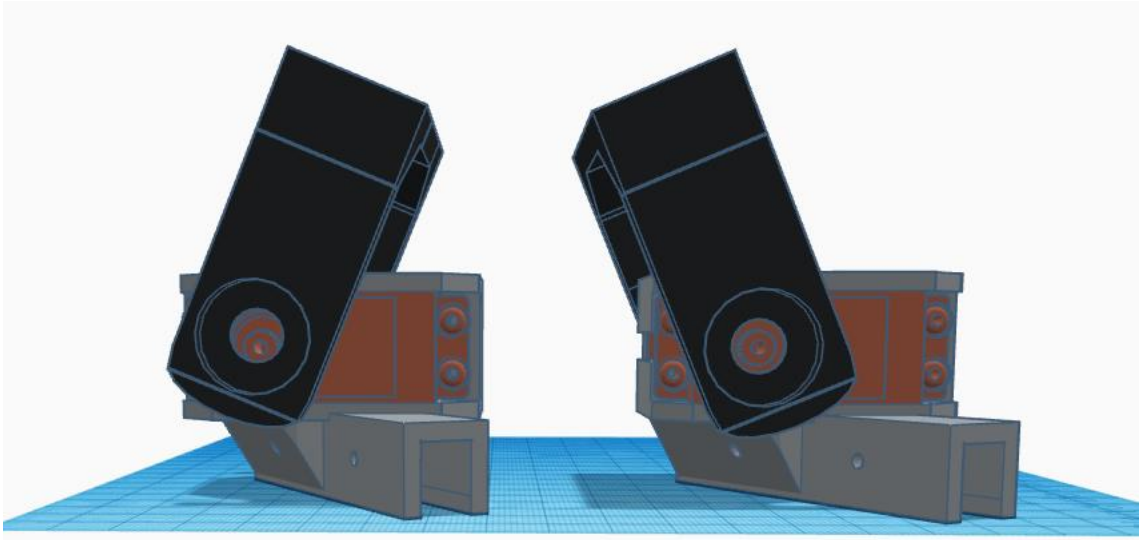


Figura 61. *Muestra de funcionamiento de la pieza móvil, incorporando el servomotor.*

Continuando con las piezas de los motores, queda mostrar la pieza del motor trasero, que irá montado en la varilla central, puede visualizarse en la siguiente figura.

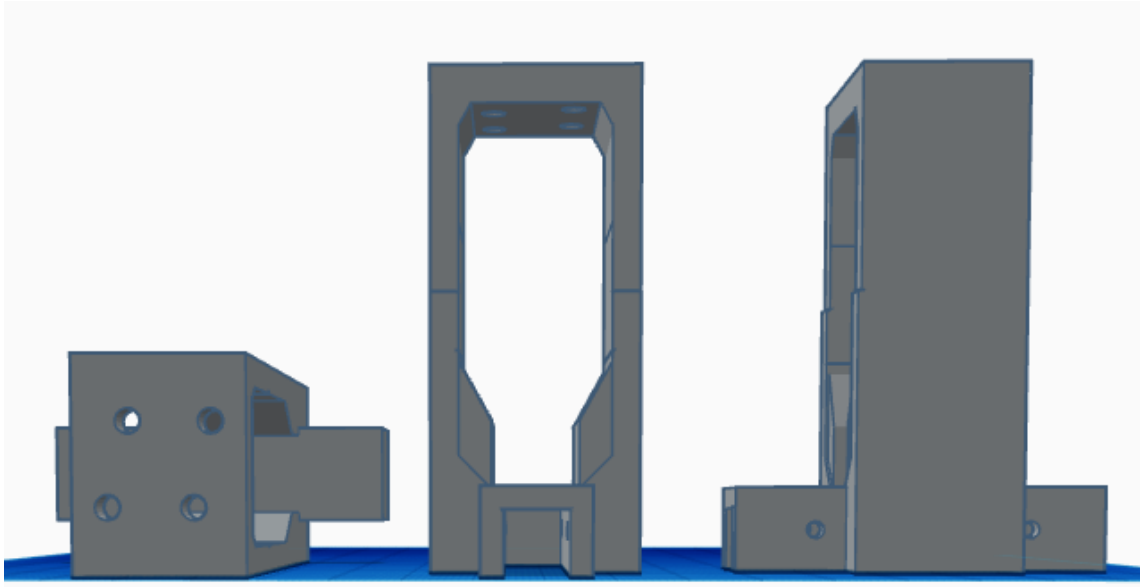


Figura 62. *Pieza del motor trasero central en múltiples vistas.*

Sobre esta misma varilla se colocará el soporte de la batería, aportado en la ***Figura 63***. Nótese que cuenta con un pequeño orificio que atraviesa la pieza, su objetivo es poder incorporar un mecanismo de sujeción adicional como una goma elástica para el prototipado, asegurando completamente la batería, pero facilitando su reemplazo para la realización de pruebas.

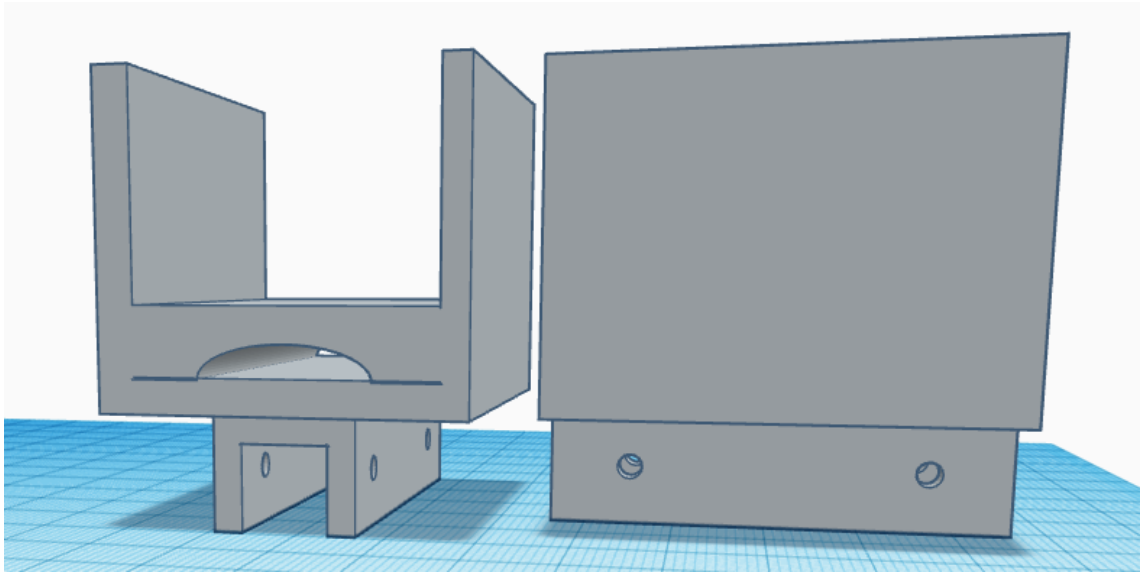


Figura 63. *Soporte de batería situado sobre la varilla central.*

A lo largo del proyecto se llevaron a cabo múltiples diseños para poder ajustar los coeficientes de los PID, configurando el *roll*, *pitch* y *yaw* por separado, para ello se diseñaron las piezas mostradas en la ***Figura 64***. Estas piezas en ocasiones se acoplaban a estructuras de madera para realizar las sujeciones.

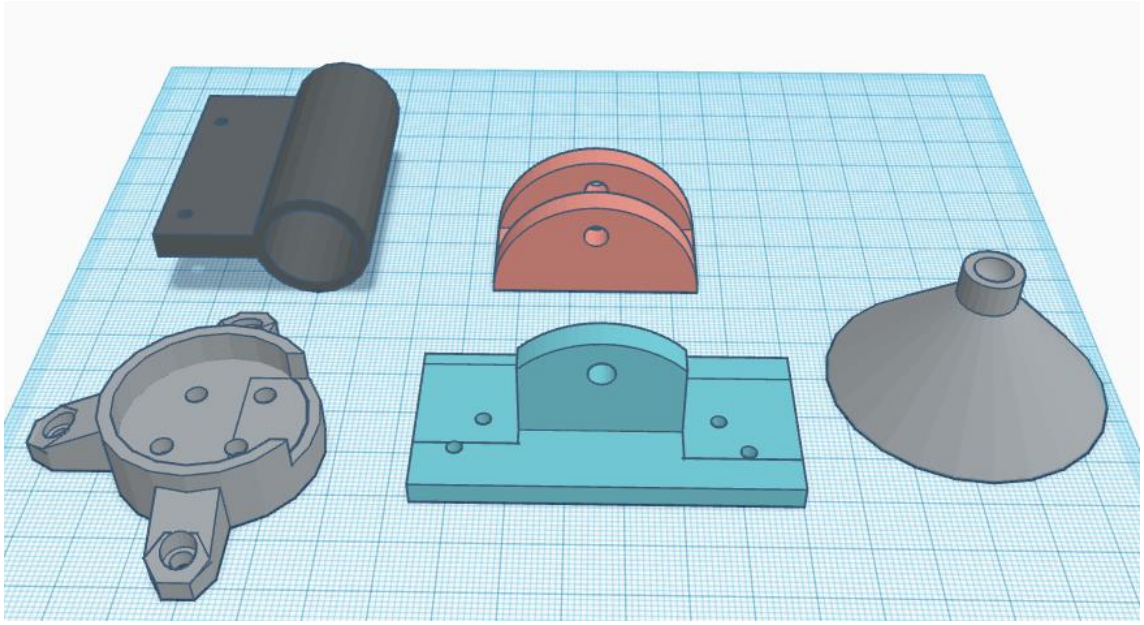


Figura 64. *Sujeciones para el cálculo práctico de los coeficientes de los PID de estabilización de vuelo ajuste del roll(izquierda), ajuste del pitch (central) y soporte de desequilibrio para roll, pitch y yaw.*

De izquierda a derecha y de abajo hacia arriba las dos primeras piezas sirvieron para calibrar el *roll* con la pieza inferior para sujetar los motores en una balanza y con la superior para introducir la varilla central solo con los motores delanteros montados.

La parte central muestra 2 piezas que combinadas permiten bloquear el *roll* y el *yaw*, permitiendo equilibrar de forma única el *pitch.*

Finalmente se calibró el *yaw* con la pieza de la derecha, utilizando un tornillo largo como punto central, incorporado en el orificio de la parte central de la pieza de la ***Figura 57***.

También se diseñaron algunos complementos como unas patas a incorporar, pero sus diseños iniciales se descartaron debido a que, tras su montaje no se conseguía una base totalmente plana e interfería a la hora de calibrar el sensor MPU6050.

Puede visualizarse en la ***Figura 65***. Estas patas provocaban además accidentes, ya que, al tener un punto de apoyo, si el VTOL caía se inclinaba, descargando toda la energía potencial del golpe en las hélices, mucho más quebradizas.

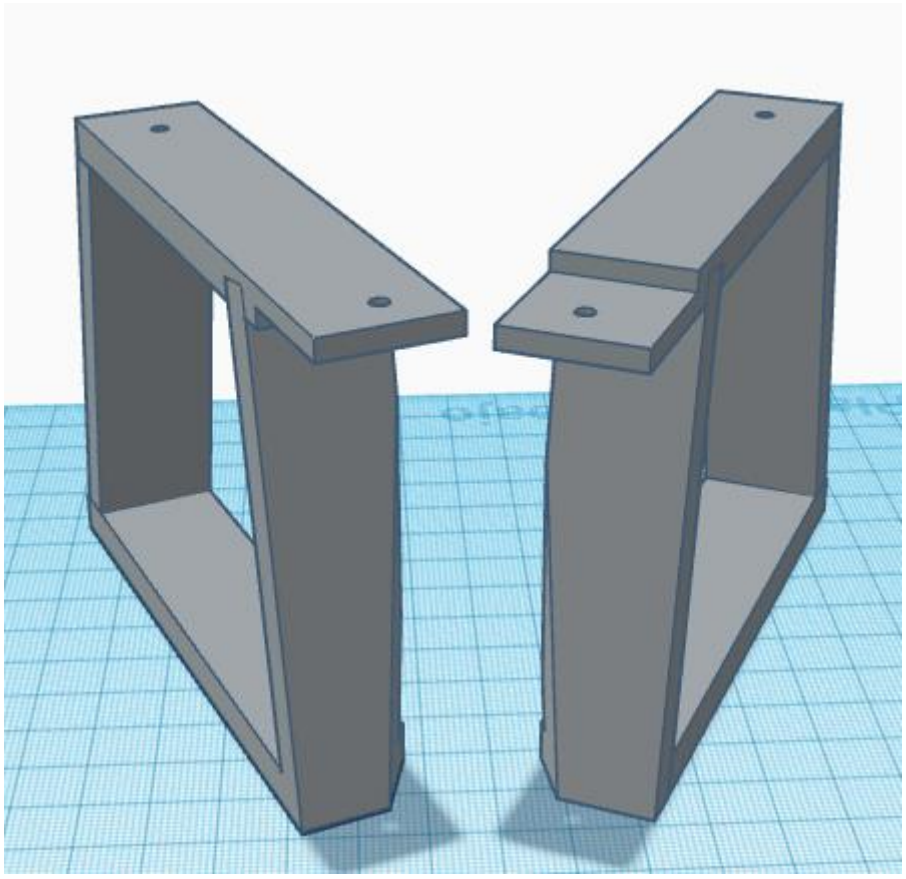


Figura 65. *Diseño de patas para el VTOL, no empleado en la implementación final.*

Se optó por diseñar unas patas de menor altura que permitieran calibrar la parte central del VTOL en una superficie completamente plana, estas se muestran en la ***Figura 66***.

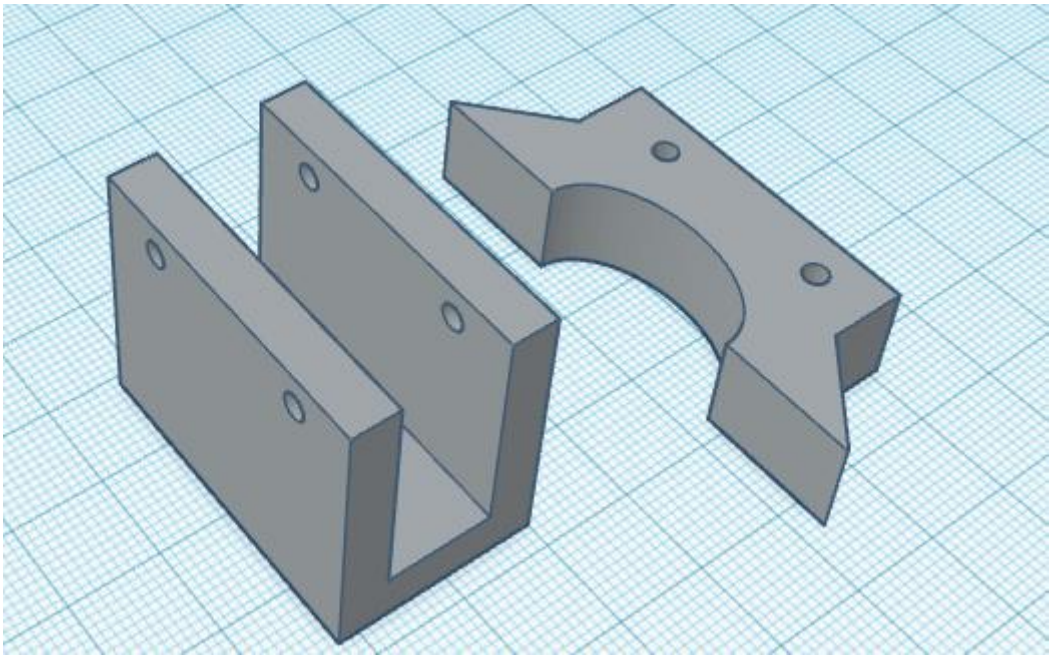


Figura 66. *Implementación final de las patas, a colocar en los extremos del VTOL.*

6. COSTES

A pesar de que la mayoría de los componentes electrónicos han sido proporcionados por la UVA (ETSIT), empleados de los disponibles en el laboratorio, se realizará una estimación de costes materiales. También se realizará una estimación de los costes humanos y la gestión de tiempos empleada durante el desarrollo de este proyecto.

6.1 Costes de la prueba de concepto y prototipado

A la hora de visualizar los costes, no se han ceñido de forma exacta a los empleados durante el desarrollo, como consecuencia de emplear materiales proporcionados por la UVA.

COMPONENTE	MODELO	CANTIDAD	PRECIO POR UNIDAD	PRECIO TOTAL	VENDEDOR
Microcontrolador	ESP-WROOM-32 DevKitv1	3	0,99	2,97	Aliexpress
Motores BLDC	A2212/13T - 1000KV	3	9,99	29,97	Amazon
ESC	ESC SIMONK30A	3	14,99	44,97	ElectroStore
Servomotores	MG996R	2	11,99	23,98	Amazon
IMU	MPU-6050	1	5,99	5,99	Amazon
Sensor de ultrasonidos	HC-SR04	1	1,8	1,8	Amazon
Sensor de presión	BMP390	1	5,39	5,39	Aliexpress
Fotosensor	ADNS3080	1	13,78	13,78	Aliexpress
GNSS	NEO-M8N	1	12,35	12,35	Aliexpress
Baterías	35c - 2200mAh	3	22,05	66,15	Amazon
Cargador de baterías	IMAX-B6	1	17,42	17,42	Amazon
Varillas	16mm x 16mm x 1m	3	6,69	20,07	Leroy Merlin
PLA de impresión	1,75mm - 1 Kg	1	24,99	24,99	Amazon
Protoboard (soldadura)	(Pack)	1	14,44	14,44	Amazon
Cables de prototipado	(Pack)	1	8,48	8,48	Amazon
Tornillos	M5 (Pack)	1	11,99	11,99	Amazon
Mando de control	PlayStation 3	1	28,54	28,54	Eneba
			TOTAL	**333,28 €**	

Figura 67. *Estimación de costes sobre los principales materiales que permiten desarrollar el proyecto.*

Los datos mostrados anteriormente corresponden con anuncios actuales que no se reflejarán en las referencias de este documento, pero serán expuestos en el anexo final. En los casos que no cuentan con total exactitud se exponen alternativas viables con un compromiso disponibilidad precio moderado.

El coste de los materiales no aporta de forma única el conformado final del VTOL, sino que incluye componentes dedicados a pruebas de concepto con la finalidad de su estudio de viabilidad.

6.2 Costes temporales

A la hora de exponer los costes temporales de la realización del proyecto se aportarán los periodos dedicados al estudio, composición y fabricación de la estructura, desarrollo electrónico, *software* y pruebas.

Cabe destacar que el coste temporal ha venido condicionado principalmente por la realización de pruebas de concepto y pruebas de vuelo. Se muestra a continuación un diagrama Gantt realizado con la herramienta GanttProject [107].

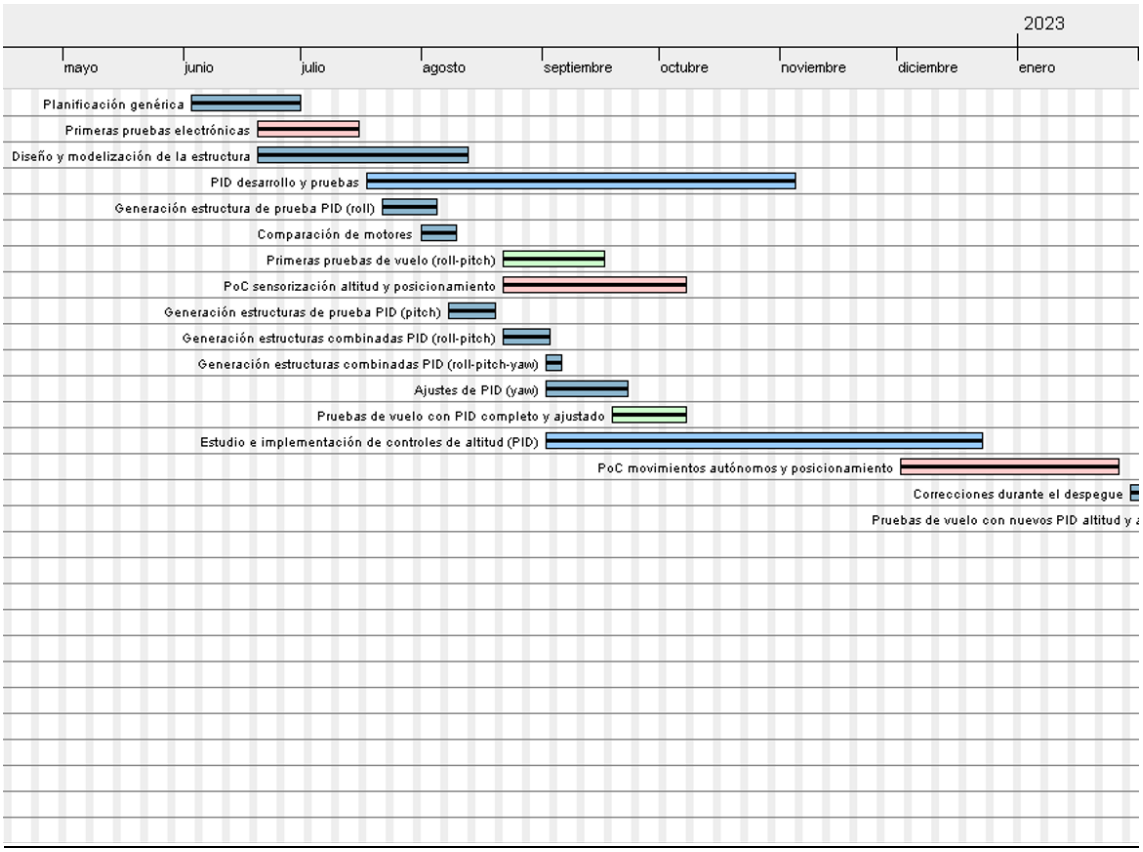


Figura 68. *Diagrama de Gantt de costes temporales (primera parte)*

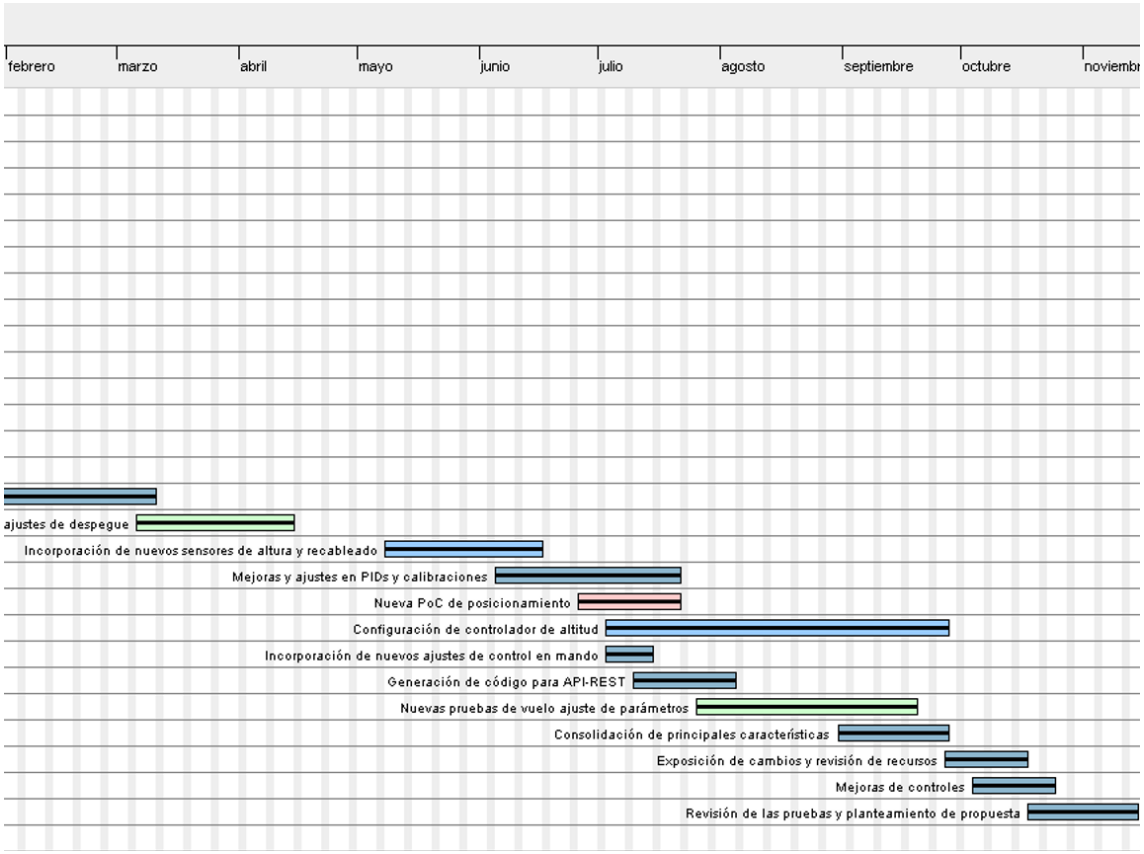


Figura 69. *Diagrama de Gantt de costes temporales (segunda parte)*

Las figuras anteriores muestran las tareas con mayor repercusión durante el proyecto, diferenciando entre pruebas de vuelo, pruebas de concepto e investigaciones y avance teórico.

El grueso del proyecto comprende un tiempo de un año y medio aproximadamente sin tener en cuenta la parte de la documentación. Durante este periodo se ha desarrollado de forma constante en función del tiempo disponible.

Las principales partes que más tiempo han consumido están relacionadas con la obtención de datos fiables de los sensores (compromiso de varianza de datos frente a obtención de datos viables), así como la realización de las pruebas asociadas a dicho funcionamiento.

Esta técnica se ha desarrollado de forma reiterada en la práctica, aplicando la base teórica matemática, realizando pruebas y volviendo a realizar nuevos ajustes en los algoritmos, con la finalidad de perfeccionar el funcionamiento y la estabilidad.

En la ***Figura 70*** se muestra un resumen textual del informe generado a partir del anterior diagrama de Gantt.

Nombre	Fecha de inicio	Fecha de fin
Planificación genérica	3/6/22	30/6/22
Primeras pruebas electrónicas	20/6/22	15/7/22
Diseño y modelización de la estructura	20/6/22	12/8/22
PID desarrollo y pruebas	18/7/22	4/11/22
Generación estructura de prueba PID (roll)	22/7/22	4/8/22
Comparación de motores	1/8/22	9/8/22
Primeras pruebas de vuelo (roll-pitch)	22/8/22	16/9/22
PoC sensorización altitud y posicionamiento	22/8/22	7/10/22
Generación estructuras de prueba PID (pitch)	8/8/22	19/8/22
Generación estructuras combinadas PID (roll-pitch)	22/8/22	2/9/22
Generación estructuras combinadas PID (roll-pitch-yaw)	2/9/22	5/9/22
Ajustes de PID (yaw)	2/9/22	22/9/22
Pruebas de vuelo con PID completo y ajustado	19/9/22	7/10/22
Estudio e implementación de controles de altitud (PID)	2/9/22	22/12/22
PoC movimientos autónomos y posicionamiento	2/12/22	26/1/23
Correcciones durante el despegue	30/1/23	10/3/23
Pruebas de vuelo con nuevos PID altitud y ajustes de despegue	6/3/23	14/4/23
Incorporación de nuevos sensores de altura y recableado	8/5/23	16/6/23
Mejoras y ajustes en PIDs y calibraciones	5/6/23	21/7/23
Nueva PoC de posicionamiento	26/6/23	21/7/23
Configuración de controlador de altitud	3/7/23	27/9/23
Incorporación de nuevos ajustes de control en mando	3/7/23	14/7/23
Generación de código para API-REST	10/7/23	4/8/23
Nuevas pruebas de vuelo ajuste de parámetros	26/7/23	19/9/23
Consolidación de principales características	31/8/23	27/9/23
Exposición de cambios y revisión de recursos	27/9/23	17/10/23
Mejoras de controles	4/10/23	24/10/23
Revisión de las pruebas y planteamiento de propuesta	18/10/23	14/11/23
Documentación del proyecto	2/1/24	10/6/24

Figura 70. *Resumen del diagrama de Gantt expuesto de forma textual.*

7. IMPLEMENTACIÓN FINAL

Se aportan en las siguientes figuras, muestras de la implementación final del prototipado.

Figura 71. *Implementación final VTOL, vista completa.*

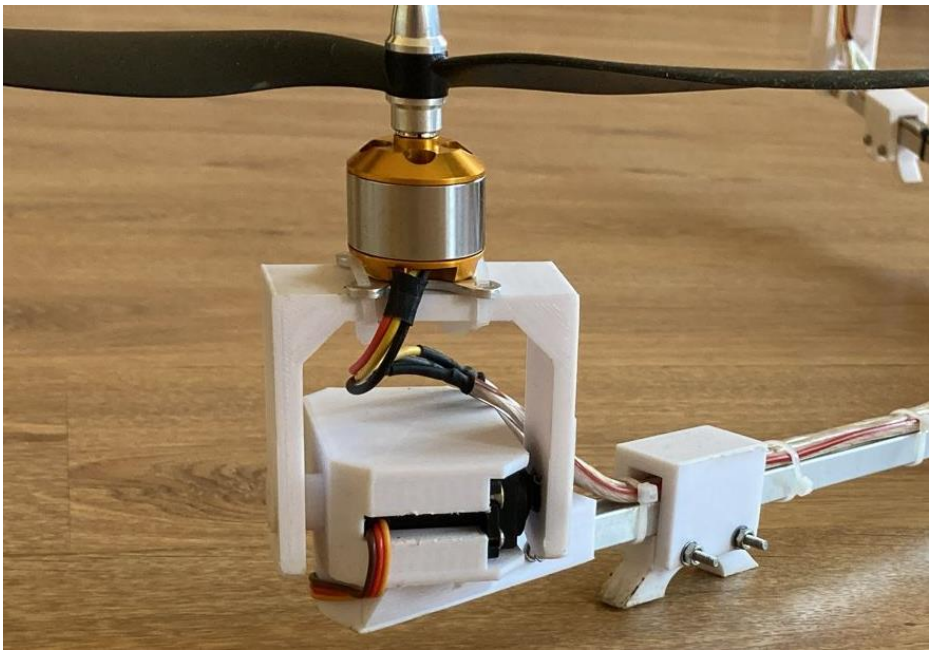


Figura 72. *Implementación final VTOL, pieza del motor delantero, parte móvil.*

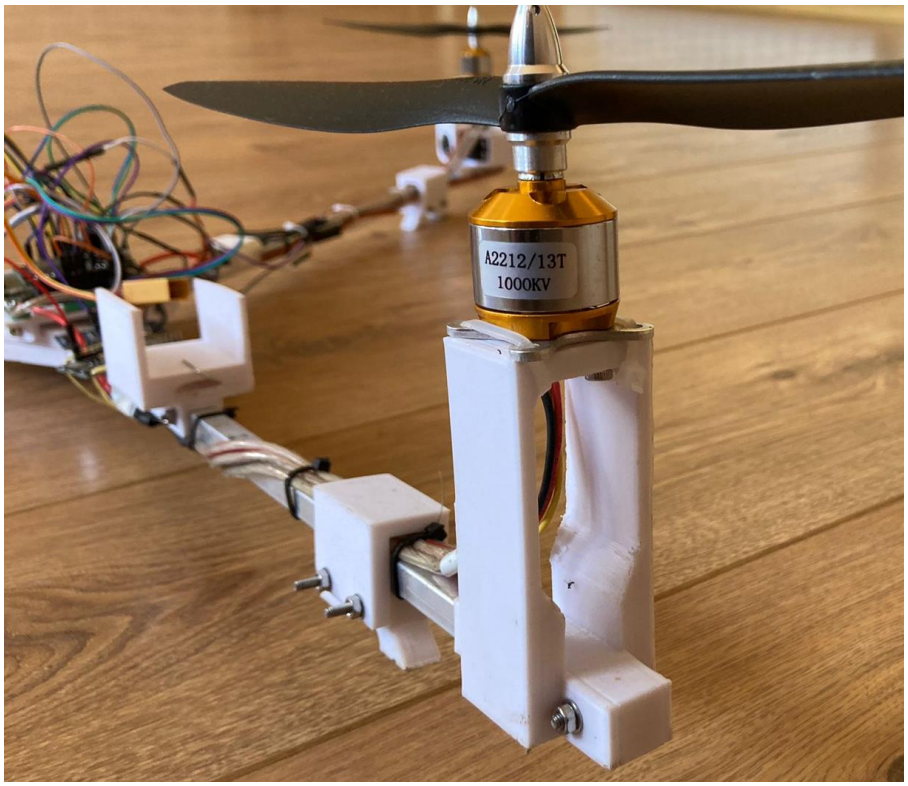


Figura 73. *Implementación final VTOL, pieza del motor trasero.*

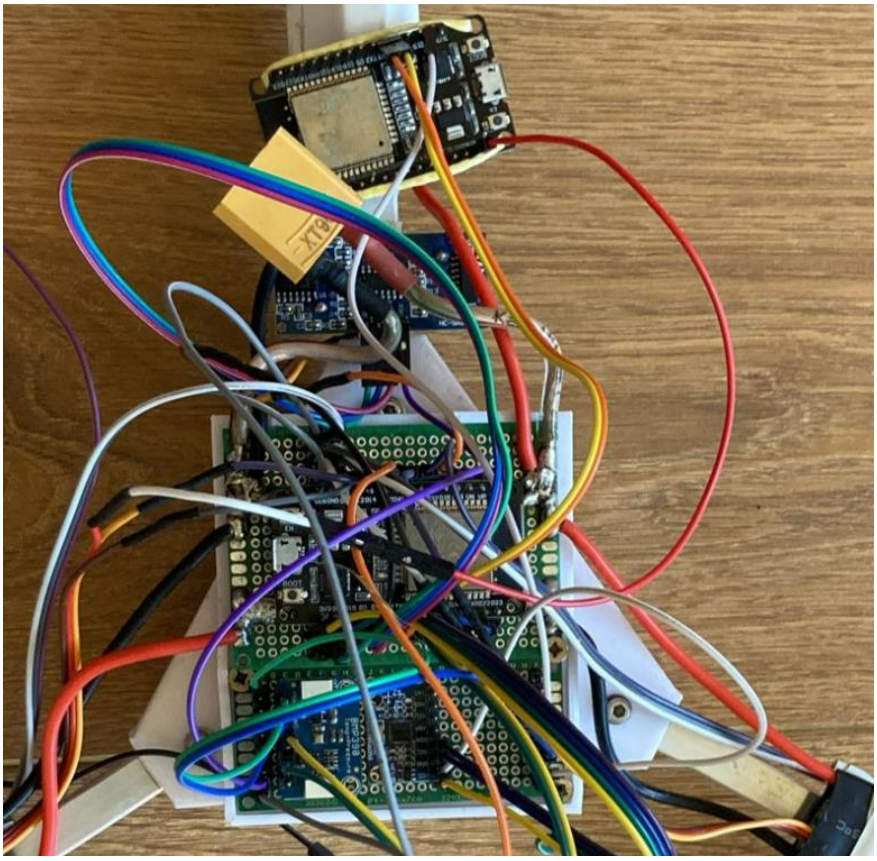


Figura 74. *Implementación final VTOL, electrónica principal.*

ANEXO

Se aportan elementos asociados al desarrollo del VTOL como el código del programa principal, el código empleado durante las pruebas de concepto y las imágenes y enlaces de compra del cálculo de costes materiales estimado.

A.1 Código principal del programa

A.1.1 Código VTOL

```
//ESP32 v1.0.2

#include <ESP32Servo.h>//Biblioteca Servo para generar pulsos PWM ESC
y servos v0.11.0
#include <Wire.h> //Comunicaciones con MPU6050
#include <Ps3Controller.h> //Conexión bluetooth con mando de PS3
v1.0.0
#include <Adafruit_Sensor.h>//Barómetro_sensor_altitud v1.1.6
#include "Adafruit_BMP3XX.h"//BMP390 (necesario Adafuit BusIO v.2.1.2)



#define SEALEVELPRESSURE_HPA
(1013.25)//https://meteologix.com/xx/observations/castilla-y-
leon/pressure-qnh/20220919-1400z.html presión nivel del mar españa
//Defino los pines SDA y SCL para I2C del sensor y así mantener cada
sensor en pines I2C aislados para un mejor rendimiento
#define I2C_SDA_BMP 5
#define I2C_SCL_BMP 18

//Para el IMU
#define I2C_SDA_IMU 21
#define I2C_SCL_IMU 22

//Para la altura de con ultrasonido
#define TRIGPIN 4
#define ECHOPIN 2
#define SOUND_SPEED 0.034


//----------Variable_Test_IMU_ONLY----------
bool test=0;
bool calibracion_altitud=1; //Controla si se utilizará el sensor de
presión en la altura (implementado porque la calibración es muy larga
para los test)
bool calibracion=1;//Conveniente calibrar cada vez que se inicie el
dron para evitar errores
int iteraciones_calibracion=10000;
bool test_motores=0;//Solo funciona si la calibración está en cero
int iteraciones_calibracion_altitud=300;//Debe ser mayor de
iteraciones_estabilizacion_calibracion
int iteraciones_estabilizacion_calibracion_altitud=150;

//----------Control_Manual_Mando_variables----------
//Pin de potencia de motores Joystick (altura)
int JoyIzq=0;
int JoyDer_servDer= 0;
```

```
int JoyDer_servIzq= 0;


//----------Motores_Biblioteca_servo_variables----------

//----------ESC----------
//Pines de PWM de los ESC
#define escPinDer (26) //Motor derecho
#define escPinIzq (25) //Motor Izquierdo
#define escPinCent (32) //Motor Central

//Declaro dos ESC que utilizarán la biblioteca ESP32Servo
Servo esc_der;
Servo esc_izq;
Servo esc_cent;

//Valores máximo y mínimo (Duty Cycle PWM) para los motores en
microsegundos
int minPWM=1000;
int maxPWM=2000;

//Valor inicial de los motores
int throttle=1200;//No se inician hasta 1200 (1350 cuasi vuelo)
//----------Servos----------
Servo servo_der;
Servo servo_izq;
int servoPinDer = 14;
int servoPinIzq = 33;
int servoCentPosDer=63;
int servoCentPosIzq=142;
int servoMinPosDer=servoCentPosDer-15;//Valores máximos y mínimos de
los servos
int servoMaxPosDer=servoCentPosDer+15;
int servoMinPosIzq=servoCentPosIzq-15;
int servoMaxPosIzq=servoCentPosIzq+15;

int servoCentPosIzqDespegue=servoCentPosIzq+8;//Creo una variable para
el despegue evitando la rotación, dado que se anula el PID del yaw,
esto hará que no gire


//----------IMU_variables----------

TwoWire I2CIMU = TwoWire(0);

//Variables para recoger lecturas del IMU
int16_t Acc_X, Acc_Y, Acc_Z,Gyr_X, Gyr_Y, Gyr_Z;
float Gyro_X, Gyro_Y, Gyro_Z, Gyr_X_error, Gyr_Y_error, Gyr_Z_error;

//Se declaran las variables para tratamiento de datos del IMU
float Acceleration_angle[2];
float Acceleration_angle_error[2];
float Total_angle[3];

//Se declaran las variables de temporizadores
//Establecidos como float para que las divisiones den todos los
decimales
double elapsedTime, timeNow, timePrev;

//Conversión de datos radianes a grados
```

```
float rad_to_deg = 180/3.141592654;

//Utilización de la velocidad algular para PID
float gyro_roll_input, gyro_pitch_input, gyro_yaw_input;

//Valores de resolucion
//Accel
//AFS_SEL=0 ±2 g  --> 16384
//AFS_SEL=1 ±4 g  --> 8192
//AFS_SEL=2 ±8 g  --> 4096
//AFS_SEL=3 ±16 g --> 2048

float acc_scale=4096;

//Gyro
//FS_SEL=0 131 LSB/(°/s) --> ±250 °/s
//FS_SEL=1 65.5 LSB/(°/s)--> ±500 °/s
//FS_SEL=2 32.8 LSB/(°/s)--> ±1000 °/s
//FS_SEL=3 16.4 LSB/(°/s)--> ±2000 °/s

float gyro_scale=65.5;


//----------Barómetro (altitud)----------

//Altitud en centímetros
float altitud;
float altitud_filtrada;
float alt_err_calibr;
float altitud_biblioteca;
int offset_altitud=200;//(está en centímetros)Permitirá volar por
debajo de la altitud de calibración con modo de altitud automática,
está en cm, permite volar 2m por debajo del nivel de calibración
float altitud_with_off=0;//Altitud con offset
bool first_read_altitud =true;//Acelerará la inicialización del filtro
//Lo dejo en 0 porque lo calibro sobre un murete de a unos 40 cm del
suelo y luego lo bajo al suelo
bool X_on=false;
bool control_altitud=false;

TwoWire I2CBMP = TwoWire(1);//Creo una instancia de TwoWire (puntero)
para el bus I2C y poner los pines que yo quiero (mirar biblioteca)
Adafruit_BMP3XX bmp;//Declaración del objeto bmp para el BMP390

//----------Ultrasonido (altitud)----------
long duration=0;
float distanceCm=0;
int distanceDm=0;
float altitud_ult=0;
bool desired_alt_fixed=false; //Se activará cuando activemos el
control de altura, para fijar la altura a la que está en ese momento


//----------PID_variables_rate_angle----------
float pid_max_value_roll_pitch_yaw=400;
float PID_proportion=1;//Se añade este valor, para ajustar el despegue
y las bajas alturas

//----------roll----------
double kp_roll=1.2;//1.2
double ki_roll=0.01;//0.015
```

```
double kd_roll=0.3;//0.2

float PID_total_roll, pwmEscIzq, pwmEscDer, error_roll,
previous_error_roll;
float pid_p_roll=0;
float pid_i_roll=0;
float pid_d_roll=0;

//----------pitch----------
double kp_pitch=1.2;//1.2
double ki_pitch=0.01;//0.01
double kd_pitch=0.3;//0.2

float PID_total_pitch,pwmEscCent, error_pitch, previous_error_pitch;
float pid_p_pitch=0;
float pid_i_pitch=0;
float pid_d_pitch=0;

//----------yaw----------
double kp_yaw=3;//3
double ki_yaw=0.06;//0.06
double kd_yaw=0.6;//0.1

float PID_total_yaw, pwmServoDer, pwmServoIzq, error_yaw,
previous_error_yaw;
float pid_p_yaw=0;
float pid_i_yaw=0;
float pid_d_yaw=0;

bool ON_yaw=false;//Esto nos permite tener un despegue suave sin que
se acumule la integral del yaw y gire descontroladamente

//Ángulo deseado
float desired_rate_angle_roll = 0;
float desired_rate_angle_pitch = 0;
float desired_rate_angle_yaw = 0;



//----------PID_variables_angle----------
//----------roll----------
float kp_angle_roll=1;
float PID_total_angle_roll,error_angle_roll;
float pid_p_angle_roll=0;

//----------pitch----------
float kp_angle_pitch=1;//4
float PID_total_angle_pitch, error_angle_pitch;
float pid_p_angle_pitch=0;

//----------yaw----------
double kp_angle_yaw=1;
float PID_total_angle_yaw, error_angle_yaw;
float pid_p_angle_yaw=0;

//Ángulo deseado
float desired_angle_roll = 0;
float desired_angle_pitch = 0;
float desired_angle_yaw = 0;
```

```
//----------PID_variables control altura----------
double kp_alt_ref=0.1;
double kp_alt=kp_alt_ref;
double ki_alt=0;//
double kd_alt=0;//

int PID_total_alt; //Tiene que ser un entero porque se va a sumar al
throttle
float error_alt, previous_error_alt;
float pid_p_alt=0;
float pid_i_alt=0;
float pid_d_alt=0;

float desired_alt = 0;//Altitud deseada en centímetros (se predefine
en 0, cuando se active se cambia a la altura que haya)
int throttle_actual =0;//Throttle de referencia por seguridad cuando
se activa el control de altitud se modifica

//Despegue en superficies con rozamiento
bool despegue_inic=false; //Se utiliza para no emplear el yaw hasta
que se eleve del suelo unos 15 cm, evitando que se acumule en la
integral el error por rozamiento
int altitud_corte_despegue=25;
bool preparacion_despegue=false;//Creado para evitar que tras calibrar
se mueva el dron y se pase la altura máxima, habrá que presionar START
en el mando para que empiece a calcular la altura
bool START_on = false; //Para el botón start en el mando


//Remotelogs para envío vía serie a un segundo ESP32 que actuará como
servidor web (API REST)
//double timeNow_serial2, timePrev_serial2, elapsedTime_serial2;
const byte numCharsLogs = 36;// 2 (<>) + 5 altitud, 5 altitud deseada,
5 error, 5 PID,4 throttle + 5X2=10 (::)
String logs;

//-------------------------------------------------DECLARACIÓN FUNCIONES-
-----------------------------------------------------
//-------------------------------------------------------------------------
-----------------------------------------------------

//Cálculo sensores altitud
void calculoAltitudRelativaCalibrado();
void computoAltitudPresion();
void computoAltitudUltraSonido();

//PID's
void ajustePidAngle();
void ajustePidRateAngle();
void ajustePidAltura();

//Control manual
void ps3Control();

//Calibración
void calculoErrorImuMedio();
void calculoAltitudRelativaCalibrado();
```

```
//-------------------------------------------------------------PROGRAMA-
-----------------------------------------------------
//-------------------------------------------------------------------------
-----------------------------------------------------


//----------SETUP----------
void setup() {
  delay(5000);//Para depurar

  //Inicializo el puerto serie para depuración de datos
  Serial.begin(9600);
  //Si hay algún tipo de error en la comunicación Serie se para
  while(!Serial) {}
  Serial2.begin(500000);


  //Se inicia la comunicación con el sensor MPU6050
  I2CIMU.begin(I2C_SDA_IMU,I2C_SCL_IMU,100000);
//Inicio la comunicación
  I2CIMU.beginTransmission(0x68);           //Envío la dirección del
esclavo 0x68
  I2CIMU.write(0x6B);                       //Selecciono el registro
de administración de potencia (0x6B)
  I2CIMU.write(0x00);                       //Establezco todos los
valores a cero
  I2CIMU.endTransmission(true);

  I2CIMU.beginTransmission(0x68);           //Envío la dirección del
esclavo 0x68
  I2CIMU.write(0x1B);                       //Modificaré el registro
del giroscopio(0x1B)
  I2CIMU.write(0x08);                       //Escribo en el registro 0
0 0 01 0 0 0 (+-500°/s) FSR
  I2CIMU.endTransmission(true);             //Finalizo la comunicación

  I2CIMU.beginTransmission(0x68);           //Envío la dirección del
esclavo 0x68
  I2CIMU.write(0x1C);                       //Modifico el registro del
acelerómetro (0x1C)
  I2CIMU.write(0x10);                       //Escribo en el registro 0
0 0 10 0 0 0 (+- 8g) FSR
  I2CIMU.endTransmission(true);

  I2CIMU.beginTransmission(0x68);           //Envío la dirección del
esclavo 0x68
  I2CIMU.write(0x1A);                       //Escribo en el registro
de configuración (0x1A)
  I2CIMU.write(0x06);                       //Introduzco 00 000 011
(Seleccionando el LPF en 44Hz para acelerometro y 42Hz para
giroscopio)
  I2CIMU.endTransmission(true);

  if (calibracion_altitud==1){
     I2CBMP.begin(I2C_SDA_BMP, I2C_SCL_BMP, 100000);//Inicializo la
comunicación I2C con las direcciones del sensor de presión

    //Se comprueba el I2C para el BMP
    if (!bmp.begin_I2C(0x77, &I2CBMP)) {   // Indico la dirección del
sensor, así como el puntero que contiene las direcciones SDA y SCL
personalizadas
```

```
      Serial.println("Could not find a valid BMP3 sensor, check
wiring!");
      while (1);
    }

      //Configuración de los valores del barómetro
      //DEJO LOS VALORES RECOMENDADOS PARA EL DRONE EN EL DATASHEET

bmp.setPressureOversampling(BMP3_OVERSAMPLING_4X);//Sobremuestreo x4
      bmp.setTemperatureOversampling(BMP3_OVERSAMPLING_2X);//Me dicen
que X1, pero la biblioteca solo tiene X2 o deshabilitado
      bmp.setIIRFilterCoeff(BMP3_IIR_FILTER_COEFF_3); //Filtro de
orden 3 para suavizar los picos un poco
      bmp.setOutputDataRate(BMP3_ODR_50_HZ);//Aumento la tasa de
muestreo a 50Hz

  }

  pinMode(TRIGPIN, OUTPUT); // Sets the trigPin as an Output
  pinMode(ECHOPIN, INPUT); // Sets the echoPin as an Input


  //Se inicia la conexión con el mando
  Ps3.begin("14:d4:24:aa:a4:bc");
  Ps3.setPlayer(1);

  if (test==0){
  //Se realizan las correspondencias de pines con los servos
  servo_der.attach(servoPinDer);
  servo_izq.attach(servoPinIzq);
  }

  //Armo los ESC (Señal de activación de los ESC un pulso PWM con un
Duty Cycle bajo)
  esc_der.attach(escPinDer, minPWM, maxPWM);
  esc_der.writeMicroseconds(1000);
  esc_izq.attach(escPinIzq, minPWM, maxPWM);
  esc_izq.writeMicroseconds(1000);
  esc_cent.attach(escPinCent, minPWM, maxPWM);
  esc_cent.writeMicroseconds(1000);
  delay(500);
  esc_der.writeMicroseconds(0);
  esc_izq.writeMicroseconds(0);
  esc_cent.writeMicroseconds(0);

  logs="<ESCs armed - iniciando calibracion>";
  for (int i=0; i<numCharsLogs;i++)
  {
      if(i<logs.length()){
      Serial2.write(logs[i]);
      }else{
      Serial2.write(' ');
      }
  }


  if (calibracion==1){
  calculoErrorImuMedio();
  if (calibracion_altitud==1){
    calculoAltitudRelativaCalibrado();
  }
```

```
  }else{
  //calibrado en superficie plana para varillas
  //Calib_angle_acc_roll:-
5.91,Calib_gyro_roll:0.90,Calib_angle_acc_pitch:-
0.22,Calib_gyro_pitch:-3.69,Calib_gyro_yaw:-0.14,

  //Patas inclinado:
  //Calib_angle_acc_roll:-
5.82,Calib_gyro_roll:0.77,Calib_angle_acc_pitch:-
8.64,Calib_gyro_pitch:-3.82,Calib_gyro_yaw:-0.09,

  //Suelo inclinado un poco ladrillo
  //Calib_angle_acc_roll:-
5.94,Calib_gyro_roll:0.91,Calib_angle_acc_pitch:-
3.58,Calib_gyro_pitch:-3.72,Calib_gyro_yaw:-0.21,

    //roll
    Acceleration_angle_error[1]=-5.94;
    Gyr_Y_error=0.91;
    //Pitch
    Acceleration_angle_error[0]=-3.58;
    Gyr_X_error=-3.72;
    //Yaw
    Gyr_Z_error=-0.21;

  }

  logs="<Calibracion OK - Inicio Programa>";
  for (int i=0; i<numCharsLogs;i++)
  {
      if(i<logs.length()){
      Serial2.write(logs[i]);
      }else{
      Serial2.write(' ');
      }
  }

delay(2000);
      Ps3.setPlayer(2);//Cuando se ha calibrado se cambia la luz del
mando para avisar


    //Se guarda el estado temporal del programa para posteriormente
pasar de °/s a °
  timeNow = millis();
}


//----------LOOP----------
void loop() {

if(test==0){//DEPURACIÓN

  //En el caso de que se pierda la conexión con el mando, se para el
programa por precaución
    if(!Ps3.isConnected()){
      throttle=1000;//Se baja el throttle a mil, para evitar daños
físicos en el VTOL, este valor es prácticamente 0
      while(true){//Se mantiene el programa en un bucle, esto está
hecho para que en caso de que se acabe la batería del mando no haya
problemas
```

```
      }
    }
}
   //Se llama a la función que ajusta el throttle e inclinación con el
mando
   //Solo modifica de forma directa el throttle, el resto de cambios
se efectúan en el segundo PID
    ps3Control();



  // computoAltitud(); De momento para pruebas seguras
  //Se calcula la altitud, si no se activa la preparación de despegue,
no se empieza a calcular la altitud
  // (evita errores al desplazar el dispositivo tras el calibrado)
   if (preparacion_despegue==true){
    if (despegue_inic==false){ //En cuanto se inicia el despegue se
deja de usar el ultrasonido
      computoAltitudUltraSonido();
    }


    if(calibracion_altitud==1){
    computoAltitudPresion();
    }
    Serial.println();
    Serial.print("Altitud: ");
    Serial.print(altitud_ult);
    if(calibracion_altitud==1){
      Serial.print(" Altitud Presion: ");
      Serial.print(altitud);
    }
    Serial.println();
   }


  //Si se presiona el botón X del mando se activa la altura automática
    if(control_altitud==true){//Determinará si se emplea el PID de
altura o no
      if(desired_alt_fixed==false){
            desired_alt=altitud_with_off;
            throttle_actual=throttle; //Se guarda el valor actual del
throttle para limitarlo por seguridad
            desired_alt_fixed=true;
      }
      ajustePidAltura();
    }else{//Si se deshabilita reinicio los valores del PID de altura
      PID_total_alt=0;
      error_alt=0;
      previous_error_alt=0;
      pid_p_alt=0;
      pid_i_alt=0;
      pid_d_alt=0;
      desired_alt_fixed=false;
    }

  //Si la altura es superior al umbral y no se ha despegado aún
significa que ya se ha conseguido el despegue, se reinician valores
del PID del yaw y se marca el despegue a true
     if(altitud_ult>altitud_corte_despegue && despegue_inic==false){
      //Se establecen los valores en cero por precaución
```

```
        PID_total_angle_yaw=0;
        error_angle_yaw=0;
        pid_p_angle_yaw=0;


        error_yaw=0;
        previous_error_yaw=0;
        PID_total_yaw=0;
        pid_p_yaw=0;
        pid_i_yaw=0;
        pid_d_yaw=0;

     //Se establece como ángulo deseado el que tenga en ese momento,
aunque no sea cero, hará que no gire descontroladamente
        desired_angle_yaw = Total_angle[2];
     //PROBLEMA, COMO NO TIENE EL PID EN EL YAW GIRA EN SENTIDO
ANTIHORARIO, SE ADELANTA EL MOTOR IZQUIERDO AL INICIO

     //Se usa la variable despegue_inic para avisar a los Servos de
que inicien con el PID
      despegue_inic=true;
     //Volver a ponerlo a false en caso en que se quiera aterrizar y
volver a despegar sin parar

      logs="<Despegue +25 cm - (PID_yaw_ON)>";
      for (int i=0; i<numCharsLogs;i++)
      {
        if(i<logs.length()){
        Serial2.write(logs[i]);
        }else{
        Serial2.write(' ');
        }
      }


     }


  //Llamada a la función de cálculos del IMU
    calculosImuAngulo();


  //Envío de datos vía puerto serie al segundo ESP32 con servidor web

logs="<"+String(altitud)+"::"+String(desired_alt)+"::"+String(error_al
t)+"::"+String(PID_total_alt)+"::"+String(throttle)+">";
    for(int i=0;i<numCharsLogs;i++){//Siempre se envían numCharsLogs
characteres, porque es lo que se espera recibir
      if(i<logs.length()){
      Serial2.write(logs[i]);
      }else{
      Serial2.write(' ');
      }
    }




  //Hasta que el throttle no alcanza un valor significativo no se
inician los PID, para evitar errores en la integral
```

```
  if (throttle>1100||test==1){
    //PID del ángulo total, que permitirá equilibrar sin importar la
posición, trata posiciones absolutas
    ajustePidAngle();
    //PID de la variación, permite una mayor estabilidad, al tratar
posiciones relativas que parten de las absolutas del PID anterior
    ajustePidRateAngle();


    }else{
    esc_der.writeMicroseconds(throttle);
    esc_izq.writeMicroseconds(throttle);
    esc_cent.writeMicroseconds(throttle);
    if(test==0 && despegue_inic == false){//DEPURACIÓN si se activa el
test se evitan errores de consumo de potencia al no conectar la
bateria
    servo_der.write(servoCentPosDer);
    servo_izq.write(servoCentPosIzqDespegue);
    }else if (test==0 && despegue_inic == true){
    servo_der.write(servoCentPosDer);
    servo_izq.write(servoCentPosIzq);
    }


    //Se reinician los valores cuando se vuelve a cero, para que no
haya errores
    //PID angle
      PID_total_angle_roll=0;
      error_angle_roll=0;
      pid_p_angle_roll=0;

      PID_total_angle_pitch=0;
      error_angle_pitch=0;
      pid_p_angle_pitch=0;


      PID_total_angle_yaw=0;
      error_angle_yaw=0;
      pid_p_angle_yaw=0;


    //PID rate angle
     //Roll
     error_roll=0;
     previous_error_roll=0;
     PID_total_roll=0;
     pid_p_roll=0;
     pid_i_roll=0;
     pid_d_roll=0;
     //Pitch
     error_pitch=0;
     previous_error_pitch=0;
     PID_total_pitch=0;
     pid_p_pitch=0;
     pid_i_pitch=0;
     pid_d_pitch=0;
     //Yaw
     error_yaw=0;
     previous_error_yaw=0;
     PID_total_yaw=0;
     pid_p_yaw=0;
```

```
      pid_i_yaw=0;
      pid_d_yaw=0;

  }

}//---FIN LOOP---

//-------------------------------------------------------------
FUNCIONES-----------------------------------------------------
//-------------------------------------------------------------------------
-------------------------------------------------

////////////////////////////////////////////////////////////////////////
/
//////////////////     CALCULOS SENSORES
//////////////////////
////////////////////////////////////////////////////////////////////////
/

//----------calculosImuAngulo----------
void calculosImuAngulo(){
  //Se guarda el tiempo de ejecución para poder pasar de °/s a °
  //Se guardará el estado anterior del ciclo temporal y el actual,
para ver cuanto
  //tiempo ha pasado.
  timePrev = timeNow;
  timeNow = millis();
  elapsedTime = (timeNow - timePrev)/1000;

  //Se solicita al sensor los datos del acelerómetro
  I2CIMU.beginTransmission(0x68);
  I2CIMU.write(0x3B); //Se solicita el registro 0x3B que corresponde
con el valor de aceleración del eje X
  I2CIMU.endTransmission(false);
  I2CIMU.requestFrom(0x68,6,true);//Se solicitan 6 registros de 8 bit,
correspondientes con los valores de aceleración de los tres ejes

  //A continuación se guarda la lectura de datos de aceleración del
IMU en m/(s^2)
  Acc_X=I2CIMU.read()<<8|I2CIMU.read(); //Son datos de 16 bit, así que
se desplazan 8 bit y se aplica la operación OR
  Acc_Y=I2CIMU.read()<<8|I2CIMU.read();
  Acc_Z=I2CIMU.read()<<8|I2CIMU.read();

  //Se calcula en ángulo de aceleración aplicando la ecuación de Euler
y se pasa a grados
  //Aquí también se aplican las escalas en función de la resolución
deseada
  //(giro en X)
  //No es necesario aplicar atan2, dado que la solución aportará un
resultado entre -90 y 90
  Acceleration_angle[0] =
(atan((Acc_Y/acc_scale)/sqrt(pow((Acc_X/acc_scale),2) +
pow((Acc_Z/acc_scale),2)))*rad_to_deg)-Acceleration_angle_error[0];
  //(giro en Y)
  //Se utiliza atan2, porque de esta forma obtenemos el cuadrante en
el que se situa, eliminando el problema
  //de ángulos desde -180 a 180
  Acceleration_angle[1] = (atan2((-
Acc_X/acc_scale),(Acc_Z/acc_scale))*rad_to_deg)-
Acceleration_angle_error[1];
```

```
  //(giro en Z)
  //No se pueden utilizar solo las ecuaciones de EULER para
calcularlo, se empleará solo el giroscopio

  //Se obtienen los datos del giroscopio del IMU en radianes por
segundo
   I2CIMU.beginTransmission(0x68);
   I2CIMU.write(0x43); //Datos del giroscopio en el eje X
   I2CIMU.endTransmission(false);
   I2CIMU.requestFrom(0x68,6,true); //Solo se solicitan los datos de
los ejes X,Y y Z

   Gyr_X=I2CIMU.read()<<8|I2CIMU.read(); //Se vuelven a guardar los
valores desplazando y aplicando OR
   Gyr_Y=I2CIMU.read()<<8|I2CIMU.read();
   Gyr_Z=I2CIMU.read()<<8|I2CIMU.read();

   //Se aplican las escalas en función de la resolución de giro
deseada
   //Se meten los float
   Gyro_X = Gyr_X/gyro_scale;
   Gyro_Y = Gyr_Y/gyro_scale;
   Gyro_Z = Gyr_Z/gyro_scale;

  //Se corrige el error medio
  Gyro_X = Gyro_X-Gyr_X_error;
  Gyro_Y = Gyro_Y-Gyr_Y_error;
  Gyro_Z = Gyro_Z-Gyr_Z_error;

  //Utilizo la velocidad angular para el PID
  gyro_roll_input = (gyro_roll_input * 0.7) + (Gyro_Y  * 0.3);
//Gyro pid input is deg/sec.
  gyro_pitch_input = (gyro_pitch_input * 0.7) + (Gyro_X  * 0.3);//Gyro
pid input is deg/sec.
  gyro_yaw_input = (gyro_yaw_input * 0.7) + (Gyro_Z  * 0.3);//Gyro pid
input is deg/sec.


  //traspaso el ángulo de pitch a roll si hay rotacion
  Gyro_X+= (Gyro_Y*elapsedTime) * sin((Gyro_Z*elapsedTime) *
(1/rad_to_deg));
  Gyro_Y-= (Gyro_X*elapsedTime) * sin((Gyro_Z*elapsedTime) *
(1/rad_to_deg));



  //Se calcula el ángulo total, tomando los valores del giroscopio y
el acelerómetro
  //Se pasan los datos del giroscopio de grados/seg a seg
multiplicando por el tiempo transcurrido desde la lectura
  //Estas variables implementan un filtro complementario que aportará
mayor precisión
  //Los coeficientes se han ajustado en base a un compromiso entre
respuesta rápida y suavidad de los cambios en la respuesta
  //Ángulo en eje X
  Total_angle[0] = 0.995*(Total_angle[0]+(Gyro_X*elapsedTime)) +
0.005*Acceleration_angle[0];
  //Ángulo en eje Y
  Total_angle[1] = 0.995*(Total_angle[1]+(Gyro_Y*elapsedTime)) +
0.005*Acceleration_angle[1];
  //Paso los datos del Gyr_Z a grados
```

```
  Total_angle[2]=Total_angle[2]+(Gyro_Z*elapsedTime);
}



//---------------computoAltitudPresion()--------------
void computoAltitudPresion(){
    if (! bmp.performReading()) {
    Serial.println("Failed to perform reading :(");
    return;
  }

  //Cojo los datos y le resto el error de la calibración
  altitud_biblioteca=bmp.readAltitude(SEALEVELPRESSURE_HPA)-
alt_err_calibr;

  if (first_read_altitud==true){//Acelero la estabilización del filtro

    altitud_filtrada=altitud_biblioteca;
    first_read_altitud=false;
  }


  //Filtro solo con la altitud: (filtrado para que sea rápido y con un
poco de ruido)
  altitud_filtrada=altitud_filtrada*0.9+altitud_biblioteca*0.1;


  altitud=altitud_filtrada;

  //Se supondrá que la altura nunca puede ser negativa, para eso se ha
introducido un offset, no hace falta recortarla
  //Pero sí hace falta un offset
  if (altitud>0){
    altitud=altitud*100;//La paso a centímetros
  }

  if (altitud_biblioteca>0){
    altitud_biblioteca=altitud_biblioteca*100;
    }

  altitud_with_off=offset_altitud+altitud;//Genero la altitud con
offset (Siempre tiene que haber offset, porque no se trunca la
altitud)

  //El dato usable será altitud_with_off, ya que la otra altitud puede
ser negativa

  //Depuración
  /*
  Serial.print("Altitud:");
  Serial.print(altitud);
  Serial.print(",");
  Serial.print("Altitud_biblioteca:");
  Serial.print(altitud_biblioteca);
  Serial.println("");
  */
```

```
}


//---------------computoAltitudUltraSonido()--------------
void computoAltitudUltraSonido(){
// Se limpia el pin del trigger
  digitalWrite(TRIGPIN, LOW);
  delayMicroseconds(2);
  // Se pone el pin del trigger 10us en alta y luego se baja
  digitalWrite(TRIGPIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGPIN, LOW);

  // Se lee el rebote de la onda ultrasónica
  duration = pulseIn(ECHOPIN, HIGH);

  // Se calcula la distancia
  distanceCm = round(duration * SOUND_SPEED/2);//Se trunca para evitar
saltos y malas medidas

  altitud_ult=round((altitud_ult*0.7+distanceCm*0.3)*10)/10;

  //altitud_ult=distanceCm;


}


////////////////////////////////////////////////////////////////////////
/
//////////////////       CALCULOS PIDs
/////////////////////
////////////////////////////////////////////////////////////////////////
/

//----------ajustePidAngle()----------
void ajustePidAngle(){//Es solo P, pero lo he llamado PID :)
  //Primero se calcula el error entre el ángulo obtenido y el deseado
  //Según la posición del sensor el ángulo de giro en X del sensor
regulará el pitch y el ángulo Y del sensor el ROLL
  error_angle_roll = Total_angle[1] - desired_angle_roll;
  error_angle_pitch = Total_angle[0] - desired_angle_pitch;
  error_angle_yaw = Total_angle[2] - desired_angle_yaw;

  //Después se añade el valor proporcional
  pid_p_angle_roll = kp_angle_roll*error_angle_roll;
  pid_p_angle_pitch = kp_angle_pitch*error_angle_pitch;
  pid_p_angle_yaw = kp_angle_yaw*error_angle_yaw;


  //Se realiza la suma de las tres partes
   PID_total_angle_roll = pid_p_angle_roll;
   PID_total_angle_pitch = pid_p_angle_pitch;
   PID_total_angle_yaw = pid_p_angle_yaw;


  //Se establecen límites de salida
  if(PID_total_angle_roll < -pid_max_value_roll_pitch_yaw)
  {
    PID_total_angle_roll=-pid_max_value_roll_pitch_yaw;
```

```
  }
  if(PID_total_angle_roll > pid_max_value_roll_pitch_yaw)
  {
    PID_total_angle_roll=pid_max_value_roll_pitch_yaw;
  }

  if(PID_total_angle_pitch < -pid_max_value_roll_pitch_yaw)
  {
    PID_total_angle_pitch=-pid_max_value_roll_pitch_yaw;
  }
  if(PID_total_angle_pitch > pid_max_value_roll_pitch_yaw)
  {
    PID_total_angle_pitch=pid_max_value_roll_pitch_yaw;
  }

  if(PID_total_angle_yaw < -pid_max_value_roll_pitch_yaw)
  {
    PID_total_angle_yaw=-pid_max_value_roll_pitch_yaw;
  }
  if(PID_total_angle_yaw > pid_max_value_roll_pitch_yaw)
  {
    PID_total_angle_yaw=pid_max_value_roll_pitch_yaw;
  }



      //Finalmente modifico la tasa deseada del PID rate angle, con la
finalidad de anidar los PID
  //Se iguala en negativo
  desired_rate_angle_roll = -PID_total_angle_roll;
  desired_rate_angle_pitch = -PID_total_angle_pitch;
  desired_rate_angle_yaw   = -PID_total_angle_yaw;


}



//----------ajustePidRateAngle()----------
void ajustePidRateAngle(){



  //Primero se calcula el error entre el ángulo obtenido y el deseado
  //Según la velocidad angular en X del sensor regulará el pitch y la
velocidad angular de Y del sensor el ROLL
  error_roll = gyro_roll_input - desired_rate_angle_roll;
  error_pitch = gyro_pitch_input - desired_rate_angle_pitch;
  error_yaw = gyro_yaw_input - desired_rate_angle_yaw;


  //Después se añade el valor proporcional
  pid_p_roll = kp_roll*error_roll;
  pid_p_pitch = kp_pitch*error_pitch;
  pid_p_yaw = kp_yaw*error_yaw;

  //Se añade el valor integral, esto permitirá paliar los errores
estacionarios
  pid_i_roll = pid_i_roll+(ki_roll*error_roll);
  pid_i_pitch = pid_i_pitch+(ki_pitch*error_pitch);
  pid_i_yaw = pid_i_yaw+(ki_yaw*error_yaw);
```

```
  //Se calcula la parte derivativa, respecto al tiempo que ha pasado
desde la
  //lectura del sensor se calcula la diferencia del error respecto al
tiempo (derivada del error)
   pid_d_roll = kd_roll*((error_roll -
previous_error_roll)/elapsedTime);
   pid_d_pitch = kd_pitch*((error_pitch -
previous_error_pitch)/elapsedTime);
   pid_d_yaw = kd_yaw*((error_yaw - previous_error_yaw)/elapsedTime);

  //Se realiza la suma de las tres partes
  PID_total_roll = pid_p_roll + pid_i_roll + pid_d_roll;
  PID_total_pitch = pid_p_pitch + pid_i_pitch + pid_d_pitch;
  PID_total_yaw = pid_p_yaw + pid_i_yaw + pid_d_yaw;

  //Se realiza un escalado proporcional del PID, para ajustar sus
valores a baja altitud
  PID_total_roll=PID_proportion*PID_total_roll;
  PID_total_pitch=PID_proportion*PID_total_pitch;
  PID_total_yaw=PID_proportion*PID_total_yaw;


  //Se establecen límites de salida
  if(PID_total_roll < -pid_max_value_roll_pitch_yaw)
  {
    PID_total_roll=-pid_max_value_roll_pitch_yaw;
  }
  if(PID_total_roll > pid_max_value_roll_pitch_yaw)
  {
    PID_total_roll=pid_max_value_roll_pitch_yaw;
  }

  if(PID_total_pitch < -pid_max_value_roll_pitch_yaw)
  {
    PID_total_pitch=-pid_max_value_roll_pitch_yaw;
  }
  if(PID_total_pitch > pid_max_value_roll_pitch_yaw)
  {
    PID_total_pitch=pid_max_value_roll_pitch_yaw;
  }

  if(PID_total_yaw < -pid_max_value_roll_pitch_yaw)
  {
    PID_total_yaw=-pid_max_value_roll_pitch_yaw;
  }
  if(PID_total_yaw > pid_max_value_roll_pitch_yaw)
  {
    PID_total_yaw=pid_max_value_roll_pitch_yaw;
  }


  //Mapeo el valor del yaw en función de la diferencia máxima de los
servos
  PID_total_yaw=map(PID_total_yaw, 0,pid_max_value_roll_pitch_yaw , 0,
1000);//Es como aplicar map de 0 a 10 pero con decimales, de 0 a 1000
/100
  PID_total_yaw/=100;
```

```
  //Finalmente se realiza el cálculo de los motores y se truncan a los
límites
  //El roll únicamente se gestiona con los motores delanterios,
mientras que en el pitch actúan los 3

  if (throttle==1000){//En el caso de cancelar el throttle con el
mando, no actúal el PID
    pwmEscIzq=1000;
    pwmEscDer=1000;
    pwmEscCent=1000;
    pwmServoDer=servoCentPosDer;
    pwmServoIzq=servoCentPosIzqDespegue;
  }else{
  pwmEscIzq = throttle - PID_total_roll - PID_total_pitch;
  pwmEscDer = throttle + PID_total_roll - PID_total_pitch;
  pwmEscCent = throttle + PID_total_pitch;
  //Se le suman los valores del joystic (alante y atrás)
  if (test==0){//Para poder probar enchufado sin que se cuelgue
  pwmServoDer = JoyDer_servDer - PID_total_yaw;
  pwmServoIzq = JoyDer_servIzq + PID_total_yaw;
  }
  }


  //Derecha
  if(pwmEscDer < minPWM)
  {
     pwmEscDer= minPWM;
  }
  if(pwmEscDer > maxPWM)
  {
     pwmEscDer=maxPWM;
  }
  //Izquierda
  if(pwmEscIzq < minPWM)
  {
    pwmEscIzq= minPWM;
  }
  if(pwmEscIzq > maxPWM)
  {
    pwmEscIzq=maxPWM;
  }
  //Central
  if(pwmEscCent < minPWM)
  {
     pwmEscCent= minPWM;
  }
  if(pwmEscCent > maxPWM)
  {
     pwmEscCent=maxPWM;
  }
  //Servo der
  if (pwmServoDer<servoMinPosDer){
    pwmServoDer=servoMinPosDer;
  }
  if (pwmServoDer>servoMaxPosDer){
    pwmServoDer=servoMaxPosDer;
  }
  //Servo izq
  if (pwmServoIzq<servoMinPosIzq){
```

```
    pwmServoIzq=servoMinPosIzq;
  }
  if (pwmServoIzq>servoMaxPosIzq){
    pwmServoIzq=servoMaxPosIzq;
  }

  //Se escriben los valores en los motores
    esc_der.writeMicroseconds(pwmEscDer);
    esc_izq.writeMicroseconds(pwmEscIzq);
    esc_cent.writeMicroseconds(pwmEscCent);

    if (despegue_inic==true){ //Si el despegue ha terminado escribe
los valores del PID
      servo_der.write(pwmServoDer);
      servo_izq.write(pwmServoIzq);
    }else{ //Si no ha terminado se mantienen las posiciones de
despegue que evitan la rotación
      servo_der.write(servoCentPosDer);
      servo_izq.write(servoCentPosIzqDespegue);
    }

  //Se guarda el valor del error anterior
  previous_error_roll = error_roll;
  previous_error_pitch = error_pitch;
  previous_error_yaw = error_yaw;

}


//----------ajustePidAltura()----------
void ajustePidAltura(){

  //Habrá que tener cuidado porque el error puede ser decimal y si
sumamos un valor decimal al throttle no se incrementará porque es un
entero
  //la solución pasa por poner la altura en cm


  //Primero se calcula el error entre la altura obtenida y deseada
  error_alt = altitud_with_off - desired_alt;

    if (error_alt<10 && error_alt>-10){//Margen más estable
     kp_alt=kp_alt_ref;
    }else if(error_alt<=0){
     kp_alt=kp_alt_ref+((abs(error_alt)-10)/20);//Se genera un kp no
lineal, dado que no le cuesta lo mismo subir o bajar si tiene un
throttle más alto o bajo,
                                               //Se endurece la condición a
medida que aumenta el error también aumenta el KP.
    }else{
     kp_alt=kp_alt_ref+((abs(error_alt)-10)/15);//Además hago que se
endurezcan aún más las condiciones cuando se está por encima de la
altura deseada
    }

  //Después se añade el valor proporcional
  pid_p_alt = kp_alt*error_alt;

  //Se añade el valor integral, esto permitirá paliar los errores
estacionarios
  pid_i_alt = pid_i_alt+(ki_alt*error_alt);
```

```
  //Se calcula la parte derivativa, respecto al tiempo que ha pasado
desde la
  //lectura del sensor se calcula la diferencia del error respecto al
tiempo (derivada del error)
   pid_d_alt = kd_alt*(error_alt - previous_error_alt);


  //Se realiza la suma de las tres partes
  PID_total_alt = round(pid_p_alt + pid_i_alt + pid_d_alt);

  //Se limitan los valores totales del PID entre 3 y -3 por seguridad
  if(PID_total_alt < -3)
  {
    PID_total_alt=-3;
  }
  if(PID_total_alt > 3)
  {
    PID_total_alt=3;
  }

  //Se establecen limitaciones de throttle, respecto al throttle
existente cuando se activo
  if(throttle-PID_total_alt > throttle_actual+20){
    throttle=throttle_actual+20;
  }else if(throttle-PID_total_alt < throttle_actual-20){
    throttle=throttle_actual-20;
  }

  //Se suma al throttle el ajuste del PID de altura
  throttle = throttle - PID_total_alt;

  //Se limita el throttle por precaución
    if(throttle < 1000)
  {
     throttle = 1000;
  }
  if(throttle > 1780)
  {
     throttle = 1780;
  }


  //Depuración
  Serial.println();
  Serial.print(" Altura: ");
  Serial.print(altitud);
  Serial.print(" Altura con offset: ");
  Serial.print(altitud_with_off);
  Serial.print (" Altitud deseada: ");
  Serial.print(desired_alt);
  Serial.print (" Error altura: ");
  Serial.print(error_alt);
  Serial.print (" PID altura: ");
  Serial.print(PID_total_alt);
  Serial.print (" Trhottle: ");
  Serial.print(throttle);
  Serial.println();
```

```
   previous_error_alt = error_alt;

}



////////////////////////////////////////////////////////////////////////
/
//////////////////   CONFIGURACIÓN MANDO
//////////////////////
////////////////////////////////////////////////////////////////////////
/


void ps3Control(){

 //---Joystick_Izquierdo---
    if(((Ps3.data.analog.stick.ly*-1)<50) &&
((Ps3.data.analog.stick.ly*-1)>-50)){
    JoyIzq=0;//Si está en la posición central no modifico el throttle
  }else if(((Ps3.data.analog.stick.ly*-1)>50)){
    JoyIzq=1; //Si subo el Joystick incremento en 1 el throttle en
cada iteración
  }else if(((Ps3.data.analog.stick.ly*-1)<-126)){
    JoyIzq=-5;
  }else{
    JoyIzq=-1; //Si bajo el Joystick decremento en 1 el throttle en
cada iteración
  }

  //Ajusto la elevación en función del Joystick
  //Establezco límites, si bajo el Joystick hasta abajo para
inmediatamente, si lo bajo un poco resta hasta llegar al mínimo, si lo
subo aumenta el throttle hasta 1800, si no lo toco sumará cero y se
mantiene

    if(JoyIzq==-5){
      throttle=1000;
    }else if (throttle+JoyIzq>1799){
      throttle=1800;
    }else if (throttle+JoyIzq<1001){
     throttle=1000;//Lo que hace es parar los motores
    }else if (throttle>1000 && throttle<1400){//Para conseguir un
despegue rápido
      throttle+=50*JoyIzq;
    }else{//Si no se pasa de los límites ni para funciona
incrementando o decrementando el throttle
      throttle+=JoyIzq;
    }

/*
  //Depuración
  Serial.print("JoyIzq=");
  Serial.print(JoyIzq);
  Serial.println("");
  */


  //---Joystick_Derecho---
```

```
    //Motor derecho
  if(((Ps3.data.analog.stick.ry*-1)<50) &&
((Ps3.data.analog.stick.ry*-1)>-50)){
    JoyDer_servDer=servoCentPosDer; //Se ha medido el desfase en
grados entre motores (37 grados) y se ha buscado de forma práctica el
punto medio
  }else if(((Ps3.data.analog.stick.ry*-1)>50)){
    JoyDer_servDer=map(Ps3.data.analog.stick.ry*-1, 50, 128,
servoCentPosDer, servoMaxPosDer);
  }else{
    JoyDer_servDer=map(Ps3.data.analog.stick.ry*-1, -128, -50,
servoMinPosDer, servoCentPosDer);
  }

    //Depuración
  /*
  Serial.print("   JoyDer_servDer=");
  Serial.print(JoyDer_servDer);
 */

  //Motor izquierdo
  if(((Ps3.data.analog.stick.ry*-1)<50) &&
((Ps3.data.analog.stick.ry*-1)>-50)){
    JoyDer_servIzq=servoCentPosIzq;
  }else if(((Ps3.data.analog.stick.ry*-1)>50)){
    JoyDer_servIzq=map(Ps3.data.analog.stick.ry*-1, 50, 128,
servoCentPosIzq, servoMaxPosIzq);
  }else{
    JoyDer_servIzq=map(Ps3.data.analog.stick.ry*-1, -128, -50,
servoMinPosIzq ,servoCentPosIzq);
  }

    //Depuración
  /*
  Serial.print("   JoyDer_servIzq=");
  Serial.println(JoyDer_servIzq);
  */

  //Movimientos
  //Virar izquierda y derecha
  if(((Ps3.data.analog.stick.rx*-1)<50) &&
((Ps3.data.analog.stick.rx*-1)>-50)){
    desired_angle_roll=0;
  }else if(((Ps3.data.analog.stick.rx*-1)>50)){
    desired_angle_roll=-10;
  }else{
    desired_angle_roll=10;
  }


  //Rotación
  //Rotación hacia la derecha
    if( Ps3.data.analog.button.r1>5 ){
        desired_angle_yaw-=1;
     }
  //Rotación hacia la izquierda
    if( Ps3.data.analog.button.l1>5 ){
        desired_angle_yaw+=1;
    }
```

```
if( Ps3.data.button.start ){
            Serial.println("Pressing the START button");
        }

  //---Activación despegue---
  if (Ps3.data.button.start==true  && START_on==false){
      preparacion_despegue=!preparacion_despegue;
      if (preparacion_despegue){
        logs="<Preparacion Despegue Iniciada>";
        for (int i=0; i<numCharsLogs;i++)
         {
            if(i<logs.length()){
            Serial2.write(logs[i]);
            }else{
            Serial2.write(' ');
            }
        }
        Ps3.setPlayer(7);
      }else{
        Ps3.setPlayer(2);
      }
      START_on=true;
    }
    if (Ps3.data.button.start ==false){
      START_on=false;
    }

  //---Activación control altura---
    if (Ps3.data.analog.button.cross>5 && X_on==false){
      control_altitud=!control_altitud;
      if (control_altitud){
        Ps3.setPlayer(6);
      }else{
        Ps3.setPlayer(2);
      }
      X_on=true;
    }
    if (Ps3.data.analog.button.cross==0){
      X_on=false;
    }


}



////////////////////////////////////////////////////////////////////////
/
//////////////////   CALCULOS CALIBRACIONES
//////////////////////
////////////////////////////////////////////////////////////////////////
/

//----------calculoErrorImuMedio----------
void calculoErrorImuMedio(){//Vale para calibrar si está perfectamente
recto
  //Esta función calculará el error medio al inicio, para ello se
leerán 500 valores del acelerometro y giroscopio
  //Y se irán sumando, para después dividir entre el total de medidas
```

```
  //Como el dispositivo no estará en movimiento existirá un error
medio entre medidas que después se utilizará
  //para calibrar las medidas reales tomadas

  //Error medio acelerómetro
  for(int i=0;i<iteraciones_calibracion;i++){
  //Se solicita al sensor los datos del acelerómetro
    I2CIMU.beginTransmission(0x68);
    I2CIMU.write(0x3B); //Se solicita el registro 0x3B que corresponde
con el valor de aceleración del eje X
    I2CIMU.endTransmission(false);
    I2CIMU.requestFrom(0x68,6,true);//Se solicitan 6 registros de 8
bit, correspondientes con los valores de aceleración de los tres ejes

    //A continuación se guarda la lectura de datos de aceleración del
IMU en m/(s^2)
    Acc_X=I2CIMU.read()<<8|I2CIMU.read(); //Son datos de 16 bit, así
que se desplazan 8 bit y se aplica la operación OR
    Acc_Y=I2CIMU.read()<<8|I2CIMU.read();
    Acc_Z=I2CIMU.read()<<8|I2CIMU.read();

    //Se va acumulando el error
    //(giro en X)
    Acceleration_angle_error[0] +=
atan((Acc_Y/acc_scale)/sqrt(pow((Acc_X/acc_scale),2) +
pow((Acc_Z/acc_scale),2)))*rad_to_deg;
    //(giro en Y)
    //Se utiliza la solución de matriz de giro de otra rotación debido
a que se emplean más datos~~
    Acceleration_angle_error[1] += atan2((-
Acc_X/acc_scale),(Acc_Z/acc_scale))*rad_to_deg;


    //Se aplica el error medio
    if (i==iteraciones_calibracion-1){
      Acceleration_angle_error[0] =
Acceleration_angle_error[0]/iteraciones_calibracion;
      Acceleration_angle_error[1] =
Acceleration_angle_error[1]/iteraciones_calibracion;
    }
  }

  for(int i=0;i<iteraciones_calibracion;i++){

      //Se obtienen los datos del giroscopio del IMU en radianes por
segundo
   I2CIMU.beginTransmission(0x68);
   I2CIMU.write(0x43); //Datos del giroscopio en el eje X
   I2CIMU.endTransmission(false);
   I2CIMU.requestFrom(0x68,6,true); //Solo se solicitan los datos de
los ejes X,Y y Z

   Gyr_X=I2CIMU.read()<<8|I2CIMU.read(); //Se vuelven a guardar los
valores desplazando y aplicando OR
   Gyr_Y=I2CIMU.read()<<8|I2CIMU.read();
   Gyr_Z=I2CIMU.read()<<8|I2CIMU.read();

   //Se aplican las escalas en función de la resolución de giro
deseada
   Gyro_X = Gyr_X/gyro_scale;
   Gyro_Y = Gyr_Y/gyro_scale;
```

```
   Gyro_Z = Gyr_Z/gyro_scale;

  //Se va acumulando el error
   Gyr_X_error+=Gyro_X;
   Gyr_Y_error+=Gyro_Y;
   Gyr_Z_error+=Gyro_Z;

  //Se aplica el error medio
   if (i==iteraciones_calibracion-1){
    Gyr_X_error=Gyr_X_error/iteraciones_calibracion;
    Gyr_Y_error=Gyr_Y_error/iteraciones_calibracion;
    Gyr_Z_error=Gyr_Z_error/iteraciones_calibracion;
   }

  }

}


//---------------calculoAltitudRelativaCalibrado()--------------
void calculoAltitudRelativaCalibrado(){


    for(int i=0;i<iteraciones_calibracion_altitud;i++){
      if (! bmp.performReading()) {
        Serial.println("Failed to perform reading :(");
      }else{

        if(i==10){//Para que no tarde en establecerse el valor
          altitud_filtrada=bmp.readAltitude(SEALEVELPRESSURE_HPA);
        }



      //Implemento un filtro complementario para evitar las
fluctuaciones
      altitud_filtrada=altitud_filtrada*(float)0.9 +
bmp.readAltitude(SEALEVELPRESSURE_HPA)*(float)0.1;



      //Depuración
        Serial.print("Altitud_filtrada:");
        Serial.print(altitud_filtrada);
        Serial.print(",");
        Serial.print("Altitud biblioteca:");
        Serial.print(bmp.readAltitude(SEALEVELPRESSURE_HPA));
        Serial.print(",");
        Serial.println("");


        if(i> iteraciones_estabilizacion_calibracion_altitud-1){//Solo
cuenta tras un periodo de adaptación
          alt_err_calibr+=altitud_filtrada;
        }


        //Se aplica el error medio
         if (i==iteraciones_calibracion_altitud-1){
```

```
alt_err_calibr=alt_err_calibr/(iteraciones_calibracion_altitud-
iteraciones_estabilizacion_calibracion_altitud);
          Serial.println("ERROR");
          Serial.println(alt_err_calibr);
          Serial.println("ERROR");
         }
      }


  }//for


}
```

A.1.2 Código WebServer REST API

```
/*
SERVIDOR ESP32
*/

// Import required libraries
#include "WiFi.h"
#include "ESPAsyncWebServer.h"


// Set your access point network credentials
const char* ssid = "VTOL";
const char* password = "vtollogs";


// Create AsyncWebServer object on port 80
AsyncWebServer server(80);


const byte numChars = 36;
char receivedChars[numChars];
boolean newData = false;

String logs="NO_LOG";

String readLogs() {
  return String(logs);
}


void setup(){
  // Serial port for debugging purposes
  Serial.begin(9600);
  Serial.println();

  // Setting the ESP as an access point
  Serial.print("Setting AP (Access Point)…");
  // Remove the password parameter, if you want the AP (Access Point)
to be open
  WiFi.softAP(ssid, password);

  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
```

```
  Serial.println(IP);

  server.on("/logs", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", readLogs().c_str());
  });




  // Start server
  server.begin();

  //Inicio la comunicación serie2 con el VTOL
  Serial2.begin(500000);

}

void loop(){
    recvWithStartEndMarkers();
    showNewData();
}



void recvWithStartEndMarkers() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;

    while (Serial2.available() > 0 && newData == false) {
        rc = Serial2.read();

        if (recvInProgress == true) {
            if (rc != endMarker) {
                receivedChars[ndx] = rc;
                ndx++;
                if (ndx >= numChars) {
                    ndx = numChars - 1;
                }
            }
            else {
                receivedChars[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                ndx = 0;
                newData = true;
            }
        }

        else if (rc == startMarker) {
            recvInProgress = true;
            memset(receivedChars,0, sizeof(receivedChars));
        }
    }
}

void showNewData() {
  int i=0;
```

```
    if (newData == true) {
        //Serial.print("This just in ... ");
        //Serial.println(receivedChars);

          logs=receivedChars;
          Serial.println(logs);


        newData = false;
    }
}
```

Obtenido principalmente de [108] con modificaciones.

A.1.3 Código WebSClient REST API

```
#include <WiFi.h>
#include <HTTPClient.h>

const char* ssid = "VTOL";
const char* password = "vtollogs";

//Your IP address or domain name with URL path
const char* serverNameLogs = "http://192.168.4.1/logs";


#include <Wire.h>

String logs;
String logs_anteriores;
int counter=0;

unsigned long previousMillis = 0;
const long interval = 100;



void setup() {
  Serial.begin(9600);


  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");

  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP Address: ");
  Serial.println(WiFi.localIP());

}

void loop() {
```

```
  unsigned long currentMillis = millis();

  if(currentMillis - previousMillis >= interval) {
     // Check WiFi connection status
    if(WiFi.status()== WL_CONNECTED ){
      logs = httpGETRequest(serverNameLogs);//Se queda haciendo
poolling cada intervalo

      if (logs!=logs_anteriores){ //Si los logs no han cambiado no
hagas nada
          Serial.println(logs);
      }



      // save the last HTTP GET Request
      previousMillis = currentMillis;
      logs_anteriores=logs;
      counter=0;
    }
    else {
      if (counter<3){//Solo se muestra 3 veces para no saturar los
logs
        Serial.println("WiFi Disconnected");
        counter++;
      }

    }
  }
}




String httpGETRequest(const char* serverName) {
  WiFiClient client;
  HTTPClient http;

  // Your Domain name with URL path or IP address with path
  http.begin(client, serverName);

  // Send HTTP POST request
  int httpResponseCode = http.GET();

  String payload = "--";

  if (httpResponseCode>0) {
    //Serial.print("HTTP Response code: ");
    //Serial.println(httpResponseCode);
    payload = http.getString();
  }
  else {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
  // Free resources
  http.end();

  return payload;
```

```
}
```

Obtenido principalmente de [108] con modificaciones.

A.2 Pruebas de concepto

A.2.1 Posicionamiento MPU6050

```
////////////////////////////////////////////////////////////////////////
/
//////////////////   CALCULOS    UAV
//////////////////////
////////////////////////////////////////////////////////////////////////
/
//----------calculosImuPosicion----------
void calculosImuPosicion(){
  //Obtengo las aceleraciones en m/s^2 teniendo en cuenta el error (de
cada eje del sensor)
  aceleracion_x=((Acc_X-Acc_X_error)/acc_scale)*gravity;
  aceleracion_y=((Acc_Y-Acc_Y_error)/acc_scale)*gravity;
  aceleracion_z=((Acc_Z-Acc_Z_error)/acc_scale)*gravity;

  //Si el sensor está girado tendré que aplicar matrices de rotación:
  //Paso angulos a radianes
  alfa = Total_angle[2]*(1/rad_to_deg);
  beta = Total_angle[0]*(1/rad_to_deg);
  ganma = Total_angle[1]*(1/rad_to_deg);

  //Dado que el vector de gravedad solo tiene componente en Z, se
simplifican las matrices de rotación
  //Se han cambiado rotación_x y rotación_y de sitio debido a la
posición del sensor y los vectores de este
  rotacion_y = ((cos(alfa)*cos(beta))*vector_gravedad[0]) +
(((cos(alfa)*sin(beta)*sin(ganma))-
(sin(alfa)*cos(ganma)))*vector_gravedad[1]) +
(((cos(alfa)*sin(beta)*cos(ganma))+(sin(alfa)*sin(ganma)))*vector_grav
edad[2]);
  rotacion_x = ((sin(alfa)*cos(beta))*vector_gravedad[0]) +
(((sin(alfa)*sin(beta)*sin(ganma))+(cos(alfa)*cos(ganma)))*vector_grav
edad[1]) + (((sin(alfa)*sin(beta)*cos(ganma))-
(cos(alfa)*sin(ganma)))*vector_gravedad[2]);
  rotacion_z = ((-sin(beta))*vector_gravedad[0]) +
((cos(beta)*sin(ganma))*vector_gravedad[1]) +
((cos(beta)*cos(ganma))*vector_gravedad[2]);


   //Ahora paso a restar las componentes en cada dirección para
obtener la aceleración con independencia de la gravedad
   aceleracion_total_x=aceleracion_x-rotacion_x;
   aceleracion_total_y=aceleracion_y-rotacion_y;

  //si se desactiva la posición habrá que resetear las variables a
cero

//------------X------------
velocidad_x += (aceleracion_total_x*elapsedTime);
```

```
posicion_x += 
(velocidad_x*elapsedTime)+(0,5*(aceleracion_x*pow(elapsedTime,2)));


//------------Y------------
velocidad_y += (aceleracion_total_y*elapsedTime);

posicion_y += 
(velocidad_x*elapsedTime)+(0,5*(aceleracion_y*pow(elapsedTime,2)));


}
```

Se aportan en las siguientes figuras, imágenes de las pruebas. En la primera de ellas puede visualizarse cómo se consigue paliar la acción de la gravedad en el acelerómetro, con este inclinado.

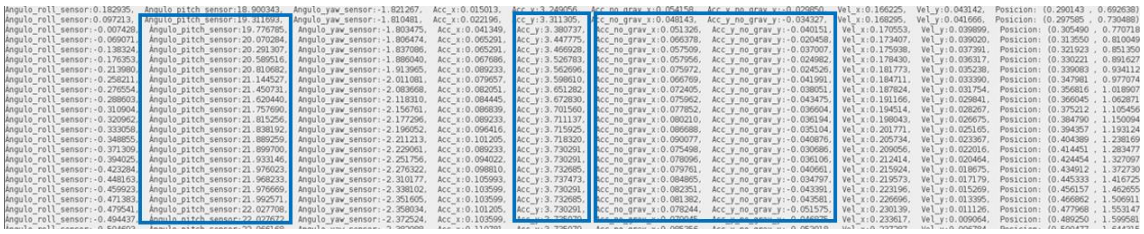


Figura 75. *Muestra de eliminación de las componentes de gravedad del eje Z.*

La segunda captura muestra la deriva de los datos con el sensor inmóvil, tras el paso del tiempo.

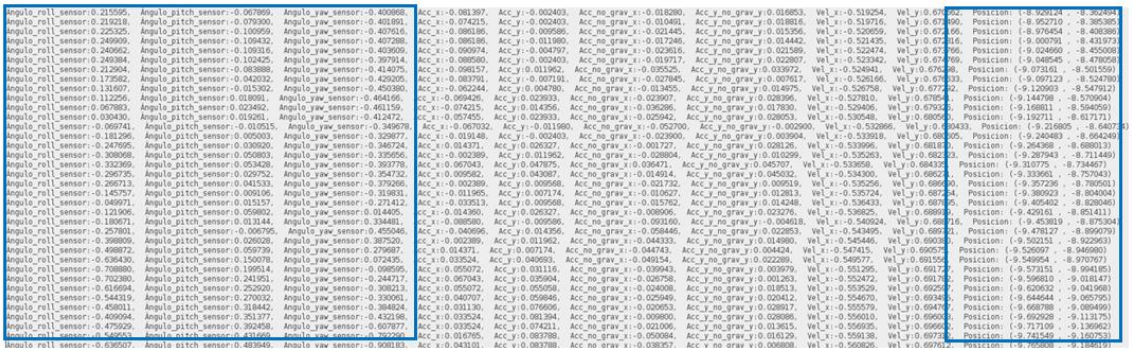


Figura 76. *Deriva en los cálculos de posición con el paso del tiempo.*

A.2.2 Posicionamiento ADNS3080

```
/*
 This sketch shows how to retrieve displacement data.
 */

#include <ADNS3080.h>

// SPI pins:
#define PIN_RESET     9
#define PIN_CS        10

ADNS3080 <PIN_RESET, PIN_CS> sensor;

// Initial position
int x = 0;
int y = 0;

void setup() {
  sensor.setup();
  Serial.begin(9600);
}

void loop() {
  int8_t dx, dy;      // Displacement since last function call

  sensor.displacement( &dx, &dy );

  // Integrate displacements
  x += dx;
  y += dy;

  // Displacement:
  Serial.print( " dx=" );
  Serial.print( dx );
  Serial.print( " dy=" );
  Serial.print( dy );
  Serial.print( " x=" );
  Serial.print( x );
  Serial.print( " y=" );
  Serial.print( y );
  Serial.println();
}
```

El código procede directamente de los ejemplos de la biblioteca [99], se realizaron pruebas con otras implementaciones obteniendo resultados similares. En lo referente al programa Python utilizado para la calibración y el reconocimiento de imágenes, se expone en la ***Figura 73*** una comparative de las imágenes de muestra del repositorio frente a los resultados obtenidos.

Las imágenes se han obtenido en durante el verano con altas condiciones de luminosidad.

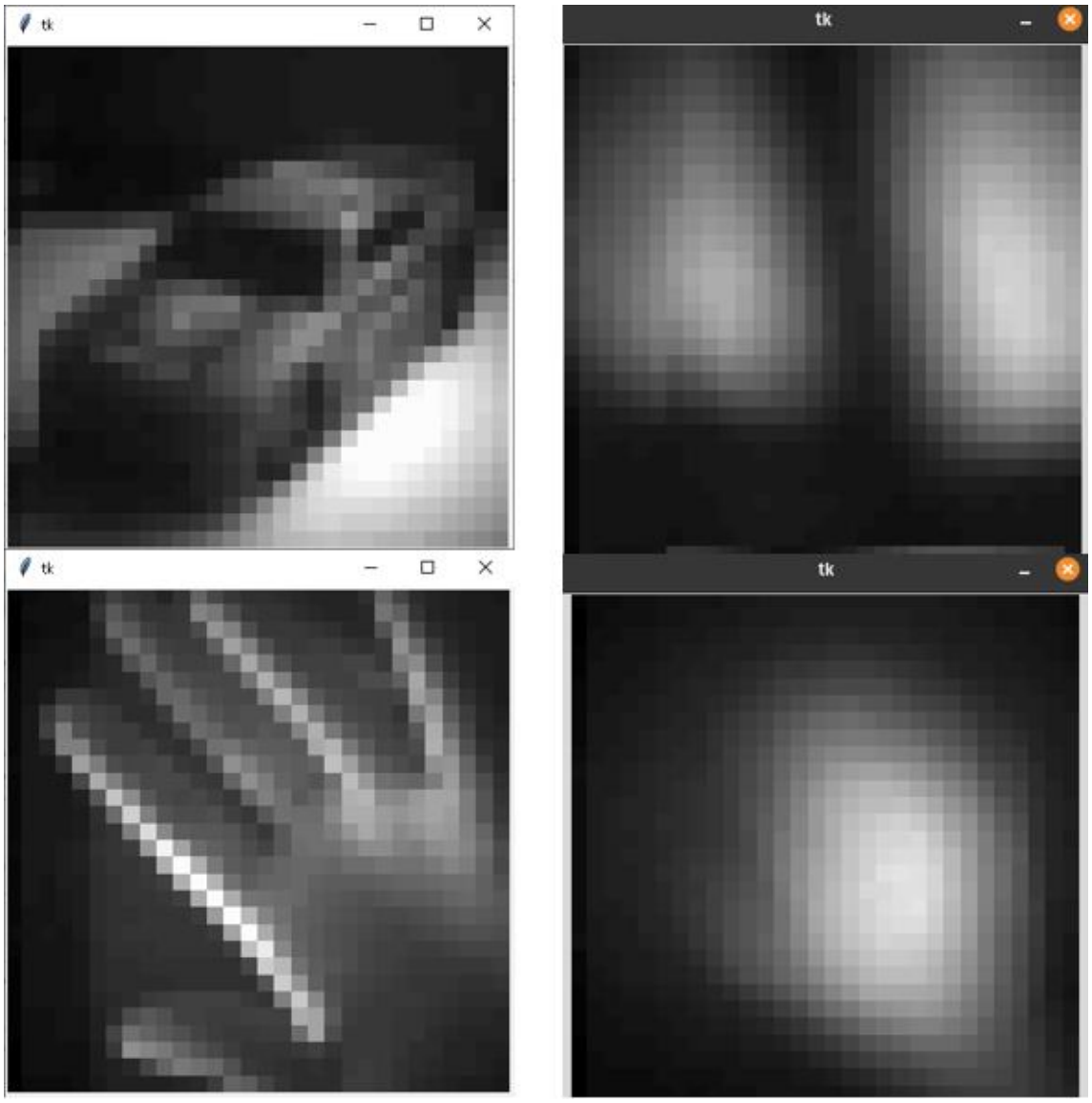


Figura 77. *Imágenes de muestra de la biblioteca ADNS_frame_capture* [100]*(izquierda) frente a imágenes obtenidas en la práctica (derecha).Coche de juguete (imagen superior izquierda), mano (imagen inferior izquierda), ventana (imagen superior derecha) y rostro (imagen inferior derecha).*

A.2.3 Posicionamiento GNSS

A.2.3.1 NeoGPS

```
#ifndef GPSport_h
#define GPSport_h

#define gpsPort Serial
#define GPS_PORT_NAME "Serial"
#define DEBUG_PORT Serial

#endif

#include <NeoGPS_cfg.h>
#include <ublox/ubxGPS.h>
```

```
//====================================================================
==
//  Program: ublox.ino
//
//  Prerequisites:
//     1) You have a ublox GPS device
//     2) PUBX.ino works with your device
//     3) You have installed the ubxGPS.* and ubxmsg.* files.
//     4) At least one UBX message has been enabled in ubxGPS.h.
//     5) Implicit Merging is disabled in NMEAGPS_cfg.h.
//
//  Description:  This program parses UBX binary protocal messages
from
//     ublox devices.  It shows how to acquire the information
necessary
//     to use the GPS Time-Of-Week in many UBX messages.  As an offset
//     from midnight Sunday morning (GPS time), you also need the
current
//     UTC time (this is *not* GPS time) and the current number of GPS
//     leap seconds.
//
//  Serial is for debug output to the Serial Monitor window.
//
//  License:
//    Copyright (C) 2014-2017, SlashDevin
//
//    This file is part of NeoGPS
//
//    NeoGPS is free software: you can redistribute it and/or modify
//    it under the terms of the GNU General Public License as
published by
//    the Free Software Foundation, either version 3 of the License,
or
//    (at your option) any later version.
//
//    NeoGPS is distributed in the hope that it will be useful,
//    but WITHOUT ANY WARRANTY; without even the implied warranty of
//    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//    GNU General Public License for more details.
//
//    You should have received a copy of the GNU General Public
License
//    along with NeoGPS.  If not, see <http://www.gnu.org/licenses/>.
//
//====================================================================
==

#include <GPSport.h>

#include <Streamers.h>

//--------------------------------------------------------------
// Check that the config files are set up properly

#ifndef NMEAGPS_DERIVED_TYPES
  #error You must "#define NMEAGPS_DERIVED_TYPES" in NMEAGPS_cfg.h!
#endif

#if !defined(UBLOX_PARSE_STATUS)  & !defined(UBLOX_PARSE_TIMEGPS) & \
    !defined(UBLOX_PARSE_TIMEUTC) & !defined(UBLOX_PARSE_POSLLH)  & \
    !defined(UBLOX_PARSE_DOP)     & !defined(UBLOX_PARSE_PVT)     & \
```

```
    !defined(UBLOX_PARSE_VELNED)  & !defined(UBLOX_PARSE_SVINFO)  & \
    !defined(UBLOX_PARSE_HNR_PVT)

  #error No UBX binary messages enabled: no fix data available.

#endif

#ifndef NMEAGPS_RECOGNIZE_ALL
  //  Resetting the messages with ublox::configNMEA requires that
  //    all message types are recognized (i.e., the enum has all
  //    values).
  #error You must "#define NMEAGPS_RECOGNIZE_ALL" in NMEAGPS_cfg.h!
#endif

//--------------------------------------------------------------------
//  Derive a class to add the state machine for starting up:
//    1) The status must change to something other than NONE.
//    2) The GPS leap seconds must be received
//    3) The UTC time must be received
//    4) All configured messages are "requested"
//         (i.e., "enabled" in the ublox device)
//  Then, all configured messages are parsed and explicitly merged.

class MyGPS : public ubloxGPS
{
public:

    enum
      {
        GETTING_STATUS,
        GETTING_LEAP_SECONDS,
        GETTING_UTC,
        RUNNING
      }
        state NEOGPS_BF(8);

    MyGPS( Stream *device ) : ubloxGPS( device )
    {
      state = GETTING_STATUS;
    }

    //--------------------------

    void get_status()
    {
      static bool acquiring = false;

      if (fix().status == gps_fix::STATUS_NONE) {
        static uint32_t dotPrint;
        bool            requestNavStatus = false;

        if (!acquiring) {
          acquiring = true;
          dotPrint = millis();
          DEBUG_PORT.print( F("Acquiring...") );
          requestNavStatus = true;

        } else if (millis() - dotPrint > 1000UL) {
          dotPrint = millis();
          DEBUG_PORT << '.';
```

```
          static uint8_t requestPeriod;
          if ((++requestPeriod & 0x07) == 0)
            requestNavStatus = true;
        }

        if (requestNavStatus)
          // Turn on the UBX status message
          enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_STATUS );

      } else {
        if (acquiring)
          DEBUG_PORT << '\n';
        DEBUG_PORT << F("Acquired status: ") << (uint8_t) fix().status
<< '\n';

        #if defined(GPS_FIX_TIME) & defined(GPS_FIX_DATE) & \
            defined(UBLOX_PARSE_TIMEGPS)

          if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEGPS ))
            DEBUG_PORT.println( F("enable TIMEGPS failed!") );

          state = GETTING_LEAP_SECONDS;
        #else
          start_running();
          state = RUNNING;
        #endif
      }
    } // get_status

    //--------------------------

    void get_leap_seconds()
    {
      #if defined(GPS_FIX_TIME) & defined(GPS_FIX_DATE) & \
          defined(UBLOX_PARSE_TIMEGPS)

        if (GPSTime::leap_seconds != 0) {
          DEBUG_PORT << F("Acquired leap seconds: ") <<
GPSTime::leap_seconds << '\n';

          if (!disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEGPS ))
            DEBUG_PORT.println( F("disable TIMEGPS failed!") );

          #if defined(UBLOX_PARSE_TIMEUTC)
            if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEUTC ))
              DEBUG_PORT.println( F("enable TIMEUTC failed!") );
            state = GETTING_UTC;
          #else
            start_running();
          #endif
        }
      #endif

    } // get_leap_seconds

    //--------------------------

    void get_utc()
    {
      #if defined(GPS_FIX_TIME) & defined(GPS_FIX_DATE) & \
          defined(UBLOX_PARSE_TIMEUTC)
```

```
    lock();
      bool            safe = is_safe();
      NeoGPS::clock_t sow  = GPSTime::start_of_week();
      NeoGPS::time_t  utc  = fix().dateTime;
    unlock();

    if (safe && (sow != 0)) {
      DEBUG_PORT << F("Acquired UTC: ") << utc << '\n';
      DEBUG_PORT << F("Acquired Start-of-Week: ") << sow << '\n';

      start_running();
    }
  #endif

} // get_utc

//--------------------------

void start_running()
{
  bool enabled_msg_with_time = false;

  #if defined(UBLOX_PARSE_POSLLH)
    if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_POSLLH ))
      DEBUG_PORT.println( F("enable POSLLH failed!") );

    enabled_msg_with_time = true;
  #endif

  #if defined(UBLOX_PARSE_PVT)
    if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_PVT ))
      DEBUG_PORT.println( F("enable PVT failed!") );

    enabled_msg_with_time = true;
  #endif

  #if defined(UBLOX_PARSE_VELNED)
    if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_VELNED ))
      DEBUG_PORT.println( F("enable VELNED failed!") );

    enabled_msg_with_time = true;
  #endif

  #if defined(UBLOX_PARSE_DOP)
    if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_DOP ))
      DEBUG_PORT.println( F("enable DOP failed!") );
    else
      DEBUG_PORT.println( F("enabled DOP.") );

    enabled_msg_with_time = true;
  #endif

  #if defined(UBLOX_PARSE_SVINFO)
    if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_SVINFO ))
      DEBUG_PORT.println( F("enable SVINFO failed!") );

    enabled_msg_with_time = true;
  #endif

  #if defined(UBLOX_PARSE_TIMEUTC)
```

```
        #if defined(GPS_FIX_TIME) & defined(GPS_FIX_DATE)
          if (enabled_msg_with_time &&
              !disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEUTC ))
            DEBUG_PORT.println( F("disable TIMEUTC failed!") );

        #elif defined(GPS_FIX_TIME) | defined(GPS_FIX_DATE)
          // If both aren't defined, we can't convert TOW to UTC,
          // so ask for the separate UTC message.
          if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEUTC ))
            DEBUG_PORT.println( F("enable TIMEUTC failed!") );
        #endif

      #endif

      state = RUNNING;
      trace_header( DEBUG_PORT );

    } // start_running

    //--------------------------

    bool running()
    {
      switch (state) {
        case GETTING_STATUS      : get_status      (); break;
        case GETTING_LEAP_SECONDS: get_leap_seconds(); break;
        case GETTING_UTC         : get_utc         (); break;
      }

      return (state == RUNNING);

    } // running

} NEOGPS_PACKED;

// Construct the GPS object and hook it to the appropriate serial
device
static MyGPS gps( &gpsPort );

#ifdef NMEAGPS_INTERRUPT_PROCESSING
  static void GPSisr( uint8_t c )
  {
    gps.handle( c );
  }
#endif

//--------------------------

static void configNMEA( uint8_t rate )
{
  for (uint8_t i=NMEAGPS::NMEA_FIRST_MSG; i<=NMEAGPS::NMEA_LAST_MSG;
i++) {
    ublox::configNMEA( gps, (NMEAGPS::nmea_msg_t) i, rate );
  }
}

//--------------------------

static void disableUBX()
{
```

```
  gps.disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEGPS );
  gps.disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEUTC );
  gps.disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_VELNED );
  gps.disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_POSLLH );
  gps.disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_DOP );
}

//--------------------------

void setup()
{
  // Start the normal trace output
  DEBUG_PORT.begin(9600);
  while (!DEBUG_PORT)
    ;

  DEBUG_PORT.print( F("ublox binary protocol example started.\n") );
  DEBUG_PORT << F("fix object size = ") << sizeof(gps.fix()) << '\n';
  DEBUG_PORT << F("ubloxGPS object size = ") << sizeof(ubloxGPS) <<
'\n';
  DEBUG_PORT << F("MyGPS object size = ") << sizeof(gps) << '\n';
  DEBUG_PORT.println( F("Looking for GPS device on " GPS_PORT_NAME) );
  DEBUG_PORT.flush();

  // Start the UART for the GPS device
  #ifdef NMEAGPS_INTERRUPT_PROCESSING
    gpsPort.attachInterrupt( GPSisr );
  #endif
  gpsPort.begin(9600);

  // Turn off the preconfigured NMEA standard messages
  configNMEA( 0 );

  // Turn off things that may be left on by a previous build
  disableUBX();

  #if 0
    // Test a Neo M8 message -- should be rejected by Neo-6 and Neo7
    ublox::cfg_nmea_v1_t test;

    test.always_output_pos  = false; // invalid or failed
    test.output_invalid_pos = false;
    test.output_invalid_time= false;
    test.output_invalid_date= false;
    test.use_GPS_only       = false;
    test.output_heading     = false; // even if frozen
    test.__not_used__       = false;

    test.nmea_version = ublox::cfg_nmea_v1_t::NMEA_V_4_0;
    test.num_sats_per_talker_id =
ublox::cfg_nmea_v1_t::SV_PER_TALKERID_UNLIMITED;

    test.compatibility_mode = false;
    test.considering_mode   = true;
    test.max_line_length_82 = false;
    test.__not_used_1__     = 0;

    test.filter_gps    = false;
    test.filter_sbas   = false;
    test.__not_used_2__= 0;
    test.filter_qzss   = false;
```

```
    test.filter_glonass= false;
    test.filter_beidou = false;
    test.__not_used_3__= 0;

    test.proprietary_sat_numbering = false;
    test.main_talker_id = ublox::cfg_nmea_v1_t::MAIN_TALKER_ID_GP;
    test.gsv_uses_main_talker_id = true;
    test.beidou_talker_id[0] = 'G';
    test.beidou_talker_id[1] = 'P';

    DEBUG_PORT << F("CFG_NMEA result = ") << gps.send( test );
  #endif

  while (!gps.running())
    if (gps.available( gpsPort ))
      gps.read();
}

//--------------------------

void loop()
{
  if (gps.available( gpsPort ))
    trace_all( DEBUG_PORT, gps, gps.read() );

  // If the user types something, reset the message configuration
  //   back to a normal set of NMEA messages.  This makes it
  //   convenient to switch to another example program that
  //   expects a typical set of messages.  This also saves
  //   putting those config messages in every other example.

  if (DEBUG_PORT.available()) {
    do { DEBUG_PORT.read(); } while (DEBUG_PORT.available());
    DEBUG_PORT.println( F("Stopping...") );

    configNMEA( 1 );
    disableUBX();
    gpsPort.flush();
    gpsPort.end();

    DEBUG_PORT.println( F("STOPPED.") );
    for (;;);
  }
}
```

Procedente de NeoGPS[97].

A.2.3.2 TinyGPS

```
#include <SoftwareSerial.h>

#include <TinyGPS.h>

/* This sample code demonstrates the normal use of a TinyGPS object.
   It requires the use of SoftwareSerial, and assumes that you have a
   4800-baud serial GPS device hooked up on pins 4(rx) and 3(tx).
*/

TinyGPS gps;
```

```
SoftwareSerial ss(4, 3);

void setup()
{
  Serial.begin(115200);
  ss.begin(4800);

  Serial.print("Simple TinyGPS library v. ");
Serial.println(TinyGPS::library_version());
  Serial.println("by Mikal Hart");
  Serial.println();
}

void loop()
{
  bool newData = false;
  unsigned long chars;
  unsigned short sentences, failed;

  // For one second we parse GPS data and report some key values
  for (unsigned long start = millis(); millis() - start < 1000;)
  {
    while (ss.available())
    {
      char c = ss.read();
      // Serial.write(c); // uncomment this line if you want to see
the GPS data flowing
      if (gps.encode(c)) // Did a new valid sentence come in?
        newData = true;
    }
  }

  if (newData)
  {
    float flat, flon;
    unsigned long age;
    gps.f_get_position(&flat, &flon, &age);
    Serial.print("LAT=");
    Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat,
6);
    Serial.print(" LON=");
    Serial.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon,
6);
    Serial.print(" SAT=");
    Serial.print(gps.satellites() == TinyGPS::GPS_INVALID_SATELLITES ?
0 : gps.satellites());
    Serial.print(" PREC=");
    Serial.print(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 :
gps.hdop());
  }

  gps.stats(&chars, &sentences, &failed);
  Serial.print(" CHARS=");
  Serial.print(chars);
  Serial.print(" SENTENCES=");
  Serial.print(sentences);
  Serial.print(" CSUM ERR=");
  Serial.println(failed);
  if (chars == 0)
    Serial.println("** No characters received from GPS: check wiring
**");
```

}

Procedente de TinyGPS [98].

A.3 Enlaces de estimación de costes

Microcontrolador

https://es.aliexpress.com/item/1005005562548949.html?algo_pvid=9fe1feb1-49bb-41bb-92df-b0397911ddab&algo_exp_id=9fe1feb1-49bb-41bb-92df-b0397911ddab-0&pdp_npi=4%40dis%21EUR%214.88%210.99%21%21%215.20%211.05%21%402100391a017175320643383973e771c%2112000034113114728%21sea%21ES%210%21AB&curPageLogUid=2drgazU3i2Go&utparam-url=scene%3Asearch%7Cquery_from%3A

Motores BLDC

https://www.amazon.es/CENPEK-escobillas-Outrunner-Multi-Copter-Quadcopter/dp/B098STSYD8/ref=sr_1_2?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=327XAN9WY8B5R&dib=eyJ2IjoiMSJ9.tL8-6sgB_qLtBtYBYJbZlEaDhJyhZzmtQRDmYDjCn7xG2XMJkPHww6sfKCSu8Onl_FumsVroZIHZAXeE94ytEPZaSSlToGLO3gIOHjBEZChJYfzt8zH3mChewN1moRNsuPSc3T2Nf9bWJ1c96QssogIG5ixicGKnAvFia4Q2WxKoPibH5gFo9VwXiRthTaTt1QjtDDEvxt2xqAh7sWo4l-b0Jsy6sdl4QyHC3muoB9aOAaaOTNrkm4HbxSCxyVbkcHlfN0uaYYj5bxokI0IlIIDYMtIE51c0OkTvNQD0oFM.9SA7c3v--ZFopJRkTryzoTE6Bop5MEbISI_T0oPibhg&dib_tag=se&keywords=bldc+a2212+1000kv&qid=1717532114&sprefix=bldc+a2212+1000kv%2Caps%2C115&sr=8-2

ESC

https://grupoelectrostore.com/shop/motores/controladores/esc-brushless/esc-30a-simonk-para-motor-brushless/

Servomotores

https://www.amazon.es/AZDelivery-%E2%AD%90%E2%AD%90%E2%AD%90%E2%AD%90%E2%AD%90-hubchrauber-MG996R-Digital/dp/B07H87592P?source=ps-sl-shoppingads-lpcontext&ref_=fplfs&smid=A1X7QLRQH87QA3&th=1

IMU

https://www.amazon.es/ARCELI-aceler%C3%B3metro-Aceler%C3%B3metro-girosc%C3%B3pico-Transductor/dp/B07BVXN2GP/ref=sr_1_5?dib=eyJ2IjoiMSJ9.6WZyh-bXpazpT59jV2bla3vRpPJ4tertHJMPD1GEMegj8FC21Q76qYkbaT7JVpSIyvRJw4sAu8az-cSaonX4Q6oU0X2B9I_jdJet2jZaiNyvYSfgY4d66HmUQ928gv6PM4EHXmUfrpIO2uDVrGpvQ6sUeEvab2DB1J0Xk8DK6PbzENeOXhV3qnYukUUA5bGFFm2V8BIJXPicGAbiCW6p0jun8d_NlgAjAqyspXLCbMnNzZQi1fb5-7D4_1nu_oMRKezSuToEbzXlgzGwaaB2MenhYjMNSWZ59Gp25ga9TTA.xTRQJvkajKsCj5kcFf4ltJ42797m9T0pWX7Jbxf0NGQ&dib_tag=se&keywords=mpu6050&qid=1717532258&sr=8-5

Sensor de ultrasonidos

https://www.amazon.es/ARCELI-transductor-medici%C3%B3n-Distancia-ultras%C3%B3nico/dp/B07MPZR59P/ref=sr_1_5?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=2RK8PMEYVVBFT&dib=eyJ2IjoiMSJ9.Gi1Y5lYk0Ir72ac5dQlQDfnHVjjV3BLScVg9mxVLRC2UnVNclE9n79mf0FN8ZRnwI79yzM3RnGU0LGpkrK2FUQKMwdH8mv8QY06h9Nt7MooKXQAXUENCepbjFPBqbILkuFxVL0HHXAQcpG_Oq7stVK2Zwr1Q2cUtn4yAe24b-kDhsOSJhW08bJ3liJ6WtBa_y_4d4m7AsDLhNNmGWlV9Ki8HAoct52gHFdphsViwlxbggIJPdi5mR7FmVmYGQD8qheM19R5URiE6sFADMrL0IVfiyt7kQtrZRyKr42Cqjlk.TnHEQKZucw83qv2ab85Mzt68wP5Z5xL7FTsj-2lB-tQ&dib_tag=se&keywords=hc-sr04&qid=1717532277&sprefix=hc-sr04%2Caps%2C134&sr=8-5

Sensor de presión

https://es.aliexpress.com/item/1005006838209282.html?spm=a2g0o.productlist.main.3.7f6436c60JpoBX&algo_pvid=6d7f4da2-740a-49c9-87c6-407171c04e2d&algo_exp_id=6d7f4da2-740a-49c9-87c6-407171c04e2d-1&pdp_npi=4%40dis%21EUR%2118.58%215.39%21%21%21143.58%2141.64%21%40211b664d17175323988907239ea291%211120000038467788181%21sea%21ES%210%21AB&curPageLogUid=DQlclw6eSE4H&utparam-url=scene%3Asearch%7Cquery_from%3A

Fotosensor

https://es.aliexpress.com/item/1005005928511999.html?spm=a2g0o.productlist.main.3.377e10631Ot7H8&algo_pvid=17799a7b-15f9-416f-9216-cea8e87eab89&algo_exp_id=17799a7b-15f9-416f-9216-cea8e87eab89-1&pdp_npi=4%40dis%21EUR%2113.78%2113.78%21%21%21106.48%21106.48%21%402103919917175324204908449ea0d4%211200003488913745 3%21sea%21ES%210%21AB&curPageLogUid=r5nIw6EOTo6I&utparam-url=scene%3Asearch%7Cquery_from%3A

GNSS

https://es.aliexpress.com/item/32748573256.html?src=google

Baterías

https://www.amazon.es/RoaringTop-2200mAh-Bater%C3%ADa-Helic%C3%B3ptero-Paquetes/dp/B08H82KFS6

Cargador de baterías

https://es.aliexpress.com/item/1005005028773427.html?spm=a2g0o.productlist.main.7.6d1c10e9VRlr7o&algo_pvid=95037b67-de37-43f1-a68f-a63497a32c50&aem_p4p_detail=202406041322308328451669961900000996648&alg

o_exp_id=95037b67-de37-43f1-a68f-a63497a32c50-3&pdp_npi=4%40dis%21EUR%2186.38%2117.42%21%21%21667.52%21134.57%21%402103919917175325505264152ea0ec%21120000031387031898%21sea%21ES%210%21AB&curPageLogUid=nKUv1LjSIoa2&utparam-url=scene%3Asearch%7Cquery_from%3A&search_p4p_id=20240604132230832845166996190000099664 8_1

Varillas

https://www.leroymerlin.es/productos/ferreteria-y-seguridad/perfiles-pletinas-chapas-y-rejillas/perfiles/perfil-aluminio/perfil-forma-cuadrada-de-aluminio-en-bruto-standers-alt-16-x-an-16mm-x-l-1-m-87825791.html

PLA de impresión

https://www.amazon.es/dp/B083YX1TZG?psc=1&ref=ppx_yo2ov_dt_b_product_details

Protoboard

https://www.amazon.es/ELEGOO-Breadboard-Prototipo-Electr%C3%B3nicos-Compatibles/dp/B072Z7Y19F/ref=sr_1_13?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=2SHL87UTTGEVW&dib=eyJ2IjoiMSJ9.8uriLRwUu-DSdazoeIh3ri3WUKcyagC0PsqrA11KpIqkjqNyxHvcZqRP0SsJeBgzuUIQCI6oLMPgDPrbF6ccUFITXdbWTJvSFuEAmSG_YVXCNKyTz02UUG_EMhhxTemlHirmGZnktjrGvFpv1ewedPE1Gc10XKzz0FaWOz01wQnOl65hM6KUgo4Dar4IWUT2exYJbfQwid4c4qUDQY7BUDd1N1AyXz4Qjlx1ptzFDB011sV1INSCe9N2uOTZC6odo-lGfVcFcjkAN9wtKHCSQXhW3z8QVtU8An92cdbEpNs.cGk-E3-RmJawlbi7OqkzncLo5C3xK6bzgZzHKkMeBms&dib_tag=se&keywords=protoboard&qid=1717531770&sprefix=protoboard%2Caps%2C122&sr=8-13

Cables de prototipado

https://www.amazon.es/Macho-Hembra-Macho-Macho-Hembra-Hembra-Prototipo-Protoboard/dp/B01NGTXASZ/ref=sr_1_5?dib=eyJ2IjoiMSJ9.lRvFWkXHlJfQN2AEa5s0aUIb6uSP0wcinD_8edzK5ZtEjDa5V-dkkfBfpsRdCIsSqcqQGqzj1b30FNY8uHNaOB2INJG5agsiDlthJWmlLrfmHExjDFfKaB8ACkKYkppqRxi6kikBvExRt33TFXyYMD6PEffzPSOD6C5Wvc703U250JZMO2U8WUXJ8bFOk0jhzfCVIrqeAhjKn2pda1KTOQthlWy8mPOfkqleudH8c9WqHHnZm5QtyRgZjse40g12qY1D0Xb2-4F_yU4H49mJ3KaWOtU4xkBsI8lBMcAFEbU.ZPK9UmnvttLUBjb2iu8OaFz8BWgNh7gVvMLaejHo_Ks&dib_tag=se&keywords=cable+prototipado&qid=1717531831&sr=8-5

Tornillos

https://www.amazon.es/520-tornillos-tuercas-hexagonales-roscados/dp/B09LRS8WLD/ref=sr_1_5?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=W14A7QILN3J2&dib=eyJ2IjoiMSJ9.G_VDHTKwH47Sg2hYsIdGP_0oYKQ5vVmNJbDHhU88bEMlJGAQURlQ4Kdf5oNebW3sFRzYwJjZVndgJI-F9RVg9e5AtKrouFY_Ha9r81BHxWMonaaYK8b5ltCBG-q65_wdPyjYKbgZjKUzU5XlBfkhlqTNYn0MxWp9cvrcpImHEkodhO-51Hdet1opKIKtf6aB01QEzYn1yqdxSWS9UG-mt9602ni9YEIEtsIvpEfJV-uJRAUoLcOaI2YuXWWKtKlbJluvm9AwDBMGh6rounwJeOd368s0UquJdWLd6LBncJk.34PnSZ_thdiw8GvGRkM7PA-5B10HhPmDtQU31FxBdGc&dib_tag=se&keywords=tornillos%2Bm5&qid=1717531844&sprefix=tornillos%2Bm%2Caps%2C122&sr=8-5&th=1

Mando de control

https://www.eneba.com/es/2-mandos-blancos-ps3-e757c04d

SIGLAS

ADC (*Analog to Digital Converter*)

AEMET (Agencia Estatal de Meteorología)

AP (*AccessPoint*)

BLDC (*BrushLess Direct Current*)

CAD (*Computer Aided Design* (CAD)

CEP (*Circular Error Probability*)

DMP (*Digital Motion Processor*)

DSP (*Digital Signal Processor*)

EMF (*Electromotive Force*)

ESC (*Electronic Speed Controllers*)

ESR (*Equivalent Series Resistance*)

FSR (*Full Scale Range*)

GLONASS (*Global'naya Navigatsionnaya Sputnikovaya Sistema*)

GNSS (*Global Navigation Satellital System*)

GPS (*Global Positioning System*)

HTTP (*Hyper Text Transfer Protocol*)

I+D (Investigación y Desarrollo)

I2C (*Inter-Integrated Circuit*)

IAS (*Image Adquisition System*)

IDE (*Integrated Development Environment*)

IDF (*IoT Development Framework*)

IRR (*Infrared Radiometer*)

LED (*Light Emitting Diode*)

Li-Po (Polímero de Litio)

LPF (*Low Pass Filter*)

MCU (*Micro Controller Unit*)

MEMS (*microelectromechanical systems*)

NIMH (Niquel-Metalhidruro)

PCB (*Printed Circuit Board*)

PID (*Proportional Integral Derivative controller* (PID)

PLA (*Poliácido Láctico*)

PWM (*Pulse Width Modulation*)

RAE (Real Academia Española)

RAE (Real Academia Española)

REST (*Representational State Transfer*)

RMS (*Root Mean Square*)

RMS (*Root Mean Square*)

RPM (*Revolutions Per Minute*)

SBAS (*Satellite Based Augmentation System*)

SONAR (*Sound Navigation And Rangign*)

SPI (*Serial Peripheral Interface*)

UAV (*Unmanned Aerial Vehicle* (UAV)

USB (*Universal Serial Bus*)

VTOL (*Vertical Take-Off and Landing*)

WiFi (*Wireless Fidelity*)

Referencias

[1] T. Hartsfield, “15th century futurism: da Vinci’s helicopter finally takes flight - Big Think.” Accessed: Dec. 27, 2023. [Online]. Available: https://bigthink.com/the-past/da-vinci-helicopter/

[2] National Geographic - Historia, “El globo aerostático y la conquista de los cielos,” National Geographic. Accessed: Aug. 24, 2022. [Online]. Available: https://historia.nationalgeographic.com.es/a/globo-aerostatico-y-conquista-cielos_7848

[3] David Daly, “A Not-So-Short History of Unmanned Aerial Vehicles (UAV),” Consortiq. Accessed: Aug. 24, 2022. [Online]. Available: https://consortiq.com/uas-resources/short-history-unmanned-aerial-vehicles-uavs

[4] Tim Sharp, “The First Powered Airship - The Greatest Moments in Flight - Space.” Accessed: Dec. 27, 2023. [Online]. Available: https://www.space.com/16623-first-powered-airship.html

[5] National Air and Space Museum, “1903 Wright Flyer.” Accessed: Dec. 27, 2023. [Online]. Available: https://airandspace.si.edu/collection-objects/1903-wright-flyer/nasm_A19610048000

[6] K. Vyas, “A Brief History of Drones: The Remote Controlled Unmanned Aerial Vehicles (UAVs),” Interesting Engineering. Accessed: Aug. 24, 2022. [Online]. Available: https://interestingengineering.com/innovation/a-brief-history-of-drones-the-remote-controlled-unmanned-aerial-vehicles-uavs

[7] RAE, “dron | Definición | Diccionario de la lengua española | RAE - ASALE.” Accessed: Aug. 24, 2022. [Online]. Available: https://dle.rae.es/dron?m=form

[8] Merriam-Webster, “Drone Definition & Meaning - Merriam-Webster.” Accessed: Aug. 24, 2022. [Online]. Available: https://www.merriam-webster.com/dictionary/drone

[9] Igor I.Sikorsky and Nichols, “Direct Lift Aircraft - United States Patent Office,” 1994488, 1931 Accessed: Dec. 27, 2023. [Online]. Available: https://docs.google.com/viewer?url=patentimages.storage.googleapis.com/pdfs/US1994488.pdf

[10] Connecticut History Org., “World’s First Helicopter .” Accessed: Dec. 27, 2023. [Online]. Available: https://connecticuthistory.org/worlds-first-helicopter-today-in-history/

[11] U.S. Government Printing Office, “UNMANNED AERIAL VEHICLES AND THE NATIONAL AIRSPACE SYSTEM.” Accessed: Aug. 24, 2022. [Online]. Available: https://www.govinfo.gov/content/pkg/CHRG-109hhrg28275/html/CHRG-109hhrg28275.htm

[12] JetPack Aviation, “The History Of Personal VTOL Technology .” Accessed: Aug. 24, 2022. [Online]. Available: https://jetpackaviation.com/history-of-personal-vtol-technology/

[13] USA TODAY, “Amazon testing delivery by drone, CEO Bezos says,” USA TODAY. Accessed: Aug. 24, 2022. [Online]. Available: https://eu.usatoday.com/story/tech/2013/12/01/amazon-bezos-drone-delivery/3799021/

[14] El español, “UPS empieza a repartir sus paquetes con drones,” El español. Accessed: Aug. 24, 2022. [Online]. Available: https://www.elespanol.com/invertia/empresas/20191001/ups-empieza-repartir-paquetes-drones/433457185_0.html

[15] UPS, “UPS realiza la primera entrega de vacunas contra la COVID-19 con un dron en EE. UU.,” UPS-About Us. Accessed: Aug. 24, 2022. [Online]. Available: https://about.ups.com/mx/es/our-stories/innovation-driven/drone-covid-vaccine-deliveries.html

[16] Judit Castaño, “Amazon empezará a entregar pedidos con drones a finales de año,” La Vanguardia. Accessed: Aug. 24, 2022. [Online]. Available: https://www.lavanguardia.com/tecnologia/20220617/8344743/amazon-entregara-pedidos-drones-finales-ano-pmv.html

[17] I. Barba, L. Palacio, and E.T.S.I.T. Universidad de Valladolid, “Física - Tema 9: Ecuaciones de Maxwell y Ondas electromagnéticas,” 2017.

[18] P. A. Tipler and G. Mosca, *Física para la ciencia y la tecnología - Volumen 2*, 6th ed. Editorial Reverté, 2010.

[19] The Engineering Mindset, “BLDC motor .” Accessed: Jan. 15, 2024. [Online]. Available: https://theengineeringmindset.com/arduino-code-bldc-motor/

[20] R. Gambhir and A. Kumar Jha, “Brushless DC Motor: Construction and Applications,” *Int J Eng Sci (Ghaziabad)*, 2013, Accessed: Jan. 15, 2024. [Online]. Available: www.theijes.com

[21] M. Yildirim, H. Kurum, D. Miljavec, and S. Corovic, *Eng. Technol. Appl. Sci. Res. - Influence of Material and Geometrical Properties of Permanent Magnets on Cogging Torque of BLDC*, 2nd ed., vol. 8, no. 2. 2018. Accessed: Jan. 15, 2024. [Online]. Available: www.etasr.com

[22] Y. L. Karnavas, I. D. Chasiotis, and D. N. Stravoulellis, “A practical BLDC motor design procedure for diver propulsion vehicle applications,” *Proceedings - 2018 23rd International Conference on Electrical Machines, ICEM 2018*, pp. 513–519, Oct. 2018, doi: 10.1109/ICELMACH.2018.8507046.

[23] CLR (Compañía Levantina de Reductores), “Single-phase, two-phase and three-phase motors.” Accessed: Aug. 29, 2022. [Online]. Available: https://clr.es/blog/en/single-phase-two-phase-three-phase-motors/

[24] Homebuilt Electric Motors, “Common Winding Schemes.” Accessed: Jan. 17, 2024. [Online]. Available: https://www.bavaria-direct.co.za/scheme/common/

[25] Grupo Electrostore, “MOTOR BRUSHLESS A2212 1000KV 13T .” Accessed: Jan. 16, 2024. [Online]. Available: https://grupoelectrostore.com/shop/motores/motores-brushless/motor-brushless-a2212-1000kv-13t/

[26] Hyperdrive Hobby, “Crazepony DX2205 2300KV .” Accessed: Jan. 16, 2024. [Online]. Available: https://hyperdrivehobby.com/products/crazepony-dx2205-2300kv?variant=19464934916169

[27] Emrik Joner, “Brushless Motor Power and Efficiency Calculations,” Tyto Robotics. Accessed: Jan. 23, 2024. [Online]. Available: https://www.tytorobotics.com/blogs/articles/brushless-motor-power-and-efficiency-analysis

[28] Kevin Beck, “Motor Efficiency,” Sciencing. Accessed: Jan. 23, 2024. [Online]. Available: https://sciencing.com/convert-horsepower-kwh-7613597.html

[29] megaAVR and Microchip Technology Inc., “ATmega8A - Data Sheet,” 2020.

[30] TOSHIBA, “Silicon N-Channel MOS TPCA8057-h.” Accessed: Feb. 21, 2024. [Online]. Available: https://pdf1.alldatasheet.es/datasheet-pdf/view/540551/TOSHIBA/TPCA8057-H.html

[31] J. Zhao and Y. Yu, “Brushless DC Motor Fundamentals Brushless DC Motor Fundamentals Application Note AN047,” *MPS - MonolithicPower*, 2011, Accessed: Feb. 21, 2024. [Online]. Available: www.MonolithicPower.com

[32] Ø. Magnussen, G. Hovland, M. Ottestad, and S. Kirby, “Experimental study on the influence of controller firmware on multirotor actuator dynamics,” *ROSE 2014 - 2014 IEEE International Symposium on RObotic and SEnsors Environments, Proceedings*, pp. 106–111, Nov. 2014, doi: 10.1109/ROSE.2014.6952992.

[33] Simon Kirby, “GitHub - Open Source Firmware for ATmega-based Brushless ESCs.” Accessed: Jan. 30, 2024. [Online]. Available: https://github.com/sim-/tgy

[34] Yin Yan Model Tech MFT, “EMAX User Instruction for SimonK Series ESC.”

[35] RAE - ASALE, “servomotor | Definición | Diccionario de la lengua española .” Accessed: Feb. 27, 2024. [Online]. Available: https://dle.rae.es/servomotor?m=form

[36] Wally Gastreich, “What is a Servo Motor and How it Works?,” RealPars. Accessed: Feb. 27, 2024. [Online]. Available: https://www.realpars.com/blog/servo-motor

[37] Paul Evans, “Servo Motor Explained - The Engineering Mindset.” Accessed: Feb. 27, 2024. [Online]. Available: https://theengineeringmindset.com/servo-motor-xplained/

[38] M. A. Cebrián and L. E. Giraudo, “TEMA 7: DEL AMPLIFICADOR DIFERENCIAL AL AMPLIFICADOR OPERACIONAL (A.O.),” *Circuitos Electrónicos Analógicos - UVA*, 2017.

[39] M. A. Cebrián and L. E. Giraudo, “TEMA 8: APLICACIONES LINEALES DEL A.O.,” *Circuitos Electrónicos Analógicos - UVA*, 2017.

[40] D. M. Alter, “Application Report Using PWM Output as a Digital-to-Analog Converter on a TMS320F280x Digital Signal Controller,” *Texas Instruments*,

2008, Accessed: Feb. 28, 2024. [Online]. Available: http://www.ti.com/lit/zip/SPRAA88.

[41] K. Burgess, “PWM to DC Voltage Conversion,” Sep. 2015.

[42] “PWM Controller | PSpice.” Accessed: Feb. 28, 2024. [Online]. Available: https://www.pspice.com/power-management/state-average-controllers/pwm-controller

[43] “MG996R Servo Datasheet.” Accessed: Feb. 26, 2024. [Online]. Available: https://html.alldatasheet.es/html-pdf/1131873/ETC2/MG996R/110/1/MG996R.html

[44] “SG90 Micro Servo Datasheet.” Accessed: Feb. 26, 2024. [Online]. Available: https://html.alldatasheet.es/html-pdf/1572383/ETC/SG90/59/1/SG90.html

[45] InvenSense Inc., “MPU-6000 and MPU-6050 Product Specification Revision 3.4,” Aug. 2013, Accessed: Mar. 20, 2024. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[46] InverSense Inc., “MPU-9250 Product Specification Revision 1.1,” Jun. 2016.

[47] O. González González and J. M. H. Mangas, “Diseño y fabricación de un controlador de videojuegos con interfaz USB y forma de arma,” *Universidad de Valladolid*, 2022, Accessed: Mar. 20, 2024. [Online]. Available: https://uvadoc.uva.es/handle/10324/50030

[48] L. Pelaz and P. López, “TEMA 2: SENSORES,” *SISTEMAS ELECTRÓNICOS DE INSTRUMENTACIÓN - UVA*, 2022.

[49] ANALOG DEVICES, “Accelerometer and Gyroscopes Sensors: Operation, Sensing, and Applications.” Accessed: Mar. 20, 2024. [Online]. Available: https://www.analog.com/en/resources/technical-articles/accelerometer-and-gyroscopes-sensors-operation-sensing-and-applications.html

[50] Admat Inc and Eric von Spreckelsen, “Why Tantalum Capacitors Are Essential in Modern Electronics .” Accessed: Apr. 01, 2024. [Online]. Available: https://www.admatinc.com/why-tantalum-capacitors-are-essential-in-modern-electronics/

[51] Elijah J. Morgan, “HCSR04 Ultrasonic Sensor.” Accessed: Apr. 01, 2024. [Online]. Available: https://pdf1.alldatasheet.es/datasheet-pdf/view/1132203/ETC2/HC-SR04.html

[52] J. Vicente Antón, “Tema 2 - Ejemplos y aplicaciones de sensores,” *Instrumentación y equipos electrónicos - UVa*, pp. 1–19, 2021.

[53] Bosch Sensortec, “Pressure Sensor BMP390 .” Accessed: Apr. 07, 2024. [Online]. Available: https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/pressure-sensors-bmp390.html

[54] Adafruit, “Adafruit BMP390 - Precision Barometric Pressure and Altimeter .” Accessed: Apr. 07, 2024. [Online]. Available: https://www.adafruit.com/product/4816

[55] Bosch Sensortec, “Barometric pressure sensor: working principle.” Accessed: Apr. 11, 2024. [Online]. Available: https://www.youtube.com/watch?v=0bw5UJQxkhA

[56] Limor Fried *et al.*, “GitHub - adafruit/Adafruit_BMP3XX.” Accessed: Apr. 15, 2024. [Online]. Available: https://github.com/adafruit/Adafruit_BMP3XX/tree/master

[57] Bosch Sensortec, “BMP180 Digital pressure sensor,” 2013.

[58] Agencia Estatal de Meteorología - AEMET and Gobierno de España, “Castilla y León - Datos horarios - Mapa - Presión (hPa) .” Accessed: Apr. 15, 2024. [Online]. Available: https://www.aemet.es/es/eltiempo/observacion/ultimosdatos?k=cle&w=0&datos=img&x=h24&f=presion

[59] Avago Technologies, “ADNS-3080 High-Performance Optical Mouse Sensor,” 2008.

[60] ArduPilot Dev Team, “Mouse-based Optical Flow Sensor (ADNS3080) — Copter documentation.” Accessed: Apr. 20, 2024. [Online]. Available: https://ardupilot.org/copter/docs/common-mouse-based-optical-flow-sensor-adns3080.html

[61] Branch Education Channel, “How does a Mouse know when you move it?,” Youtube. Accessed: Apr. 21, 2024. [Online]. Available: https://www.youtube.com/watch?v=SAaESb4wTCM

[62] Ossila, “Solar Spectrum: Solar Radiation and Irradiance.” Accessed: Apr. 21, 2024. [Online]. Available: https://www.ossila.com/pages/the-solar-spectrum

[63] NASA Earth Observatory, “Incoming Sunlight.” Accessed: Apr. 21, 2024. [Online]. Available: https://earthobservatory.nasa.gov/features/EnergyBalance/page2.php

[64] U.S. Government, “GPS: The Global Positioning System.” Accessed: Apr. 23, 2024. [Online]. Available: https://www.gps.gov/

[65] “Sobre el sistema GLONASS.” Accessed: Apr. 23, 2024. [Online]. Available: https://glonass-iac.ru/spa/about_glonass/

[66] “GALILEO | European Global Navigation Satellite System.” Accessed: Apr. 23, 2024. [Online]. Available: https://galileognss.eu/

[67] “BeiDou Navigation Satellite System.” Accessed: Apr. 23, 2024. [Online]. Available: http://en.beidou.gov.cn/SYSTEMS/System/

[68] u-blox, “NEO-M8 concurrent GNSS modules Data sheet ”, Accessed: Apr. 21, 2024. [Online]. Available: https://www.u-blox.com/en/product/neo-m8-series

[69] R. de la R. Steinz and M. G. Gadañón, “Diseño y aplicaciones de sistemas de radiocomunicaciones y radiodeterminación - Tema 4: Introducción a los sistemas de radionavegación y posicionamiento,” *Universidad de Valladolid*, pp. 1–21, 2022.

[70] Harnam Arneja, Andrew Bender, Sam Jugus, and Tim Reid, “Solving the GPS Equations.” Accessed: Apr. 25, 2024. [Online]. Available: http://mason.gmu.edu/~treid5/Math447/GPSEquations/

[71] Frank van Diggelen and Ashtech Inc., “GPS accuracy: Lies, damn lies and statistics - GPS World.” Accessed: Apr. 25, 2024. [Online]. Available: https://www.gpsworld.com/gps-accuracy-lies-damn-lies-and-statistics/

[72] EU Agency for the Space Programme, “EGNOS.” Accessed: Apr. 25, 2024. [Online]. Available: https://www.euspa.europa.eu/eu-space-programme/egnos

[73] Ministerio de Transportes y Movilidad Sostenible and AESA-Agencia Estatal de Seguridad Aérea, "¿Tienes un UAS/dron? | Requisitos mínimos para volar cualquier dron a partir del 31 de diciembre de 2020." Accessed: Apr. 28, 2024. [Online]. Available: https://www.seguridadaerea.gob.es/es/ambitos/drones/tienes-un-uas-dron

[74] I. y U. Ministerio de Ciencia, "Requisitos mínimos para volar cualquier dron desde 2021." Accessed: Apr. 28, 2024. [Online]. Available: https://www.ciencia.gob.es/dam/jcr:849a914e-e1ef-4b5f-9ab4-45dc47e94be9/Requisitos%20minimos%20para%20volar%20cualquier%20dron%20desde%202021.pdf

[75] EASA, "Easy Access Rules for Unmanned Aircraft Systems (Regulations (EU) 2019/947 and 2019/945) - Revision from April 2024." Accessed: Apr. 28, 2024. [Online]. Available: https://www.easa.europa.eu/en/document-library/easy-access-rules/easy-access-rules-unmanned-aircraft-systems-regulations-eu

[76] Espressif Systems, "ESP32WROOM32 Datasheet," 2023, Accessed: Apr. 28, 2024. [Online]. Available: https://www.espressif.com/en/support/download/documents.

[77] Arduino, "Arduino." Accessed: Apr. 28, 2024. [Online]. Available: https://www.arduino.cc/

[78] Makerguides and Hiren Tailor, "Comparación de la velocidad de ESP32 vs Arduino ." Accessed: Apr. 28, 2024. [Online]. Available: https://www.makerguides.com/es/esp32-vs-arduino-speed-comparison/

[79] USAWire and Muhammad Zaman, "Understanding 8 Basic Points about 3S Lipo Batteries." Accessed: Apr. 28, 2024. [Online]. Available: https://usawire.com/understanding-8-basic-points-about-3s-lipo-batteries/

[80] "J.S.T. Connector's Web Site." Accessed: Apr. 28, 2024. [Online]. Available: https://www.jst-mfg.com/index.php

[81] Raytrax Crilax (User), "Manual iMAX B6." Accessed: Apr. 28, 2024. [Online]. Available: https://issuu.com/raytrax/docs/manual_imax_b6_-_espa_ol/1

[82] Espressif, “ESP-IDF Programming Guide v5.2.1 documentation.” Accessed: May 12, 2024. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/index.html

[83] Espressif, “Arduino ESP32 latest documentation.” Accessed: May 12, 2024. [Online]. Available: https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html

[84] Kevin Harrington, “GitHub - madhephaestus/ESP32Servo: Arduino-compatible servo library for the ESP32.” Accessed: May 12, 2024. [Online]. Available: https://github.com/madhephaestus/ESP32Servo

[85] Arduino, “Wire - Arduino Reference.” Accessed: May 12, 2024. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/communication/wire/

[86] Jeffrey Van Pernis, “GitHub - jvpernis/esp32-ps3: Control your ESP32 projects with a PS3 controller!” Accessed: May 12, 2024. [Online]. Available: https://github.com/jvpernis/esp32-ps3

[87] Adafruit Industries, “GitHub - adafruit/Adafruit_Sensor: Common sensor library.” Accessed: May 12, 2024. [Online]. Available: https://github.com/adafruit/Adafruit_Sensor

[88] Adafruit Industries, “GitHub - adafruit/Adafruit_BusIO: Arduino library for I2C & SPI abstractions.” Accessed: May 12, 2024. [Online]. Available: https://github.com/adafruit/Adafruit_BusIO

[89] ElectronicCats Open Hardware, “GitHub - ElectronicCats/mpu6050: MPU6050 Arduino Library.” Accessed: May 12, 2024. [Online]. Available: https://github.com/ElectronicCats/mpu6050

[90] K. S. Nicholas Zambetti, “Inter-Integrated Circuit (I2C) Protocol | Arduino Documentation.” Accessed: May 12, 2024. [Online]. Available: https://docs.arduino.cc/learn/communication/wire/

[91] InvenSense Inc., “MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2 ,” vol. 4, Autumn 2013.

[92] Ian Beavers, “The Case of the Misguided Gyro | Analog Devices.” Accessed: May 13, 2024. [Online]. Available: https://www.analog.com/en/resources/analog-dialogue/raqs/raq-issue-139.html

[93] Kenneth Rohde Christiansen and Alexander Shalamov, “Motion Sensors Explainer,” W3C Working Group Note. Accessed: May 13, 2024. [Online]. Available: https://www.w3.org/TR/motion-sensors/#complementary-filters

[94] Joop Brokking, “Create a 6dof IMU with a gyro and accelerometer for (Arduino) multicopters.” Accessed: May 13, 2024. [Online]. Available: http://www.brokking.net/imu.html

[95] G. G. Slabaugh, “Computing Euler angles from a rotation matrix”, Accessed: May 13, 2024. [Online]. Available: https://eecs.qmul.ac.uk/~gslabaugh/publications/euler.pdf

[96] Joop Brokking, “The STM32 Arduino autonomous quadcopter.” Accessed: May 19, 2024. [Online]. Available: http://www.brokking.net/ymfc-32_auto_main.html

[97] Andrew Young, Christian Loitsch, Paul Stoffregen, and Sébastien, “GitHub - SlashDevin/NeoGPS: NMEA and ublox GPS parser for Arduino, configurable to use as few as 10 bytes of RAM.” Accessed: May 20, 2024. [Online]. Available: https://github.com/SlashDevin/NeoGPS

[98] Juhnyuk Lee, “GitHub - neosarchizo/TinyGPS: A compact Arduino NMEA (GPS) parsing library.” Accessed: May 20, 2024. [Online]. Available: https://github.com/neosarchizo/TinyGPS

[99] RCmags, “GitHub - RCmags/ADNS3080: Arduino library for the ADNS3080 mouse sensor.” Accessed: May 20, 2024. [Online]. Available: https://github.com/RCmags/ADNS3080

[100] RCmags, “GitHub - RCmags/ADNS3080_frame_capture: Python script to render images from ADNS3080 sensor.” Accessed: May 20, 2024. [Online]. Available: https://github.com/RCmags/ADNS3080_frame_capture

[101] CH Robotics, “Using Accelerometers to Estimate Position and Velocity.” Accessed: May 20, 2024. [Online]. Available: https://web.archive.org/web/20201111202735/http:/www.chrobotics.com/library/accel-position-velocity

[102] J. B. Prieto and E. T. S. I. T. U. de Valladolid, “Fundamentos de Transmisión por Radio Tema 1 - Fundamentos de radiación,” 2018.

[103] “Tinkercad - Cree diseños digitales 3D con CAD en línea.” Accessed: May 23, 2024. [Online]. Available: https://www.tinkercad.com/

[104] “UltiMaker Cura - UltiMaker.” Accessed: May 23, 2024. [Online]. Available: https://ultimaker.com/es/software/ultimaker-cura/

[105] “BIQU-Design, desarrollando producción de impresoras 3D y accesorios – Biqu Equipment.” Accessed: May 23, 2024. [Online]. Available: https://biqu.equipment/es

[106] “MG996R Servo Model by PagliaIndustries - Thingiverse.” Accessed: May 23, 2024. [Online]. Available: https://www.thingiverse.com/thing:4894842

[107] “GanttProject .” Accessed: Jun. 11, 2024. [Online]. Available: https://www.ganttproject.biz/about

[108] “ESP32 Client-Server Wi-Fi Communication Between Two Boards | Random Nerd Tutorials.” Accessed: Jun. 12, 2024. [Online]. Available: https://randomnerdtutorials.com/esp32-client-server-wi-fi/