PRON: Diseño y construcción de un dron basado en Raspberry Pi

Jorge López García

Plan de Estudios del Estudiante

Electronica

Aleix López Anton

Carlos Monzo Sánchez

03/01/2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*PRON: Diseño y construcción de un dron basado en Raspberry Pi*
Nombre del autor:	*Jorge López García*
Nombre del consultor/a:	*Aleix Lopez Anton*
Nombre del PRA:	*Carlos Monzo Sánchez*
Fecha de entrega (mm/aaaa):	01/2020
Titulación:	*Plan de estudios del estudiante*
Área del Trabajo Final:	*Electrónica*
Idioma del trabajo:	*Español*
Palabras clave	*Dron, Raspberry, Naze32*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El objeto de este proyecto, tal y como se indica en el título, es el diseño y construcción de un dron basado en Raspberry Pi, se desea desarrollar todo el proceso constructivo de un UAV (Vehículo aéreo no tripulado) de bajo coste, para ello se dividirá el proyecto en tres partes.

En la primera parte se realizará un estudio de los modelos y prototipos existentes en el mercado, con el fin de realizar un diseño estructural que permita un vuelo más estable, además se realizarán análisis de peso, resistencia y coste de todos los componentes necesarios, los cuales podrán ser comprados o construidos de manera autónoma, buscando siempre obtener materiales ligeros, resistentes y a una buena relación calidad precio.

En la segunda de este proyecto es la correspondiente a la configuración interna del dron, habiendo elegido la Raspberry Pi como ordenador de a bordo, se seleccionará una tarjeta controladora de vuelo compatible con esta, para posteriormente realizar todas las calibraciones necesarias para lograr un vuelo estable, además, se buscará la manera de añadir funcionalidades extra al dron como la grabación de imágenes o la recopilación de datos atmosféricos.

La última parte de este proyecto se centrará en la forma de control del dron, se sopesarán las opciones control por RC o por WI-FI mediante aplicación móvil, en caso de que la seleccionada fuese esta última, se estudiaría la opción de diseñar una aplicación de cero, modificar una app "open source" o usar una ya programada.

Abstract (in English, 250 words or less):

The purpose of this project, as indicated in the title, is the design and construction of a drone based on Raspberry Pi, it is desired to develop the entire construction process of a UAV

(Unmanned Aerial Vehicle) of low cost, for this the project is divided into three parts.

In the first part, a study of the models and prototypes existing in the market will be carried out in order to achive an structural design that allows a more stable flight. Also the analysis of weight, resistance and cost of all the necessary components, which can be purchased or built autonomously, will be made always seeking to obtain lightweight, resistant materials and a good value for money.

The second part of this project, corresponds to the internal configuration, a Raspberry Pi has been chosen as on-board computer, a compatible flight controller seens necesary to achive the proyect and set all the necessary calibrations to achieve a stable flight. In addition, add additional functionalities to the drone such as image recording or atmospheric data collection will be an important part too.

The last part of this project will focus on the form of drone control, control options will be RC or by WI-FI through the mobile Application.

Tabla de contenidos

Tabla de ilustraciones

1. Introducción

1.1. Resumen

En la actualidad, la mayoría de los drones realizados para uso recreativo se apoyan en un controlador tipo Arduino, esto se debe sobre todo a su gran capacidad de prototipado y a su facilidad para conectarse a casi cualquier sistema, tanto analógico como digital, además, cuenta con la capacidad de procesado suficiente como para controlar los motores. dron El problema aparece a la hora de introducir funciones añadidas como la recogida de datos en las que esta capacidad se queda pequeña. Es aquí cuando aparecen los sistemas en chips (SoC) como la Raspberry Pi, que al poseer mayor capacidad de cálculo permiten, por ejemplo, el control del dron en tiempo real desde el móvil y el envío de datos (temperatura, presión, altura, etc.…) e imágenes al mismo, aunque es justo añadir que se antoja necesaria una tarjeta controladora para los motores ya que la Raspberry no es capaz de realizar esta función.

El objeto de este proyecto, tal y como se indica en el título, es el diseño y construcción de un dron basado en Raspberry Pi, se desea desarrollar todo el proceso constructivo de un UAV (Vehículo aéreo no tripulado) de bajo coste, para ello se dividirá el proyecto en tres partes.

En la primera parte se realizará un estudio de los modelos y prototipos existentes en el mercado con el fin de realizar un diseño estructural que permita un vuelo más estable, además se realizarán análisis de peso, resistencia y coste de todos los componentes necesarios, los cuales podrán ser comprados o construidos de manera autónoma, buscando siempre obtener materiales ligeros, resistentes y a una buena relación calidad precio.

En la segunda de este proyecto es la correspondiente a la configuración interna del dron, habiendo elegido la Raspberry Pi como ordenador de a bordo, se seleccionará una tarjeta controladora de vuelo compatible con esta, para posteriormente realizar todas las calibraciones necesarias para lograr un vuelo estable, además, se buscará la manera de añadir funcionalidades extra al dron como la grabación de imágenes o la recopilación de datos atmosféricos.

La última parte de este proyecto se centrará en la forma de control del dron, se sopesarán las opciones control por RC o por WI-FI mediante aplicación móvil, en caso de que la seleccionada fuese esta última, se estudiaría la opción de diseñar una aplicación de cero, modificar una aplicación “open source” o usar una ya programada.

Teniendo todo esto en cuenta, el proyecto luce como apasionante, presentando al alumno restos en diferentes materias como la electrónica o el diseño, además de permitir ahondar en estos campos de conocimiento de una manera amena y entretenida.

1.2. Objetivos principales del proyecto

El objetivo fundamental de este proyecto será la creación de un dron totalmente operativo utilizando una Raspberry Pi como núcleo, para ello será necesario el desarrollo de todas las partes asociadas al mismo:

- *Diseño o selección del hardware*: Se buscarán o diseñarán las piezas óptimas para la construcción de la estructura del dron, esto incluye los motores, chasis, hélices, sensores, batería, tren de aterrizaje.

- *Selección de una tarjeta controladora para los motores*: La Raspberry Pi no es capaz de controlar los motores en tiempo real por lo que se antoja necesario la selección de una tarjeta controladora.

- *Diseño del software*: Se diseñará el software necesario para el correcto funcionamiento del dron, esto incluye la creación de los algoritmos de control y el diseño de la comunicación entre el teléfono o control remoto y el UVA.

- *Construcción del Dron*: Por último, se integrarán todas las piezas (hardware, software, controladora y Raspberry) para crear un dron totalmente operativo.

Siguiendo la filosofía de la comunidad de creadores de Raspberry Pi, se intentará reutilizar y/o adaptar todo el código libre posible, además se utilizarán proyectos de código abierto como guía, esto quiere decir que el proyecto se abre a la posibilidad usar, modificar, y compartir cualquier proyecto para casi cualquier propósito siempre que tenga una licencia open Source.

1.3. Motivación y contexto.

Este proyecto se realiza bajo el contexto del trabajo fin de master, de ahora en adelante TFM, del área de electrónica de la UOC, con el fin de aunar los conocimientos obtenidos a lo largo de la trayectoria del alumno en la UOC, diseñando y construyendo un Dron desde cero se aplican conocimientos de electrónica y microelectrónica, sistemas de radionavegación, gestión avanzada de proyectos y, dependiendo del tipo de dispositivo seleccionado para controlar el dron, diseño y aplicaciones de antenas.

En la realización de este TFM se pretende de realizar un estudio de los diferentes drones existentes en el mercado, cada uno con diferentes características y configuraciones, además de profundizar en cuál es el mejor método de control de drones "*low cost*".

El autor cree que el mundo de los drones está en proceso de expansión, siendo una de las vías de negocio más fructíferas de aquí a unos años, si bien es cierto que el dron que se va a desarrollar en este proyecto no puede ser lanzado al mercado por su tosquedad, puede servir para sentar las bases de conocimiento para futuros drones con unas condiciones más apropiadas para los posibles mercados, drones más pequeños para inspección o con más potencia de empuje para mensajería.

Hay que destacar además que la principal motivación que ha llevado a desarrollar este TFM, es la creencia de que se tratará de un reto complicado, pero a la vez llamativo y divertido, que podrá meter de lleno al alumno en el desarrollo y la historia de los drones.

Por último, otro punto de interés de este proyecto es la manufactura en si del dron, la construcción y el soldado de todas las piezas de este, así como la circuitería necesaria.

1.4. Metodología

Para la realización de este TFM se ha utilizado una metodología analítica que ha constado de cuatro fases que se muestran en el esquema siguiente:

ILUSTRACIÓN 1: METODOLOGÍA SEGUIDA

El autor cree que usar este tipo de metodología basada en la descomposición del producto final en todas y cada una de sus partes es la mejor manera para entender el funcionamiento de un dron, a continuación, se explicaran las cuatro fases:

- *Investigación*: Esta ha sido la fase más larga del proyecto, el autor se ha centrado en investigar sobre cada una de las piezas que forman el dron y cómo funcionan, los distintos tipos de comunicación entre el usuario y el dron, la disposición de motores idónea y la forma del dron.

- *Proconstrucción*: Una vez que ya se habían seleccionado todas las piezas se pasó a la preconstrucción, en este punto se ensamblaron la mayoría de las piezas como el chasis, los ESC y los motores con el fin de tener algo parecido al producto final para la realización de pruebas. En este apartado también se configuro la Raspberry Pi y la Acro Naze 32.

- *Pruebas*: Con todo el producto en su estado casi final, se realizaron pruebas de funcionamiento de motores, de conectividad entre App – Raspberry – Controladora y de funcionamiento de las baterías.

- *Ensamblaje final*: Una vez que todo estuvo testado se pasó a la fase de ensamblaje final en la que el dron tomo su forma final.

1.5. Planificación.

La estimación de tiempos es que el proyecto se realizara en 4 meses, se ha seguido la misma planificación que las Pec's con el fin de hacer más organizado el trabajo, quedando un diagrama de Gantt abajo indicado.

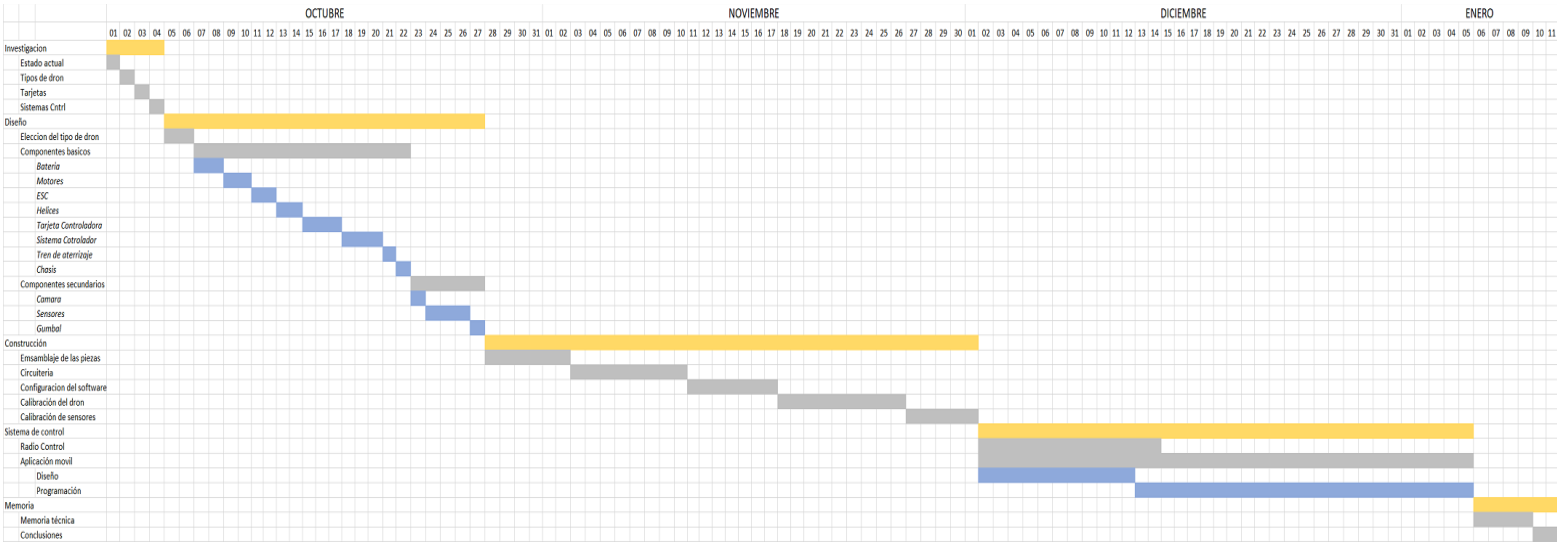

ILUSTRACIÓN 2: DIAGRAMA DE GANTT

1.6. Sumario productos obtenidos

Este apartado no aplica mucho ya que en este TFM el único producto que se ha obtenido ha sido el propio PRON, un dron construido sobre una Raspberry Pi que era la base fundamental del proyecto.

1.7. Descripción del resto de capítulos

Este TFM se encuentra dividido en siete capítulos, 6 más a parte de este en el que se encuentra el lector, a continuación, se explicara brevemente cada uno de ellos:

- *Estado del arte:* Breve explicación sobre la situación actual de los drones en cuanto a desarrollo y legislación.
- *Diseño del dron:* Primera parte de la construcción del dron en la que se muestra la investigación realizada y la selección de los componentes.
- *Implementación del dron:* Segunda parte de la construcción del dron, en la que se muestra el ensamblado y la configuración tanto de la Raspberry como de la controladora.
- *Conclusiones*: Apartado en el que se muestran los pensamientos finales sobre el TFM
- *Glosario*: Explicación de palabras técnicas aparecidas en el TFM.

- *Bibliografía:* Conjunto de obras consultadas para la realización del proyecto.

- *Anexos*: Código, manuales y presupuesto del proyecto.

2. Estado del Arte

2.1. Un poco de historia

El termino dron puede sonar reciente, pero su historia se remonta 100 años atrás, hasta la primera guerra mundial. Como la gran mayoría de los avances tecnológicos su primer uso, sobre el año 1914, fue militar. Aunque principio se usaron de dianas de entrenamiento para pilotos, rápidamente comenzaron a desarrollarse proyectos como el *Ruston Proctol Aerial Target*, el primer avión sin piloto controlado por la tecnología RC desarrollada por tesla, o el *Kettering Bug* que se trataba de un torpedo aéreo guiado por un giroscopio capaz de alcanzar blancos terrestres situados a 120Km. Más tarde en 1944 los Nazis crearían la "*buzz bomb*" capaz de alcanzar los 602Km/h.

En la década de los 60 las empresas empezaron a entrar en el mundo de los *UAV*, siendo la NASA la primera organización no militar en desarrollar y utilizar drones. Además, se produjo el primer boom de popularidad de los aviones RC, esto fue debido a un avance en la tecnología de transistores que permitió la reducción de tamaño y coste de estos, surgiendo con ello la primera comunidad *Maker*.

En los 2000 la Nasa continúo mejorando el desempeño de los drones, creando diseños que eran capaces de volar a más de 30 km de altura. En el 2006 la *FAA* emitió la primera licencia para un Dron no militar, reconociendo el potencial de estos, lo que permitió que se eliminaran limitaciones impuestas a los drones recreativos y que permitió que profesionales y empresas vieran diferentes usos para los *UAV*.

Y con esto estamos en la entrada del año 2010, donde el abaratamiento de los equipos electrónicos, sensores, baterías y demás piezas hacen que empresas como *Parrot* o *DJI*, lancen al mercado drones listos para pilotar, generando así el segundo boom de popularidad de los *UAV* y reviviendo el mercado artesanal que ya se había iniciado en la década de los 60.

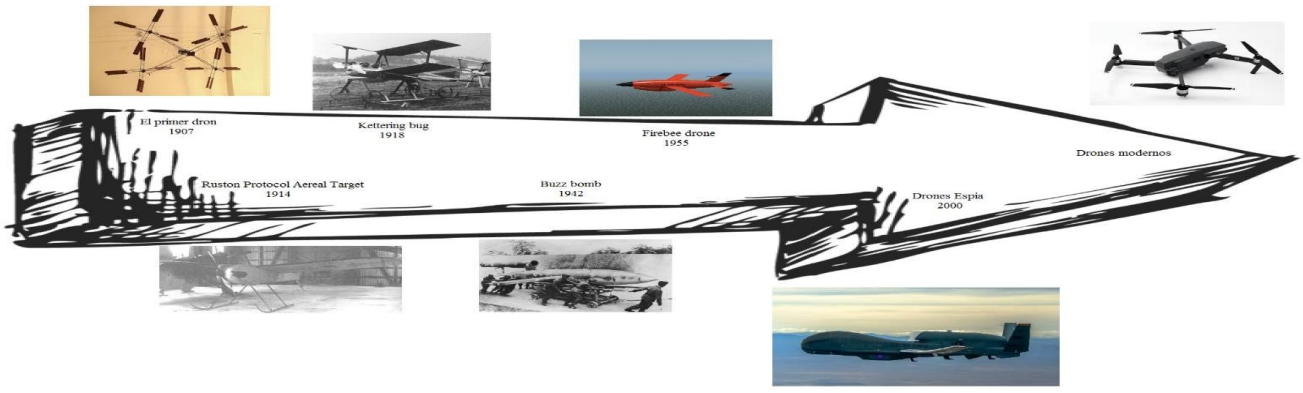

ILUSTRACIÓN 3: EVOLUCIÓN DEL DRON

2.2. Estado actual del Mercado

En el año 2016 el mercado de los drones de uso civil creció un 30% con respecto a 2016, siendo el mayor aumento de ventas en los últimos años, *Global Markets Inside* prevé que el mercado de los drones comerciales alcance un valor de 17.000 millones de dólares en 2024, llegando hasta casi los 70.000 millones en 2027, generando además cerca de 127.000 millones de dólares de manera indirecta. Esto da una idea clara de que el mercado del dron está en alza.

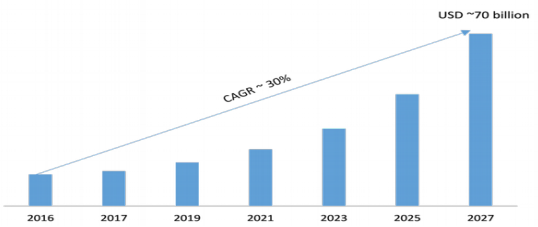

ILUSTRACIÓN 4: EVOLUCIÓN DE VENTA DE DRONES EN USD

En la actualidad, el líder indiscutible en desarrollo y ventas de esta tecnología es la empresa china *DJI*, fundada hace solo 13 años, ha pasado de 20 empleados a 14000 en la actualidad, liderando el mercado asiático, europeo y americano. Si bien es cierto que actualmente existe una guerra comercial entre China y estados unidos, parece que los drones *DJI* no se ven afectados ya que el 75% de los *UAV* utilizados en los estados unidos son fabricados por esta marca, desplazando a la empresa Francesa *Parrot* en volumen de ventas.

2.3. Estado actual del desarrollo

Durante los últimos años tanto el mercado global como el nacional ha experimentado una extraordinaria evolución. Consolidando al sector del dron como uno de los negocios con mayor potencial de expansión, en el que la innovación y el continuo avance tecnológico se antojan como elementos fundamentales.

Cada día aparecen nuevas aplicaciones y servicios aplicables a los drones en el ámbito civil (Agricultura, minería, filmación, mantenimiento, medioambiente, obra civil, etc.…) que muestran, con su versatilidad y eficiencia, claras ventajas respecto a las soluciones tradicionales. Esto es posible gracias al continuo aumento experimentado en las prestaciones de estas aeronaves, que las posiciona como herramientas idóneas para su uso en gran parte de los sectores de producción y servicios.

Hay que indicar que además existen una cantidad destacable de proyectos las que uno se puede basar para la construcción de un dron, independientemente del número de hélices. Según el enfoque que hagamos de estos proyectos, se pueden dividir en dos categorías: Drones comerciales o Drones Open Source.

2.3.1. Drones comerciales

Al contrario que los drones de la comunidad *Maker*, los drones comerciales son drones privativos, que se venden como elementos cerrados y no admiten modificaciones sin perder la garantía de uso. Algunos de los drones comerciales más vendidos en los últimos años son:

El *Syma* X11C, se trata de un mini dron cuadricóptero manejado mediante control remoto, con cámara HD y que incorpora un giroscopio de 6 ejes para dotar de una mayor estabilidad al vuelo, capaz de hacer maniobras en 360º.

ILUSTRACIÓN 5: SYMA X11C

El *Ryze* Tello de DJI también se trata de un mini dron cuadricóptero con una autonomía de vuelo de 13 minutos, cámara HD de 720p, procesador Intel y estabilizador de vuelo. Se controla desde el teléfono móvil mediante una aplicación desarrollada por la propia empresa DJI

ILUSTRACIÓN 6: RYZE TELLO

El *Hubsan*, otro cuadricóptero con cámara HD, control por RC, estabilizador de vuelo de 6 ejes y con 7 minutos de autonomía. También es capaz de hacer *flips* en 360º

ILUSTRACIÓN 7: HUBSAN

El *kingstoy* mini RC es un mini dron muy ágil y rápido que cuenta con 6 ejes giroscópicos con aterrizaje automático y que se controla mediante RC

ILUSTRACIÓN 8: KINGSTOY

Como se puede observar, los drones más vendidos son todos cuadricópteros, esto es debido a que su manera de volar es la más estable de todas y requieren, por tanto, menos tiempo de aprendizaje para controlarlos.

2.3.2. Drones Open Source, los drones en la comunidad Maker

La comunidad Maker, podría decirse que es algo así como la comunidad hágalo usted mismo, que se apoya en los medios digitales colaborativos como base para el intercambio de conocimientos, de esto podemos deducir que esta comunidad se basa en las practicas open source para realizar sus proyectos, haciendo que sigan trayectorias similares a las de la programación. Podría decirse que este tipo de filosofía proporciona conocimientos accesibles por todo el mundo, como planos de impresión 3D de las piezas o software de control, y que pueden ser modificados.

En lo que atañe a este proyecto, se diseñara un dron basado en una RaspBerry pi, la cual es otro paradigma de la comunidad Open Source.

2.4. Desarrollo del Prone

Para la realización de este proyecto se barajan varias alternativas en cuanto al control del dron y a su estructura, en cambio, el tipo de dron (Cuadricóptero), la tarjeta procesadora será una Raspberry Pi 3b+ y la tarjeta navegadora una Acro Naze 32, recordar que la primera elección fue una Crius AIO, pero llego defectuosa y ubo que cambiar el desarrollo. Quedaría por tanto un esquema de construcción similar a este:

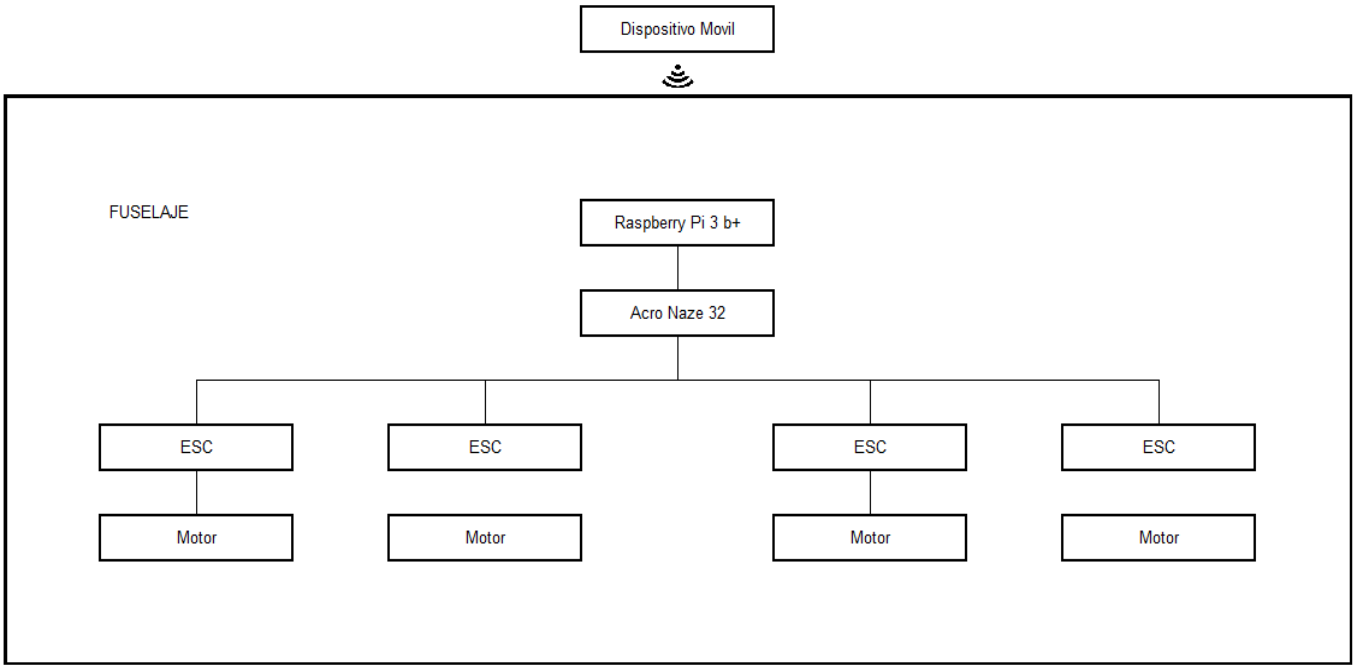

ILUSTRACIÓN 9: ESTRUCTURA DEL DRON

2.4.1. Tipo de Dron

El tipo de dron se tratará de un multicópielo, específicamente de un cuadricóptero, esto se debe a que tras analizar los pros y los contras de los distinto tipos de drones. A continuación, se muestra una tabla de pros y contras.

Ala Fija		Cuadricóptero		Multicópielo	
Pros	Contras	Pros	Contras	Pros	Contras
Gran autonomía de vuelo. Gran velocidad Muy ligero	No despegan en estático Incapacidad de pararse en el aire Diseño complejo	Buena estabilidad Bajo coste Buena autonomía de vuelo Poco peso	Incapacidad de volar cuando falla un motor	Mayor capacidad de carga Mayor estabilidad Capacidad de volar con un motor en fallo	Peso Coste Necesidad de mayor batería

Viendo esta clasificación, se puede llegar a la conclusión de que el cuadricóptero es la mejor opción al tener mayor autonomía de vuelo que el multicópielo aportando más estabilidad y maniobrabilidad que el ala fija y siendo el que menor coste tiene.

2.4.2. Procesador del Dron

El corazón del dron será una Raspberry Pi, que es una placa computadora (SBC) de bajo coste, consta de puertos USB, Ethernet, HDMI y dependiendo del modelo una cantidad de memoria RAM y un procesador distinto. La elección de la Raspberry pi se basa en su poder de procesamiento y su capacidad para crear la red WiFi con la que, en caso de considerarlo oportuno, conectar la APP de control del dron o enviar datos recogidos por los sensores del dron. Además, la bibliografía que consultar en este tipo de proyectos es muy extensa. En la figura 3 una Raspberry Pi

ILUSTRACIÓN 10: RASPBERRY PI

2.4.3. Tarjeta controladora

Como tarjeta controladora, en un principio se iba a usar una Crius AIO PRO, que se muestra en la figura 9, esta es una placa de control que dispone de una interfaz en serie múltiple y un interfaz I2C dedicado, además lleva un microcontrolador ATMega 2560 y un chip de memoria flas de 16bits.

La placa dispone de 8 salidas para motores, más que suficiente para este proyecto, 4 puertos serie. 2 puertos servo y 6 salidas analógicas. Además, es compatible con MegaPirateNG y Multiwii, softwares de control de la Crius que explicaremos a continuación.

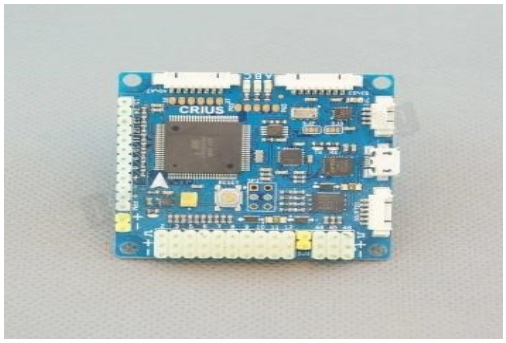

ILUSTRACIÓN 11: CRIUS AIO PRO

Como se explicará más adelante, debido a problemas técnicos se ha optado por cambiar la tarjeta controladora por una Acro Naze 32, que cuenta con un procesador de 32 bits, capacidad para hasta ocho motores y camara gumbal.

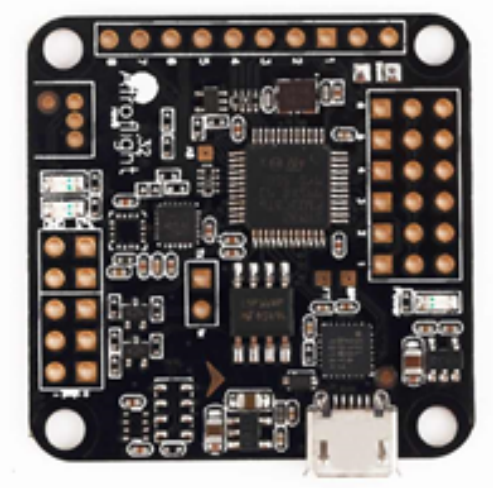

ILUSTRACIÓN 12: ACRO NAZE 32

2.4.3.1. Programa de calibración

La Crius utiliza el software llamado MegaPirateNG , que fue creado originalmente como un repositorio hijo del original Ardupilot, en base a esto, este software permite utilizar un código prácticamente idéntico en funcionalidades.12 Para cargar este software sería necesario un IDE propio de MegaPirateNG, el cual podemos ver en la siguiente figura.

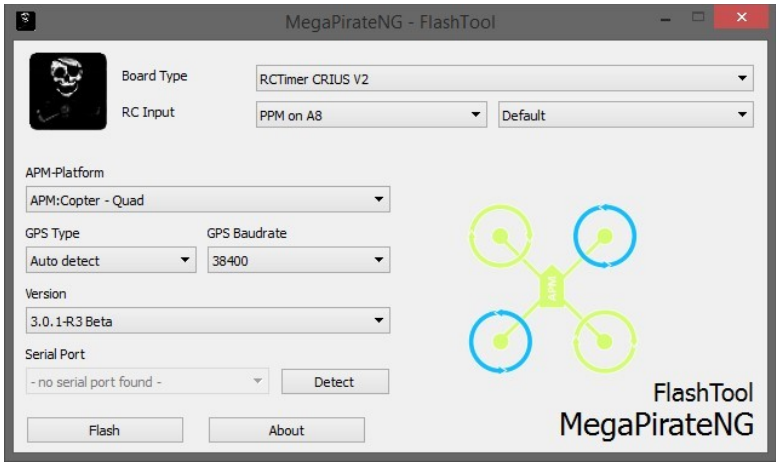

ILUSTRACIÓN 13: MEGAPIRATENG

En cambio, la nueva tarjeta que se va a utilizar utiliza un programa llamado baseflight para la calibración, esta tarjeta también se basa en el protocolo multiwii para la recepción de órdenes de control, por lo que realmente solo difiere el programa usado. En la siguiente figura se observa el entorno grafico de baseflight.

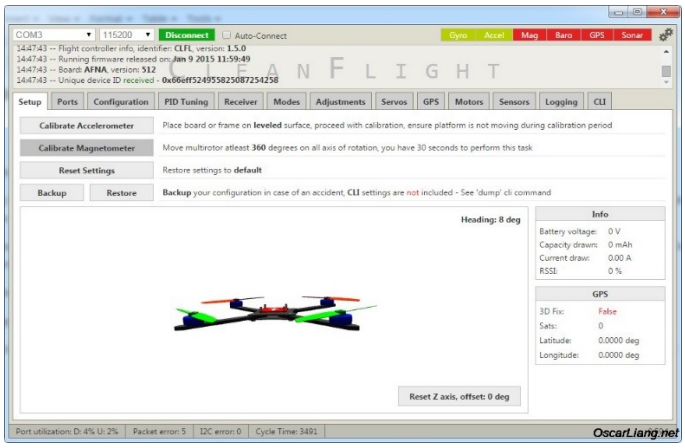

ILUSTRACIÓN 14: BASEFLIGHT

2.4.4. MultiWii

Se trata de un esfuerzo por aprovechar los sensores de los controladores de una consola Nintendo Wii para controlar el vuelo de un cuadricóptero. El proyecto ahora ha madurado un poco y se puede decir que la controladora MultiWii SE es de uso generalizado.

Es otra solución de firmware, la más sencilla de configurar pero la más limitada en funciones, aunque se consigue un vuelo muy estable, la mayor problemática es la adquisición de datos, ya que habría que modificar la manera de recepción de datos, ya viene no viene configurado, esto se solucionaría a través la Raspberry Pi que enviaría los datos de manera que el programa funciones, por lo que solo habría que cambiar el archivo de configuración indicando las especificaciones del dron que vamos a construir.

ILUSTRACIÓN 15: IDE MULTIWII

2.4.5. Estructura o Core

Es este punto se presentan dos posibilidades, la primera sería comprar la estructura del dron, lo que facilitaría mucho el proceso constructivo del mismo, pero nos limitaría a la hora de personalizar el dron y la segunda opción sería obtener el Core mediante impresión 3D lo que permitiría una mayor profundidad de personalización, pero llevaría más tiempo. Dentro de esta opción también podemos buscar planos de impresión Open Source que se adapten a nuestro diseño.

2.4.6. Tipo de control

En este punto la decisión se tomará en función del tiempo disponible, si bien es verdad que la opción de una aplicación es la preferida y que existen diferentes códigos de aplicaciones open source que se podrían modificar y adaptar para este proyecto, si el tiempo no fuera suficiente para la óptima configuración de esta se optaría por un control RC. Un código Open source que se podría utilizar es Apache Cordova y como control RC un Goolsky a 2,4G.

2.4.6.1. Mando RC Goolsky

Este es un mando RC con capacidad anti-jamming que funciona a una frecuencia de 2.4GHz, tiene muy bajo consumo de energía y cubre bastante bien todo el ancho de banda de la antena, tiene 4 canales y funciona en la banda 160m. El control es mostrado a continuación.

ILUSTRACIÓN 16: RC GOOLSKY

2.4.6.2. App mediante Apache Cordova

Es un entorno de desarrollo de aplicaciones móviles utilizando CSS3, HTML y JavaScript en vez de Apis, esto permite crear aplicaciones hibridas entre app móviles y app web. Este entorno puede ser extendido con complementos nativos de los móviles, a los que se les llama a través de JavaScript y permiten el uso de elementos como son el acelerómetro, la cámara o la brújula. Actualmente soporta las siguientes características, aplicables al proyecto, de teléfonos Android:

- Acelerómetro
- Camara
- Brújula
- Geolocalización
- Archivos
- Almacenamiento

3. Diseño del Dron

3.1. Introducción

Este apartado se dividirá en 3 partes, elección del tipo de dron, elección de los componentes, y diseño estructural, en cada uno se hará un análisis de las diferentes opciones existentes para diseñar del dron, explicando el uso de cada una de las partes y la elección tomada.

Comenzaremos explicando el tipo de dron elegido, a continuación, pasaremos a explicar el diseño estructural y por último explicaremos los materiales y componentes elegidos.

3.2. Tipo de dron

Como se explicó en el capítulo del Estado del Arte los drones se pueden agrupar en dos grandes grupos, los drones de ala fija y los drones de ala rotatoria:

3.2.1. Drones de ala fija

Cuya principal característica es que necesitan una velocidad de despegue inicial para que se produzca la sustentación en el ala, es decir, un dron de ala fija debe despegar como un avión convencional, por ello suelen ser lanzados con catapulta o a través de la fuerza del brazo, aunque los más modernos tienen sistema de despegue mediante pista de aterrizaje convencional.

Este tipo de dron se ha descartado por la dificultad de construcción, así como el coste, los diseños de este tipo de UAV requieren de un estudio de aerodinámica para que el diseño de las alas produzca la sustentación deseada y no provoque una caída libre del dron en pleno vuelo.

ILUSTRACIÓN 17: CATAPULTA

ILUSTRACIÓN 18: LANZAMIENTO

ILUSTRACIÓN 19: PISTA DE ATERRIZAJE

3.2.2. Drones de ala rotatoria

Los drones de ala rotatoria, normalmente, se dividen en 4 grupos dependiendo del número de rotores que tengan: Tricótelo, Cuadricóptero, Hexacóptero y Octacóptero.

ILUSTRACIÓN 20: TRICÓPTERO

ILUSTRACIÓN 21: CUADRICÓPTERO

ILUSTRACIÓN 22: HEXACÓPTERO

ILUSTRACIÓN 23: OCTACÓPTERO

Los drones de ala rotatoria también son conocidos como drones multirotor, estos UAV se basan, al igual que los de ala fija, en el principio de sustentación, este principio se basa en el movimiento relativo entre el aire y el perfil de la hélice o del ala, en el caso de los drones de ala fija sería el movimiento del viento a altas velocidades a través del perfil del ala lo que produciría esta sustentación.

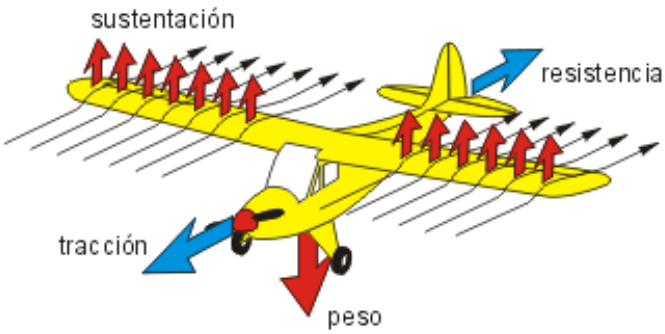

ILUSTRACIÓN 24: SUSTENTACIÓN ALA FIJA

En cambio, los de ala giratoria, se regiría según el teorema de Bernoulli, que expone que a mayor velocidad menor presión, entonces como el aire recorre un camino más largo por arriba de la hélice

que por abajo existe una mayor velocidad y menor presión, luego la mayor presión que se genera debajo "empuja" para arriba el perfil y por tanto el UAV, grosso modo sería el movimiento de la hélice producido por el rotor lo que produciría esta sustentación, esto permite que su despegue y aterrizaje sea vertical, los dota a estos drones de una excelente maniobrabilidad y les permite mantener un vuelo fijo.

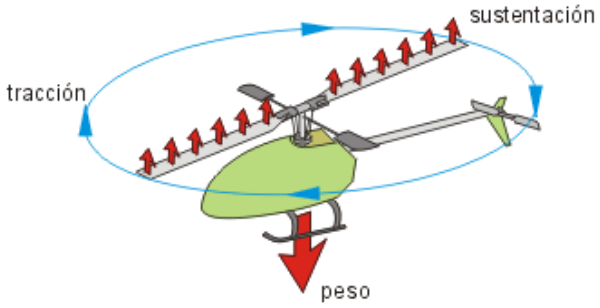

ILUSTRACIÓN 25: SUSTENTACIÓN ALA ROTATORIA

La configuración de hélices más utilizada es la del cuadricóptero, que consta de cuatro rotores colocados en las extremidades de la estructura del dron, las formas más comunes son en X o en H, la diferencia más sustancial entre estas dos configuraciones es la posición de la cámara, en la forma X la cámara normalmente suele estar colocada en la base del dron, en cambio, en la forma H suele estar colocada en la parte delantera, se podría decir que los de forma H serían ideales para las aplicaciones que requieran vista en primera persona, mientras que los de forma X serían perfectos para fotografía.

ILUSTRACIÓN 26: TIPO H

ILUSTRACIÓN 27: TIPO X

Para la realización de este Trabajo Fin de Máster, de ahora en adelante TFM, se ha decidido utilizar un cuadricóptero con la configuración de tipo X, las razones principales han sido las siguientes:

- *Gran cantidad de material didáctico y de apoyo*: Existe una gran base de conocimiento sobre el diseño, construcción y configuración de drones cuadricópteros, lo que permite resolver muchas de las dudas que se plantean en este TFM.

- *Configuración con la mejor relación estabilidad/peso*: Comparándolo con sus hermanos, (Tricóptero, Hexacóptero y Octacóptero), es el que mejor relación peso/estabilidad tiene, esto se debe a que sus 4 hélices le dan mayor estabilidad que al Tricóptero, tiene una mayor fuerza de empuje y su peso no llega a los niveles del Hexacóptero y Octacóptero.

- *Buena potencia de elevación*: Debido a las cuatro hélices tiene una buena potencia de elevación, no llega a niveles de sus hermanos mayores, pero esto también repercute en un uso menor de la batería

- *Gran maniobrabilidad*: Permite vuelos controlados, con mucha precisión, como por ejemplo vuelos a baja velocidad con mayor precisión a la hora de realizar un trabajo que merite precisión exacta además pueden mantener la posición en vuelo estático, lo que les permite realizar trabajos que requieran gran estabilidad y exactitud.

3.3. Modo de vuelo

En un UAV de ala rotatoria el único elemento controlable y que a su vez interacciona con el medio son los motores mediante sus respectivas hélices, estas realizan esta actuación con el medio mediante el sentido de rotación y la velocidad de giro de los motores.

Existen tres variables principales que definen el vuelo de un dron, *pitch* (cabeceo), *roll* (alabeo) y *yaw* (guiñada), que se corresponden con los principales ángulos de Euler, además también aparece la variable altitud, que corresponde simplemente con la distancia al suelo:

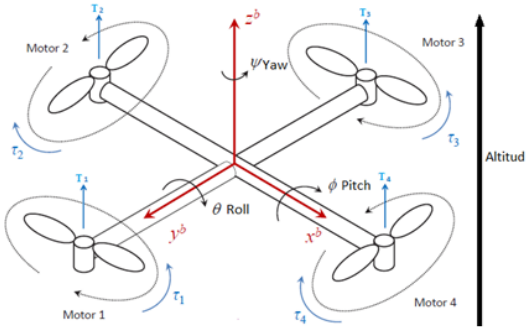

ILUSTRACIÓN 28: VARIABLES DE VUELO DEL DRON

Como se puede observar, se han numerado los motores para facilitar su identificación, de tal manera que la siguiente explicación de los distintos movimientos se entienda de mejor manera. A continuación, se explicarán los 4 movimientos de los que dispone el dron:

3.3.1. Altitud

Para variar la altitud en un dron solo hay que aumentar o disminuir la velocidad de los 4 motores por igual, consiguiendo mayor altura a mayor velocidad y menor altura a menor velocidad, la posición base son los 4 motores aplicando la misma fuerza de empuje, lo que permite:

- Estabilizar el dron
- Aumentar la altura
- Disminuir la altura

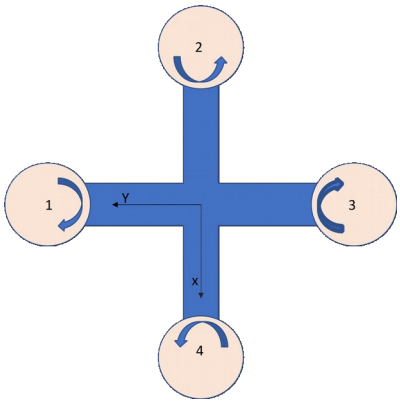

ILUSTRACIÓN 29: CONTROL DE ALTURA

3.3.2. Roll

El movimiento de *roll* es la variación con respecto al eje X, es decir, aplicando rotación en el eje y, para ello se juega con las potencias del motor 4 y 2, al girar los motores con varias velocidades distintas, se genera un desequilibrio con respecto al eje, gracias a esto se crea una componente horizontal, en sentido X, que genera el movimiento en el sentido hacia donde se ha inclinado el dron.

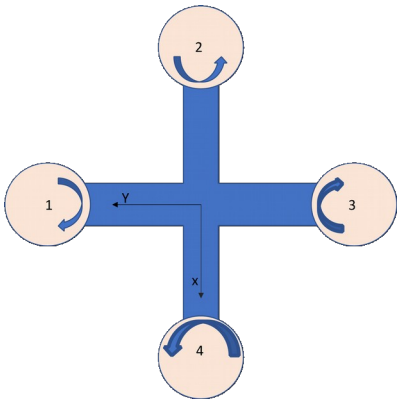

ILUSTRACIÓN 30: CONTROL DE ROLL

3.3.3. Pitch

El movimiento de *pitch* es la variación con respecto al eje y, es decir, aplicando rotación en el eje x, para ello se juega con las potencias del motor 1 y 3, al igual que en el caso del roll, al girar los motores con varias velocidades distintas, se genera un desequilibrio con respecto al eje y gracias a esto se crea una componente horizontal, en sentido Y, que genera el movimiento en el sentido hacia donde se ha inclinado el dron.

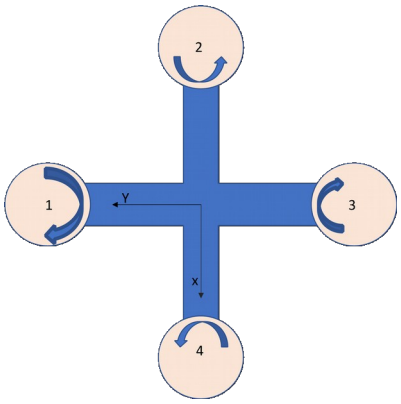

ILUSTRACIÓN 31: CONTROL DE PITCH

3.3.4. Yaw

Este movimiento es distinto a los dos anteriores ya que no genera translación alguna. Como en los movimientos anteriores se basa en la variación de giro de los motores, solo que esta vez actuando sobre los cuatro a la vez. Este movimiento es un movimiento de lado a lado desde su centro de gravedad. Para ello, los motores actúan por pares y tienen sentidos de rotación distintos, en este caso el motor 1 y el 3 giran en sentido horario y el 2 y el 4 en sentido antihorario.

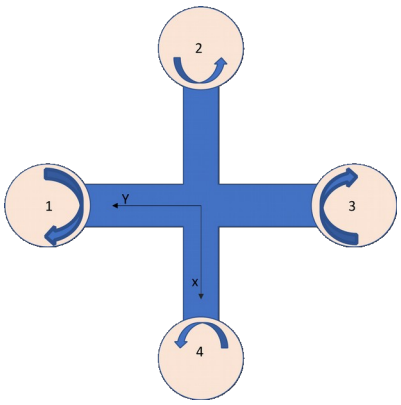

ILUSTRACIÓN 32: CONTROL DE YAW

3.4. Diseño estructural

3.4.1. Estructura funcional

La estructura funcional estará compuesta por los diferentes componentes que en este apartado se explican. Uno de los elementos más importantes son los 4 motores con velocidades de giro independientes controladas por una tarjeta controladora de vuelo sobre la que programamos el código que convertirá los datos de entrada en valores de tensiones que los ESC convertirán en valores de velocidad para los motores. Estas tensiones tienen dos orígenes diferentes de datos de entrada: los sensores dinámicos, el GPS y la brújula, de la tarjeta controladora y el control manual. Los primeros indican la orientación espacial de la aeronave, y el segundo será la interfaz con la persona que lo esté pilotando. Esta persona envía datos

inalámbricamente mediante un módulo de comunicación. Toda la electrónica mencionada anteriormente se alimentará a partir de una batería. Por último, todos estos elementos irán sujetos mediante un chasis.

3.4.2. Chasis

Se ha elegido un chasis en forma de X, ya que es la configuración más cómoda a la hora instalar tanto la tarjeta controladora como la Raspberry pi, además, se busca utilizar un chasis con el menor peso posible, pero con el espacio suficiente para poder instalar todos los componentes, para ello se han investigado distintos materiales:

- ***Aluminio***: El aluminio es muy útil en ingeniería debido a sus propiedades mecánicas como una alta resistencia mecánica que puede llegar hasta los 690 MPa, dependiendo de la aleación que disponga el metal y una densidad baja de 2.7g/cm^3, que lo hace muy maleable. Además, se trata de un metal no ferromagnético, por lo que no causa interferencias con los elementos electrónicos.
- ***Poliestireno***: Este material resulta de la extrusión del poliestireno en presencia de un gas espumante, lo que forma una espuma con una consistencia rígida. La principal propiedad que presenta es su bajísima densidad de 0,033 g/cm^3, lo que genera una baja resistencia, en torno a 250 KPa, además también es un gran aislante térmico. También se usa como protector.
- ***Polímeros plásticos***: Conocido común mente como plástico, este material se produce mediante la polimerización, el tamaño y la estructura de la molécula del polímero determinan las propiedades de los distintos plásticos, a lo que si a continuación aplicamos presión y calor se obtiene el producto final, que es este caso se trata de Polietileno de alta densidad o HDPE, tiene una densidad en torno a 0.97 g/cm^3 y un peso específico en torno a 0.93, sus resistencia es de 50MPa.
- ***Fibra de vidrio***: La fibra de vidrio está formada por una gran cantidad de fibras extremadamente finas de vidrio, esto le proporciona una alta densidad, en torno a 2.58g/cm^3 siendo su resistencia longitudinal 1,08 nPa, pero la transversal esta sobre los 50 MPa.
- ***Fibra de Carbono***: Es una fibra constituida por finos filamentos de carbono de entre 5- 10 µm de diámetro. Cada filamento está formado a su vez por millares de fibras de carbono. Tiene una baja densidad, de unos 1.6g/cm^3 y sus propiedades resistivas son similares al acero, su resistencia longitudinal anda sobre los 1,1 nPa, y la transversal, al igual que la fibra de vidrio, esta sobre los 50 MPa

3.4.2.1 Chasis seleccionado

Tras analizar los diversos materiales, y buscar un chasis que se amolde a el proyecto, se ha decidido usar un chasis de polímero plástico reforzado con fibra de vidrio, este chasis tiene un peso aproximado de 350g y una distancia entre ejes del motor del 45cm.

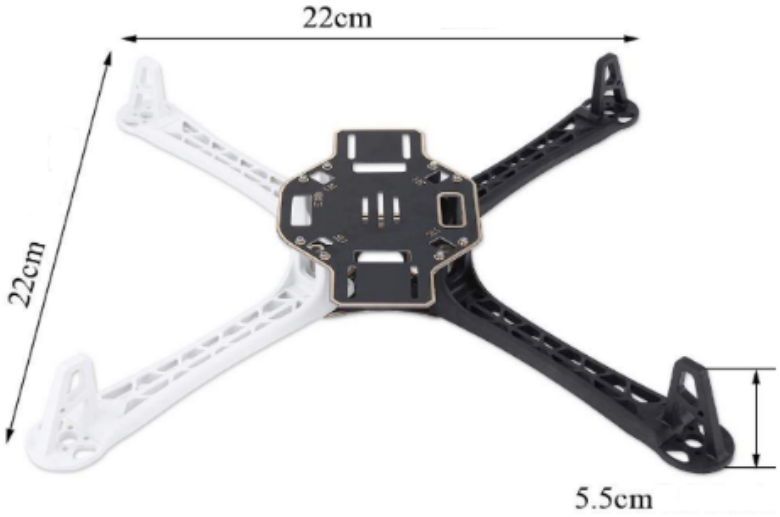

ILUSTRACIÓN 33: CHASIS

La estructura X es flexible y estable, lo que permite que el dron puede alcanzar buenas velocidades, además cuenta con espacio suficiente para instalar la tarjeta controladora de vuelo y la Raspberry Pi, así como el resto de los componentes. Además, los tableros sobre los que iría la electrónica son placas de circuito impreso, o PCB, lo que permite soldar directamente sobre ellos.

3.4.3. Ordenador central

El ordenador central será el encargado del procesamiento de datos ajenos a la estabilidad del vuelo, así como el que se encargará de la comunicación entre el dispositivo móvil y el dron. Para ello, utilizaremos una placa de tipo SBC, existen una gran variedad de marcas y de placas:

- ***Raspberry Pi 3 B+:*** La Raspberry pi 3 B+ es el segundo evolutivo del modelo 3 de Raspberry, es el miniordenador más conocido del mundo, cuesta en torno a los 40€ y cuenta con las siguientes características:
    - *CPU: Broadcom BCM2837B0, quad core Cortex-A53 de 64 bits a 1,4 GHz*
    - *RAM: 1 GB LPDDR2*
    - *LAN: Gigabit Ethernet*
    - *WiFi: dual 2,4 GHz y 5 GHz. 802.11 b/g/n/ac*
    - *Bluetooth: 4.2*
    - *GPIO de 40 pines*
    - *USB: 4 x USB 2.0*
    - *HDMI: tamaño estándar*
- ***Orange Pi Zero:*** Es uno de los modelos más baratos de este tipo de miniordenadores, comparándola con las Raspberry pi 3B+ se observa que, aunque es más barata, sus características son:
    - *CPU: Allwinner H2+ ARM Cortex-A7 quad core a 1.2 GHz*
    - *RAM: 256 o 512 MB*
    - *LAN: 10/100*
    - *WiFI: 802.11 b/g/n*
    - *GPIO: 23 pines compatible con Raspberry Pi B+*
    - *USB: 2 x USB 2.0, uno como host y el otro funcionando como OTG*
    - *HDMI: carece de salida de vídeo*

- ***Pine A64+:*** Este modelo tiene un procesador más potente y el doble de memoria RAM que la Raspberry Pi 3 B+ pero también es algo más cara, como dato interesante tiene dos GPIO, lo que permite conectar más componentes a este mini PC, sus características son:
    - *CPU: Allwinner ARM Cortex-A53 a 1,2 GHz*
    - *RAM: 2 GB DDR3*
    - *LAN: Gigabit Ethernet*
    - *WiFi: 802.11 b/g/n*
    - *Bluetooth: 4.0*
    - *GPIO: 1 Euler bus y otro Pi-2 bus*
    - *USB: 2 x USB 2.0*
    - *HDMI: tamaño estándar*
- ***Asus Tinker Board:*** 30 € más cara que la Raspberry Pi, cuenta con un procesador más potente y el doble de memoria RAM, sus características son:
    - *CPU: Allwinner ARM Cortex-A53 a 1,2 GHz*
    - *RAM: 2 GB DDR3*
    - *LAN: Gigabit Ethernet*
    - *WiFi: 802.11 b/g/n*
    - *Bluetooth: 4.0*
    - *GPIO: 1 Euler bus y otro Pi-2 bus*
    - *USB: 2 x USB 2.0*
    - *HDMI: tamaño estándar*
- ***Banana Pi m64:*** Este miniordenador cuenta con una potencia similar, pero tiene el doble de memoria RAM para las aplicaciones, cuesta 20 euros más que la RaspBerry, tiene las siguientes características:
    - *CPU: Allwinner ARM Cortex-A53 a 1,2 GHz*
    - *RAM: 2 GB DDR3*
    - *LAN: Gigabit Ethernet*
    - *WiFi: 802.11 b/g/n*
    - *Bluetooth: 4.0*
    - *GPIO: 1 Euler bus y otro Pi-2 bus*
    - *USB: 2 x USB 2.0*
    - *HDMI: tamaño estándar*
- ***Odroid xu4q:*** Creada por *hardkernel*, cuenta con un procesador Octacore lo que le da mucho más musculo de procesado que a la Raspberry, cuentan 2 GB RAM DDR3 y dos USB 3.0 y del precio ronda los 120€
    - *CPU: Samsung Exynos 5422 octa core Cortex-A15 a 2 GHz + Cortex -A7 a 1 GHz*
    - *RAM: 2 GB LPDDR3*
    - *LAN: Gigabit Ethernet*
    - *WiFI: No*
    - *Bluetoth: No*
    - *GPIO: 30 pines*
    - *USB: 2 x USB 3.0 + 1 x USB 2.0*
    - *HDMI: 1.4a tamaño estándar*

3.4.3.1. Ordenador central selecionado

La Raspberry pi será la placa computadora seleccionada para hacer de "*cerebro*" del dron, su capacidad computacional, su poco peso y su diseño reducido, han sido datos determinantes en la selección de esta. La Raspberry pi salió al mercado en febrero de 2012, diseñada y fabricada casi en su totalidad por la *Raspberry Pi fundation*, ha sido un hito en la informática y la electrónica, un ordenador funcional y con una potencia aceptable por solo 35€. Existen varios modelos en el mercado, la decisión sobre utilizar uno u otro modelo se centró en los dos siguientes:

- *Raspberry Pi ZERO*: Este modelo es la versión más reducida, tanto en tamaño y peso, como en precio. Cuenta con las funcionalidades de sus predecesoras, aunque no llega al nivel de la Raspberry Pi 3 B+. La potencia de esta placa viene dada por un procesador ARM11 de 1 GHz, cuenta con 512MB de memoria RAM. Además, dispone de los siguientes puertos, esta placa cuenta con un Mini-HDMI de 1080p, dos Micro USB uno para la alimentación y otro para la transferencia de datos y una entrada ara una tarjeta microSD. Hay que destacar que su peso es de solo 9 gramos.

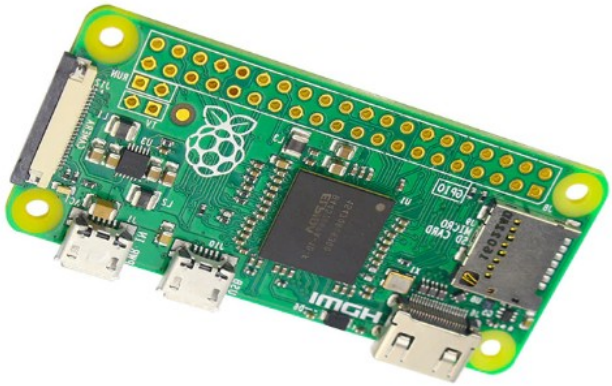

ILUSTRACIÓN 34: RASPBERRY PI ZERO

- *Raspberry Pi 3 B+*: Este modelo llegó al mercado por sorpresa en marzo de 2018. Después de la evolución más importante a nivel de potencia y posibilidades que se dio con el modelo 3, se da un paso más con esta actualización aumentando sus prestaciones. Lleva un nuevo procesador Broadcom BCM2837B0, Cortex-A53 Soc. de 64 bits con cuatro núcleos que funciona a 1,4 GHz, el cual puede llegar a ser 10 veces más potente que el original, banda dual de 2.4Ghz y LAN inalámbrica de 5GHz. Incorpora los nuevos componentes para la conectividad, también se ha cambiado la fuente de alimentación siendo necesario utilizar una alimentación de como mínimo, 5.1V y 2.5A para poder sacar todo el rendimiento.

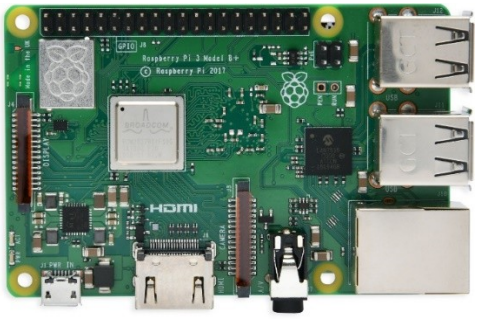

ILUSTRACIÓN 35: RASPBERRY PI 3 B+

Finalmente, se va a utilizar este último modelo, ya que, aunque el peso es mayor que en la Raspberry Pi ZERO, las prestaciones computacionales que nos ofrece superan con creces a las del otro modelo.

3.4.4. Controladora de vuelo

La controladora de vuelo es la parte más importante de un UAV, es el responsable de que todo funcione de manera correcta, se podría decir que es el cerebro del dron. Este controlador es una tarjeta integrada que contiene un procesador capaz de interpretar las señales recibidas (mediante un receptor RX, un procesador, etc..) y traducirlas a señales que entiendas lo controladores electrónicos de velocidad o ESC de los que hablaremos más adelante.

Como se comenta en el párrafo anterior, la controladora es el cerebro del dron, contiene todos los sensores necesarios para el velo (Giroscopio, GPS, barómetro, acelerómetro, etc…) o al menos los pines para conectarlos además de todos los periféricos que pueda llevar como son la telemetría o la caja negra. Las tarjetas más modernas cada vez tienen más puertas serie para incorporar un mayor número de dispositivos, así pues, hoy en día nos podemos encontrar cámaras HD controladas por radiocontrol a través de la controladora.

En el mercado actual hay una gran variedad de tarjetas controladoras de vuelo, para la realización de este TFM se busca que él controlador sea capaz de controlar los 4 rotores y que al ser posible tenga licencia pública, por ello se han mirado los siguientes modelos atendiendo a nuestras necesidades.

- *OMNIBUS F4 PRO V2*: Consta de 6 salidas PWM para motores, tiene un giroscopio MPU6000 y es configurable desde betafligth, como principales características son que puede sportar un voltaje alto de entrada, contiene una ranura para una tarjeta microSD para actuar de caja negra e incluye un sensor de corriente.
- *SP RACING F3 ACRO 6DOF*: No dispone de OSD integrado, es copia del famoso controlador de vuelo SPRacing F3. Tiene giroscopio MPU6050 que funciona a través del protocolo lento I2C, es un modelo viejo, pero incluye todas las características básicas de una controladora: 3UARTS, soporta SBus, SumH, SumD, Spektrum1024 / 2048, XBus, PPM, receptores PWM, se puede conectar tiras LED, zumbador, GPS y sonar, además tiene 16 PWM estándar.
- *LUX V1*: Es una copia del controlador de vuelo Lumenier LUX original, está basada en el procesador F3 e incluye un giroscopio MPU6500, tiene sensor de correinte, voltaje de la batería, entrada SUMD y es compatible con Betaflight y raceflight. Consta de 4 salidas PWM para los motores.

- *CRIUS AIO PRO*: Este es un proyecto *open source* que comenzó usando los controladores de la wii para hacer una placa controladora, consta de 8 salidas Axis para el motor y 8 canales de entrada, con teglador de voltaje y chip de memoria flash de 16Mbit de forma integrada para grabar los datos, usa un procesador de 32 bits STM 32.
- *NAZE32 REV 6*: Esta tarjeta cuenta con un sensor MPU 6500, con configuración de hasta 8 motores, tiene un procesador interno STM32F103CB de 32 bits, indicador de voltaje de batería, conector para cámara Gimbal y se configura con baseflight. Se podría decir que es una hermana de la CRIUS ya que en sus inicios se basó en el proyecto MultiWii del que nació la CRIUS.

3.4.4.1. Controladora de vuelo seleccionada

En este punto hay que destacar que el diseño del dron ya ha sufrido un pequeño contratiempo, la tarjeta CRIUS AIO PRO ha llegado en mal estado y el proveedor no puede entregar otra hasta dentro de 2 meses, por lo que no deja más remedio que cambiar la controladora de vuelo a una NAZE32 REV 6, la configuración será ligeramente diferente y alargará un poco el proceso.

La tarjeta NAZE32 REV 6, tiene un procesador interno de 32 bits y capacidad de conexión de 8 motores, en la siguiente imagen se muestra un esquema de esta:

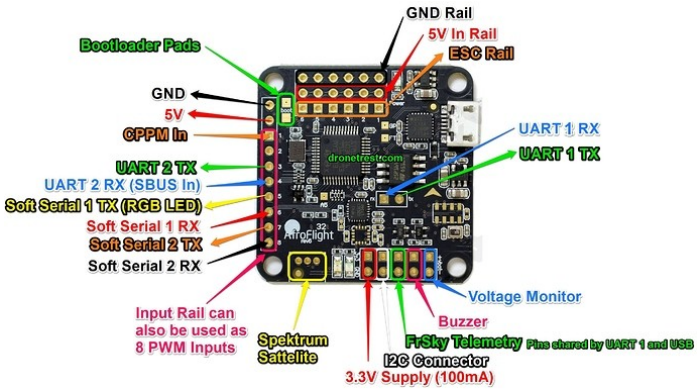

ILUSTRACIÓN 36: DESCRIPCIÓN NAZE32

Esquemáticamente, la configuración de la conexión de la tarjeta con los ESC, los motores y la batería quedaría de la siguiente manera:

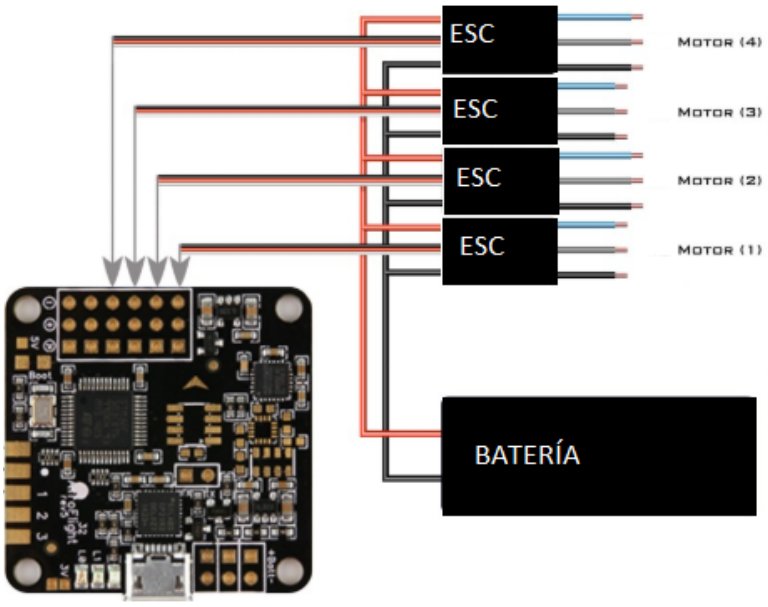

ILUSTRACIÓN 37: CONFIGURACIÓN FÍSICA NAZE32

La controladora se puede configurar a través del programa baseflight o a través del cleanflight, ambos son programas de licencia abierta y descargables desde la propia web del proveedor. Con ambos programas podemos usar el firmware multiWii, de hecho, ambos son una evolución de dicho firmware.

MultiWii es un proyecto que se basa en un software de propósito general que fue desarrollado para usarse con los giroscopios y acelerómetros del mando de la consola Nintendo Wii. Desarrollado sobre la plataforma Arduino, actualmente, Multiwii, es capaz de controlar desde 3 a 6 rotores y permite usar una gran variedad de sensores que permiten obtener una estabilidad bastante buena en los UAV, además, también permite realizar algunas acrobacias.

Dentro de las tarjetas controladoras corren unos parámetros PID (Proporcional Integral derivativo) que dependen del chasis del dron, del peso o del tipo de vuelo, básicamente se trata de un mecanismo de control que calcula el error entre un valor medido y el deseado aplicando una acción correctora. Los valores PID son correlacionados, si se cambia un valor hay que modificar el resto, cada valor significa:

- Proporcional: Es el valor encargado de la estabilidad y control
- Integral: Indica la velocidad con la que se repite la acción proporcional
- Derivativo: Cambia la fuerza aplicada a corregir el error de posición

3.4.5. Motores

Para seleccionar los motores es necesario estimar la carga que deberán mover los motores, para ello se ha realizado esta tabla con los pesos de cada uno de los componentes del dron y con la suma final del peso de cada uno de ellos, que será lo que deberán levantar los motores del dron:

	Peso (g)	Unidades	Total(g)

Chasis	353	1	353
Motor +ESC	28	4	112
Hélices	2	4	8
Batería	135	1	135
RaspBerry Pi	50	1	50
Naze32	5,3	1	5,3
Antena Wifi	31,8	1	31,8
Total			696

Para levantar la aeronave, sé necesita que los motores sean capaces de elevar 680 g sobredimensionado la medida. Sabemos que el peso es una fuerza(N) que son que en este caso vienen dados por la formula m(Kg)*g(m/s2), es decir, para que pueda sustentarse en el aire llevando una carga igual a su peso necesitamos que se cumpla:

$$E_{motor} = \frac{P_{AUV}}{4} * 2 = \frac{696}{4} * 2 = 348\,g$$

3.4.5.1 Motores brushless

Los motores Brushless son muy superiores a los motores con escobillas, principalmente debido a que este tipo de motores no tiene rozamiento, esto permite a los Brushless funcionar a mayor velocidad.

Este tipo de motores está formado por 3 electroimanes formados por 4 bobinas cada uno, el funcionamiento seria de la siguiente manera. Para facilitar la explicación se propone que la configuración interna de los motores es de la siguiente manera. El movimiento del motor se explicara en 6 pasos:

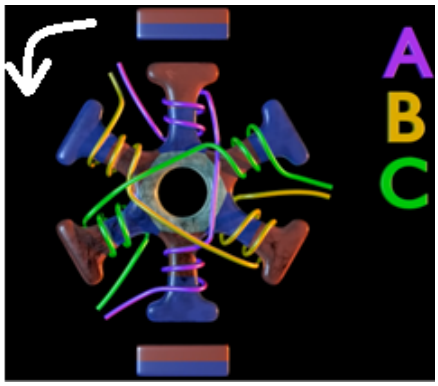

ILUSTRACIÓN 38: ESQUEMA MOTOR BRUSHLESS

1. Se activa el canal A que atrae el imán, siguiendo la dirección de la flecha, a esa bobina
2. Se activa el canal B, pero con la polaridad invertida para seguir moviendo el imán
3. Se activa el canal C y continua el movimiento
4. Se vuelve a activar el canal A pero con la polaridad invertida
5. A continuación, se activa el canal B para continuar el movimiento
6. Por último, el canal C con la polaridad invertida

Se podría decir que se van alternando activaciones de canales con distintas polaridades para hacer que el imán gire en círculos, además, dependiendo de la polaridad el motor puede girar en sentido horario o antihorario. En el siguiente apartado se explicara que es un ESC, que grosso modo es el encargado de activar dichos canales y controlar la velocidad de giro de uno de estos motores Brushless.

3.4.5.2 Motores seleccionados

El motor seleccionado ha sido el LHI 2212 920KV Brushless Motor, con una fuerza de empuje según fabricante de 350g, se ha elegido este modelo ya que al ser un modelo sin escobillas consigue más RPM y por tanto una mayor potencia, además de dotarlos de un menor tamaño y peso, pudiendo cumplir así las exigencias del modelado del dron. El motor es el mostrado en la siguiente imagen:

ILUSTRACIÓN 39: MOTOR LHI SIN ESCOBILLAS

3.4.6. ESC

Siguiendo con el siguiente elemento, los motores deben utilizar un dispositivo que controle el nivel de voltaje que le llegue para poder controlar su velocidad. Estos dispositivos se llaman control electrónico de velocidad (ESC). Los cuales constan de un circuito electrónico controlado por un microcontrolador que transforma la tensión en pulsos para los motores.

Existen dos tipos de ESC, para los motores con escobillas y para los motores sin escobillas que no son compatibles los unos con los otros, se diferencian de manera fácil porque los variadores para motores con escobillas llevan dos cables, mientras que los sin escobillas llevan tres. Como en este proyecto los motores seleccionados son sin escobillas, obviaremos el funcionamiento de los ESC para motores con escobillas. Los ESC tienen su funcionamiento basado en un pulso de energía que se envía al motor, que funciona en una base de frecuencia que generalmente va desde los 2KHz hasta los 12Khz.

Su funcionamiento es el de un interruptor ultrarrápido que se enciende y se apaga, entregando o no, la energía al motor, esto difiere mucho de las típicas resistencias variables que ajustan el voltaje entregado, como las vistas en Instrumentación Electrónica. Los ESC son controladores de modulación por ancho de puntos (PWM) para controlar motores eléctricos. El receptor del dron envía una señal PWM al ESC con variaciones de 1 a 2 milisegundos. En 1 milisegundo el motor estar parado, a 1.5 milisegundos el motor está a media potencia y a 2ms el motor funcionará al máximo.

Los variadores para motores sin escobillas crean una corriente alterna trifásica a partir de la corriente continua de la batería. Uno de los polos genera un pequeño voltaje proporcional a la velocidad de giro del motor conocido como fuerza electromotriz. Este voltaje le sirve al ESC para determinar la velocidad y la dirección de giro del motor, de manera que, con esta información, el ESC es capaz de enviar la corriente necesaria a los electroimanes del motor para que este funcione.

Estos ESC funcionan de la siguiente manera, el dispositivo recibe la energía de manera bruta de la batería por los cables positivo y negativo, además de la señal que proviene de la tarjeta controladora, dentro, un microprocesador recibe las señales y transmite por cada canal la energía necesaria para hacer que el motor se mueva según lo explicado en el punto anterior.

3.4.6.1. ESC seleccionados

Para este proyecto se han seleccionado unos ESC de la misma marca que los motores, LHI, concretamente el modelo SIMONK, con amperaje máximo admitido de 30A, son ideales para los motores LHI 2212 920KV, funcionas con baterías 3s o de 3 celdas.

ILUSTRACIÓN 40: ESC SELECCIONADO

3.4.7. Hélices

Como se explicó en el modo de vuelo, las hélices son las responsables de que el dron vuele debido al efecto Bernoulli. Su único uso no es el del transporte, hoy en día también tienen otros nombres y usos como turbina o ventilador, que se pueden emplear en la refrigeración, la compresión de fluidos, propulsión de vehículos o generación de electricidad.

En el mercado actual hay una gran variedad de hélices modernas, para un cuadricóptero necesitamos dos tipos de hélices, iguales en diseño distintas en forma, como las que se pueden ver en la imagen siguiente.

ILUSTRACIÓN 41: MODELO A

ILUSTRACIÓN 42: MODELO B

Una vez tenemos ambos tipos de hélices, deben quedar dispuestas en el dron en forma de cruz, de tal manera que las hélices sean iguales dos a 2 es decir las dos traseras modelo A las dos delanteras modelo B.

3.4.7.1. Hélices seleccionadas

Para este proyecto se han seleccionado unas hélices de 10" de longitud especiales para el modelo de chasis seleccionado, son las Bull Nos 1045 de la KEESIN4 hechas en plástico ABS, lo que les da una buena flexibilidad. Este modelo es el mostrado en la imagen anterior, con dos colores para diferenciar las hélices delanteras con las traseras.

3.4.8. Batería

La batería es, junto a la tarjeta controladora y el ordenador central, la parte más importante de un dron ya que es la encargada de suministrar la energía necesaria a todos los componentes del dron, además, el tiempo de vuelo dependerá directamente de ella.

En el mercado actual existen varios tipos de composiciones de baterías cada uno de ellos con sus ventajas e inconvenientes, aunque si bien es cierto, que el tipo de batería más usado y con mejor rendimiento en la actualidad es la batería LiPo o de polímero de iones de litio.

Estas son baterías recargables compuestas generalmente de varias celdas en paralelo para aumentar la capacidad de carga. Su funcionamiento es idéntico a las baterías de iones de litio, es decir, se basa en el intercambio de electrones entre el material del electrodo negativo y el del nodo positivo mediante un medio conductor.

Su uso en el mundo de los drones está muy estandarizado ya que al poseer una gran capacidad de almacenaje de energía y un peso muy ligero son ideales para el desarrollo de UAV. En cuanto a las características en las que hay que fijarse a la hora de seleccionar la batería apropiada están:

- *Tamaño*: Lógicamente el tamaño de la batería es un tema importante, es importante buscar una batería que se ajuste a las necesidades del proyecto sin excederse en tamaño y peso ya que eso repercutiría en una experiencia de vuelo insatisfactoria.
- *Voltaje de la batería*: Este parámetro indica el voltaje al que puede funcionar tu batería, si una batería tiene una mayor clasificación de voltaje, podrá hacer que los motores del dron giren con mayor potencia.
- *Conectores*: Este elemento es importante más por comodidad que por funcionalidad, si la batería trae conectores, esta será más fácil de instalar en el dron.
- *Tasa de descarga*: Es la cantidad máxima de energía que una batería puede descargar sin dañarse a sí misma.
- *Capacidad*: Este parámetro ayuda a saber cuál es el tiempo máximo de tiempo que una batería puede estar suministrando energía a un dispositivo, se mide en miliamperio hora (mAh).

Existen bario tipo de baterías, los más significativos son las baterías de Ion Litio, las de Litio polímero, las de níquel cadmio y las de níquel hidruro. A continuación, se explicarán algunas de sus características más relevantes:

3.4.8.1. Ion litio Cadmio (LiOn).

Este tipo de baterías, que se comenzaron a comercializar en los años 90, emplean un electrolito de una sal de litio que consigue una reacción electroquímica reversible entre el catodo y el anado, destacan por ser baterías muy ligeras con una gran resistencia a la descarga eléctrica y al efecto memoria. La mayor de las desventajas es que se degradan muy rápidamente y tienen mucha sensibilidad a las altas temperaturas pudiendo incluso producir una explosión de esta.

3.4.8.2. Níquel Cadmio (NiCd).

Este tipo de baterías fueron inventadas en 1899 por Waldmar Jungner. En la actualidad son las más utilizadas y en encuentran en casi un 70 % de los equipos a nivel mundial. Estas baterías proporcionan un alto nivel de energía a un bajo coste además de una gran durabilidad, el mayor problema es que sus materiales son muy tóxicos para las personas y el medio ambiente. Otro inconveniente es la baja densidad de energía en comparación con otras baterías además de una alta tasa de descarga.

3.4.8.3. Níquel hidruro (NiMH).

Parecidas a las baterías de níquel cadmio, cuentan con un ánodo de níquel, oxihidroxido de níquel, y un cátodo de una aleación de hidruro metálico, permitiendo así la eliminación del cadmio que es muy caro y peligroso para el medio ambiente. Este tipo de baterías poseen una gran capacidad de carga y un efecto memoria menor que las NiCd, su principal desventaja es que solo puede proporcionar un voltaje de 1.2 voltios con un amperaje de 2.9 amperios hora lo que la hace poco atractiva para el desarrollo de este dron.

3.4.8.4. Polímero de Litio (LiPo).

Estas baterías son pilas recargables compuestas normalmente por varias celdas secundarias idénticas en paralelo para aumentar la capacidad de descarga de corriente. Funcionan intercambiando electrones entre el electrodo positivo y el electrodo negativo al igual que las baterías de Ion Litio, realmente, las baterías LiPo y LiOn son muy parecidas en cuanto a ventajas y desventajas, en este proyecto se decidió utilizar una batería LiPo por dos motivos, las baterías LiPo son algo más flexibles por lo que resisten un poco mejor los golpes y además son el modelo más utilizado en el mundo del radiocontrol.

3.4.8.5. Batería seleccionada

La batería elegida ha sido una batería de LiPo, concretamente la Goolsky 3S 11.1V 1000mAh 30C, esta es una batería de 3 celdas con un amperaje por hora de 1000mAh, una tasa de descarga de 30C y un voltaje de 11,1V, sus dimensiones son 60*30*27 mm y su peso 85.2 g, lo que la hace ideal para un dron. Además, incluye conectores JST, lo que facilitara el montaje.

ILUSTRACIÓN 43: BATERÍA SELECCIONADA

Puede llamar la atención que la batería elegida solo sea de 1000mAh, lo que implica poco tiempo de vuelo, se decidió elegir esta batería ya que al ser un prototipo solo se busca que el dron pueda elevar el vuelo, aun así, para aumentar la autonomía solo sería necesario comprar una batería de hasta 10000mAh, pero siempre teniendo en cuenta que el peso de esta aumentara y repercutirá directamente sobre la autonomía.

3.4.9. Método de comunicación

Para seleccionar el método de comunicación, primero analizaremos las 3 posibles soluciones que se pueden implementar: radio control, bluetooth, WiFi.

- *Radio control (RC)*: Para este proyecto sería la solución más sencilla, ya que solo haría falta comprar un módulo receptor y un emisor y conectarlo. Un sistema radio control es un sistema que combina electrónica, electricidad y mecánica con el fin de mover un objeto.
    - *La electrónica*: se encarga de transformar los comandos dados en ondas de radio en el transmisor y a la inversa en el receptor,
    - *La electricidad*: encargada de proporcionar la energía necesaria a los dispositivos tanto el comando como el receptor y
    - *La mecánica*: Se encargada de mover los servos que dan las señales eléctricas demoduladas o decodificadas en movimiento mecánico.
- *Bluetooth*: Es una red inalámbrica de área personal, también conocida por sus siglas en ingles WPAN, que permite la transmisión de datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de 2,4GHz. Es un protocolo diseñado para el bajo consumo, pero tiene un corto alcance, de unos metros. Según su potencia de transmisión se puede clasificar en:
    - *Clase A*: con una potencia permitida de 10mW tiene un alcance de 100m
    - *Clase B*: Con una potencia permitida de 2.5mW tiene un alcance de 5 a 10 metro
    - *Clase C*: Con una potencia permitida de 1mW tiene un alcance de aprox. 1 metro
- *WiFi*: También conocido como WLNA, es realmente el estándar IEEE802.11, que básicamente es una familia de normas inalámbricas. Esta tecnología permite la conexión de dispositivos sin necesidad de cables. Existen diversos tipos de estándares red WiFi con distintas velocidades y radios de cobertura.

Normas (capa física y de acceso al medio)	Velocidad transmisión máxima (Mbps)	*Throughput* máximo típico (Mbps)	Numero máximo de redes colocalizadas	Banda de frecuencia	Radio de cobertura típico (interior)	Radio de cobertura típico (exterior)
IEEE 802.11a/h	54 Mbps	22 Mbps	14 (5.7 GHz)	5 GHz	85 m	185 m
IEEE 802.11b	11 Mbps	6 Mbps	3	2.4 GHz	50 m	140 m
IEEE 802.11g	54 Mbps	22 Mbps	3	2.4 GHz	65 m	150 m
IEEE 802.11n (40 MHz)*	>300 Mbps	>100 Mbps	1 (2.4 GHz) 7 (5.7 GHz)	5 GHz	120 m	300 m
IEEE 802.11n (20 MHz)*	144 Mbps	74 Mbps	3 (2.4 GHz) 14 (5.7 GHz)	2.4 GHz y 5 GHz	120 m	300 m

ILUSTRACIÓN 44: ESTÁNDARES WIFI

3.4.9.1. Método de comunicación seleccionada

La comunicación con el dron se realizará mediante WiFi, para ello dispondremos de una aplicación en el móvil que conectará a través de una red WiFi y se comunicará mediante WebSockets con la Raspberry Pi, para ello se creará en la placa un servidor DHCP que hará de punto de acceso.

Una vez que los datos sean recibidos por la Raspberry, esta conectara mediante un USB con la Naze32 que transformara las señales en pulsos para los motores. El esquema de comunicación que seguirá el dron, desde la APP hasta los motores sería el siguiente:

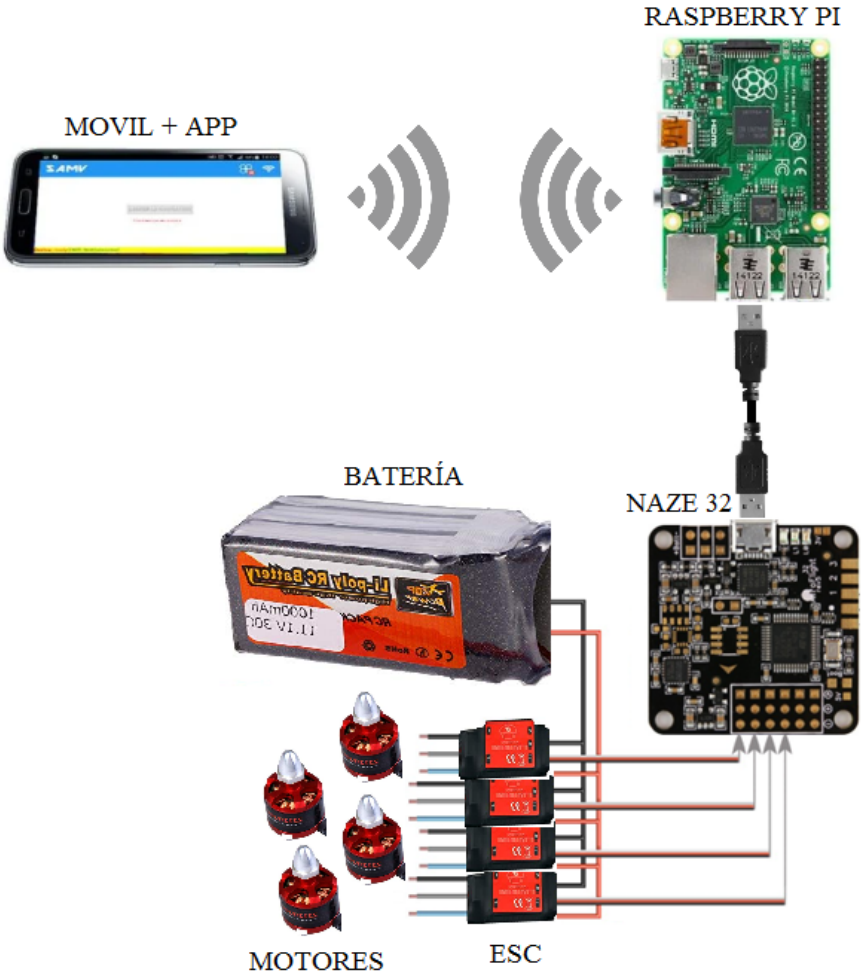

ILUSTRACIÓN 45: ESQUEMA GENERAL DEL DRON

4. Implementación del Dron

4.1. Introducción

En este apartado se explicará el proceso de construcción y de configuración del dron, para ello se hará una subdivisión de la siguiente forma:

- Construcción del dron
- Configuración de la Raspberry Pi
- Configuración de la comunicación
- Configuración de la NAZE32:

En cada parte se explicará el proceso seguido, los problemas encontrados y la solución realizada, además se añadirá una breve explicación del código de la APP y del servidos.

4.2. Construcción del dron

4.2.1. Breve explicación

La construcción del dron posiblemente sea el paso más sencillo de todos, ya que, aunque es un proceso largo, no requiere de grandes conocimientos, simplemente paciencia y algo de maña. En esta sección se explicará un poco el proceso realizado.

4.2.2. Montaje.

El primer paso para montar el dron es pre-estañar la placa de potencia para facilitar así la soldadura de la batería y los ESC, en las tres imágenes siguientes se muestra todo el proceso realizado

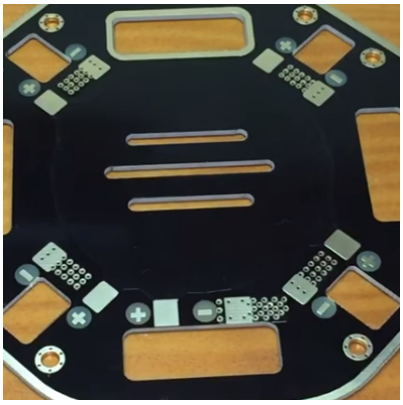

ILUSTRACIÓN 46: PBC INICIAL

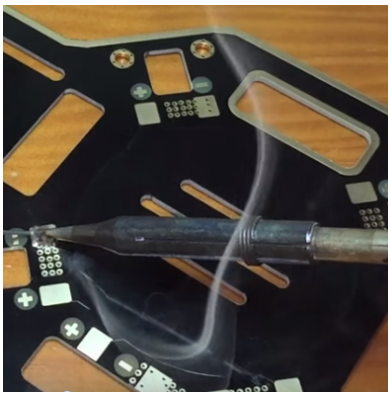

ILUSTRACIÓN 47: PRE-ESTAÑADO

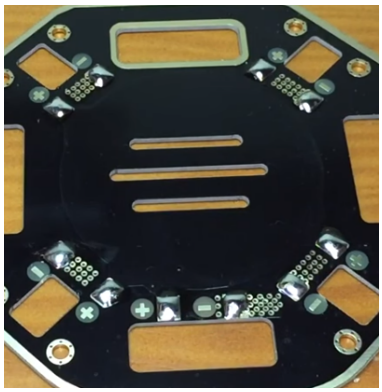

ILUSTRACIÓN 48: PBC FINAL

El siguiente paso sería soldar los ESC y los cables de la batería a la placa de potencia, también es necesario estañar unos conectores banana hembra al ESC ya que viene sin ellos. A la hora de estañar los cables es muy importante hacerlo bien, estañar el cable negro al negativo y el rojo al positivo.

En la imagen que se muestra a continuacion se muestra como queda las soldaduras con los cables ya estañados y las posiciones de los componentes en la placa, este proceso hay que hacerlo

con calma ya que cualquier fallo en el estañado puede producir cortos en la placa y generar un gran problema en el dron.

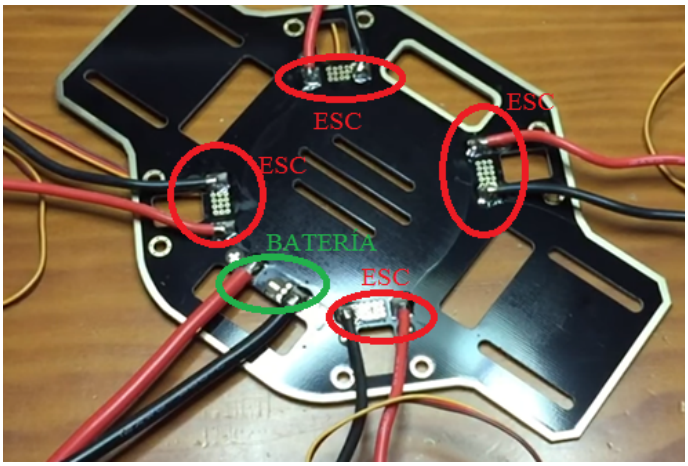

ILUSTRACIÓN 49: IMAGEN POSICIÓN COMPONENTES

Una vez tenemos preparada la placa, se pasaría a instalar la estructura, en este paso es importante calcular bien la longitud de los cables para que no sobre en exceso, aunque siempre se pueden enrollar alrededor de la estructura para que no queden cables colgando.

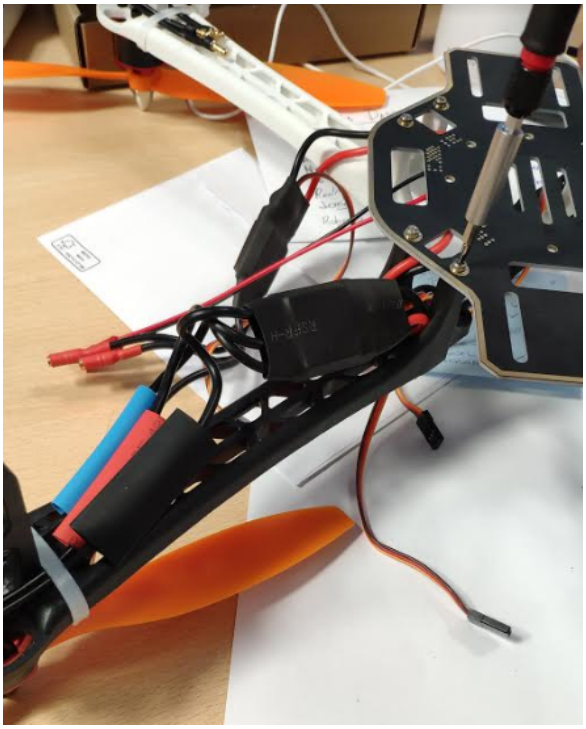

ILUSTRACIÓN 50: ESTRUCTURA Y CABLEADO

Después, se pasa a colocar los motores y conectarlos con los ESC, hay que fijarse bien en la posición en la que se colocan, ya que, como habíamos visto durante el desarrollo, la rotación de los motores es de dos en sentido horario y dos en sentido antihorario.

ILUSTRACIÓN 51: COLOCACIÓN MOTORES Y ESTRUCTURA

Después, pasamos a colocar el tren de aterrizaje, en este caso es un tren de aterrizaje casero formado por cuatro corchos, una solución muy práctica y muy barata.

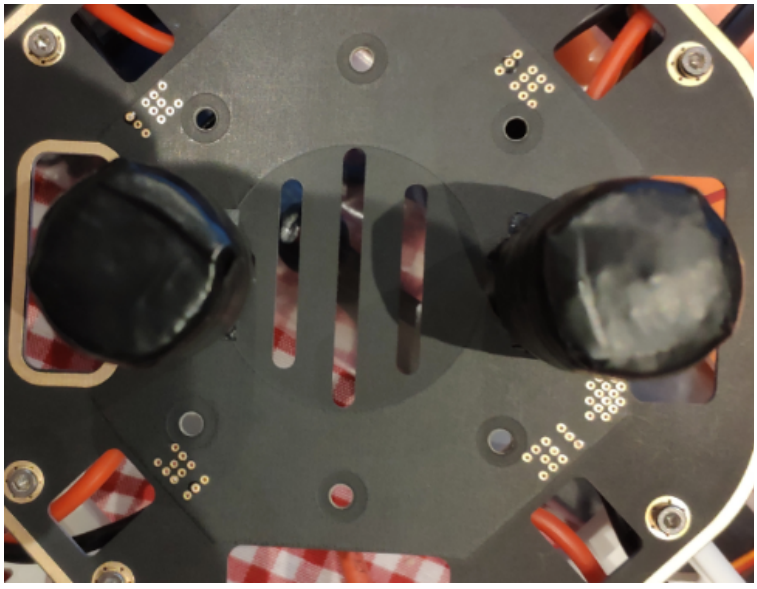

ILUSTRACIÓN 52: TREN DE ATERRIZAJE

A continuación, se colocan tanto la tarjeta controladora de vuelo como la Raspberry Pi, ambas se deben colocar juntas, una encima de otra, es ideal colocar unas gomas antivibración entre ellas para evitar problemas futuros. También tenemos que buscar una posición para la cámara y anclarla.

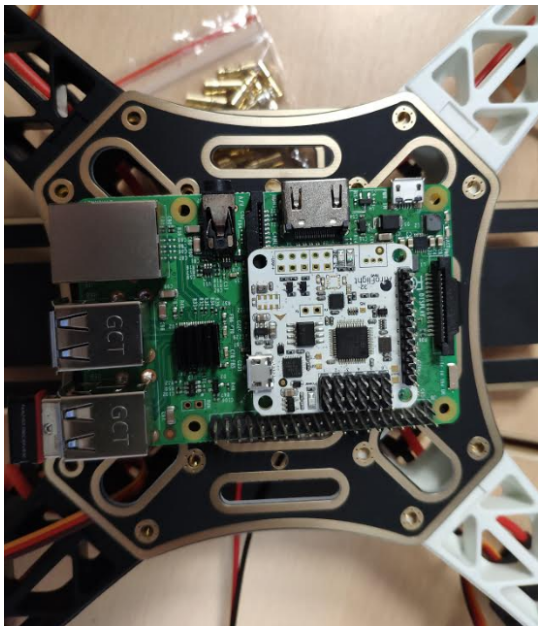

ILUSTRACIÓN 53: ACRO NAZE 32 Y RASPBERRY

Por último, colocamos la batería, esta ira colocada en la parte inferior del dron, para evitar que se estropee al aterrizar, es importante que el tren de aterrizaje sea lo suficiente mente largo para que la batería no choque con el suelo, para mayor fijación se usara velcro y una brida.

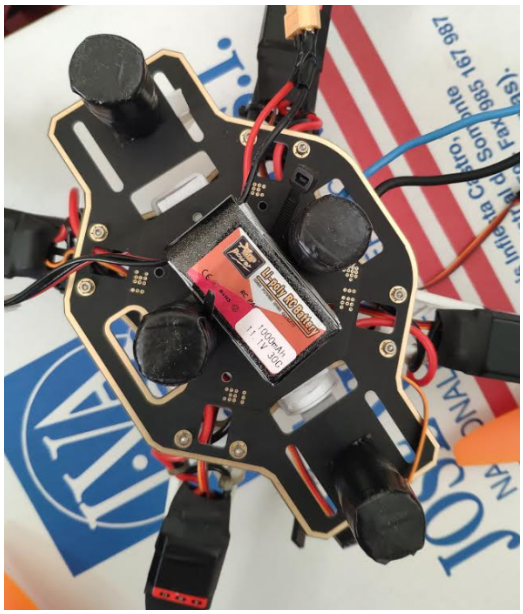

ILUSTRACIÓN 54: BATERÍA

4.2. Configuración de la Raspberry Pi

4.2.1. Breve explicación

La Raspberry pi es el eje central del proyecto, es la encargada de crear una red WiFi privada que no tendrá acceso a la red, simplemente nos servirá para la comunicación entre el dron y el teléfono móvil. Para ello, se debe utilizar un servidor DCHP. A continuación, inicializa un programa que crea

una conexión vía WebSockets permanente entre el servidor y el cliente, la aplicación, este programa será el encargado de recoger la información enviada desde el teléfono móvil y enviarla vía USB a la tarjeta controladora.

4.2.2. Instalando el Sistema Operativo

Lo primero de todo es instalar el SO en la Raspberry pi, en este caso se utilizara Raspbian, para ello, lo primero es descargarse la imagen del archivo, se pude obtener en la web oficial de Raspberry (*https://www.raspberrypi.org/downloads/raspbian/*).

Una vez dentro de la página web aparecen tres opciones, dos con escritorio y una versión Lite sin escritorio, usaremos la versión con escritorio, aunque la comunicación para la configuración se realice mediante SSH y un entorno amigable no sea necesario.

Para poder instalar el SO sin mayores complicaciones es formatear la tarjeta SD en formato FAT32, este proceso se puede hacer desde el explorador de archivos de Windows.

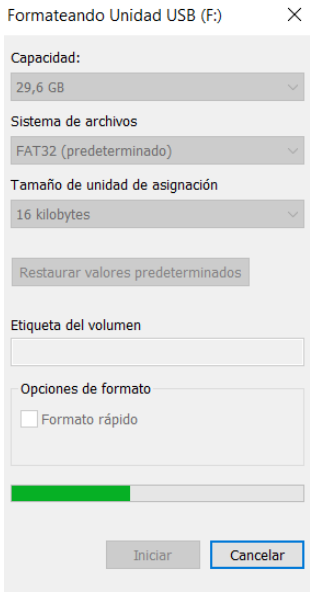

ILUSTRACIÓN 55: FORMATEO DE LA SD

A continuación, se debe copiar el archivo a una tarjeta SD, para ello se pueden usar varios métodos, el más sencillo es utilizar un programa, para "*quemar*" la imagen en la tarjeta desde Windows. La siguiente imagen muestra un programa muy usado para este cometido como es el Ballena Escher.

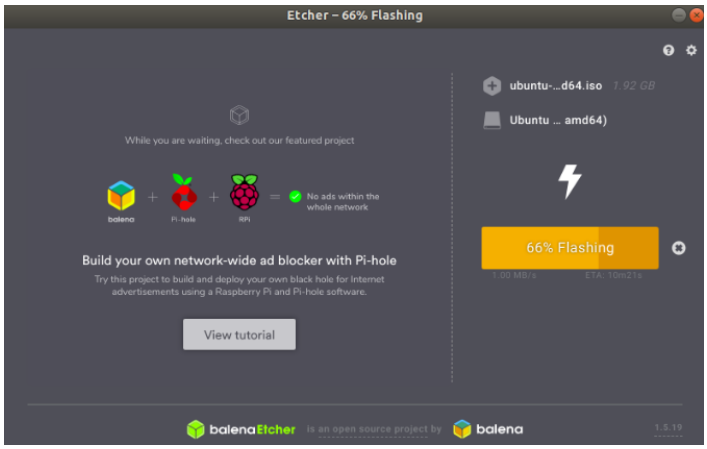

ILUSTRACIÓN 56: BALENA ETCHER

Otra opción es crearla desde la pantalla del terminal del sistema Linux, para ello lo primero es obtener el nombre que se le ha dado a la tarjeta SD. Para ello se introduce la SD en la Raspberry y se ejecuta la orden:

df -h

Este hará que aparezcan las particiones que existen actualmente, una vez obtenido el nombre de la SD, hay que buscar la carpeta donde se encuentra en archivo, a continuación, utilizar el comando

sudo dd bs=4M if="Nombre del archivo" of="nombre de la SD" status=progress conv=fsync

Dependiendo del rendimiento del equipo y del tipo de tarjeta SD el proceso puede tardar más o menos, una vez finalizado, solo tendremos que comprobar que todo está correctamente escrito en la tarjeta, para ellos usamos.

sudo sync

Este comando escribe la información que se encuentra en la memoria cache pero que aún no se encuentra actualizada en el disco, no es necesario utilizarlo, pero si es recomendable ya que si no se usa puede generar pérdidas de información.

4.2.3. Habilitando SSH

El siguiente paso es habilitar el servicio SSH, esto nos permitirá conectarnos a la Raspberry de manera remota, existen dos opciones, la primera y más sencilla consiste en usar el comando *systemctl* de la siguiente manera

sudo sysyemctl enable ssh

sudo systemctl start ssh

El primer comando habilitara el SSH en arranques futuros y el segundo lo inicia en la sesión actual. La segunda forma sería a través de la configuración de la Raspberry, para ello introducimos.

Lo que nos lleva a una pantalla como la mostrada en la siguiente imagen, en ella solo habría que seleccionar *Advance Options*, ir a la opción *SSH* y activarla, dicha opción se encuentra en *Advance Options>Interfacin Options>SSH.*

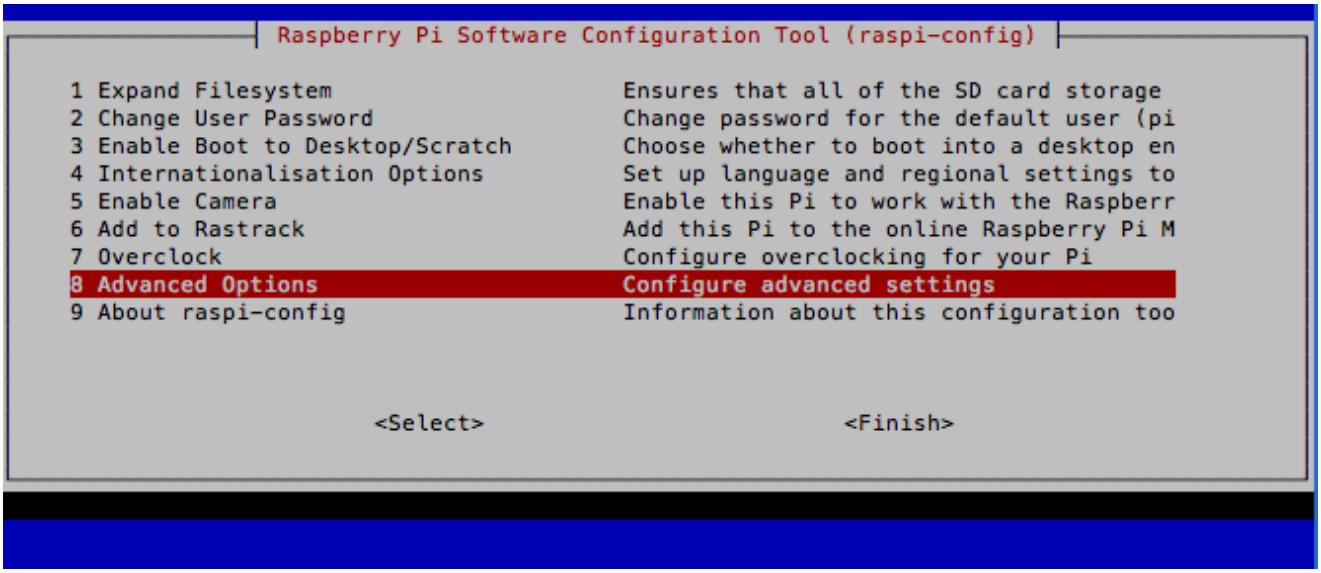

ILUSTRACIÓN 57: RASPI-CONFIG

4.2.4. Creando el punto de acceso

Una vez conseguido acceso de manera remota a la Raspberry, el siguiente paso es crear el punto de acceso, este paso supuso un gran quebradero de cabeza, ya que el modelo B+ tiene un *bug* en el proceso de activación del demonio encargado del *access point* que hace que el servicio no se active. A continuación, se explicará todo el proceso.

El primero paso es conectarnos a la placa desde un programa externo como puede ser el *putty*, para ello se necesita saber la dirección IP y el puerto de escucha, como se trata del protocolo SSH el puerto es el 22 y la IP se puede obtener mediante la instrucción.

Ifconfig

Una vez se ha establecido la conexión, se puede comenzar a configurar el *Access Point*, Ap de ahora en adelante. Esta red que va a producir la Raspberry no va a tener acceso a internet, simplemente se busca que actué como lazo de unión entre el cliente y el servidor, por ello la configuración será menos extensa que si se fuera a realizar un acceso completo. Lo primero es actualizar los paquetes e instalar el programa *Hostapd* (Host Access Point Daemon) que es el encargado de convertir el WiFi en un AP. Indicar que será necesario conectar la Raspberry mediante el RJ45 ya que al cambiar la configuración del interfaz wlan0 perderemos la conexión WiFi, a continuación, se muestra el código a seguir.

sudo apt-get update && sudo apt-get upgrade

sudo apt-get install hostapd

Una vez tenemos instalado el programa, se deben de modificar una serie de ficheros del sistema para que funciones correctamente, para ello se usan los siguientes comandos.

sudo nano /etc/network/interfaces

y se añade al final del fichero.

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet static
address 192.168.5.1
netmask 255.255.255.0
network 192.168.5.0
broadcast 192.168.5.255

Quedando como muestra la foto de la izquierda.

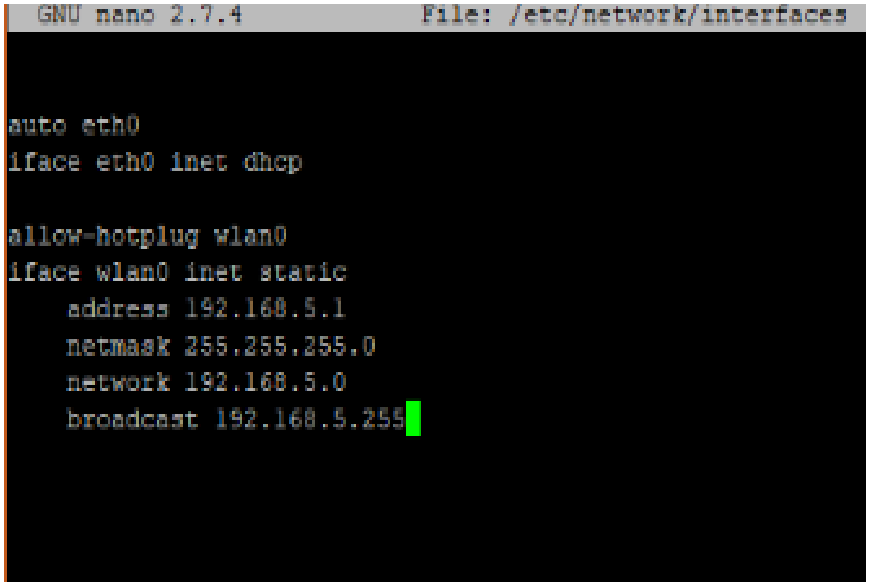

ILUSTRACIÓN 58: CONFIGURACIÓN INTERFACES

El siguiente paso es crear el archivo *Hostapd.conf*, este es el fichero que usara el demonio para generar la red WiFi a la que conectarse, en este fichero se incluyen datos como el nombre de la red, el canal por el que emitirá, la interfaz a usar o la contraseña de la red, para ello se abre el fichero para editarlo usando *sudo nano /etc/hostapd/hostapd.conf y* se añade el siguiente código

interface=wlan0

driver=nl80211

ssid=PDron

hw_mode=g

channel=3

ieee80211n=1

wmm_enabled=1

ht_capab=[HT40][SHORT-GI-20][DSSS_CCK-40]

macaddr_acl=0

auth_algs=1

ignore_broadcast_ssid=0

wpa=2

wpa_key_mgmt=WPA-PSK

wpa_passphrase=Volare

rsn_pairwise=CCMP

Una vez se ha creado el fichero se modifica el script inicial del hostapd para que sepa donde localizar el fichero de configuración.

sudo nano /etc/default/hostapd

y se añade al final del fichero

DAEMON_CONF="/etc/hostapd /hostapd.conf"

Por último se intenta lanzar la red

sudo hostapd /etc/hostapd/hostapd.conf

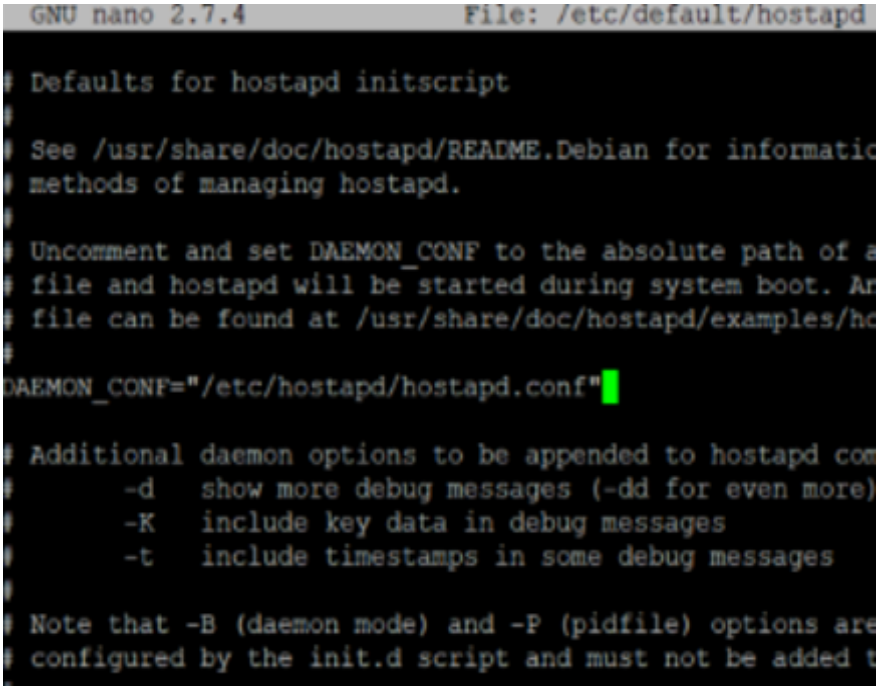

ILUSTRACIÓN 59: CONFIGURACIÓN HOSTAPD.CONF

En este punto se encuentra el primer problema, ya que el demonio no arranca correctamente porque no puede configurar el driver nl80211, para solucionarlo se debe deshabilitar la radio y desbloquear la wlan, reiniciar el servicio.

sudo systemctl stop hostapd

sudo systemctl unmask hostapd

sudo systemctl start hostapd

sudo hostapd /etc/hostapd/hostapd.conf

sudo systemctl restart hostapd

Sudo systemctl status hostapd

Con esto ya estaría instalado y habilitado correctamente el servicio *hostapd*, el siguiente paso es instalar un servidor DCHP, que es el encargado de asignar las direcciones IP dentro de la red, y a continuación modificar su fichero de configuración

sudo apt-get install dnsmasq

Dnsmasq es un programa que proporciona un servidor DNS, un servidor DHCP con soporte para ipV6 y un servidor TFTP, para el proyecto que nos ocupa solo se necesitaría el servidor DCHP, pero *dnsmasq* es tan ligero y consume tan pocos recursos que es perfecto para este desarrollo del dron. Una vez esté instalado el programa, se abre el fichero *dhcp.conf* para modificarlo, para ello.

sudo nano /etc/dhcp/dhcpd.conf

Y se añade al final del fichero

denyinterfaces wlano;

Quedando de manera parecida a como se muestra en la imagen.

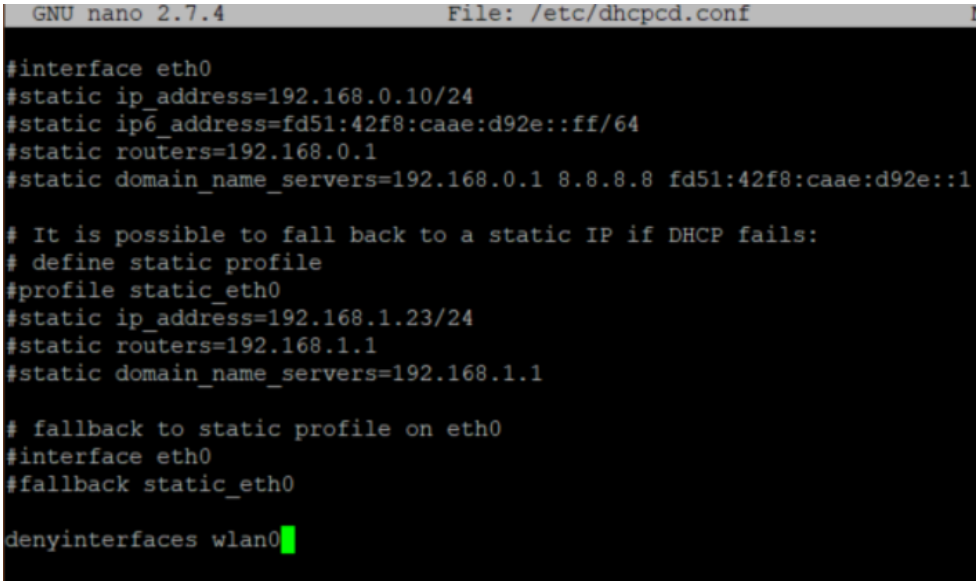

ILUSTRACIÓN 60: CONFIGURACIÓN DHCP.CONF

Además, se instalará el programa *usbmount* que permite montar, de manera automática, cualquier dispositivo de memoria USB que se conecte en la siguiente dirección */mnt/usbx*, donde *x* varía entre 0 y 9 según el número de dispositivos conectados. Permitiendo así grabar video de forma sencilla.

sudo apt-get install usbmount

Para activar la cámara hay que hacerlo a través de *Raspi-Config* y activarlo desde la opción *Enable Camera*, se confirma la activación y se reinicia la Raspberry.

sudo raspi-config

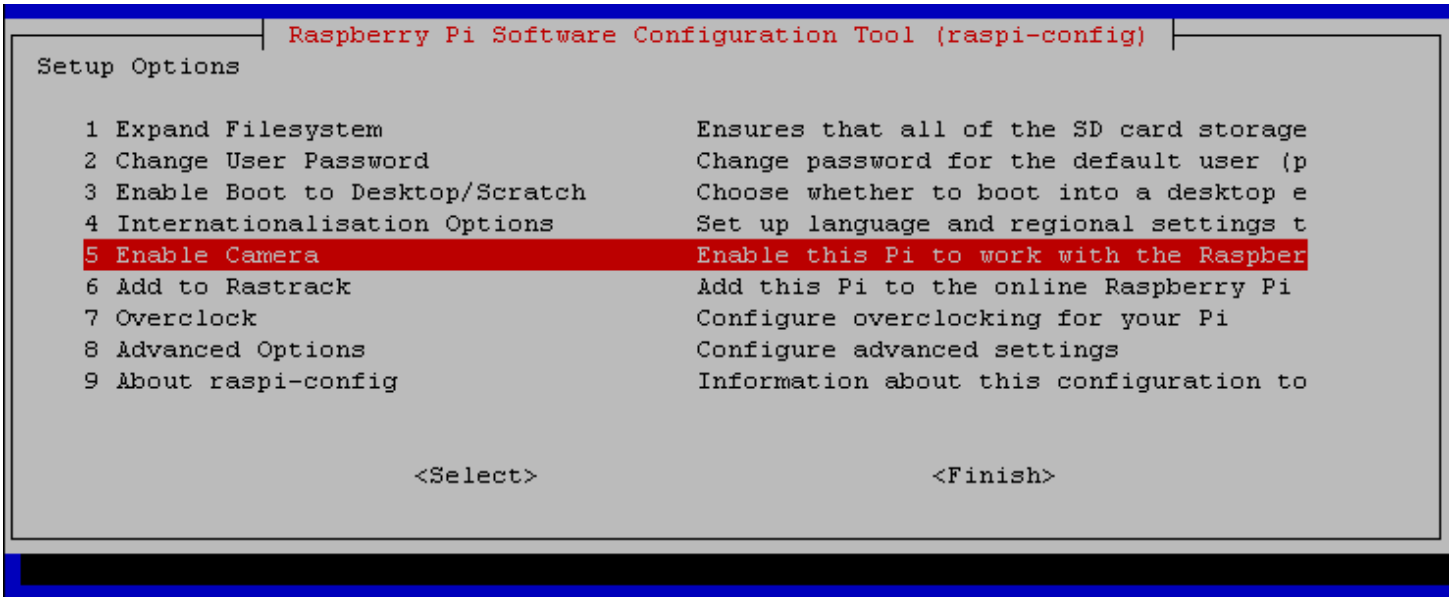

ILUSTRACIÓN 61: ACTIVACIÓN CÁMARA PASO 1

ILUSTRACIÓN 62: ACTIVACIÓN CÁMARA PASO 2

4.3. Configuración de la comunicación

4.3.1. Breve explicación

Para la comunicación del dron, se ha optado por hacerlo mediante una conexión WiFi y un servicio de Web Socket, se han buscado proyectos de código libre en los que apoyarse encontrado un proyecto francés de comunicación aplicable a este proyecto. El código está muy bien depurado y en vista de que no es propio se explicaran las diferentes partes de las que consta tanto el servidor como la app.

4.3.2. Instalando el servidor

Para instalar el servidor que recibirá los datos y se los pasara a la tarjeta controladora mediante USB, tenemos que instalar el entorno de desarrollo en Python y el framework tornado, que es un complemento para Python ideal para conexiones WebSocket en tiempo real. Indicar que el servidor que se va a utilizar está programado en Python y se encuentra alojado en *github*.

sudo apt-get install python-pip build-essential python-dev

sudo pip install tornado

git clone https://github.com/reglisse44/Multiwii-raspberry-drone-server.git multiwiiControll

Se ha decidido utilizar este código libre debido a que es la solución más depurada que se ha encontrado y se prefería usar una app que un control remoto, a continuación, se explicaran cada uno de los procesos del servidor. Comenzamos con el archivo main.py, este fichero incluye la clase *main*, es decir, el inicio del programa que ejecuta el servidor, a continuación se explica el codigo.

```
class Main():
    def start(self, board, camera):
        self.hello = "hello"
        test = "test"
        self.board = board
        self.camera = camera
        self.webServer = server.server(80, self.board, self.camera)
        self.webServer.start(True)

    def stop(self):
        self.webServer.stop()
        self.board.stop()


if __name__ == "__main__":
    global board
    board = multiwii.drone('/dev/ttyUSB0')
    camera = picamera.PiCamera()
    camera.vflip = True
    camera.hflip = True

    start = Main()
    MainThread = threading.Thread(target=start.start, args=(board, camera))
    MainThread.start()
    """MainThread.join()"""

    def signal_handler(signal, frame):
        print('You pressed Ctrl+C!')
        GPIO.cleanup()
        start.stop()
        sys.exit(0)

    signal.signal(signal.SIGINT, signal_handler)
    signal.pause()
```

- El cuerpo de la clase comienza declarando las variables que se van a usar, se usa self para indicar que esos parámetros son los primeros que reciben los métodos cuando se los invoca, es decir, sirve como referencia a un lugar de la memoria, esto se usa para guardar las variables que van a ser usadas durante todo el programa.
- A se define la función stop que pararía el servidor web y la conexión con la placa a través de multiwii.
- El siguiente define las variables antes creadas, en *board* se guarda el parámetro de conexión con la controladora, y en camera los de la cámara, además se inicia la grabación de esta.
- Se crea el hilo que manejara los estados del dron
- Y por último se crean los controladores de las señales, los *handler* para las señales, estos son unas funciones que proporcionan una API de alto nivel para programas que requieren una gran respuesta a actualizaciones de datos o respuesta de datos.

El siguiente fichero para explicar sería *servidor.py*, este fichero es el cuerpo del

servidor propiamente dicho, en él se declaran una serie de funciones necesarias para la comunicación con la aplicación móvil.

```
from tornado.options import define, options
camera = False
board  = False
information = {
    'record': False,
    'main_controller': False,
    'usb': []
}
class server:
    def __init__(self, port, b, cam):
        define("port", default=80, help="run on the given port", type=int)
        self.port = 80
        global camera
        camera = cam
        global board
        board = b
        tornado.options.parse_command_line()
        self.app = Application()
    def start(self, async):
        self.started = True
        self.app.listen(self.port)
        if async:
            try:
                (self.tornadoThread = threading
                .Thread(target=tornado.ioloop.IOLoop.current().start))
                self.tornadoThread.start()
                (self.securityCheckThread = threading
                .Thread(target=self.securityCheck()))
                self.securityCheckThread.start()
                print "async"
                return False
            except Exception, errtxt:
                logging.error("The server failed to start", exc_info=True)
        else:
            try:
                tornado.ioloop.IOLoop.instance().start()
            except Exception, errtxt:
                logging.error("The server failed to start", exc_info=True)
    def stop(self):
        self.started = False
        tornado.ioloop.IOLoop.current().stop()
    def securityCheck(self):
        while self.started:
            if(len(ChatSocketHandler.waiters) == 0):
                board.rcData = [1500, 1500, 1500, 1000]
            mount = commands.getoutput('mount -v')
            lines = mount.split('\n')
            points = map(lambda line: line.split()[2], lines)
            information["usb"] = []
            for point in points:
                if point[:10] == "/media/usb":
                    if point in information["usb"]:
                        pass
                    else:
                        information["usb"].append(point)
            print information
            time.sleep(1)
class Application(tornado.web.Application):
    def __init__(self):
        handlers = [
            (r"/", MainHandler),
            (r"/chatsocket", ChatSocketHandler),
        ]
        settings = dict(
            cookie_secret="__TODO:_GENERATE_YOUR_OWN_RANDOM_VALUE_HERE__",
            template_path=os.path.join(os.path.dirname(__file__), "templates"),
            static_path=os.path.join(os.path.dirname(__file__), "static"),
            xsrf_cookies=True,
        )
        tornado.web.Application.__init__(self, handlers, **settings)

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.render("index.html", messages=ChatSocketHandler.cache)
```

- Como se puede observar, el cometido de servidor.py es abrir la conexión, permanecer a la escucha y crear, para ello usa el firmware Tornado, que como se ha explicado es ideal para este cometido.

- Comienza con la clase server, que comienza definiendo una serie de variables como es la cámara o el puerto.

- El principal cometido de esta clase es crear, mantener y parar la conexión, es decir, define una función *star* que crea una instancia de escucha por el puerto definido anteriormente, por defecto el 80.
- También define una función *Stop* dedicada a finalizar la conexión y parar la escucha.

- Por último, esta clase incluye *securityCheck*, que grosso modo se encarga de la correcta recepción de los comandos y de guardarlos en el USB a modo de caja negra.

- La clase *Application*, crea el entorno de comunicación con la aplicación móvil. Creando uno de los hilos que lleva la aplicación.

```
class ChatSocketHandler(tornado.websocket.WebSocketHandler):
    waiters = dict()
    id_counter = 1
    def get_compression_options(self):
        # Non-None enables compression with default options.
        return {}
    def check_origin(self, origin):
        return True
    def open(self, *args):
        self.id = ChatSocketHandler.id_counter
        ChatSocketHandler.id_counter = ChatSocketHandler.id_counter + 1
        ChatSocketHandler.waiters[self.id] = { "id": self.id, "obj": self}
        #self.write_message('{ "action": "set_id", "data": "'+ str(self.id) +'"}')
    def on_close(self):
        global information
        ChatSocketHandler.waiters.pop(self.id, None)
        information["main_controller"] = False
        self.send_updates()
    @classmethod
    def update_cache(cls, chat):
        cls.cache.append(chat)
        if len(cls.cache) > cls.cache_size:
            cls.cache = cls.cache[-cls.cache_size:]
    def send_updates(cls):
        print "send update"
        global information
        print information
        for key in cls.waiters:
            try:
                information_temp = information
                information_temp["id"] = cls.waiters[key]["id"]
                (cls.waiters[key]["obj"].write_message
                ('{ "action": "get_info", "data": '
                + json.dumps(information_temp) + ' }'))
            except:
                logging.error("Error sending message", exc_info=True)
    def on_message(self, message):
        global information
        message = json.loads(message)
        if message["action"] == "rcData":

            if information["main_controller"] == self.id:
                global board
                board.rcData = [
                    ((message["data"]["right"]["deltaY"]/100)*500+1500),
                    ((message["data"]["right"]["deltaX"]/100)*500+1500),
                    ((message["data"]["left"]["deltaX"]/100)*500+1500),
                    (message["data"]["left"]["deltaY"]/100)*500+1500,
                ]
                print(board.rcData)
```

- Y se llega a la parte más importante del servidor, la clase que controla el envío de mensajes entre el App y el servidor.

- Esta clase básicamente realiza la función de traducir los datos que recibe la Raspberry del controlador principal, es decir, de la App.

- Desde ella se envían los cambios en los joysticks del dron y se controla la grabación de la cámara.

- En este punto se ve como el programa recibe los cambios en los joysticks y traduce la posición en el eje x y en el eje y del mensaje rcData a posiciones que entienda la tarjeta controladora.

```
        if message["action"] == "record":
            print "record"
            global camera
            if information["record"] == False:
                if len(information["usb"]) == 1:
                    if os.path.isdir(information["usb"][0]+'/Video drone') == False:
                        os.mkdir(information["usb"][0]+'/Video drone')
                    (camera.start_recording(information["usb"][0]
                    +"/Video drone/"+time.strftime("%y-%m-%d %H.%M")+".h264"))
                    information["record"] = True
            else:
                camera.stop_recording()
                information["record"] = False
            self.send_updates()
        if message["action"] == "get_info":
            print "get_info"
            self.send_updates()
        if message["action"] == "mainControllerRequest":
            print "mainControllerRequest"
            if information["main_controller"] == False:
                information["main_controller"] = self.id
            self.send_updates()
        if message["action"] == "mainControllerRelease":
            if self.id == information["main_controller"]:
                information["main_controller"] = False
                self.send_updates()

def main():
    tornado.options.parse_command_line()
    app = Application()
    app.listen(options.port)
    tornado.ioloop.IOLoop.current().start()
if __name__ == "__main__":
    main()
```

- El siguiente código controla la grabación de la cámara. Habilitando o deshabilitando la grabación siempre y cuando haya un USB para guardar los datos.

- Por último, tenemos la función central que realiza la función de inicio de la escucha.

El último fichero que hay en la Raspberry es MultiWii.ph, en él se encuentra el código que traduce los datos que recibe la Raspberry desde la aplicación del teléfono móvil para que sean entendibles por la tarjeta controladora. En las siguientes líneas se explicará su funcionamiento.

```
class drone:
    def __init__(self, port):
        self.started = True
        self.ATT     =   0
        self.ALT     =   0
        self.RC      =   1
        self.SET_RC  =   1
        self.MOT     =   0
        self.RAW     =   0
        self.CMD     =   0
        self.UDP     =   0
        self.ASY     =   0
        self.SCK     =   0
        self.SCKSRV  =   0
        self.PRINT   =   0
        self.port = port
        self.ser=serial.Serial()
        self.ser.port=self.port
        self.ser.baudrate=115200
        self.ser.bytesize=serial.EIGHTBITS
        self.ser.parity=serial.PARITY_NONE
        self.ser.stopbits=serial.STOPBITS_ONE
        self.ser.timeout=0
        self.ser.xonxoff=False
        self.ser.rtscts=False
        self.ser.dsrdtr=False
        self.ser.writeTimeout=2
        self.timeMSP=0.02
        try:
            self.ser.open()
        except Exception, e:
            logging.error("Error while open serial port: " + str(e))
            exit()
        self.BASIC="\x24\x4d\x3c\x00"
        self.MSP_IDT=self.BASIC+"\x64\x64"
        self.MSP_STATUS=self.BASIC+"\x65\x65"
        self.MSP_RAW_IMU=self.BASIC+"\x66\x66"
        self.MSP_SERVO=self.BASIC+"\x67\x67"
        self.MSP_MOTOR=self.BASIC+"\x68\x68"
        self.MSP_RC=self.BASIC+"\x69\x69"
        self.MSP_RAW_GPS=self.BASIC+"\x6A\x6A"
        self.MSP_ATTITUDE=self.BASIC+"\x6C\x6C"
        self.MSP_ALTITUDE=self.BASIC+"\x6D\x6D"
        self.MSP_BAT = self.BASIC+"\x6E\x6E"
        self.MSP_COMP_GPS=self.BASIC+"\x71\x71"
        self.MSP_SET_RC=self.BASIC+"\xC8\xC8"


        self.CMD2CODE = {
            # Getter
            'MSP_IDENT':100,
            'MSP_STATUS':101,
            'MSP_RAW_IMU':102,
            'MSP_SERVO':103,
            'MSP_MOTOR':104,
            'MSP_RC':105,
            'MSP_RAW_GPS':106,
            'MSP_COMP_GPS':107,
            'MSP_ATTITUDE':108,
            'MSP_ALTITUDE':109,
            'MSP_ANALOG':110,
            'MSP_RC_TUNING':111,
            'MSP_PID':112,
            'MSP_BOX':113,
            'MSP_MISC':114,
            'MSP_MOTOR_PINS':115,
            'MSP_BOXNAMES':116,
            'MSP_PIDNAMES':117,
            'MSP_WP':118,
            'MSP_BOXIDS':119,

            # Setter
            'MSP_SET_RAW_RC':200,
            'MSP_SET_RAW_GPS':201,
            'MSP_SET_PID':202,
            'MSP_SET_BOX':203,
            'MSP_SET_RC_TUNING':204,
            'MSP_ACC_CALIBRATION':205,
            'MSP_MAG_CALIBRATION':206,
            'MSP_SET_MISC':207,
            'MSP_RESET_CONF':208,
            'MSP_SET_WP':209,
            'MSP_SWITCH_RC_SERIAL':210,
            'MSP_IS_SERIAL':211,
            'MSP_DEBUG':254,
        }
```

- La primera parte de este archivo se encarga de inicializar tanto los procesos de traducción a *serial port*, para ello comienza configurando una serie de valores, en este caso habilita la comunicación por RC y la petición y guardado de los datos dados por el piloto a través del control. Para a continuación configurar los parámetros de comunicación propiamente dichos indicando el puerto y la tasa de baudios, de la que hablaremos más adelante, entre otros.

- Las líneas de código que siguen después del *exit()*, inicializa los parámetros usados por el MSP, *MultiWii Serial Protocol*, este protocolo solo entiende datos en Hexadeciamal.

```
        self.latitude = 0.0
        self.longitude = 0.0
        self.altitude = -0
        self.heading = -0
        self.timestamp = -0
        self.gpsString = -0
        self.numSats = -0
        self.accuracy = -1
        self.beginFlag = 0
        self.roll = 0
        self.pitch = 0
        self.yaw = 0
        self.throttle = 0
        self.angx = 0.0
        self.angy = 0.0
        self.m1 = 0
        self.m2 = 0
        self.m3 = 0
        self.m4 = 0
        self.message = ""
        self.ax = 0
        selfay = 0
        self.az = 0
        self.gx = 0
        self.gy = 0
        self.gz = 0
        self.magx = 0
        self.magy = 0
        self.magz = 0
        self.elapsed = 0
        self.flytime = 0
        self.numOfValues = 0
        self.precision = 3
        self.rcData = [1500, 1500, 1500, 1500]

        self.loopThread = threading.Thread(target=self.loop)
        if self.ser.isOpen():
            print("Wait 5 sec for calibrate Multiwii")
            time.sleep(5)
            self.loopThread.start()

    def stop(self):
        self.started = False
    def stop(self):
        self.started = False

    def littleEndian(self, value):
        length = len(value)
        actual = ""
        for x in range(0, length/2):
            actual += value[length-2-(2*x):length-(2*x)]
            x += 1
        intVal = self.twosComp(actual)
        return intVal

    def twosComp(self, hexValue):
        firstVal = int(hexValue[:1], 16)
        if firstVal >= 8:
            bValue = bin(int(hexValue, 16))
            bValue = bValue[2:]
            newBinary = []
            length = len(bValue)
            index = bValue.rfind('1')
            for x in range(0, index+1)
                if x == index:
                    newBinary.append(bValue[index:])
                elif bValue[x:x+1] == '1':
                    newBinary.append('0')
                elif bValue[x:x+1] == '0':
                    newBinary.append('1')
                x += 1
            newBinary = ''.join(newBinary)
            finalVal = -int(newBinary, 2)
            return finalVal

        else:
            return int(hexValue, 16)
    def sendData(self, data_length, code, data):
        checksum = 0
        total_data = ['$', 'M', '<', data_length, code] + data
        for i in struct.pack('<2B%dh' % len(data), *total_data[3:len(total_data)]):
            checksum = checksum ^ ord(i)

        total_data.append(checksum)
        try:
            b = None
            b = self.ser.write(struct.pack('<3c2B%dhB' % len(data), *total_data))
        except Exception, ex:
            print 'send data error'
            print(ex)
        return b
    def askRC(self):
        self.ser.flushInput()
        self.ser.flushOutput()
        if str(response) == "":
                return
        else:
            msp_hex = response.encode("hex")
            if msp_hex[10:14] == "":
                print("roll unavailable")
            else:
                self.roll = float(self.littleEndian(msp_hex[10:14]))
            if msp_hex[14:18] == "":
                print("pitch unavailable")
            else:
                self.pitch = float(self.littleEndian(msp_hex[14:18]))
            if msp_hex[18:22] == "":
                print("yaw unavailable")
            else:
                self.yaw = float(self.littleEndian(msp_hex[18:22]))
            if msp_hex[22:26] == "":
                print("throttle unavailable")
            else:
                self.throttle = float(self.littleEndian(msp_hex[22:26]))
```

- Para finalizar esta parte, se inicializan las variables globales que se van a utilizar. Y se inicia el loop del hilo de calibración.

- Las dos funciones que vienen a continuación se usan para traducir los valores a enteros, littleEndian se encarga de traducir un hexadecimal a un valor entero utilizando complemento a dos, funciona cambiando byte a byte para convertir el littleEndian en BigEndian para pasárselo a la función TwoComp

- La función twosComp hace la función de complemento a 2 para la función anterior, recibe el dato en valor Bigendian y devuelve el valor decimal del mismo en forma de *integer*.

- La función sendData, como su propio nombre indica, envía una estructura *struck con los datos recibidos*
- La función *AskRc* que simplemente pregunta a la tarjeta por nuevos datos.

```
def setRC(self):
    self.sendData(8, self.CMD2CODE["MSP_SET_RAW_RC"], self.rcData)
    time.sleep(self.timeMSP)
def loop(self):
    print('success')
    try:
        while self.started:
            if self.SET_RC:
                self.setRC()
            if self.ATT:
                askATT()
            if self.ALT:
                askALT()
            if self.RC:
                self.askRC()
            if self.MOT:
                askMOTOR()
            if self.RAW:
                askRAW()
            if self.SCK:
                getUDP()
        self.ser.close()
        file.close()

    except Exception,e1:
        print("Error on main: "+str(e1))
```

- La función *setRC* guarda esos datos en las variables globales
- Y la función *Loop* se encarga de repetir el proceso.

4.3.3. La APP

La aplicación móvil está hecha mediante Apache Cordova, que es un framework de desarrollo móvil de código abierto. Es decir, permite utilizar las tecnologías web como CSS3, JavaScript o HTML5 para desarrollar aplicaciones multiplataforma evitando tener que desarrollar en el lenguaje nativo de cada plataforma. Básicamente se trata de desarrollar una App como si de una página web se tratase.

Esta aplicación está formada por 3 archivos de códigos de javasScript, varios archivos de estilo CSS y de estructura HTML, se explicarán los códigos javascript comenzando por *AppControlles.js*

también tenemos el fichero *config.js*, este es el fichero de configuración de toda la aplicación.

```
var app = angular.module('app', ['ngMaterial', 'ngRoute']);

app.config(function($mdThemingProvider, $mdIconProvider, $routeProvider, $mdGestureProvider){

    $routeProvider.
        when('/home', {
            templateUrl: 'partials/home.html',
        }).
        when('/fly', {
            templateUrl: 'partials/fly.html'
        }).
        otherwise({
            redirectTo: '/home'
        });


    $mdThemingProvider.theme('default')
        .primaryPalette('blue')
        .accentPalette('indigo')
        .backgroundPalette('grey');

    $mdIconProvider
        .icon('rocket', '/svg/rocket20.svg');
});
```

- Este código indica simplemente la estructura interna de la app, y como se mueve el usuario por ella.

```
app.controller('AppController', function($scope, $rootScope){
        $scope.system = {
            deviceState: 'notReady',
            wifi: {
                state: 'NotConnected'
            },
            information: {
                record: false
            }
        }
        $scope.initialize = function() {
            $scope.system.deviceState = 'ready';
            $scope.lastCall = Date.now();
            $scope.initWebsocket();
            // WIFI
            //$scope.system.wifi.state = $cordovaNetwork.-
            //.getNetwork() == "wifi" ? 'connected' : 'connected';
            //$scope.$apply();
        }
        $scope.initWebsocket = function(){
            console.log('conection en cours');
            $rootScope.ws = new WebSocket("ws://192.168.10.1/chatsocket");

            $rootScope.ws.onopen = function(){
                console.log('connected');
                $rootScope.ws.opened = true;
                $rootScope.ws.send(JSON.stringify({ action: "get_info" }));
            }
            $rootScope.ws.onerror = function(error){
                $scope.$apply();
            }
            $rootScope.ws.onmessage = function(message){
                message = JSON.parse(message.data);
                if (message.action = "get_info"){
                    console.log(message);
                    $scope.system.information = message.data;
                    $scope.$apply();
                }
            }

            $rootScope.ws.onclose = function(){
                $rootScope.ws.opened = false;
            }

            $scope.record = function(){
                console.log('--: ' + (Date.now() - $scope.lastCall));
                if (Date.now() - $scope.lastCall < 100){
                    return false;
                }
                $scope.lastCall = Date.now();
                $rootScope.ws.send(JSON.stringify({ action: "record" }));
            }
        }

        $scope.reconect = function(){
            $scope.initWebsocket();
        }

        $scope.initialize();

        document.addEventListener("pause", function(){ navigator.app.exitApp(); }, false);
        document.addEventListener("deviceready", function () {
            //$scope.initialize();
        }, false);
    });
```

- Este código es el encargado de establecer la conexión con el servidor e iniciar el chatWebScoket.

- Se puede observar como abre una conexión WebScoket llamando a la IP 192.168.10.1. la comunicación se hace mediante peticiones de tipo json.

- En este punto el programa falla, ya que la configuración realizada en la RaspBerry añadía otra dirección como puerta de enlace por lo que las opciones existentes serían, cambiar la IP a la que se está realizando la llamada o cambiar la IP del Access Point de la Raspberry. Por facilidad se cambia la IP en el punto de acceso modificando el archivo interfaces.

El siguiente fichero es joystickController.js, este archivo es el encargado de controlar todo lo relacionado con los controles de movimiento del dron.

```
app.controller('JoystickController', function($scope, $rootScope, $interval, $location){
    $rootScope.ws.send(JSON.stringify({ action: 'mainControllerRequest' }));
    $scope.fakeJoystickLeft = new VirtualJoystick({
        mouseSupport    : true,
        limitStickTravel: true,
        stickRadius : 80
    });
    $scope.fakeJoystickLeft.addEventListener('touchStartValidation', function(event){
        console.log($scope);
        var touch = event.changedTouches[0];
        if (touch.pageX < $('#leftJoystickOverlay').width() + $('#leftJoystickOverlay').offset().left
            && touch.pageX > $('#leftJoystickOverlay').offset().left
            && touch.pageY < $('#leftJoystickOverlay').height() + $('#leftJoystickOverlay').offset().top
            && touch.pageY > $('#leftJoystickOverlay').offset().top){
            return true;
        };
        return false;
    });
    $scope.joystickLeft = new VirtualJoystick({
        strokeStyle     : 'rgba(255, 255, 255, 0)',
        mouseSupport    : true,
        stickRadius : 80
    });
    $scope.joystickLeft.addEventListener('touchStartValidation', function(event){
        var touch = event.changedTouches[0];
        if (touch.pageX < $('#leftJoystickOverlay').width() + $('#leftJoystickOverlay').offset().left
            && touch.pageX > $('#leftJoystickOverlay').offset().left
            && touch.pageY < $('#leftJoystickOverlay').height() + $('#leftJoystickOverlay').offset().top
            && touch.pageY > $('#leftJoystickOverlay').offset().top){
            return true;
        };
        return false
    });
    $scope.fakeJoystickRight = new VirtualJoystick({
        strokeStyle     : 'orange',
        mouseSupport    : true,
        limitStickTravel: true,
        stickRadius : 80
    });
    $scope.fakeJoystickRight.addEventListener('touchStartValidation', function(event){
        var touch = event.changedTouches[0];
        if (touch.pageX < $('#rightJoystickOverlay').width() + $('#rightJoystickOverlay').offset().left
            && touch.pageX > $('#rightJoystickOverlay').offset().left
            && touch.pageY < $('#rightJoystickOverlay').height() + $('#rightJoystickOverlay').offset().top
            && touch.pageY > $('#rightJoystickOverlay').offset().top
            ){
            return true;
        };
        return false
    });
```

- El funcionamiento de esta parte del programa es bastante sencillo, primero se encarga de colocar los dos joysticks en la pantalla del dispositivo, tanto el derecho como el izquierdo.

- A continuación, crea unos escuchadores para recibir los datos de movimiento de esos joysticks, esos datos no dejan de ser la posición de los dedos del piloto en la pantalla.

```
    $scope.joystickRight.addEventListener('touchStartValidation', function(event){
        var touch = event.changedTouches[0];
        if (touch.pageX < $('#rightJoystickOverlay').width() + $('#rightJoystickOverlay').offset().
            && touch.pageX > $('#rightJoystickOverlay').offset().left
            && touch.pageY < $('#rightJoystickOverlay').height() + $('#rightJoystickOverlay').offse
            && touch.pageY > $('#rightJoystickOverlay').offset().top
            ){
            return true;
        };
        return false
    });
    var i = 0;
    var joystickSocketUpdater = $interval(function(){
        if (i % 10 == 0){
            console.log($scope.ws);
            console.log($scope.fakeJoystickLeft);
        }
        i++
        $scope.joystick = {
            left: {
                deltaX : ((Math.sqrt(Math.pow($scope.joystickLeft.deltaX(), 2))>Math.sqrt(Math.pow(
                    ?(($scope.joystickLeft.deltaX()<0)?-100:100)
                    :($scope.joystickLeft.deltaX()/$scope.joystickLeft._stickRadius)*100),
                deltaY : ((Math.sqrt(Math.pow($scope.joystickLeft.deltaY(), 2))>Math.sqrt(Math.pow(
                    ?(($scope.joystickLeft.deltaY()<0)?-100:100)
                    :($scope.joystickLeft.deltaY()/$scope.joystickLeft._stickRadius)*100)*(
            },
            right: {
                deltaX : ((Math.sqrt(Math.pow($scope.joystickRight.deltaX(), 2))>Math.sqrt(Math.pow
                    ?(($scope.joystickRight.deltaX()<0)?-100:100)
                    :($scope.joystickRight.deltaX()/$scope.joystickRight._stickRadius)*100),
                deltaY : ((Math.sqrt(Math.pow($scope.joystickRight.deltaY(), 2))>Math.sqrt(Math.pow
                    ?(($scope.joystickRight.deltaY()<0)?-100:100)
                    :($scope.joystickRight.deltaY()/$scope.joystickRight._stickRadius)*100)'
            }
        };
        console.log(JSON.stringify($scope.joystick));
        if($scope.ws.readyState == 1){
            $rootScope.ws.send(JSON.stringify({ action: 'rcData', data: $scope.joystick }));
        }
    }, 10);
    $scope.$on('$destroy', function() {
        $scope.fakeJoystickLeft.destroy();
        $scope.joystickLeft.destroy();
        $scope.fakeJoystickRight.destroy();
        $scope.joystickRight.destroy();
        $interval.cancel(joystickSocketUpdater);

        $rootScope.ws.send(JSON.stringify({ action: 'mainControllerRelease' }));
    });
});
```

- Los datos son traducidos a posiciones en el eje de las X's y en el eje de las Y's para posteriormente ser almacenados. Estos son los datos que usa el servidor para indicarle a la tarjeta controladora las posiciones a las que se desea mover el dron.

En las siguientes imagenes se muestra el entorno grafico de la aplicación, como se puede observar es bastante intuitivo.

ILUSTRACIÓN 63: ICONO DE LA APP

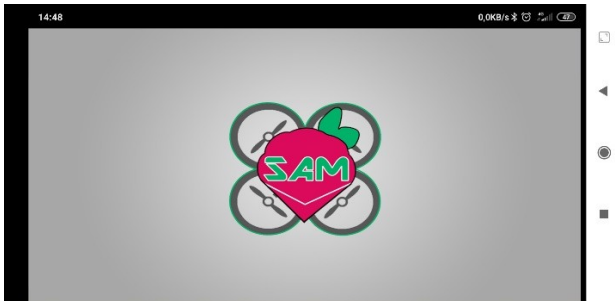

ILUSTRACIÓN 64: PANTALLA DE CARGA

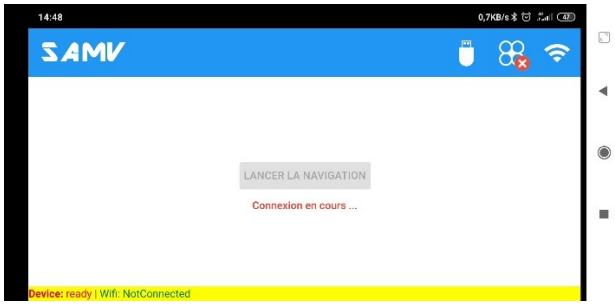

ILUSTRACIÓN 65: PANTALLA DE NAVEGACIÓN

4.4. Configuración de la NAZE32

4.4.1. Breve explicación

La correcta configuración de la Naze32 se antoja primordial para poder realizar un dron con un vuelo estable. La controladora de vuelo usada es una Acro Naze32 rev6 que monta una versión de Multiwii llamado *Baseflight* específica para el STM32. La conexión de la tarjeta con la Raspberry pi se hace a través del micro USB que incorpora.

4.4.2. Pasos previos

Lo primero y fundamental es conectar la Naze32 al ordenador desde el que se va a configurar la controladora, para ello es necesario descargarse los drivers e instalar el *Baseflight Configurator* que es una extensión de Google Chrome que permita configurar el firmware que lleva la placa.

El siguiente paso, después de conectar la tarjeta, es actualizar el firmware, el proceso es bastante sencillo ya que se hace desde la aplicación a través de la opción Firmware Flasher, una vez dentro, simplemente habría que descargar la versión nueva, en *Load firmware*, y *flashearla* sobre la tarjeta, en la siguiente figura se muestran los pasos.

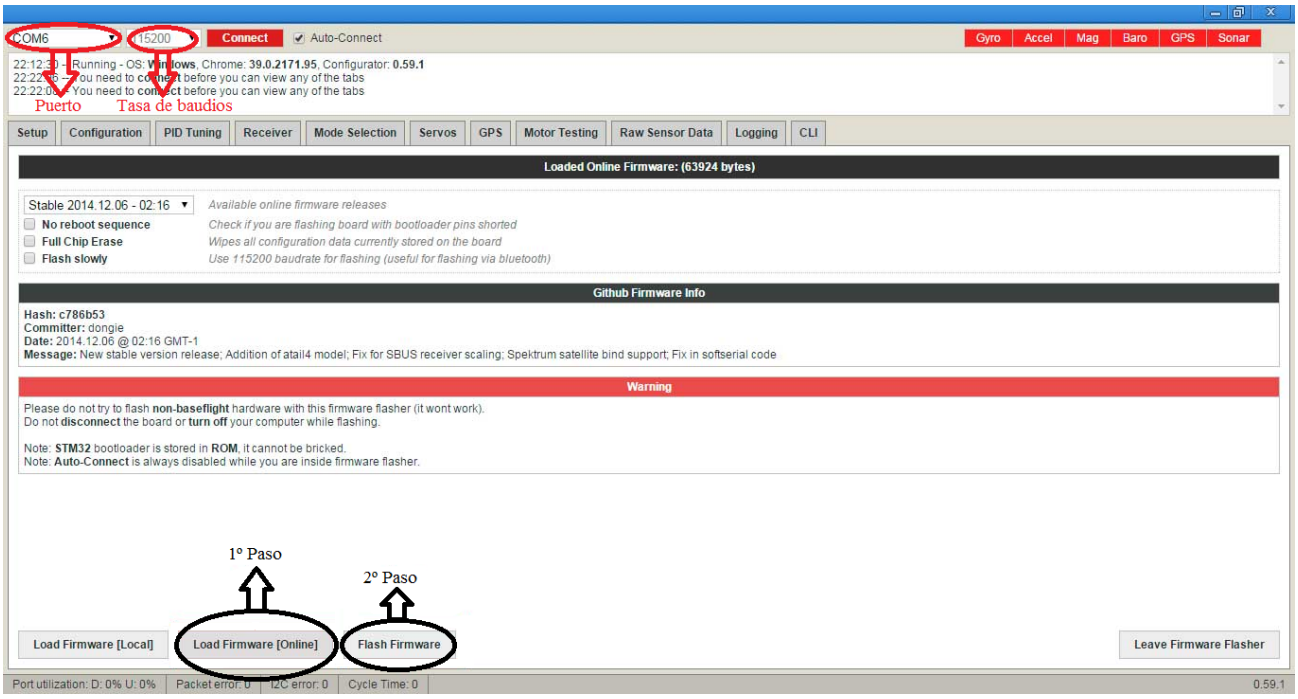

ILUSTRACIÓN 66:ACTUALIZACIÓN DEL FIRMWARE Y CONFIGURACIÓN DE PUERTO Y TASA DE BAUDIOS

Una vez actualizado, es necesario asignar el puerto y la tasa de baudios que es la velocidad a la que se transfiere la información en un canal de comunicación, habitualmente en los puertos serie es de 115200 es quiere decir que dicho canal de comunicación admite como máximo 115200 bits por segundo. Esto se realiza en los desplegables marcados en rojo en la figura anterior. En este punto, si al dar el botón *connect* el programa no da ningún error se puede decir que la configuración previa ha sido un éxito.

4.4.3. Calibrando el dron

Después de realizar los pasos previos, se puede comenzar a calibrar los sensores que trae la tarjeta y por consiguiente calibrar el dron. En la siguiente imagen se muestran los sensores activos en la tarjeta y las opciones de configuración de cada sensor.

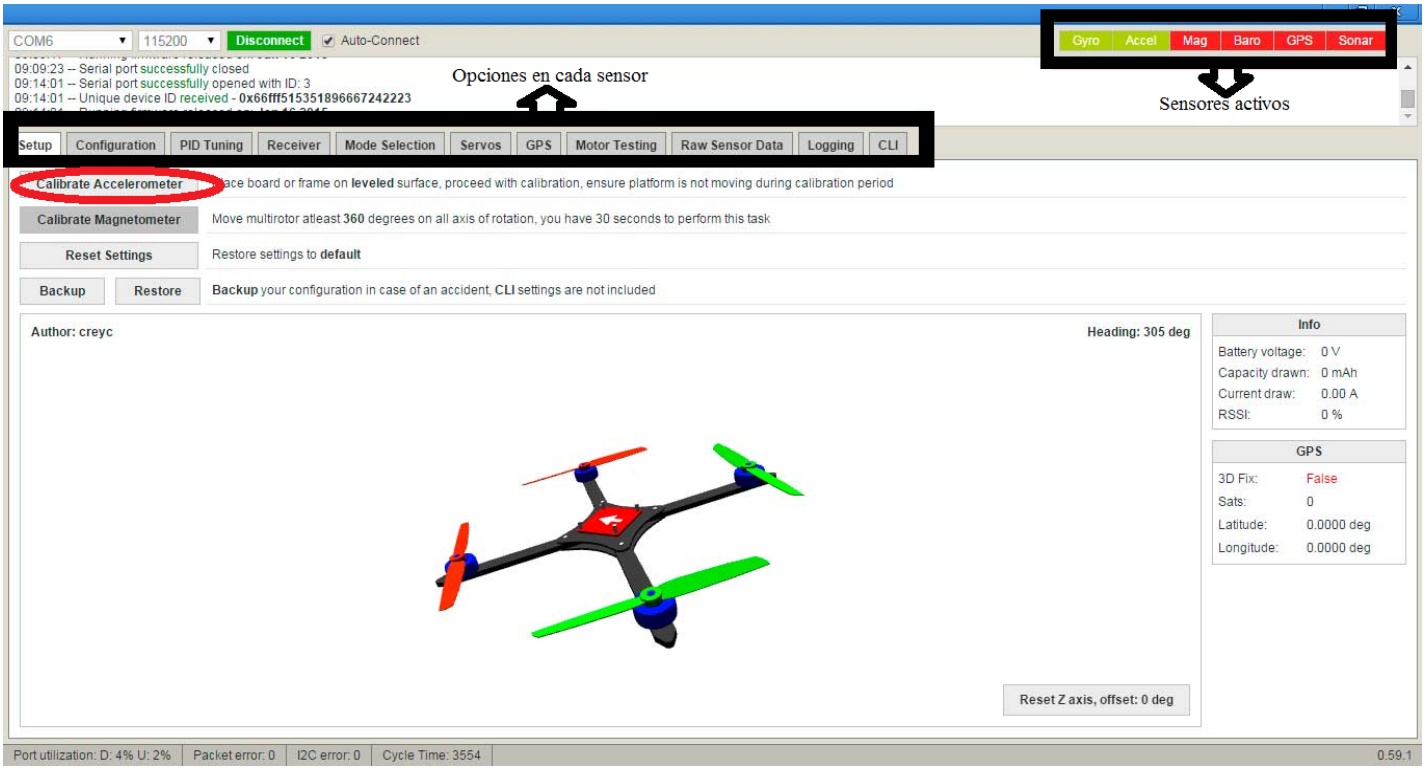

ILUSTRACIÓN 67: CALIBRACIÓN DE LOS SENSORES

Marcada en rojo se encuentra la opción de *Calibrate Accelerometer*, con esta opción pasamos a calibrar el acelerómetro para ello se coloca la placa en posición totalmente horizontal y se pulsa el botón. Este paso se realiza de manera automática. En la pestaña configuración se activa la opción *Don't spin the motors when armed* para que los motores no comienzan a girar cuando se arranque la controladora. De manera opcional se pueden añadir comando mediante CLI, *command line interface*, desde dicho interfaz podemos cambiar los valores del looptime, que es el tiempo que tarda en hacer un bucle de control, o los valores del *failsafe*, que son una serie de rutinas que se activan cuando aparece un determinado evento en la placa como puede ser la pérdida de señal, en este caso se introduce:

set looptime = 3000

failsafe_throttle = 900

failsafe_delay = 15

failsafe_off_delay = 300

Con esto se consigue que el *looptime* este en 333Hz, que la señal de *failsafe* se active a los 1.5 segundos, que a los 3 segundos desde la perdida se apaguen los motores y que el motor desacelere, intentando así que el "aterrizaje de emergencia" sea lo más suave posible.

Por último, configuramos el modo de vuelo a *Angle*, este modo hace que el dron se auto nivele cuando soltamos los controles, de esta manera minimizamos la inclinación del dron procurando un comportamiento más estable. Se pude limitar o aumentar la inclinación a través del CLI introduciendo:

*set max_angle_inclination = Nº de grados*10*

Con estas configuraciones el dron ya está listo para elevarse del suelo y realizar un vuelo totalmente satisfactorio, pero si se quisiera, se podrían configurar muchas más cosas dentro de la tarjeta controladora para afinar, aún más, las características de vuelo del dron. En un futuro, cuando se domine mejor el manejo del dron, se podría cambiar el modo de vuelo a *Horizon*, este modo es una mezcla de *Angle* y *Acro*, este último es el modo acrobacia, un pilotaje totalmente manual. Este modo *Horizon* se comporta como el modo Acro mientras usas los controles, pero si sueltas se auto nivela.

Por ultimo tenemos el valor de los PID de la tarjeta controladora, se recomienda modificarlos para mejorar el vuelo del dron, en este TFM se han dejado los valores por defecto, ya que aunque hacen que el vuelo en ocasiones sea algo menos suave se consigue un vuelo suficientemente controlado.

5. Conclusiones

5.1. Conclusiones

Este proyecto ha sido más complejo de lo que era de esperar, si bien es cierto que se trataba de un TFM muy ambicioso, los problemas a la hora de configurar la Raspberry, como se indicó en el apartado de la configuración, el demonio que habilita el host en la Raspberry no se activaba lo que derivó en una inversión de tiempo considerable investigando posibles causas y soluciones, a ello se le suma el tiempo perdido por culpa de la tarjeta controladora averiada, con el cambio del consiguiente método configuración que conlleva, además del montaje en si del propio dron han hecho que se venga el tiempo encima, por ello ha quedado una parte muy bonita sin realizarse, la App y el servidor de comunicación, bien es cierto que los elementos que se han usado en su lugar han sido estudiados concienzudamente para saber y poder explicar su funcionamiento, pero al autor del TFM le queda la espina de no haber podido desarrollar el suyo propio. Creo que con algo más de tiempo podría haberse realizado un TFM más profundo y más personalizado.

Aun así, el proyecto ha sido muy satisfactorio en lo personal, aplicando conocimientos de distintas ramas y obteniendo un resultado factible en forma de dron. La fase de investigación de los posibles usos de la Raspberry Pi, aprender sobre los drones, sus distintos tipos, como construirlos y como configurarlos y, por último, aprender a manejarlos, así como el proceso manual de ensamblaje, conexión y soldado de este han hecho de este proyecto una experiencia muy edificante.

5.2. Evaluación de los objetivos

En cuanto a los objetivos, recuperando los que el autor había definido al principio de este documento, se realizara una explicación punto por punto de estos:

- *Diseño o selección del hardware*: Este objetivo se considera cumplido, ya que se ha investigado sobre la mejor manera de hacer un dron para principiantes, realizando análisis y comparativas sobre las distintas opciones de piezas existentes en el mercado, aunque es cierto que la opción de la impresión 3D podría haberse barajado, se descartó debido al hecho de no disponer de dicha impresora.

- *Selección de una tarjeta controladora para los motores*: Aunque en un principio la elección de la primera tarjeta fue un fracaso, se entiende que el error no fue debido a dicha selección si no a un fallo de la tarjeta en si, por lo que también entendemos que este objetivo se ha cumplido con la posterior elección de la ACRO NAZE32 la cual ha realizado su función perfectamente.

- *Diseño del software*: En este objetivo es donde el autor cree que no se ha llegado a cumplir del todo, ya que al no poder desarrollar el mismo entiende que no se puede dar un 100% de éxito a este punto. Es cierto que el método seleccionado es *open source* y ha sido estudiado para saber cómo funciona y si necesitaba alguna configuración adicional, por ello se piensa que este punto se ha realizado en un 50%.

- *Construcción del Dron*: Por último, la construcción del dron se ha realizado de manera satisfactoria, toda la circuitería, y las piezas han sido ensambladas de manera correcta.

Resumiendo, el autor pieza que los objetivos marcados se han cumplido en un 87% del proyecto, por lo que considera satisfactorio el desarrollo de este.

5.3. Seguimiento de la planificación

Siendo críticos con el TFM, el autor confiesa que no esta muy contento con el seguimiento de la planificación, en muchas fases del proyecto no ha sido capaz de seguir el ritmo de este llegando incluso a plantearse el posible fracaso al no ser capaz de llegar a tiempo a la entrega final del TFM.

Hay que destacar que la mayoría de los problemas han sido de índole externa, sobre todo por motivos laborales o por retrasos con la recepción de las piezas necesarias. Por otro lado, también existieron problemas técnicos en dos fases del proyecto, la primera fue con la primera tarjeta controladora, la cual estaba en mal estado, y la segunda con la PiCamera, ya que por una mala manipulación se estropeo el conector y hubo que pedir un recambio.

Resumiendo, si bien los problemas propios del diseño del proyecto no han sido muy relevantes, siendo el más problemático la creación del *Host Access Point* en la Raspberry, el autor cree que la planificación de tiempos y el seguimiento de los mismos no ha sido acertado, tal vez se pudiera haber organizado de manera distinta comenzando por la configuración de la Raspberry y la controladora, mientras se esperaban por las piezas en vez de esperar a tener todo montado para comenzar con dicha configuración.

5.4. Líneas futuras

Al autor, también le gustaría mostrar algunas de las posibles evoluciones o pasos futuros que pueden aplicar a proyecto:

- La primera de ellas sería añadirle una batería de mayor potencia junto con algunos sensores, barométricos, de temperatura y velocidad del viento y así poder usarlo de estación meteorológica portable, ya que, gracias a la Raspberry, los datos recogidos se podrían guardar en un equipo gracias al host WiFi que ya incorpora.
- El segundo posible evolutivo sería añadirle un sensor de partículas y usarlo para medir las emisiones de las centrales o la calidad del aire en una zona.
- El tercer posible cambio podría ser la conversión en un dron de carreras, para facilitar el manejo del dron se tendría que usar un control RC por ergonomía y la raspberry podría usarse para enviar el video en directo al móvil haciendo un FPV (*first person view*) lo que facilitaría la conducción del mismo en el circuito.
- El cuarto posible paso futuro sería el desarrollo de un programa capaz de medir la fuerza ascendente de las térmicas de aire, y que estos datos se envíen a un dispositivo móvil para que por ejemplo, un piloto de ala delta o de parapente sea capaz de ir hacia esas zonas y así mantenerse más tiempo en el aire. Aunque este

desarrollo tendría que ser muy a largo plazo ya que la autonomía del dron es el mayor de los problemas en este caso y tiene difícil solución.

5.4. Agradecimientos

Por último, agradecer a todos los que me apoyaron durante el proceso de desarrollo de este TFM, a mi madre por apoyarme y a mi novia por aguantarme y estar siempre ahí cuando la necesite.

A mis compañeros de trabajo por liberarme de carga para poder finalizar a tiempo y en especial a Ruben, que con sus consejos sobre soldadura y circuitería han conseguido que no acabase con las manos quemadas por el soldador.

6. Glosario

- Acro Naze32: Tarjeta controladora de vuelo.
- Anti-jamming: Protocolo anti-interferencias.
- App: Aplicación para teléfonos móviles
- Baseflight: Programa para configurar la tarjeta Acro Naze
- Brushless: Tipo de motor sin escobillas
- CCS3: (Cascading Style Sheets) lenguaje de programación de hojas de estilo en cascada que permite configurar el entorno grafico de manera más sencilla
- CLI: (Command Line Interface) Interfaz para introducir ordenes por la línea de comando
- Comunidad Maker: Comunidad de internet en la que la premisa fundamental es hacer las cosas de manera manual.
- Crius Aio Pro: Tarjeta controladora de vuelo.
- Dhcpd: (Dynamic Host Configuration Protocol) protocolo de red que permite a los clientes de una red IP obtener sus parámetros de configuración automáticamente
- Dron: Vehículo aéreo que vuela sin tripulacion
- ESC: (Electronic Speed Contoller) Es un sistema capaz de definir la velocidad de giro de un motor brushless mediante la generación de pulsos compatibles con este tipo de motores
- Hostapd: Protocolo que activa al demonio que habilita la creación de un host en la Raspberry pi.
- HTML: (HyperText Markup Language) Es un lenguaje de programación que e utiliza para el desarrollo de páginas de Internet
- JavaScript: Lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas
- LiOn: Batería de tipo Ion Litio
- LiPo: Batería de tipo Polimero de Litio
- Megapirate NT: Programa utilizado para configurar la Crius Aio Pro
- Mutiwii: Es un proyecto de código abierto para el control de modelos RC que se basa en una extension de Wii Motion Plus
- NiCd: Bateria de Niquel Cadmio
- NiMH: Bateria de Niquel metal hidruro
- Open Source: Todos aquellos programas informáticos que disponen el acceso gratuito a su código de programación facilitando por parte de otros programadores ajenos la modificación del mismo
- PCB: (Printed Board Circuit) Un circuito impreso es una placa que cuenta con pistas conductoras.
- Pitch: Angulo de inclinación, rotación respecto al eje ala-ala
- Raspberry Pi: Placa computadora de bajo coste con todas las ventajas de un ordenador.
- RC: (Radio Control) Permite el control de objetos a distancia de manera inalámbrica.
- Roll: Angulo de escora, rotación respecto al eje morro-cola del avión.
- SBC: (Singles Board Computer) Placas que contienen todos los componentes de un PC.
- SoC: (Sistem on Chip) Chip que integra todos o casi todos los módulos que componen un computador
- SSH: (Secure Shell) Protocolo de administración remota
- UAV: (Unmanned Aerial Vehicle) Vehículo aéreo no tripulado
- WebSocket: Tecnología que proporciona comunicación bidireccional full-duplex sobre un único socket TCP.
- Yaw: Angulo de deriva, rotación intrínseca respecto al eje vertical perpendicular al avión.

7. Bibliografía

- QUEADRACOPTER, Sree Harsha, agosto de 2014, http://socialledge.com/sjsu/index.php/S14:_Quadcopter#Acknowledgement, Proyecto publicado por Sree Harsha en 2014 sobre la realización de una de los UAV con una mayor capacidad de vuelo, el cuadricóptero.

- HOW TO BUILD A DRONE: CONSTRUCT YOUR DRONE FROM SCRATCH, Jack Brown, N/A, https://www.mydronelab.com/blog/how-to-build-a-drone.html, Primera introducción a la creación de un dron desde cero, este artículo de la web mydronelab.com muestra una visión global de todo el proceso.

- INSTRUCTABLES.COM, N/A, N/A, https://www.instructables.com , Su sección de tecnología sobre drones es una base de conocimiento con todo lo relacionado con la historia y los tipos de drones, explicaciones destalladas de los tipos de drones del número de hélices y manuales de pilotaje y construcción.

- COMO CONSTRUIR SU PROPIO DRON. Alex Elliott, 13/07/2019, Macombo, Este libro es una completa guía para poder seleccionar los mejores componentes y configuraciones de entre una amplia gama de requisitos y presupuestos

- HANDOBOOK OF UNMANNED AERIAL VEHICLES, Kimon P. Valavanis / George j. Vachtsevanos, N/A, Springer Reference (https://www.springer.com/), Guía maestra sobre el mundo dron, consejos, fundamentos y legislación.

- RASPBERRYPI.ORG, Raspberry fundation, N/A, https://www.raspberrypi.org/forums/, La mayor base de conocimiento sobre Raspberry, infinidad de información sobre cualquier problema que se pueda dar a la hora de usar una Raspberry.

- GITHUB.COM, N/A,N/A, https://github.com/, Pagina que recoge gran cantidad de recursos *Open Source*, muy útil para encontrar programas o códigos.

- PLAN ESTRATEGICO PARA EL DESARROLLO DEL SECTOR CIVIL DE LOS DRONES EN ESPAÑA 2018-2021, Ministerio de Fomento, 2017, https://www.fomento.gob.es/NR/rdonlyres/7B974E30-2BD2-46E5-BEE5-26E00851A455/148411/PlanEstrategicoDrones.pdf, Plan estratégico para el desarrollo de los drones en España, el futuro uso de los drones y los desarrollos que se están llevando a cabo, además de legislación vigente y futura.

- SOLDAR CON ESTAÑO (TRUCOS Y CONSEJOS) YOUTUBE.COM, N/A,N/A, https://www.youtube.com/watch?v=y_Q2kFfXxUY, Video con trucos y consejos para soldar con estaño, muy necesario para la parte de circuitería.

8. Anexos

8.1. main.py

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
from __future__ import unicode_literals

import logging
import os.path
import uuid
import signal
import sys
import threading
import json
import encodings
import time
import serial
import RPi.GPIO as GPIO
import picamera
import multiwii
import server
class Main():
        def start(self, board, camera):
                self.hello = "hello"
                test = "test"
                self.board = board
                self.camera = camera
                self.webServer     =     server.server(80,     self.board,
self.camera)
                self.webServer.start(True)

        def stop(self):
                self.webServer.stop()
                self.board.stop()
if __name__ == "__main__":
        global board
        board = multiwii.drone('/dev/ttyUSB0')
        camera = picamera.PiCamera()
        camera.vflip = True
        camera.hflip = True
        start = Main()
        MainThread  =  threading.Thread(target=start.start,  args=(board,
camera))
        MainThread.start()
        """MainThread.join()"""

        def signal_handler(signal, frame):
                print('You pressed Ctrl+C!')
                GPIO.cleanup()
                start.stop()
                sys.exit(0)

        signal.signal(signal.SIGINT, signal_handler)
        signal.pause()
```

8.2. multiwii.py

```
#/!usr/bin/env python

import time
import logging
import serial
import threading
import struct           # for decoding data strings


class drone:

        def __init__(self, port):

                self.started = True

                self.ATT        =       0       # Ask and save the attitude
of the multicopter
                self.ALT        =       0       # Ask and save the altitude
of the multicopter
                self.RC         =       1       # Ask and save the pilot
commands of the multicopter
                self.SET_RC     =       1       # Set rc command
                self.MOT        =       0       # Ask and save the PWM of
the motors that the MW is writing to the multicopter
                self.RAW        =       0       # Ask and save the raw imu
data of the multicopter
                self.CMD        =       0       # Send commands to the MW
to control it
                self.UDP        =       0       # Save or use UDP data (to
be adjusted)
                self.ASY        =       0       # Use async communicacion
                self.SCK        =       0       #   Use   regular   socket
communication
                self.SCKSRV     =       0       #      Use      socketserver
communication
                self.PRINT      =       0       # Print data to terminal,
useful for debugging

                self.port = port

                self.ser=serial.Serial()
                self.ser.port=self.port
                self.ser.baudrate=115200
                self.ser.bytesize=serial.EIGHTBITS
                self.ser.parity=serial.PARITY_NONE
                self.ser.stopbits=serial.STOPBITS_ONE
                self.ser.timeout=0
                self.ser.xonxoff=False
                self.ser.rtscts=False
                self.ser.dsrdtr=False
                self.ser.writeTimeout=2

                self.timeMSP=0.02

                try:
                        self.ser.open()
```

```
                except Exception, e:
                        logging.error("Error while open serial port: " +
str(e))
                        exit()

                self.BASIC="\x24\x4d\x3c\x00" #MSG   Send   Header   (to
MultiWii)
                self.MSP_IDT=self.BASIC+"\x64\x64"    #MSG ID: 100
                self.MSP_STATUS=self.BASIC+"\x65\x65" #MSG ID: 101
                self.MSP_RAW_IMU=self.BASIC+"\x66\x66"#MSG ID: 102
                self.MSP_SERVO=self.BASIC+"\x67\x67"  #MSG ID: 103
                self.MSP_MOTOR=self.BASIC+"\x68\x68"  #MSG ID: 104
                self.MSP_RC=self.BASIC+"\x69\x69"             #MSG ID: 105
                self.MSP_RAW_GPS=self.BASIC+"\x6A\x6A"#MSG ID: 106
                self.MSP_ATTITUDE=self.BASIC+"\x6C\x6C"       #MSG ID: 108
                self.MSP_ALTITUDE=self.BASIC+"\x6D\x6D"       #MSG ID: 109
                self.MSP_BAT = self.BASIC+"\x6E\x6E"  #MSG ID: 110
                self.MSP_COMP_GPS=self.BASIC+"\x71\x71"       #MSG ID: 111
                self.MSP_SET_RC=self.BASIC+"\xC8\xC8"         #MSG ID: 200

                self.CMD2CODE = {
                        # Getter
                        'MSP_IDENT':100,
                        'MSP_STATUS':101,
                        'MSP_RAW_IMU':102,
                        'MSP_SERVO':103,
                        'MSP_MOTOR':104,
                        'MSP_RC':105,
                        'MSP_RAW_GPS':106,
                        'MSP_COMP_GPS':107,
                        'MSP_ATTITUDE':108,
                        'MSP_ALTITUDE':109,
                        'MSP_ANALOG':110,
                        'MSP_RC_TUNING':111,
                        'MSP_PID':112,
                        'MSP_BOX':113,
                        'MSP_MISC':114,
                        'MSP_MOTOR_PINS':115,
                        'MSP_BOXNAMES':116,
                        'MSP_PIDNAMES':117,
                        'MSP_WP':118,
                        'MSP_BOXIDS':119,

                        # Setter
                        'MSP_SET_RAW_RC':200,
                        'MSP_SET_RAW_GPS':201,
                        'MSP_SET_PID':202,
                        'MSP_SET_BOX':203,
                        'MSP_SET_RC_TUNING':204,
                        'MSP_ACC_CALIBRATION':205,
                        'MSP_MAG_CALIBRATION':206,
                        'MSP_SET_MISC':207,
                        'MSP_RESET_CONF':208,
                        'MSP_SET_WP':209,
                        'MSP_SWITCH_RC_SERIAL':210,
                        'MSP_IS_SERIAL':211,
                        'MSP_DEBUG':254,
```

```
                }
                self.latitude = 0.0
                self.longitude = 0.0
                self.altitude = -0
                self.heading = -0
                self.timestamp = -0
                self.gpsString = -0
                self.numSats = -0
                self.accuracy = -1
                self.beginFlag = 0
                self.roll = 0
                self.pitch = 0
                self.yaw = 0
                self.throttle = 0
                self.angx = 0.0
                self.angy = 0.0
                self.m1 = 0
                self.m2 = 0
                self.m3 = 0
                self.m4 = 0
                self.message = ""
                self.ax = 0
                selfay = 0
                self.az = 0
                self.gx = 0
                self.gy = 0
                self.gz = 0
                self.magx = 0
                self.magy = 0
                self.magz = 0
                self.elapsed = 0
                self.flytime = 0
                self.numOfValues = 0
                self.precision = 3
                self.rcData = [1500, 1500, 1500, 1500] #order -> roll,
pitch, yaw, throttle

                self.loopThread = threading.Thread(target=self.loop)
                if self.ser.isOpen():
                        print("Wait 5 sec for calibrate Multiwii")
                        time.sleep(5)
                        self.loopThread.start()

        def stop(self):
                self.started = False
        def littleEndian(self, value):
                length = len(value)
                actual = ""
                for x in range(0, length/2):
                        actual += value[length-2-(2*x):length-(2*x)]
                        x += 1
                intVal = self.twosComp(actual)
                return intVal
def twosComp(self, hexValue):
                firstVal = int(hexValue[:1], 16)
                if firstVal >= 8:
                        bValue = bin(int(hexValue, 16))
                        bValue = bValue[2:]
```

```
                        newBinary = []
                        length = len(bValue)
                        index = bValue.rfind('1')
                        for x in range(0, index+1):
                                if x == index:
                                        newBinary.append(bValue[index:])
                                elif bValue[x:x+1] == '1':
                                        newBinary.append('0')
                                elif bValue[x:x+1] == '0':
                                        newBinary.append('1')
                                x += 1
                        newBinary = ''.join(newBinary)
                        finalVal = -int(newBinary, 2)
                        return finalVal

                else:
                        return int(hexValue, 16)



        def sendData(self, data_length, code, data):
                checksum = 0
                total_data = ['$', 'M', '<', data_length, code] + data
                for    i    in    struct.pack('<2B%dh'    %    len(data),
*total_data[3:len(total_data)]):
                        checksum = checksum ^ ord(i)

                total_data.append(checksum)

                try:
                        b = None
                        b    =    self.ser.write(struct.pack('<3c2B%dhB'    %
len(data), *total_data))
                except Exception, ex:
                        print 'send data error'
                        print(ex)
                return b



        def askRC(self):
                self.ser.flushInput()  # cleans out the serial port
                self.ser.flushOutput()
                self.ser.write(self.MSP_RC)    # gets RC information
                time.sleep(self.timeMSP)
                response = self.ser.readline()
                if str(response) == "":
                        #print(msp_hex)
                        #print("Header: " + msp_hex[:6])
                        #payload = int(msp_hex[6:8])
                        #print("Payload: " + msp_hex[6:8])
                        #print("Code: " + msp_hex[8:10])
                        #print("RC data unavailable")
                        return
                else:
                        msp_hex = response.encode("hex")
```

```
                        if msp_hex[10:14] == "":
                                print("roll unavailable")

                        else:
                                self.roll                                       =
float(self.littleEndian(msp_hex[10:14]))

                        if msp_hex[14:18] == "":
                                print("pitch unavailable")

                        else:
                                self.pitch                                      =
float(self.littleEndian(msp_hex[14:18]))

                        if msp_hex[18:22] == "":
                                print("yaw unavailable")

                        else:
                                self.yaw                                        =
float(self.littleEndian(msp_hex[18:22]))

                        if msp_hex[22:26] == "":
                                print("throttle unavailable")

                        else:
                                self.throttle                                   =
float(self.littleEndian(msp_hex[22:26]))

                        #print(str(self.roll) + " " + str(self.pitch) + "
" + str(self.yaw) + " " + str(self.throttle))

        def setRC(self):
                self.sendData(8,          self.CMD2CODE["MSP_SET_RAW_RC"],
self.rcData)
                time.sleep(self.timeMSP)
                #print self.rcData









        def loop(self):
                print('success')
                try:

                        while self.started:

                                if self.SET_RC:
                                        self.setRC()

                                if self.ATT:
                                        askATT()
```

```
                                if self.ALT:
                                        askALT()

                                if self.RC:
                                        self.askRC()

                                if self.MOT:
                                        askMOTOR()

                                if self.RAW:
                                        askRAW()

                                if self.SCK:
                                        getUDP()


                        self.ser.close()
                        file.close()

                except Exception,e1:   # Catches any errors in the serial
communication
                        print("Error on main: "+str(e1))
```

8.3. server.py

```
import time
import commands
import logging
import tornado.escape
import tornado.ioloop
import tornado.options
import tornado.web
import tornado.websocket
import os.path
import uuid
import threading
import json

from tornado.options import define, options

camera = False
board  = False
information = {
        'record': False,
        'main_controller': False,
        'usb': []
}

class server:
        def __init__(self, port, b, cam):
                define("port", default=80, help="run on the given port",
type=int)
                self.port = 80
                global camera
                camera = cam
                global board
                board = b
                tornado.options.parse_command_line()
                self.app = Application()
                #self.board = board
                #self.port = port
                #global camera
                #camera = cam
                #tornado.options.parse_command_line()
                #self.app = Application()

        def start(self, async):
                self.started = True
                self.app.listen(self.port)
                if async:
                        try:
                                self.tornadoThread                      =
threading.Thread(target=tornado.ioloop.IOLoop.current().start)
                                self.tornadoThread.start()

                                self.securityCheckThread                =
threading.Thread(target=self.securityCheck())
                                self.securityCheckThread.start()
                                print "async"
                                return False
                        except Exception, errtxt:
```

```
                                logging.error("The    server    failed    to
start", exc_info=True)
                else:
                        try:
                                tornado.ioloop.IOLoop.instance().start()

                        except Exception, errtxt:
                                logging.error("The    server    failed    to
start", exc_info=True)

        def stop(self):
                self.started = False
                tornado.ioloop.IOLoop.current().stop()

        def securityCheck(self):
                while self.started:
                        if(len(ChatSocketHandler.waiters) == 0):
                                board.rcData = [1500, 1500, 1500, 1000]

                        mount = commands.getoutput('mount -v')
                        lines = mount.split('\n')
                        points = map(lambda line: line.split()[2], lines)
                        information["usb"] = []
                        for point in points:
                                if point[:10] == "/media/usb":
                                        if point in information["usb"]:
                                                pass
                                        else:

        information["usb"].append(point)

                        print information
                        time.sleep(1)


class Application(tornado.web.Application):
        def __init__(self):
                handlers = [
                        (r"/", MainHandler),
                        (r"/chatsocket", ChatSocketHandler),
                ]
                settings = dict(

        cookie_secret="__TODO:_GENERATE_YOUR_OWN_RANDOM_VALUE_HERE__",

        template_path=os.path.join(os.path.dirname(__file__),
"templates"),

        static_path=os.path.join(os.path.dirname(__file__), "static"),
                        xsrf_cookies=True,
                )
                tornado.web.Application.__init__(self,           handlers,
**settings)


class MainHandler(tornado.web.RequestHandler):
        def get(self):
```

```
                self.render("index.html",
messages=ChatSocketHandler.cache)

class ChatSocketHandler(tornado.websocket.WebSocketHandler):
        waiters = dict()
        id_counter = 1

        def get_compression_options(self):
                # Non-None enables compression with default options.
                return {}
        def check_origin(self, origin):
                return True
        def open(self, *args):
                self.id = ChatSocketHandler.id_counter
                ChatSocketHandler.id_counter                                =
ChatSocketHandler.id_counter + 1
                ChatSocketHandler.waiters[self.id]  =  {  "id":  self.id,
"obj": self}
                #self.write_message('{  "action":  "set_id",  "data":  "'+
str(self.id) +'"}')
        def on_close(self):
                global information
                ChatSocketHandler.waiters.pop(self.id, None)
                information["main_controller"] = False
                self.send_updates()
        @classmethod
        def update_cache(cls, chat):
                cls.cache.append(chat)
                if len(cls.cache) > cls.cache_size:
                        cls.cache = cls.cache[-cls.cache_size:]
        def send_updates(cls):
                print "send update"
                global information
                print information
                for key in cls.waiters:
                        try:
                                information_temp = information
                                information_temp["id"]  =  cls.waiters[key]
["id"]
                                cls.waiters[key]
["obj"].write_message('{    "action":    "get_info",    "data":    '    +
json.dumps(information_temp) + ' }')
                        except:
                                logging.error("Error   sending   message",
exc_info=True)

        def on_message(self, message):
                global information
                message = json.loads(message)
                if message["action"] == "rcData":
                        if information["main_controller"] == self.id:
                                global board
                                board.rcData = [
                                        ((message["data"]["right"]
["deltaY"]/100)*500+1500),
                                        ((message["data"]["right"]
["deltaX"]/100)*500+1500),
```

```
                                        ((message["data"]["left"]
["deltaX"]/100)*500+1500),
                                        (message["data"]["left"]
["deltaY"]/100)*500+1500,
                                ]
                                print(board.rcData)

                if message["action"] == "record":
                        print "record"
                        global camera
                        if information["record"] == False:
                                if len(information["usb"]) == 1:
                                        if os.path.isdir(information["usb"]
[0]+'/Video drone') == False:
                                                os.mkdir(information["usb"]
[0]+'/Video drone')

        camera.start_recording(information["usb"][0]+"/Video
drone/"+time.strftime("%y-%m-%d %H.%M")+".h264")
                                        information["record"] = True
                        else:
                                camera.stop_recording()
                                information["record"] = False
        self.send_updates()
                if message["action"] == "get_info":
                        print "get_info"
                        self.send_updates()

                if message["action"] == "mainControllerRequest":
                        print "mainControllerRequest"
                        if information["main_controller"] == False:
                                information["main_controller"] = self.id
                        self.send_updates()

                if message["action"] == "mainControllerRelease":
                        if self.id == information["main_controller"]:
                                information["main_controller"] = False
                                self.send_updates()
def main():
        tornado.options.parse_command_line()
        app = Application()
        app.listen(options.port)
        tornado.ioloop.IOLoop.current().start()


if __name__ == "__main__":
        main()
```

8.4. AppController.js

```
app.controller('AppController', function($scope, $rootScope){
        $scope.system = {
                deviceState: 'notReady',
                wifi: {
                        state: 'NotConnected'
                },
                information: {
                        record: false
```

```
                }
        }



        $scope.initialize = function() {
                $scope.system.deviceState = 'ready';
                $scope.lastCall = Date.now();

                $scope.initWebsocket();


        }

        $scope.initWebsocket = function(){
                console.log('conection en cours');
                $rootScope.ws                    =                    new
WebSocket("ws://192.168.10.1/chatsocket");



                $rootScope.ws.onopen = function(){
                        console.log('connected');
                        $rootScope.ws.opened = true;
                        $rootScope.ws.send(JSON.stringify({         action:
"get_info" }));
                }

                $rootScope.ws.onerror = function(error){
                        $scope.$apply();
                }

                $rootScope.ws.onmessage = function(message){
                        message = JSON.parse(message.data);
                        if (message.action = "get_info"){
                                console.log(message);
                                $scope.system.information = message.data;
                                $scope.$apply();
                        }
                }




                $rootScope.ws.onclose = function(){
                        $rootScope.ws.opened = false;
                }

                $scope.record = function(){
                        console.log('--:      '      +      (Date.now()      -
$scope.lastCall));
                        if (Date.now() - $scope.lastCall < 100){
                                return false;
                        }
                        $scope.lastCall = Date.now();
                        $rootScope.ws.send(JSON.stringify({         action:
"record" }));
```

```
                }
        }

        $scope.reconect = function(){
                $scope.initWebsocket();
        }

        $scope.initialize();

        document.addEventListener("pause",                    function()
{ navigator.app.exitApp(); }, false);

        document.addEventListener("deviceready", function () {

        }, false);
});
```

8.5. JoystickController.js

```
app.controller('JoystickController', function($scope, $rootScope, $interval, $location){

	$rootScope.ws.send(JSON.stringify({ action: 'mainControllerRequest' }));

	$scope.fakeJoystickLeft = new VirtualJoystick({
		mouseSupport    : true,
		limitStickTravel: true,
		stickRadius     : 80
	});

	$scope.fakeJoystickLeft.addEventListener('touchStartValidation', function(event){
		console.log($scope);
		var touch = event.changedTouches[0];
		if (touch.pageX < $('#leftJoystickOverlay').width() + $('#leftJoystickOverlay').offset().left
			&& touch.pageX > $('#leftJoystickOverlay').offset().left
			&& touch.pageY < $('#leftJoystickOverlay').height() + $('#leftJoystickOverlay').offset().top
			&& touch.pageY > $('#leftJoystickOverlay').offset().top){
			return true;
		};
		return false;
	});

	$scope.joystickLeft = new VirtualJoystick({
		strokeStyle : 'rgba(255, 255, 255, 0)',
		mouseSupport    : true,
		stickRadius     : 80
	});

	$scope.joystickLeft.addEventListener('touchStartValidation', function(event){
		var touch = event.changedTouches[0];
		if (touch.pageX < $('#leftJoystickOverlay').width() + $('#leftJoystickOverlay').offset().left
			&& touch.pageX > $('#leftJoystickOverlay').offset().left
			&& touch.pageY < $('#leftJoystickOverlay').height() + $('#leftJoystickOverlay').offset().top
			&& touch.pageY > $('#leftJoystickOverlay').offset().top){
			return true;
		};
		return false
	});

	$scope.fakeJoystickRight = new VirtualJoystick({
		strokeStyle : 'orange',
		mouseSupport    : true,
```

```
                limitStickTravel: true,
                stickRadius     : 80
        });

        $scope.fakeJoystickRight.addEventListener('touchStartValidation',
function(event){
                var touch = event.changedTouches[0];
                if (touch.pageX < $('#rightJoystickOverlay').width() + $
('#rightJoystickOverlay').offset().left
                        &&          touch.pageX          >          $
('#rightJoystickOverlay').offset().left
                        &&          touch.pageY          <          $
('#rightJoystickOverlay').height()                  +                  $
('#rightJoystickOverlay').offset().top
                        &&          touch.pageY          >          $
('#rightJoystickOverlay').offset().top
                        ){
                        return true;
                };
                return false
        });
        $scope.joystickRight = new VirtualJoystick({
                strokeStyle            : 'rgba(255, 255, 255, 0)',
                mouseSupport    : true,
                stickRadius     : 80
        });

        $scope.joystickRight.addEventListener('touchStartValidation',
function(event){
                var touch = event.changedTouches[0];
                if (touch.pageX < $('#rightJoystickOverlay').width() + $
('#rightJoystickOverlay').offset().left
                        &&          touch.pageX          >          $
('#rightJoystickOverlay').offset().left
                        &&          touch.pageY          <          $
('#rightJoystickOverlay').height()                  +                  $
('#rightJoystickOverlay').offset().top
                        &&          touch.pageY          >          $
('#rightJoystickOverlay').offset().top
                        ){
                        return true;
                };
                return false
        });
        var i = 0;
        var joystickSocketUpdater = $interval(function(){
                if (i % 10 == 0){
                        console.log($scope.ws);
                        console.log($scope.fakeJoystickLeft);
                }
                i++
                $scope.joystick = {
                        left: {
                                deltaX                                  :
((Math.sqrt(Math.pow($scope.joystickLeft.deltaX(),
2))>Math.sqrt(Math.pow($scope.joystickLeft._stickRadius, 2)))
                                                                        ?
(($scope.joystickLeft.deltaX()<0)?-100:100)
```

```
                                                        :
($scope.joystickLeft.deltaX()/$scope.joystickLeft._stickRadius)*100),
                                deltaY                                  :
((Math.sqrt(Math.pow($scope.joystickLeft.deltaY(),
2))>Math.sqrt(Math.pow($scope.joystickLeft._stickRadius, 2)))
                                                        ?
(($scope.joystickLeft.deltaY()<0)?-100:100)
                                                        :
($scope.joystickLeft.deltaY()/$scope.joystickLeft._stickRadius)*100)*(-1)
                        },
                        right: {
                                deltaX                                  :
((Math.sqrt(Math.pow($scope.joystickRight.deltaX(),
2))>Math.sqrt(Math.pow($scope.joystickRight._stickRadius, 2)))
                                                        ?
(($scope.joystickRight.deltaX()<0)?-100:100)
                                                        :
($scope.joystickRight.deltaX()/$scope.joystickRight._stickRadius)*100),
                                deltaY                                  :
((Math.sqrt(Math.pow($scope.joystickRight.deltaY(),
2))>Math.sqrt(Math.pow($scope.joystickRight._stickRadius, 2)))
?(($scope.joystickRight.deltaY()<0)?-100:100)
                                                        :
($scope.joystickRight.deltaY()/$scope.joystickRight._stickRadius)*100)*(-
1)
                        }
                };
                console.log(JSON.stringify($scope.joystick));
                if($scope.ws.readyState == 1){
                        $rootScope.ws.send(JSON.stringify({         action:
'rcData', data: $scope.joystick }));
                }
        }, 10);
        $scope.$on('$destroy', function() {
                $scope.fakeJoystickLeft.destroy();
                $scope.joystickLeft.destroy();
                $scope.fakeJoystickRight.destroy();
                $scope.joystickRight.destroy();
                $interval.cancel(joystickSocketUpdater);

                $rootScope.ws.send(JSON.stringify({                 action:
'mainControllerRelease' }));
        });
});
```

8.6. Config.js

```
var app = angular.module('app', ['ngMaterial', 'ngRoute']);

app.config(function($mdThemingProvider,  $mdIconProvider,  $routeProvider,
$mdGestureProvider){

        $routeProvider.
                when('/home', {
                        templateUrl: 'partials/home.html',
```

```
                }).
                when('/fly', {
                        templateUrl: 'partials/fly.html'
                }).
                otherwise({
                        redirectTo: '/home'
                });

        /*$mdThemingProvider.definePalette('orange', {
                '50': 'FFF3E0',
                '100': 'FFE0B2',
                '200': 'FFCC80',
                '300': 'FFB74D',
                '400': 'FFA726',
                '500': 'FF9800',
                '600': 'FB8C00',
                '700': 'F57C00',
                '800': 'EF6C00',
                '900': 'E65100'
        });*/
        $mdThemingProvider.theme('default')
                .primaryPalette('blue')
                .accentPalette('indigo')
                .backgroundPalette('grey');

        $mdIconProvider
                .icon('rocket', '/svg/rocket20.svg');

});
```

8.7. Manual de usuario

Una vez encendido el dron sera necesario conectarse a la red wifi que se acaba de crear, SSID: Prone, PASSWORD: raspberry.

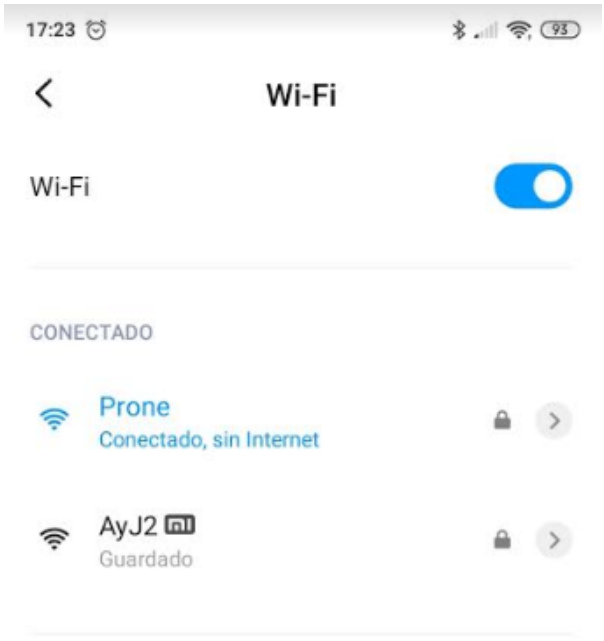

ILUSTRACIÓN 68: RED WIFI

A continuación, solamente habrá que entrar en la APP llamada SAM.

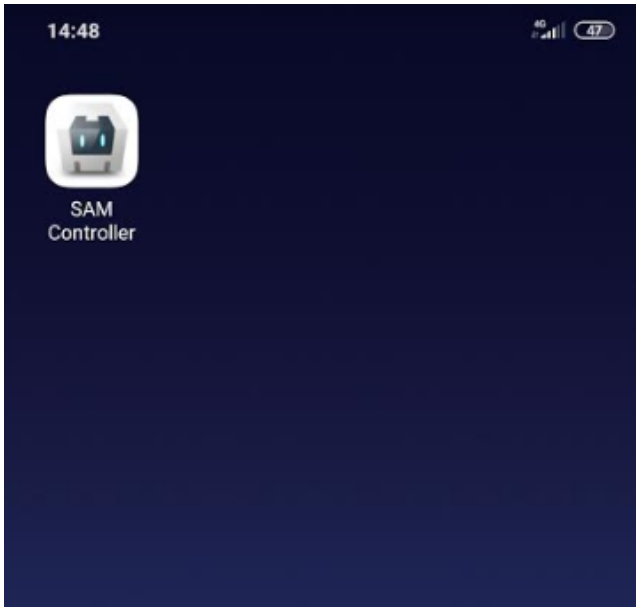

ILUSTRACIÓN 69: ICONO DE LA APP

Lanzar la conexión.

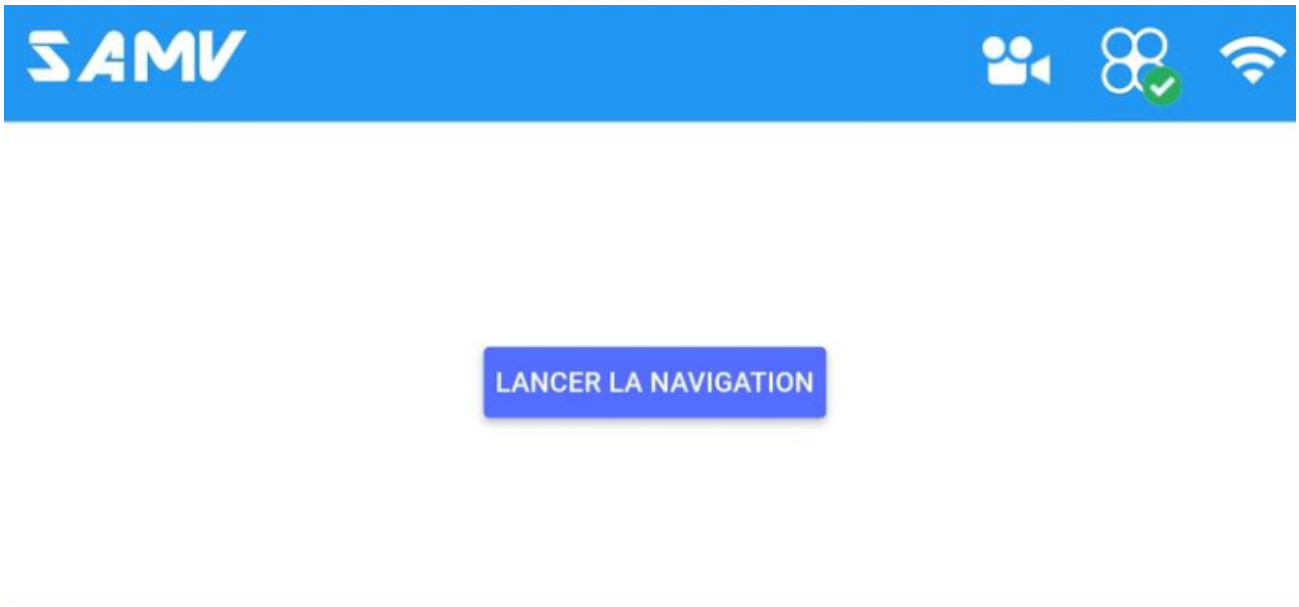

ILUSTRACIÓN 70: LANZAR CONEXIÓN

Disfrutar del vuelo, para ello colocar dos dedos sobre la pantalla como si se tratase de un mando, la mano izquierda marcara la potencia de los motores y la mano derecha la dirección.

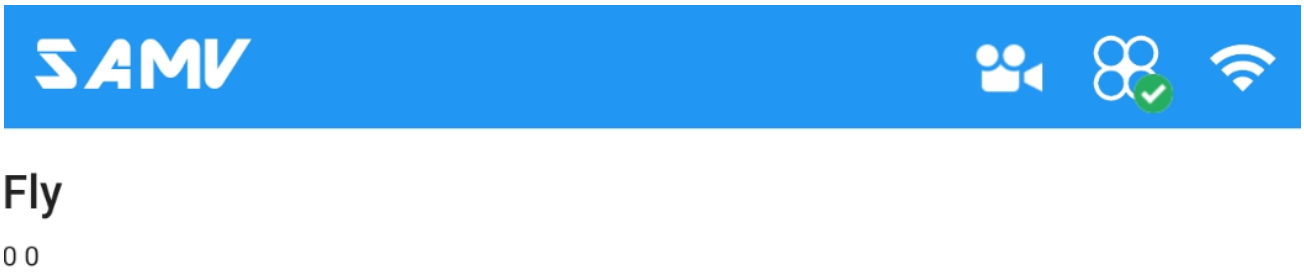

ILUSTRACIÓN 71: PANTALLA DE CONTROL