Dron de vuelo autónomo con reconocimiento basado en inteligencia artificial.

Autonomous flight drone with recognition based on artificial intelligence.

UNIVERSIDAD
COMPLUTENSE
MADRID

Trabajo de fin de grado del grado en ingeniería informática.
Curso académico 2019-2020.
Realizado por Juan Carlos De Alfonso Juliá.
Dirigido por Juan Jiménez Castellanos.

Dedicado a mis padres,
que siempre han creído en mí.

Resumen.

El desarrollo actual de las tecnologías de la información está haciendo cada vez más relevante el papel de la robótica y la inteligencia artificial en los distintos campos de trabajo e investigación (robots médicos, de exploración, de rescate, etc.).

Partiendo de estos avances, la idea de este proyecto surgió al ver en el telediario un helicóptero llevando a cabo la búsqueda de dos excursionistas perdidos en una zona boscosa. Pensando en esto, se me ocurrió si toda esta explosión de avances en el campo de la robótica se podía aplicar a esta tarea. Fue suficiente una búsqueda preliminar en Internet, para comprobar que, como era de esperar, la idea llevaba ya bastantes años en desarrollo y cubría múltiples facetas: El tipo de vehículos, los posibles escenarios de rescate, la planificación de la búsqueda, el guiado de los vehículos, su navegación segura, etc. De entre todas estas facetas, me pareció interesante proyectar la construcción de un cuadrotor, empleando componentes comerciales, y el desarrollo de los algoritmos necesarios para su navegación autónoma. Además, el vehículo llevaría como *payload* una cámara y un computador que permitiera, empleando redes neuronales, identificar posibles personas extraviadas.

Se decidió por tanto diseñar desde cero un aparato que fuera capaz de, dado un marco de coordenadas, realizar la búsqueda de objetivos sin necesidad de la supervisión ni el control humano, en concreto se pretende en este trabajo sentar las bases hardware y software que componen este tipo de aparatos.

Debido a las circunstancias del presente curso no ha sido posible llevar a cabo pruebas reales con el dron construido. Por ello, se desarrolló un modelo, fiel al sistema real, para demostrar la viabilidad de la propuesta mediante diversas pruebas ejecutadas en un simulador.

Palabras clave.

Dron, cuadricóptero, clasificación, detección, simulador, control, navegación, guiado, vehículo aéreo autónomo, UAV.

Summary.

The current development of data technology is making it more relevant the role of robotics and artificial intelligence within different areas of investigation and working programs (medical robots, exploring, rescue, etc).

Based on these advances, the idea of this project came from watching a helicopter, in a news program, that was searching for missing travelers in a wooded area. Thinking of this event, the idea of applying all this robotic improvement to this kind of task came to mind. A simple research on the internet was enough to confirm that this idea was being developed from many years ago, and it covered multiple facets. Such as the vehicle type, the diverse possible scenes for the rescue, the search planning, how to control and guide the vehicles, safe navigation, etc. From among these facets, it looked appealing to design a quadrotor using commercial components, just as the development of the necessary algorithm for its autonomous navigation. Furthermore, the vehicle would incorporate as a “payload”, a camera, and a computer that allowed it, with a neural net, identify possible missing people.

Therefore, I decided to design from the beginning a device that was able to search for targets without human supervision nor human control, giving a coordinates frame. It is expected to establish the hardware and software basis that put together this kind of device.

Due to present circumstances, it has not been possible to carry out a real test on the drone. Therefore, it developed a model, faithful to the real system, to demonstrate the proposal viability by several tests executed in a simulator.

Key words.

Drone, quadcopter, classification, detection, simulator, control, navigation, guidance, autonomous air vehicle, UAV.

Índice.

1. Introducción.

El uso de vehículos no tripulados tanto aéreos como terrestres o marinos ha experimentado en los últimos años un enorme desarrollo. Centrándonos en el mundo de los vehículos aéreos no tripulados (UAVs), una interesante revisión del estado del arte y sus aplicaciones se puede encontrar en la referencia (1), donde se destaca su uso en aplicaciones civiles tales como: la monitorización en tiempo real, la cobertura *wireless*, la búsqueda y rescate o la inspección de estructuras, entre otras. Se indica también que la nueva gran revolución en la tecnología de los UAVs será el desarrollo de UAVs inteligentes (*Smart UAVs*). Por último, se estima un potencial mercado para sus aplicaciones de unos 127 billones (USA) de dólares.
El concepto *Smart UAVs* supone dotar a los vehículos de la inteligencia necesaria para que puedan tomar decisiones, no solo relacionadas con el control y la estabilidad de su movimiento, sino también con los objetivos de la misión. Esto nos lleva a la integración de IA en los sistemas de abordo y en particular en el manejo del llamado *payload*, entendido como el hardware específico para la misión que debe llevar a cabo el UAV, no directamente relacionado con su control. En muchos casos, y en particular en el de los drones de pequeño tamaño, las tareas relacionadas con IA, se delegan a un computador específico conocido como *companion computer*, distinto del microcontrolador encargado del control de vuelo.
Dotar a los UAV de inteligencia, permite hacerlos colaborar entre ellos y con otros elementos *Smart* presentes en el escenario, por ejemplo, podrían integrarse como elementos móviles del *edge* un sistema IoT.
Entre las aplicaciones de los UAVS, en tareas civiles, una de las más prometedoras es su uso en tareas de búsqueda y rescate SAR (Search And Rescue). Equipos de drones, podrían sustituir a los actuales aviones y helicópteros empleados actualmente con notable ahorro de costes y riesgos, especialmente en casos de grandes desastres tanto naturales como de origen humano (2), (3).
En un contexto SAR, son muchos los aspectos que es necesario cubrir: El tipo de vehículos, los posibles escenarios de rescate, la planificación de la búsqueda, el guiado de los vehículos, su navegación segura, el uso coordinado (cooperativo) de múltiples vehículos, etc.

De entre todas estas facetas, el presente proyecto se centra en el primer nivel básico: la construcción de un UAV, empleando componentes comerciales, y el desarrollo de los algoritmos necesarios para su navegación autónoma. Además, el vehículo llevaría como *payload* una cámara y un computador que permitiera, empleando redes neuronales, identificar posibles personas extraviadas.

Partiendo de la idea básica de que un dron (UAV) es un vehículo aéreo no tripulado, este trabajo tiene la intención de, primero dar al lector unas nociones básicas sobre las características más relevantes de este tipo de aparatos, a continuación, se procederá a construir un prototipo funcional desde el punto de vista hardware: explicando la toma de decisiones en cuanto a selección de componentes se refiere e intentando detallar todo el proceso de construcción.

Una vez se tiene el prototipo, la intención es resumir la programación de cada uno de los sensores necesarios para el vuelo de este tipo de aparatos demostrando el funcionamiento de estos mediante una serie de pruebas teniendo en cuenta que, como el modelo final será virtual, no se usará esta parte para el desarrollo del simulador.

A continuación, y siguiendo la estructura en la que está dividida la memoria se explica la creación desde cero de una aplicación de programación de tareas y su funcionamiento. De igual manera se describirá el reconocedor que empleará inteligencia artificial para reconocer sus objetivos.

Por último, en esta memoria se encuentra la especificación y creación de un modelo dinámico, así como de un sistema de control de vuelo con las características físicas del sistema hardware creado al principio. Una parte del capítulo se dedicará al análisis del funcionamiento de este modelo realizando recorridos usando la aplicación de programación de tareas como guía y una interfaz donde se detalla el estado del aparato.

1.1 Plan de trabajo.

1. Realización de una investigación previa sobra las características y tipos de drones, así como de los elementos que este pueda necesitar para ser construido.

2. Construir, basándose en lo aprendido en el punto uno un prototipo viable desde el punto de vista hardware.

3. Una vez terminado el desarrollo hardware, analizar qué sensores son vitales para que este pueda funcionar y dar un programa para el control de estos.

4. Añadir una interfaz de programación de tareas, estudiando los distintos métodos de comunicación de redes y tipos de implementación.

5. Especificar la funcionalidad de reconocimiento basada en un algoritmo de inteligencia artificial y realizar una serie de pruebas para comprobar su funcionamiento.

6. Crear un modelo virtual semejante al dron real y realizar una simulación completa que demuestre la funcionalidad de este.

1.2 Objetivos.

Diseñar y construir un prototipo funcional y viable.

Sentar las bases de programación de los diversos sensores que conforman un cuadricóptero.

Crear una aplicación de programación de tareas completa.

Diseñar un reconocedor basado en inteligencia artificial y comprobar su funcionamiento.

Proporcionar un modelo virtual semejante en cuanto a diseño y especificaciones al prototipo real, que sea capaz de simular por completo el funcionamiento del aparato, así como su interacción con el medio y con la aplicación de control.

Adquirir distintos conocimientos en el ámbito del diseño hardware, la física y la robótica.

Poner en práctica lo aprendido en asignaturas como Inteligencia artificial para el reconocedor o redes para el protocolo de comunicación.

1.3 Información adicional.

Con el fin de proporcionar al lector una explicación más detallada del proyecto, en los apartados siguientes se hace referencia a partes del código desarrollado. Todo este código junto con otra información se puede encontrar en el repositorio de GitHub juancalf/TFG (4).

1. Introduction.

The use of non-driven vehicles as aerial, terrestrial, and marine has experienced an important growth in the last few years. Focusing our attention on the non-driven aerial vehicles (UAVs), there is an interesting review of its art state and application that can be found on the reference (1), where its use is highlighted in civil applications such as real-time monitoring, the wireless coverage, searching, and rescue, or the structure inspection, among others. It's also indicated that the great revolution of the UAVs' technology will be the development of smart UAVs. Lastly, it is estimated a potential market for its applications of 127 billion dollars (USA).

The Smart UAVs concept entails providing the vehicles with the necessary intelligence so they can make decisions, not only associated with control and movement stability, but also with the mission's objects. This takes us to the integration of artificial intelligence onboard systems, particularly, on the use of payload. Understood as the specific hardware for the mission that must carry out the UAV, not directly related to its control. In many cases, especially in small size drones, the tasks associated with artificial

intelligence are delegated to a specific computer known as “companion computer”, distinct to the microcontroller in charge of the flight control.

Providing the UAV with intelligence allows them to cooperate and with other Smart elements present on stage, for example, they could integrate themselves as mobile elements from the edge in a loT system.

Among the applications on UAVs, civil job, one of the most promising is its use on searching and rescuing tasks (SAR). Drones equipment could replace the current helicopters and airplanes with a significant cost and risk saving, especially in great natural or human (2) disasters (3).
In a SAR context, many aspects need to be covered up. The vehicle type, the possible rescue stages, the search planning, the vehicle guide, save navigation, the coordinate use of diverse vehicles, etc.

Of all these facets, this project is focused on the basic level, the UAV construction using commercial components, and putting into practice the required algorithm for its autonomous navigation. Besides, the vehicle would carry as a payload a camera and a computer that would allow it to identify possible missing people by neuronal nets.

Based on the basic idea of a UAV, a non-driven aerial vehicle, this project is intended to provide the reader a few basic notions of the more relevant characteristics of this kind of device, continuing with constructing a functional prototype from a hardware point of view. Explaining the decision making regarding the selection of components and trying to detail the entire construction process.

Once the prototype is done, the intention is to summarize each sensor programming necessary for the flight, demonstrating correct functioning by way of different tests. Keeping in mind that the final model will be virtual, this part of the project won’t be used in the simulator development.
Continuing with the report structure, it is explained the creation of a programming task application and its operation right from the start. In the same way, the recognizer that will use artificial intelligence on its target will be explained.

Lastly, in this report, the specification and creation of a dynamic model are found, just as a flight control system with the hardware system physics characteristic, created from the beginning. A part of the chapter is dedicated to the analysis of the operation of this model, conducting routs using the programming of task application as a guide, and an interface where the condition of the device is detailed.

1.1 Workplan.

1. Execution of a previous investigation about the characteristics and types of drones, just as the elements that this might need to be constructed.

2. To build the drone, based on what has been learned on point number one. A viable prototype from the hardware point of view.

3. Once the hardware progress is finished, it is necessary to analyze which sensors are vital to the drone to work as planned and to give a program for its control.

4. To add a task programming interface, studying various methods of net communication and types of implementation.

5. To specify the reconnaissance functionality based on an artificial intelligence algorithm and conducting serial tasks to verify its functioning.

6. To create a virtual model similar to the real drone, and conduct a complete simulation that demonstrates its working.

1.2 Objectives.

To design and build a functional and viable prototype.

To set the programming basis of the different sensors that define a quadcopter.

To create a complete task program application.

To design a recognizer based on artificial intelligence and to check its working.

To provide a virtual model similar to the real prototype in terms of design and specifications, which must be able to simulate completely the device functioning, just as its interaction with the environment and the control application.

To acquire knowledge in the hardware design field, physics, and robotics.

To put into practice all that's been learned in subjects such as artificial intelligence for the recognizer, or networking for the communication protocols.

1.3 Additional Information.

With the purpose to provide the reader with a more detailed explanation of the project, in the following chapters refer to parts of the developed code. All this code joined to information can be found on GitHub archive juancalf/TFG (4).

2. Diseño hardware.

2.1 Tipos de drones.

A grandes rasgos, un dron se puede clasificar en tres categorías (5):

- Drones de ala fija: debido a su construcción aerodinámica poseen una gran autonomía y están destinados principalmente a labores donde es clave la posibilidad de recorrer áreas extensas, su gran inconveniente es que debido a su estructura necesitan de una zona de despegue y aterrizaje pues no son capaces de hacerlo verticalmente.

Fig 1. Dron de ala fija (6).

- Helicópteros: con una autonomía normalmente inferior a la de un dron de ala fija pero superior a la de un multirotor, son aparatos de uso polivalente con amplia capacidad de carga. La principal desventaja que presentan estos aparatos es, además de la complejidad mecánica, la dificultad de pilotaje tanto si se pilotan de forma manual o automática.

- Multirotores: son los drones que más éxito están teniendo actualmente debido a la polivalencia que presentan y a su bajo precio en relación a los dos tipos anteriores, constan de una serie de motores en posición vertical que le proporciona una gran estabilidad en el aire permitiendo realizar labores donde esta característica es clave, como la fotografía.

Fig 2. Dron multirotor (7).

Según la cantidad de motores que lleva los podemos subclasificar en cuadricópteros (4 motores), hexacópteros (6 motores), etc.

Para este proyecto se han descartado los dos primeros tipos por su dificultad técnica y coste, decidiéndose por un cuadricóptero. Otro factor clave para la selección de este modelo es que su desarrollo se puede simplificar aplicando las ideas desarrolladas por B. Francis (56) para desarrollar un modelo y un sistema de control de vuelo para un dron, sencillo y robusto.

2.2 Elementos que componen un cuadrotor.

Los elementos que componen este tipo de aparatos son muy variados según el uso al que se destinen, pero generalmente están compuestos de los siguientes elementos (8):

El primer elemento que se necesita para crear un aparato de esta clase es un marco o chasis, podría decirse que es el esqueleto sobre el cual se van a colocar el resto de componentes.

A continuación, se necesita un grupo de propulsión formado en este caso por cuatro motores con sus respectivas hélices. Estos motores deben poder variar la potencia para realizar distintas maniobras y eso hace que sea necesario un nuevo elemento por cada motor: el variador o ESC[1] que permitirá variar la velocidad de giro de los motores.

Fig 3. Power distribution board de la marca Matek (9).

Además, se necesitará una batería que suministre corriente al aparato durante el vuelo y una forma de que esa energía llegue a todas partes, Es preciso además que la potencia y tensión suministrada a cada elemento sea la adecuada, según sus especificaciones. Aunque sería posible diseñar reguladores independientes, es más práctico y fiable emplear una placa con ese fin, esta placa recibe el nombre de pdb[2], su único fin es regular los voltajes que llegan a cada parte del dron.

A todo esto, hay que sumarle una variedad de sensores que permiten al dron interactuar con el entorno y recibir información sobre él, siendo los más importantes los siguientes:

- Un módulo GPS que permita saber en todo momento donde se encuentra el aparato.
- Un sensor barométrico para conocer la altura a la que se está volando.
- Una cámara.

[1] Electronic Speed Controller.
[2] Power Distribution Board.

- Una unidad de movimiento inercial o IMU que permitirá saber la actitud del dron y que incluya una brújula (orientación).

Si bien aquí se ha dado una breve definición de los sensores, en capítulos posteriores se harán descripciones más detalladas del funcionamiento y uso de estos.

Por último, se necesita un controlador de vuelo (un microprocesador), que se encargue de tomar todas las decisiones y recibir la telemetría de los sensores.

2.2 Selección definitiva de componentes para nuestro proyecto.

Para el chasis se ha elegido el modelo F450 de la marca DJI dado que al tener una superficie amplía, la colocación de los componentes es más sencilla, además cuenta con un tablero con una pdb integrada lo que hace mucho más fácil soldar y colocar los distintos cables.

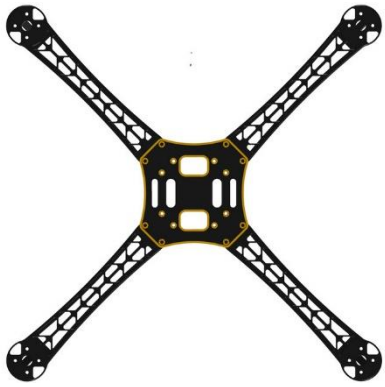

Fig 4. Diseño de un marco F450 (10).

Respecto al microprocesador hay infinidad de placas en el mercado válidas para este proyecto, se ha elegido una Raspberry pi modelo 3B+, como *companion compute;* es decir, como computador compañero del controlador de vuelo del dron, encargado de llevar a cabo las tareas computacionales más complejas. Al contar con wifi y la posibilidad de instalar un sistema operativo basado en Linux facilita la tarea de comunicación con el dron, el procesado de las imágenes recogidas por la cámara durante el vuelo y la de ejecución de un algoritmo de AI para la identificación de imágenes.

La elección de la Rasperry pi tiene otro factor clave, y es su completa compatibilidad con el hat NAVIO2. Se trata de un controlador de vuelo diseñado específicamente para esta placa de desarrollo. Dicho hat permite integrar una de las partes más complejas, los sensores.

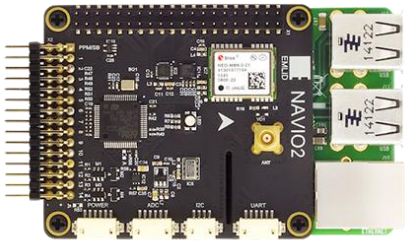

Fig 5. Placa Navio2 montada sobre Raspberry (11).

Navio2 cuenta con un barómetro de alta resolución (MS5611), dos unidades inerciales IMU (MPU9250 y LSM9DS1), la posibilidad de ser alimentado desde una batería LIPO mediante un adaptador, un módulo GPS (Ublox M8N) y 14 canales de salida PWM[3] para poder controlar los motores (solo se usarán 4). Esta hat lleva otros sensores y conectores que no se mencionan dado que no se usarán para este proyecto (12).

El siguiente elemento es la cámara, se ha elegido una fabricada por Raspberry, el modelo RPI-CAM-V2 por su pequeño tamaño y reducido peso, además de por su resolución que llega hasta los 1080p (13).

Para la propulsión se han seleccionado los motores a2212 13t con hélices de 25cm de 10” de diámetro y 4.5” de paso. Se ha seleccionado este motor principalmente por el peso que es capaz de levantar, el dron pesará aproximadamente 2 kilos por lo que cada motor deberá poder levantar 500 gramos y este motor según se especifica tiene un rango de trabajo de entre 300 y 800 gramos lo cual proporciona un margen de trabajo seguro (14).

Respecto al variador, los motores venían con un ESC 30A (Electronic Speed Controller) soldado que soportaba 30A y sabiendo que los motores como máximo consumirán 10A parecían los idóneos y no se han cambiado.

Con este último elemento se han elegido todos los elementos necesarios para construir un prototipo, en el capítulo siguiente se explicará cómo ha sido el ensamblaje de los distintos componentes además de darse una explicación más detallada de alguno de ellos.

2.3 Construcción del prototipo.

2.3.1 Conexión entre variadores y motores.

El motor seleccionado en el apartado anterior es un motor trifásico sin escobillas, esto quiere decir que su estructura interna simplificada corresponde a la de la figura 6, en donde hay tres cables con el mismo voltaje, cada uno conectado a una bobina separadas entre si 120º.

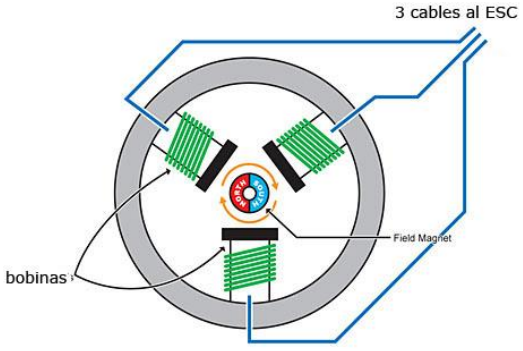

Fig 6. Esquema de un motor trifase (15).

Estos cables están conectados al variador que tiene la función de activar cada una de las bobinas en orden, creando un campo magnético que produce el giro del motor.

[3] Pulse Wide Modulation, se describirá más adelante.

Teniendo esto en cuenta es relativamente sencillo cambiar el sentido del giro, bastaría con cambiar el orden de conexión de los dos cables exteriores dejando el central inmóvil, la razón es que el cable central siempre se activará en segundo lugar y no tiene sentido cambiarlo para cambiar el sentido del giro.

Es importante entender este concepto dado que va a ser necesario invertir el sentido del giro en ocasiones para el correcto funcionamiento del dron, la razón se explicará en el siguiente apartado.

2.3.2 Instalación de las hélices y ajuste del sentido de giro.

Cuando a un motor se le instala una hélice y esta gira en horizontal va a generar una fuerza de sustentación que es lo que se busca para mantener al dron en el aire, sin embargo, debido al giro de la hélice, se genera un par motor en puntos distintos que tiende a parar la hélice, haciendo girar el dron a su alrededor. Por este motivo, la colocación de las hélices y su sentido de giro debe ser simétrico para poder contrarrestar los momentos angulares generados, evitando así que el dron gire sobre sí mismo y se mantenga en una posición estable elevándose (16). Para lograr este propósito es necesario recurrir al concepto explicado en el apartado anterior sobre como invertir el sentido de giro de un motor.

Por lo tanto, la situación final debe ser igual a la de la figura 7, en la que, como ya se ha mencionado anteriormente, la simetría de los motores anula los momentos angulares de giro del dron. Además, los motores llevarán hélices de empuje directo CB/CCB (Sentido horario/Sentido antihorario).

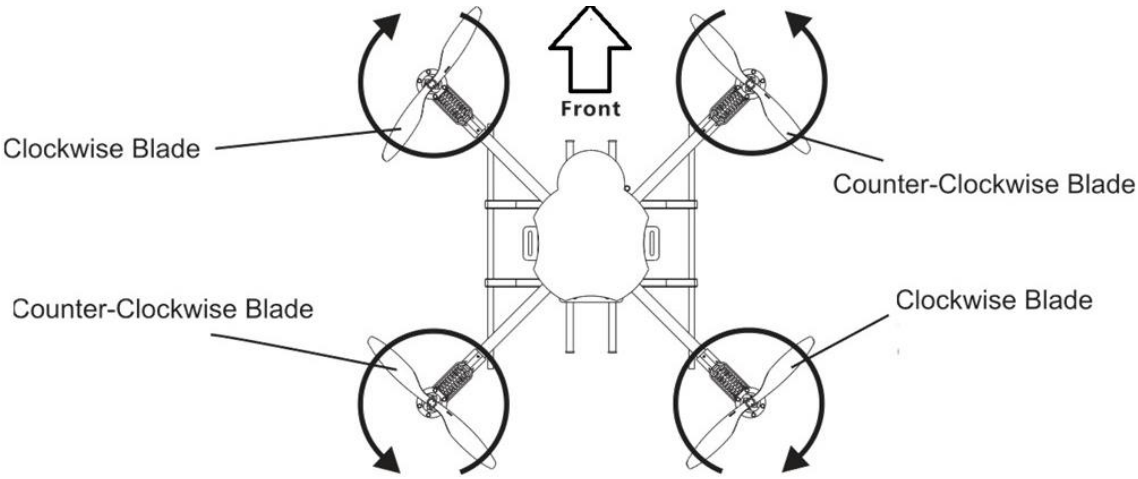

Fig 7. Esquema de rotación de los motores (17).

2.3.3 Conexión entre la Raspberry, la batería y los variadores.

Respecto a la parte interna del variador hay 4 cables, dos de ellos (normalmente rojo y negro) son los cables de alimentación que irán conectados a la pdb y posteriormente a la batería. Los otros dos cables que normalmente vienen en un conector gpio hembra van conectados al controlador (en este caso a los puertos pwm de la Navio2), son de color negro y blanco, siendo el primero tierra (GND) y el segundo la señal PWM[4].

Una vez que está claro qué cables deben ir a la batería hay que soldarlos a la pdb del chasis junto, con el módulo de corriente siguiendo el esquema de la figura 8.

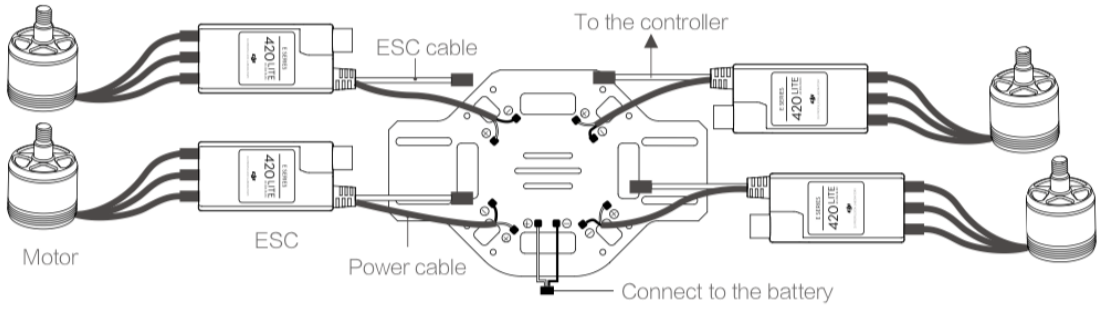

Fig 8. Diagrama de instalación de los variadores y el conector de potencia a una placa F450 (18).

2.3.4 Instalación de la Raspberry en el chasis.

La instalación de las placas Navio2 y Raspberry plantean un problema, durante el vuelo los motores al girar generan una vibración que puede afectar de manera considerable las lecturas de sensores como el barómetro o la IMU, este inconveniente se puede solucionar utilizando un soporte diseñado por Navio (19).

Este diseño consiste en una funda impresa íntegramente en 3D con unos espacios en la parte inferior para colocar unos elementos de silicona anti-vibración. A este diseño base se le han añadido diferentes modificaciones que permiten atornillar el conjunto a la parte superior del dron.

[4] Una señal PWM (Pulse Width Modulation) permite modificar en voltaje eficaz suministrado a los motores, modificando el ancho de un pulso periódico que actúa conectando y desconectando la fuente de voltaje a las fases del motor.

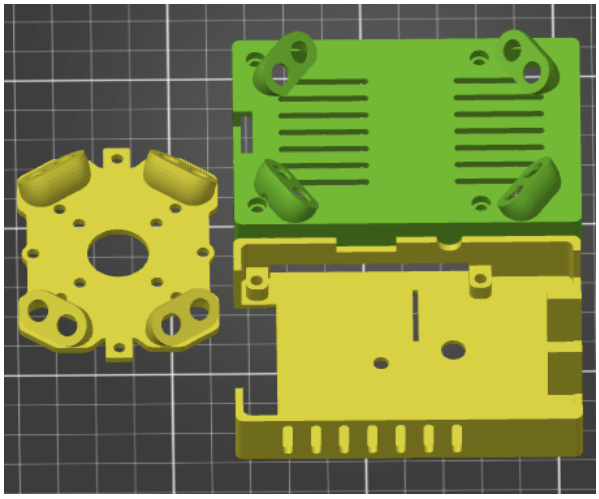

Fig 9. Diseño de la funda modificada.

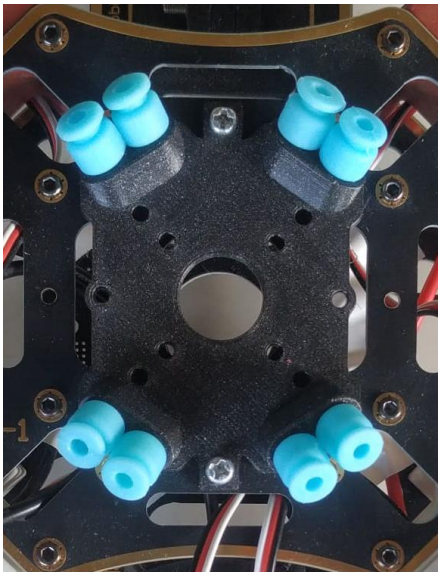

Fig 10. Parte inferior del soporte atornillado al chasis.

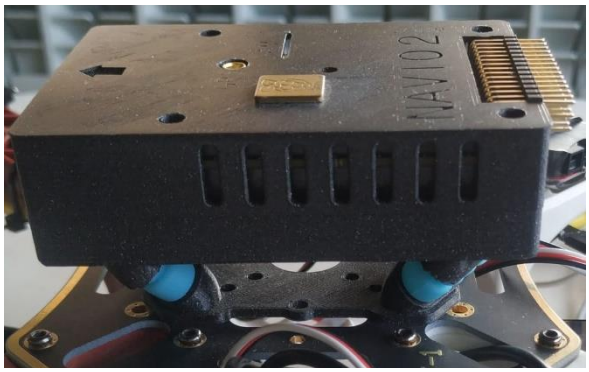

Fig 11. Imagen final de la Raspberry pi instalada.

2.3.5 Instalación de la cámara.

La dificultad de instalación de este tipo de cámaras radica en la correcta colocación en la parte delantera del dron, de forma que se pueda inclinar para grabar el suelo.

Dado que la cámara no traía ningún armazón adicional se ha recurrido nuevamente a la impresión 3D rediseñando un diseño encontrado (20), añadiéndole un soporte con el que poder orientar el ángulo de forma sencilla y unos agujeros adicionales para poder atornillarla tal y como se ve en las figuras 12 y 13.

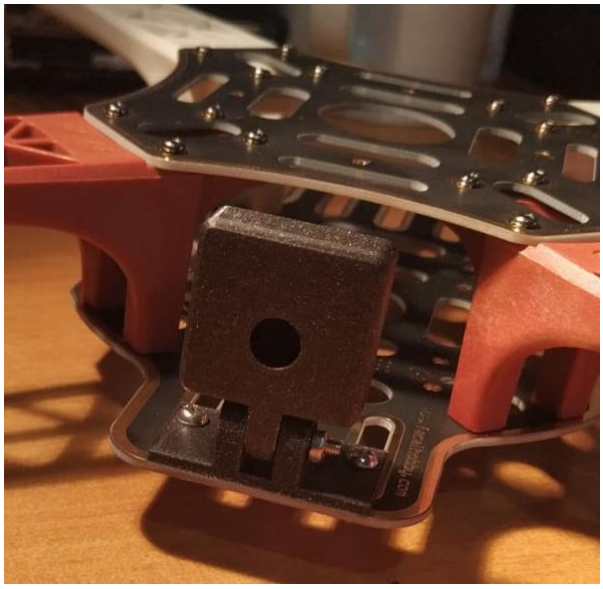

Fig 12. Soporte para la cámara (I).

Fig 13. Soporte para la cámara (II).

Por otro lado, la conexión con la Raspberry es trivial, solo hay que introducir el cable plano en una ranura de la placa.

2.3.6 Otros elementos.

En este punto, el dron está terminado a falta de dos elementos, el primero es la antena GPS, esta antena va separada del resto del hat Navio y se va a colocar en la parte trasera del aparato con una almohadilla adhesiva, se unirá a la placa a través de un conector rf situado en la parte superior.

Fig 14. conector rf del hat Navio2.

El otro elemento que resta para la finalización de la fase de construcción es un sensor que se conecta a la batería y que da un aviso sonoro en caso de que el nivel de batería sea bajo, este sensor no es necesario para el funcionamiento, pero permite usar el prototipo con la seguridad de que no se quedará sin batería en vuelo.

2.3.7 Prototipo final.

Una vez finalizados todos los pasos anteriores el dron está terminado a nivel de hardware. A continuación, en las figuras 15 y 16 se puede ver el resultado final.

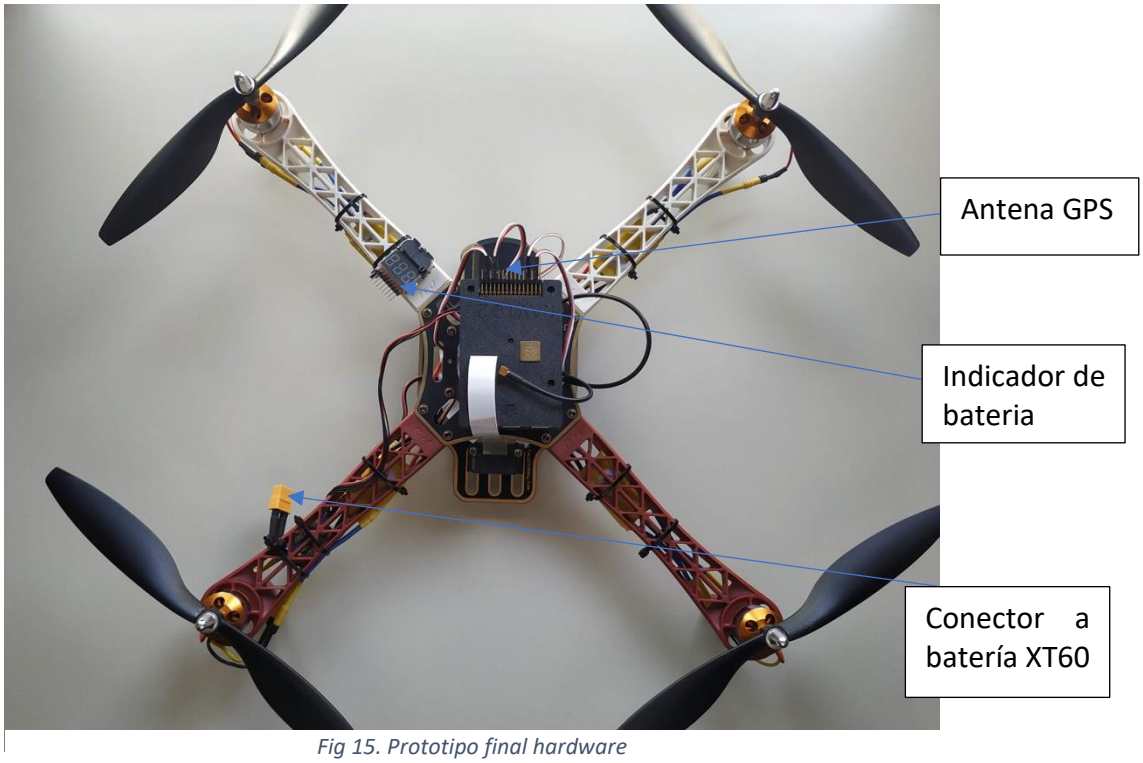

Fig 15. Prototipo final hardware

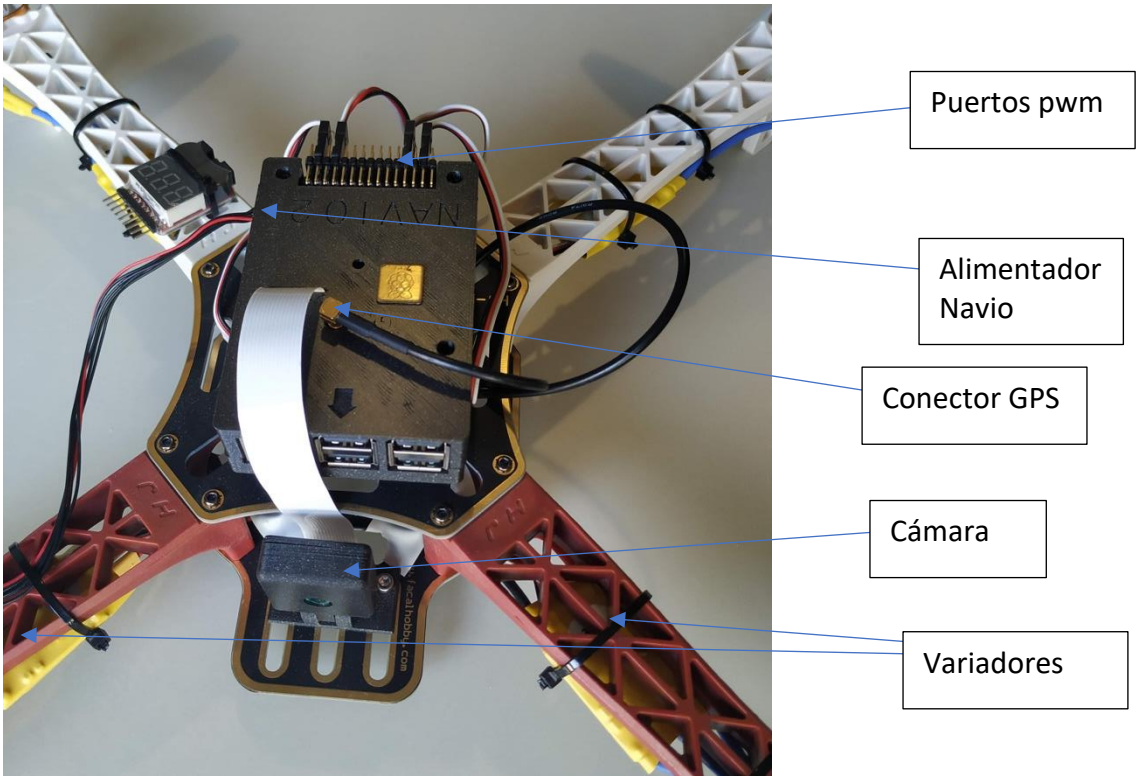

Fig 16. Prototipo final hardware (vista ampliada).

2.4 Resumen de las propiedades.

En este apartado se busca hacer referencia a las características más relevantes del aparato construido con el fin de que en el punto 6 (construcción de un modelo virtual) se puedan introducir de manera más clara estas especificaciones:

Especificación	Valor	Unidad
Peso total	2102	gramos
Peso de cada motor	190.3	gramos
Longitud de cada brazo	18.5	centímetros
Relación par motor	1.	-

3. Diseño software.

3.1 Introducción.

El propósito de este capítulo es el de explicar y mostrar el uso de cada uno de los sensores principales que conforman un cuadricóptero, si bien los scripts se pueden utilizar y son válidos, dado que nuestro modelo final es virtual no se hará uso de ellos en el simulador.

Para la lectura bruta de los sensores y las comunicaciones entre la Navio y la Raspberry se ha empleado el software original proporcionado por el fabricante del hat. Al estar todo el código escrito en Python, se ha decidido usar este lenguaje para implementar el desarrollo software de a bordo.

3.2 Requisito previo. Configuración del sistema operativo.

Para poder utilizar la placa Navio2 es necesario utilizar una versión modificada del sistema operativo Raspbian. Este software lo proporciona el fabricante del microcontrolador y no lleva incorporada interfaz gráfica, por lo que en este caso lo más cómodo desde mi punto de vista ha sido trabajar mediante ssh[5], para el desarrollo y la depuración del código de abordo. Este protocolo permite administrar de forma remota otros dispositivos a través de internet mediante un mecanismo de autenticación (21).

Según especifica el fabricante en su documentación, para poder conectarnos a internet es necesario modificar el archivo */boot/wpa_supplicant.conf* añadiendo la ssid de la red (nombre de la red), la contraseña y el tipo de seguridad de la red wifi (22). La modificación debería ser parecida a esta:

[5] Secure Shell.

```
network ={
  ssid = “ssid de la red”
  psk = “contraseña de la red”
  key_mgmt = “seguridad de la red (WPA-PSK, etc.)”
}
```

3.3 Sensor barométrico.

3.3.1 Introducción teórica.

Un barómetro es un aparato utilizado para medir la presión atmosférica del aire a una determinada altura, esto implica que a menor altura la presión será mayor y viceversa. Usando este tipo de sensores y aplicando una serie de cálculos es posible establecer la relación de la presión atmosférica con la altura a la que se encuentra el barómetro (23).

Si se quiere obtener la altura en metros basta con aplicar la siguiente fórmula:

$$h = \left(1 - \left(\frac{p}{p0}\right)\right)^{\frac{1}{5.255}}$$

En donde:

- h: altura en metros a la que se encuentra el barómetro
- P_0: presión media a nivel del mar (1013.25 hPa) (24).

Se debe tener en cuenta que la fórmula expuesta no va a proporcionar la altura a la que se encuentra el sensor en función del suelo, sino al nivel del mar. De modo que para obtener la altura según el nivel del suelo es necesario medir en primer lugar, la altura del suelo sobre el nivel del mar. Dicho valor se toma como referencia antes del vuelo, de modo que al resto de las medidas realizadas por el barómetro durante el vuelo se les resta dicha referencia, obteniendo así la altura de vuelo relativa la suelo.

3.3.2 Aplicación práctica.

En este caso concreto, la placa Navio2 lleva un sensor barométrico incorporado (MS5611).
El código referente al uso de dicho sensor se encuentra en el archivo *sensores/ barómetro.py* en el repositorio mencionado en la sección 1.

Dentro de este archivo hay varias subrutinas, pero para poder ser capaces de utilizar correctamente este sensor, hay que primero llamar a la de inicialización, la cual realizará las mediciones a nivel de tierra necesarias para conocer posteriormente la altura de vuelo, aplicando la fórmula del apartado anterior en la función *_getAltitud_*.

El archivo cuenta con dos funciones distintas para leer la altura, una que proporciona la altitud real a nivel de tierra (*getAltitud_real*) y otra a nivel del mar (*getAltitud_abs*).

Cabe mencionar que ambas funciones llevan dos parámetros, uno de muestras y otro de decimales, el primero hace referencia al número de muestras a tomar de forma seguida para hacer una media entre ellas, y el segundo a la precisión en coma simple de éstas. Esto permite que un sensor barométrico, que es un componente que se ve afectado por diversos factores (temperatura, radiación, etc.), nos proporcione medidas más precisas.

3.3.3 Demostración de funcionamiento.

Para poder demostrar que el barómetro se ha programado correctamente se va a colocar primero el dron en el suelo en donde se llamará a la función de inicialización, al hacer esto el programa deberá guardar la altura a la que se encuentra el sensor a nivel del mar. A continuación, se eleva el dron exactamente 80cm y se realiza la llamada a la función *getAltitud_real* que debería devolver un valor aproximado a 0.8m. En las figuras 17 y 18 se puede ver cómo ha sido colocado el dron para tomar las medidas.

Fig 17. Dron a nivel del suelo.

Fig 18. Dron a 80cm del suelo.

```
pi@navio:~/my_navV2 $ python -i barometro.py
>>> import barometro
>>> bar = barometro.Barometro()
calibrando...
altitudBase: 623.83
>>> bar.getAltitud_real(3,3)
altitud: -0.004
>>> bar.getAltitud_real(3,3)
altitud: 0.792
```

Fig 19. Terminal con la telemetría del sensor.

En este fragmento de terminal se puede comprobar cómo se realiza la prueba, teniendo en cuenta que ejecutar el script con el parámetro -i permite entrar en el modo terminal.

Primero se realizan las importaciones necesarias para luego leer la altitud a nivel del mar en tierra (623.83 metros). A continuación, se realiza una segunda medición donde el desvío es mínimo (0.004 metros). Posteriormente se eleva hasta los 80cm y se obtiene otra medición de 79.2cm que es aproximadamente el valor esperado.

Lógicamente el valor no ha coincidido exactamente, pero la variación es despreciable (inferior a un cm), aun así, las llamadas a la función de leer la altitud se realizan solo con tres muestras y tres decimales, si se quisiera se podría ampliar este valor para aumentar la precisión, sin embargo, dada la naturaleza del sistema es difícil mejorar la exactitud de las medidas.

3.4 Módulo GPS.

3.4.1 Introducción teórica.

Dentro del hat Navio2 se encuentra el módulo GPS NEO-M8N. Dicho módulo va a permitir obtener una serie de datos de los distintos satélites a los que este se conecte. En principio un módulo GPS necesita de al menos tres satélites para poder funcionar y poder triangular la posición del receptor siendo posible conectarse a más satélites para mejorar la precisión de dicha triangulación.

El intercambio de mensajes entre el receptor y los satélites se realiza mediante el protocolo ubx, este protocolo implica que, a cada mensaje enviado por un satélite, este lleva adjunto un identificador que permite saber qué contiene dicho mensaje y cómo debemos decodificarlo (25).

En este apartado se van a explicar los mensajes con Id NAV-POSLLH, que son aquellos que proporcionan información referente a la posición geodésica y por tanto son cruciales para localizar y controlar el dron. Un mensaje calificado con este Id concretamente va a proporcionar los datos mostrados en la siguiente figura:

Byte Offset	*Number Format*	*Scaling*	*Name*	*Unit*	*Description*
0	U4	-	`iTOW`	ms	GPS Millisecond Time of Week
4	I4	1e-7	`lon`	deg	Longitude
8	I4	1e-7	`lat`	deg	Latitude
12	I4	-	`height`	mm	Height above Ellipsoid
16	I4	-	`hMSL`	mm	Height above mean sea level
20	U4	-	`hAcc`	mm	Horizontal Accuracy Estimate
24	U4	-	`vAcc`	mm	Vertical Accuracy Estimate

Fig 20. Contenido de un mensaje NAV-POSLLH (26).

En el siguiente apartado se va a implementar únicamente la lectura de la longitud y latitud, si bien es cierto que se podría también usar la altitud, la precisión de este sensor para este valor es muy inferior respecto a los valores proporcionados por el barómetro.

3.4.2 Aplicación práctica.

El código que implementa este módulo se encuentra en el archivo */sensores/gps.py*.

Dentro de este archivo se encuentra, primero la declaración de una instancia del módulo GPS en la línea 6.

A continuación, hay dos métodos: el primero es un método de inicialización de parámetros proporcionado por Navio para poder utilizar el módulo, seguido del método *get_coords*. Este método es el que, al llamarlo, va a devolver la latitud y longitud en una tupla de tipo *float.*
Como se ha explicado en el apartado anterior, el protocolo UBX proporciona varios tipos de mensajes de forma periódica, cada uno con una información distinta. Dado que la latitud y la longitud se encuentran en el protocolo NAV-POSLLH, en el código hay una condición que comprueba el tipo mensaje recibido mediante un bucle que descartará todos los mensajes recibidos hasta encontrar uno correspondiente al protocolo deseado (líneas 48 a 50).

La siguiente acción de la función es, de todos los parámetros del método, obtener la latitud y la longitud que se encuentran en la segunda y tercera posición del vector de datos recibido.
Una vez que tenemos la latitud y la longitud, hay que procesarlas para eliminar los nombres asociados dado que el array contendrá algo parecido al vector siguiente:
[“Longitude=-53.800018”,” Latitude=41.4087435”]
Una vez eliminados los nombres de cada variable, basta con convertirlos a formato decimal llamando al método float() y devolver la tupla resultante.

Aclaración: el módulo tarda entre uno y tres minutos en realizar la conexión con tres satélites y devolver datos válidos, hasta entonces la función retornará cero.

3.5 Controlador PWM.

3.5.1 Introducción teórica.

Un controlador PWM[6] permite controlar la velocidad a la que girará un motor. Para realizar esta tarea, genera una señal cuadrada que es interpretada por los variadores, y dependiendo del ancho del pulso generado, hará girar a los motores a una velocidad u otra.

El primer punto a tener en cuenta es que, por convenio, todos los motores y servos funcionan a una frecuencia de 50hz. Esto implica que nuestro periodo será de 0.02s o bien 20 ms, como se explica a continuación (27).

$$T = \frac{1}{f} = \frac{1}{50hz} = 0.02s = 20ms$$

Como se ha mencionado antes, para poder controlar estos motores debemos variar el ancho del pulso de nuestra señal, sabiendo que:

[6] Pulse Width Modulation

- La velocidad más baja (motor parado) será cuando el ancho del pulso sea de 1ms (Fig 21).

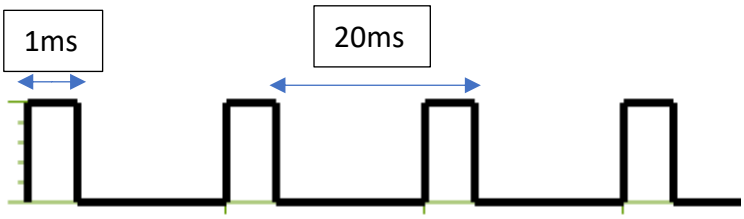

Fig 21. Señal pwm de 1ms (min RPM) ilustrativo: pulso no a escala.

- La velocidad más alta ocurrirá cuando el ancho del pulso sea de 2ms (fig 22).

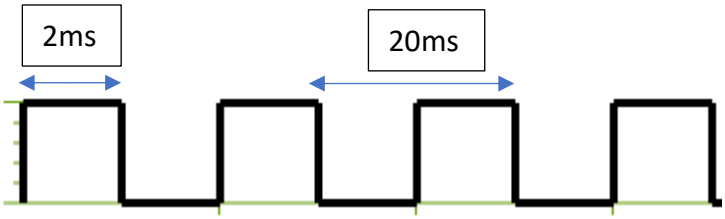

Fig 22. Señal pwm de 2ms (max RPM) ilustrativo: pulso no a escala).

Deducimos que, si por ejemplo se establece un ancho de banda de 1.5ms el motor funcionará al 50% de su potencia total.

El siguiente punto a tener en cuenta para poder usar correctamente un variador es calibrarlo. Es preciso ajustar los límites de potencia del variador a un valor mínimo y a uno máximo del ancho del pulso de la señal PWM. De no hacer esta calibración a la hora de volar el aparato, los motores podrían estar descompensados, generando diversos problemas.
Para poder solventar este problema los variadores llevan incorporada una función de calibrado que se realiza de la siguiente manera:

1. Se desconecta la batería de los motores y variadores.
2. Se establece la potencia del variador al mínimo (1ms).
3. Se conecta la batería y se espera al sonido de confirmación (un pitido).
4. Se establece la potencia del variador al máximo (2ms).
5. Se recibe otro sonido de confirmación que indica que el variador ha sido calibrado.

Dependiendo del propósito para el que se usen los variadores, el calibrado se puede realizar manualmente a través de un mando o, de forma automática a través de un programa (en este proyecto se usa la segunda forma).

3.5.2 Aplicación práctica. Calibración de los variadores y uso del controlador PWM.

Todo el código referente al control de los motores se ha implementado en el archivo */sensores/pwm.py.*

El código se ha dividido en una primera parte de declaración de variables. En primer lugar, se han declarado los puertos pwm que se van a usar entre las líneas 25 y 30.

A continuación, se crea una instancia pwm por cada variador conectado usando la siguiente estructura: `pwm_x = navio.pwm.PWM(ESC_INPUT_x).`
También se declaran como constantes los valores máximos y mínimos que pueden tomar los motores en las constantes SERVO_MIN y SERVO_MAX, estableciendo en las líneas siguientes que todos los motores comenzarán con el valor SERVO_MIN (motor apagado).

Respecto a los métodos, no se va a entrar en detalle en el primero de ellos ("init"). Cómo su nombre indica es una función de inicialización donde se habilitan los controladores PWM y la frecuencia de 50hz para cada uno de ellos.

Antes de explicar el resto de métodos, hay que entender cómo actúa el envío de un pulso PWM a un motor, este pulso al enviarse hace que gire el motor de forma momentánea, por lo que, si queremos que un motor permanezca funcionando durante un periodo largo de tiempo, es necesario repetir la llamada usando por ejemplo, un bucle.

La dificultad radica en que ese bucle se debe ejecutar de forma paralela en otro hilo distinto del principal, de modo que no se bloquee la ejecución y además se permita variar dentro de ese bucle los valores de velocidad.

Una vez sabemos cómo funciona el generador PWM, el primer método que encontramos es el método "_loop_" descrito en las líneas 56 a 62
Se trata de la función que se ejecutará paralelamente y que se ejecutará mientras el booleano *funcionando* permanezca a *true*. El uso de este booleano simplifica la tarea de parar la ejecución del controlador PWM al finalizar el vuelo, basta con poner esa variable declarada globalmente a *false*.

La siguiente función es la de *aplicarThrottle,* descrita entre las líneas 65 y 69.
Este método cambia los valores de las variables globales de los motores, haciendo que en la siguiente iteración del bucle del método *_loop_* se apliquen las nuevas velocidades de giro dado que por defecto se encuentran iniciadas a motor parado (SERVO_MIN) en las líneas 25 a 29.

A continuación, encontramos el método *conectarMotores,* cuya función es poner el booleano a *true* y dar comienzo a la ejecución del hilo paralelo declarado en la instrucción `thread=threading.Thread(target=self._loop_)` con el método `thread.start()`.

La siguiente función corresponde al calibrador de los variadores descrito entre las líneas 77 a 95. Si se ha entendido la explicación teórica, este método no presenta dificultad. Se establece la velocidad al punto más alto indicando que se conecte la batería, hace un *delay* de 10 segundos para que los variadores confirmen y a continuación establece la velocidad a cero repitiendo el proceso.

Por último, enciende los motores al 10% de su potencia uno por uno para que se pueda ver que, en efecto, la calibración ha sido exitosa antes de apagarlos y dejarlos en espera listos para su uso.

Por último la función *desconectarMotores* desconecta el controlador finalizando el bucle poniendo la variable *funcionando* a falso y parando el thread con el comando `self.thread.join()`.

3.5.3 Demostración de funcionamiento.

Dentro del archivo /videos/pwm_calibration.mp4 se puede ver la calibración usando el script descrito en el apartado anterior de los motores, mostrando los sonidos de confirmación durante la calibración, así como la demostración de su funcionamiento encendiéndolos uno por uno.

3.6 Sensor de navegación inercial.

3.6.1 Introducción teórica.

Un sensor de navegación inercial o IMU[7] es un dispositivo electrónico capaz de percibir cambios referentes a la orientación de éste además de proporcionar información acerca de su aceleración. En concreto una IMU de 9 ejes lleva integrada un acelerómetro, un giroscopio y un magnetómetro (28).

El propósito de usar este dispositivo es el de obtener tres valores necesarios a la hora de orientar un dispositivo en vuelo:

- Pitch: inclinación morro-cola (+90 grados, -90 grados).
- Roll: rotación morro-cola (+180 grados, -180 grados).
- Yaw: dirección en la que se orienta el morro (360 grados).

Estos valores se ubican dentro de un sistema de coordenadas NED[8]. Este sistema de coordenadas es estándar en aeronáutica y se define por tres ejes y un punto (29):

- Eje x o X_n: apunta al norte geodésico.
- Eje y o y_n: apunta al este geodésico.
- Eje z o z_n: apunta en sentido hacia la tierra (hacia abajo).
- Punto arbitrario O_n sobre la superficie terrestre, origen del sistema de coordenadas.

[7] Inertial Measurement unit.
[8] North, east, down

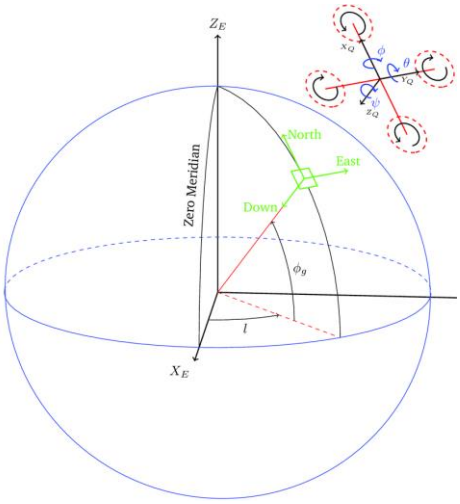

Fig 23. Sistema NED de coordenadas (30).

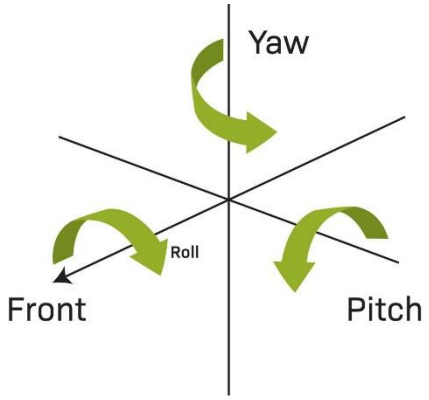

Fig 24. Diagrama de rotaciones pitch, roll, yaw (31).

Además, hay que especificar cuando se considera positivo o negativo el pitch y el roll. Por convenio la notación NED establece que las rotaciones son positivas en el sentido de las agujas del reloj en torno al eje de rotación (32).

La obtención del pitch y el roll en base a los datos de la IMU es directa usando el acelerómetro siempre y cuando la única aceleración sea la de la gravedad, si hubiera otras aceleraciones, estas deberían restarse a la aceleración total dictada por la IMU. Las ecuaciones son las siguientes en donde *acc* quiere decir aceleración en el eje correspondiente:

$$Pitch = \operatorname{atan}\frac{accY}{\sqrt{accX^2 + accZ^2}} \quad (33) \qquad Roll = \operatorname{atan}\frac{-accX}{\sqrt{accZ}} \quad (33)$$

Dependiendo del propósito para el que se usen las fórmulas hay que tener precaución con la segunda ecuación, dado que no es válida cuando las aceleraciones en el eje x y en el z valen cero (34).

El yaw por el contrario no se puede obtener valiéndose del acelerómetro, la razón es la siguiente. Si suponemos que el dron se encuentra en posición horizontal, y partiendo de la premisa de que el yaw es una rotación horizontal, esta rotación no va a ser percibida por el acelerómetro pues este va a mantener un valor constante de $9.8\ m/s^2$ en dirección a la tierra (z), para poder solventar este problema es necesario recurrir al magnetómetro que va a permitir saber la orientación del aparato en relación al norte geográfico, siempre respetando la norma de que la única aceleración es la gravedad.

Por lo tanto, recurriendo al uso del magnetómetro que en la fórmula se ha especificado como "mag" y utilizando los valores calculados en las fórmulas anteriores podemos conocer el Yaw (35).

$$Yaw = atan\frac{-magY*\cos(Roll)+magZ*\sin(Roll)}{magX*\cos(Pitch)+magY*\sin(Pitch)*\sin(Roll)+magZ*\sin(Pitch)*\cos(Roll)}$$

El principal obstáculo que se presenta ahora con el uso del magnetómetro es que este a diferencia del acelerómetro es fácilmente perturbable por campos magnéticos o metales cercanos al sensor, siendo estas distorsiones denominadas *hard iron* y *soft iron*.

Distorsiones hard iron.
Corresponden a distorsiones producidas por el hombre (campos magnéticos) e independientes al campo magnético terrestre, por ejemplo: un motor girando, un cable por el que pasa electricidad o el imán de un altavoz (36).

Distorsiones soft iron.
Son distorsiones producidas por elementos que por sí solos no generan un campo magnético pero que al integrarse con otros sí que los producen, por ejemplo, que haya una fuente de hierro o algún otro metal cerca (36).

De modo que para poder llevar a cabo mediciones precisas usando el magnetómetro con el fin de calcular el yaw, es necesario que este se calibre previamente, acción llevada a cabo en el siguiente apartado.

3.6.2 Aplicación práctica y calibración del magnetómetro.

Pese a que sería posible crear un programa que aplicara las fórmulas anteriores, Navio proporciona una versión de ese script mucho más eficiente que usa distintos filtrados para mejorar la precisión en la lectura del pitch, roll y el yaw.
Sin embargo, para que dicho código funcione se necesita, en el caso del magnetómetro, una serie de parámetros que compensen las distorsiones antes explicadas.

Para entender qué son estos tres parámetros, primero hay que explicar que las distorsiones producidas pueden interpretarse matemáticamente como una variación permanente capaz de ser corregida aplicando esa variación a las lecturas del magnetómetro. Este sesgo permanente es lo que se denomina en inglés cómo bias, dado que el magnetómetro tiene tres ejes es lógico que tengamos tres sesgos distintos (uno para cada eje), los denominaremos Bx, By y Bz, estos tres valores serán los que deberán incluirse en el código para que funcione la IMU (37).

Una vez que sabemos qué son estos tres valores, hay que calcularlos. Esta tarea puede llevarse a cabo a través del programa Magneto12 (38), para usarlo basta con realizar medidas aleatorias del magnetómetro girando el dron e introducir estos datos en el programa, es lógico por tanto que, cuantos más datos se recojan para luego introducirlos en el programa, mejor serán los datos del sesgo.

A continuación, se ha de comprobar que el magnetómetro se ha calibrado correctamente para lo cual lo más sencillo y eficaz es realizar medidas continuas del sensor en los planos xyz, mientras este se mueve durante dos minutos aproximadamente y mostrar los resultados, por ejemplo, en una nube de puntos donde cada punto es una medida del magnetómetro mientras este se mueve.

Dado que el magnetómetro es de tres ejes, este va a proporcionar tres valores (uno para cada eje), para que visualizar estos puntos sea más sencillo se ha transformado el gráfico a dos dimensiones dividiendo cada punto en coordenadas XY, XZ e YZ y se ha pintado el gráfico resultante usando la herramienta GNUplot.

En la figura 25 se muestran las mediciones del sensor sin ninguna calibración, pudiéndose ver el descalibrado en los tres planos, además mostrándose en el plano XZ una elipse en vez de una circunferencia. Esto provocará que si se realizan mediciones del yaw en algunos puntos del plano no se produzcan apenas variaciones, mientras que en otros se produzcan de forma errática y desmedida.

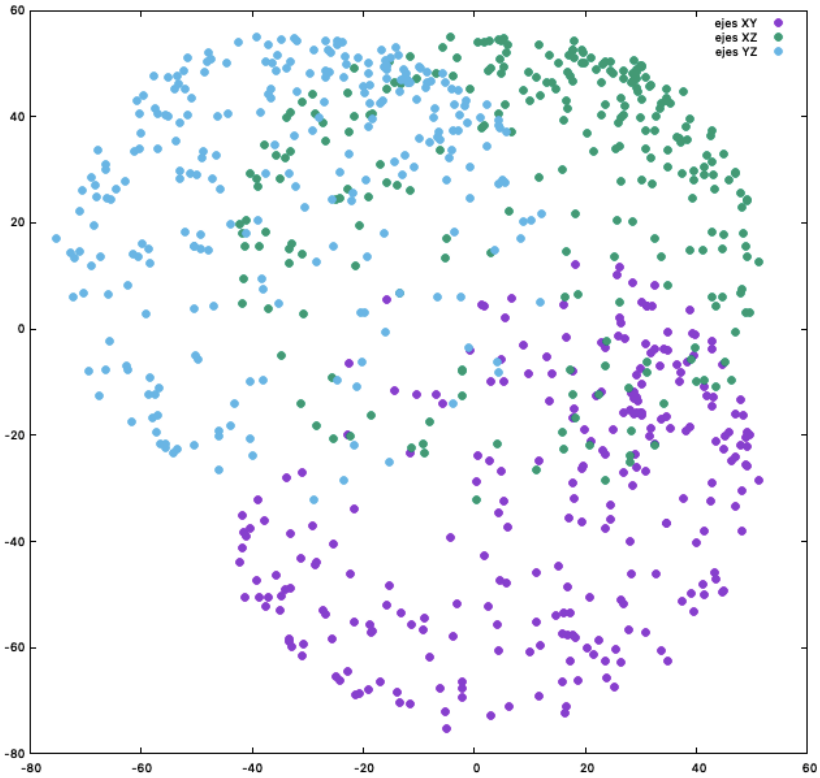

Fig 25. Mediciones del magnetómetro antes de la calibración.

Si ahora se observa la figura 26 que corresponde a nuevas medidas, pero esta vez con el magnetómetro calibrado, se obtienen tres círculos apenas desplazados y del mismo tamaño, lo que indica que el magnetómetro ha sido calibrado correctamente.

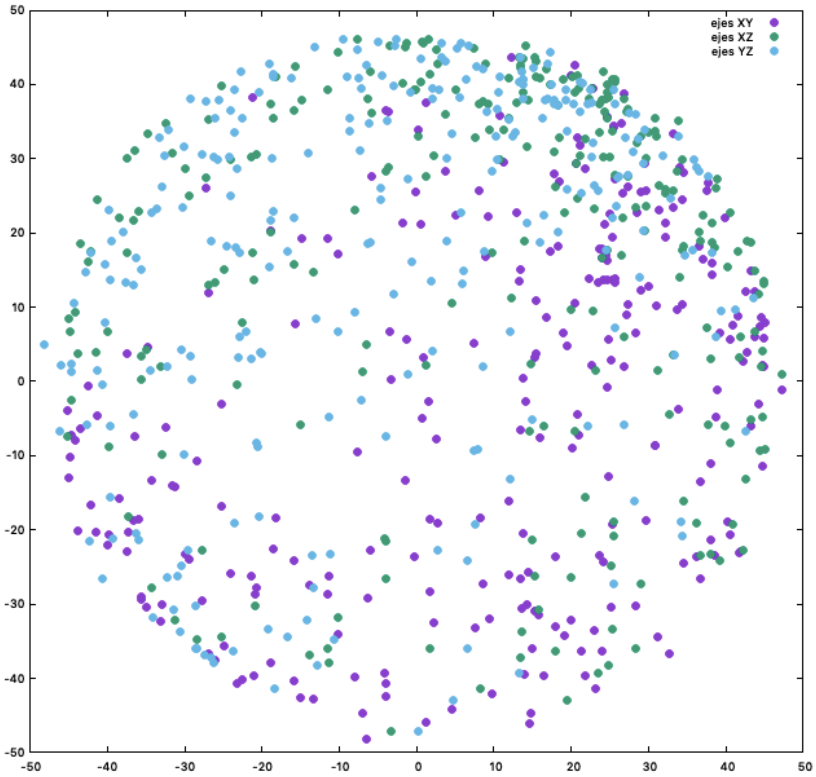

Fig 26. Mediciones del magnetómetro después de la calibración.

Tras esto, ya es posible obtener los valores correctos del Yaw usando el software de Navio.

3.6.3 Demostración de funcionamiento.

En este apartado se van a presentar una serie de videos donde se puede comprobar que, en efecto, es correcta la obtención de los valores Pitch y Roll. Se ha omitido el yaw dado que en el apartado anterior ya se demuestra su correcta calibración.

Para demostrar, que en efecto la obtención del pitch y el roll es correcta será necesario inclinar de diversas formas el cuadrotor y analizar los valores proporcionados por la IMU, hacer esto a mano queda descartado por la poca precisión que generaría el resultado. Es por esto por lo que se ha creado un programa para comprobar la veracidad de los datos calculados.

Este programa contenido en el archivo /sensores/calibraciónPitchRoll.py lo que hará será unido a un servomotor, y usando nuevamente canales PWM de la forma explicada anteriormente, inclinar el dron de forma controlada en ambos sentidos realizando mediciones de la IMU y mostrando por consola los valores.
Dado que en ocasiones el dron puede oscilar más de lo debido hasta estabilizarse, hay un tiempo de espera de 3 segundos entre cada cambio de orientación.

Fig 27. Cuadrotor inclinado -45 grados en Pitch.

El programa, se ejecuta vía terminal con un parámetro que especifica si estamos comprobando la calibración del Pitch o el Roll, esta comprobación se realiza entre las líneas 7 y 17, usando el método sys.argv para leer los parámetros especificados.
Tras esta parte, se inicializan los valores pwm por defecto igual que ocurría en el programa que controlaba los motores, esto ocurre entre las líneas 19 y 24. Por último se van realizando llamadas al método write(ángulo) que inclina el servo en el ángulo especificado.

Las figuras 28 y 29 corresponden a los resultados de dos calibraciones hechas por el programa creado, la primera corresponde al Pitch y la segunda al Roll. Se puede comprobar en ambos casos que la diferencia entre el ángulo esperado y el obtenido apenas varía nunca superando los 2,5 grados de diferencia.

Además, los videos correspondientes a las dos calibraciones mostradas se pueden encontrar en la carpeta videos/sensores con el nombre Pitch.mp4 y Roll.mp4.

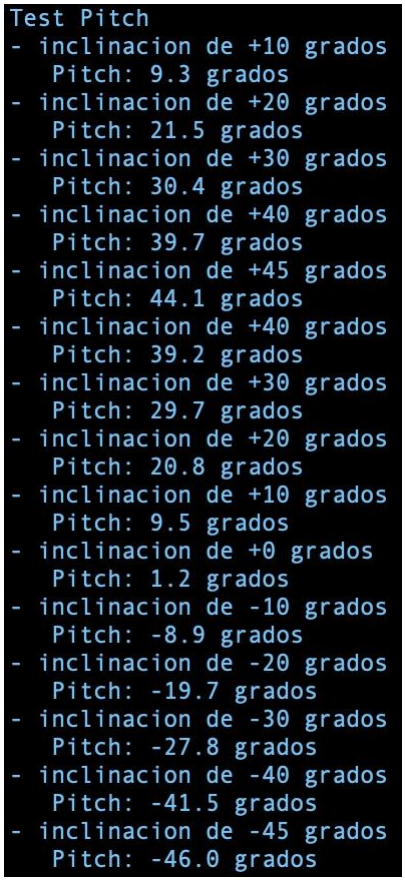

Fig 28. Resultado de calibración del Pitch.

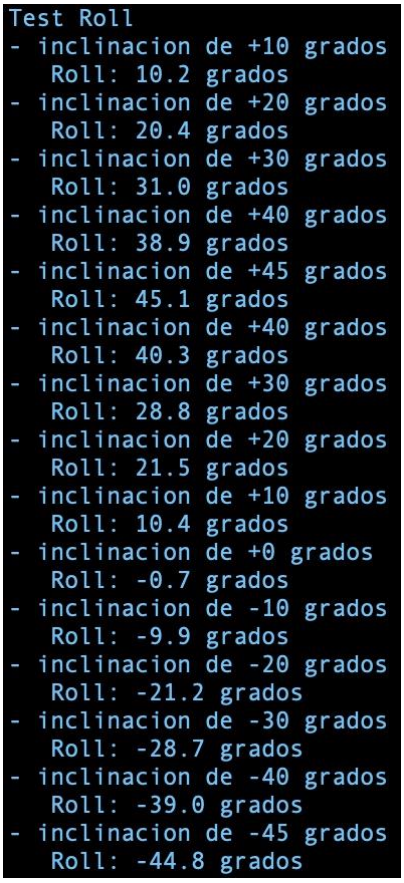

Fig 29. Resultado de calibración del Roll.

3.7 Esquema de trabajo.

En el esquema de la figura 30 se proporciona una posible forma de interconectar los sensores, para poder entenderlo es necesario introducir primero el término controlador. Un controlador puede ser definido como un programa que va a recibir una o varias entradas y en base a eso va a generar una salida.
Para este proyecto el controlador recibirá una serie de parámetros de entrada, ruta de vuelo, información del magnetómetro, de la inclinación o del rumbo, y en base a estos datos generará una salida que serán los pulsos PWM de los motores.

Aunque en este proyecto no se va a implementar dicho controlador para el sistema real, sino que se implementará para el modelo virtual, el siguiente esquema ejemplifica como estaría interconectado.

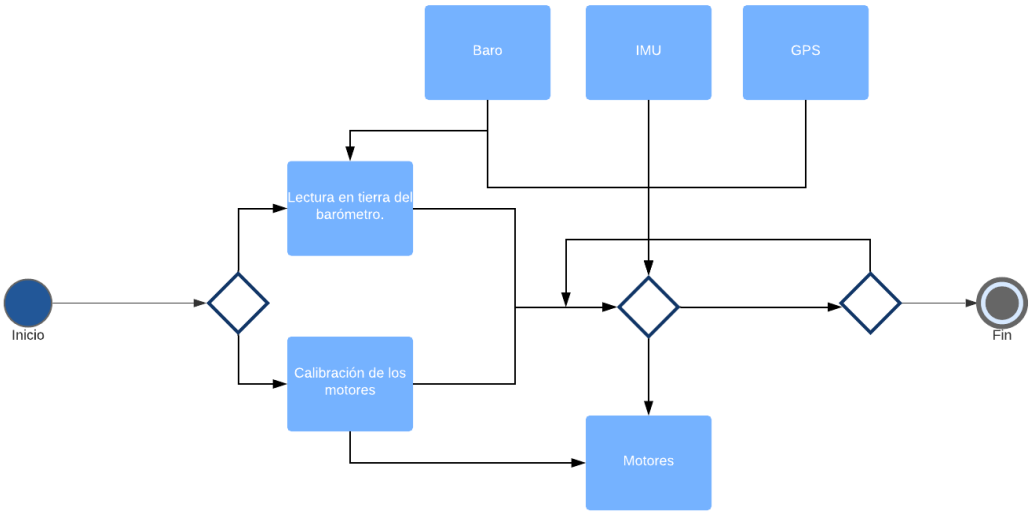

Fig 30. Esquema de funcionamiento.

Este esquema presenta una de las muchas formas en las que se podrían interconectar todos los sensores, los rombos corresponden a las decisiones tomadas por el controlador y se van a explicar a continuación de izquierda a derecha.

Las dos primeras decisiones nada más iniciar el aparato y antes de su despegue deben ser:

- Lectura en tierra del barómetro: tal y como se explicó en el desarrollo del sensor barométrico y en el correspondiente ejemplo, es necesario establecer la altura en tierra a nivel del mar para luego poder obtener la altura a nivel de suelo durante el vuelo.

- Calibración de los motores: ejecutar el script de calibración para que estos estén listos para poder volar igual que en el video de ejemplo.

Dado que ambas tareas son independientes unas de otras, sería interesante que estas se realizaran paralelamente para reducir el tiempo de esta primera parte.

A continuación, el dron debería de ser capaz de moverse, para lo cual necesita leer los sensores nuevamente y en base a eso especificar unas salidas para los cuatro motores, nuevamente la lectura de los sensores se puede realizar de forma concurrente para acelerar la respuesta del dron.

El siguiente paso del controlador será decidir si se ha finalizado con el recorrido, en caso de ser así terminaría su cometido y en caso contrario volvería al punto anterior repitiendo el mismo proceso.

Si bien en este apartado se ha explicado muy a groso modo el cometido del controlador y su funcionamiento, en el último capítulo se hace referencia a él y a todo su funcionamiento para su uso en el modelo virtual.

4. Método de comunicación y aplicación de programación de tareas.

4.1 Introducción.

En este capítulo se va a explicar todo lo referente a la aplicación que se usará para enviar los parámetros de funcionamiento al dron, así como la forma de comunicación con este.

La aplicación de nombre Pathfinder en honor a la sonda enviada a explorar Marte, ha sido creado desde cero en los lenguajes java y XML, usando Android Studio como entorno de desarrollo. El propósito de dicha aplicación es poder gestionar el recorrido del dron de forma íntegra, permitiendo la selección de un plan de vuelo incluyendo el recorrido, la altitud o el lugar de aterrizaje.
La conexión entre el dispositivo móvil y el prototipo se ha realizado mediante un modelo cliente-servidor a través de un protocolo TCP explicado más adelante.

Nota: la aplicación ha sido testeada en un móvil Xiaomi MI A2 con Android 10 como sistema operativo por lo que en versiones anteriores y o posteriores no se asegura el funcionamiento de esta, del mismo modo es necesario conceder los permisos de ubicación y almacenamiento a esta aplicación para poder usarla.

4.2 Protocolo de comunicación.

4.2.1 Modelo cliente servidor.

El modelo cliente servidor es una estructura de comunicación entre dos dispositivos (cliente y servidor) en donde, generalmente el servidor es un proceso activo que espera la conexión de un cliente que se conectará únicamente cuando éste necesite algo del servidor (39).

Dependiendo de las necesidades, es posible que el servidor sea capaz de trabajar de forma concurrente admitiendo varios clientes conectados de forma simultánea.

Cabe mencionar que el modelo cliente servidor proporciona la conexión entre los dispositivos, sin embargo, se necesita algún tipo de protocolo para poder mandar y recibir información de tal forma que esta sea legible por ambas partes, para esto tenemos varias alternativas (protocolos TCP, UDP, etc.) (39).

4.2.2 Protocolo TCP.

Una vez sentadas las bases de lo que es el modelo cliente servidor, necesitamos un protocolo de envío de mensajes entre ambos, el más usado en aplicaciones web es el protocolo TCP dado que garantiza la fiabilidad en cuanto a recepción de paquetes y tratamiento de errores, dicho protocolo funciona de la siguiente manera:

El emisor al enviar un paquete asocia a este un número de 32 bits que actúa como identificador (este valor no es aleatorio, corresponde al primer byte del segmento de datos), a cada paquete enviado el receptor envía un mensaje de confirmación o ack, que corresponde al primer byte que espera recibir (40). Hay dos características a tener en cuenta del ack:

- Es acumulativo: confirma el mensaje actual y todos los anteriores.
- Piggibacking: este término hace referencia a que el mensaje ack va solapado al mensaje de envío de datos.

El uso de este protocolo garantiza que, en el caso de pérdida de un paquete, el emisor lo sepa dado que no recibe un ack, pudiendo así reenviar el paquete y de la misma manera ocurre en el caso de envío de un paquete erróneo, dado que el ack no va a corresponder con el siguiente número al enviado (40).

4.2.3 Funcionamiento interno.

En el caso concreto del dron y el teléfono el primero actuará como servidor, ejecutando un bucle de espera hasta que un cliente (el teléfono) se conecte y transmita la información.

Únicamente se transmitirán dos mensajes, el primero por parte del teléfono que contendrá una línea del siguiente estilo: 6.3|40.34523,-3.43463|42.343243,-3.3532…
El primer valor corresponde a la altura que debe llevar el dron durante todo su recorrido, y el resto de los valores corresponden a pares de coordenadas por donde debe pasar el aparato.
El segundo mensaje corresponde a una confirmación de recibido (ack) que pondrá fin a la comunicación.

4.3 Estructura del Servidor.

4.3.1 Estructura interna.

Como se ha mencionado anteriormente, el servidor se aloja en la Raspberry. Todo el código de éste se encuentra en el archivo */sensores/serv_socket.py*.

Dicho archivo es un script dividido en dos funciones principales:

Función run: Es la función principal del script, se subdivide en los siguientes cometidos:

- Creación de variables para la conexión: El primer paso para poder crear una conexión TCP es declarar la ip del servidor (la de la Raspberry en la que se está ejecutando este script), así como el puerto de conexión y el tamaño máximo del buffer de envío. En este paso hay que tener cuidado pues es posible que, si la Raspberry no está configurada para que tenga una ip estática, ésta cambie cada

vez que se realiza la conexión a un punto wifi. En este caso como la Raspberry está configurada para que mantenga una ip estática, no es necesario comprobarlo, los datos que por tanto se van a usar son:
    - Ip del servidor(estática): 192.168.1.52.
    - Puerto del servidor: 12345.
    - Tamaño del buffer: 1024 bytes.

- El siguiente paso consiste en crear el socket del servidor con estos parámetros. Este paso se realiza entre las líneas 15 y 22.

  La línea más importante de este fragmento de código es la última, en concreto la función *s.accept()* que, como el propio nombre indica, es la encargada de aceptar la conexión de un cliente, además la función también tiene un papel bloqueante dado que si no hay ningún cliente a la espera de conexión, bloqueará la ejecución esperando hasta que haya algún cliente, esto permite que el dron al ejecutar este código pueda ejecutar una única vez el script y permanecer en modo de espera a que el cliente mande los datos mediante el teléfono móvil.

- Por último, el resto de líneas de la función no presentan dificultad, únicamente reciben el mensaje y lo decodifican con ayuda de la función *convertStr(msg),* eliminando caracteres no deseados.

La función convertStr: Es un método de conversión que decodifica el string recibido del cliente devolviendo la altura en formato decimal, y una tupla de decimales con los pares de coordenadas GPS ordenados por recorrido.

Aunque en este capítulo no se proporcione más información acerca de la conexión de este servidor con el resto del controlador, en el apartado seis se integrará con el modelo virtual para poder usar la aplicación y el simulador en conjunto usando este script como puente entre ambos.

4.4 Estructura de la aplicación del cliente.

4.4.1 Logo de la aplicación.

El logo se ha creado usando la herramienta *Freelogodesign*, donde el servicio permite el uso de los logotipos diseñados con fines personales o comerciales de forma gratuita (41).

4.4.2 Ventanas de la aplicación.

La aplicación la forman cinco ventanas:

La primera corresponde a la que se ve al ejecutar la aplicación (fig 31). Se trata de un menú donde se pueden seleccionar las opciones de iniciar vuelo, ajustes e información junto con el logo ampliado de la aplicación.

Si seleccionamos el botón de información se abrirá una nueva ventana meramente informativa con información acerca de la app (instrucciones, autoría, etc.) (fig 32).

En el caso de seleccionar ajustes, la interfaz mostrará una serie de parámetros configurables que se aplicarán durante el vuelo del dron o en la conexión con éste (fig 33), en concreto encontramos dos submenús, el primero subtitulado *Protocolo TCP* permite configurar dos parámetros relativos a la comunicación:

- Dirección IP: corresponde a la dirección ip asignada a la Raspberry dentro de la red.
- Puerto: identificador único del puerto del servidor de la Raspberry, por defecto usaremos el puerto 12345.

La razón principal de que estos dos parámetros sean configurables es que es posible que en ocasiones la ip estática configurada no pueda estar accesible por estar ocupada y se necesite establecer otra, de igual manera sucede con el puerto.

El segundo submenú referente a la navegación permite configurar los siguientes dos parámetros:

- Altura: la altura en metros a la que debe volar el aparato.
- Vuelta al origen: representa un switch que, en el caso de estar seleccionado indicará al dron que al finalizar el recorrido debe volver al punto del que partió y aterrizar ahí, en caso de no seleccionarse aterrizará en el último punto establecido por el usuario.

La tercera de las ventanas (fig 34) a la que se llega si se selecciona el botón *Iniciar vuelo*, abrirá una interfaz más compleja formada por un mapa en donde aparecerá la ubicación de nuestro aparato, así como dos botones *iniciar* y *reset*.
En esta ventana debemos seleccionar el recorrido que queremos que haga el aparato pulsando en el mapa, esto hará que se creen diferentes puntos (con un máximo de diez) que se recorrerán en orden de selección, el botón reset permite borrar todos los puntos y volver a crear el recorrido, y el botón iniciar nos lleva a la última ventana explicada a continuación.

La última ventana (fig 35) es meramente informativa, dado que, al iniciarse, gráficamente no realiza ninguna acción. El código de ésta en cambio es muy importante, pues realizará la comunicación inalámbrica enviando todos los datos necesarios y usando la dirección ip y el puerto de los ajustes para dicha acción.

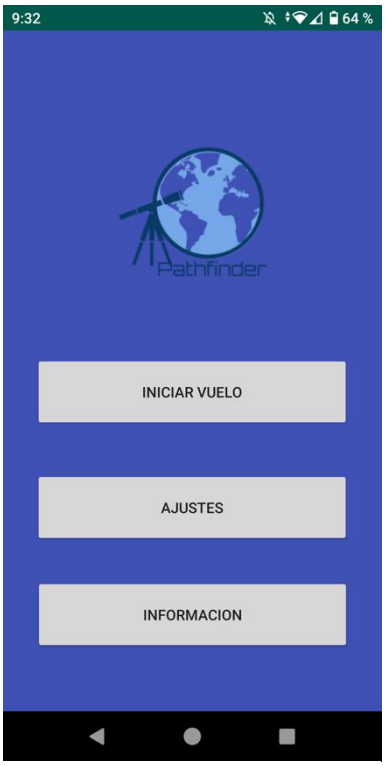

Fig 31. Pantalla principal de la aplicación.

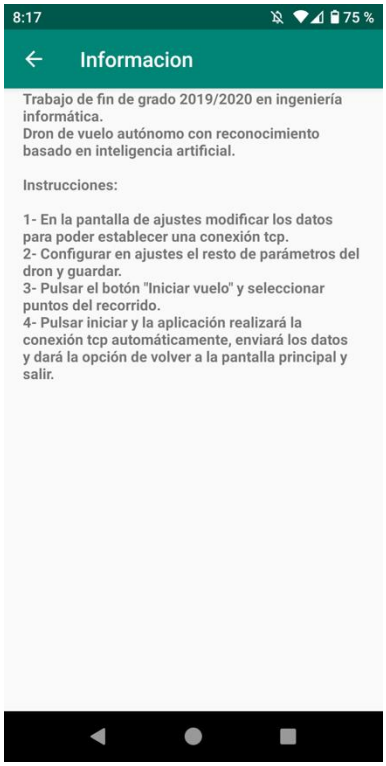

Fig 32. Pantalla de información.

Fig 33. Pantalla de ajustes.

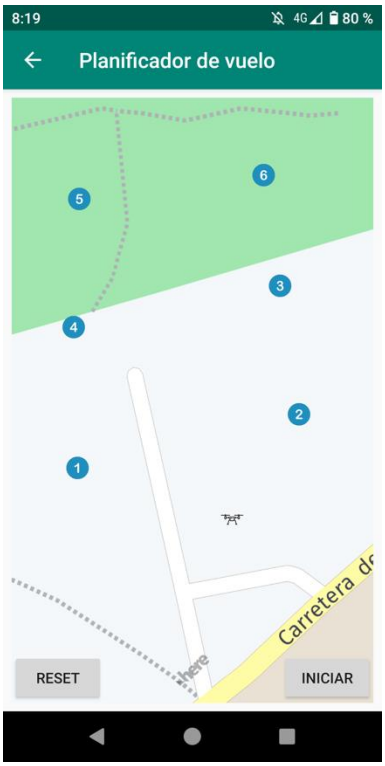

Fig 34. Pantalla de creación de recorrido

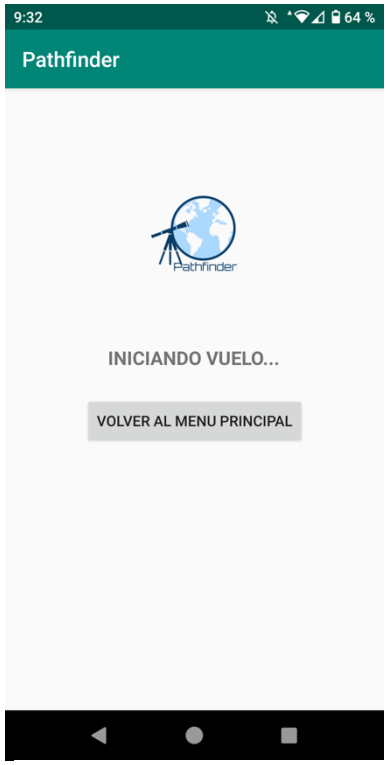

Fig 35. Pantalla informátiva final.

4.4.3 Estructura interna.

Archivos .java

Cada una de las clases java corresponden al controlador de cada una de las vistas de la aplicación. De modo que, habiendo 5 vistas, hay 5 clases que serán explicadas a continuación.

Como aclaración previa, cada uno de los archivos lleva el método *onCreate(Bundle savedInstanceState),* es un método obligatorio que se ejecuta nada más crearse la vista de la clase.

- AjustesActivity.java

El método principal de esta clase es *btnGuardar* ejecutado cuando se pulse dicho botón en la vista, el código primero crea instancias de todos los campos de la ventana entre las líneas 20 y 23.

A continuación, se crea un objeto del tipo *Intent* en la línea 28 usado para almacenar parámetros y poder enviarlos a otras actividades (en este caso los valores de los ajustes).

Por último, usando el método *intent.putExtra* podemos almacenar la información que se enviará al cambiar de actividad de la forma descrita entre las líneas 27 y 31:

- InfoActivity.java

Su única tarea es, al crearse, llamar a la función *setText* que abrirá el archivo *info.txt* alojado en la memoria del teléfono y cargar en la vista el texto que contiene ese archivo, usando para ello un objeto *inputStream* y un *buffer* donde almacenar el texto cargado.

- MainActivity.java

Es la clase principal llamada al inicio de la aplicación, está formada por varios métodos que actúan como *listeners* de los botones de la interfaz.
Es la encargada de pasar los parámetros de configuración de vuelo y TCP a la siguiente clase teniendo en cuenta que los parámetros, por defecto, han podido ser modificados. Esta acción se realiza recogiendo los datos enviados por los ajustes entre las líneas 22 y 33

En donde si el *Intent ajustes* mencionado anteriormente es nulo, significa que no hemos cambiado ningún ajuste y podemos mantener los que teníamos por defecto, en caso contrario, usando el método *get*, podemos leer los datos nuevos y sobrescribirlos.

Los métodos no se van a explicar en profundidad, dado que como se ha mencionado son meros *listeners* que al llamar al método *startActivity(),* crean una nueva vista pasando si es necesario la información de los ajustes.

• VueloActivity_Map.java
Esta clase es la encargada de crear el mapa usado para el recorrido, así como de tratar las pulsaciones en dicho mapa, por lo que es la clase más compleja dentro del dominio de la aplicación.

El primer proceso llevado a cabo en la clase tras la declaración de variable es dentro del método *onCreate* entre las líneas 76 y 85, se trata de la creación de un fragmento que sea capaz de leer la ubicación actual del teléfono para centrar el mapa en dicho punto.

El código para realizar la acción comienza con unas comprobaciones obligadas por Android para comprobar que se han concedido los permisos necesarios a la app, a continuación, leemos la última ubicación conocida con el método *getLastKnowLocation* y actualizamos la ubicación.

Una vez conocida la ubicación, creamos el mapa entre las líneas 88 y 111.
En este fragmento se crea además del mapa, un *listener,* para que cada vez que se pulse la pantalla, se establezca un punto y se añadan las coordenadas a una lista. También se configuran otros parámetros como el zoom o el icono del dron que aparece en la posición actual de este.

Una vez explicado el proceso de creado del mapa hay varias funciones de las que solo se mencionarán las más importantes:

La función *ini* encargada de inicializar las listas de coordenadas. Se llama al crear la clase, y cada vez que borremos los puntos para reinicializar el recorrido.

La siguiente función es *setDroneMarker*, que es la encargada de coger el archivo con la imagen del icono del dron "*dron.png"* y establecerlo en el mapa.

La función *actualizarMarker* es parecida a la anterior, sin embargo, en vez de establecer un icono, lleva asociada un contador para saber cuántos puntos llevamos colocados y así establecer el icono correcto. Además, esta función debe ser capaz de transformar el punto seleccionado a coordenadas, esto se realiza entre la línea 164 y la 168:

La última función que se va a explicar es *onTapEvent(pointF),* que implementa el *listener* del principio (cuando se pulsa en el mapa) en las líneas 221 y 222.

Como dicen los comentarios, la primera línea añade el punto seleccionado a una lista para añadirle el icono y la segunda línea es vital pues, aunque se haya añadido a la vista un nuevo punto, es necesario actualizar toda la vista para poder percibir los cambios.

• VueloActivity_Msg.java
Esta última clase tiene el cometido de crear el cliente TCP y enviar los datos recogidos por la app. Para realizar dicha tarea, en primer lugar, en la función *onCreate* se recogen todos los datos y se almacenan en variables. Este proceso no presenta dificultad y ya ha sido explicado en apartados anteriores.

A continuación, hay definido el método *btnSalir* que se ejecuta al pulsar el botón de salida y nos lleva al menú principal.

El siguiente método *CrearMSG* es el encargado de codificar, en un solo string, toda la información pertinente para enviarla devolviendo dicho string.

Dentro de esta clase además se encuentra la clase *backGroundTask,* que es la clase que creará el cliente dentro de su único método *doInBackground*. Pese a que la creación del cliente no es difícil y es fácilmente entendible viendo el código, hay que tener en cuenta la necesidad de usar esta clase para crearlo, dado que si no se usara se produciría un bloqueo en la app. Es por esto por lo que Android obliga al uso de esta clase que creará un hilo nuevo para ejecutar el cliente.

Archivos.xml

Respecto a los archivos xml podemos distinguirlos en dos tipos: los que contienen información acerca del funcionamiento de la app denominados *manifest,* y los referentes al estilo de las vistas, llamados *layout.*
En esta memoria se va a omitir la explicación de los archivos que hacen referencia al estilo de las ventanas pues su estructura es similar, una lista de elementos que conforman la vista (botones, cuadros de texto, y demás elementos visuales).

Por poner un ejemplo de uno de los elementos que puede contener una vista, se va a explicar el fragmento de código referente a uno de los botones, dichas líneas se pueden encontrar en el archivo TFG/android_app/archivos_xml/activity_main.xml entre las líneas 27 y 42.

Como se puede ver, el código es trivial, el botón lleva un id único asociado seguido por varios parámetros para ajustar su tamaño y posición, así como la etiqueta que aparecerá centrada al dibujar el botón y la función que se llamará al pulsar que será *btnAjustes*.

Por último, hay que mencionar que, aunque no se expliquen los archivos xml referentes al estilo, si se quieren leer están en el repositorio de Github.

Fuera de lo que es el estilo hay un archivo que juega un papel muy relevante a la hora del funcionamiento de una app en Android, el *Manifest/AndroidManifest.xml*

Es un archivo que proporciona al teléfono información acerca de la aplicación (42). Dentro de este archivo se debe incluir lo siguiente:

- Token para poder usar el sdk que proporciona Here, se puede encontrar entre las líneas 23 y 28.

- Permisos: funciones sensibles que la aplicación realizará. En este caso concreto necesitamos acceso a la ubicación, internet y almacenamiento. La sintaxis para reclamar estos permisos tiene una estructura similar a la descrita a continuación:

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

  En este ejemplo estamos reclamando permiso de acceso a la ubicación del teléfono, en el archivo del repositorio estos permisos están incluidos entre las líneas 5 y 10.

- Actividades: cada una de las vistas existentes llevan asociada una etiqueta en este archivo con información acerca de diversos aspectos (actividad padre, nombre, etc.).
  Por ejemplo, la actividad descrita entre las líneas 30 y 37 corresponde a la de la ventana de información.

4.5 Información adicional.

En este capítulo se han omitido las pruebas y demostraciones de la aplicación puesto que estas se realizarán conjuntas con el simulador para dar una visión más global del flujo de funcionamiento.

5. Integración del algoritmo de reconocimiento.

5.1 Introducción.

La siguiente función que se implementará es la de reconocimiento, el que se puedan localizar los objetivos buscados de una forma rápida y efectiva. Para dicha tarea hay diversas alternativas, por ejemplo, el envío de video en tiempo real o el análisis de éste una vez que el aparato haya regresado del recorrido establecido, sin embargo, dado que buscamos que el conjunto de tareas tenga la menor interacción humana lo lógico es que el propio ordenador de abordo sea capaz de realizar la tarea mencionada.
Esto es posible usando algoritmos de inteligencia artificial, que son capaces de detectar patrones durante el análisis de video y por tanto reconocer objetivos con una determinada certeza, en concreto en este proyecto se usa una técnica de Deep learning llamada SSD explicada más adelante.

De igual manera que en el apartado de los sensores, el reconocedor no se va a usar en nuestro simulador virtual, en lugar de eso tras explicar la programación y funcionamiento de este, se va a verificar su uso en una serie de imágenes que posteriormente se analizarán.

5.2 Contexto teórico.

5.2.1 Clasificación vs detección.

Dentro del paradigma de la identificación de objetos tenemos dos alternativas, la clasificación y la detección. El primero, como su nombre indica, permite dada una imagen saber qué contiene, por ejemplo, si diseñamos un clasificador de aves nuestro algoritmo debería ser capaz de dada una imagen decirnos la especie del ave.
En cambio, en un algoritmo de detección no solo buscamos clasificar una imagen según el tipo de esta o según algún parámetro, sino además saber su posición en la imagen acotándola normalmente con algún recuadro (43). Estos dos conceptos se entienden mejor en la imagen inferior correspondiente a la figura 36.

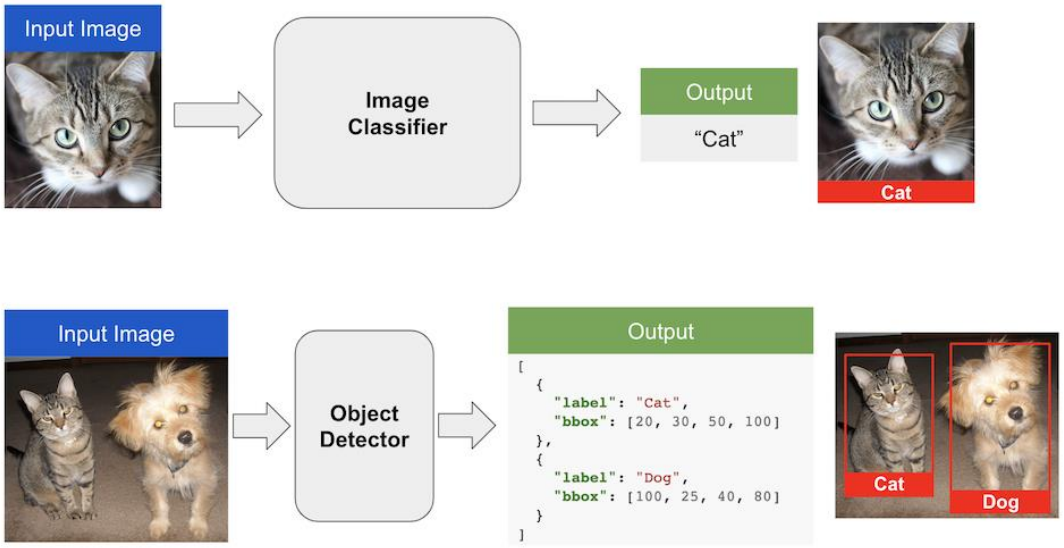

Fig 36. Diferencias entre clasificación y detección (44).

5.2.2 SSD – Single Shot Detector.

Es un algoritmo de Deep learning, considerado uno de los más rápidos que para poder realizar la detección de elementos en una imagen, combina la detección de regiones de interés mediante una red RPN[9] y de forma simultánea la clasificación de estas regiones para saber a qué corresponden (45).

Este algoritmo realiza una descomposición de la imagen usada como entrada en distintas partes de distintos tamaños, de modo que, a cada sub-imagen obtenida, se le aplique un algoritmo de clasificación y si se ha obtenido un resultado concluyente, se guardará la posición de la sub-imagen para luego añadirle un recuadro con el nombre asociado a la clase detectada, así como su porcentaje de certeza (45).

[9] Region Proposal Network

5.3 Tensorflow API.

5.3.1 Comparación de modelos pre-entrenados.

Tensorflow proporciona varias redes SSD entrenadas y validadas con COCO, que es un dataset ampliamente usado en problemas de detección de objetos por su gran tamaño y sus 91 categorías que incluyen desde personas a animales o toda clase de objetos, además proporciona por cada imagen múltiples objetos con lo que su eficacia es notable (46).

Estos modelos listados en la figura 37 se encuentran alojados en el repositorio de Tensorflow subdivididos en velocidad(ms) y eficacia de reconocimiento(mAP[10]), para el uso en aparatos con poca potencia recomienda el uso de los modelos especificados como "mobilenet", que son modelos cuyos requisitos de ejecución son notablemente más bajos en relación al resto (47).

Model name	Speed (ms)	COCO mAP[^1]
ssd_mobilenet_v1_coco	30	21
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18
ssd_mobilenet_v1_quantized_coco ☆	29	18
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16
ssd_mobilenet_v1_ppn_coco ☆	26	20
ssd_mobilenet_v1_fpn_coco ☆	56	32
ssd_resnet_50_fpn_coco ☆	76	35
ssd_mobilenet_v2_coco	31	22
ssd_mobilenet_v2_quantized_coco	29	22
ssdlite_mobilenet_v2_coco	27	22
ssd_inception_v2_coco	42	24

Fig 37. Modelos ssd mobilenet proporcionados por tensorFlow (47).

Para este caso en particular se ha elegido el modelo *ssdlite_mobilenet_v2_coco,* que es uno de los modelos más modernos proporcionados con una buena relación entre eficacia y tiempo de detección.

5.3.2 Transfer learning.

Pese a la eficacia del modelo SSD elegido en el apartado anterior, este a pesar de estar preparado para reconocer personas, no estaba diseñado para reconocerlas a media y larga distancia y eso podría generar un problema grave dado que el dron reconoce a estos objetivos a varios metros y normalmente vistos desde arriba.

Para solventar este problema se ha llevado a cabo el uso de transfer learning. Este concepto consiste en coger un modelo pre-entrenado y basándose en él, añadirle nuevas imágenes para que entrene con ellas y sea capaz de reconocer nuevas categorías.

[10] mAP: mean average Precision.

Si bien es posible entrenar una red desde cero, este método ofrece grandes ventajas, la primera es el coste de tiempo, entrenar un modelo completamente nuevo puede llevar varios días mientras que usando este método ese tiempo se reduce considerablemente, el otro factor clave es que este método permite conservar el entrenamiento de las diferentes capas y de esta manera conservar su estructura y eficacia (48).

Estos factores son especialmente interesantes para una red tan exhaustivamente entrenada como la elegida *(ssdlite_mobilenet_v2_coco).*

Para realizar este método el primer paso es recopilar una serie de imágenes de personas vistas desde arriba y desde lejos, esta tarea debido a la gran cantidad de datasets es sencilla, se han elegido dos:

- Okutama Action: consiste en una serie de videos grabados desde un dron con personas realizando varias acciones (50).
- Stanford Drone Dataset: nuevamente son varios videos de personas vistas a distintas alturas caminando o de pie (51).

Según se especifica en la documentación de TensorFlow con entre 50 y 100 imágenes por categoría son suficientes para emplear esta técnica, se han seleccionado 80 capturas de pantalla (30 de cada dataset) junto con otras veinte obtenidas de internet para proporcionar cierta variedad.

El siguiente paso es “dibujar” la zona en la que se encuentra el objetivo (la persona en este caso), para esta tarea se puede recurrir al programa LabelImg (49)que mediante una sencilla interfaz permite marcar los objetivos que se buscan reconocer tal y como se muestra en la figura 38.

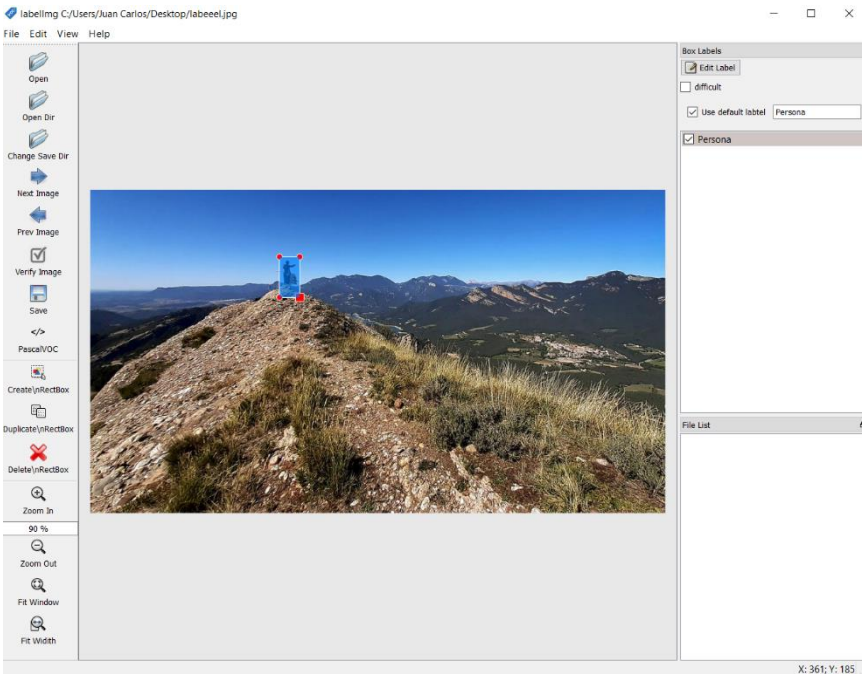

Fig 38. Captura de pantalla del programa LabelImg.

Una vez hecho esto con todas las imágenes y usando la api de TensorFlow se procede a entrenar el modelo, el proceso es plenamente automático a excepción de la parada, la api de TensorFlow no proporciona un mecanismo de parada y este ha de hacerse manualmente. Para ello es necesario introducir el término de pérdida o como se conoce en inglés loss, este término hace referencia al error que se produce a la hora de verificar las predicciones hechas por el algoritmo, es consecuente por tanto que este error debe ser decreciente, comúnmente a esto se le llama función de perdida (52). Según marca TensorFlow el valor de loss debe rondar un 2% para que se considere que las predicciones del algoritmo son correctas, en este caso se ha tardado aproximadamente seis horas en llegar a dicho valor como se muestra en la figura inferior.

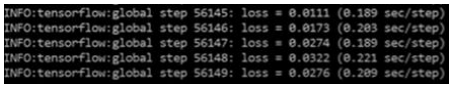

Fig 39. Captura de pantalla de la consola durante el entrenamiento.

Tras esto solo resta exportar el modelo y usarlo igual que si fuera uno descargado de los repositorios de TensorFlow.
En el punto 5.6 se realizará un análisis del funcionamiento de este modelo mostrando varias imágenes del resultado.

5.4 Esquema de ejecución.

El funcionamiento del reconocedor se puede subdividir en los siguientes pasos:

1- Inicialización y carga de parámetros:
2- Ejecución en bucle del algoritmo:
    a. Captura de un fotograma de la webcam.
    b. Análisis de ese fotograma usando el algoritmo ssd.
    c. Comprobación de requisitos de detección.
        i. Requisitos de umbral.
        ii. Requisitos geográficos.
    d. Guardado del fotograma: en el caso de que los requisitos anteriores se cumplan la imagen es considerada válida y un posible resultado positivo.

Los requisitos de umbral hacen referencia a la certeza con la que la red detecta un objetivo. En un principio se ha configurado un nivel de certeza del 75%, pero esto es fácilmente configurable dentro del archivo *reconocedor.py* modificando la variable umbralCaptura.

Los requisitos geográficos a los que se refiere el esquema especifican que solo se guardarán imágenes distanciadas entre sí ocho metros como mínimo, la razón es que, si el dron detecta un objetivo y se acerca a él, dependiendo de la velocidad con la que el algoritmo procese cada fotograma, este podría guardar cientos de imágenes muy similares.

Por último, es importante destacar que, pese a que el bucle es infinito, el algoritmo no se ejecutará siempre, dado que intentamos el mayor paralelismo posible, esta fase se realizará en un hilo separado del resto de la ejecución por lo que cuando no se necesite usar el reconocedor (finalización del recorrido), el coordinador parará el hilo finalizando de esta manera la ejecución del algoritmo.

5.5 Funcionamiento interno.

La estructura de carpetas que se encuentra en el repositorio es la misma que debe haber dentro del dron para poder funcionar correctamente la API. Se subdivide en:

- Capturas: carpeta dende se guardan las imágenes durante un recorrido.
- Modelos: carpeta donde se almacenan los modelos.
- TestVideos: carpeta con videos usados para testear el funcionamiento del algoritmo.
- Labels: contiene el archivo "label.pbtxt" con la clase que buscamos reconocer.
- Utils: contiene todas las librerías y archivos que la api necesita para funcionar.

La implementación del algoritmo se ha realizado íntegramente en el archivo *reconocedor.py*.

En dicho archivo se han establecido diferentes apartados, el primero de ellos corresponde a la declaración de los parámetros de funcionamiento explicados en el apartado anterior que se encuentran entre las líneas 26 y 30.

A continuación, se especifica la entrada de datos por cámara en la línea 33, como información adicional en la línea siguiente se puede seleccionar la entrada a través de un video, esta entrada solo tiene interés de depuración por lo que se podría suprimir si se quisiera.

También se da la opción de usar varios modelos, ya sea entrenado por un usuario o uno de los existentes en los repositorios de TensorFlow, para este proceso habría que guardar el modelo en la carpeta “modelos” y añadir una nueva línea con su nombre, como ya se ha mencionado, se va a usar el modelo *ssd_inception_v1* con transfer learning.

A continuación, se especifican varias rutas a distintos directorios (directorio de capturas y librerías) y se configura el reconocedor, finalizando en un bucle infinito en donde se ha modificado el código original de Tensorflow para que trabaje con videos en lugar de imágenes, esto es posible añadiendo la línea `ret, image_np = inputVideo.read()` que transforma el video en distintos fotogramas.

Por último, una de las modificaciones respecto al código original de la API ha sido el añadir el guardado de las imágenes que son válidas para nuestro programa. Esto se realiza en el fragmento comprendido entre las líneas 90 y 101.

En este fragmento se realiza una comprobación del vector de puntuaciones (scores) y se comprueba que la puntuación obtenida es mayor o igual al de nuestro umbral. A continuación, hay otra condición que comprueba que la distancia en metros entre la anterior captura y esta sea mayor a la especificada, para realizar esta acción se utiliza el método distanciaGPS explicado al final.

En el caso de que ambas condiciones se satisfagan, guardamos las coordenadas actuales del dron y mediante el método *putText* las adjuntamos en la parte inferior izquierda.
Por último, se guarda la captura dándole de nombre la fecha y hora actuales usando el método *imwrite*.

Por último, en la línea 104 se declara la función distanciaGPS antes mencionada, el funcionamiento de esta función se basa en la aplicación de la fórmula de Haversine que permite dados dos puntos, calcular su distancia teniendo en cuenta y compensando la curvatura terrestre (53).

$$Distancia = 2 * R * asin\sqrt{sin^2\left(\frac{\Delta lat}{2}\right) + \cos(lat1) * \cos(lat2) * sin^2\left(\frac{\Delta lon}{2}\right)}$$

Lat → latitud

Lon → Longitud

(lat1, lon1) → Latitud y longitud en el punto 1

(lat2, lon2) → Latitud y longitud en el punto 2

Δlat = lat2 – lat1

Δlon = lon2 – lon1

R = 6372.795477598 Km (Radio de la Tierra)

Fig 40. Fórmula de Haversine (54).

5.6 Análisis de los resultados.

A continuación, se van a presentar tres imágenes donde se puede ver la salida del reconocedor re-entrenado.
Las dos primeras imágenes no incluyen pie de foto con las coordenadas dado que han sido sacadas en internet y no captadas por la cámara en vivo.

Fig 41. Imagen 1 de prueba sin procesado (55).

Fig 42. Imagen 1 de prueba procesada.

Fig 43. Imagen 2 de prueba sin procesado (55).

Fig 44. Imagen 2 de prueba con procesado.

Tras examinar estas dos imágenes se puede señalar que el algoritmo en estos casos es capaz de identificar personas a una distancia considerable. Tal y como se esperaba, el algoritmo ha tenido un mayor éxito al identificar al excursionista en la figura 42 dado que se encuentra en un entorno parecido al usado para re-entrenar el modelo. La segunda imagen en cambio, aunque identifica correctamente al objetivo a una distancia mayor la hace con una certeza inferior (54%) debido en mi opinión a que el algoritmo no ha sido entrenado con imágenes con nieve y esto puede interferir además de la gran altura en la capacidad para detectar personas.

Por último, la figura 45 se ha tomado colocando el microcontrolador junto con la cámara en un dron teledirigido en el parque del retiro de Madrid para poder dar también una imagen del resultado aplicando el pie de foto con las coordenadas GPS proporcionadas por el sensor además de la fecha y hora de la captura.

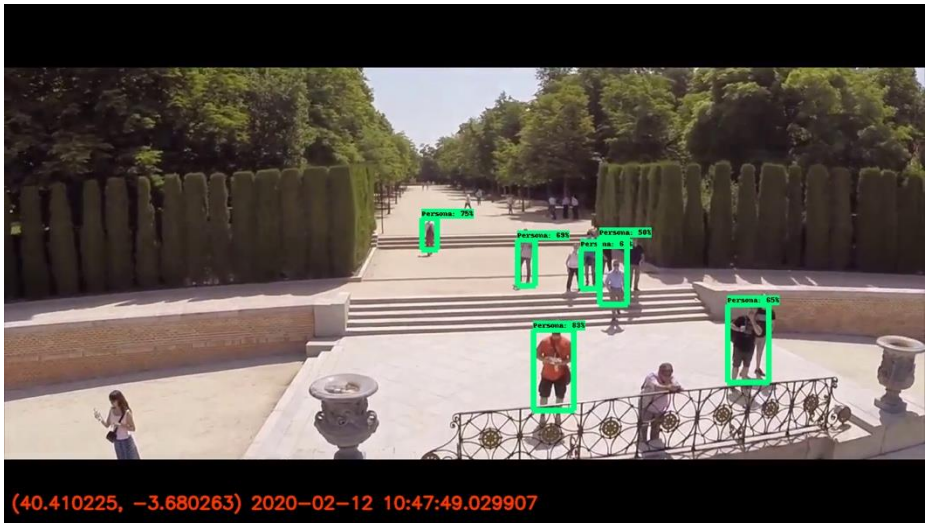

Fig 45. Ejemplo de captura proporcionada por el algoritmo.

6. desarrollo de un simulador.

6.1 Introducción.

En este apartado a modo de finalización del trabajo se va a desarrollar un entorno virtual en Python usando entre otros el módulo Scipy [11] con el que poder simular numéricamente el funcionamiento de un dron con características similares al sistema real descrito en esta memoria utilizando parámetros físicos fieles a los reales.
Posteriormente se unirá a ese simulador una interfaz y se la conectará a la aplicación Android desarrollada para dar una visión más general todo el proceso.

6.2 Representación espacial.

El primer paso lógico previo desarrollo de un sistema complejo de control o el manejo de señales es crear una representación espacial en 3D del dron clara, que permita ver su evolución durante el vuelo.

Durante el desarrollo del modelo y del control se van a emplear siempre coordenadas NED, ya que constituyen una forma de representación estándar en aeronáutica, este sistema se recuerda también fue usado en el desarrollo del modelo inercial del apartado 3.6.

Basándose en esta definición, la representación de los ejes xyz quedará de la siguiente manera:

- Eje x: orientado en dirección del norte geográfico.
- Eje y: orientado al este geográfico.
- Eje z: apuntando en la dirección del centro de la tierra.

A partir de esta definición, se van a definir dos sistemas de coordenadas distintos:

- Sistema de coordenadas tierra: sistema NED con centro en un punto fijo de la superficie de la tierra.
- Sistema de coordenadas cuerpo: sistema de coordenadas unido al dron

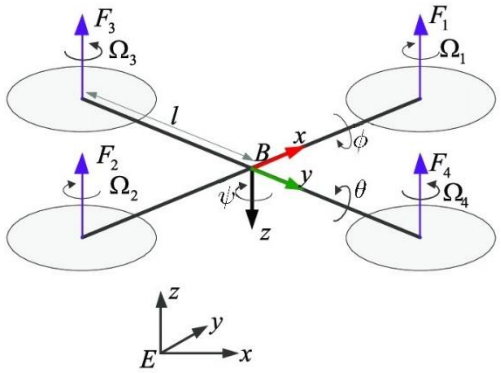

Fig 46. Esquema de un dron (57).

[11] Biblioteca de código abierto para python con herramientas matemáticas, científicas y de ingeniería (58).

La figura 46 muestra los tres ejes cuerpo, b_1(rojo), b_2(verde) y b_3(negro). Estos ejes se pueden representar como vectores en ejes cuerpo, en donde las coordenadas serán siempre:

$$b_{1B} = \begin{bmatrix}1\\0\\0\end{bmatrix}; b_{2B} = \begin{bmatrix}0\\1\\0\end{bmatrix}; b_{3B} = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

Dado que estos ejes giran y se mueven con el cuerpo del dron, es necesario representarlos en base tierra, tomando en general coordenadas del tipo:

$$b_{1T} = \begin{bmatrix}b_{1Tx}\\b_{1Ty}\\b_{1Tz}\end{bmatrix}; b_{2T} = \begin{bmatrix}b_{2Tx}\\b_{2Ty}\\b_{2Tz}\end{bmatrix}; b_{3T} = \begin{bmatrix}b_{3Tx}\\b_{3Ty}\\b_{3Tz}\end{bmatrix}$$

Sin embargo, estas coordenadas cambian continuamente y es necesario establecer una relación entre las coordenadas tierra y cuerpo.
Para obtener una notación más simplificada, se omitirá la T al referirse a la representación de los ejes cuerpo visto desde el sistema tierra, es decir b_{iT} se representará como b_i. Si ahora se construye una matriz con los vectores b_i como columnas, se obtendrá una matriz de cambio de coordenadas de ejes cuerpo a ejes tierra (una matriz de rotación *R*).

$$R = [b_1\ b_2\ b_3] = \begin{bmatrix}b_{1x} & b_{2x} & b_{3x}\\b_{1y} & b_{2y} & b_{3y}\\b_{1z} & b_{2z} & b_{3z}\end{bmatrix}$$

Por lo tanto, si se conocen las coordenadas de un vector en el sistema cuerpo de la forma (v_x, v_y, v_z), las coordenadas de ese mismo vector en coordenadas tierra serán:

$$\begin{bmatrix}w_x\\w_y\\w_z\end{bmatrix} = R\begin{bmatrix}v_x\\v_y\\v_z\end{bmatrix}$$

6.4 Dinámica.

La dinámica hace referencia a los cambios de un sistema a lo largo del tiempo. En este caso la dinámica del dron vendrá dada por la posición y orientación de este, así como por sus velocidades lineales y angulares.
En primer lugar, el dron siempre estará sometido a la gravedad *g*. En coordenadas NED y ejes tierra se trata de una fuerza que apunta en la dirección del eje z, si ahora llamamos $e_3 = [0,0,1]$ al vector en dicha dirección podemos representar la fuerza de la gravedad como mge_3 donde m representa la masa del dron.
Si ahora se encienden los motores del dron, se generará una fuerza negativa en el eje z en coordenadas cuerpo, se puede por tanto representar la fuerza total como:

$$f = -uRe_3$$

Por último, el dron al avanzar genera una resistencia al avance debida al rozamiento, si se representa la posición del dron por el vector *r*, entonces la resistencia se representa como:

$$f_d = -b \cdot \dot{r}$$

Donde *b* es el coeficiente de rozamiento.
Las ecuaciones diferenciales resultantes, aplicando la segunda ley de Newton, son:

$$m\ddot{r} = mge_3 - uRe_3 - b \cdot \dot{r}$$

El paso siguiente es realizar el modelado de rotación del dron. Para ello, se parte de la matriz R. Si el dron gira, R cambia en función del tiempo de giro, en otras palabras $R = R(t)$. Si ahora se parte de la relación $R^T \cdot R = I$ y calculamos la derivada de dicho producto respecto al tiempo se obtiene:

$$R^T \cdot \dot{R} = -(R^T \cdot \dot{R})^T$$

Es decir, el producto $R^T \cdot \dot{R}$ es una matriz antisimétrica. En general se puede representar esta matriz antisimétrica de la forma:

$$S(\omega) = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$$

En nuestro caso particular, el vector $S(\omega) = [\omega_1, \omega_2, \omega_3]^3$ recibe el nombre de velocidad angular del sistema cuerpo con respecto al sistema tierra representado en el sistema cuerpo. Desde el punto de vista del problema que nos atañe, la dirección de ω define el eje instantáneo de rotación en torno al cual está girando el dron además de su módulo que corresponde al valor de la velocidad instantánea de rotación.

Dado que R es ortogonal, se puede despejar $\dot{R}$.

$$\dot{R} = R \cdot S(\omega)$$

Tras esto ya tenemos una ecuación diferencial para la variación temporal de la matriz de rotación, solo falta estudiar la evolución del vector ω.
La segunda ley de Euler establece una variación del momento angular de un sistema y los pares de fuerza aplicados al mismo.

$$\frac{d}{dt}(RJ\omega) = (R\tau)$$

Donde J representa la matriz de inercia del sistema y Jω el momento angular en el sistema cuerpo. Por último τ representa los pares de fuerzas ejercidos.
Si se multiplican los dos miembros de la ecuación anterior por R^T y se desarrolla la derivada se consigue:

$$R^T(\dot{R}J\omega + RJ\dot{\omega} = \tau)$$

Por último, se sustituye $\dot{R}$ por S(ω) y se despeja $\dot{\omega}$.

$$\dot{\omega} = -J^{-1}S(\omega)J\omega + J^{-1}\tau$$

6.5 Fuerzas y pares en un cuadrotor.

Si uno se fija de nuevo en la figura 46, se puede considerar que cada motor ejerce una fuerza τ_{ri}, con i = 1,2,3,4. El valor de estos pares es proporcional al valor PWM suministrado a estos, por lo que podemos considerar estos pares como las señales de control del sistema.
La dinámica de las hélices se puede asociar al par generado por los motores como:

$$\tau_{ri} = \mathrm{B}\omega_{r_i}^2$$

Donde $\mathrm{B}\omega_{r_i}^2$ representa la resistencia al giro de la hélice por el aire y B, es el coeficiente de resistencia viscosa.
Otro dato que se conoce es que la hélice al girar genera una fuerza proporcional al cuadrado de su velocidad por lo que se puede relacionar directamente con el par del motor.

$$\mathrm{f}_{ri} = \mathrm{k}\omega_{r_i}^2 \rightarrow \mathrm{f}_{ri} = \frac{k}{B}\tau_{ri}$$

A partir de la fuerza ejercida por los motores, se puede calcular el valor de *u* y los pares ejercidos en los tres ejes del cuadrotor.

$$u = \sum_{i=1}^{4} \mathrm{f}_{ri}$$
$$\tau_1 = l(\mathrm{f}_{r1} - \mathrm{f}_{r2})$$
$$\tau_2 = l(\mathrm{f}_{r3} - \mathrm{f}_{r4})$$
$$\tau_3 = \tau_1 + \tau_2 - \tau_3 - \tau_4$$

Se ha supuesto en todo momento que los motores y hélices son idénticos y que suministran el mismo par y empuje respectivamente. En el mundo real sería necesario comprobar estos valores para si no son idénticos, modificar las ecuaciones o trimar los motores y las hélices. Tal y como se introdujo en el apartado 2.4 donde había una tabla resumen con algunas características del sistema real, estos junto con otros parámetros se encuentran especificados entre las líneas 7 y 28 del archivo uav.py.

6.6 Integración de las ecuaciones diferenciales del modelo.

El uso de algoritmos numéricos para obtener la solución de un sistema exige que estas estén escritas en variables de estado (ecuaciones de primer orden). Se debe por tanto reescribir las ecuaciones de movimiento para que tengan esa forma.
En primer lugar, se define el vector "*y*" como un vector que contendrá todas las variables de estado del sistema, si nos fijamos ahora en la tabla inferior, se muestran todas las componentes del vector, así como las variables de estado que representan.

y_i	variable	observaciones	uds
y_0	v_x	C. x de la velocidad en ejes tierra	m/s
y_1	v_y	C. y de la velocidad en ejes tierra	m/s
y_2	v_z	C. z de la velocidad en ejes tierra	m/s
y_3	x	C. x de la posición en ejes tierra	m
y_4	y	C. y de la posición en ejes tierra	m
y_5	z	C. z de la posición en ejes tierra	m
y_6	b_{11}	C. x en ejes T del eje x del sistema B	VU
y_7	b_{21}	C. y en ejes T del eje x del sistema B	VU
y_8	b_{31}	C. z en ejes T del eje x del sistema B	VU
y_9	b_{12}	C. x en ejes T del eje y del sistema B	VU
y_{10}	b_{22}	C. y en ejes T del eje y del sistema B	VU
y_{11}	b_{32}	C. z en ejes T del eje y del sistema B	VU
y_{12}	b_{13}	C. x en ejes T del eje z del sistema B	VU
y_{13}	b_{23}	C. y en ejes T del eje z del sistema B	VU
y_{14}	b_{33}	C. z en ejes T del eje z del sistema B	VU
y_{15}	ω_1	C. x de la ω en ejes B	rad/s
y_{16}	ω_2	C. y de la ω en ejes B	rad/s
y_{17}	ω_3	C. z de la ω en ejes B	rad/s

B (ejes cuerpo) T (ejes Tierra).

A continuación, se redefinen las ecuaciones en base a los parámetros del vector construido.

$$\begin{bmatrix} \dot{y_0} \\ \dot{y_1} \\ \dot{y_2} \end{bmatrix} = ge_3 - u \cdot \begin{bmatrix} y_6 & y_9 & y_{12} \\ y_7 & y_{10} & y_{13} \\ y_8 & y_{11} & y_{14} \end{bmatrix} \cdot e_3 - b \cdot \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

$$\begin{bmatrix} \dot{y_3} \\ \dot{y_4} \\ \dot{y_5} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

$$\begin{bmatrix} \dot{y_6} & \dot{y_9} & \dot{y_{12}} \\ \dot{y_7} & \dot{y_{10}} & \dot{y_{13}} \\ \dot{y_8} & \dot{y_{11}} & \dot{y_{14}} \end{bmatrix} = \begin{bmatrix} y_6 & y_9 & y_{12} \\ y_7 & y_{10} & y_{13} \\ y_8 & y_{11} & y_{14} \end{bmatrix} \cdot \begin{bmatrix} 0 & -y_{17} & y_{16} \\ y_{17} & 0 & -y_{15} \\ -y_{16} & y_{15} & 0 \end{bmatrix}$$

$$\begin{bmatrix} \dot{y_{15}} \\ \dot{y_{16}} \\ \dot{y_{17}} \end{bmatrix} = -J^{-1} \cdot \begin{bmatrix} 0 & -y_{17} & y_{16} \\ y_{17} & 0 & -y_{15} \\ -y_{16} & y_{15} & 0 \end{bmatrix} \cdot J \cdot \begin{bmatrix} y_{15} \\ y_{16} \\ y_{17} \end{bmatrix} + J^{-1}\tau$$

Ahora, para poder emplear los métodos de solución de ecuaciones diferenciales de Scipy, se crea una función cuya entrada será el vector "*y"*, el tiempo y que tenga como salida el vector $\dot{y}$ con las derivadas de todos los estados. Dicha función denominada *dt()* ha sido creada en el archivo *uav.py* entre las líneas 126 y 184

La especificación de los pares generados por los motores τ se obtendrán más adelante a partir de un controlador.
Por último para resolver el sistema de ecuaciones se emplean dos métodos implementados en la función *simulador* entre las líneas 186 y 200.

Esta función retorna dos objetos que permitirán obtener de forma iterativa las soluciones al problema:

- Sol2: emplea un "solver" conocido como RK45 (Runge-Kutta 45), permite calcular de forma iterativa todas las soluciones desde un tiempo inicial 0, hasta un final Tf. Es importante destacar que solo se ha creado con fines de depuración.

  Sol: emplea el método *solve_ivp* de Scipy, la función espera como entrada la función *dt()* pasada como función lambda. Este objeto es un envoltorio que permite emplear distintos métodos de integración de modo iterativo entre un instante inicial y un instante final de tiempo. Por defecto emplea también el solver RK45.

6.7 Control de vuelo.

Desde un punto de vista técnico el control de un dron se puede abordar de varias maneras distintas, para el caso concreto de este simulador se ha desarrollado un controlador PD (controlador proporcional y derivativo).

El controlador busca con esto llevar en todo momento unos valores de consigna fijados por tres variables del sistema (altura, rumbo y velocidad), que al variar estos permiten al dron variar su estado pudiendo completar el recorrido.

Es importante especificar que cada una de las tres variables del sistema llevará su propio PD cuyo resultado será aditivo, es decir se suman las contribuciones de cada controlador a un valor constante que se suministra a los motores para que estos obtengan una fuerza cero.

En los siguientes tres apartados se explicarán cada uno de los tres controladores por separado.

6.7.1 Controlador de altura.

Partiendo de la idea del dron en posición horizontal, para modificar la altura es necesario modificar el par aplicado por cada uno de los motores una cantidad proporcional al error de altura, dentro del simulador la variable que contiene la altura (z) es *y5.* Llamando *ca* a la altura deseada la parte proporcional queda de la siguiente manera:

$$\mathrm{d}\tau_{ri} = -\mathrm{k}_{\mathrm{pa}}(-\mathrm{ca} - \mathrm{y}_5)$$

Donde los signos negativos se introducen para tener en cuenta que la dirección en la que aumenta la altura es negativa en el eje z para coordenadas NED. Además k_{pa} corresponde a la constante de proporcionalidad del controlador.

Para añadir el control derivativo, basta con derivar el error de altura respecto al tiempo.

$$\frac{d(-ca - y_5)}{dt} = -y_2$$

Donde suponemos que la consigna de altura se mantiene constante, o varía de forma muy lenta, la variación del error en altura dependerá exclusivamente de la velocidad en el eje z. Añadiendo entonces el termino derivativo y su constante ajustable k_{da} obtenemos:

$$\mathrm{d}\tau_{ri} = -\mathrm{k}_{\mathrm{pa}}(-\mathrm{ca} - \mathrm{y}_5) + k_{da}y_2$$

El ajuste de las constantes k_{pa} y k_{da} se puede hacer sobre el modelo del dron teniendo en cuenta dos factores:

- Aumentar el control proporcional acelera la respuesta, pero la hace más oscilatoria.
- Aumentar el control derivativo ralentiza la respuesta, pero la hace menos oscilatoria y contribuye a estabilizar el sistema.

Otro dato a tener en cuenta es que hay que limitar el par del dron para que no sea menor que cero y para que no sobrepase el voltaje máximo que admite.

Este controlador ha sido implementado sin control integral, la razón es que no es necesario pues el dron ya actúa como integrador en sí mismo. A pesar de ello una implementación de este controlador podría aumentar la velocidad de respuesta del dron.

6.7.2 Controlador de rumbo.

Un dron por su estructura es capaz de avanzar en cualquier dirección sin cambiar su rumbo. Sin embargo, para este simulador se ha optado por suprimir esta característica y obligar a que cuando el dron cambie de dirección, cambie su rumbo.

Siguiendo el esquema de la figura 46, y la colocación del hat Navio, en el dron real, podemos considerar que la dirección x del magnetómetro corresponde a la bisectriz de los brazos b_1 y b_2, las direcciones x e y en coordenadas cuerpo del dron.

La dirección resultante se representa como la suma de b_1 [y_6, y_7, y_8] y b_2 [y_9, y_{10}, y_{11}]:

$$h \equiv \begin{bmatrix} h_x \\ h_y \\ h_z \end{bmatrix} = \begin{bmatrix} y_6 \\ y_7 \\ y_8 \end{bmatrix} + \begin{bmatrix} y_9 \\ y_{10} \\ y_{11} \end{bmatrix}$$

A continuación, se elimina la componente z del vector debido a que no se necesita para obtener la dirección (únicamente necesitamos la proyección de h sobre el plano x e y). Y normalizamos el vector resultante para obtener la dirección.

$$\hat{h} = \frac{h_{(h_z = 0)}}{\left| h_{(h_z = 0)} \right|}$$

Una vez calculado el vector $\hat{h}$ que marca el rumbo actual, es necesario compararlo con el de consigna (deseado). Para el sistema desarrollado se dará el valor de consigna en grados en referencia al norte, por lo tanto, el rango de este valor será de -180 <= ch <= 180, donde los valores positivos marcan el sentido norte-este-sur y los negativos norte-oeste-sur. Se genera por tanto un vector *vch* correspondiente al ángulo de consigna *ch* para poder compararlo con $\hat{h}$.

$$vch = \begin{pmatrix} cos(ch) \\ sin(ch) \\ 0 \end{pmatrix}$$

Para calcular el error de rumbo se calcula el producto vectorial $vch \times \hat{h}$ y el producto escalar $vch \cdot \hat{h}$ del rumbo actual y de consigna. El signo del producto vectorial será mayor que cero si el ángulo que forman *vch* y $\hat{h}$ es positivo, y negativo en caso contrario. Si se asocia el error de rumbo al producto vectorial, se cumplirá la propiedad mencionada, haciendo que el controlador lleve el dron hacia el rumbo de consigna mediante el giro más corto.

Esta aproximación cuenta con un inconveniente, el valor máximo de un producto vectorial es ∓ 1 cuando los vectores forman ∓ 90 grados teniendo en cuenta que, si forman un ángulo mayor, el producto vectorial vuelve a disminuir a 0. Para evitar que el controlador tenga poca intensidad en los ángulos más grandes, se añade una corrección al error dada por el producto escalar de los vectores. Esta corrección es la siguiente, si $vch \cdot \hat{h} < 0$ (el ángulo es de más de 90 grados o menor que -90 grados), se añade un 1 al producto vectorial, si es positivo y un -1 si es negativo.

$$err\ rumbo = \begin{cases} vch \cdot \hat{h} \geq 0 \rightarrow vch\ x\ \hat{h} \\ vch \cdot \hat{h} = 0 \rightarrow 1 \\ -1 < vch \cdot \hat{h} < 0 \begin{cases} vch\ x\ \hat{h} > 0 \rightarrow vch\ x\ \hat{h} + 1 \\ vch\ x\ \hat{h} < 0 \rightarrow vch\ x\ \hat{h} - 1 \end{cases} \end{cases}$$

Hay que tener en cuenta una situación peculiar, es el caso de que la diferencia de rumbos sea de 180º, en cuyo caso el producto vectorial será cero. Para subsanar este problema se usa el valor del producto escalar para asignar un valor de uno al error. De esta forma se podrá corregir el rumbo en este caso particular girando siempre en el sentido de las agujas del reloj.

El siguiente paso una vez calculado el error es aplicarlo en los motores para reducir este. Para ello hay que tener en cuenta si uno se fija en la figura 46 que:

- $\tau_{r1}\ y\ \tau_{r2}$ hacen girar al dron en el sentido de las agujas del reloj.
- $\tau_{r3}\ y\ \tau_{r4}$ hacen girar al dron en el sentido contrario al de las agujas del reloj.

Además, es preferible no alterar las sumas totales de los cuatro pares de motores, para evitar que ocurra esto se debe aumentar y disminuir en la misma cantidad siempre. Además, se considera que estos aumentos y disminuciones son proporcionales al error. El resultado de estas premisas es el siguiente:

$$d\tau_{r2} = d\tau_{r1} = -k_{ph} \cdot errorRumbo$$
$$d\tau_{r4} = d\tau_{r3} = k_{ph} \cdot errorRumbo$$

Donde k_{ph} corresponde a la constante proporcional del rumbo.
A continuación, es necesario añadir un control derivativo. Es sencillo comprender que la velocidad a la que disminuye el error del rumbo es proporcional a la velocidad angular en torno a z. Si suponemos una consigna de rumbo constante la velocidad angular en torno al eje z equivale a y_{17} de acuerdo con la tabla de variables, así concluimos que se puede usar directamente esta variable para añadir el control derivativo al error de rumbo.

$$d\tau_{r2} = d\tau_{r1} = -k_{ph} \cdot errorRumbo - k_{dh} \cdot y_{17}$$
$$d\tau_{r4} = d\tau_{r3} = k_{ph} \cdot errorRumbo + k_{dh} \cdot y_{17}$$

Siendo k_{dh} la constante del control derivativo. Al igual que en el caso del control de altura, esta constante puede ser ajustada directamente sobre el modelo.
Por último, al igual que ocurría con la altura no necesitamos un control integral pues el dron actúa por sí mismo como tal.

6.7.3 Controlador de velocidad.

Al igual que en el rumbo, también es posible controlar la velocidad del dron en cualquier dirección. De nuevo el controlador diseñado mantendrá dicha velocidad en la dirección del eje x marcada por el magnetómetro.

Antes de especificar el controlador de velocidad, hay que tener en cuenta que un dron no tiene apenas rozamiento por el aire por lo que es relativamente fácil que este se desplace de forma indeseada. Para evitar este desplazamiento se debe especificar la dirección de vuelo y establecer la premisa de que la velocidad en cualquier otra dirección será cero.

Respecto al eje z, la velocidad la establece el controlador de altura que se hace cero una vez alcanzada la altura de vuelo especificada, por lo que en este apartado no se hará más mención al control en dicho eje.

El primer paso para desarrollar el controlador es obtener la velocidad en los ejes rumbo, para ello se emplean las componentes x e y del vector $\hat{h}$ calculado en la sección anterior.

$$H = \begin{bmatrix} \hat{h}_x & \hat{h}_y \\ \widehat{-h}_y & \hat{h}_x \end{bmatrix}$$

A continuación, se puede representar la velocidad en el plano xy.

$$v = H \cdot \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$$

Una vez conocida la velocidad, se resta con la consigna de velocidad para obtener el error entra ambas.

$$err\ velocidad(erv) = \begin{bmatrix} cv \\ 0 \end{bmatrix} - v$$

En la ecuación, cv corresponde a la consigna de velocidad, además se impone un 0 como consigna en la dirección perpendicular.

Ahora para que el dron avance en la dirección esperada, se necesita inclinar el dron hacia delante aumentando los pares $\tau_{r1}\ y\ \tau_{r4}$ y disminuyendo los otros dos en la misma proporción.

$$\mathrm{d}\tau_{r1} = \mathrm{d}\tau_{r4} = k_{pv} \cdot erv_1$$
$$\mathrm{d}\tau_{r2} = \mathrm{d}\tau_{r3} = -k_{pv} \cdot erv_1$$

Donde k_{pv} es la constante del error proporcional ajustable y erv_1 es la primera componente del error de velocidad.
Del mismo modo, se añade la corrección del error en la dirección perpendicular que buscamos que sea cero. Para corregirlo se debe hacer girar el dron lateralmente (roll) aumentando los pares $\tau_{r1}\ y\ \tau_{r3}$ y disminuyendo los pares $\tau_{r2}\ y\ \tau_{r4}$.

$$\mathrm{d}\tau_{r1} = \mathrm{d}\tau_{r3} = k_{pv2} \cdot erv_2$$
$$\mathrm{d}\tau_{r2} = \mathrm{d}\tau_{r4} = -k_{pv2} \cdot erv_2$$

Donde k_{pv2} es la constante ajustable del error proporcional y erv_2 es la segunda componente del error de velocidad.

Por último, se añade el control derivativo. Al igual que en el caso del rumbo, se puede asociar el cambio de velocidad al ritmo de inclinación del dron. Se puede asociar ese ritmo de cambio a la velocidad angular. Así, el ritmo al que se inclina el dron hacia adelante y hacia atrás viene determinado por la velocidad angular en torno al eje

perpendicular a la dirección de avance. Mientras que el ritmo de inclinación lateral viene determinado por la velocidad angular en torno al eje marcado por la dirección de avance. Dichas velocidades angulares se pueden obtener a partir de ω_1 y ω_2. Para ello es necesario tener en cuenta que estas variables se miden en ejes cuerpo. Para obtener las velocidades angulares en torno a los ejes que nos interesa, es suficiente girarlas 45º y sumar los resultados.

$$erdv = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} \\ \frac{-\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \cdot \begin{bmatrix} y_{15} \\ y_{16} \end{bmatrix}$$

La contribución de los pares quedará por tanto como:

$$\begin{gathered} \mathrm{d}\tau_{r1} = k_{dv1} \cdot erdv_2 - k_{dv2} \cdot erdv_1 \\ \mathrm{d}\tau_{r2} = -k_{dv1} \cdot erdv_2 + k_{dv2} \cdot erdv_1 \\ \mathrm{d}\tau_{r3} = -k_{dv1} \cdot erdv_2 - k_{dv2} \cdot erdv_1 \\ \mathrm{d}\tau_{r4} = k_{dv1} \cdot erdv_2 + k_{dv2} \cdot erdv_1 \end{gathered}$$

Donde k_{dv1} y k_{dv2} son las constantes ajustables del error derivativo.

Solo resta para terminar, sumar todas las contribuciones calculadas previamente a los pares para obtener el incremento total de cada uno de los motores. A continuación, se expone el incremento para el primer par motor (el resto de ellos presentan un esquema similar).

$$\begin{gathered} \mathrm{d}\tau_{r1} = -k_{pa}(-ca - y_5) + k_{da} y_2 - k_{ph} \cdot errum \\ -k_{dh} \cdot\ y_{17} + k_{pv} \cdot erv_1 + k_{pv2} \cdot erv_2 + k_{dv1} \cdot erdv_2 - k_{dv2} \cdot erdv_1 \end{gathered}$$

Por último, indicar que todo el código que implementa este controlador se encuentra en la función *controlador* del archivo *uav.py*.

6.8 Interfaz de simulación.

La figura 47, es una captura de la salida proporcionada por el simulador durante un recorrido dado.

Como se puede ver hay tres gráficos, el primero de ellos, situado arriba a la izquierda, corresponde a un gráfico tridimensional del vuelo del aparato, dicha representación puede ser alterada moviendo el cursor para cambiar la perspectiva de visión. Además, permite ver en forma de puntos las coordenadas geográficas seleccionadas como puntos de paso.

En el caso del gráfico a la derecha del anterior, corresponde a la vista desde arriba en dos dimensiones del recorrido que está haciendo el aparato, donde se aprecia que ha

pasado por los puntos cero, uno y dos representados en el plano cartesiano, como aclaración estos puntos son los mismos que los seleccionados por la aplicación como puntos de paso.

Por último, el gráfico inferior corresponde al registro de la altura del aparato durante todo el recorrido, como se aprecia lleva en el momento de la captura una altura de diez metros.

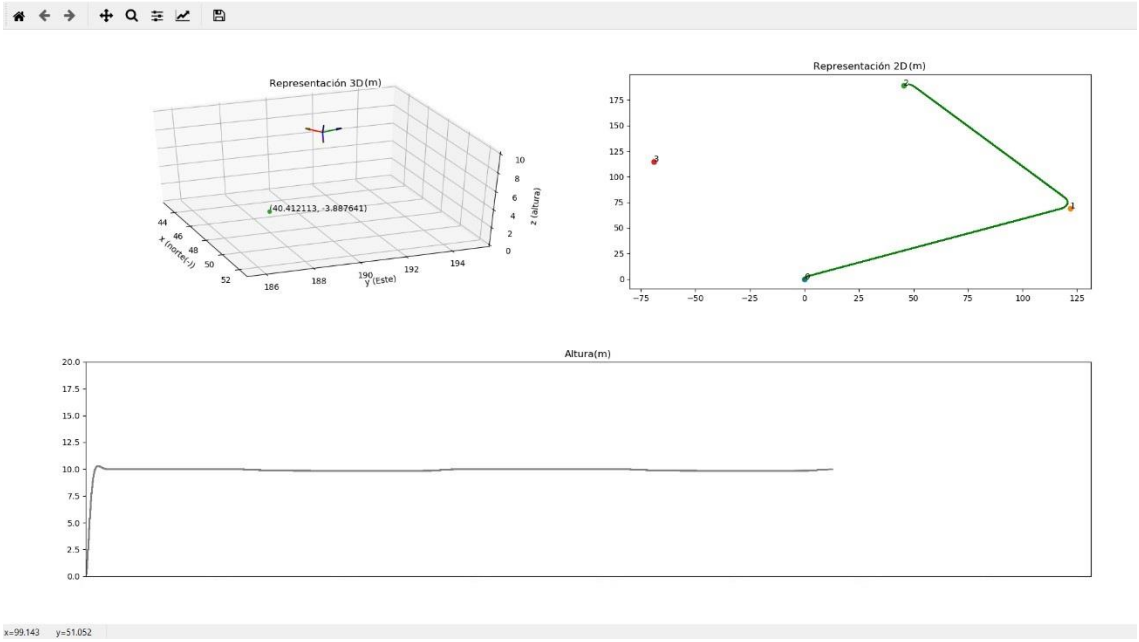

Fig 47. Captura del simulador durante una ejecución

Para la elaboración de los tres gráficos se ha utilizado la biblioteca matplotlib, la implementación se ha limitado a recoger los valores de telemetría del dron en cada iteración de tiempo y almacenarlos en un vector para luego usando el método *animation.FuncAnimation* crear una simulación formada por sucesivos fotogramas.

6.9 Procesado e integración del recorrido.

El primer paso para procesar el recorrido es obtenerlo, aquí entra la aplicación desarrollada, de igual manera que ocurría con el sistema real que alojaba el servidor, el simulador también lo hará, esta tarea es sencilla dado que el servidor estaba programado en el apartado 4, basta con añadir esas funciones al simulador entre las líneas 79 y 120 del archivo simulador.py.

Tras este paso el simulador recibirá una serie de coordenadas geográficas, así como otros valores como la altura especificada en los ajustes. La dificultad de este paso radica en cómo convertir coordenadas geográficas a cartesianas, para ello se han seguido los siguientes pasos:

- Se especifica que las coordenadas de despegue del dron proporcionadas por el GPS serán el punto (0,0) del gráfico.
- A continuación, se calcula la distancia entre los pares de puntos usando la fórmula de Haversine (54) y se almacena en un vector.
- Después, se calcula la orientación de cada punto en función del anterior en las líneas 54 a 62 utilizando las fórmulas siguientes sabiendo que p1 y p2 corresponden al punto uno y dos respectivamente (59):

$$X = \cos(latP2) \cdot \sin(longP2 - longP1)$$
$$Y = \cos(\text{latP1}) \cdot \sin(\text{latP2}) - \sin(\text{latP1}) \cdot \cos(\text{latP2}) \cdot \cos(longP2 - longP1)$$
$$orientacion = \operatorname{atan}\left(\frac{X}{Y}\right)$$

- Por último, usando los datos de los dos puntos anteriores, en las líneas 73 a 76 se calcula el punto cartesiano correspondiente a cada una de las coordenadas geográficas utilizando las fórmulas inferiores teniendo en cuenta que (x2,y2) es el punto buscado, (x1,y1) el actual y la variable orientación es la calculada en la fórmula del punto anterior (60):

$$X2 = X1 + (X2 - X1) \cdot \cos(orientacion)$$
$$Y2 = Y1 + (Y2 - Y1) \cdot \cos(orientacion)$$

Tras estos cálculos ya se pueden rellenar los gráficos tal y como se hace en las líneas 122 a 137. Por último resta que el dron recorra esos puntos, para lograr eso, en el archivo uav.py se especifica entre las líneas 137 a 165 una serie de condiciones que se limitan a reducir la velocidad al acercarse a un punto de paso para variar el rumbo de forma controlada y más cerrada a dos metros de él, su otro propósito es cuando se va acercando al último punto reducir la velocidad y la altura lentamente (líneas 159 y 163) para aterrizar de forma controlada, tras varias depuraciones los valores establecidos son para girar una velocidad de 1m/s y para navegar de 2m/s. Estos valores en principio no son configurables desde la app, la razón es que si se modificaran podrían generar que el dron recorriera de forma muy ineficiente y nada ajustada los puntos.

6.10 Análisis del simulador

6.10.1 Introducción.

El objetivo de este subapartado es demostrar la validez del modelo virtual creado realizando una serie de recorridos utilizando la aplicación *Pathfinder* para la entrada de los parámetros.

Con el fin de documentar de manera más clara todos los pasos que realiza el programa hasta finalizar, la aplicación que le transmitirá los datos se ejecutará en un emulador de móvil para grabar junto al simulador el proceso completo y poder alojar el resultado en el repositorio del trabajo en la carpeta videos/simulaciones.

6.10.2 Ejecución de recorridos

Recorrido de ejemplo 1.

En este primer recorrido tal y como se refleja en la figura 48, se han seleccionado cuatro puntos de paso que hacen una curva para comprobar el comportamiento y la exactitud del dron. Si ahora nos fijamos en la figura 49, se puede ver una captura de pantalla del video completo denominado simulacion1.mp4 del repositorio, en esta captura se ha finalizado el recorrido y el dron se encuentra parado en tierra.

Fig 48. Mapa del recorrido.

La primera impresión al ver la imagen es que en el punto uno y cuatro hay un ligero desvío, pero sin importancia dado que como se puede ver en el gráfico la distancia que se desvía no llega al metro, y la única razón para percibir este desvío es que, al ser un recorrido muy corto para no alargar demasiado el video, el gráfico se amplía mucho para captar los movimientos.

Por lo demás, el recorrido es exitoso, en el gráfico inferior se puede ver que ha alcanzado la altura por defecto de diez metros de forma muy rápida y que ha descendido a tierra de forma paulatina conservando en todo el recorrido la misma altura.

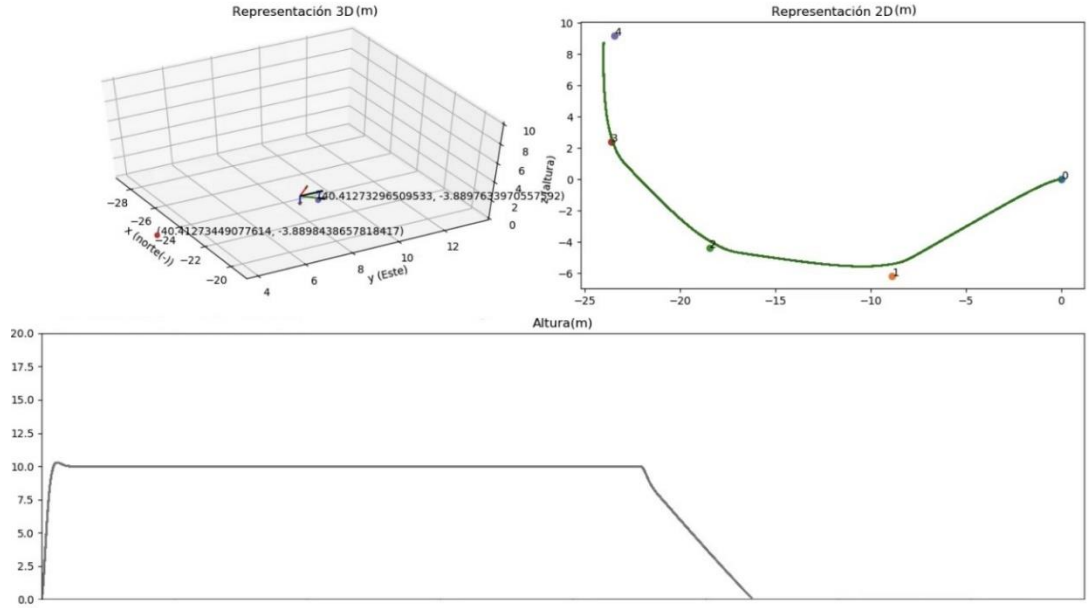

Fig 49. Recorrido finalizado.

Recorrido de ejemplo 2.

En este segundo recorrido grabado y denominado en el repositorio simulacion2.mp4 el primer paso tal y como se ve en la figura 50 ha sido modificar los parámetros por defecto de la pantalla de ajustes para establecer una altura de ocho metros y el regreso al origen de despegue una vez finalizado el recorrido.

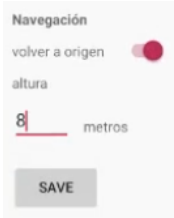

Fig 50. Imagen de la pantalla de configuración de navegación

Tras ello se repite el proceso de dibujar el recorrido teniendo en cuenta que tras el punto cuatro deberá volver el dron al origen.

Fig 51. Mapa del recorrido (II).

Por último, en la figura 52 se puede ver el recorrido asociado a esta simulación donde la vista en dos dimensiones demuestra que el recorrido es prácticamente perfecto y como se especificaba en los ajustes, el dron ha regresado del punto cuatro al de despegue.

La otra especificación (altura de 8 metros) también se cumple correctamente tal y como se ve en la figura despegando y aterrizando correctamente en el mismo lugar.

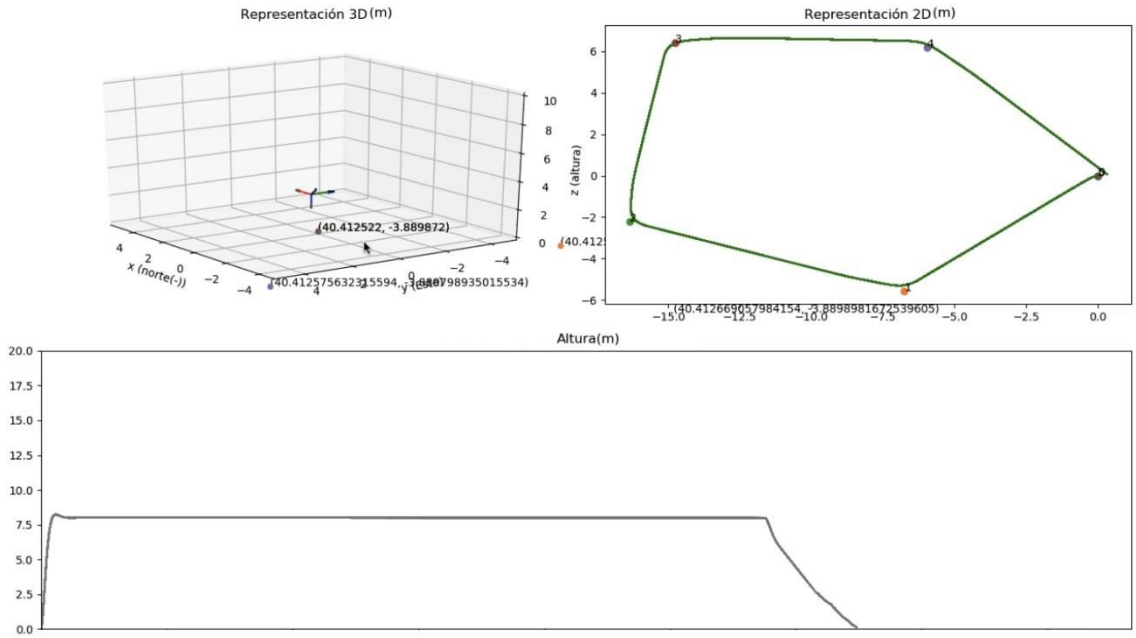

Fig 52. Recorrido finalizado (II).

6.11 Adaptación del controlador al sistema real.

La adaptación del sistema de control construido para el modelo, al sistema real es directa. Lo único necesario, es sustituir o estimar las variables de estado suministradas por el modelo, por los datos obtenidos a partir de los sensores de NAVIO.
Así, los valores de las posiciones en ejes tierra (y_3, y_4) se obtienen a través de los datos del GPS, mediante el script gps.py, y la fórmula de Haversine, anteriormente descritas. La altura se determina a través de la lectura del barómetro mediante las funciones descritas en el archivo barómetro.py. Las velocidades en los tres ejes, pueden estimarse a partir de la diferencia entre posiciones sucesivas y el tiempo transcurrido entre ellas, empleando un filtro de media móvil para suavizar el resultado. Esto daría como resultado los valores de y_0, y_1, y_2
Los valores y_5 a y_{14} Constituyen la matriz del cambio de coordenadas de ejes cuerpo a ejes tierra. Se trata de una matriz de rotación que puede obtenerse directamente a partir de los ángulos de Euler,

$$R = \begin{bmatrix} y_6 & y_9 & y_{12} \\ y_7 & y_{10} & y_{13} \\ y_8 & y_{11} & y_{14} \end{bmatrix} =$$

$$\begin{bmatrix} \cos\psi\cos\phi - \cos\theta\sin\phi\sin\psi & \cos\psi\sin\phi + \cos\theta\cos\phi\sin\psi & \sin\theta\sin\psi \\ -\sin\psi\cos\phi + \cos\theta\sin\phi\cos\psi & -\sin\psi\sin\phi + \cos\theta\cos\phi\cos\psi & \sin\theta\cos\psi \\ \sin\theta\sin\phi & -\sin\theta\cos\phi & \cos\theta \end{bmatrix}$$

Donde se ha seguido el orden de rotación estándar *roll*(ψ), *pitch*(θ), *yaw*(ϕ). Los valores de los ángulos se obtienen directamente de las lecturas de la unidad inercial a través de acelerometro.py.

Las tres últimas variables de estado: y_{15}, y_{16}, y_{17}, correspondientes a las velocidades angulares en ejes cuerpo, se pueden obtener también de la lectura de la unidad inercial, a través del archivo acelerometro.py cambiando el *return* de la línea 47 por el comentado de la línea 46.

Por último, habría que calcular el valor de la señal de PWM correspondiente para mantener el dron horizontal y estacionario. Obtener experimentalmente el ajuste de la salida del controlador, para obtener directamente los incrementos/decrementos en las señales de PWM enviadas a los motores del dron.

7. Conclusión.

La realización de este trabajo ha supuesto un gran reto debido a las condiciones vividas y la necesidad a causa de estas de modificar el fin último del proyecto de crear un dron real, completo y funcional, a pesar de ello, la creación del modelo virtual permite simular como sería el vuelo de este y la interacción con la aplicación de programación de tareas y con el medio. Además, se han proporcionado varias pruebas (imágenes, videos y gráficos) donde se detalla el funcionamiento simulado de la mayoría de las partes que conforman el trabajo.

En concreto, las características físicas que conforman el dron parecen ser admisibles para el vuelo, así lo demuestran los resultados de las simulaciones.

Además, tal y como se buscaba, el uso de la aplicación permite configurar los recorridos de una forma muy rápida y eficaz.

Por último, el reconocedor proporciona una notable eficacia en las imágenes ejemplo estudiadas

Se deja abierta la posibilidad de continuar con este proyecto mejorando varios aspectos, en el que se incluye principalmente el ajustar el controlador del modelo virtual del apartado 6 al diseño hardware para así tener un prototipo plenamente funcional, tal y como se expone en el apartado 6.11.

Así mismo se puede replantear el objetivo del trabajo y adaptarlo a otros fines como, la localización de vehículos, en el que sería necesario replantear la SSD y modificarla para detectar otro tipo de parámetros (matrícula, color, etc.).

Otro de los aspectos interesantes con los que se puede continuar, es la mejora de la aplicación Android añadiéndola nuevas funciones, dado que al ya tener una estructura de comunicación funcional no debería presentar mucha dificultad modificarla para enviar o recibir nuevos datos.

7. Conclusion.

The realization of this project has entailed a great challenge due to the lived conditions and the need to modify the final purpose to create a real, complete, and functional drone. Despite this, the creation of a virtual model allows to simulate how the flight would be, and the interaction with the tasks programming application and the environment. Furthermore, it's been provided several tests (pictures, videos, and graphics) where the simulated operation of most of the parts that form the project is detailed.

Specifically, the physical characteristics that define the drone seem to be admissible for the flight, just like this are shown by the simulation results.

Besides, just as wanted, the use of the application permits us to configure the itinerary quickly and effectively.
Lastly, the recognizer provides significant efficacy on the images used as a model for the study

The possibility to continue with this project, improving several aspects, is left open. Mainly to adjust the virtual model controller from chapter 6 to the hardware design to have a thoroughly functional model, just as is told in chapter 6.11.

Moreover, the objective of this work can be reconsidered and adapt it to other purposes like vehicles localization, where it would be necessary to modify the SSD to detect different parameters (license plate, color, etc)

One more aspect that could be interesting to be continued with, is the Android application upgrade, adding new functions, due to there being a communication working structure, it shouldn't present much of a complication to modify it to send or receive new data.

Bibliografía.

1) H. Shakhatreh et al., "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges," in IEEE Access, vol. 7, pp. 48572-48634, 2019, doi: 10.1109/ACCESS.2019.2909530.

2) S. Waharte and N. Trigoni, "Supporting Search and Rescue Operations with UAVs," 2010 International Conference on Emerging Security Technologies, Canterbury, 2010, pp. 142-147, doi: 10.1109/EST.2010.31.

3) M. Silvagni, A. Tonoli, E. Zenerino, and M. Chiaberge, `` Multipurpose UAV for search and rescue operations in mountain avalanche events,''Geomatics, Natural Hazards Risk, vol. 8, no. 1, pp. 1833, 2017.

4) https://github.com/juancalf/TFG

5) https://dronespain.pro/tipos-de-drones-aereos/

6) https://www.dronevision.es/drones-de-ala-fija/

7) https://search.creativecommons.org/photos/b141da05-8875-4b0b-a08c-6aaafe1f2edf

8) https://www.innodrone.es/partes-de-un-dron/

9) http://www.mateksys.com/?portfolio=hub5v12v

10) https://www.stlfinder.com/model/f450-arm/7396490/

11) https://emlid.com/navio/

12) https://emlid.com/navio/#navio-specs

13) https://www.raspberrypi.org/products/camera-module-v2/

14) https://www.rhydolabz.com/documents/26/BLDC_A2212_13T.pdf

15) https://corchitosrc.foroactivo.com/t2-motores-brushless

16) https://www.comprardrones.online/como-funcionan-las-helices-de-un-drone/

17) https://i.stack.imgur.com/fbjwb.jpg

18) http://dl.djicdn.com/downloads/flamewheel/en/F450_User_Manual_v2.2_en.pdf

19) https://www.thingiverse.com/thing:3723178

20) https://www.thingiverse.com/thing:92208

21) https://www.hostinger.es/tutoriales/que-es-ssh

22) https://docs.emlid.com/navio2/common/ardupilot/configuring-raspberry-pi/

23) https://www.meteorologiaenred.com/barometro.html

24) https://www.ugr.es/~andyk/Docencia/Metclim/Constantes.pdf

25) https://docs.emlid.com/navio2/common/dev/gps-ublox/

26) https://www.u-blox.com/sites/default/files/products/documents/u-blox6_ReceiverDescrProtSpec_%28GPS.G6-SW-10018%29_Public.pdf?utm_source=en%2Fimages%2Fdownloads%2FProduct_Docs%2Fu-blox6_ReceiverDescriptionProtocolSpec_%28GPS.G6-SW-10018%29.pdf

27) https://howtomechatronics.com/tutorials/arduino/arduino-brushless-motor-control-tutorial-esc-bldc/

28) http://blascarr.com/lessons/introduccion-al-imu-sistemas-de-navegacion-inercial/

29) http://bibing.us.es/proyectos/abreproy/12130/fichero/Cap%C3%ADtulo+5.pdf

30) Héctor García de Marina Ph D Th Distributed Formation Control for autonomous robot 2016, fig 7.2 .

31) https://diyodemag.com/_images/5abc7a3ec672e0e30e9bb110

32) AN5017. Aerospace Android and Windows Coordinate Systems. Rev. 2.0 — 21 June 2016 Application note.

33) https://www.nxp.com/docs/en/application-note/AN3461.pdf

34) https://theccontinuum.com/2012/09/24/arduino-imu-pitch-roll-from-accelerometer/

35) https://sites.google.com/site/myimuestimationexperience/sensors/magnetometer

36) https://www.fierceelectronics.com/components/compensating-for-tilt-hard-iron-and-soft-iron-effects

37) http://esprtk.wap.sh/tt/tt_w_mpu9250_calibrate_magnetometer_esprtk.html

38) http://esprtk.wap.sh

39) https://es.wikipedia.org/wiki/Cliente-servidor

40) Diapositivas de la asignatura Redes. Tema 5: La capa de transporte. Protocolos TCP y UDP. Universidad complutense de Madrid.

41) https://es.freelogodesign.org/terms-of-use

42) https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es

43) https://developers.arcgis.com/python/guide/how-ssd-works/

44) https://www.learnopencv.com/wp-content/uploads/2019/06/image-classification-vs-object-detection.png

45) https://www.deeplearningitalia.com/uso-del-aprendizaje-profundo-para-el-reconocimiento-de-objetos/

46) https://tech.amikelive.com/node-718/what-object-categories-labels-are-in-coco-dataset/

47) https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

48) https://www.tensorflow.org/tutorials/images/transfer_learning

49) https://github.com/tzutalin/labelImg

50) http://okutama-action.org/

51) https://cvgl.stanford.edu/projects/uav_data/

52) https://planetachatbot.com/conceptos-fundamentales-en-machine-learning-funci%C3%B3n-de-perdida-y-optimizaci%C3%B3n-e30c25404622

53) https://www.genbeta.com/desarrollo/como-calcular-la-distancia-entre-dos-puntos-geograficos-en-c-formula-de-haversine

54) https://i.imgur.com/BRtaU94.png?1

55) https://elements.envato.com/es-419/aerial-view-of-people-walking-in-forest-KM9C276

56) Bruce A. Francis and Manfredi Maggiore. Flocking and rendezvous in distributed robotics, 2016.

57) https://www.researchgate.net/profile/Ha_Thanh5/publication/324961790/figure/fig6/AS:760800683769856@1558400426053/Quadcopter-configuration.png

58) https://www.scipy.org/

59) https://www.igismap.com/formula-to-find-bearing-or-heading-angle-between-two-points-latitude-longitude/

60) https://sites.google.com/site/tovepet/Home/unidad-08