UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Máster en Ingeniería de Computadores y Redes

Trabajo Fin de Master

Sistema anti-colisiones para multicópteros basado en comunicaciones inalámbricas

Autor: Fabra Collado, Francisco José

Director: Tavares Calafate, Carlos Miguel

7 de julio de 2017

RESUMEN

El uso de multicópteros se está extendiendo rápidamente debido a los muchos campos de aplicación que tienen, por lo que cada vez más hay riegos de colisión entre estos aparatos. Además, se están empezando a proponer soluciones donde enjambres de multicópteros logren solucionar problemas más complejos mediante cooperación. Ambos casos requieren que dichos multicópteros se comuniquen entre ellos para coordinar sus trayectorias, algo que actualmente aún se está realizando solo a nivel experimental.

El principal objetivo de este proyecto es el desarrollo de un entorno de simulación para diseñar protocolos de comunicación entre multicópteros, que les permitan colaborar entre sí. El simulador permitirá analizar los protocolos implementados sin necesidad de realizar pruebas de campo, lo que reducirá el tiempo de implementación y el riesgo de que los multicópteros sufran daños.

Para emular correctamente el enlace de comunicación, ha sido necesario realizar pruebas de campo y analizar la pérdida de paquetes que se produce en función de la distancia entre los multicópteros. Como base, se ha utilizado un enlace WiFi Ad-hoc en la banda de los 5 GHz.

El simulador está completado en su totalidad, y es totalmente estable con hasta seis multicópteros funcionando a la vez. La escalabilidad de la aplicación está limitada por el uso de máquinas virtuales, aunque algunas modificaciones en su diseño permitirán ampliarla considerablemente en el futuro.

La primera aplicación del simulador ha sido el desarrollo de un protocolo para evitar la colisión entre multicópteros.

El protocolo de evitación de colisiones es suficientemente robusto como para haber evitado la colisión en todas las pruebas que se han realizado.

Palabras clave: Dron, Pixhawk, Raspberry Pi, WiFi, Ad-hoc, MAVLink, simulador, máquina virtual, colisión

ABSTRACT

The use of multicopters is spreading rapidly due to their wide scope of application. This means that, as time goes by, the risk of collision between these devices significantly increases. Moreover, solutions where swarms of multicopters are used to solve more complex problems through cooperation are being proposed. Both cases require multicopters to communicate among them to coordinate their flight, something that is only being done at an experimental level at this moment.

The main objective of this project is to develop a simulation environment to design communication protocols among multicopters, enabling collaboration among them. The simulator will permit to analyse the implemented protocols with no need for field tests, which will reduce the implementation time and the risk of damage on multicopters.

Field tests have been made to analyse the packet loss according to the distance between multicopters to properly emulate the communications link. As reference, a WiFi Ad-hoc link in the 5 GHz band has been used.

The simulator has reached its final development status, and is completely stable with up to six multicopters running at the same time. The application scalability is limited due to the use of virtual machines, but a few changes in its design will allow scaling it significantly in the future.

The development of a collision avoidance solution among multicopters has been the first use of the simulator.

Experiments show that the collision avoidance protocol is robust enough, as it has avoided collisions on all the tests that have been done.

Keywords: Drone, Pixhawk, Raspberry Pi, WiFi, Ad-hoc, MAVLink, simulator, virtual machine, collision

Índice de contenidos

Índice de figuras

Índice de tablas

1 Introducción

1.1 Motivación

Los drones son vehículos aéreos no tripulados (VANT) que realizan vuelos programados o dirigidos desde control remoto. Se trata de vehículos que han ganado gran popularidad a escala internacional durante los últimos años, siendo capaces de realizar gran variedad de tareas:

- Uso militar para reconocimiento y ataque
- Levantamientos topográficos
- Gestión de cultivos
- Lucha contra incendios
- Seguridad civil
- Rescate de personas
- Vigilancia de oleoductos
- Toma de muestras en volcanes
- Realización de filmaciones
- Uso recreativo

A medida que se extiende el uso de drones se detectan nuevas aplicaciones, entre las que se encuentran aquellas que requieren la actuación coordinada de varios de ellos. Por otro lado, la proliferación de drones también crea la necesidad de establecer protocolos y enlaces de comunicación entre los mismos para evitar su colisión cuando se encuentren relativamente cerca entre sí. Teniendo en cuenta estos factores, es más que pertinente estudiar tecnologías y protocolos mediante los cuales los drones puedan trabajar en equipo.

En este contexto, la tecnología WiFi surge como una opción viable ya que está ampliamente desarrollada y extendida, lo que la hace una alternativa adecuada para resolver la problemática planteada. Además, dado que los drones se desplazan por un volumen de espacio extenso durante sus vuelos, el uso de puntos de acceso (PA) queda totalmente descartado, ya que el enlace entre dron y PA podría perderse con facilidad. Por este motivo, la única alternativa viable a día de hoy es el uso de una red Ad-hoc sostenida por los mismos drones.

1.2 Objetivos y etapas del proyecto

En el contexto planteado, el presente trabajo tiene como **objetivo** el desarrollar un simulador en el que se pueda emular la posible interacción entre drones, así como implementar un protocolo de evitación de colisiones por medio de dicho simulador. Para lograrlo es necesario modelar también el enlace de comunicación entre los drones.

El simulador desarrollado permitirá comprobar la corrección de protocolos de comunicación y de coordinación entre drones sin necesidad de emplear drones reales, lo que supone un ahorro considerable de tiempo y recursos para los desarrolladores e investigadores.

Para lograr el objetivo antes definido, los pasos seguidos durante el desarrollo del proyecto son los siguientes:

1. Configuración de dos drones dotados de Raspberry Pi 3 para comunicarse mediante una red ad-hoc WiFi en la banda de los 5 GHz.
2. Pruebas de campo y modelado de la calidad del enlace de comunicación en función de la distancia entre los drones, mediante dos drones reales y la aplicación *Dronning*.
3. Preparación del entorno de desarrollo (instalación de drones virtuales en máquinas virtuales).
4. Desarrollo del simulador de vuelo.
5. Validación de la aplicación.
6. Ampliación del simulador con el protocolo de evitación de colisiones.
7. Validación del protocolo.

1.3 Estructura del documento

En el siguiente apartado se proporciona información sobre trabajos realizados en la presente área de investigación. A continuación, en el apartado 3, se describen los medios utilizados durante el desarrollo, incluyendo las características del ordenador, el software y los drones reales.

En el apartado 4 se detallan las fases por las que pasa el simulador cada vez que se realiza una prueba, y se explica el funcionamiento de la interfaz gráfica de usuario.

El apartado 5 describe la estructura interna del simulador, con énfasis en el rol que adopta cada uno de los hilos de ejecución que se lanzan durante la ejecución.

En el punto 6 se indican los pasos seguidos para modelar el enlace WiFi entre los drones virtuales. Partiendo de trabajos anteriores, se ha medido la calidad del enlace de comunicación en la banda de los 5 GHz con dos drones dotados de una Raspberry Pi 3, y haciendo uso de la aplicación *Dronning*, desarrollada por este mismo autor en su Trabajo de Fin de Grado.

El apartado 7 detalla los trabajos realizados para comunicar el simulador con los drones virtuales mediante mensajes en formato MAVLink, incluyendo los protocolos de comunicación, así como las funciones de control implementadas.

En el apartado 8 se expone con detalle el funcionamiento del protocolo de evitación de colisiones. Primero se explica el rol que adopta cada uno de los hilos de ejecución que completan el protocolo, y luego se entra en detalles de la implementación, como el formato de la trama de datos enviada o los cálculos que se realizan; después se detallan los parámetros del protocolo que se pueden modificar directamente desde la interfaz gráfica de usuario, y por último se muestra un ejemplo en el que se analiza el comportamiento de los drones durante una prueba, desde el punto de vista del protocolo de evitación de colisiones.

En los apartados 9 y 10 se validan el correcto funcionamiento del simulador y del protocolo de evitación de colisiones, respectivamente.

En el apartado 11 se indican las conclusiones inferidas del presente Trabajo Fin de Máster, y en el 12 se indican posibles vías de ampliación del estudio iniciado en este proyecto.

Tras la bibliografía y el glosario, se incluyen seis anexos.

El primer anexo describe las características técnicas de los drones empleados para modelar el enlace de comunicaciones WiFi entre los drones, mientras que el segundo se centra en la configuración que fue necesario realizar en los mismos para realizar las tomas de datos en campo.

En el tercer anexo se pormenoriza la preparación del entorno de desarrollo, con la instalación de los drones virtuales que son controlados por el simulador.

El anexo cuatro describe los cálculos realizados para descargar y visualizar el mapa geoposicionado mostrado en pantalla, sobre el que los drones realizan la prueba.

El quinto anexo describe los cálculos necesarios para determinar las coordenadas a las que se debe apartar un dron para ceder paso a otro dron, en caso de ser necesario.

El último anexo enumera los parámetros empleados en cada una de las pruebas de validación del protocolo de evitación de colisiones. Estos datos permitirán repetir los experimentos mediante el uso del simulador, siempre que se desee.

2 Trabajos existentes en el área

En lo que respecta a la evitación de colisiones, existen numerosos estudios centrados en vehículos aéreos no tripulados (UAV) de ala fija como [1], por tratarse de una aplicación ampliamente utilizada en aviación.

Por otro lado, también se han realizado estudios para evitar la colisión entre multicópteros cuando se agrupan en un enjambre [2], utilizando principalmente como estrategia el guardar una distancia de seguridad en base a sensores no cooperativos.

En [3] se realiza un estudio teórico de las características que debe tener un protocolo de evitación de colisiones, describiendo sus elementos y diferenciando entre el uso de sensores no cooperativos, como un sensor de proximidad [4] o una cámara [5], y sensores cooperativos, como el envío de información de vuelo al resto de UAVs.

Se prevé que en el futuro se utilice drones para realizar tareas planificadas en una misión, como la supervisión de campos de cultivo o el reparto de paquetes a domicilios aislados. En este contexto, el uso de sensores no cooperativos es útil para evitar la colisión con objetos, pero no es lo suficientemente eficiente como para evitar colisionar con otros multicópteros, por lo que resulta conveniente utilizar técnicas de sensorización cooperativa para evitar la colisión entre drones, como se hace en el presente proyecto.

En lo que respecta a la simulación de drones, en [6] se incluye una lista de simuladores de vuelo ya implementados. Varios de ellos emulan con considerable precisión las características y propiedades físicas de un dron, aunque son de pago, con código no accesible, compatibles solamente con algunas plataformas, y permiten el control del dron exclusivamente mediante un mando de control remoto o un teclado.

El simulador Software In The Loop (SITL) [7], por el contrario, es gratuito, compatible con Linux, MAC y Windows, simula el dron con gran precisión, y además permite comunicarse directamente con el mismo por TCP.

Todos los simuladores existentes hasta el momento se centran en emular un único dron, por ser lo demandado por la inmensa mayoría de los usuarios. El presente proyecto pretende, por el contrario, desarrollar un simulador con el que poder coordinar tareas entre varios drones virtuales.

3 Equipamiento utilizado

La implementación del simulador se ha realizado sobre un **ordenador** con las siguientes características:

- Placa base: MSI B250M PRO-VDH
- Memoria RAM: 32 GB DDR3 2133 MHz
- CPU: Intel Core i7-7700 3.6 GHz (4 cores hyperthreading → 8 threads)
- Tarjeta gráfica: Incorporada en la CPU
- Disco duro: SanDisk SDSSDA48 (500GB tipo SSD)
- Sistema operativo: Ubuntu 16.10

El desarrollo se ha realizado tomando como base el simulador **Software In The Loop** (SITL), capaz de simular un dron virtual.

SITL consiste en una compilación en lenguaje C++ de ArduPilot, el software que se instala en la Pixhawk para controlar un dron. La instalación se puede realizar sobre Linux, Windows o sobre una máquina virtual con el sistema operativo Linux.

Para controlar el dron virtual se utiliza la aplicación **MAVProxy** [44], que se conecta a SITL por TCP y proporciona una consola e interfaces UDP y TCP para enviar órdenes a los drones, así como recibir información de vuelo de los mismos.

Desgraciadamente, la aplicación MAVProxy no permite controlar simultáneamente más de un dron, ni tampoco ejecutar varias instancias de la misma de forma simultánea. Por lo tanto, para poder simular varios drones en el mismo ordenador, es necesario instalar SITL según la tercera alternativa, usando una máquina virtual para cada dron virtual.

El uso de máquinas virtuales supone un coste computacional elevado, por lo que se decidió utilizar un ordenador de elevadas prestaciones con el objetivo de poder simular un número elevado de drones, tal y como permite el simulador desarrollado.

La instalación y gestión de las máquinas virtuales que albergan los drones virtuales se ha facilitado con el uso del software **Vagrant**. En el anexo III se detalla el proceso necesario para instalar drones virtuales para su utilización en el simulador implementado.

El medio inalámbrico no proporciona la misma fiabilidad que el cableado, por lo que los paquetes de datos transmitidos entre drones no siempre llegan a su destino. El simulador tiene en cuenta esta eventualidad, y para ello ha sido necesario modelar la tasa de pérdida de paquetes de datos en función de la distancia entre el dron emisor y el dron receptor del paquete. Para ello se han utilizado dos **multirrotores GRCQuad** y la aplicación **Dronning**, desarrollada por este mismo autor para el Trabajo Fin de Grado titulado "*Desarrollo, evaluación y modelado de un sistema de comunicaciones para multicópteros*" presentado en la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia durante el curso 2015-2016.

Se ha considerado la transmisión en las bandas de 2,4 y 5 GHz, concluyendo la necesidad de utilizar la segunda, como se detalla en el apartado 6.

Para caracterizar la banda de los 2,4 GHz en términos de prestaciones del canal se ha utilizado información disponible en trabajos anteriores, mientras que para la banda de los 5 GHz ha sido necesario realizar mediciones con los multirrotores, por medio de la aplicación *Dronning*.

En ambos multirrotores se ha instalado una Raspberry Pi 3 en su parte superior para dotarles de capacidad de comunicación Wi-Fi y de capacidad de cómputo suficiente para realizar la toma de datos.

En el anexo I se adjunta información detallada sobre las características del multirrotor.

4 Funcionamiento del simulador e interacción con el usuario

La aplicación **CollisionSimulator** ha sido desarrollada de forma modular, separando el código necesario para gestionar la interfaz gráfica y los drones virtuales, del código necesario para el protocolo de evitación de colisiones. De este modo, si más adelante se desea adaptar la aplicación a otros usos, será una tarea abordable, sin necesidad de realizar grandes modificaciones.

Una posibilidad sería eliminar el código que ordena a los drones seguir una misión predefinida para, a continuación, implementar un algoritmo que permita la coordinación de un enjambre de drones. Esto permitiría validar algoritmos de coordinación entre drones sin necesidad de pruebas de campo, lo que reduciría considerablemente el tiempo necesario para el desarrollo del algoritmo, y evitaría el riesgo de dañar o perder los drones durante el proceso.

La función básica de la aplicación es la de proporcionar a los drones virtuales una misión o lista de *waypoints* por los que el dron tendrá que pasar.

Utilizando un número elevado de drones se podría utilizar este comportamiento para analizar la diseminación de información entre los drones y, por lo tanto, se podría estudiar redes DTN y algoritmos de encaminamiento.

En la figura 1 se observa la máquina de estados finitos que rige el comportamiento de la aplicación.

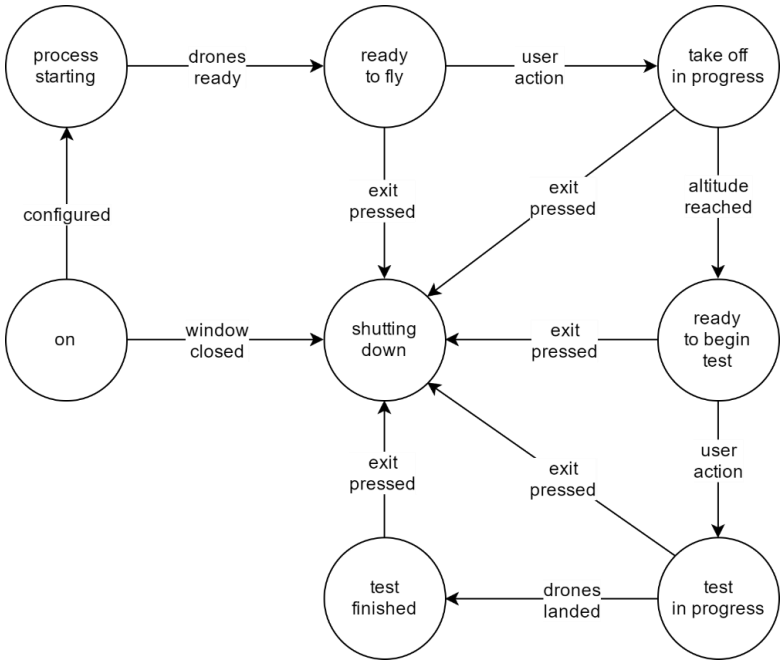

Figura 1: Máquina de estados finitos del simulador.

Los círculos representan los estados por los que va pasando la aplicación, mientras que las flechas son los eventos que provocan el cambio de estado. Primero se introduce la configuración de la prueba, y después se ponen en marcha los drones (*"process starting"*). A continuación, el usuario inicia el despegue (de *"ready to fly"* a *"take off in progress"*), espera a

que se alcance la altitud de vuelo, y después comienza la prueba (de *"ready to begin test"* a *"test in progress"*). Por último, cuando la prueba ha terminado, el usuario detiene la aplicación (de *"test finished"* a *"shutting down"*).

Como puede observarse, excepto durante el arranque de las máquinas virtuales, el usuario puede detener la aplicación en cualquier momento.

Como primer uso del simulador, se ha decidido desarrollar un protocolo para evitar colisiones en vuelo, puesto que los drones pueden coincidir en el espacio y en el tiempo mientras realizan una misión.

Dado que la aplicación se orienta a la investigación y no a usuarios convencionales, el diseño de la misma se ha centrado en la facilidad de manejo y en la corrección de su funcionamiento, no habiendo considerado el atractivo visual como una prioridad.

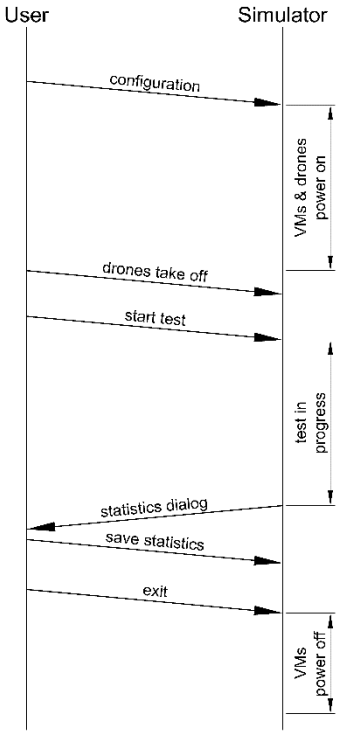

Figura 2: Diagrama de interacción con el usuario.

Otra forma de ver el proceso es teniendo en cuenta la evolución de una simulación en el tiempo. Como se observa en la figura 2, al arrancar la aplicación se solicita al usuario que introduzca la configuración de la prueba a realizar. A continuación, la aplicación arranca las máquinas virtuales que albergan los drones y, seguidamente, los enciende, indicándoles la ubicación inicial para la prueba. Una vez los drones están en línea, el usuario puede realizar el despegue, para después iniciar la prueba. Dicha prueba no termina hasta que los drones han llegado a destino y han aterrizado. En ese momento, la aplicación muestra un resumen con los resultados. El usuario puede decidir si guarda la información visualizada o la descarta. Por último, el usuario cierra la aplicación, no sin que antes el simulador detenga las máquinas virtuales utilizadas durante el proceso.

A continuación se explica con detalle el uso del simulador. Cada sub-apartado corresponde a uno de los pasos ya indicados.

4.1 Configuración

Al arrancar la aplicación se muestra el cuadro de diálogo de la figura 3.

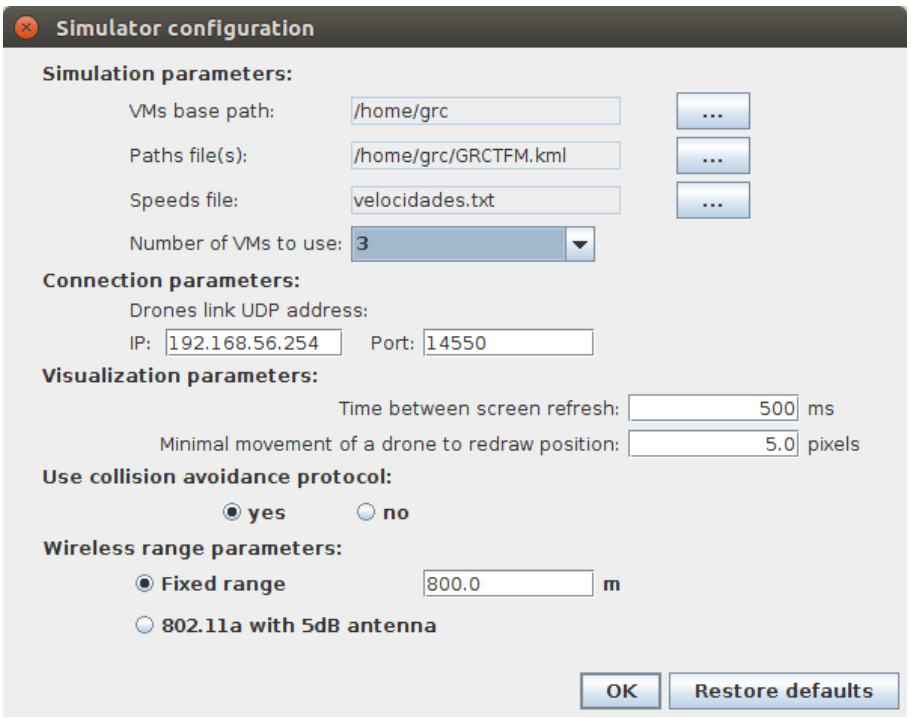

Figura 3: Cuadro de diálogo de configuración.

En el primer grupo de parámetros se indica, entre otros, la carpeta donde están almacenadas las máquinas virtuales. Al seleccionar la carpeta, la aplicación comprueba que las máquinas están dadas de alta en Vagrant y que, por lo tanto, disponen de un dron virtual para ser controlado.

Como ya se ha dicho, el simulador requiere introducir una misión o lista de *waypoints* para cada dron implicado en la prueba. Se barajó como primera opción implementar una ventana para introducir manualmente los *waypoints* por los que tiene que pasar cada dron. Sin embargo, esta opción se descartó por dos motivos. Por un lado, la introducción de la misión sería un proceso lento, y si se desea repetir una prueba es necesario reintroducir a mano todos los datos, lo que no es eficiente. Por otro lado, la solución no sería escalable, pues la complejidad de la ventana y de la introducción de datos crece linealmente con el número de drones que se desea controlar.

La solución adoptada consiste en importar ficheros con la misión a seguir por cada dron. De esta forma, repetir una prueba no implica reintroducir información.

El simulador puede importar dos tipos de fichero:

- **kml de Google Earth**. Permite introducir la misión de uno o varios drones en un solo fichero. Esta opción tiene dos limitaciones. Primero, la precisión al definir los puntos de paso del dron es menor, ya que se realiza de forma gráfica en Google Earth, por medio del ratón. Segundo, solamente se puede definir un tipo

de *waypoint*, el que corresponde a un punto por el que debe pasar el dron. Por otro lado, esta solución tiene una enorme ventaja, y es la velocidad para definir pruebas. Al definir misiones con unos pocos clics de ratón, construir una batería de pruebas es un proceso sencillo y rápido.

- **waypoint file**. Es el formato utilizado para almacenar la misión de un dron en QGroundControl y otras implementaciones. Como inconveniente, es necesario generar el fichero manualmente, introduciendo los puntos de paso en coordenadas geográficas, o mediante una aplicación de terceros. Como ventaja, permite definir los puntos de paso con precisión, así como definir otros tipos de *waypoint*, como aterrizar, despegar de nuevo, permanecer estático en vuelo al llegar a un punto, volver al punto de origen, etc. [46].

El usuario puede seleccionar un fichero kml o un grupo de ficheros de *waypoints*.

El tercer parámetro permite cargar un fichero donde se define la velocidad a la que tienen que circular los drones. Se trata de un fichero de texto donde se debe almacenar un valor de tipo *"double"* por cada fila, es decir, por cada dron.

Es de notar que, si la misión se carga desde un fichero del tipo QGroundControl, ésta puede incluir algún *waypoint* que modifique posteriormente la velocidad de vuelo.

El número de drones que se pueden volar se determina a partir del número de máquinas virtuales presentes, de misiones disponibles, y de velocidades definidas, y rellena el desplegable, permitiendo seleccionar cualquier número de drones entre 1 y el calculado.

El segundo grupo de parámetros incluye la IP y el puerto por los que el simulador se comunica con los drones ubicados en las máquinas virtuales. Debe ser una IP asignada a un adaptador de red del ordenador que corre el simulador, y dentro de la subred creada por Virtualbox para comunicar con las máquinas virtuales.

El tercer grupo de parámetros incluye la tasa de refresco de la pantalla y en número mínimo de píxeles que se tiene que mover un dron para que sea conveniente dibujar su movimiento. Estos parámetros se han incluido para permitir la posibilidad de minimizar el impacto que la interfaz gráfica produce en el rendimiento del ordenador. Esto permitirá ejecutar la aplicación en un rango de ordenadores muy amplio, tengan mucha o poca capacidad de cómputo.

El cuarto grupo de parámetros se limita a especificar si se utilizará la primera utilidad que le ha dado al simulador, el análisis del protocolo de evitación de colisiones.

El último grupo de parámetros especifica el modelo del enlace WiFi entre drones que se va a utilizar. Se puede utilizar uno de dos modelos:

- **Alcance fijo**. A modo binario, si se selecciona este modelo, los paquetes de datos llegarán siempre a destino si la distancia entre dos drones es inferior a la especificada, y no llegarán si la distancia es mayor.
- **Modelo 802.11a con antena de 5dBi**. Se ha definido de forma experimental. Define la probabilidad de que un paquete de datos llegue de un dron a otro en función de la distancia entre ambos.

En caso de utilizar el protocolo de evitación de colisiones, una vez cumplimentada la configuración, se muestra la figura 26 del apartado 8.6, con la configuración específica del protocolo de evitación de colisiones.

4.2 Arranque de máquinas virtuales y drones

Tras cumplimentar la configuración de la prueba comienza el arranque de las máquinas virtuales y se lanza la ventana principal del programa.

Se emplea un hilo de ejecución para supervisar el arranque de cada máquina virtual.

Cada hilo lanza una consola no visible [52], y dentro de la misma realiza los siguientes pasos:

a) Comprobación de si la máquina virtual ya está en marcha:

```
cd xxx
vagrant status
```

… donde xxx representa la ruta absoluta hasta la carpeta *ardupilot* correspondiente a la máquina virtual.

b) Si ya estaba en marcha la detiene:

```
vagrant halt
```

c) Por último, arranca la máquina virtual y cierra la consola:

```
vagrant up
```

Al depender de aplicaciones externas, todas las operaciones tienen definido un *timeout*, ante la posibilidad de que se produzca algún tipo de error. En caso de no completarse la operación correctamente, se informa al usuario y se detiene el simulador.

La salida de cada uno de los comandos es conocida. Esto permite determinar si el comando se ha completado correctamente, simplemente capturando e interpretando la salida estándar del mismo [54].

La máquina virtual construida funciona con Ubuntu 16.10, y aunque no tiene interfaz gráfica requiere bastantes recursos y tiempo durante el arranque.

Una vez ha terminado el arranque de la máquina virtual, el hilo se conecta por SSH a la misma con la librería Ganymed SSH-2 [53], con el usuario por defecto “Ubuntu” y la contraseña “vagrant”, definida durante la instalación según se detalla en el anexo III. Inmediatamente después se lanza el siguiente comando:

```
sudo /vagrant/Tools/autotest/sym_vehicle.py -v ArduCopter -j4 -l
     lat,lon,z,heading --out 192.168.56.254:14550; /bin/bash
```

… donde:

- -v ArduCopter. Indica que el vehículo es un multirrotor, no un aeroplano.
- -j4. Indica que se utilicen 4 hilos de ejecución para compilar, si es necesario.
- -l lat,lon,z,heading. Latitud y longitud en grados (coordenadas geográficas) donde el dron debe comenzar la prueba, cota relativa en metros, y orientación respecto al norte.

- --out 192.168.56.254:14550. IP y puerto hacia los que MAVProxy tiene que enviar la información de vuelo mediante mensajes MAVLink por UDP. En el apartado 7 se explica la comunicación drones-simulador con mayor detalle.

Los valores de las variables lat, lon, z y heading los determina el simulador a partir del primer *waypoint* de la misión introducida para cada dron.

Tal y como está diseñado el script de Python "sim_vehicle.py", en caso de cerrarse la conexión SSH, la ejecución del dron virtual se detendría inmediatamente. El comando adicional "/bin/bash" impide el cierre de la conexión [55].

Sería conveniente controlar MAVProxy a través de la conexión SSH, lo que permitiría lanzar comandos en su consola sin necesidad de enviar mensajes codificados con el protocolo MAVLink. Sin embargo, esto no es posible, pues analizando el script de Python se ha comprobado que MAVProxy obedece a interrupciones de teclado, pero no a la entrada estándar, lo que impide controlarlo externamente sin emular un teclado hardware. De cualquier modo, se intentó sin éxito acceder a la entrada estándar del programa mediante tuberías Linux [47] y también se probó a identificar el proceso para después escribir directamente en la entrada estándar en "/proc/numproceso/fd/0" [48]. Con la segunda opción, la aplicación incluso mostraba el texto introducido en su consola, pero no lo interpretaba por la causa ya indicada.

La consecuencia del problema expuesto es que no se conoce el estado en el que queda la emulación del dron tras el cierre de la aplicación. Por no hallar otra solución más eficiente, se decidió detener la máquina virtual tras cada prueba, para cerciorarse de que al iniciar una prueba los drones empiezan el proceso desde cero.

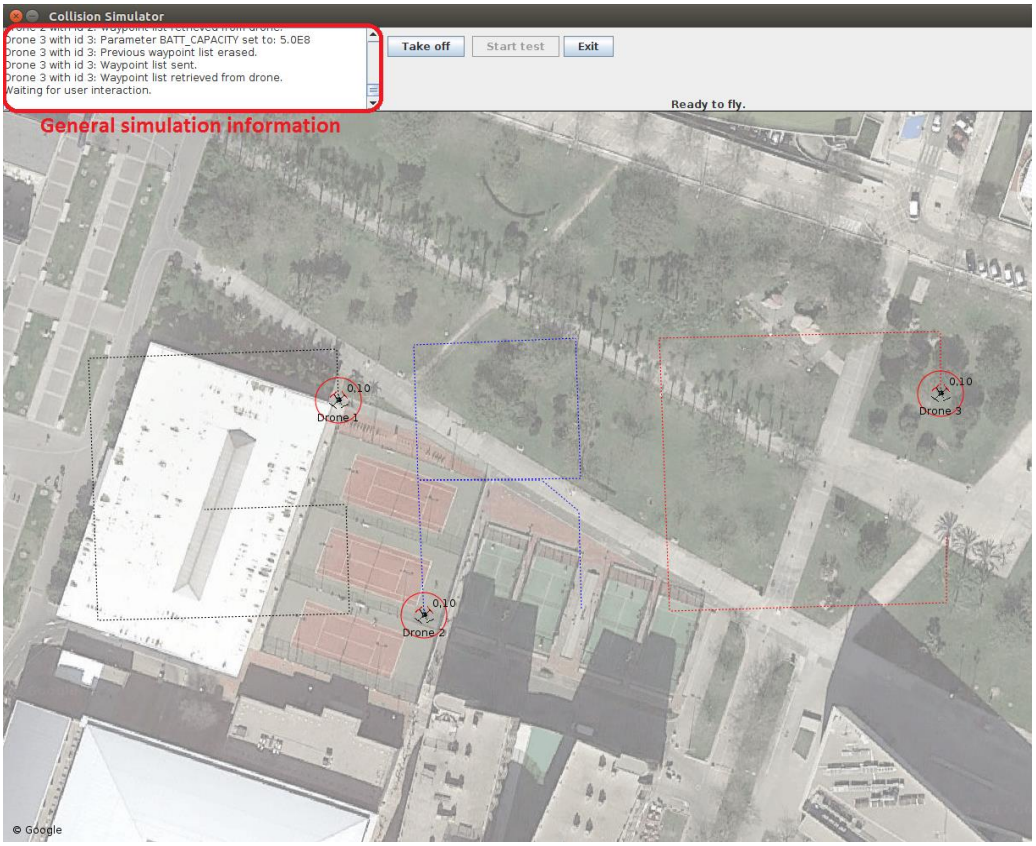

Figura 4: Proceso de arranque finalizado.

En la figura 4 se muestra el progreso de la aplicación una vez han arrancado todos los drones.

Como se observa, la ventana principal se subdivide en tres zonas.

En la esquina superior izquierda se observa el log de la aplicación. En él se muestra el progreso de la aplicación. Si se trata de información específica sobre un dron, se indica a qué dron corresponde, y el id asignado por el simulador.

A la derecha del log se hallan los campos con los que el usuario puede interaccionar con el simulador. Mientras éste arranca, los tres botones permanecen deshabilitados. Una vez los drones están listos para despegar, se habilita el primer botón y la posibilidad de cerrar la aplicación con el tercero. Cuando finaliza el despegue, se deshabilita el primer botón y se habilita el de inicio de la prueba. Por último, el botón para salir de la aplicación es el único activado mientras dura la prueba.

La mayor parte de la ventana está ocupada por la vista aérea de la zona por donde se moverán los drones. La ventana en sí siempre se abre maximizada y no permite modificar su tamaño. Esto se ha definido así para maximizar esta área y facilitar la visualización del proceso.

Junto a cada dron se muestra su identificador y la altitud a la que se halla. El círculo sobre el dron representa el área de seguridad alrededor del mismo. Si otro dron entra en él, el simulador considera que se ha producido una colisión entre los drones, los aterriza y da por finalizada la prueba.

En cuanto los drones están listos, se les envían dos órdenes:

- **Batería**. Se modifica la capacidad a una cantidad absurdamente elevada (500.000.000 mA) con el objeto de que la autonomía del dron virtual no influya en la prueba.
- **Misión**. Se borra la misión que el dron pudiese tener almacenada y se carga en el mismo la misión obtenida desde fichero. Previamente, se añade la orden de aterrizaje al final de la lista de *waypoints*. Cada tramo entre *waypoints* se dibuja con una línea discontinua.

Una vez los drones están ubicados y se ha cargado la misión prevista para cada uno, se puede definir el rectángulo que los envuelve. Tras calcular la correspondencia entre coordenadas geográficas, coordenadas UTM y píxeles de pantalla, se descarga un mosaico de imágenes de Google para formar el mapa de fondo. Entre otras tareas, es necesario atenuar el brillo, re-escalar, rotar y trasladar cada imagen. En el anexo IV se detalla el proceso seguido.

4.3 Despegue

En la figura 5 se observa cómo progresa el despegue.

El proceso no concluye hasta que todos los drones han alcanzado cota suficiente para realizar la prueba. La cota de vuelo mínima se ha establecido en 5 m, en una variable interna.

Además del círculo de seguridad, ahora aparece un segundo círculo concéntrico. Si otro dron entra dentro del mismo, se considera que hay riesgo de colisión y es necesario activar el protocolo de evitación de colisiones. El proceso se explica con detalle en el apartado 8.

Cuando se carga la misión desde un fichero kml de Google Earth se utiliza esta cota de vuelo, pues este tipo de ficheros no tiene información de elevación sobre el terreno.

Sin embargo, si la misión se carga desde un fichero estándar de QGroundControl, se utiliza la cota incluida en los *waypoints*, o la referencia mínima ya indicada en caso de que la cota del fichero sea inferior.

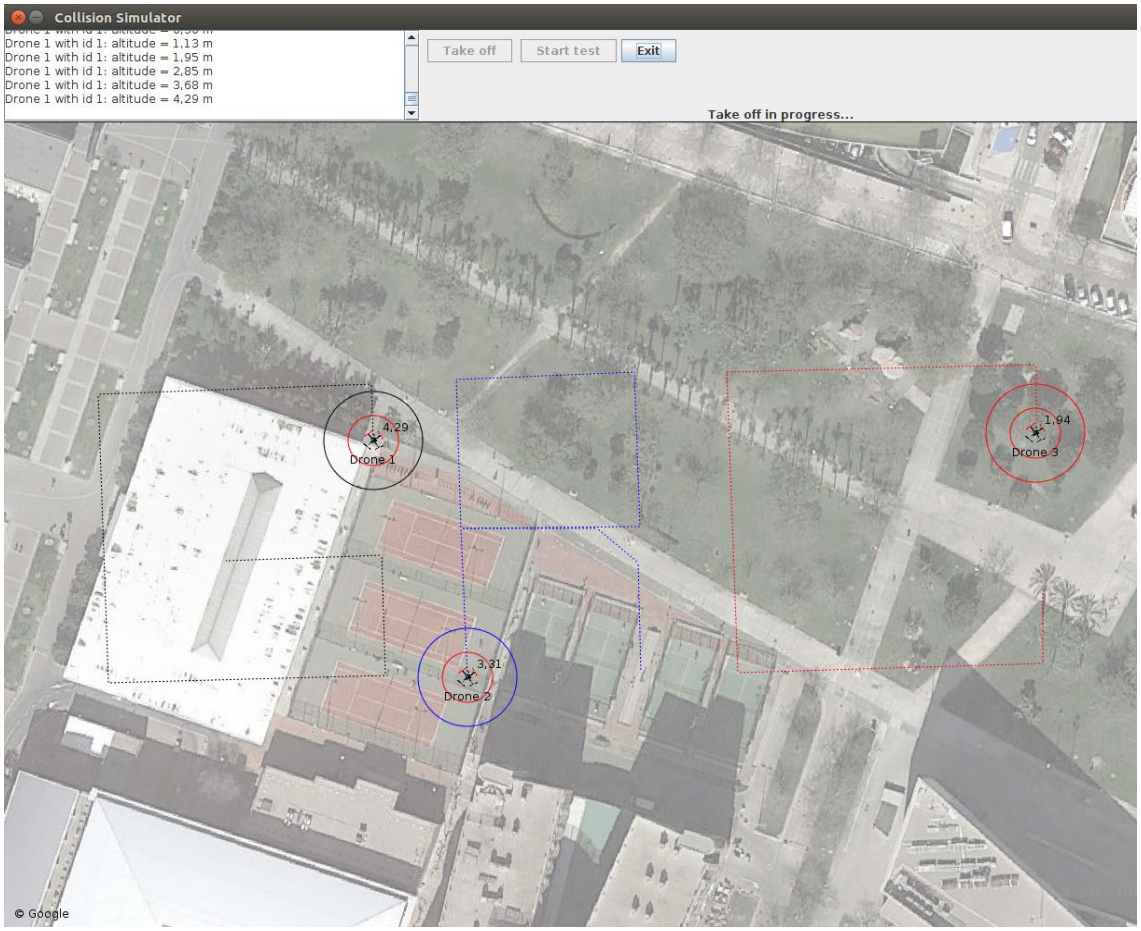

Figura 5: Despegue en progreso.

El control se devuelve al usuario cuando todos los drones han alcanzado el 95% de la cota a la que se deben ubicar.

4.4 Prueba

En cuanto el usuario inicia la prueba, los drones empiezan a moverse siguiendo la misión planificada, como se observa en la figura 6.

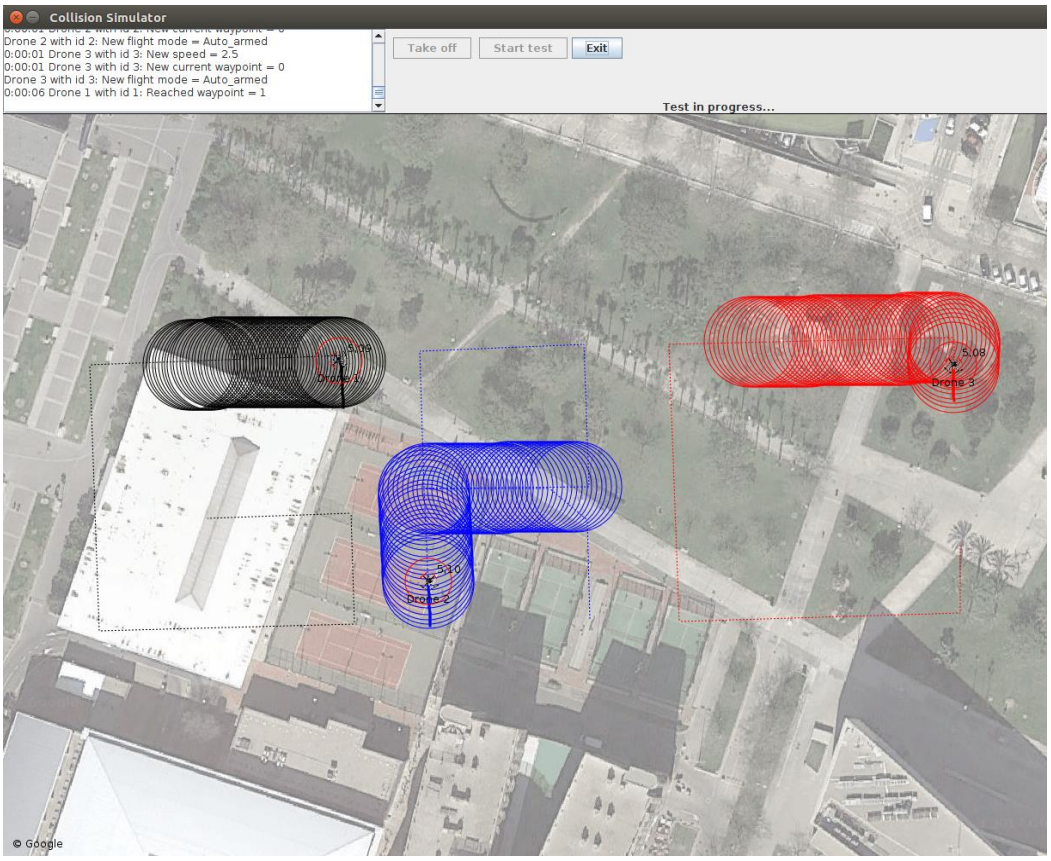

Figura 6: Prueba en progreso.

La banda de círculos concéntricos representa el área de seguridad contemplada por el protocolo de evitación de colisiones, y su cálculo se explica con detalle en el apartado 8.

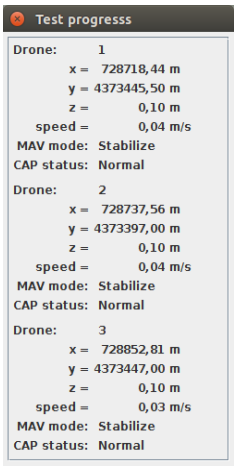

Figura 7: Diálogo de progreso de la prueba.

Además de la ventana principal, también se muestra el cuadro de diálogo de la figura 7. En él, se refleja en tiempo real diversos parámetros de cada dron, como la posición, velocidad, el modo de vuelo y el estado del protocolo de evitación de colisiones en el que se encuentra en ese momento.

A medida que la prueba progresa, los drones van alcanzando *waypoints*. Este suceso, y los cambios en el modo de vuelo, se reflejan en el log de la parte superior de la ventana principal, precedidos del instante de tiempo desde el inicio de la prueba.

4.5 Fin de la prueba

La prueba termina cuando todos los drones han llegado al último *waypoint* de su misión y aterrizan.

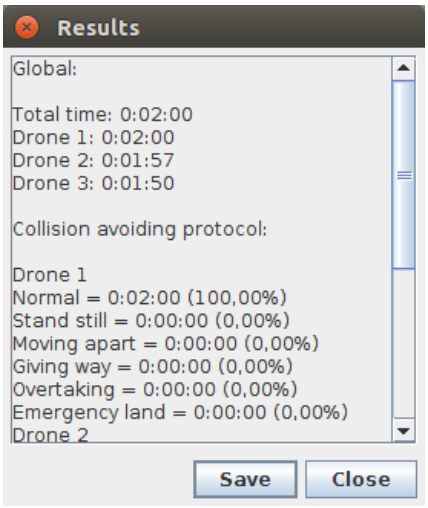

Figura 8: Diálogo de resultados de la prueba.

En ese momento, aparece el cuadro de diálogo de la figura 8, donde se detalla el tiempo de vuelo de cada dron y el tiempo que ha estado en cada uno de los distintos estados del protocolo de evitación de colisiones.

Esta información se puede guardar en un fichero de texto para su posterior análisis, o se puede descartar.

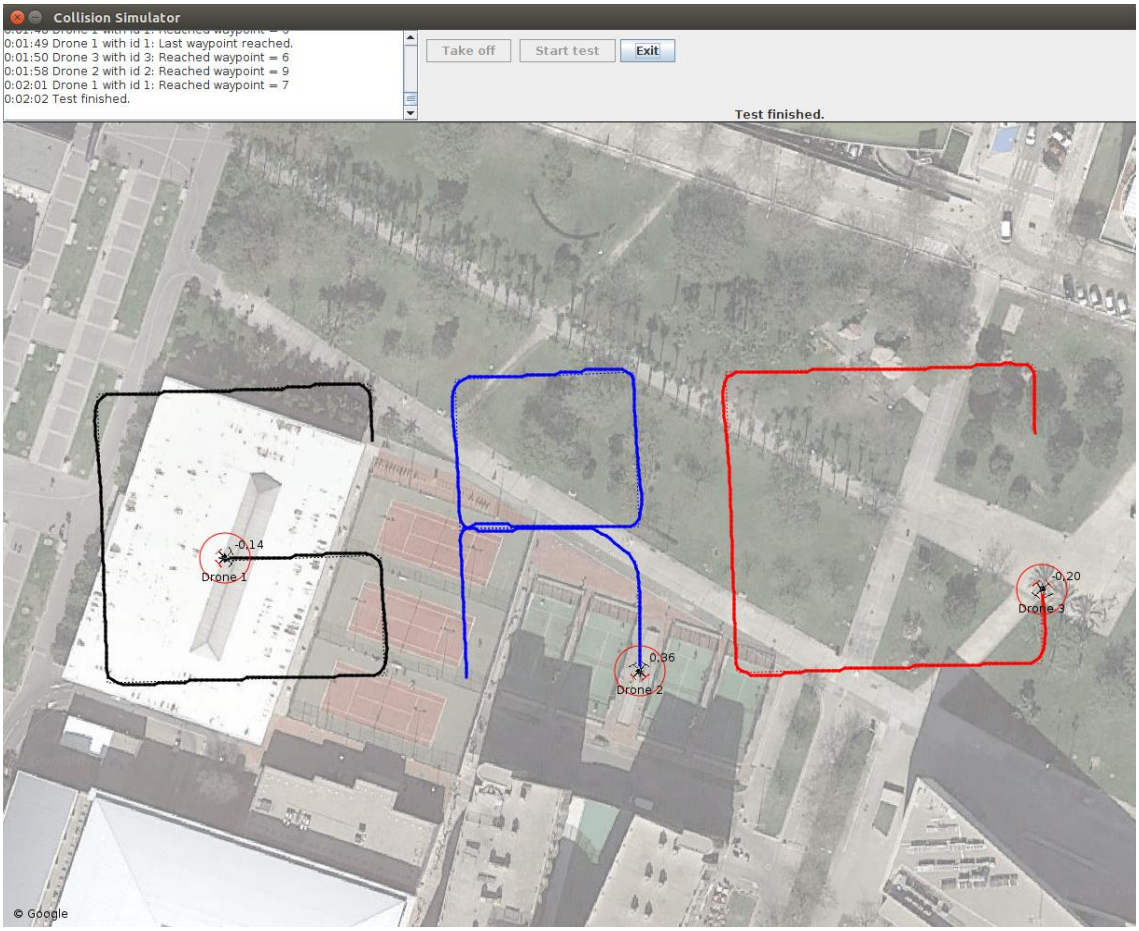

Figura 9: Prueba finalizada.

En la figura 9 se muestra la ventana principal cuando la prueba ha terminado.

Se observa que los drones han recorrido el trazado previsto en la misión, y que el botón para cerrar la aplicación es lo único habilitado, pues se puede realizar una única prueba cada vez que se arrancan las máquinas virtuales.

4.6 Cierre de la aplicación

En la figura 10 se muestra el proceso de cierre de la aplicación.

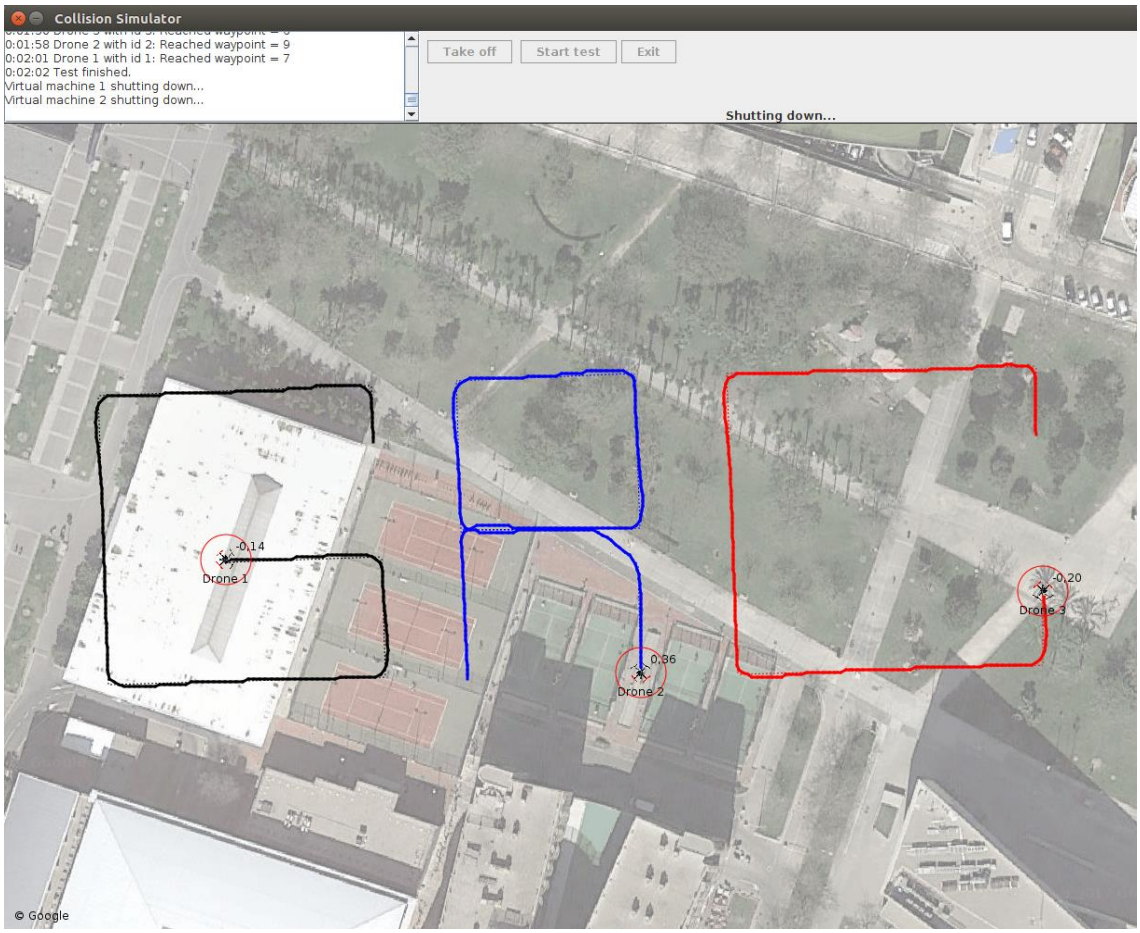

Figura 10: Cierre del simulador.

El simulador lanza una consola oculta en la que ejecuta, para cada dron, la orden:

```
vagrant halt
```

Si el cierre de alguna de las máquinas virtuales no ha sido satisfactorio, se avisa al usuario antes de cerrar definitivamente la aplicación.

Como ya se ha comentado, tras cada prueba se detienen las máquinas virtuales utilizadas por necesidad de controlar el estado en el que se encuentra cada dron virtual al iniciar una nueva prueba. En cualquier caso, como se indica en el apartado 4.2, antes de arrancar una máquina virtual se comprueba si ya estaba en marcha, y se detiene si es necesario.

5 Arquitectura de la aplicación

El simulador permite utilizar un número de drones indefinido entre 1 y 253, por lo que será necesario arrancar un número de hilos de ejecución distinto en función de dicho número.

La limitación a 253 drones se debe a que se utiliza IPv4 con máscara de red de 24 bits, y se utilizan las tres IPs faltantes para la dirección de subred, para *broadcast* y para el host principal, donde se hospeda el simulador. Se sugiere que la IP del host termine en 254 para que los drones estén numerados a partir del 1, aunque no es condición indispensable.

Se considera que 253 drones son más que suficientes para prácticamente cualquier aplicación. Es más, ninguna máquina moderna es capaz de ejecutar simultáneamente tantos drones, como se verá en el apartado 9.

En la figura 11 se observa la estructura interna de la aplicación.

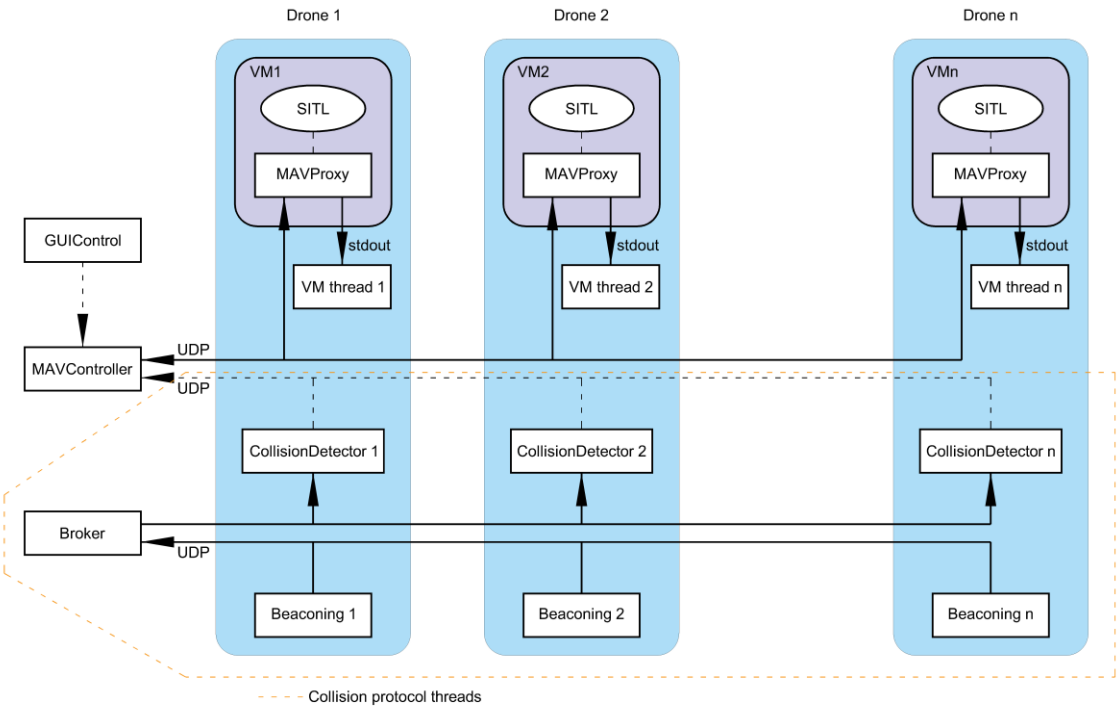

Figura 11: Arquitectura del simulador.

Para cada dron hay una máquina virtual y tres hilos de ejecución.

En la máquina virtual hay una instancia de SITL, para simular el dron, y una instancia de MAVProxy, aplicación que permite controlar el dron mediante consola (no accesible) o también con mensajes de tipo MAVLink (por UDP).

El hilo de ejecución *"VM thread x"* arranca la máquina virtual, conecta por SSH, y arranca SITL y MAVProxy.

Los hilos de ejecución *"CollisionDetector x"* y *"Beaconing x"* correspondientes a cada dron forman parte del protocolo de evitación de colisiones.

El hilo *"MAVController"* envía órdenes a los drones por UDP, y recibe por el mismo enlace información de vuelo y la respuesta a las órdenes. Este hilo es el motor de la aplicación, pues es el encargado del control de los drones, propósito principal del simulador.

Tanto la propia aplicación como el usuario pueden enviar órdenes al hilo *"MAVController"* desde el hilo *"GUIControl"*, para modificar el comportamiento de los drones.

La ejecución de la simulación sobre Linux tiene una singularidad. La interfaz de red del host real que está conectada a los drones permanece inhabilitada hasta que arranca al menos una máquina virtual. Por este motivo, no se puede lanzar el hilo *"MAVController"* hasta que al menos una máquina esté en marcha y, por lo tanto, la red esté disponible. Esto se debe a que dicho hilo tiene que escuchar en una IP y puerto dentro de la red, y se produce un error si ningún interfaz de red dispone de esa IP.

En lo que respecta al protocolo de evitación de colisiones, cada dron utiliza el hilo *"Beaconing x"* para enviar periódicamente información sobre su estado de vuelo al hilo *"Broker"*, proceso que simula el envío *broadcast*, que se deberá realizar con drones reales.

El hilo *"Broker"* determina los drones a los que debe llegar el paquete de datos, y pasa la información al hilo *"CollisionDetector x"* de cada uno de ellos, utilizando para ello el modelo del enlace WiFi definido en la configuración de la prueba, es decir, determina qué drones están dentro del alcance del dron emisor. Además, este hilo también comprueba si el algoritmo ha fallado porque dos drones han colisionado, en cuyo caso los aterriza e informa al usuario.

El hilo *"CollisionDetector x"* simula la recepción en un dron del paquete de datos enviado por otro dron, e implementa el algoritmo de evitación de colisiones. En caso de riesgo de colisión, adopta las medidas necesarias para evitar que se produzca.

La aplicación ha sido diseñada de tal modo que tanto el simulador como las máquinas virtuales se ejecuten en la misma máquina. Aunque teóricamente puede gestionar hasta 253 drones a la vez, en la realidad no es así, pues ningún ordenador actual puede ejecutar tantas máquinas virtuales a la vez.

En versiones futuras se podría implementar una versión distribuida del simulador, con un servidor que contendría la interfaz gráfica y los hilos *"GUIControl"*, *"MAVController"* y *"Broker"*, y clientes que contendrían una máquina virtual y los hilos correspondientes a un dron. En esta solución, los clientes se loguearían al servidor desde otras máquinas de la misma red, lo que mejoraría la escalabilidad del simulador, permitiendo la ejecución simultánea de un número muy elevado de drones, probablemente más allá de los 253.

Por otro lado, los hilos relacionados con el protocolo de evitación de colisiones no arrancan hasta que todos los drones tienen coordenadas gps, pues el hilo *"Beaconing x"* intentaría enviar las coordenadas de vuelo cuando todavía no se conoce la ubicación del dron.

6 Modelado del enlace WiFi entre drones

El medio inalámbrico no proporciona la misma fiabilidad que el cableado, por lo que los paquetes de datos transmitidos entre drones no siempre llegan a su destino. El simulador tiene en cuenta esta eventualidad, y para ello ha sido necesario modelar la tasa de pérdida de paquetes de datos, en función de la distancia entre el dron emisor y el dron receptor del paquete.

Para realizar el modelado se ha partido de la información disponible en dos papers preparados anteriormente por este mismo autor y otros.

En [8] se presenta la aplicación *Dronning*, desarrollada por este mismo autor para el Trabajo Fin de Grado titulado "*Desarrollo, evaluación y modelado de un sistema de comunicaciones para multicópteros*" presentado en la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia durante el curso 2015-2016. Se trata de una herramienta de investigación que permite medir la calidad del enlace de comunicación entre dos drones, conectados mediante una red ad-hoc basada en WiFi. Para ello se integra una Raspberry Pi 3 en los drones, y se utiliza una aplicación distribuida que mide la pérdida de paquetes de datos que se produce durante la transmisión, registrando parámetros de vuelo del dron y la distancia entre los drones. La herramienta va acompañada de scripts que permiten realizar análisis estadísticos de los datos recolectados, así como generar diversas gráficas de interés.

En [9] se muestran los resultados obtenidos con la aplicación *Dronning* en la banda WiFi de los 2,4 GHz. Además, se llega a la conclusión que la banda de los 2,4 GHz es totalmente inadecuada, pues la mayoría de los mandos de control remoto para drones funcionan en dicha banda y con la técnica de salto de frecuencia *Advanced Continuous Channel Shifting* [10], que es muy agresiva, pues ocupa toda la banda y provoca colisiones masivas con los paquetes de datos. Por lo tanto, el alcance de la señal en esta banda de frecuencias es muy reducido mientras se utilizan mandos de control remoto. Para modelar la tasa de pérdida de paquetes mientras no se usan mandos de control remoto, sería necesario desarrollar una nueva aplicación para medir la pérdida de paquetes que se produce al realizar vuelos planificados mediante misiones. La magnitud y complejidad de dicha tarea se escapa del alcance del presente trabajo, por lo que esta opción se ha descartado.

La solución obvia es cambiar la comunicación entre drones a la banda de los 5 GHz. Como no hay información disponible al respecto, ha sido necesario tomar datos de campo para modelar la tasa de pérdida de paquetes en función de la distancia.

Para ello se han utilizado dos drones y un ordenador portátil. En el anexo I se muestra información sobre los drones empleados, y en el anexo II se detallan los pasos necesarios para configurarlos.

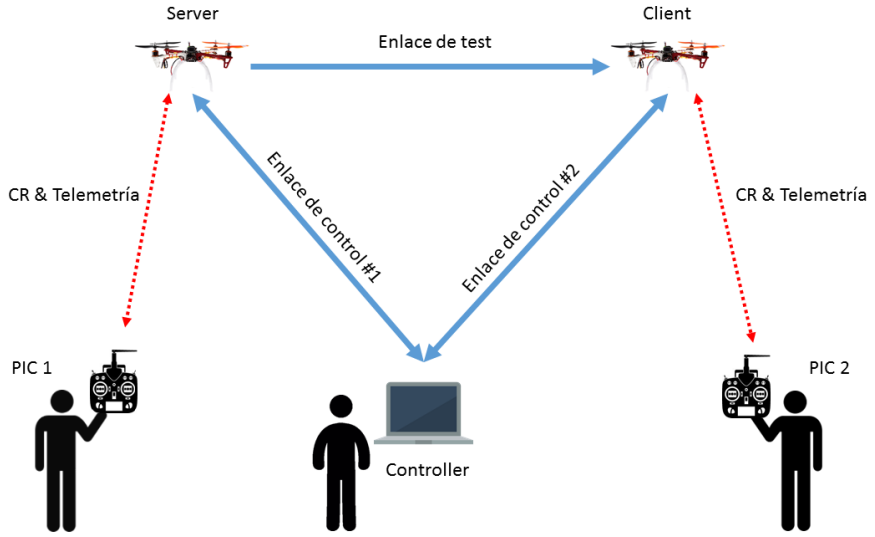

Figura 12: Toma de datos con la aplicación *Dronning*.

En la figura 12 se observa que, para realizar tomas de datos, hace falta un mínimo de tres personas, dos pilotos para controlar los drones, y una para establecer los parámetros de la prueba en el ordenador portátil, que ejerce el rol *controller*.

La normativa temporal actual para el vuelo de drones [11], establecida por la Agencia Estatal de Seguridad Aérea (AESA) [12], es muy restrictiva en cuanto a las condiciones que se deben cumplir para poder volar un dron:

- El piloto debe superar un curso de formación autorizado por la AESA para el rango de pesos en el que se incluye el dron que pretende volar. Además, debe acreditar los conocimientos suficientes para pilotar el modelo de dron concreto a utilizar.
- Está prohibido volar por la noche, en zona urbana, en espacio aéreo restringido (aeropuertos, …), sobre aglomeraciones de gente, en condiciones climatológicas adversas y a más de 120 m de altitud.

A lo largo de 2017 está previsto que salga una nueva ley [13] menos restrictiva que permitirá, entre otras cosas, volar dentro de espacio urbano o en espacio aéreo restringido, eso sí, cumpliendo condiciones especiales.

En el caso concreto que nos atañe, las pruebas se deben realizar en espacio abierto y con buenas condiciones climatológicas, para que la lluvia no dañe a los drones y el viento no influya significativamente en los resultados.

La batería de 3.300 mAh permite una duración útil de vuelo menor a 10 minutos, lo que condiciona considerablemente la planificación de las tomas de datos, que deben realizarse en serie reduciendo al mínimo posible el tiempo empleado entre tomas para modificar adecuadamente las condiciones de la misma. Por este motivo, antes de salir a campo, se debe planificar detalladamente las tomas de datos que se desea realizar.

Considerando todos estos condicionantes, se decidió medir la tasa de pérdida de paquetes de datos variando la distancia entre los drones directamente apoyados cada uno sobre un vehículo, a una altitud relativa sobre el suelo de 1,6 m. Los paquetes se enviaron a una tasa de 50 por segundo durante 60 segundos, y con un tamaño de 1500 bytes equivalente a la MTU máxima de Ethernet.

La medición fue dinámica, desplazando un vehículo respecto al otro sobre un vial en línea recta y sin cambios de rasante, acumulando los paquetes recibidos y perdidos en tramos de 50 m, pasa así calcular la tasa.

En la figura 13 se muestran los resultados. La señal tiene un alcance muy elevado, con una tasa de pérdidas inferior al 50% a 850 m de distancia. El siguiente polinomio de grado dos es el que mejor se ajusta a los datos disponibles, pasando por el origen:

$$y = 5.335 \cdot 10^{-7} \cdot x^2 + 3.395 \cdot 10^{-5} \cdot x$$

La señal es sobradamente buena para evitar la colisión entre dos drones, pues la probabilidad de que no se detecten antes de colisionar es ínfima, siempre y cuando se transmitan paquetes de datos con la suficiente frecuencia (la opción por defecto es cada 200 ms).

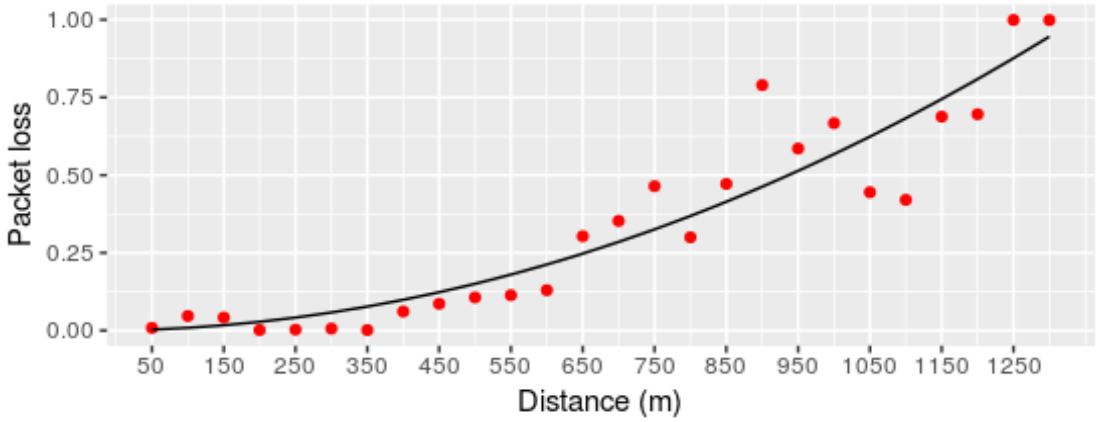

Figura 13: Modelo del enlace WiFi en el canal 36 (5,18 GHz) con antena de 5 dBi.

7 Control de los drones

La comunicación con los drones es el apartado que mayor complejidad ha supuesto, ya que ha sido necesario analizar, y en algunos casos deducir, los protocolos a cumplir para dar órdenes a los drones. Parte de estos protocolos está documentada, como el envío y recepción de la misión [14], mientras que otros no lo están.

7.1 Protocolo MAVLink

El primer punto a resolver es la forma de comunicarse con el dron virtual.

Para ello se utiliza el protocolo MAVLink, utilizado actualmente por SITL y por cualquier controladora de dron real.

En un dron real, la comunicación con la controladora se realiza por puerto serie.

En la simulación, para comunicarse con SITL se utiliza el programa MAVProxy, que se conecta a SITL por TCP. MAVProxy permite enviar órdenes mediante su consola, y también reenvía los comandos en formato MAVLink recibidos a través de la conexión con otras aplicaciones, tanto por UDP como por TCP. En la solución desarrollada, se envían mensajes MAVLink por UDP.

El protocolo MAVLink empaqueta en una trama los datos de control y de información, para la transmisión entre dispositivos. Está implementado a través de una librería muy ligera implementada en C++ y portada a otros lenguajes como Python y Java.

En la actualidad existen dos versiones, la 0.9 y la 1.0, aunque está previsto el lanzamiento de una versión 1.1, compatible con 1.0 que añadirá nuevos tipos de mensaje [15].

Además de los mensajes ya implementados en el estándar [43], el desarrollador puede definir sus propios mensajes en formato XML [16].

1B	1B	1B	1B	1B	1B	nB	1B
SOF	PLL	PS	SID	CID	MID	Payload	CRC

Figura 14: Estructura de un mensaje MAVLink v1.0.

…donde:

SOF: *Start of frame* (0xFE). Carácter especial necesario para identificar el comienzo del mensaje cuando la transmisión es por el puerto serie.
PLL: *Payload length*. Longitud n de los datos incluidos en el mensaje.
PS: *Packet sequence*. Número de secuencia del mensaje desde el inicio de la comunicación entre la Pixhawk y cualquier otro dispositivo.
SID: *System ID*. Identificación del sistema emisor.

CID: *Component ID*. Identificación del componente del sistema que envía el mensaje.
MID: *Message ID*. Identificador del tipo de mensaje según el protocolo.
Payload: Datos incluidos en el mensaje.
CRC: *Checksum* del paquete para evitar errores de transmisión.

Como se observa en la figura 14, los mensajes MAVLink tienen longitud variable dependiendo del contenido o *payload* de un mensaje.

Cuando se recibe un mensaje, el dispositivo correspondiente decodifica el *payload* considerando que el contenido se ajusta al mensaje del tipo especificado en el campo *Message ID*.

El *payload* del mensaje puede contener distintos tipos de datos, que requieren una longitud en bytes distinta para su almacenamiento. Para evitar problemas de alineación de los datos, éstos se reordenan antes de su codificación [17].

El CRC permite ignorar los mensajes con errores en la transmisión. En su cálculo, al final del mensaje, se incluye un byte calculado como *checksum* del XML que define el tipo de mensaje, lo que evita que el origen y el destino utilicen distintas versiones del mismo tipo de mensaje. Esta mejora se introdujo en la versión 1.0 para impedir errores de interpretación cuando los dispositivos origen y destino utilizan distintas versiones del protocolo o de los mensajes.

El simulador envía y recibe los mensajes MAVLink mediante la compilación y uso de una librería. La librería MAVLink [18] dispone de una descripción de cada tipo de mensaje en ficheros XML, aunque está concebida para su uso en el lenguaje de programación C. Mediante una utilidad adicional [19] se puede compilar una librería alternativa para su uso con Java, utilizando para ello los mensajes definidos en la librería diseñada para C.

7.2 Acciones implementadas

El siguiente punto a resolver consiste en definir las órdenes que se deberán dar a los drones.

La controladora del dron también envía información de vuelo en tiempo real. Luego, también es necesario determinar los mensajes que interesa procesar para proporcionar información al simulador.

Teniendo en cuenta el funcionamiento de la aplicación descrito en el apartado 4, será necesario enviar las siguientes órdenes:

- **setParam(numDron, parámetro, valor)**
  Al arrancar el dron es conveniente aumentar el tamaño de la batería para que la autonomía de vuelo no afecte a la prueba. El parámetro que es necesario modificar se describe en [45].
- **setMode(numDron, modo)**
  El dron arranca en modo *"stabilize"*. Sin embargo, es necesario cambiar de modo de vuelo varias veces durante la prueba. Primero se pasa al modo *"guided"* durante el despegue, para iniciar la prueba se pasa a *"auto_armed"* y, por último, cuando los drones aterrizan se pasa automáticamente a "auto".

Además, durante la evitación de colisiones, es necesario pasar a modo *"loiter_armed"* para detener el dron en pleno vuelo, al modo *"guided_armed"* para apartarlo de la trayectoria del otro dron, si es necesario, al modo *"auto_armed"* para reanudar la misión, y al modo *"land_armed"* para aterrizar en caso de que el protocolo haya entrado en alguna clase de interbloqueo.
Los modos de vuelo pueden ser específicos de la implementación de cada dron. Por este motivo, fue necesario aplicar ingeniería inversa para determinar el valor de los parámetros *"base mode"* y *"custom mode"*, a partir de la respuesta recibida por la controladora de vuelo cuando se cambia el modo directamente desde la consola de MAVProxy, y a partir de la información obtenida de [49].

- **armEngines(numDron)**
  Función que arma los motores antes de despegar. Si transcurren más de 15 segundos entre el armado y el despegue, la controladora desarma los motores por motivos de seguridad.
- **doTakeOff(numDron)**
  Inicia el proceso de despegue y eleva el dron hasta una altitud relativa respecto al suelo, especificada previamente. En [50] se especifican los parámetros empleados en el mensaje MAVLink.
- **setSpeed(numDron, velocidad)**
  Cuando se inicia una prueba, lo primero que hace la aplicación es indicar a los drones que vuelen a la velocidad especificada por el usuario en la interfaz gráfica.
- **setCurrentWP(numDron, numWP)**
  Esta función indica a la controladora de vuelo que, cuando se le ordene seguir la misión programada, deberá comenzar dirigiéndose al *waypoint* indicado.
- **takeControl(numDron)**
  Cuando se utiliza un mando de control remoto, para detener una misión y asumir el control del dron, basta con cambiar el modo de vuelo a *"loiter_armed"*, como se ha indicado con anterioridad. Este modo de vuelo detiene el dron en las coordenadas (x, y) en las que se encuentra al recibir la orden. Sin embargo, en el simulador, si se cambia el modo de vuelo del dron éste se estrella, pues no hay ningún mando de control remoto que asuma el control del dron.
  La controladora de vuelo de un dron debe estar siempre calibrada de tal modo que, si se aplica un valor de 1500 a la potencia de los motores (*throttle*), debe quedarse estable en la altitud actual.
  Para evitar que se estrelle el dron, virtual o real, basta con indicar un *throttle* de 1500 justo antes de pasar al modo de vuelo *"loiter_armed"*.
- **moveDrone(numDron, coordendas_geográficas)**
  En caso de que el dron esté en la trayectoria de otro dron con prioridad de paso, es necesario que se haga a un lado para evitar la colisión. Esta función le indica el punto del espacio al que se tiene que mover.
  Para garantizar que el movimiento se ha completado, se incluyen dos bucles de espera: el primero para detectar que el dron ha iniciado el movimiento (adquiere velocidad), y el segundo para detectar que ha alcanzado el destino (la velocidad es casi nula).
  El comando MAVLink utilizado está incorrectamente documentado en el protocolo. Según se deduce de la función *"mission_item_send"* en [56] y [57], el

parámetro *"autocontinue"* debe tener el valor 2, cuando según el protocolo únicamente puede tener el valor 1 o 0 para indicar si el dron debe continuar o no al llegar al *waypoint* especificado. Sin embargo, contra toda lógica, se envía un mensaje que representa un *waypoint*, sin que éste haya sido solicitado por la controladora de vuelo, y con valores no coherentes con el protocolo. Se deduce que este tipo de mensaje es una ampliación, no muy coherente, de la versión anterior del protocolo.

- **clearWPList(numDron)**
  Función que borra la misión actual de la memoria de la controladora de vuelo.
- **sendFirstWPList(numDron, lista_de_waypoints)**
  Función que envía al dron la misión introducida durante la configuración de la prueba.
  Antes de enviar la misión se comprueba que todos los puntos por los que tendrá que pasar el dron tienen una altitud igual o superior a la mínima de vuelo.
  También se comprueba si el fichero especifica cuál debe ser el primer *waypoint* al que el dron debe dirigirse. En caso de que no se especifique, se considera que la misión debe comenzar en el primer waypoint encontrado en el fichero.
- **getWPList(numDron)**
  Consulta a la controladora de vuelo la misión actual.
  Se ha utilizado programación defensiva por medio de una comprobación que, en principio, no es necesaria. Inmediatamente después de enviar la misión al dron, ésta se recupera del mismo y se muestra en pantalla. En caso de que lo recibido por el dron no coincida con lo esperado por el usuario de la aplicación, éste podrá detectarlo observando en la pantalla la trayectoria prevista.

En caso de adaptar el simulador para cualquier otro uso distinto de la detección de colisiones, bastaría con llamar a estas funciones para controlar los drones virtuales. Todas las funciones comprueban si la operación ha sido ejecutada correctamente por la controladora de vuelo, por lo que el desarrollador puede considerar estos errores para implementar protocolos y funcionalidades con garantías de correcto funcionamiento.

Todas las funciones indicadas devuelven un booleano con el valor *"true"* si la operación ha sido correcta, y *"false"* en caso de haberse producido algún problema. Este enfoque facilita la programación, pues si se necesita encadenar varias órdenes se puede utilizar el siguiente esquema:

```
if ( orden1 && orden2 && orden3) { hacer algo; }
```

Se realizará una tarea final, solamente en caso de que todas las operaciones hayan concluido con éxito. Además, las órdenes solamente se ejecutan si las anteriores se han completado con éxito.

Este esquema se ha aplicado para definir funciones más complejas:

- La configuración inicial de cada dron incluye cuatro operaciones. Primero se aumenta el tamaño de la batería del dron, luego se borra la misión que pudiese tener, se le transfiere la misión cargada desde fichero, y por último se recupera dicha misión para mostrarla por pantalla.
- El despegue incluye tres operaciones. Primero se cambia el modo de vuelo a *"guided"*, luego se arman los motores, y por último se ordena despegar.

- El inicio de la prueba incluye también tres operaciones. Primero se fija la velocidad de vuelo, luego se indica cuál debe ser el primer *waypoint* de la misión, y por último se cambia el modo de vuelo a *"auto_armed"*, para que se inicie el movimiento.
- Si hay riesgo de colisión se detiene el dron en pleno vuelo. Para ello, primero se asume el control de la altitud del dron, y después se cambia al modo de vuelo *"loiter_armed"*, para que el dron quede estabilizado en las coordenadas actuales.

7.3 Máquina de estados finitos

Las funciones detalladas en el apartado anterior se limitan a cambiar el valor de una variable global que representa el estado actual de cada dron. Después, esperan a que el valor de dicha variable vuelva a cambiar, según sea la respuesta recibida de la controladora de vuelo.

De este modo, el comportamiento del dron se simplifica a una máquina de estados finitos.

El hilo de ejecución *"MAVController"* es el que se encarga de transmitir las órdenes a la controladora de vuelo mediante mensajes de tipo MAVLink. Cada vez que se recibe un paquete de datos de un dron comprueba si hay órdenes que enviarles. Como se recibe al menos un paquete de datos cada 100 ms desde la controladora, el envío de las órdenes es prácticamente inmediato.

Se han definido un total de 33 estados. Se trata de una cifra elevada, pues para cada orden transmitida al dron puede haber varios estados. Por ejemplo: orden solicitada, en espera de respuesta, respuesta correcta y respuesta con error.

En la figura 15 se muestra una versión simplificada de la máquina de estados finitos relativa a un dron.

Cada vez que se recibe un paquete datos desde el dron puede tratarse de información general (*"Data received"*) o de la respuesta a alguna orden enviada al mismo.

Si se trata de información de vuelo, se procesa y luego, en caso de que alguna de las funciones descritas en el apartado anterior haya solicitado el envío de alguna orden, ésta se envía y se cambia al estado correspondiente.

Si se trata de la respuesta a alguna orden, se procesa y se cambia de estado.

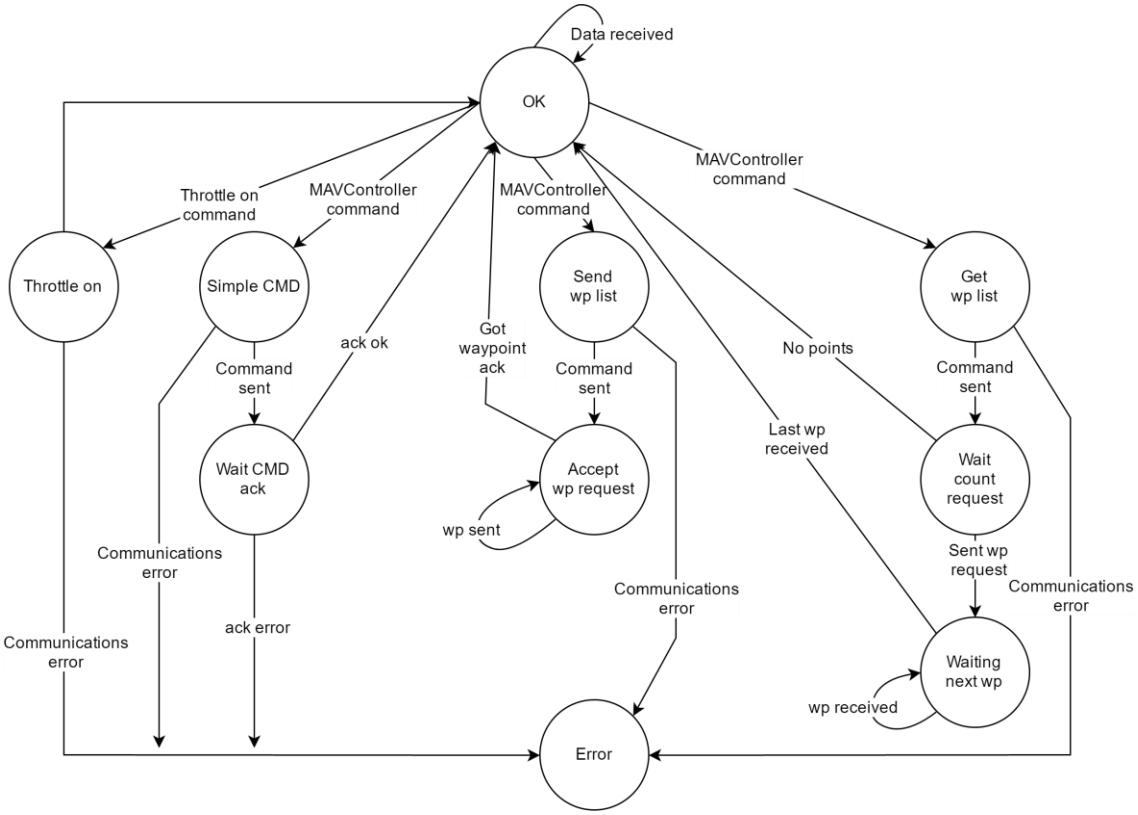

Figura 15: Máquina de estados finitos de un dron.

La mayoría de las órdenes se limitan a una interacción sencilla. Tras recibir la orden de enviar un mensaje (tipo *"Simple CMD"*), ésta se envía al dron. Si no se ha producido un error de comunicación, se espera la recepción de un ACK por parte de la controladora de vuelo.

Asumir el control de la altitud de vuelo (*"Throttle on"*) es una orden aún más básica, pues ni siquiera espera un ACK.

El proceso de enviar la misión al dron (*"Send wp list"*) implica varios pasos. Primero se envía el número de *waypoints* que forman la misión, a lo que la controladora reacciona solicitando uno a uno los *waypoints*, y el hilo los va suministrando hasta que la controladora envía un ACK confirmando que ya ha recibido todos los *waypoints*.

La recepción de la misión almacenada en el dron (*"Get wp list"*) comienza solicitando la lista de *waypoints*. La controladora envía por su parte un mensaje que indica el número de *waypoints* que forman la misión que almacena. Si el dron contiene una misión, es decir, el número de *waypoints* no es nulo, se solicitan uno a uno hasta tenerlos todos, y se envía a la controladora un ACK.

A continuación, se explica mejor el envío de órdenes a cada dron, expandiendo el diagrama de flujo anterior para estudiarlo con mayor detalle.

En la figura 16 se muestra el diagrama de flujo cuando se envía una orden para tomar el control de la altitud de vuelo. Como se observa, si se ha solicitado la orden *"Throttle on"*, se envía el comando a la controladora de vuelo. Si no ha habido error durante la transmisión se pasa directamente al estado *"ok"*, pues esta orden no espera la recepción de un ACK de la controladora de vuelo.

El estado *"ok"* es aquel en el que no hay ninguna orden siendo procesada. Solamente se reciben y procesan paquetes de datos con información de vuelo.

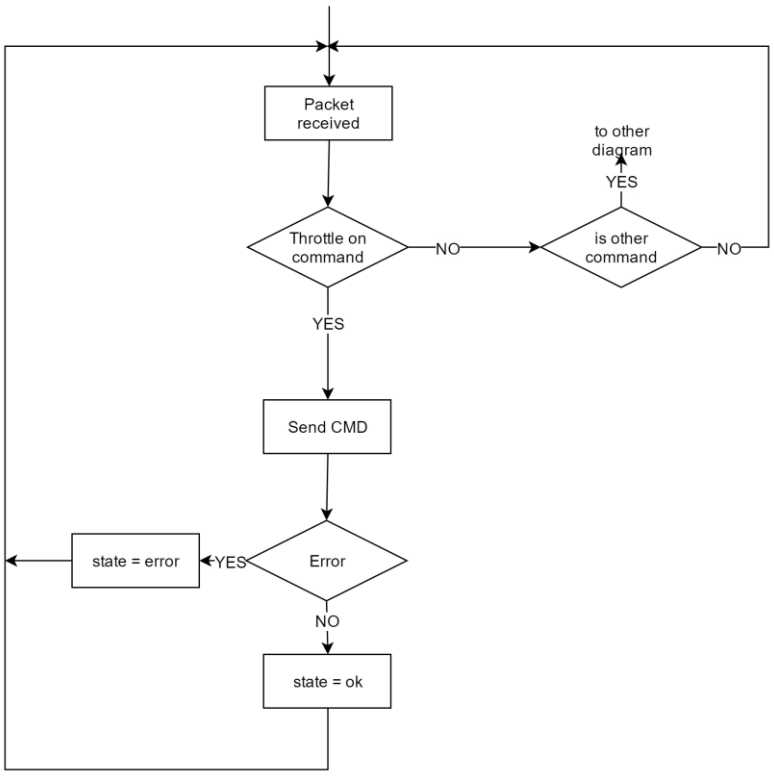

Figura 16: Detalle del envío de la orden *"Thottle on"*.

En la figura 17 se muestra en detalle el procesado de las órdenes sencillas (*"Simple CMD"*), con indicación de los estados por los que pasa el dron.

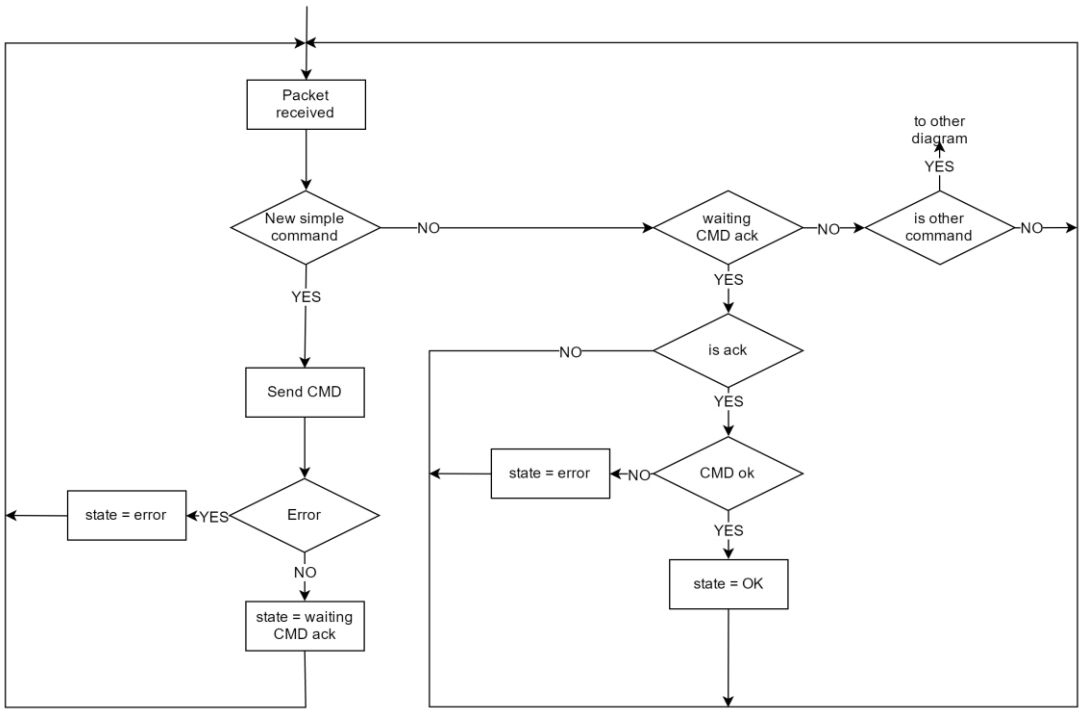

Figura 17: Detalle del envío de órdenes sencillas.

Las órdenes que se atienen a este esquema son:

- Modificación de parámetro del dron (ej. tamaño de batería).
- Cambio de modo de vuelo.
- Cambio de la velocidad de vuelo.
- Armado de motores.
- Orden de despegue.
- Definición del *waypoint* actual.
- Borrado de la misión almacenada en el dron.
- Acción de mover al dron a unas coordenadas.

En este caso, tras enviar la orden, se pasa a esperar el reconocimiento de la recepción de la orden, en el estado *"waiting CMD ack"*. Cuando se recibe dicho ACK, se comprueba si la controladora ha aceptado la orden y se pasa al estado *"ok"*, o si la orden ha sido rechazada y se pasa al estado *"error"*.

La figura 18 representa el diagrama de flujo que se sigue cuando se envía la misión al dron.

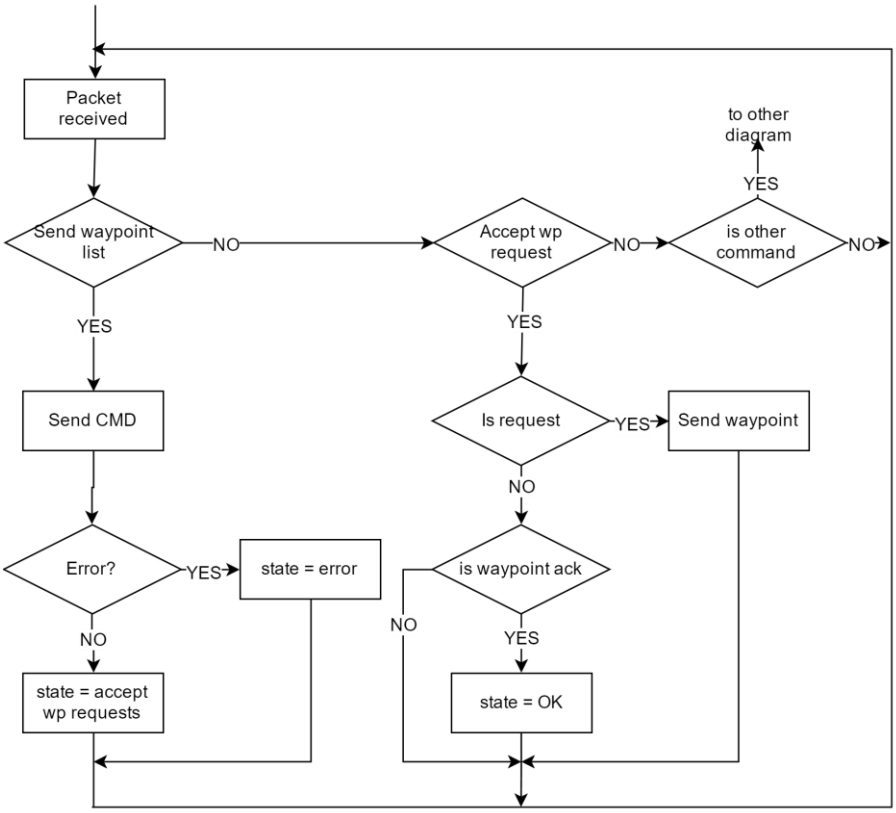

Figura 18: Detalle del envío de la misión al dron.

En el caso de que haya una orden de enviar la misión actual al dron (estado *"Send waypoint list"*), se envía la orden respectiva a la controladora, especificando el número de *waypoints* que forman parte de la misión. Si no se produce ningún error de comunicación, entonces se pasa al estado *"accept wp requests"*.

A partir de este momento, es la controladora del dron la que asume el control, solicitando los *waypoints* uno a uno. Cada vez que se recibe la petición de un *waypoint* determinado, éste se envía. El protocolo está definido así en el estándar [14] para que el emisor retransmita el mensaje con el *waypoint* en caso de que no llegue a destino.

Por último, si se sigue en el mismo estado y lo que se recibe es un *"waypoint ack"*, entonces significa que todos los *waypoints* están ya en el dron, y que se puede pasar al estado *"ok"*, finalizando el proceso.

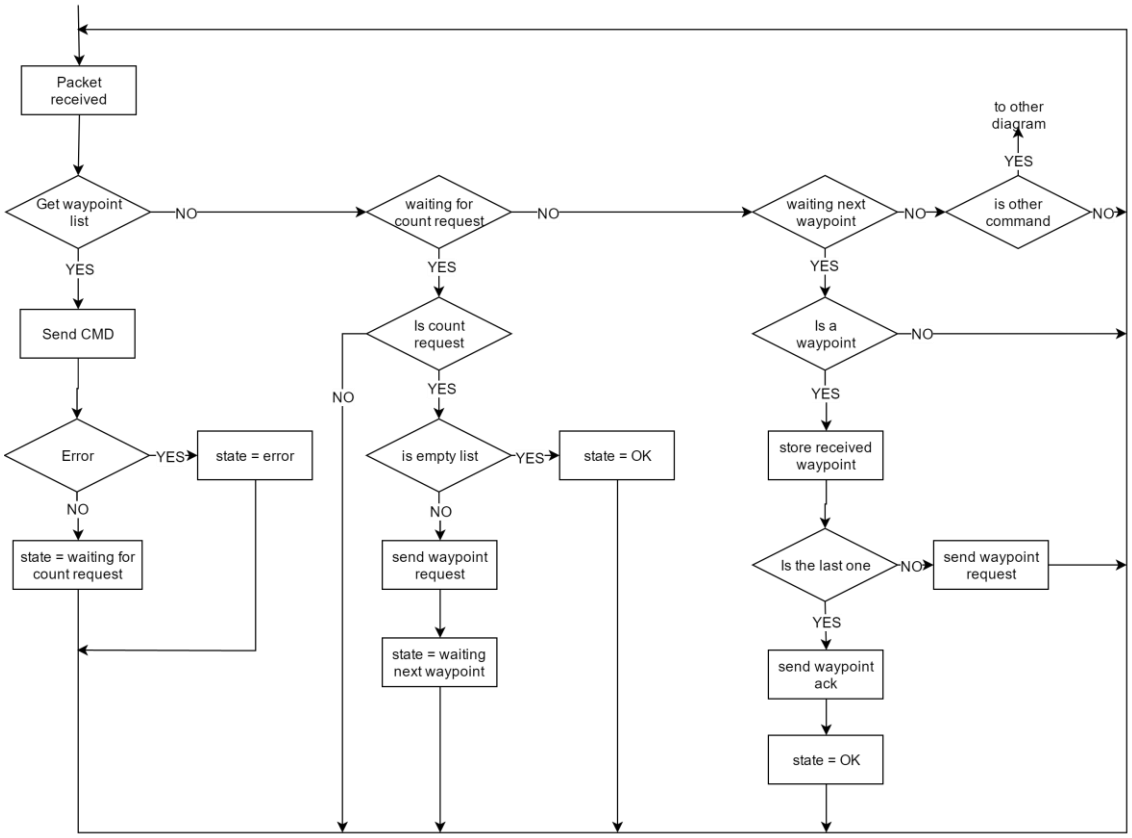

Figura 19: Detalle de la recepción de la misión contenida en el dron.

En caso de haber orden de leer la misión almacenada actualmente en el dron (estado *"Get waypoint list"*), se envía dicha petición al dron (figura 19). Si no hay error de transmisión se pasa al estado *"waiting for count request"*.

Será la controladora del dron la que envíe un mensaje indicando el número de puntos que forman la misión. Si el dron no contiene ninguna misión, es decir, no hay *waypoints*, entonces se pasa directamente al estado *"ok"*. En caso contrario, se solicita el primer *waypoint* de la lista y se pasa al estado *"waiting next waypoint"*.

En algún momento, el dron responderá con un mensaje con el *waypoint* solicitado. Será entonces cuando se guarde en una lista. Mientras queden *waypoints* por recibir, se enviará la solicitud del siguiente. En caso de haber recibido ya todos los *waypoints* que forman la misión, se envía a la controladora la confirmación y se pasa al estado *"ok"*.

7.4 Mensajes recibidos del dron

Además de las respuestas a las órdenes, el hilo de ejecución *"MAVController"* recibe mensajes con información de vuelo útil para el simulador. Entre todos los mensajes recibidos, se procesan los siguientes:

MAVLINK_MSG_ID_GLOBAL_POSITION_INT. Incluye información de posición del dron, así como la velocidad en los tres ejes cartesianos y la orientación en el plano horizontal.

MAVLINK_MSG_ID_HEARTBEAT. Este mensaje se envía periódicamente y contiene el modo de vuelo actual del dron.

MAVLINK_MSG_ID_EKF_STATUS_REPORT. Contiene un *flag* que indica el estado general de la controladora. Cuando el valor de este *flag* supera un umbral, significa que el GPS ya ha obtenido coordenadas válidas, y que por lo tanto ya se puede dibujar el dron en pantalla.

MAVLINK_MSG_ID_PARAM_VALUE. Cuando se cambia algún parámetro general del dron, como por ejemplo el tamaño de la batería, se recibe un mensaje de este tipo en el que se confirma el nuevo valor del parámetro.

MAVLINK_MSG_ID_MISSION_CURRENT. Se recibe un mensaje de este tipo cuando se cambia el *waypoint* hacia el que tiene que dirigirse el dron, y en el que se confirma el número de *waypoint* especificado en la orden.

MAVLINK_MSG_ID_MISSION_ITEM_REACHED. A medida que progresa la prueba se recibe un mensaje de este tipo cada vez que el dron alcanza un *waypoint*. Esto permite mostrar en el log general de la pantalla el progreso del dron.

MAVLINK_MSG_ID_MISSION_COUNT. Es el mensaje que la controladora de vuelo envía al simulador cuando éste le ha solicitado la misión que tiene almacenada. De esta forma, la aplicación averigua cuántos *waypoints* le tiene que solicitar al dron.

MAVLINK_MSG_ID_MISSION_ITEM. El mensaje contiene toda la información relativa a un *waypoint*, y se recibe cada vez que el simulador solicita uno al dron.

MAVLINK_MSG_ID_MISSION_REQUEST. Contiene el identificador del *waypoint* que el dron está solicitando al simulador. Esta situación se da cuando se desea transferir una misión al dron.

MAVLINK_MSG_ID_MISSION_ACK. Es un mensaje que confirma la finalización de una transacción relacionada con *waypoints*. En concreto, se envía o recibe al enviar una misión, al recibirla, al eliminar la misión almacenada en el dron, o al ordenar al dron que se mueva a unas determinadas coordenadas.

MAVLINK_MSG_ID_COMMAND_ACK. La controladora de vuelo envía este mensaje para confirmar la recepción de la orden cuando se trata de comandos sencillos (*"Simple CMD"* en la figura 15). Devolverá un 0 en caso de aceptar la orden, y un número del 1 al 4 en caso de error o de no aceptarla por incumplir algún protocolo de comunicación, es decir, cuando la orden se haya enviado sin cumplir la secuencia de órdenes correcta.

8 Protocolo de evitación de colisiones

Una vez desarrollada la estructura básica mostrada en la mitad superior de la figura 11 del apartado 5, se han implementado tres tipos de hilo de ejecución, para dar soporte a un algoritmo de evitación de colisiones.

La idea subyacente es sencilla, y consiste en que cada dron virtual utilice un hilo (*"Beaconing x"*) para enviar periódicamente una trama de datos con la lista de posiciones en las que se encontrará durante varios segundos en el futuro. Se trata de una secuencia de puntos superpuestos a la traza que debe seguir el dron, y con una separación variable en función de la velocidad actual de vuelo. Esa información la recibe un hilo común a todos los drones (*"Broker"*) que simula la dirección de *broadcast*, ya que, en el mundo físico, los drones enviarán así la trama. Esta característica permitirá que cualquier dron que se acerque se una al protocolo.

Este segundo hilo analiza si los otros drones lanzados por el simulador están dentro del alcance del dron emisor. Si es así, les pasa la información, como si la hubiesen recibido ellos mismos. Además, comprueba periódicamente si los drones están tan cerca entre sí como para considerar que se ha producido una colisión, es decir, si el protocolo de evitación de colisiones ha fallado.

El tercer hilo es el que recibe la información (*"CollisionDetector x"*) y se encarga de aplicar el protocolo de evitación de colisiones. Si lo considera necesario, detendrá los drones y moverá un dron para permitir el paso de un dron con mayor prioridad de paso.

8.1 Hilo Beaconing

Cada dron envía las tramas periódicamente mediante un hilo de ejecución independiente, lo que permite desacoplar la tasa de envío de la ejecución del protocolo.

La trama se envía por UDP a un puerto en el interfaz de *loopback*, dado que debe ser interpretada por el hilo *"Broker"*, dentro de la misma aplicación.

Sería conveniente explicar el formato de la trama en este apartado, ya que es enviada por este hilo. No obstante, dado que los campos que la forman están relacionados con el protocolo, se considera necesario hacerlo después de explicar el protocolo de evitación de colisiones, en el apartado 8.4.

8.2 Hilo Broker

Como ya se ha indicado, éste es el hilo encargado de recibir las tramas enviadas por todos los drones, con objeto de simular el envío por *broadcast*.

Una vez recibida e interpretada la trama, se analiza a qué drones debe ser enviada.

Si un dron está dentro del alcance de la señal inalámbrica emitida por el dron origen de la trama, entonces se le entrega en una estructura de datos concurrente (*"AtomicReference"*), para evitar problemas de acceso a variables compartidas desde hilos distintos. En caso de haber recibido previamente una trama del mismo dron, se sustituye con la nueva.

Se está tratando con drones en movimiento. No tiene sentido estar analizando la información incluida en una trama cuando el dron que la ha enviado ya ha aterrizado o se encuentra lejos de cualquier otro dron. Por este motivo, se ha definido un tiempo de caducidad para las tramas recibidas. Cada vez que un dron recibe una trama, comprueba primero si tiene una trama obsoleta proveniente de cualquier otro dron y la elimina, y sólo después guarda la trama que ha recibido.

En cuanto al modelo del enlace WiFi entre los drones, el usuario del simulador dispone de dos opciones, como se explica en el apartado 4.1:

- **Alcance fijo**. Si la distancia al dron origen de la trama supera un umbral, no se recibe en el dron destino.
- **Modelo 802.11a con antena de 5dBi**. Refleja una tasa de pérdida de paquetes de datos en función de la distancia entre los drones, obtenida experimentalmente.

En caso de que el modelo utilizado arroje una probabilidad adecuada de llegar al destino, entonces se pasa la trama al dron correspondiente.

Durante el desarrollo del protocolo de evitación de colisiones es fácil encontrarse con situaciones en las que dos drones colisionen, debido a que el protocolo pueda no funcionar correctamente, ya que no está terminado. Por esa razón, es conveniente tener en cuenta esta eventualidad e informar al usuario si se produce una colisión.

Por otro lado, hay un hilo *"Beaconing x"* y uno *"CollisionDetector x"* para cada dron emulado, pero solamente un hilo *"Broker"*. Como no es necesario que cada dron realice esta comprobación, se escogió el hilo *"Broker"* para hacerla.

La verificación se realiza periódicamente, calculando la distancia entre las últimas posiciones conocidas de cada dron. En caso de detectar la colisión, se detiene inmediatamente el protocolo de evitación de colisiones.

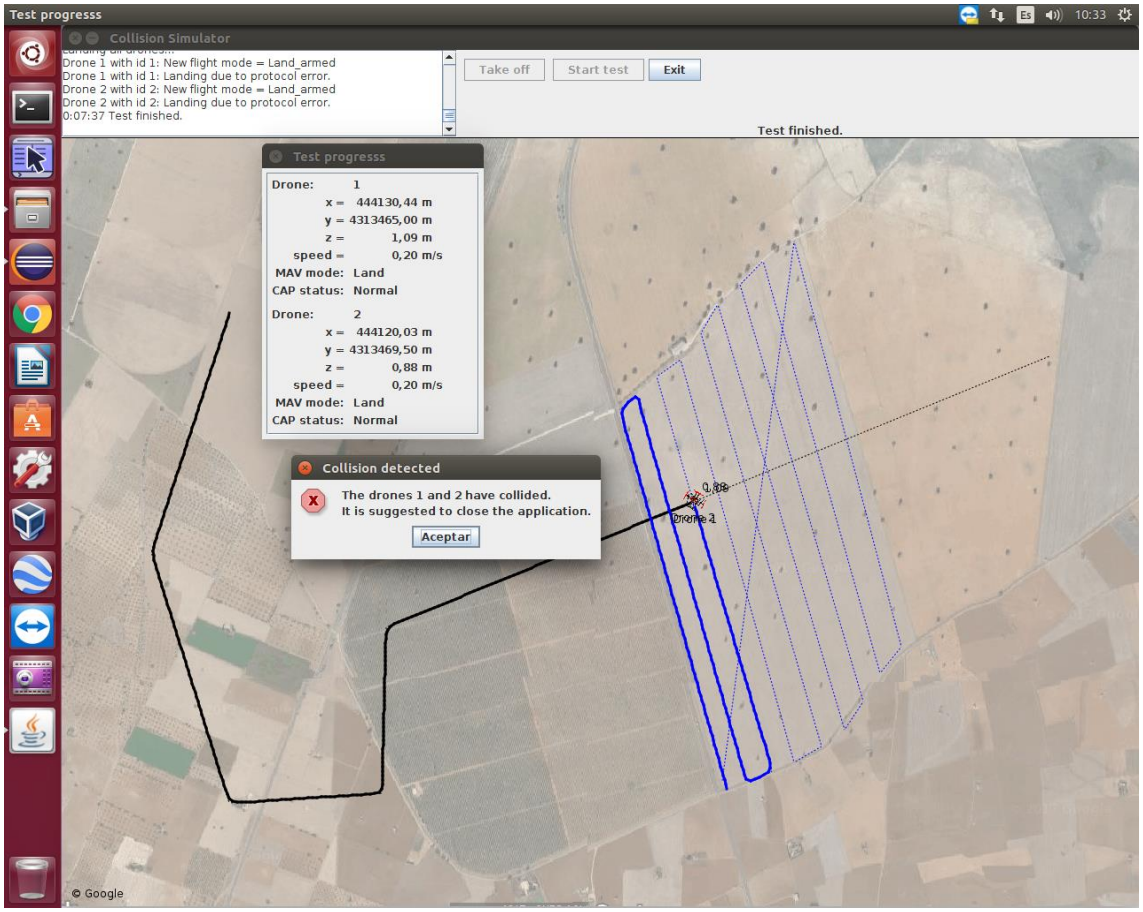

Figura 20: Colisión producida por un fallo en el protocolo de evitación de colisiones.

En la figura 20 se observa la información proporcionada al usuario cuando se produce una colisión. Inmediatamente, se abre un cuadro de diálogo en el que se indica qué drones han chocado y se da orden a todos los drones para que realicen un aterrizaje de emergencia (modo de vuelo *"land"* según el protocolo MAVLink, como se ve en el diálogo que muestra el progreso de la prueba). En el log de la parte superior izquierda de la pantalla se indica también que ha sido necesario aterrizar los drones por un error en el protocolo de evitación de colisiones.

8.3 Hilo CollisionDetector

En cada dron, el hilo *"CollisionDetector x"* es el encargado de aplicar el protocolo de evitación de colisiones y de dar las órdenes oportunas al dron.

El protocolo lo define la máquina de estados finitos de la figura 21.

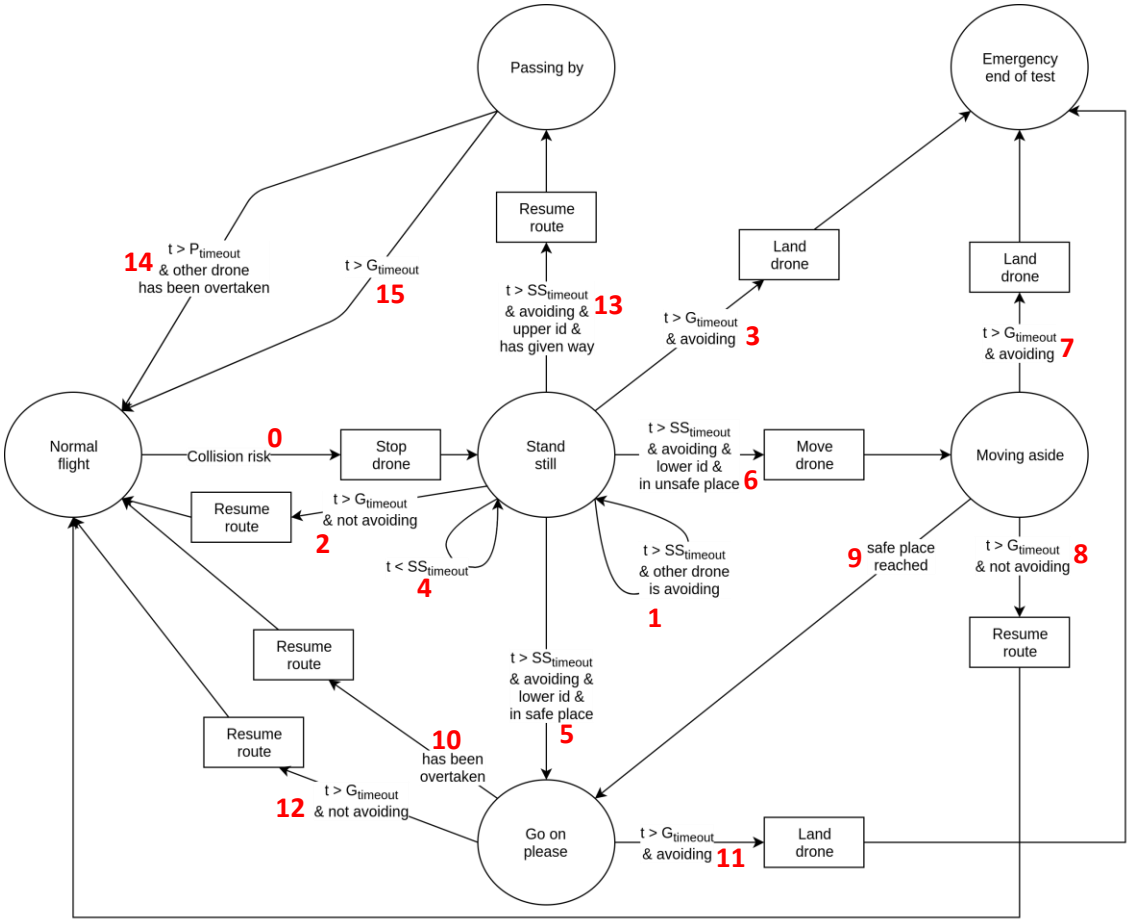

Figura 21: Máquina de estados finitos del protocolo de evitación de colisiones.

Un dron puede estar en uno de los seis estados representados mediante círculos:

- ***Normal flight***. Modo de vuelo mientras el protocolo no está siendo aplicado, es decir, cuando no se ha detectado ningún riesgo de colisión con otro dron.
- ***Stand still***. Estado en el que el dron permanece suspendido en el aire en las coordenadas actuales.
- ***Moving aside***. Puede darse el caso de que un dron deba hacerse a un lado para permitir el paso de otro dron. Éste es el modo de vuelo aplicado mientras completa el desplazamiento, hasta detenerse de nuevo.
- ***Go on please***. El dron con menor prioridad permanece en este estado mientras cede el paso al dron de mayor prioridad.
- ***Passing by***. El dron con mayor prioridad permanece en este estado mientras adelanta al dron de menor prioridad.
- ***Emergency end of test***. Este estado se ha definido por si se produce una situación no prevista en el protocolo. En tal caso, el dron aterriza para evitar que caiga si permanece en el aire hasta agotar la batería.

Los rectángulos representan las acciones emprendidas sobre el dron virtual, es decir, las órdenes que se le dan:

- ***Stop drone***. Detiene el dron en pleno vuelo. Como se ha comentado en el apartado 7.2, primero se estabiliza la altitud del dron, y luego se pasa a modo "Loiter armed" para que el dron mantenga la posición actual.
- ***Move drone***. Desplaza al dron a unas coordenadas para que no impida el paso al dron con mayor prioridad. Primero es necesario calcular dichas coordenadas, según se explica más adelante.
- ***Resume route***. Tras resolver el riesgo de colisión, se da orden al dron de continuar con la misión que estaba llevando.
- ***Land drone***. El dron aterriza en las coordenadas en las que se encuentra si se supera un tiempo máximo de espera mientras se está en alguna fase del protocolo. Se considera que se ha producido un interbloqueo.

Los números en color rojo representan las transiciones entre estados. A lo largo de la explicación, se indica entre paréntesis para facilitar la interpretación del diagrama.

El protocolo se ha definido para resolver eficientemente casi todas las situaciones de riesgo posibles entre dos drones, y para evitar la colisión entre tres o más drones en la mayoría de los casos. Se pone mayor énfasis en el primer caso, pues la probabilidad de que se encuentren más de dos drones en pleno vuelo es muy baja.

Primero se explica la situación en la solución planteada para el encuentro de tres o más drones, y después se explica con mayor detalle la solución para dos drones.

Como ya se ha comentado, los drones envían periódicamente tramas con información sobre su estado actual. Si un tercer dron se aproxima mientras dos drones están aplicando el protocolo, eventualmente recibe una trama en la que detecta que un dron, con el que puede chocar, está evitando colisionar con otro dron. En tal caso, se detendrá en el aire y esperará a que se resuelva la situación entre los otros dos drones (1 y 4).

Por otro lado, la información recibida de otros drones caduca y es eliminada si transcurre un tiempo t_1 (ej. 10 segundos).

Si se supera un tiempo $G_{timeout}$ (ej. 120 segundos), con $G_{timeout} > t_1$, y todavía se tiene información sobre un dron, se considera que el protocolo ha fallado, y se aterriza el dron (3). Por el contrario, si la información ha caducado y no se dispone de información sobre los drones que estaban evitando la colisión, se considera que esos drones han resuelto el riesgo de colisión y se han alejado lo suficiente como para que el tercer dron pueda reanudar el vuelo (2).

A partir de este punto se explica con mayor detalle la evitación de la colisión entre dos drones.

Cada dron tiene un identificador único que establece un orden determinista en la prioridad que tienen los drones para pasar, según el protocolo definido. El comportamiento que debe tener un dron depende por lo tanto de la prioridad asignada, lo que define **dos posibles casos**:

a) Dron con menor prioridad (id menor).

Estando en el modo *"Normal flight"*, si se detecta riesgo de colisión con otro dron, se detiene el dron y se entra en el modo *"Stand still"* (0). Hay riesgo de colisión con otro dron si van a coincidir en el espacio y en el tiempo, en un futuro cercano, como se explica más adelante en este mismo apartado, o también si otro dron le informa que ha detectado dicho riesgo de colisión.

Una vez se ha detenido el dron se espera un tiempo preprogramado para dar tiempo al otro dron a cambiar de estado (4), en caso de ser necesario.

El siguiente paso consiste en determinar si el dron está ocupando una región del espacio por la que tendrá que pasar el dron de mayor prioridad cuando le ceda el paso, según el procedimiento que se detalla en el anexo V.

Si no interrumpe el paso, se cambia el estado a *"Go on please"* (5), lo que permite al otro dron reanudar la marcha.

En caso contrario, se ordena al dron que se mueva a las coordenadas calculadas y se pasa al estado *"Moving aside"* (6). El dron necesitará un tiempo para alcanzar el objetivo. Si ese tiempo fuese excesivamente elevado y hubiese interbloqueo con otro dron, se considera que el protocolo ha fallado (7) y se aterriza el dron, pasando al estado *"Emergency end of test"*. Si el otro dron ya se ha ido, se reanuda la misión en el modo *"Normal flight"* (8). Lo normal es que el dron llegue al punto programado (9), con lo que se pasa al estado *"Go on please"*.

Si todo ha ido bien, llegados a este punto, el dron se halla en el estado *"Go on please"*, cediendo paso al otro dron. El dron de mayor prioridad, cuando le rebasa, actualiza la trama enviada indicando esta nueva circunstancia, lo que permite al dron reanudar la misión programada (10) y pasar al estado *"Normal flight"*. Como en casos anteriores, si el dron se mantiene en el estado *"Go on please"* demasiado tiempo, aterrizará de emergencia (11) o reanudará la misión (12) según haya bloqueo con otro dron o no, respectivamente.

Se decidió que el dron reanudase inmediatamente la marcha por tratarse de la solución más eficiente. Sin embargo, de haberse apartado para ceder el paso, se podría haber programado que el dron volviese a la posición de partida antes de reanudar la marcha. La primera opción es más eficiente y adecuada si se trata de un dron que, por ejemplo, reparte paquetes en espacio abierto. Por el contrario, la segunda opción sería más adecuada si el dron vuela en un espacio más confinado, realiza un levantamiento topográfico o cualquier otra misión que requiera referencias geográficas, aunque esta solución supondría más tiempo de vuelo y mayor complejidad en el protocolo. Se trata por lo tanto de una decisión de diseño.

b) Dron con mayor prioridad (id mayor).

Como en el caso anterior, el dron se detiene en caso de detectar riesgo de colisión, y pasa al estado *"Stand still"* (0). A continuación, permanece en este estado un tiempo mínimo para permitir al otro dron cambiar de estado, en caso de que sea necesario (4).

El otro dron es el que asume el control y determina si debe apartarse para ceder el paso. Una vez el dron menos prioritario está en sitio seguro y cede el paso, éste reanuda la misión programada y pasa al estado *"Passing by"* (13).

Una vez rebasa al otro dron (14), cambia al estado *"Normal flight"* y sale del protocolo. Como en casos anteriores, si lleva demasiado tiempo sin adelantar al dron menos prioritario (15), se considera que ha habido un error en el protocolo. En este caso, como el dron ya tenía la máxima prioridad, se limita a pasar al estado *"Normal flight"*, pues no hay necesidad de aterrizarlo. En su caso, aterrizará el dron de menor prioridad.

8.4 Formato de la trama periódica

El formato de la trama enviada periódicamente por cada dron se detalla en la figura 22, e incluye toda la información necesaria para cambiar entre los estados definidos en el protocolo.

2B	2B	4B	2B	2B	4B	8B	4B	12 x n (B)
id	evento	numSeq	modo	idEv	velocidad	Δt	n	x_1,y_1,z_1……x_n,y_n,z_n

Figura 22: Formato de una trama periódica.

Parámetros:

- ***id*** (short). Identificador del dron que envía la trama. Cuando se implemente el protocolo en drones reales, en lugar del último octeto de la IP será necesario sustituir este valor por la MAC del adaptador de red, que también es única.
- ***evento*** (short). Número que, comenzando en 0, indica el número de situaciones de riesgo de colisión que lleva resueltas hasta el momento.
- ***numSeq*** (int). Número de secuencia del paquete de datos enviado. Un entero es suficiente, pues si se enviasen 5 paquetes por segundo, no se desbordaría en más de 13 años, aun cuando en realidad el tiempo de vuelo nunca será superior a varias horas.
- ***modo*** (short). Modo de vuelo actual según el protocolo de evitación de colisiones.
- ***idEv*** (short). Cuando se está evitando colisionar con otro dron, representa el id de dicho dron. En otros caso vale -1.
- ***velocidad*** (float). Velocidad de vuelo en el plano horizontal, en m/s.
- ***Δt*** (long). Intervalo de tiempo transcurrido desde que la controladora de vuelo ha enviado la última posición del dron recibida, y el instante en el que se ha enviado la trama.
- ***n*** (int). Número de puntos incluidos en el resto de la trama, y que representan las futuras posiciones del dron según la misión planificada y la velocidad actual.
- x_i,y_i,z_i (3 x float). Coordenadas UTM de cada una de las posiciones futuras del dron.

La configuración por defecto envía en cada trama la posición futura del dron durante los próximos 25 segundos, y para ello tiene en cuenta la misión planificada, la velocidad de vuelo, y un intervalo entre dos puntos consecutivos de 500 ms, según se observa en la figura 23.

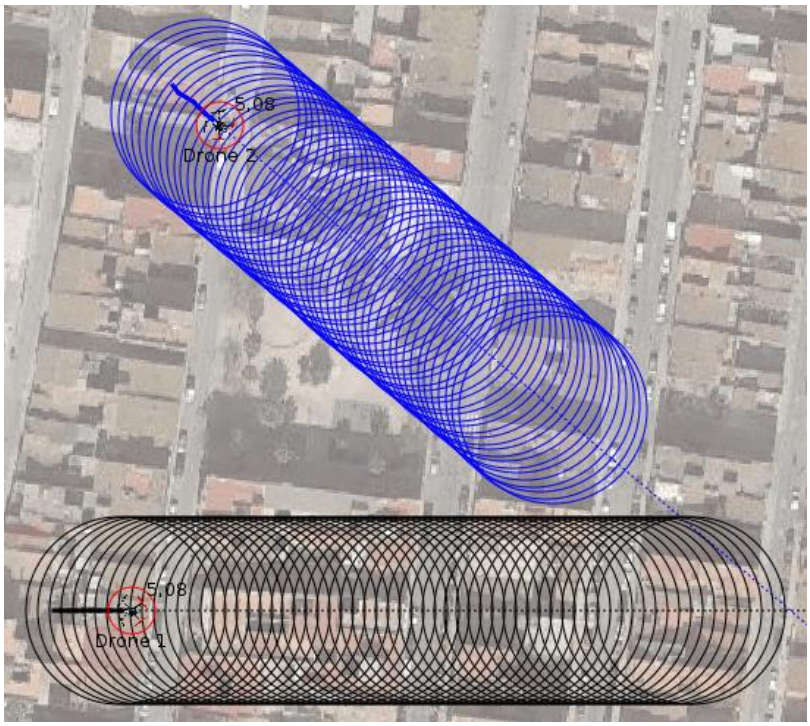

Figura 23: Proyección de futuras posiciones de dos drones.

En realidad, se envían las coordenadas de los puntos situados sobre la traza del dron, representada con una línea de puntos, aunque se representan círculos con centro en dichas coordenadas y de radio la máxima distancia a la que se considera que hay riesgo de colisión.

De este modo, si otro dron (no sus círculos) entra dentro del área definida por la superposición de las circunferencias, se interpreta fácilmente que hay riesgo de colisión. Aun así, en el apartado 8.5 se explican los cálculos con mayor detalle, pues además de coincidir en el espacio, dos drones tienen que coincidir en el tiempo para considerar que pueden colisionar.

Teniendo en cuenta la MTU de Ethernet (1500 bytes), las cabeceras IP (20 bytes) y UDP (8 bytes) y el tamaño de los ocho primeros campos de la trama (28 bytes), en una trama se puede enviar un máximo de 120 puntos:

$$1500 - 20 - 8 - 28 \geq 12 \cdot n$$

$$n \leq 120$$

Cuando se emplee el protocolo en drones reales, el identificador del dron será la MAC del interfaz de red, en vez de un número, por lo que los campos primero y quinto tendrán un tamaño de 6 en vez de 2 bytes. En tal caso, se podrá enviar un máximo de 119 puntos.

$$1500 - 20 - 8 - 36 \geq 12 \cdot n$$

$$n \leq 119$$

En ambos casos, el número de puntos es sobradamente elevado para el propósito del protocolo. La configuración por defecto envía 50 puntos, lo que supone un *payload* de 628 y 632 bytes, respectivamente, menos de la mitad del tamaño máximo de la trama.

Por otro lado, es muy probable que el protocolo se implemente en sistemas menos potentes, como una Raspberry Pi, o incluso un Arduino. Con objeto de minimizar el impacto que el cálculo de las posiciones futuras produce en el uso del procesador, dicho cálculo no se realiza cada vez que se envía una trama. Por el contrario, se realiza una vez cada varias tramas, y se actualiza en el paquete de datos el tiempo transcurrido desde que se generó la información, así como el modo de vuelo y el identificador del dron con el que se pretende evitar una colisión, por si estos valores se modifican durante la aplicación del protocolo de evitación de colisiones. Este es el motivo por el que se observa que el dron 1 (circunferencias negras) está por delante de la primera circunferencia grafiada.

La información que es necesario enviar en el paquete de datos varía según las distintas etapas del protocolo (figura 24):

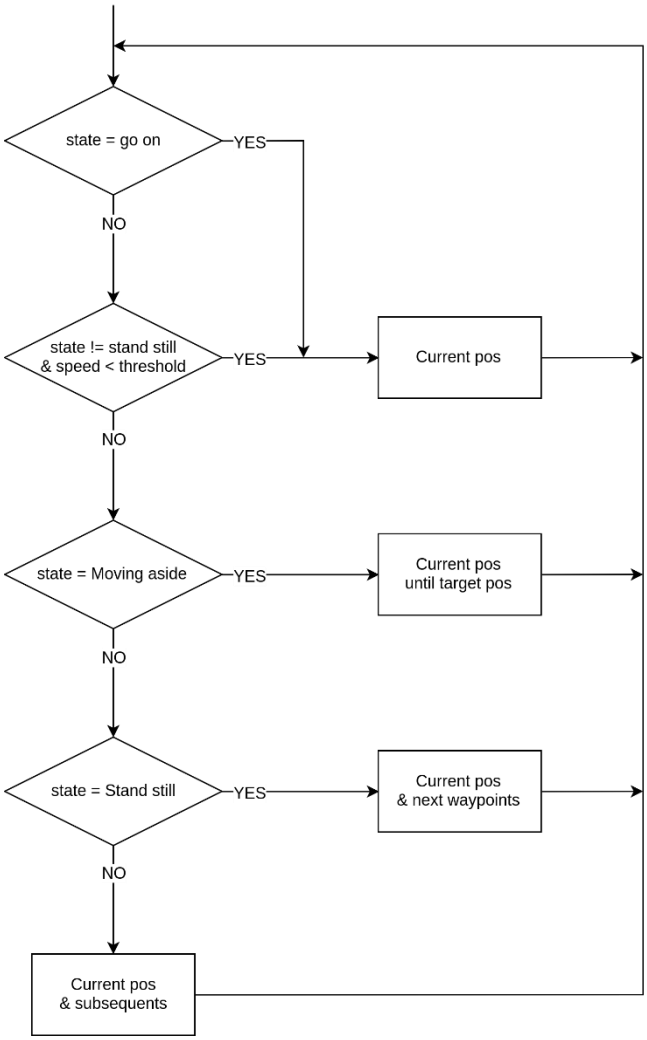

Figura 24: Selección de información a enviar en la trama según la fase del protocolo.

Mientras el dron menos prioritario da paso al otro dron, basta con enviar la posición actual, pues el dron está estacionario. Del mismo modo, si el dron no tiene activado el modo

estacionario pero su velocidad es muy reducida, entonces también basta con enviar la posición actual. Esta última situación se da cuando el dron inicia el movimiento o se está deteniendo.

En caso de que el dron menos prioritario se esté haciendo a un lado para ceder el paso, deberá enviar las posiciones futuras entre la actual y el punto hacia el que se está apartando.

Si el dron está detenido, deberá enviar la posición actual y los *waypoints* por los que todavía no ha pasado. Esta información permitirá al dron menos prioritario determinar si debe apartarse para dejar paso, en caso de que se halle próximo a las líneas que unen estos puntos, y que conforman la traza que seguirá el dron más prioritario.

En cualquier otro caso, se debe enviar las posiciones futuras del dron según la misión planificada. Según la velocidad de vuelo y el tiempo entre dos puntos consecutivos, se calcula la distancia que debe haber entre dichos puntos. A continuación, considerando dicha distancia, se calculan las posiciones futuras del dron sobre la secuencia de rectas formada por los *waypoints* por los que el dron todavía no ha pasado. El primer segmento de la secuencia lo formará la unión entre la posición actual y el primer *waypoint* por el que se tiene que pasar.

8.5 Cálculos del protocolo

Varios de los cálculos mencionados en el apartado 8.3 requieren una explicación más detallada:

1. **Cálculo del riesgo de colisión entre dos drones.**

Cuando un dron pretende determinar si va a colisionar con otro dron dispone de la información incluida en la última trama que ha enviado, y la de la última trama que ha recibido del otro dron.

Ambos paquetes de datos contienen una lista de posiciones futuras de cada dron (coordenadas espaciales), el tiempo transcurrido desde que se generó la trama y la velocidad de vuelo. Con esta información se puede calcular la posición y el instante en el que se encontrará cada uno de los dos drones en el futuro, en cada uno de esos puntos.

Primero se determina si cada posible combinación de un punto de la propia trama con un punto de la trama del otro dron coincide en el espacio y en el tiempo. En caso de ser así, se comprueba si el dron está lo suficientemente próximo al punto con riesgo de colisión como para adoptar medidas.

La necesidad de esta última comprobación viene del hecho de que puede haber riesgo de colisión, pero a una distancia elevada del dron. En tal caso, no resulta conveniente detener el dron y poner en marcha el protocolo, pues entonces el tiempo necesario para que un dron adelante al otro sería muy elevado y penalizaría excesivamente el tiempo empleado para resolver el riesgo de choque entre drones y, por lo tanto, el consumo de batería.

2. **Análisis de la necesidad de apartarse para ceder el paso.**

Cuando los drones están detenidos, en vez de enviar las posiciones futuras del dron en función de la velocidad de vuelo, envían la posición actual y los puntos correspondientes a los *waypoints* por los que todavía no han pasado (circunferencias en la figura 25).

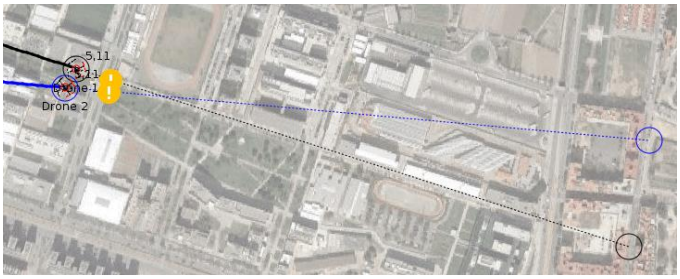

Figura 25: Drones detenidos tras detectar riesgo de colisión.

Para cada una de las rectas definidas por los puntos ya indicados, se repite el siguiente proceso (el proceso matemático se analiza con detalle en el anexo V):

- Primero se calcula la perpendicular a la misma que pasa por el dron, así como el punto de intersección recta-perpendicular.
- A continuación, si el punto de intersección está dentro del segmento formado por los puntos que determinan la recta, se comprueba si la distancia del dron a la recta es inferior a un umbral de seguridad, en cuyo caso es necesario apartar el dron para evitar que el dron con mayor prioridad choque con él cuando lo adelante.
- El proceso se repite iterativamente hasta que el dron se ha apartado lo suficiente de todas las líneas como para no suponer riesgo para el otro dron.

3. Comprobación de haber rebasado al dron menos prioritario.

Cada dron almacena en un *array* las últimas posiciones por las que ha pasado.

Mientras el dron más prioritario adelanta al menos prioritario, la distancia entre los dos disminuye. Cuando empieza a aumentar significa que ha rebasado al otro dron.

Partiendo de esta idea, determinar si se ha producido el adelantamiento se reduce simplemente a comprobar si la distancia de cada una de las posiciones anteriores del dron hasta el otro dron que está detenido va disminuyendo con el tiempo.

8.6 Configuración

Al arrancar el simulador aparece la ventana mostrada en la figura 3 del apartado 4.1. En caso de utilizar el protocolo de evitación de colisiones, cuando se termina de configurar el simulador, aparece el diálogo de la figura 26. Esta ventana permite personalizar los parámetros relativos al protocolo de evitación de colisiones.

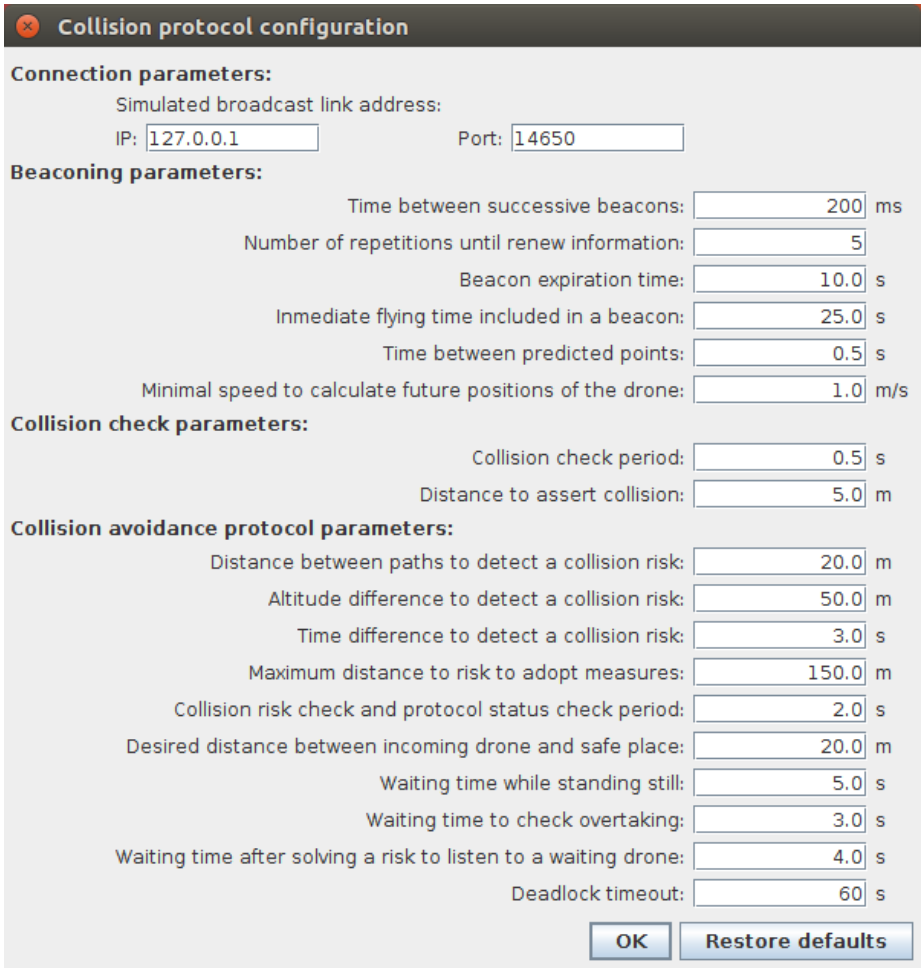

Figura 26: Configuración del protocolo de evitación de colisiones

En el mundo real, la comunicación entre los drones se deberá realizar mediante *broadcast* y dentro de una misma subred. En la aplicación, para simularlo, todos los drones envían la información a una misma IP que hace las veces de *broadcast*. El primer grupo de parámetros incluye dicha IP y el puerto de destino. Dado que la comunicación se realiza entre hilos de ejecución dentro de la misma aplicación, y corriendo en el mismo ordenador, lo más práctico es utilizar la interfaz de *loopback*.

El segundo grupo de parámetros hace referencia a las tramas transmitidas con información de vuelo.

Antes de recalcular de nuevo todo su contenido, se especifica el tiempo que transcurre entre dos tramas seguidas y el número de veces que se envía la misma trama actualizada, según se explica en el apartado 8.4. También se indican el tiempo de validez antes de que se descarte la última trama recibida, el tiempo de vuelo que se transmite en una trama, el tiempo transcurrido entre las posiciones futuras del dron que han sido calculadas, y la velocidad mínima que debe llevar para realizar dicho cálculo.

Mientras se diseña e implementa el protocolo de evitación de colisiones, es posible que tenga fallos y que se produzca la colisión entre dos drones. Por este motivo, es necesario comprobar periódicamente si dos drones han colisionado al ocupar simultáneamente la misma región del espacio. En el tercer grupo de parámetros se especifica la periodicidad de dicha

comprobación, y la distancia a la que deben estar dos drones para considerar que han colisionado.

El último grupo de parámetros está relacionado con el proceso de detección del riesgo de colisión a partir de las tramas recibidas de otros drones.

Los dos primeros parámetros marcan la distancia y la diferencia de cota entre dos drones a partir de las que se considera que hay riesgo de colisión.

Cada trama está formada por una secuencia discontinua de puntos en el espacio. Por este motivo, prácticamente nunca coincidirán las posiciones futuras de dos drones en el espacio y al mismo tiempo. Por lo tanto, además de definir un margen para la posición en el espacio, con el tercer parámetro se contempla también un margen de seguridad en el tiempo.

Por otro lado, cabe la posibilidad de que haya riesgo de colisión, pero en un punto del espacio todavía muy alejado de ambos drones. En tal caso no conviene detener los drones, pues el dron con prioridad tardaría mucho tiempo en rebasar al otro dron y sería poco eficiente. El cuarto parámetro determina la máxima distancia entre drones que se considera para tener en cuenta el cálculo realizado con los dos parámetros anteriores.

El quinto parámetro determina la frecuencia con la que se comprueba si hay riesgo de colisión con otros drones. En un ordenador personal se podría poner un periodo muy inferior, pero se han sugerido 2 segundos para no introducir mucha sobrecarga de cálculo en el que será el sistema real cuando se aplique el protocolo, una Raspberry Pi o incluso un Arduino. El mismo valor determina la frecuencia con que se determina si hay que cambiar el estado en el que se encuentra el dron dentro del protocolo de evitación de colisiones.

El sexto parámetro especifica la distancia a la que se tiene que apartar un dron para dejar paso al otro en caso de estar en la traza por la que debe pasar.

También se incluyen tiempos de espera del protocolo para permitir a los drones alcanzar un estado estable durante el vuelo antes de tomar decisiones. En el apartado 8.3 se explica en detalle la máquina de estados finitos que representa el proceso de detección y evitación de colisiones.

Como ya se ha comentado, durante el desarrollo del protocolo de evitación de colisiones pueden producirse situaciones no previstas. Entre ellas, los drones podrían quedarse en uno de los estados del protocolo indefinidamente (interbloqueo). El último parámetro indica el tiempo máximo que un dron puede estar en un estado. Si se supera ese tiempo el dron aterriza, para que la prueba pueda concluir.

Finalmente, una vez se acepta la configuración del protocolo, se abre la ventana principal del programa y se arrancan las máquinas virtuales, según se observa en la figura 4 del apartado 4.2.

En caso de haber activado el uso del protocolo de evitación de colisiones, sobre cada dron aparecen dos círculos concéntricos.

El círculo de menor radio representa el área de seguridad dentro de la cual no puede entrar ningún dron sin que se detecte que se ha producido una colisión.

El círculo de mayor diámetro es el área cubierta por la detección de colisiones. Una vez el dron entra en movimiento, en la trama se incluyen las futuras posiciones del mismo. Alrededor

de estas posiciones se grafían también círculos, definiendo un área de seguridad a lo largo de la ruta seguida, como se observa en la figura 6 del apartado 4.4.

8.7 Ejemplo

A continuación, se analiza una prueba realizada, desde el punto de vista del protocolo de evitación de colisiones.

En la figura 27 se muestra el estado en el que terminan los drones al finalizar. El dron 2 es el más prioritario, mientras que el dron 1 es el menos prioritario y, además, ha tenido que apartarse para ceder el paso al dron 2.

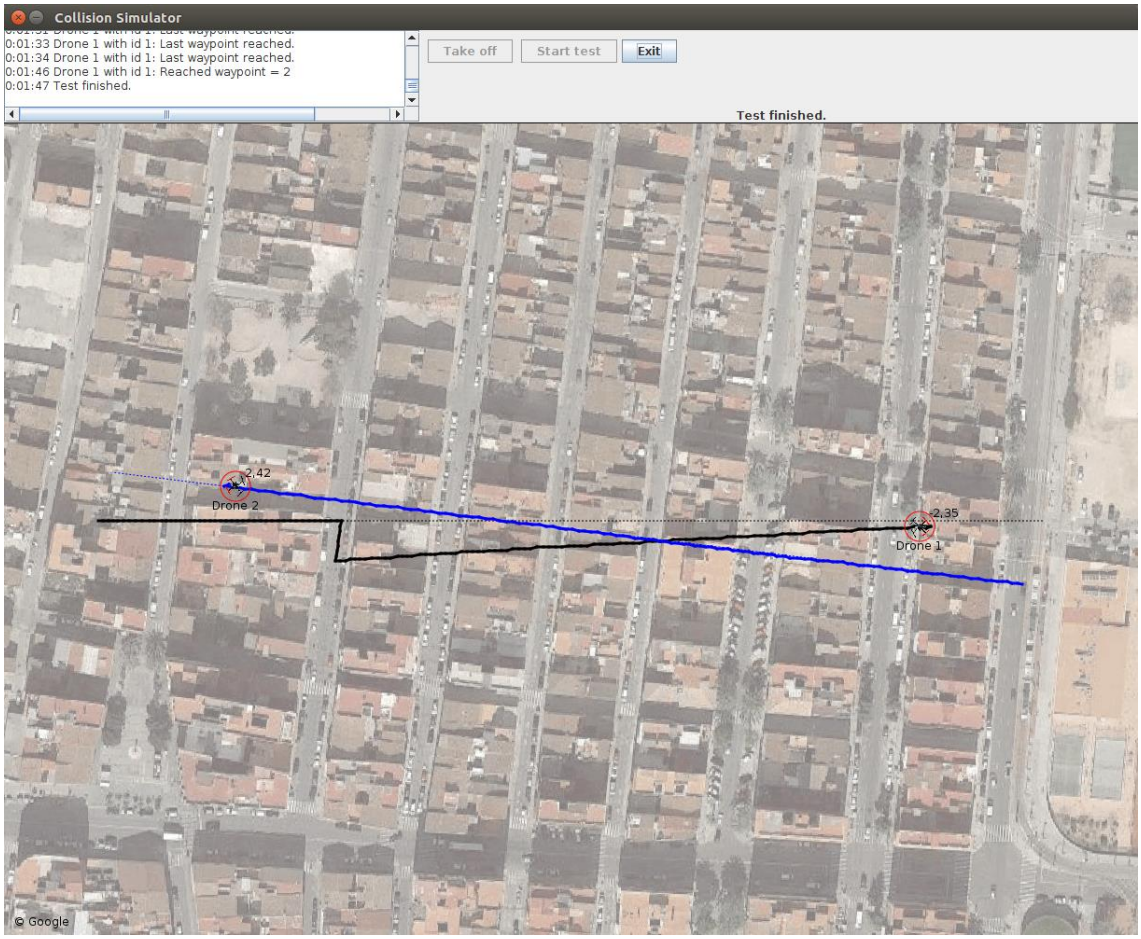

Figura 27: Ejemplo de prueba del protocolo de evitación de colisiones.

En la figura 28 se muestra la distancia a origen del dron 1 a medida que transcurre la prueba.

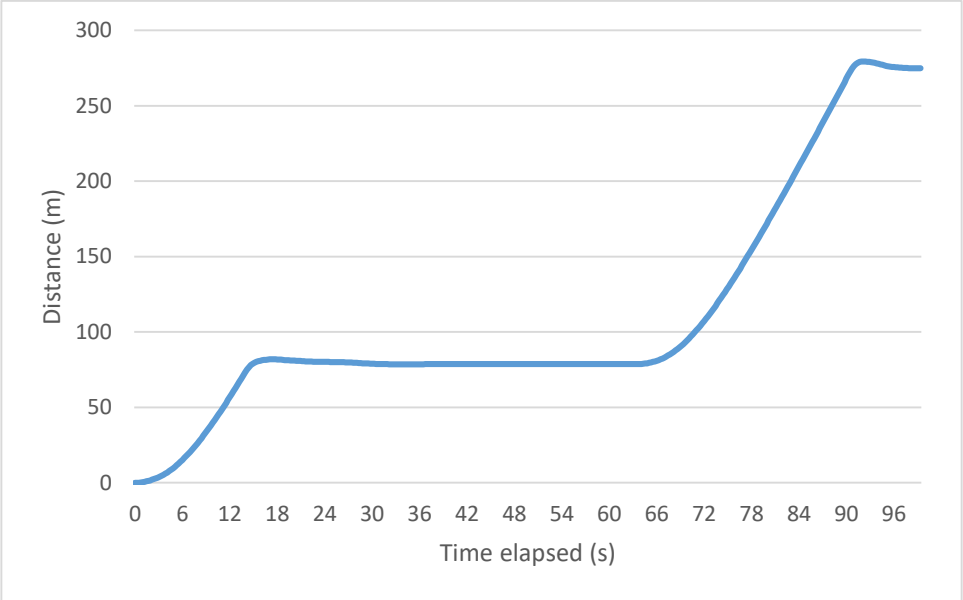

Figura 28: Distancia a origen por el dron 1 en función del tiempo.

La curva inicial indica la aceleración cuando comienza la prueba. Poco después alcanza la velocidad programada y, alrededor del segundo 14, se detecta el riesgo de colisión y se detiene.

Hasta el segundo 28 retrocede ligeramente mientras se aparta para ceder el paso al dron más prioritario. Después, permanece estacionario mientras se produce el adelantamiento.

Aproximadamente en el segundo 66 acelera hasta la velocidad máxima con vistas a alcanzar el último *waypoint* de la misión. En ese momento se produce una desaceleración brusca y se estabiliza mientras aterriza.

En la figura 29 se muestra cómo evoluciona la velocidad de ambos drones a medida que transcurre la prueba.

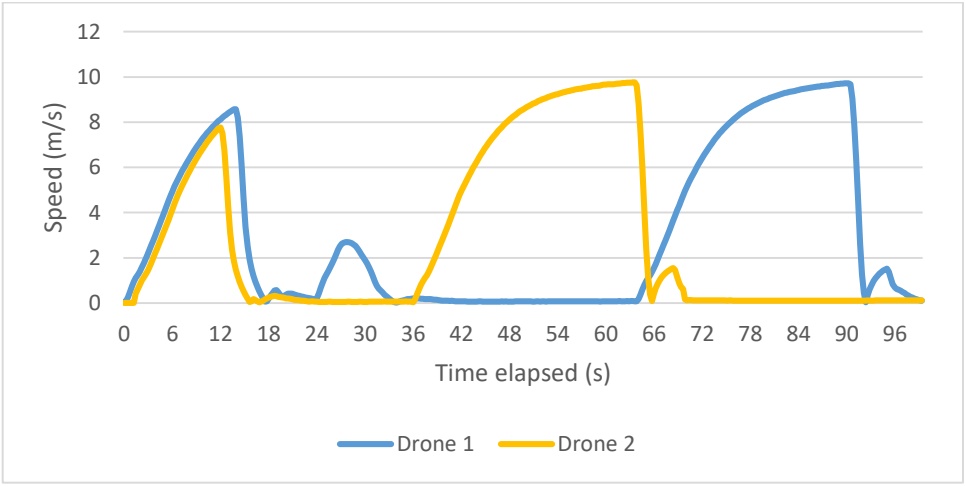

Figura 29: Velocidad de ambos drones durante el transcurso de la prueba.

En esta figura se interpreta mejor las acciones emprendidas por los drones. El riesgo de colisión se ha detectado antes de que alcanzasen la velocidad programada (10 m/s).

Tras la aceleración inicial se observa el frenado en seco de ambos drones. Pequeñas variaciones de velocidad reflejan los ajustes que hace cada dron para estabilizar la posición.

El dron 1 acelera hasta alcanzar un punto seguro, en el segundo 28, y luego frena para detenerse en esa posición.

Seguidamente, el dron 2 obtiene permiso para pasar y acelera. Cuando está a punto de alcanzar la velocidad programada, alcanza el destino final y se detiene. Tras el segundo 66 realiza unos pequeños ajustes para mantener la posición en planta mientras desciende. Aterriza en el segundo 72.

Poco antes de llegar a su destino, el dron 2 informa al dron 1 que le ha adelantado, y este último acelera para continuar con su misión. Al igual que el dron 2, cuando llega a su destino frena en seco y aterriza, mientras estabiliza su posición en planta con ligeras aceleraciones.

9 Validación de la aplicación

La validación de la aplicación incluye dos aspectos. Primero, hay que tener en cuenta que cumple correctamente con la funcionalidad para la que ha sido diseñada. Segundo, hay que tener en cuenta el consumo de recursos, ya que cada dron corre en una máquina virtual.

En lo que respecta a la funcionalidad, el simulador ha sido suficientemente robusto. Una vez se han eliminado bugs detectados durante el desarrollo, ha demostrado ser estable durante las pruebas realizadas.

Solamente se han detectado problemas de estabilidad cuando se ha utilizado un número de máquinas virtuales superior a 7. Como se detalla a continuación, la inestabilidad no se debe específicamente a la aplicación, sino al uso de máquinas virtuales, que colapsan el sistema.

En cuanto al consumo de recursos, hay que tener en cuenta el tipo de ordenador en el que se han realizado las pruebas. Se trata de una máquina de última generación y de altas prestaciones, como se deduce de las características indicadas en el apartado 3, lo que indica que los resultados serán inferiores en equipos más modestos.

Durante las pruebas de eficiencia se han utilizado los parámetros por defecto del simulador, definiendo una velocidad de vuelo de los drones de 2,5 m/s, y obteniendo el uso de memoria y CPU. En lo que respecta a E/S, se considera que el disco duro no influye en la eficiencia de la aplicación, pues se trata de un disco SSD SATA de tasa de transferencia superior a los 500 MB/s en lectura y escritura.

Tabla 1: Uso medio de memoria en MB.

	Sistema	Con 1 dron	Con 9 drones
Usuario	1.049	1.790	6.515
Sistema y caché	3.160	3.160	3.170
Total	4.209	4.950	9.685

De la tabla 1 se deduce que el arranque de un dron virtual adicional supone un incremento en consumo de memoria de aproximadamente 607 MB.

Hay que tener en cuenta que el sistema utilizado dispone de una cantidad de memoria muy elevada (32GB), lo que incrementa la cantidad de memoria utilizada como caché del sistema respecto a un sistema de menores prestaciones. Es decir, que un sistema menos potente, la cantidad de memoria utilizada por el sistema es menor, tanto en reposo como tras arrancar máquinas virtuales.

De todos modos, los resultados obtenidos sugieren que en una máquina de 8 GB de RAM convendría lanzar un máximo de cuatro o cinco drones.

En lo que respecta al uso del procesador, hay que tener en cuenta dos situaciones distintas. Por un lado, durante el arranque del simulador es necesario poner en ejecución las máquinas virtuales, lo que supone un consumo muy elevado y, por otro lado, hay que estudiar el consumo de CPU mientras se realiza una prueba con el simulador, es decir, el consumo durante el funcionamiento normal de la aplicación.

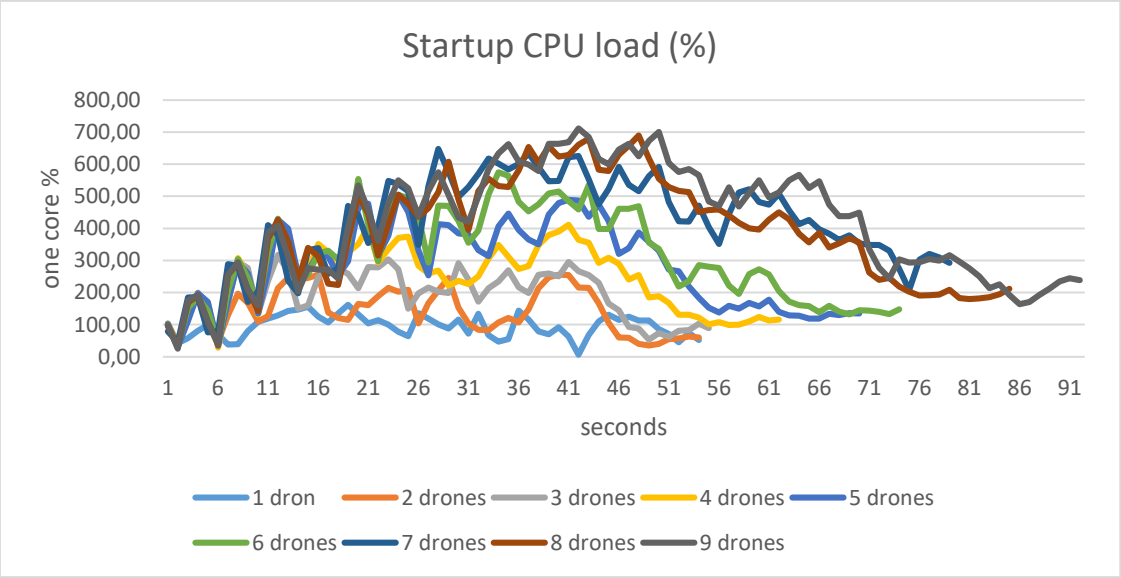

Figura 30: Consumo de CPU durante el arranque de las máquinas virtuales.

En la figura 30 se muestra la progresión del consumo medio de CPU en el tiempo, durante el arranque de las máquinas virtuales, y para un número distinto de drones utilizados.

El sistema dispone de 4 *cores* con *hyperthreading*, lo que lo dota de capacidad para ejecutar aproximadamente 8 tareas simultáneas. Teniendo en cuenta que la carga óptima de un servidor es del 70%, la carga óptima de funcionamiento para 8 tareas es del 560%.

La primera y obvia conclusión es que el tiempo de arranque aumenta con el número de máquinas virtuales que se están poniendo en marcha.

En la mitad izquierda de la gráfica también se observa con facilidad los picos que representan el instante en que se arranca una nueva máquina virtual, a intervalos equidistantes.

Teniendo en cuenta la carga óptima del sistema, se puede concluir que no es conveniente arrancar más de 6 máquinas virtuales, para no sobrepasar el 600% de uso de CPU. Una solución al consumo tan alto de CPU durante el arranque sería definir un intervalo de tiempo más elevado entre el arranque de dos máquinas virtuales. Sin embargo, este parámetro es totalmente dependiente del hardware, lo que impide definirlo con la suficiente precisión sin realizar pruebas en un número muy elevado de ordenadores personales, lo que se escapa del alcance del presente proyecto.

En la figura 31 se muestra el consumo medio de CPU durante el arranque de las máquinas virtuales.

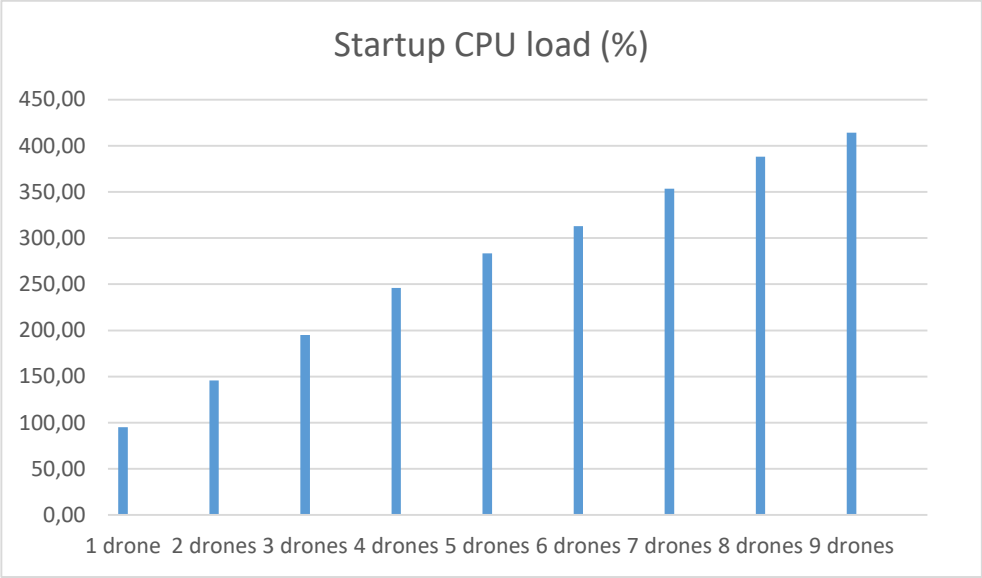

Figura 31: Consumo medio de CPU durante el arranque de las máquinas virtuales.

Se observa que la progresión es prácticamente lineal con el número de máquinas virtuales utilizadas.

Por otro lado, en la figura 32 se representa el consumo medio de CPU mientras se realiza una prueba, con distinto número de drones.

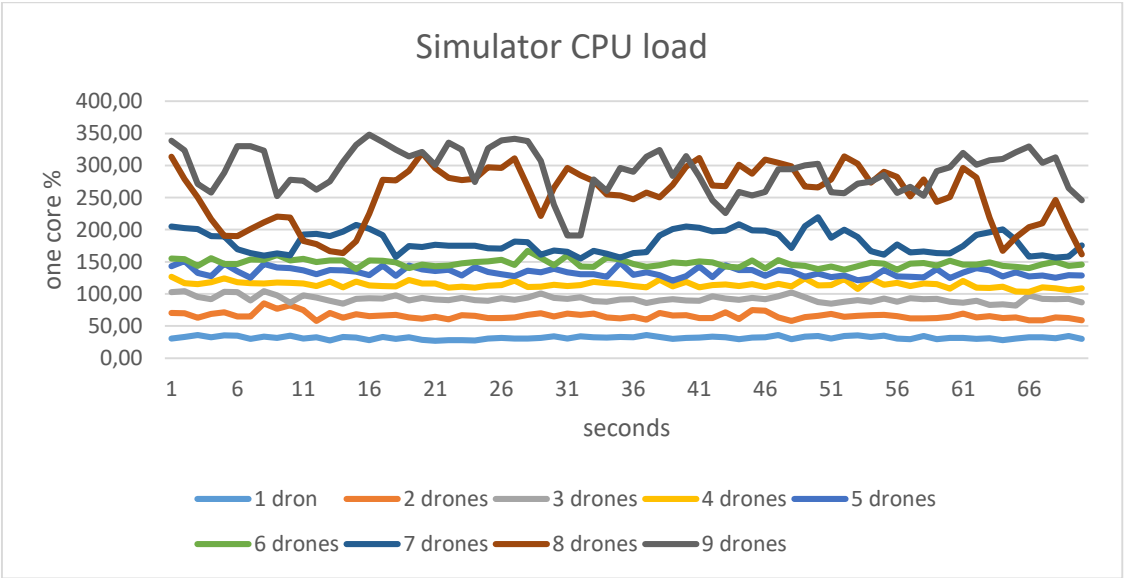

Figura 32: Uso de CPU durante la realización de una prueba.

Como se observa, el uso de CPU es muy uniforme mientras se utilizan hasta 6 drones. A partir de dicho punto se vuelve inestable, lo que demuestra que el sistema se aproxima a su capacidad máxima, y desaconseja utilizar más drones virtuales.

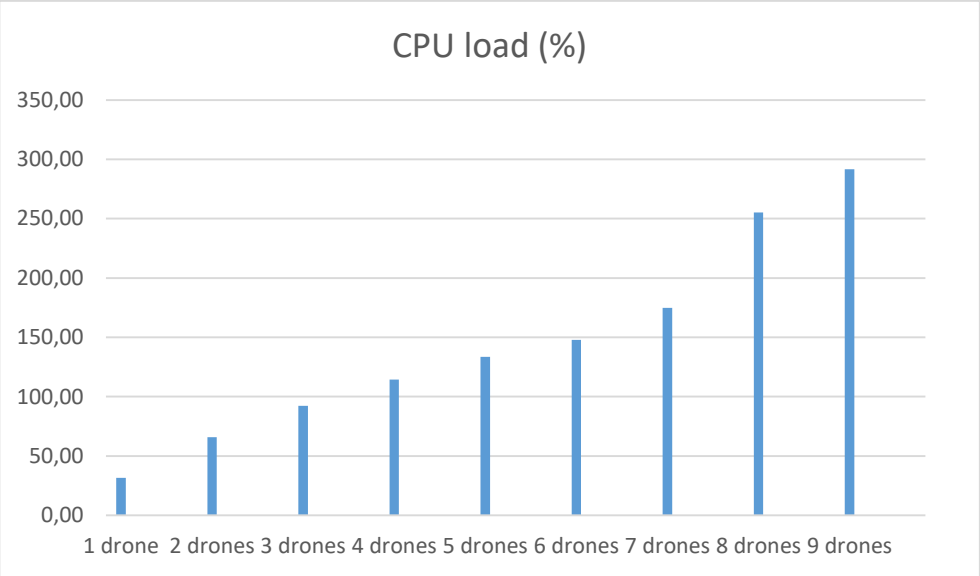

Figura 33: Uso medio de CPU durante la realización de una prueba.

La figura 33 representa el consumo medio durante la realización de una prueba en función del número de drones empleado.

En este caso, se comprueba que el consumo de CPU se dispara cuando se utilizan más de 7 drones, lo que encaja con las conclusiones anteriores. No es prudente llevar el sistema utilizado más allá de la simulación de 6 drones, si se pretende evitar que las limitaciones del sistema afecten a la calidad de los resultados de los experimentos.

Por último, cabe concluir que, mientras se utilice un sistema lo suficientemente potente, la aplicación es capaz de simular la interacción entre hasta tres drones con muy alta fiabilidad, lo cual es nuestro objetivo en este momento para la implementación de un protocolo de evitación de colisiones.

También cabe recordar que la solución implementada se puede modificar para llevar a otros ordenadores la emulación de los drones, lo que mejoraría drásticamente la escalabilidad de la aplicación en caso de ser necesario para alguna otra aplicación. Esto es así puesto que queda demostrado que el consumo de recursos se debe esencialmente a la emulación de los drones dentro de una máquina virtual, no a la lógica del simulador.

10 Validación del protocolo de evitación de colisiones

La validación del protocolo implementado debe contemplar no solo la eficiencia evitando colisiones, como también el aumento de duración del vuelo, pues la batería de los drones es un recurso crítico y se agota rápidamente.

En lo que respecta al éxito evitando colisiones, el protocolo cumple correctamente su función, pues en ninguna de las 26 pruebas realizadas se ha producido ninguna colisión. Para evaluar el tiempo de vuelo perdido por cada dron se ha definido una batería de pruebas.

Las pruebas consideradas incluyen las situaciones más comunes que pueden darse cuando se encuentran dos drones, así como un caso real:

- Cruce en perpendicular.
- Aproximación estando enfrentados.
- Adelantamiento.
- Aproximación esviada.
- Aproximación esviada, en sentido opuesto.
- Caso real. Cruce con un dron que supervisa un campo de cultivo.

En cada situación se han obtenido el tiempo de vuelo, el tiempo dedicado al protocolo de evitación de colisiones, y el tiempo que se habría empleado en caso de no haber evitado la colisión, lo que permite evaluar la sobrecarga en tiempo de vuelo que introduce el protocolo.

A continuación, se muestran los resultados obtenidos en cada una de las pruebas. En el anexo VI se detallan los parámetros de cada prueba, con objeto de que pueda ser reproducida.

En todos los casos, el dron 2 tiene prioridad de paso sobre el dron 1.

1) Cruce en perpendicular.

En la figura 34 y siguientes se muestra la situación en la que quedan los drones cuando finaliza cada una de las pruebas realizadas. En este caso se decidió probar la aproximación de los drones cuando llevan rumbos ortogonales.

El dron 1 no necesita apartarse, pues se detiene a una distancia de la traza del dron 2 lo suficientemente elevada.

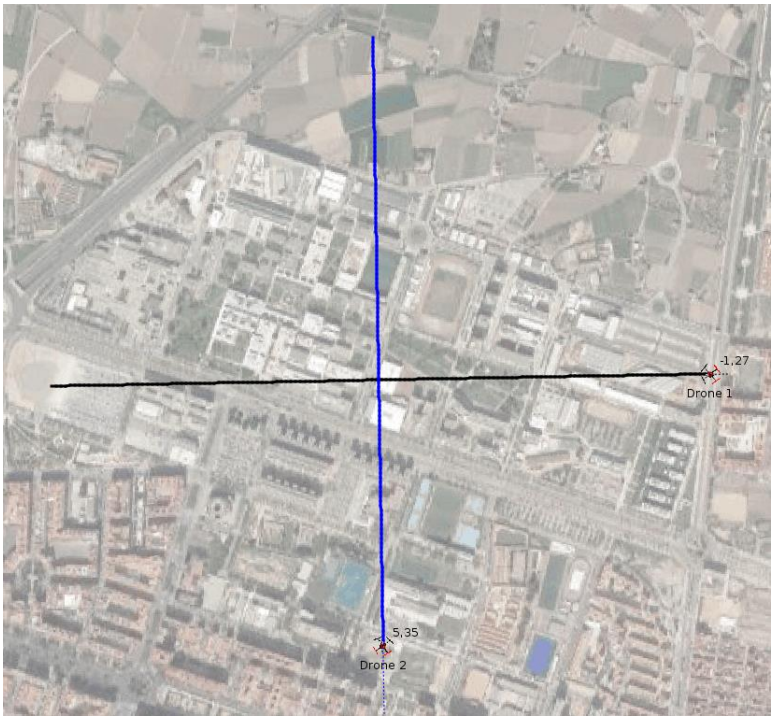

Figura 34: Cruce en perpendicular.

De la última columna de la tabla 2 se deduce que el dron 1 emplea 22 segundos más que el dron 2 para evitar la colisión. Este incremento de tiempo o uno similar se produce en todas las pruebas, pues el dron 1 debe esperar en vuelo mientras el dron 2 le adelanta.

El tiempo de vuelo en modo normal también varía. En el caso del dron 1 aumenta, pues tiene que decelerar para detenerse y acelerar para reanudar la marcha, y eso incrementa el tiempo necesario para recorrer la misma distancia. Por el contrario, en el dron 2 disminuye, pues parte de la distancia que se recorrería en vuelo normal, se recorre ahora en el modo *"Passing by"*, mientras se adelanta al dron 1.

Tabla 2: Resultado del cruce en perpendicular.

	Evitando la colisión			Sin evitar	
	Normal	Evitando	Total	Total	**Δt**
Dron 1	2:59	0:29	3:28	2:48	**0:40**
Dron 2	2:26	0:24	2:50	2:32	**0:18**

2) Aproximación estando enfrentados.

Como ambos drones están siguiendo la misma línea, aunque en sentidos opuestos, es necesario que el dron 1 se haga a un lado para que el dron 2 pueda llegar a su destino de manera segura. Tras ser rebasado por el dron 2, el dron 1 continúa la misión programada hasta alcanzar el último *waypoint*.

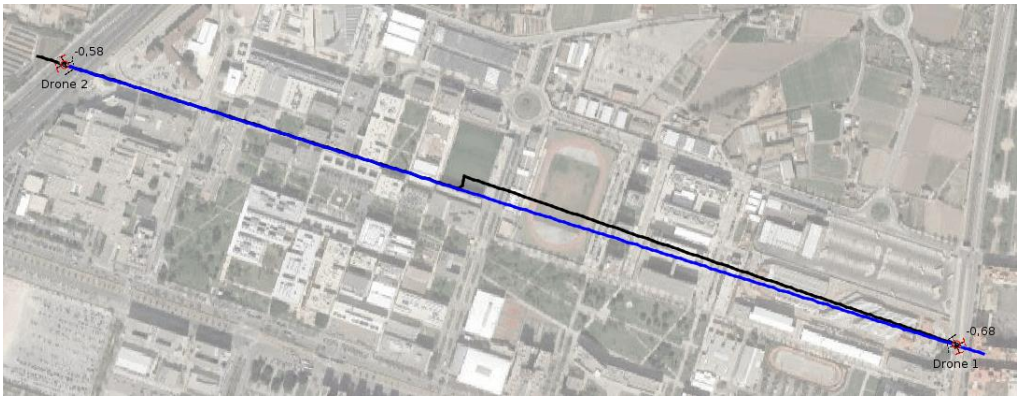

Figura 35: Aproximación estando enfrentados.

En este caso, dado que el dron 1 tiene que apartarse para ceder el paso al dron 2, el tiempo necesario para evitar la colisión aumenta considerablemente.

Tabla 3: Resultado de la aproximación estando enfrentados.

	Evitando la colisión			Sin evitar	
	Normal	Evitando	Total	Total	**Δt**
Dron 1	2:58	0:54	3:52	2:47	**1:05**
Dron 2	2:33	0:51	3:25	2:48	**0:37**

3) Adelantamiento.

En la tercera prueba, la misión de ambos drones sigue la misma dirección y con el mismo sentido. Para permitir que el dron 2 pudiese alcanzar al dron 1, se redujo la velocidad del segundo a la mitad.

Como se observa en la figura 36, el dron 2 alcanza al dron 1 a un cuarto del final de la misión. En ese momento, el dron 1 se detiene y se aparta para no ocupar la línea por la que el dron 2 debe pasar.

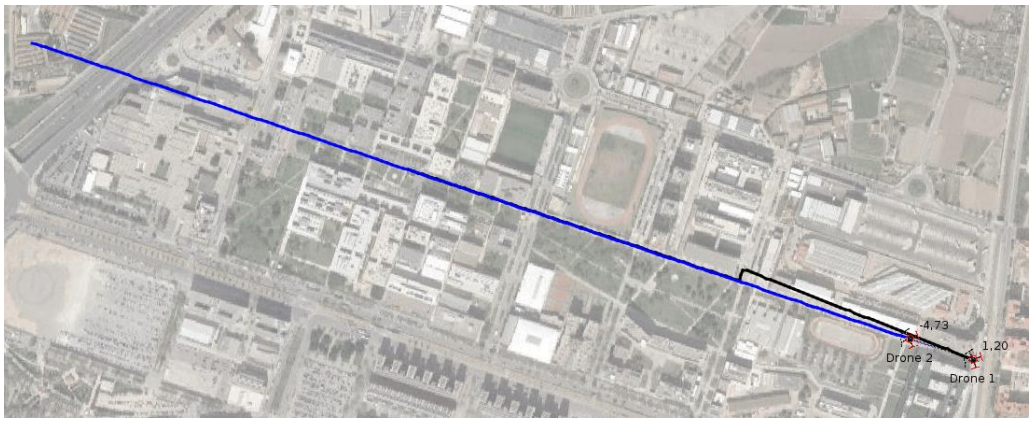

Figura 36: Adelantamiento.

El aumento de tiempo respecto a la primera prueba es muy inferior al del caso anterior, pues los drones están más cerca el uno del otro cuando se inicia el proceso de adelantamiento, y por lo tanto se hace más rápido.

Tabla 4: Resultado del adelantamiento.

	Evitando la colisión			Sin evitar	
	Normal	Evitando	Total	Total	**Δt**
Dron 1	3:28	0:40	4:08	3:20	**0:48**
Dron 2	2:47	0:38	3:25	2:55	**0:30**

4) Aproximación esviada.

En esta prueba, los drones se aproximan entre sí lentamente. El riesgo de colisión se detecta a mitad del proceso.

Dado que el dron 1 está suficientemente separado de la traza del dron 2, no es necesario que se aparte para cederle el paso.

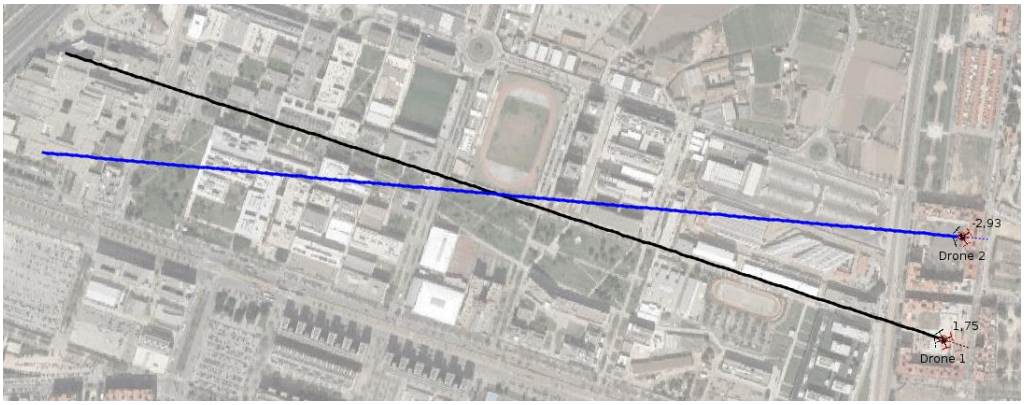

Figura 37: Aproximación esviada.

Se trata de la situación ideal, pues no es necesario que el dron 1 se aparte para ceder paso y, además, el dron 2 está prácticamente a la misma altura que el dron 1 cuando se detienen, por lo que el tiempo necesario para que lo adelante es mínimo.

Como consecuencia, se considera que los resultados obtenidos son óptimos y que, por lo tanto, el tiempo mínimo de vuelo perdido por un dron con este protocolo es de 18 segundos, para una velocidad de vuelo de 10 m/s. El dron 1 pierde 12 segundos más por la necesidad de esperar a ser adelantado por el otro dron.

Tabla 5: Resultado de la aproximación esviada.

	Evitando la colisión			Sin evitar	
	Normal	Evitando	Total	Total	**Δt**
Dron 1	2:53	0:19	3:12	2:42	**0:30**
Dron 2	2:56	0:15	3:11	2:53	**0:18**

5) Aproximación esviada, en sentido opuesto.

La prueba utiliza la misión anterior en cada uno de los drones, pero invirtiendo el sentido de la marcha en el dron 2.

Tras detectar el riesgo de colisión, el dron 1 tiene que apartarse.

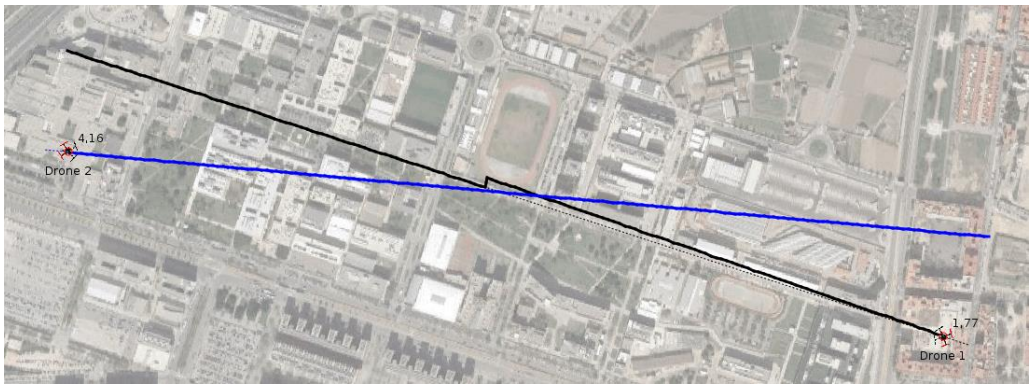

Figura 38: Aproximación esviada, en sentido opuesto.

Los resultados se asemejan más a la segunda prueba, como era de esperar, aunque no son tan negativos.

Tabla 6: Resultado de la aproximación esviada, en sentido opuesto.

	Evitando la colisión			Sin evitar	
	Normal	Evitando	Total	Total	**Δt**
Dron 1	2:53	0:44	3:37	2:42	**0:55**
Dron 2	2:27	0:44	3:11	2:42	**0:29**

6) Caso real. Cruce con un dron que supervisa un campo de cultivo.

Para terminar, se consideró conveniente probar con un caso más real.

El dron 2 está realizando una inspección programada de un campo de gran superficie (1500 x 900 m). Para ello, lo recorre en bandas paralelas separadas aproximadamente 100 m, y comenzando en la esquina inferior izquierda, según se observa en la figura 39. Cuando termina el recorrido en la esquina superior derecha, regresa al punto de origen para ser recogido por el propietario.

El otro dron está realizando una misión programada genérica que intercepta el área recorrida por el dron 2. Se encuentran, aproximadamente, de forma perpendicular, cuando el dron 2 está realizando la tercera pasada sobre el campo (punto especificado en la figura 20 del apartado 8.2).

Dado que el encuentro es prácticamente ortogonal, sucede lo mismo que en la primera prueba, es decir, no es necesario que el dron 1 se aparte para que pase el otro dron.

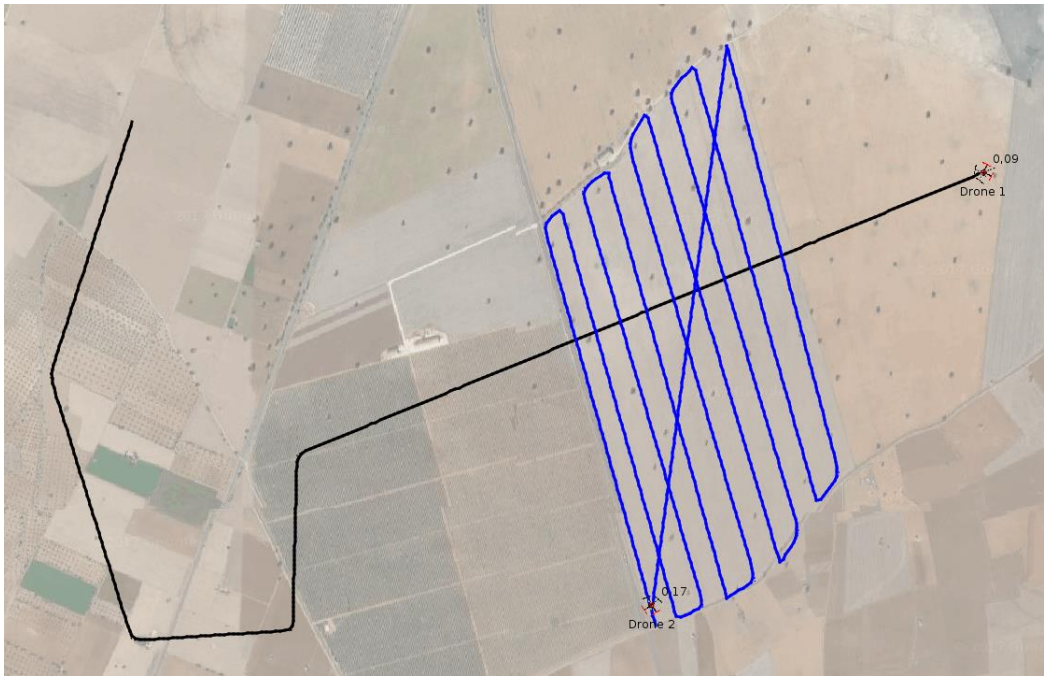

Figura 39: Caso real. Cruce con un dron que supervisa un campo de cultivo.

La misión seguida por el dron 2 implica recorrer una distancia superior a 15 km, motivo por el cual el tiempo de vuelo es muy superior a las pruebas anteriores.

Sin embargo, se constata que la pérdida de tiempo introducida al aplicar el protocolo de evitación de colisiones es independiente del tipo y duración de la misión programada, obteniendo valores similares a los de la primera prueba.

Tabla 7: Resultado del caso real. Dron que supervisa un campo de cultivo.

	Evitando la colisión			Sin evitar	
	Normal	Evitando	Total	Total	**Δt**
Dron 1	10:04	0:24	10:28	9:54	**0:34**
Dron 2	30:10	0:20	30:30	30:14	**0:16**

En general, el tiempo de vuelo se incrementa entre 16 y 37 segundos en el caso del dron con mayor prioridad, y entre 30 y 65 segundos en el caso del dron menos prioritario.

El caso más favorable se da cuando los drones se aproximan lentamente y están a la misma distancia respecto al punto de intersección de sus trazas. El dron menos prioritario no tiene que apartarse y, además, el adelantamiento requiere muy poco tiempo.

El caso más desfavorable se da cuando los drones siguen la misma ruta, pero en sentido contrario. Por un lado, como los drones se detienen estando más alejados, el proceso de adelantamiento lleva más tiempo. Además, el dron menos prioritario tiene que hacerse a un lado para que pase el dron más prioritario.

11 Conclusiones

El uso cada vez más extendido de multicópteros, usualmente denominados simplemente como drones, genera una serie de nuevos desafíos para desarrolladores, entre los que destaca la necesidad de comunicación entre dichos drones con propósitos tales como evitar choques o permitir realizar vuelos sincronizados con enjambres de drones, entre otros.

En este proyecto se propone e implementa un protocolo de evitación de colisiones entre drones. Además, se ha desarrollado un simulador de vuelo capaz de controlar varios drones, con el que se ha podido evaluar la corrección y las prestaciones del protocolo con elevada precisión, pues se basa en la simulación completa de los drones, incluyendo sus propiedades físicas. Para ello, también se ha definido un modelo realista del enlace de comunicación WiFi entre los drones virtuales en la banda de los 5 GHz, que considera la merma de la señal con la distancia entre los mismos, y que se puede utilizar a voluntad del usuario en el simulador desarrollado. La banda de 2,4 GHz se desechó por las interferencias producidas con los mandos de control remoto, según se deduce de trabajos anteriores [9] y se detalla en el apartado 6.

Globalmente, se han cumplidos los objetivos del proyecto, pues se han desarrollado completamente tanto el simulador como el protocolo de evitación de colisiones, y se ha verificado su correcto funcionamiento.

La batería de pruebas realizadas ha permitido determinar que el protocolo es garantía suficiente para impedir la colisión entre drones que vuelan según una misión programada y en un entorno sin obstáculos. Por otro lado, se ha constatado que la situación ideal se da cuando dos drones se aproximan casi en paralelo, y que el caso más desfavorable se da cuando se dirigen el uno hacia el otro siguiendo la misma recta.

12 Trabajos futuros

La evitación de la colisión entre dos drones ha quedado completamente resulta en el protocolo propuesto, mientras que la solución para la evitación de la colisión con un tercer dron dista de ser la óptima. Queda para futuros trabajos modificar el protocolo de modo que tres o más drones puedan evitarse entre sí de forma eficiente, evitando tanto mermar la eficiencia cuando se evitan solamente dos drones, como posibles interbloqueos entre los drones.

Ha quedado patente que el simulador de vuelo está limitado por el uso de recursos del sistema. Resultará conveniente segregar del simulador toda la parte relativa a la ejecución de un dron virtual en una aplicación cliente. De este modo, se podrá ejecutar toda la parte centralizada del simulador en un ordenador, y varias instancias del cliente en distintos ordenadores dentro de la misma red local, lo que incrementará drásticamente la escalabilidad del simulador.

En base a los resultados de este proyecto se abre un amplio abanico de posibilidades que permitirán profundizar en áreas como:

- Comunicaciones: estudiar las prestaciones con antenas distintas (planas, polarización circular) y de distinta ganancia, así como considerar otros factores, como la posición relativa entre las antenas, para modelar el enlace de comunicación. Permitirá incluir en el simulador más modelos de comunicación para su elección por el usuario.
- Control: desarrollar soluciones que permiten explotar el potencial de las comunicaciones entre multicópteros para crear *UAV Swarms* (enjambres de multicópteros). Requerirá modificar el simulador para el control automatizado de los drones.
- Redes: estudiar la viabilidad de distintas técnicas y algoritmos de encaminamiento para redes ad-hoc basadas en multicópteros (*Flying Ad Hoc Networks*).
- Aplicaciones: proponer soluciones basadas en uno o varios multicópteros de cara a cubrir una determinada área de estudio de la manera más eficiente posible (por ejemplo, midiendo la calidad de aire en un sector mediante multicópteros).

Bibliografía

[1] MANATHARA J. G., GHOSE D., *"Reactive collision avoidance of multiple realistic UAVs"*, en *Aircraft Engineering and Aerospace Technology: An International Journal 83/6*, 2011, PP 388-396

[2] CHMAJ G., SELVARAJ H., *"Distributed Processing Applications for UAV/drones: A Survey."*, en *Selvaraj H., Zydek D., Chmaj G. (eds) Progress in Systems Engineering. Advances in Intelligent Systems and Computing*, vol 366. Springer, Cham, 2015

[3] MAHJRI I., DHRAIEF A., BELGHITH A., *"A Review on Collision Avoidance Systems for Unmanned Aerial Vehicles"*, en *"Communication technologies for vehicles: 8th international workshop, nets4cars/nets4trains/nets4aircraft 2015"*, sousse, tunisia, mayo 2015, PP 203-214

[4] ANSHUL SINGH. *Drone Collision Avoidance* <https://create.arduino.cc/projecthub/anshulsingh163/drone-collision-avoidance-system-0b6002> [Consulta: 5 de junio 2017]

[5] ALEXANDER KHVILIVITZKY. *Visual collision avoidance system for unmaned aerial vehicles* <https://www.google.com/patents/US5581250> [Consulta: 5 de junio 2017]

[6] DRONETHUSIAST. *Drone Flight Simulator – Analysis & Comparison* <http://www.dronethusiast.com/drone-flight-simulator/> [Consulta: 5 de junio 2017]

[7] ARDUPILOT. *SITL Simulator (Software in the Loop)* <http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html> [Consulta: 5 de junio 2017]

[8] FABRA F., CALAFATE C. T., CANO J. C., MANZONI P., *"A methodology for measuring uav-to-uav communications performance"*, en *"2017 IEEE Consumer Communications and Networking Conference"*, enero 2017

[9] FABRA F., CALAFATE C. T., CANO J. C., MANZONI P., *"On the impact of inter-UAV communications interference in the 2.4 GHz band"*, en *"2017 Wireless Communications and Mobile Computing Conference"*, 28 de junio 2017

[10] RC MODEL REVIEWS. *Review: FrSky/FriSky 2.4GHz FHSS (Getting Technical)* <http://www.rcmodelreviews.com/frskyreview2.shtml> [Consulta: 5 de junio 2017]

[11] DRONEAIR. *Nueva ley sobre el uso de drones en España* <http://www.dronair.es/nueva-ley-sobre-el-uso-de-drones-en-espana-2> [Consulta: 5 de junio 2017]

[12] AGENCIA ESTATAL DE SEGURIDAD AÉREA. *Drones – Trabajos aéreos* <http://www.seguridadaerea.gob.es/lang_castellano/cias_empresas/trabajos/rpas/default.aspx> [Consulta: 5 de junio 2017]

[13] DRONEAIR. *Nueva legislación de drones en España para 2017* <http://www.dronair.es/actualizacion-de-la-ley-sobre-el-uso-de-drones-en-espana> [Consulta: 5 de junio 2017]

[14] QGROUNDCONTROL. *Waypoint Protocol* <http://qgroundcontrol.org/mavlink/waypoint_protocol> [Consulta: 5 de junio 2017]

[15] QGROUNDCONTROL. *MAVLink Release 1.1 Compatibility and Features* <http://qgroundcontrol.org/mavlink/release_11> [Consulta: 5 de junio 2017].

[16] *Programando y desarrollando – Protocolo MAVLink* <https://noescomolocuentan.wordpress.com/2014/10/18/programando-y-desarrollando-protocolo-mavlink/> [Consulta: 5 de junio 2017]

[17] QGROUNDCONTROL. *Field Reordering and CRC Extra Calculation* <http://qgroundcontrol.org/mavlink/crc_extra_calculation> [Consulta: 5 de junio 2017]

[18] QGROUNDCONTROL. *MAVLink micro air vehicle marshalling/communication library* <https://github.com/mavlink/mavlink> [Consulta: 5 de junio 2017]

[19] GHELLE. *MAVLinkJava: A Java code generator and a Java library for MAVLink* <https://github.com/ghelle/MAVLinkJava> [Consulta: 5 de junio 2017]

[20] QUATERNIUM (2015). *GRCQuad. Caracterización de la aeronave*. Documento proporcionado con el dron en su adquisición.

[21] ARDUPILOT. *Pixhawk and Telem2 port on Raspberry Pi* <http://discuss.ardupilot.org/t/pixhawk-and-telem2-port-on-raspberry-pi/3151> [Consulta: 5 de junio 2017]

[22] RASPBERRY PI FOUNDATION. *GPIO: Models A+, B+, Raspberry Pi 2 B and Raspberry Pi 3 B* <https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/README.md> [Consulta: 5 de junio 2017]

[23] JOHN WATKINS. *Configuring The GPIO Serial Port On Raspbian Jessie Including Pi 3* <http://spellfoundry.com/2016/05/29/configuring-gpio-serial-port-raspbian-jessie-including-pi-3/> [Consulta: 5 de junio 2017]

[24] MILLIWAYS. *How do I make serial work on the Raspberry Pi3* <https://raspberrypi.stackexchange.com/questions/45570/how-do-i-make-serial-work-on-the-raspberry-pi3> [Consulta: 5 de junio 2017]

[25] ARDUPILOT DEV TEAM. *Communicating with Raspberry Pi via MAVLink* <http://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html> [Consulta: 5 de junio 2017]

[26] RXTX. *Rxtx* <http://rxtx.qbang.org/wiki/index.php/Main_Page> [Consulta: 5 de junio 2017]

[27] SZATI. *How to get Serviio running on Raspbian on a Raspberry Pi* <http://wiki.serviio.org/doku.php?id=howto:linux:install:raspbian> [Consulta: 5 de junio 2017]

[28] RXTX. *Download - RXTX* <http://rxtx.qbang.org/wiki/index.php/Download> [Consulta: 5 de junio 2017]

[29] ZUI. *Java and RXTX* <https://www.raspberrypi.org/forums/viewtopic.php?t=12452> [Consulta: 5 de junio 2017]

[30] RASPBERRY PI FOUNDATION. *Download Raspbian For Raspberry Pi* <https://www.raspberrypi.org/downloads/raspbian/> [Consulta: 5 de junio 2017]

[31] ALEJANDRO ESQUIVA RODRÍGUEZ. *Tutorial Raspberry Pi - Wireless Ad-hoc Network* <https://geekytheory.com/tutorial-raspberry-pi-wireless-ad-hoc-network/> [Consulta: 5 de junio 2017]

[32] RPDOM. *Connecting to a network* <https://www.raspberrypi.org/forums/viewtopic.php?t=128458&p=859002> [Consulta: 5 de junio 2017]

[33] VIRTUALBOX. *Oracle VM Virtualbox* <https://www.virtualbox.org/> [Consulta: 5 de junio 2017]

[34] HASHICORP. *Vagrant by HashiCorp* <https://www.vagrantup.com/> [Consulta: 5 de junio 2017]

[35] *Git for Windows* <https://git-for-windows.github.io/> [Consulta: 5 de junio 2017]

[36] ARDUPILOT. *Setting up SITL using Vagrant* <http://ardupilot.org/dev/docs/setting-up-sitl-using-vagrant.html> [Consulta: 5 de junio 2017]

[37] MITCHELL HASHIMOTO. *Host-Only Networking* <https://friendsofvagrant.github.io/v1/docs/host_only_networking.html< [Consulta: 5 de junio 2017]

[38] GOOGLE. *Guía del desarrollador de Google Static Maps* <https://developers.google.com/maps/documentation/static-maps/intro> [Consulta: 5 de junio 2017]

[39] USER1691694, STACKOVERFLOW. *Getting lon/lat from pixel coords in Google Static Map* <https://stackoverflow.com/questions/10442066/getting-lon-lat-from-pixel-coords-in-google-static-map> [Consulta: 5 de junio 2017]

[40] MATT. *How To Autostart Apps In Raspbian LXDE Desktop* <http://www.raspberrypi-spy.co.uk/2014/05/how-to-autostart-apps-in-rasbian-lxde-desktop/> [Consulta: 5 de junio 2017]

[41] PX4 AUTOPILOT. *Pixhawk Autopilot. Pixhawk Flight Controller Hardware Project* <https://pixhawk.org/modules/pixhawk> [Consulta: 5 de junio 2017]

[42] USER2548538. *Java, convert lat/lon to UTM* <http://stackoverflow.com/questions/176137/java-convert-lat-lon-to-utm> [Consulta: 5 de junio 2017]

[43] QGROUNDCONTROL. *MAVLINK Common Message Set* <http://mavlink.org/messages/common> [Consulta: 5 de junio 2017]

[44] ARDUPILOT. *MAVProxy* <http://ardupilot.github.io/MAVProxy/html/index.html> [Consulta: 5 de junio 2017]

[45] ARDUPILOT. *Complete Parameter List* <http://ardupilot.org/copter/docs/parameters.html> [Consulta: 5 de junio 2017]

[46] ARDUPILOT. *Copter Mission Command List* <http://ardupilot.org/copter/docs/mission-command-list.html> [Consulta: 5 de junio 2017]

[47] SCOTT, SUPERUSER. *Can I pipe/redirect a console application through netcat so it can be used remotely?* <https://superuser.com/questions/607783/can-i-pipe-redirect-a-console-application-through-netcat-so-it-can-be-used-remot/607855> [Consulta: 5 de junio 2017]

[48] CRISTIAN CIUPITU, SERVERFAULT. *Can I send some text to the STDIN of an active process running in a screen sesión?* <https://serverfault.com/questions/178457/can-i-send-some-text-to-the-stdin-of-an-active-process-running-in-a-screen-sessi> [Consulta: 5 de junio 2017]

[49] DRONE RANGER, DIYDRONES. *Set Flight Modes With Mavlink?* <http://diydrones.com/forum/topics/set-flight-modes-with-mavlink> [Consulta: 5 de junio 2017]

[50] ARDUPILOT. *MAVLink Mission Command Messages (MAV_CMD)* <http://ardupilot.org/copter/docs/common-mavlink-mission-command-messages-mav_cmd.html#mav-cmd-nav-takeoff> [Consulta: 5 de junio 2017]

[51] TIM BENNETT, STACKEXCHANGE. *How do I stop two wireless dongles switching between wlan0 and wlan1?* <https://raspberrypi.stackexchange.com/questions/24318/how-do-i-stop-two-wireless-dongles-switching-between-wlan0-and-wlan1> [Consulta: 5 de junio 2017]

[52] ALVIN ALEXANDER. *Java exec: How to execute a system command pipeline in Java* <http://alvinalexander.com/java/java-exec-system-command-pipeline-pipe> [Consulta: 5 de junio 2017]

[53] PAVAN KUMAR. *How do I run SSH commands on remote system using Java?* <https://stackoverflow.com/questions/2514439/how-do-i-run-ssh-commands-on-remote-system-using-java> [Consulta: 5 de junio 2017]

[54] SENTHIL, STACKOVERFLOW. *java runtime.getruntime() getting output from executing a command line program* <https://stackoverflow.com/questions/5711084/java-runtime-getruntime-getting-output-from-executing-a-command-line-program> [Consulta: 5 de junio 2017]

[55] EMYL. *Running remote commands after vagrant ssh* <https://stackoverflow.com/questions/22523134/running-remote-commands-after-vagrant-ssh> [Consulta: 5 de junio 2017]

[56] ARDUPILOT. *MAVProxy, código interno* <https://github.com/ArduPilot/MAVProxy/blob/master/MAVProxy/modules/mavproxy_mode.py> [Consulta: 5 de junio 2017]

[57] GALOISINC. *ardupilot-mega, código interno* <https://github.com/GaloisInc/ardupilot-mega/blob/master/APMrover2/GCS_Mavlink.pde> [Consulta: 5 de junio 2017]

Glosario

WiFi. Tecnología de conexión inalámbrica entre dispositivos según el estándar IEEE 802.11, y que típicamente opera en las bandas de frecuencia de 2,4 y/o 5 GHz.

Red Ad-hoc. Red de comunicaciones descentralizada donde las estaciones comunican directamente entre sí.

Punto de acceso. Dispositivo que hace de puente de comunicación entre dispositivos inalámbricos, y entre éstos y alguna otra tecnología, como puede ser Ethernet sobre cable.

Dron. Vehículo aéreo no tripulado controlado por radiofrecuencia, normalmente capaz de realizar vuelos programados.

Multirrotor. Variante de dron que se caracteriza por disponer de 3 o más rotores, lo cual le permite tener una elevada estabilidad en el aire.

Pixhawk. Dispositivo hardware diseñado para el control de vuelo y navegación.

ArduPilot. Plataforma software instalada en un dispositivo tipo Pixhawk para el control de drones.

Raspberry Pi 3. Microordenador de bajo coste basado en la plataforma ARM diseñado con el objeto de fomentar el aprendizaje de las ciencias de computación en las escuelas. Dispone de puertos USB, Ethernet y suficiente capacidad de cómputo para realizar tareas medianamente complejas.

Raspbian. Sistema operativo basado en Debian con el que funcionan los dispositivos Raspberry Pi.

MAVLink. Formato de mensajes y protocolo de comunicación diseñado para la transferencia de paquetes de control y de información desde y hacia drones.

Misión. Secuencia de puntos geográficos por los que debe pasar un multicóptero mientras realiza un vuelo programado (no tripulado).

Waypoint. Punto en el espacio por el que debe pasar un multicóptero durante una misión. Entre otros parámetros, también se puede forzar a que el vehículo cambie la velocidad de vuelo o adopte un determinado comportamiento al llegar a una determinada distancia de dicho punto.

AESA. Agencia Estatal de Seguridad Aérea, encargada de regular el vuelo de drones en el territorio español.

Anexo I: El multirrotor GRCQuad

Caracterización

Se trata de una aeronave multirrotor en configuración X4 Quadcopter [20], con peso inferior a 2 kg, de la marca Quaternium modelo GRCQuad v1. Este modelo usa el *frame* DJI F330, y está propulsado por 4 motores eléctricos de 150W de potencia, con control electrónico de velocidad de 15 A, y dispone de batería LiPo de 14,8 V y 3.300MAh de alta descarga para la alimentación de los propulsores. Dispone de control de eje de cabeceo/pitch, desplazamiento lateral o alabeo (roll) y guiñada (yaw), así como controladora de estabilización giroscópica y GPS. Se controla mediante transmisor de radio control de 7 canales en la banda de 2,4 GHz.

La aeronave ofrece 5 modos de vuelo: Modo estabilizado GPS, Modo de sensor de Altitud, Modo estabilizado GPS+altura (*Loiter*), Modo de vuelta a casa, y Modo misión (automático).

Como medidas de seguridad, dispone de un sistema de protección por pérdida de enlace radio y por bajo nivel de batería.

Las baterías empleadas no se dañan con facilidad cuando se descargan rápidamente durante su uso. Sin embargo, tienen como inconveniente una limitada autonomía, lo que condiciona la planificación de la toma de datos.

Tabla 8: Prestaciones del multirrotor GRCQuad.

AERONAVE	Fabricante	Modelo	Identificador	Categoría
	Quaternium	**GRCQuad**	**CSM1150001 CSM1150002**	**RPA < 2 kg**
MOTOR	Tipo	Modelo/Kv	Potencia	Alimentación
	Eléctrico	**SunnySky X2212-980v**	**250 W**	**DC 12 V**
VARIADOR ELECTRÓNICO	Tipo	Modelo	Amp. Cont/máx	Entrada DC
	ESC	**DJI**	**15 A**	**5,6 a 12 V**
HÉLICE	Fabricante	Diámetro	Paso	Material
	GenFan	**8”**	**4,5”**	**Nylon carbon reinforced**
PESOS	En vacío	En vacío + batería	Carga máxima	MTOW
	831 g	**1196 g**	**604 g**	**1800 g**
DIMENSIONES	Anchura total	Anchura diagonal	Altura	Distancia ejes motor
	560 mm	**360 mm**	**160 mm**	**250 mm**
PRESTACIONES	Velocidad máx. ascenso	Velocidad máx. descenso	Velocidad máx. giro horizontal	Velocidad máx. de vuelo
	6 m/s	**3 m/s**	**200 º/s**	**10 m/s**
	Ángulo máx. inclinación		Autonomía (bat. 3,3 Ah)	
	45ª		**7 minutos**	

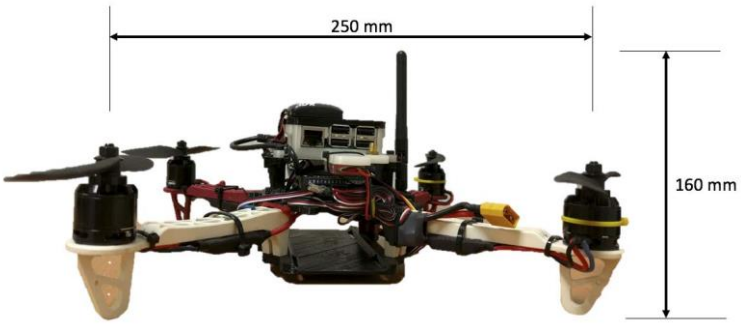

Figura 40: Vista lateral del multirrotor GRCQuad.

Figura 41: Vista superior del multirrotor GRCQuad.

Figura 42: Vista desde el motor delantero derecho del multirrotor GRCQuad.

Listado de componentes y equipos de la aeronave

Tabla 9: Componentes del multirrotor GRCQuad.

#	Descripción	Marca	Modelo	Datos
1	Propulsores eléctricos x4	SunnySky	X2212-980Kv	150 w, salida 15 A, empuje 0,96 Kg
2	Batería alimentación propulsor	Gens ace	4S1P – 3,3Ah	LiPo 3300 mAh 14,8V 25C 4S1P
3	Regulador electrónico de velocidad x4		15 A	Bec 5 V – 3 A, 15-20 A, 5,6-12 V
4	Controladora de vuelo	3DR	Pixhawk [41]	3D ACC / Gyro / MAG / Baro
5	GPS y brújula	3DR	u-blox GPS with compass	Modulo u-blox NEO-7, freq. actualización 5 Hz, HMC5883L
6	Telemetría	3DR	Radio Telemetry Kit 433 MHz	Puerto Micro-USB 33 MHz, conector DF12 de 6 posiciones, 100 mW máx., -117 dBm, conector RP-SMA, basado en el protocol MAVLink
7	Enlace radio	FrSky	X8R	8/16 Ch S.BUS ACCST receptor de telemetría con Smart Port
8	Sistema embebido	Raspberry Pi	RPi 3 model B	1,2GHz quad-core ARM Cortex-A53 CPU, 1GB RAM, 4 USB, 40 GPIO pins, HDMI, Ethernet, CSI, DSI, Micro SD
9	Webcam	Raspberry Pi	RPi Camera module	5 MP, 1080p, SCI

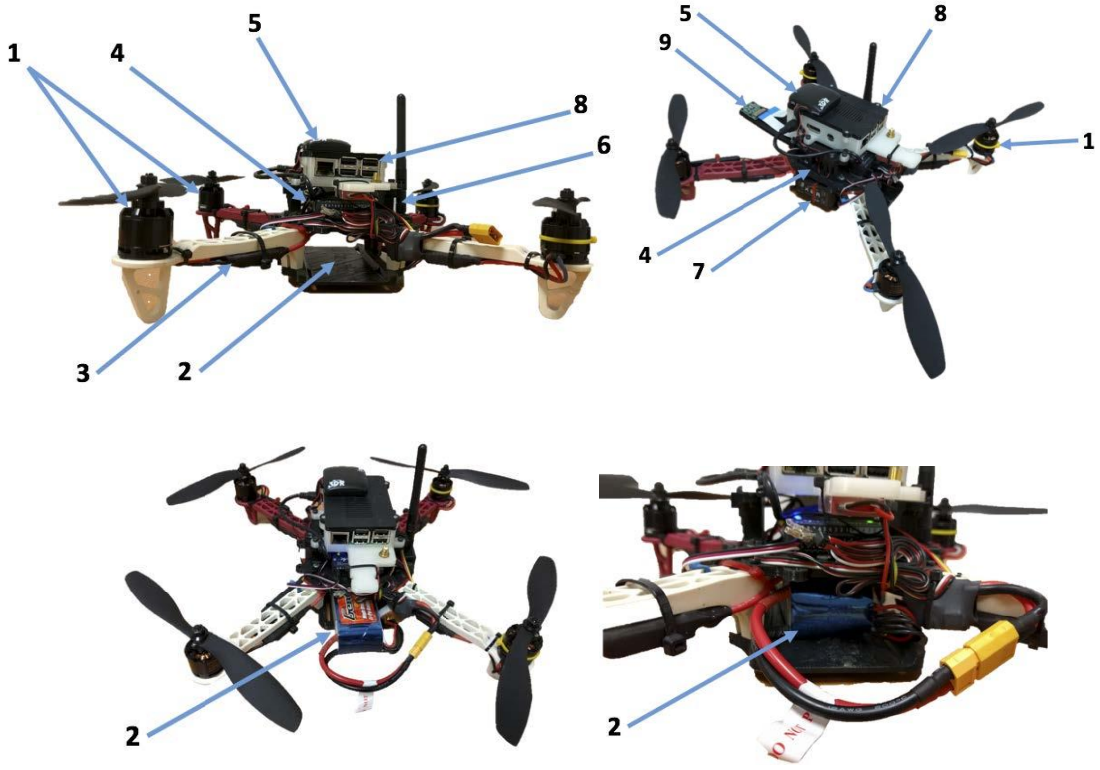

Figura 43: Ubicación de los componentes del multirrotor GRCQuad.

Descripción del sistema de navegación

El sistema de telemetría 3DR 433MHz ofrece una conexión serie emulada mediante puerto USB, lo que permite enlazar cualquier ordenador portátil o dispositivo móvil compatible con dicho dispositivo. Esto incluye a los sistemas operativos más comunes, incluyendo Windows, Linux y Android.

Para que el dispositivo móvil se convierta en un sistema de navegación, además de la interfaz de telemetría 3DR, debe tener instalada una aplicación de control de misión genérica. Actualmente, las más destacadas son APM Mission Planner 2 (para PC) y DroidPlanner 2 (para Android).

Estas aplicaciones ofrecen la funcionalidad básica de cualquier sistema de navegación, incluyendo:

- Datos de telemetría en tiempo real.
- Definición de rutas mediante introducción de *waypoints*.
- Selección de comandos de misión.
- Archivos de log de misión.

A continuación, se muestra la interfaz de cada una de estas aplicaciones.

APM Mission Planner 2

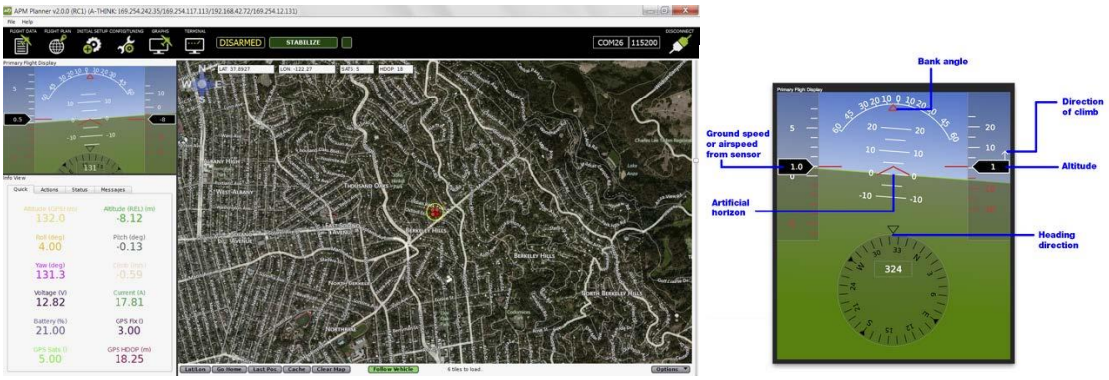

Figura 44: Interfaz gráfica de APM Mission Planner 2.

Características de APM Mission Planner 2:

- Introducción de *waypoints* mediante *point-and-click*, usando Google Maps/Bing/Open street maps o el servicio online de mapas de tu elección.
- Selección de comandos de misión mediante desplegables.
- Descarga y análisis de los archivos de log de misión.
- Configuración de los ajustes de APM para tu chasis.
- Proporciona la interfaz con un simulador de vuelo de PC para crear un completo simulador UAV hardware-in-the-loop.
- Proporciona la salida del terminal serie de APM.

DroidPlanner 2

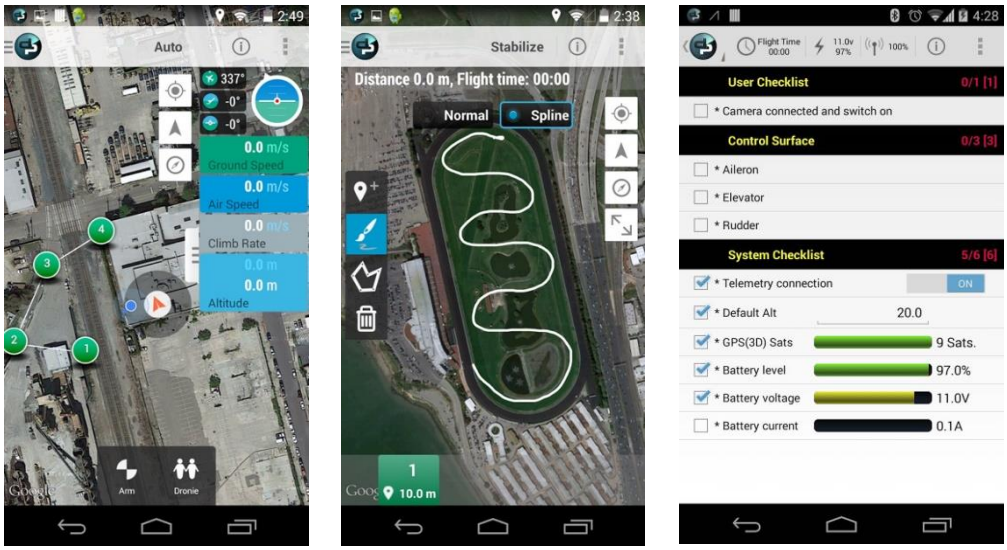

Figura 45: Interfaz gráfica de DroidPlanner 2.

Características de DroidPlanner 2:

- Especialmente diseñado para aeronaves 3DR e Iris.
- Pantalla de telemetría con datos como HUD, batería, RSSI, distancia.
- Botones *"Home"*, *"Land"* y *"Loiter"* fáciles de usar.
- Nuevo modo de control con altitud variable.
- Cambio rápido entre modos.
- Nueva pantalla de planificación para la generación rápida de misiones.
- Herramientas para la edición sencilla de misiones.
- Configuración básica de la TX radio.
- Lista de comprobación pre-vuelo.

Sistema de mando y control

Como equipo de radio control de la estación de tierra se utiliza un transmisor RC genérico que ha sido adaptado a la aeronave.

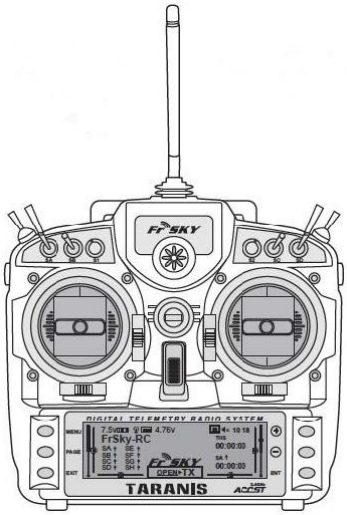

Figura 46: Mando de control remoto.

Especificaciones:

- Transmisor FrSky Taranis X9D Digital Telemetry Radio
- Frecuencia de transmisión: 2,4 GHz
- Número de canales: hasta 16
- Alimentación: 6 a 15V NiMh
- Corriente máxima: 260mA
- Temperatura de operación: -10~60°C
- Pantalla LCD retroiluminación: 212x64 píxeles monocromático
- Memorias: 60 (extensible mediante tarjeta SD)
- Modo de operación: Mode 2 (*Left hand throttle*)
- Alcance: entre 1,5 y 2 km

Anexo II: Configuración del multirrotor GRCQuad

Conexión Pixhawk - Raspberry

El multirrotor GRCQuad tiene preinstalada encima una Raspberry Pi 3. Para que la misma pueda comunicarse con el dron es necesario conectarla primero a la controladora de vuelo (Pixhawk).

El puerto Telem1 de la Pixhawk envía telemetría en tiempo real al transmisor empleado para enlazar con APM Mission Planner o con DroidPlanner. El dispositivo dispone también de un puerto Telem2 adicional que se puede habilitar para transmitir la misma información. Este es el puerto serie que se ha utilizado para conectarla a la Raspberry Pi 3. Para ello basta con conectar la Pixhawk al ordenador mediante USB y habilitar el envío de mensajes desde el programa APM Mission Planner [21].

La Raspberry Pi 3 dispone de un puerto GPIO [22] (General Purpose Input/Output) de 40 pines en el que se puede habilitar un puerto serie al que conectar el puerto Telem2 de la Pixhawk.

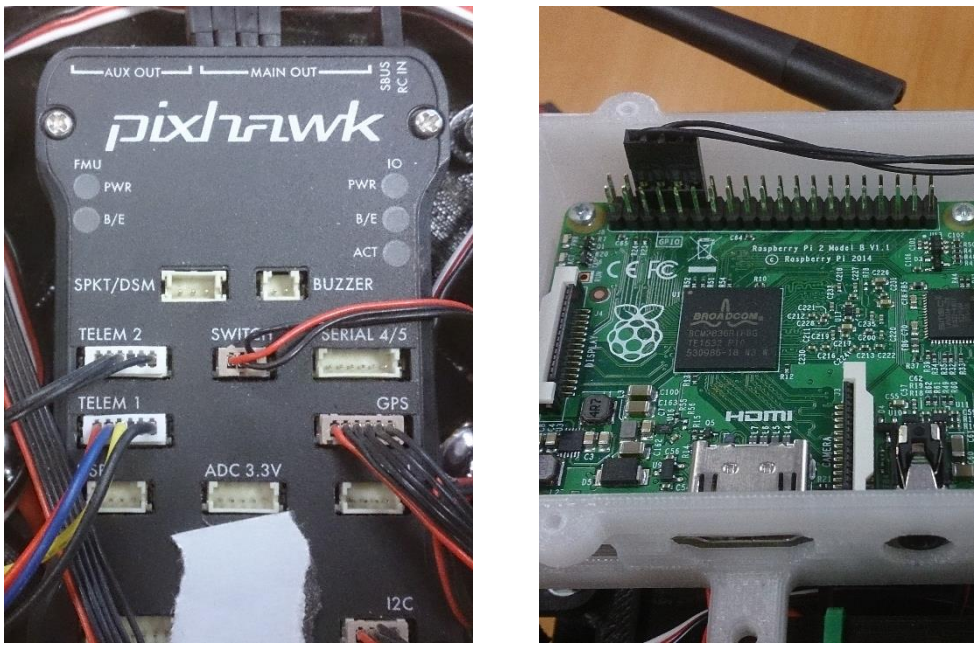

Figura 47: Puertos Telem2 en Pixhawk y GPIO en Raspberry Pi 3.

El modelo 3 de Raspberry Pi tiene una característica singular, no presente en los modelos anteriores. Los pines del puerto GPIO que antes se dedicaban al puerto serie, ahora se dedican al adaptador Bluetooth, impidiendo el correcto funcionamiento de la aplicación *Dronning* y muchas otras aplicaciones desarrolladas para usar el puerto serie *ttyAMA0*.

Para resolver este inconveniente, se puede utilizar dos soluciones:

- Deshabilitar el adaptador Bluetooth y sustituir la salida Bluetooth en GPIO por la salida serie ttyAMA0 [23].
- Habilitar otra salida serie y desactivar el ahorro de energía del dispositivo [24].

La salida serie disponible con la segunda opción está emulada por software, lo que aporta inconvenientes. La velocidad de transmisión por el puerto serie es proporcional a la velocidad del procesador. En caso que la velocidad de la CPU varíe durante el uso del puerto serie, la velocidad de transmisión también lo hace, lo que provoca errores de funcionamiento. Se requiere fijar la velocidad de la CPU, y por lo tanto la velocidad de transmisión del puerto serie.

Teniendo en cuenta que no se necesita el adaptador Bluetooth y los inconvenientes ya planteados, se ha optado por la primera opción. En este caso, basta con editar el fichero /boot/config.txt y modificar las siguientes dos líneas:

```
enable_uart=1
dtoverlay=pi3-miniuart-bt
```

Tras reiniciar, con el siguiente comando se puede comprobar que el puerto serie *ttyAMA0* vuelve a estar operativo:

```
ls -l /dev
```

El esquema de conexión entre ambos dispositivos es sencillo [25]. Es importante tener en cuenta que, por defecto, la Raspberry Pi 3 envía la salida estándar del sistema por el puerto serie, lo que podría provocar un comportamiento inadecuado de la Pixhawk. Para evitar este efecto indeseado hay que realizar tres pasos.

Primero, la herramienta “raspi-config”, en el submenú “Interfacing options”, permite habilitar la salida serie por el puerto GPIO (figura 48).

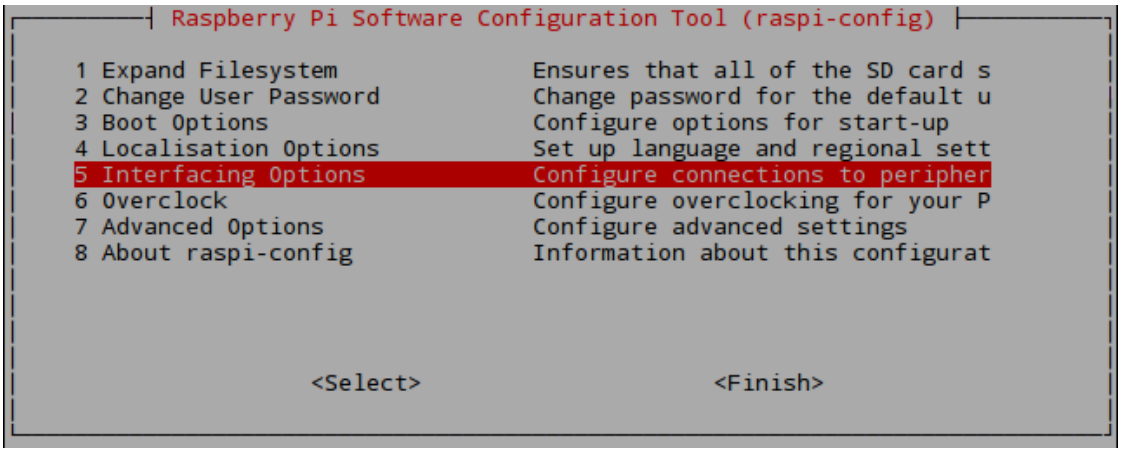

Figura 48: Utilidad raspi-config

Después, se borra el texto “console=serial0,115200” del fichero /boot/cmdline.txt”, para impedir el envío de la salida estándar del sistema al puerto serie, y por último se reinicia el dispositivo.

Una vez configurada la conexión entre ambos dispositivos es necesario instalar la librería utilizada por el programa *Dronning* para leer la información desde el puerto serie: RXTX [26].

La instalación en la Raspberry Pi 3 requiere los siguientes pasos:

1. Crear las carpetas "/home/pi/lib2 y "/home/pi/javalibs"
2. Descargar el fichero "RXTXcomm.jar" desde la web oficial a la carpeta "/home/pi/javalibs"
3. Inserción de la siguientes líneas en el fichero "/etc/environment":
```
JAVA_HOME="/usr/lib/jvm/jdk-8-oracle-arm32-vfp-hflt"
CLASSPATH="/home/pi/javalibs/RXTXcomm.jar"
```
4. Instalación de la versión 2.2pre1:
```
sudo apt-get install librtx-java
```
5. Inserción de las siguientes líneas al final del fichero "~/.bashrc":
```
export JAVA_HOME="/usr/lib/jvm/jdk-8-oracle-arm32-vfp-hflt"
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=/home/pi/javalibs/RXTXcomm.jar
export LD_LIBRARY_PATH=/home/pi/lib
```
6. Creación de enlaces simbólicos al binario de la librería:
```
sudo ln -s /usr/lib/jni/librxtxSerial-2.2pre1.so
     /home/pi/lib/librxtxSerial.so
sudo ln -s /usr/lib/jni/librxtxSerial-2.2pre1.so
     /usr/lib/jvm/jdk-8-oracle-arm32-vfp-
     hflt/jre/lib/arm/librxtxSerial.so
```

Con los pasos 1, 2 y 3 se registra la ubicación de la librería Java y se crea la carpeta para almacenar el binario [27] de la librería RXTX específico para la arquitectura ARM. Como se observa, en la actual versión de Raspbian está instalada por defecto la versión 8 de Java.

El siguiente paso (4) consiste en instalar el binario de la librería. Tal y como se comenta en la página web de descarga [28], la versión de la librería Java y la versión de la librería nativa instalada en el dispositivo no coinciden, lo que genera por la consola un mensaje de advertencia cada vez que se utiliza RXTX, sin que ello suponga ningún riesgo ni produzca ningún tipo de error.

Con los pasos quinto y sexto se hacen visibles la librería java y el binario a todos los programas que se ejecuten en el entorno Java [29].

Configuración de la red Ad-hoc entre drones

Una vez configurada la comunicación entre la aplicación *Dronning* y el dron, es necesario habilitar la comunicación entre los dos drones y el ordenador portátil, según la configuración mostrada en la figura 12 del apartado 6.

Ambos drones funcionan con el sistema operativo Raspbian, una versión de Linux basada en la distribución Debian [30]. Dicho sistema ya tiene integrados los controladores necesarios para el adaptador WiFi escogido, por lo que tan solo es necesario configurar la red. Por otro lado, la placa base ya tiene integrado un adaptador WiFi.

El adaptador incluido con el sistema no es adecuado para la toma de datos, pues se pretende utilizar la banda de los 5 GHz en vez de la de 2,4 GHz. En su lugar, se utilizará un Alfa AWUS051NH dual band con una antena de 5dBi.

Teniendo en cuenta la movilidad de los drones, no es posible utilizar un punto de acceso al que conectarlos, lo que hace necesario configurar la red en modo Ad-hoc.

Solamente se pretende tomar datos, por lo que es suficiente utilizar IPs estáticas.

El adaptador WiFi integrado tiene asociada la interfaz de red wlan0, mientras que el que se utilizará tiene la wlan1, por lo que la configuración a introducir en el fichero "/etc/network/interfaces" es la siguiente [31]:

```
auto wlan1
iface wlan0 inet static
      address 192.168.1.1
      netmask 255.255.255.0
      wireless-channel 36
      wireless-essid DRONNING5G
      wireless-mode ad-hoc
```

, donde la dirección IP es 192.168.1.1 para el dron servidor y 192.168.1.2 para el dron que ejerce el rol de cliente. A estos efectos, se ha denominado servidor al dron que envía paquetes de datos con el protocolo UDP, y cliente al dron que los recibe, detectando aquellos que se han perdido durante la transmisión. La IP 192.168.1.3 corresponde al ordenador portátil, y DRONNING5G es el nombre escogido para la red Ad-hoc.

Las pruebas se han realizado en la banda de los 5,18 GHz, que corresponde al canal 36.

Por último, se ha creado un script interactivo para cambiar entre los modos infraestructura, Ad-hoc 5 GHz y Ad-hoc 2,4 GHz por si fuese necesario realizar otro tipo de pruebas, o conectarse a internet con el modo infraestructura.

Aunque la configuración indicada debería ser suficiente, en el caso concreto del adaptador utilizado existe un problema adicional. Al arrancar la Raspberry Pi 3 y cargar los controladores del adaptador, ésta no es capaz de levantar correctamente el interfaz wlan1 en modo Ad-hoc en la banda de los 5 GHz (no hay ningún problema en modo infraestructura). En su lugar, entra en modo Ad-hoc en la banda de 2,4 GHz.

La solución consiste en utilizar el siguiente comando para cambiar la frecuencia utilizada, de forma automatizada, justo antes de lanzar la aplicación *Dronning*:

```
sudo iwconfig wlan1 freq 5.18G
```

Por otro lado, se ha comprobado que el identificador asignado al adaptador WiFi externo (wlan1) y el del adaptador interno (wlan0) se intercambian aleatoriamente durante el inicio del sistema, lo que impide la ejecución correcta de los scripts de configuración. La solución [51] consistió en hacer permanente la asignación de identificadores para que no variase durante el arranque.

Otro problema detectado ha sido un defecto en la configuración por defecto del sistema operativo Raspbian [32]. Los permisos de la aplicación "ping" son incorrectos, lo que impide utilizar dicha utilidad. La solución es sencilla:

```
sudo chmod u+s /bin/ping
```

Por último, aunque no es estrictamente necesario, se ha optado por desactivar durante el inicio el interfaz de red WiFi no utilizado:

```
sudo ifconfig wlan0 down
```

Arranque automático de la aplicación

La configuración introducida hasta el momento es suficiente para que la aplicación *Dronning* funcione correctamente. Sin embargo, para realizar tomas de datos en campo es conveniente que la aplicación arranque automáticamente con el sistema.

Añadiendo la siguiente línea al final del fichero "~/.config/lxsession/LXDE-pi/autostart" se ejecuta el fichero "inicio.sh" en un terminal cuando la interfaz gráfica está totalmente cargada [40]:

```
@/usr/bin/lxterminal -e /home/pi/Desktop/inicio.sh
```

Dicho fichero es el que lanza los comandos ya indicados y la aplicación *Dronning*.

Hay distintas fases durante el arranque del sistema en las que se puede lanzar cualquier script. Se ha escogido lanzarlo aquí porque cuando arranca la interfaz gráfica también lo han hecho ya los interfaces de red.

Anexo III: Instalación de las máquinas virtuales con SITL

Software In The Loop (SITL) es un simulador de dron virtual, que puede instalarse bajo Windows, Linux, o una máquina virtual con el sistema operativo Linux.

Por otro lado, para controlar el dron se utiliza la aplicación MAVProxy, que se conecta al dron por TCP y proporciona una consola e interfaces UDP y TCP que permiten enviar órdenes a los drones y recibir información de vuelo de los mismos.

Dado que MAVProxy permite la ejecución de una única instancia en el sistema, ha sido necesario instalar SITL en la tercera modalidad, dentro de máquinas virtuales.

El simulador desarrollado requiere instalar previamente el entorno necesario.

- VirtualBox [33]. Es un software de virtualización de licencia GPL. Permite crear la máquina virtual sobre la que funciona SITL.
- Vagrant [34]. Facilita la creación de un entorno de desarrollo sobre una máquina virtual, es decir, facilita la creación de una máquina virtual Linux y la instalación en la misma de SITL y MAVProxy.
- Git [35]. Sistema de control de versiones. Esta herramienta permite descargar Ardupilot del repositorio público en el que se encuentra, para realizar la instalación.
- SSH. Es necesario disponer del programa de línea de comandos para conexiones seguras, ya que la comunicación entre el *host* y la máquina virtual se realiza mediante conexión segura y por línea de comandos.

El desarrollo se ha realizado sobre Ubuntu 16.10, por lo que no ha sido necesario instalar SSH, aunque sí que ha sido necesario instalar VirtualBox, Vagrant y Git:

```
sudo dpkg –i vagrant_1.9.1_x86_64.deb
sudo dpkg –i virtualbox-5.1_5.1.14-112924~Ubuntu\
                                              ~yakkety_amd64.deb
sudo VBoxManage extpack install Oracle_VM_VirtualBoxExtension\
                              _Pack-5.1.14-11.29.24.vbox-extpack
sudo apt-get install git
git config –global core.autocrlf false
```

Las máquinas virtuales se conectan con el exterior mediante NAT. Sin embargo, resulta conveniente utilizar una red “solo-anfitrión” para comunicar los drones con el simulador. Para ello, en VirtualBox se abre el gestor de redes y se crea una nueva red.

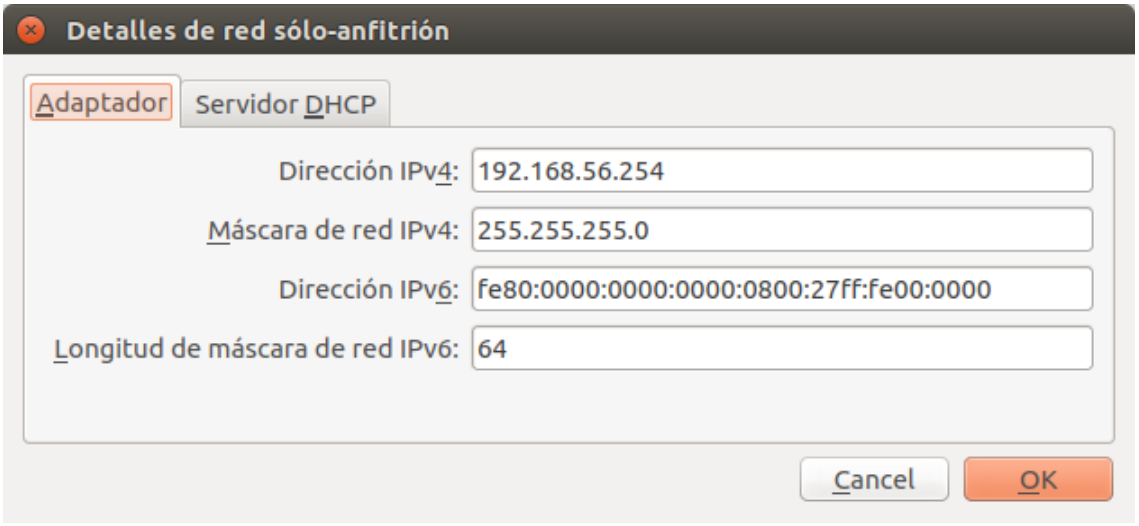

Figura 49: Red sólo-anfitrión entre los drones y el simulador.

En la figura 49 se muestran los parámetros que se pueden configurar en la nueva red. Se decidió utilizar la red 192.168.56.0/24, con la IP 192.168.56.254 para el host principal, donde se aloja el simulador, dejando las IPs 192.168.56.1 a la 192.168.56.253 para drones.

También es conveniente, aunque no imprescindible, desactivar el servicio DHCP, pues las máquinas virtuales tendrán una IP estática.

Tras realizar las instalaciones necesarias, hay que crear las máquinas virtuales siguiendo los siguientes pasos [36]:

1. Clonación de ArduPilot en local (incluye SITL y MAVProxy).

```
mkdir 10*
cd 10
git clone https://github.com/ArduPilot/ardupilot.git
```

*La carpeta que contiene una máquina virtual ha de tener como nombre un valor numérico entre el 1 y el 253. Éste, debe coincidir con el cuarto octeto de la IP que tendrá la máquina virtual en la red que comunica a todos los drones con el simulador, y que se configura en el paso 3. Esta solución facilita al simulador localizar las máquinas virtuales que contienen un dron y son aptas para su uso. Además, todas las carpetas contenedoras han de estar dentro de una misma carpeta.

2. Primer arranque para actualizar SITL.

```
cd ardupilot
vagrant up
vagrant ssh
git submodule update --init --recursive
exit
vagrant halt
```

Entrando en el proyecto clonado, con el comando “vagrant up” se arranca la máquina virtual. Vagrant se encarga de que la carpeta Ardupilot se monte en el sistema de ficheros de la máquina virtual.

La orden “vagrant ssh” entra a la máquina virtual por SSH y permite ejecutar el siguiente comando para actualizar SITL a la última versión. Por último, tras cerrar la conexión SSH, “vagrant halt” permite detener la máquina virtual.

3. Adición de un adaptador de red.

Para añadir un adaptador a la máquina virtual, y que lo gestione automáticamente la aplicación Vagrant, hay que modificar el fichero “ardupilot/Vagrantfile” añadiendo lo siguiente al final [37]:

```
Vagrant::Config.run do |config|
      config.vm.network :hostonly, “192.168.56.X”
end
```

… donde la “X” representa el último octeto de la IP de la máquina virtual en concreto, y que debe coincidir con el nombre de la carpeta que la contiene, como se ha explicado con anterioridad.

4. Cambio de contraseña.

El simulador accede a la máquina virtual mediante SSH, para ejecutar el dron virtual. Sin embargo, la contraseña del usuario por defecto (ubuntu) no es de conocimiento público, por lo que es necesario cambiarla:

```
vagrant up
vagrant ssh
sudo su
passwd ubuntu
vagrant
exit
vagrant halt
```

Con la orden “passwd” se cambia la contraseña del usuario “ubuntu” al valor “vagrant”. Éste, es el valor utilizado por la aplicación *Dronning*. En caso de querer emplear alguna otra contraseña es necesario modificar el correspondiente parámetro interno de la aplicación.

5. Primer arranque del dron virtual.

Es necesario realizar una primera ejecución del dron para compilar las librerías de SITL e inicializar los parámetros por defecto del dron:

```
vagrant up
vagrant ssh
sudo /vagrant/Tools/autotest/sim_vehicle.py -w
ctrl+d
exit
vagrant halt
```

El tercer comando arranca el dron y fuerza la inicialización de todos los parámetros del mismo. Una vez el dron está en funcionamiento, con “ctrl+d” se detiene su ejecución y se liberan todos los recursos utilizados por SITL y por MAVProxy.

A partir de este momento, el dron virtual ya está totalmente operativo para el simulador, que se encargará de arrancar y apagar tanto el dron como la máquina virtual. En cualquier momento, si se desea, el dron se puede lanzar manualmente con el siguiente comando, desde un terminal de la máquina virtual:

```
sudo /vagrant/Tools/autotest/sim_vehicle.py -v ArduCopter -j4 –l\
                    39.48,-0.341,20,0 --out 192.168.56.254:14550
```

…donde:

-v ArduCopter. Especifica que se tiene que emular un multirrotor, no un aeroplano.

-j4. Usar 4 hilos de ejecución para la compilación del dron, si es que es necesario.

-l 39.48,-0.341,20,0. Coordenadas donde debe aparecer el dron (latitud, longitud, altitud y orientación respecto al norte).

--out 192.168.56.254:14550. IP y puerto adicional por el que MAVProxy debe enviar y recibir información y comandos de control del dron. Al poner la IP de la red “solo-anfitrión”, donde también está el simulador, se permite la comunicación entre el simulador y los drones.

Anexo IV. Cálculo de la transformación de coordenadas de las ortofotos

El simulador dibuja los drones y la misión que deben trazar sobre el fondo que Java rellena por defecto, un rectángulo gris. Esta configuración permite visualizar muy fácilmente lo que hacen los drones, aunque resulta muy poco amigable para el usuario. Por este motivo, se decidió utilizar la API de Google Static Maps [38] y descargar ortofotos que representan la ubicación geográfica en la que se ha desplegado el dron virtual, es decir, el emplazamiento escogido al definir la misión en Google Earth.

Es necesario realizar cálculos matemáticos para transformar las coordenadas entre distintos sistemas de representación.

Por un lado, se tienen las coordenadas en pantalla, que se representan en píxeles. Por otro están las coordenadas UTM, representadas en metros, y que permiten calcular la distancia entre dos drones y otros parámetros. Por último, se dispone de las coordenadas geográficas (latitud y longitud), que son las que se obtienen de los mapas de Google Static Maps y de los drones.

Las imágenes descargadas tienen como referencia su centro. En [39] se ha hallado una solución para determinar las coordenadas geográficas de un pixel cualquiera de la imagen a partir de su centro y el zoom de la misma. Por otro lado, en [42] se calcula la transformación entre coordenadas geográficas y UTM.

Una limitación importante, es que no se puede descargar de Google Static Maps una imagen de más de 640x640 píxeles de forma gratuita. Por este motivo, en caso de que la resolución de pantalla permita un área de dibujo mayor, es necesario descargar más de una imagen, uniéndolas como si fuesen un mosaico.

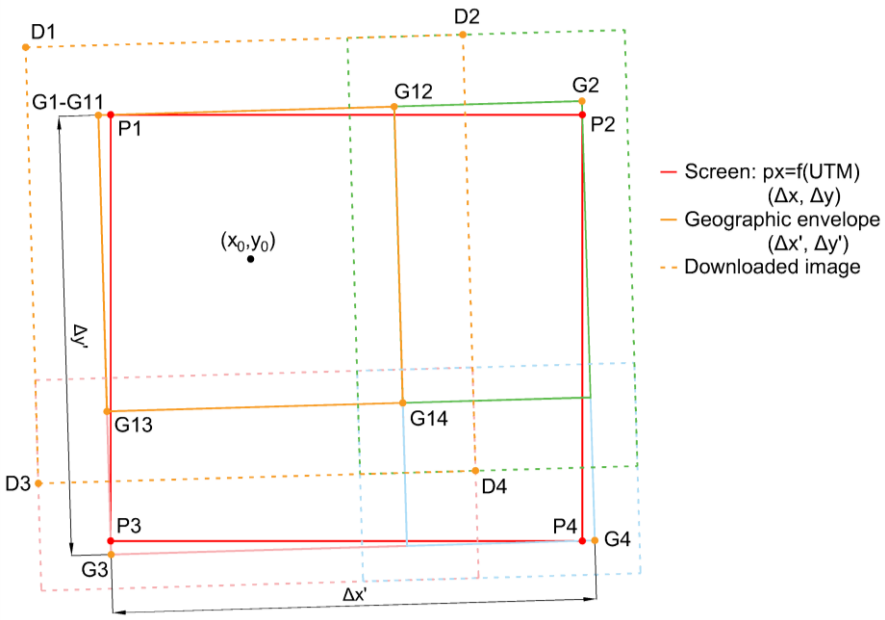

Figura 50: Transformaciones geométricas para dibujar la imagen de fondo.

En la figura 50 se muestran las transformaciones geométricas que es necesario realizar. Para la explicación, se parte de la hipótesis de que el panel de dibujo tiene más de 640 píxeles en ambas dimensiones y que, por lo tanto, es necesario construir el fondo del panel a partir de un mosaico de 2x2 imágenes.

En rojo (P1, P2, P3 y P4), se representa el rectángulo del panel del dibujo, que se mide en píxeles.

Se podría dibujar la pantalla en coordenadas geográficas o en coordenadas UTM. Se ha escogido la visualización en coordenadas UTM porque en estas coordenadas se puede medir distancias, mientras que en coordenadas geográficas no. Por lo tanto, en pantalla se representa una región rectangular del espacio, en coordenadas UTM, y en la que todos los puntos de la misma coordenada Y se visualizan como una línea horizontal de píxeles.

Inclinado respecto al anterior rectángulo, se representa la envolvente en coordenadas geográficas que incluye todos los puntos en coordenadas UTM que se van a representar en pantalla (G1, G2, G3 y G4). Como se observa, entre las proyecciones UTM y geográfica hay una rotación, que depende de la distancia al ecuador y de la distancia a la vertical que divide en dos cada una de las zonas de la proyección UTM.

Como la pantalla tiene más píxeles que el máximo permitido por Google Maps, la anterior envolvente hay que subdividirla en regiones más pequeñas (ej/ G11, G12, G13 y G14) de un tamaño menor o igual a los 640x640 píxeles o su equivalente en metros en la proyección UTM. De esta forma, se construye el mosaico de imágenes que es necesario descargar.

Sin embargo, existe una complicación adicional. Google Static Maps define distintos niveles de zoom, es decir, no se puede descargar una imagen a la escala exacta que se desea, sino a la escala más próxima que, en una imagen de 640x640 píxeles, incluye la zona que se desea dibujar.

La solución consiste en calcular los metros a los que equivalen los 640 píxeles en la escala actual de visualización, transformarlos en coordenadas geográficas, y por último calcular cuál es el zoom mayor en el que cabe una región de las coordenadas geográficas previamente obtenidas.

Una vez definido el zoom, se procede a la descarga de las imágenes, tomando el centro de la misma como referencia (ej/ x_0, y_0). Como se ha explicado, el uso de un zoom no exacto provoca que la imagen descargada incluya una región mayor (ej/ D1, D2, D3 y D4). Por este motivo, se calculan las coordenadas geográficas de la esquina superior izquierda (D1), lo que permitirá desplazar la imagen para su visualización. También se calcula el ángulo de rotación entre los sistemas de coordenadas, para rotar la imagen.

El proceso completo consta de los siguientes pasos. Se indica “px” para representar coordenadas de pantalla en píxeles y “UTM” para representar coordenadas en dicha proyección.

1. px a UTM de las esquinas de la pantalla (P1, P2, P3 y P4).
2. Obtención de las dimensiones UTM de la pantalla:
   Δx = P2-P1, Δy = P1-P3
3. Cálculo coordenadas geográficas de la envolvente:
   UTM a geográficas de la envolvente
   Envolvente que incluye los puntos calculados (G1, G2, G3 y G4)
4. Transformación de geográficas a UTM y a px (G1, G2, G3 y G4).

5. Si Las dimensiones G1-G2 y/o G1-G3 superan los 640 píxeles, se define el mosaico de imágenes que es necesario descargar.
6. Obtención del zoom a utilizar en Google Maps:
   Cálculo del centro de la primera pieza del mosaico.
   Por iteración, determinar el mayor zoom en el que cabe dicha pieza.
7. Cálculo del centro de todas las piezas del mosaico.
8. Descarga de todas las imágenes y cálculo de la esquina superior izquierda en UTM, la escala de dibujo y el ángulo de rotación a aplicar de coordenadas geográficas a UTM.
9. Dibujado de las imágenes:
   Aumento de brillo de la imagen, para que resalte menos.
   Escalado, respecto a la esquina superior izquierda.
   Rotación, respecto a la esquina superior izquierda.
   Traslación de la esquina superior izquierda a las coordenadas px obtenidas a partir de las UTM calculadas en el punto 8.

Anexo V. Cálculo del reposicionado del dron para ceder el paso a otro dron

En el apartado 8.5 se explica la solución implementada para detener el dron, en caso de detectarse riesgo de colisión.

Cabe la posibilidad de que detenerse no sea suficiente, pues el dron puede permanecer en la ruta que debe recorrer el otro dron. En tal caso, es necesario que se aparte a un lado antes de darle permiso para que pase.

La ruta que sigue el dron a esquivar está definida como una sucesión de puntos. Por lo tanto, si es necesario, hay que apartar el dron menos prioritario de la línea que forman los puntos de la ruta que ha de seguir el dron más prioritario.

La figura 51 muestra gráficamente las matemáticas implicadas en el proceso.

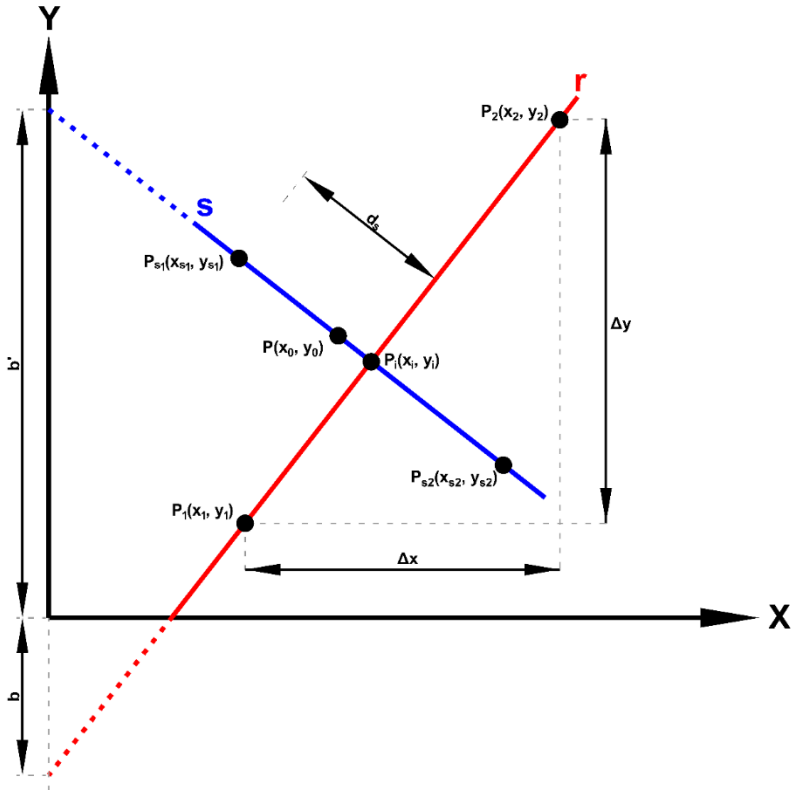

Figura 51: Cálculo de la ubicación del punto seguro P_s.

Sea r la recta que define la traza que debe seguir el dron con mayor prioridad de paso y P la ubicación del dron con menor prioridad de paso.

La recta r que sigue el dron con mayor prioridad tiene la forma:

$$\Delta x = x_2 - x_1, \quad \Delta y = y_2 - y_1$$

$$y_r = \frac{\Delta y}{\Delta x} x_r + b$$

El siguiente paso es definir la recta s perpendicular a la primera y que pasa por la posición actual P del dron con menor prioridad:

$$y_s = -\frac{\Delta x}{\Delta y} x_s + b'$$

Al ser el punto P parte de la recta, se puede despejar el valor de b':

$$b' = y_0 + \frac{\Delta x}{\Delta y} x_0$$

, y sustituyendo en la ecuación inicial:

$$y_s = -\frac{\Delta x}{\Delta y} x_s + y_0 + \frac{\Delta x}{\Delta y} x_0 = y_0 - \frac{\Delta x}{\Delta y}(x_s - x_0)$$

La distancia de P al punto de intersección P_i entre las rectas r y s determinará si el dron tiene o no que apartarse para dejar paso al dron de mayor prioridad. Por lo tanto, primero hay que obtener las coordenadas de la intersección entre las rectas P_i.

$$y_1 + \frac{\Delta y}{\Delta x}(x_i - x_1) = y_0 - \frac{\Delta x}{\Delta y}(x_i - x_0)$$

Para obtener $P_i(x_i, y_i)$, la primera coordenada se despeja de la anterior ecuación y la segunda coordenada de la ecuación de cualquiera de las dos rectas:

$$P_i = \left(\frac{y_0 - y_1 + \frac{\Delta x}{\Delta y} x_0 + \frac{\Delta y}{\Delta x} x_1}{\frac{\Delta y}{\Delta x} + \frac{\Delta x}{\Delta y}}, \quad y_1 + \frac{\Delta y}{\Delta x}(x_i - x_1)\right)$$

Si P_i no se encuentra en el segmento desde P_1 a P_2, significa que el dron no va a pasar cerca de ese segmento de la recta, y que no es necesario apartarse. En caso contrario, si la distancia de P a P_i es inferior a la distancia de seguridad, es necesario apartar el dron a lo largo de la recta s.

El punto $P_s(x_s, y_s)$ fuera de la zona de seguridad será aquel al que habrá que desplazarse. Se obtiene por proporcionalidad de triángulos entre los puntos P_1 y P_2 de la traza r del dron a esquivar, y el tramo de s entre el punto P_s y la intersección P_i entre las rectas r y s.

$$\frac{d}{|\Delta y|} = \frac{d_s}{|x_s - x_i|}$$

…donde d_s es un parámetro conocido y d se obtiene con la siguiente ecuación:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Despejando, se obtienen la coordenada x_s del punto P_s. El signo variará según el punto al que conviene apartarse esté a la izquierda o a la derecha del punto de intersección:

$$P_s = \left(x_i - \frac{d_s}{d}|\Delta y|, \quad y_0 - \frac{\Delta x}{\Delta y}(x_s - x_0)\right) \quad para\ x_0 \leq x_i$$

$$P_s = \left(x_i + \frac{d_s}{d}|\Delta y|, \quad y_0 - \frac{\Delta x}{\Delta y}(x_s - x_0)\right) \quad para\ x_0 > x_i$$

Anexo VI. Configuración de las pruebas de validación del protocolo

En todos los casos, se mantuvo la configuración por defecto de la aplicación, tal y como se muestra en las figuras 3 y 26.

La velocidad de vuelo de ambos drones fue de 10 m/s. La única excepción es el dron 1 en la prueba de adelantamiento, donde se utilizó una velocidad de 5 m/s para permitir que el dron 2 le pudiese alcanzar.

En cuanto a la misión programada para cada dron, los waypoints se detallan a continuación:

Tabla 10: Misión de drones que se cruzan en perpendicular.

dron	waypoint	longitud, latitud
1	1	-0.349832986208919, 39.48002291664762
	2	-0.3323858446839183, 39.47987251755895
2	1	-0.3412982989545824, 39.48678405952241
	2	-0.3415143406128095, 39.47327648876747

Tabla 11: Misión en aproximación estando enfrentados.

dron	waypoint	longitud, latitud
1	1	-0.34888297714526, 39.4837081827146
	2	-0.3324118588025864, 39.47928256003704
2	1	-0.3324118588025864, 39.47928256003704
	2	-0.34888297714526, 39.4837081827146

Tabla 12: Misión en un adelantamiento.

dron	waypoint	longitud, latitud
1	1	-0.3433909417208917, 39.48170738501614
	2	-0.333153304173787, 39.47877694179415
2	1	-0.3501843261128411, 39.48365486619155
	2	-0.3338898583676667, 39.4789786696093

Tabla 13: Misión en aproximación esviada.

dron	waypoint	longitud, latitud
1	1	-0.347882127992527, 39.48295200366152
	2	-0.3314736140643575, 39.47838939809228
2	1	-0.3483561201909657, 39.48155371269473
	2	-0.3310495978325404, 39.47992312749467

Tabla 14: Misión en aproximación esviada y en sentido opuesto.

dron	waypoint	longitud, latitud
1	1	-0.347882127992527, 39.48295200366152
	2	-0.3314736140643575, 39.47838939809228
2	1	-0.3310495978325404, 39.47992312749467
	2	-0.3483561201909657, 39.48155371269473

Tabla 15: Misión de interceptación durante la supervisión de un campo de cultivo.

dron	waypoint	longitud, latitud
1	1	-3.664540323594405, 38.97444382788777
	2	-3.6677468861025, 38.96654886905427
	3	-3.664389398497631, 38.95831735217047
	4	-3.657999652473852, 38.95866936089705
	5	-3.657872341541843, 38.96414141302522
	6	-3.630109445875158, 38.97322594278469
2	1	-3.643510678479164, 38.95889197486271
	2	-3.648118040543879, 38.9713630163089
	3	-3.647414003585168, 38.97182603240679
	4	-3.642697158115292, 38.95914453778239
	5	-3.641613360375142, 38.95948161041606
	6	-3.646600992996737, 38.97245814839685
	7	-3.645570992318419, 38.97300619882866
	8	-3.640692405555906, 38.95977660571751
	9	-3.639554452205075, 38.96032433828423
	10	-3.6447573346121, 38.97389164228434
	11	-3.644051304434251, 38.97486209189205
	12	-3.638578717882912, 38.96087201637219
	13	-3.637819186193823, 38.96179892277714
	14	-3.643128125318257, 38.97549575630062
	15	-3.642149998649467, 38.97629860453259
	16	-3.63711318700811, 38.96272588370956
	17	-3.636244351994544, 38.96361081265158
	18	-3.640847778497308, 38.97693160536763
	19	-3.643726385365353, 38.95914443794658