UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado

Sistema para el control en tiempo real de un enjambre de drones basado en pilotaje manual

Tutores

Hernández Orallo, Enrique

Tavares de Araujo Cesariny Calafate, Carlos Miguel

Autor

Robles Gómez, Juan Roberto

Universitat Politècnica de València

Escuela Técnica Superior de Ingeniería del Diseño

Grado en Ingeniería Aeroespacial

Curso 2020 - 2021

9 de septiembre de 2021

Resumen

En un contexto en el que los drones están en auge tanto a nivel empresarial como lúdico, el concepto de enjambre en el cual diversos drones se coordinan para realizar una tarea gana interés. ArduSim se desarrolla como herramienta para simular y estudiar dichos enjambres. Este simulador permite la creación de protocolos que regulen como los drones del enjambre se coordinan. En este trabajo se intenta mejorar un protocolo preexistente en ArduSim, llamado *FollowMe*, que indica cómo los drones de un enjambre deben seguir a un dron maestro pilotado manualmente. Para ello se emplean algoritmos de predicción de la posición, así como controladores PID.

Palabras clave: Enjambre de drones, ArduSim, control en tiempo real.

Abstract

Currently, the use of drones is raising both at a business and leisure level. Therefore, the concept of swarms in which several drones coordinate to perform a task is gaining momentum. ArduSim is developed as a tool to simulate and study such swarms. This simulator allows the creation of protocols that regulate how the drones in the swarm coordinate. In this dissertation we try to improve a pre-existing protocol in ArduSim, called *FollowMe*, which indicates how the drones in a swarm should follow a manually piloted master drone. This is done using algorithms to predict the position as well as PID controllers.

Key words: Drone swarming, ArduSim, real-time control.

Resum

En un context en el qual els drons estan en auge tant a nivell empresarial com lúdic, el concepte d'eixam en el qual diversos drons es coordinen per a fer una tasca guanya interés. ArduSim es desenvolupa com a eina per a simular i estudiar aquests eixams. Aquest simulador permet la creació de protocols que regulen com els drons de l'eixam es coordinen. En aquest treball s'intenta millorar un protocol preexistent en ArduSim, anomenat *FollowMe*, que indica com els drons d'un eixam han de seguir a un dron mestre pilotat manualment. Per a això s'utilitzen algorismes de predicció de la posició, així com controladors PID.

Paraules clau: Eixam de drons, ArduSim, control en temps real.

Agradecimentos

Me gustaría agradecer este trabajo a mi familia, a mis amigos y a mis profesores. Cada uno de ellos me ha ayudado en el camino.

Gracias a Enrique, Carlos y Jamie por la ayuda que me han dado en la realización de este trabajo.

Gracias a mis amigos, todo es más fácil con vosotros.

Por último, gracias a mi madre, a mi hermana y, en particular, a mi padre. Todo lo que me has enseñado me ha llevado hasta aquí. Espero ser tan buen ingeniero como tú.

Índice general

Índice de figuras

Índice de tablas

Parte I

MEMORIA

Capítulo 1

Introducción y objetivos

1.1. Contexto histórico de los enjambres de drones

Las aeronaves pilotadas de forma remota, o RPAS, por sus siglas en inglés (*Remotely Piloted Aircraft System*), conocidos popularmente como «drones», han ido ganando peso en el sector aeronáutico. Los avances tecnológicos en telecomunicaciones y electrónica han permitido su desarrollo, así como una reducción en su coste. De esta forma, el uso de drones como el mostrado en la Figura 1.1 se ha extendido con fines comerciales o lúdicos.

Figura 1.1: Dron moderno. (2019) [1]

.

No obstante, los primeros usos de estas aeronaves se remontan a la Primera Guerra Mundial [1], donde se crearon ciertos RPAS que eran controlados por radio. En la Segunda Guerra Mundial su desarrollo sufrió un fuerte impulso, como en el resto del sector aero-

náutico, hasta llegar a finales del siglo XX, momento en el que su empleo se extendió.

En la actualidad, los drones están suponiendo una revolución en ciertos sectores, principalmente el transporte y reparto de mercancía [2]. Los drones están siendo empleados como medio de reparto de medicinas, comida y otros paquetes gracias a su rapidez y su capacidad para llegar a zonas de difícil acceso. Uno de los ejemplos más conocidos se trata de *Amazon Prime Air*, que está desarrollando un servicio de reparto mediante este tipo de drones, entregando paquetes de menos de 2,25 kg en menos de media hora [3].

Sin embargo, estas aeronaves tienen ciertas limitaciones de carga máxima que pueden transportar o de autonomía de vuelo, entre otros. Es por eso que el concepto de enjambre cobra importancia. El vuelo en enjambre permite que los drones cubran una mayor superfície o que puedan transportar un mayor carga de pago [4], aumentando de esta forma las capacidades que tendría un solo dron.

Estos enjambres ya se emplean en la actualidad en diferentes sectores. Por ejemplo, en agricultura se emplean para medir la humedad del terreno o el nivel de clorofila de los cultivos, gracias a las imágenes del terreno que pueden tomar, así como para controlar plagas en dichos cultivos [5]. También se están desarrollando enjambres para el sector militar y para los servicios de búsqueda y rescate, así como la extinción de incendios [6].

1.1.1. Normativa actual sobre enjambres en España

En la actualidad, según explica la Agencia Estatal de Seguridad Aérea (AESA) a través de la respuesta a una consulta efectuada a la División de Sistemas de Aeronaves No Tripuladas (UAS), en España se podrían realizar operaciones civiles empleando enjambres si éstas se encuentran al amparo de una autorización operacional emitida dentro de la categoría específica. La respuesta de AESA se puede consultar en el Anexo.

La categoría específica es una de las tres categorías de operaciones con drones, además de la abierta y la certificada [7]. Esta categoría requiere que se analizen los riesgos de la operación según el Artículo 11 de *Reglamento de Ejecución (UE) 2019/947*. Para ello, se propone la metodología SORA, siglas de *Specific Operations Risk Assessment*, basada en 10 pasos [8].

En esta evaluación de los riesgos se debe especificar el nivel de intervención del piloto, que en este caso controla a uno de los drones del enjambre, como se explicará posteriormente. Se debe especificar también si el piloto tiene visión directa con el enjambre, es decir, vuelo VLOS (*Visual Line Of Sight*) o si no la tiene, siendo vuelos BVLOS (*Beyond Visual Line Of Sight*) o ELOS (*Extended Visual Line Of Sight*).

Además, se tienen que indicar otros parámetros relacionados con el vuelo como la altitud de vuelo del enjambre, su localización o si el terreno está controlado. Igualmente se deben incluir ciertos datos sobre el enjambre como el número de aeronaves implicadas, el protocolo que emplean para comunicarse o la distancia mínima entre ellas.

Siguiendo la metodología SORA, se identifican los riesgos en tierra y aire, así como las mitigaciones necesarias en la operación. Con esta información, se obtiene el SAIL (*Specific Assurance and Integrity Levels*), es decir, los niveles de riesgo asociados a la operación.

Finalmente, se obtienen los OSOs (*Operational Safety Objectives*). Estos objetivos de seguridad operacional se alcanzan cumpliendo ciertos requisitos, que dependen del nivel de robustez exigido según el SAIL (el riesgo) de la operación.

Por otro lado, además del seguimiento de la metodología SORA para la obtención de la autorización, AESA indica que se debe cumplir con el *Real Decreto 1919/2009* de 11 de diciembre, con el *Real Decreto 1036/2017* y el *Real Decreto 1180/2018*.

El primer decreto regula las demostraciones aéreas civiles, en caso de que se realice una demostración anunciada públicamente. El segundo, expone disposiciones en materia de seguridad pública o restricciones al vuelo de UAS por motivo del lugar de operación. El tercero, recoge las reglas del aire y es el decreto por el que se desarrolla el Reglamento del Aire, que sigue siendo de aplicación en operaciones con aeronaves no tripuladas.

1.2. Introducción al simulador *ArduSim* y antecedentes

En este contexto dentro del sector de los drones aparece el simulador ArduSim. ArduSim es un simulador de vuelo en tiempo real, desarrollado originalmente por Francisco Fabra dentro del Grupo de Redes de Computadores (GRC) que permite desarrollar protocolos de coordinación de drones para formar enjambres [9]. Está programado en Java, por lo que es el lenguaje que se va a emplear en el desarrollo de este trabajo.

ArduSim simula el vuelo del enjambre en tiempo real, siguiendo las instrucciones marcadas por el protocolo. Además, emula la red WiFi en la banda de los 5 GHz que emplea el enjambre para comunicarse [9].

Por otro lado, se usa el protocolo MAVLink (*Micro Air Vehicle Link*) en la comunicación

con la controladora de vuelo de cada dron [9], que destaca por su eficiencia, su fiabilidad y su capacidad para ser empleado en diferentes microcontroladores y con diferentes lenguajes de programación [10].

ArduSim está descrito con detalle en el artículo *ArduSim: Accurate and real-time multicopter simulation* [11], donde se explica la importancia de la creación de un simulador de estas características. Como ya se ha visto, el sector de los drones se encuentra en auge, pero el desarrollo de nuevos protocolos empleando drones reales supone una gran inversión de dinero que podría reducirse empleando simuladores.

Además, también se explican ciertas características del simulador, como la capacidad de desplegar los protocolos desarrollados en drones reales o la posibilidad de simular de forma realista la comunicación entre drones empleando antenas de 5 dBi, de forma que se puedan estudiar las pérdidas de información provocadas por la distancia entre los drones del enjambre [11].

Por último, en este artículo se realiza un estudio del rendimiento del simulador en función de diferentes especificaciones de *hardware*.

Otro artículo que se debe comentar antes de explicar los objetivos de este trabajo es *Automatic system supporting multicopter swarms with manual guidance* [12].

En este trabajo se presenta el protocolo *FollowMe*, el cual se detallará posteriormente. Este protocolo trata de coordinar un enjambre en el cual uno de los drones es pilotado de forma manual por un piloto cualificado. Este dron es conocido como maestro, mientras que el resto de los drones del enjambre son drones esclavos que tratan de seguir al maestro manteniendo una formación determinada.

Para ello, la solución propuesta por *FollowMe* es el envío de la localización del maestro a los esclavos, de forma que éstos puedan calcular la posición dentro del enjambre en la que deben estar y dirigirse hacia dicha localización.

En el artículo se expone que el protocolo funciona, si bien el *lag* en las comunicaciones y ciertas fluctuaciones en el rumbo del maestro provocan errores en la posición de los esclavos [12]. También se explica que el dron maestro debe enviar la posición a los esclavos cada 1 segundo, puesto que una mayor frecuencia en el envío de la posición tiende a generar errores.

1.3. Objetivos

El objetivo de este trabajo es la creación de un nuevo protocolo que permita el control en tiempo real del enjambre de drones mediante un dron maestro pilotado manualmente. Para ello, se partirá del protocolo *FollowMe* y se propondrán diferentes soluciones que intenten corregir los problemas descritos por sus creadores anteriormente.

En definitiva, se comenzará explicando *FollowMe* con detalle. Se realizarán simulaciones para obtener los resultados del protocolo y poder compararlos con los de los nuevos protocolos desarrollados. Posteriormente, se expondrán las nuevas propuestas para corregir los problemas de *FollowMe* y, finalmente, se analizarán los resultados y se comentarán las conclusiones.

Capítulo 2

El protocolo *FollowMe*

Como ya se ha comentado, el protocolo *FollowMe* consiste en que el maestro envíe su posición a los esclavos para que estos puedan calcular en qué posición deben estar. Posteriormente, los esclavos se mueven hacia dicha posición. El maestro envía el mensaje que incluye su posicón actual cada segundo.

Para crear un nuevo protocolo a partir de *FollowMe*, se ha seguido la documentación que se encuentra en el propio repositorio de ArduSim sobre el desarrollo de protocolos [13]. Los archivos donde se ha trabajado son *FollowMeTalkerThread.java* y *FollowMeListenerThread.java*. Como sus nombres indican, en el *Talker Thread* se gestiona la información que el maestro envía, mientras que en el *Listener Thread* se procesa la información recibida. Este trabajo se centra en la etapa de vuelo, por lo que las etapas de despegue y aterrizaje no se han visto modificadas.

2.1. *Talker Thread*

En el *Talker Thread* se debe obtener la información que se envía desde el maestro al resto de los drones del enajmbre. En el caso del *FollowMe* original, simplemente era necesario enviar la posición del dron maestro y su rumbo o *heading*. La posición se obtenía mediante la función *copter.getLocationUTM()*, que devuelve un objeto *Location2DUTM*. Si bien ArduSim también permite trabajar con coordenadas geográficas, se ha empleado el sistema UTM durante todo el trabajo.
Por otro lado, la altitud relativa del dron se obtiene empleando *getAltitudeRelative()*, de forma que con esta información se puede localizar el dron maestro en las 3 dimensiones del espacio.
Finalmente, la función *copter.getHeading()* devuelve el *heading* del maestro, necesario para obtener la orientación del enjambre. Si el maestro rotara, los esclavos deberían desplazarse para hacer rotar el enjambre.

Si *here* es el objeto que devuelve *getLocationUTM()* (*here.x* y *here.y* son las coordenadas x e y del dron maestro en UTM) y z es la altitud relativa, esta información se envía a los drones esclavos del enjambre como se muestra a continuación:

```
output.reset();
output.writeShort(Message.I_AM_HERE);
output.writeLong(selfId);
output.writeDouble(here.x);
output.writeDouble(here.y);
output.writeDouble(z);
output.writeDouble(heading);
output.flush();
message = Arrays.copyOf(outBuffer, output.position());
link.sendBroadcastMessage(message);
```

2.2. *Listener Thread*

Una vez el maestro ha enviado su localización, los drones esclavos la procesan para poder obtener la posición en la que deben estar y dirigirse hacia ella. Para recibir esta información se emplea *input.readDouble()* (*double* para las variables del tipo *double*, como es el caso de las coordenadas x, y, z y el rumbo).

Con las coordenadas x e y se puede crear un nuevo *Location2DUTM* en los drones esclavos que represente la posición del maestro, que en este caso se ha llamado *masterLocation*. Con la localización del maestro, su altitud y su rumbo se pueden determinar las posiciones de los esclavos según la formación del enjambre, tal y como se indica en las siguientes líneas de código:

```
targetLocation = takeOff.getFormationFlying().getLocation(
    takeOff.getFormationPosition(),masterLocation,masterHeading).getGeo();
copter.moveTo(new Location3DGeo(targetLocation, relAltitude));
```

targetLocation es un objeto del tipo *Location2DGeo*, es decir, determina la posicion del dron mediante su latitud y su longitud. Esta posición se trata de la localización hacia la que debe desplazarse el dron esclavo.
Se obtiene a partir de la posición del maestro, de su rumbo y de la formación del enjambre.

ArduSim permite simular el protocolo *FollowMe* empleando una de las formaciones que aparecen en la Tabla 2.1.

Tabla 2.1: Lista de formaciones disponibles en el protocolo *FollowMe*

Lista de formaciones del enjambre
Linear
Regular matrix
Compact matrix
Circle
Compact mesh
Random
Split-up

Finalmente, la función *copter.moveTo()* se encarga de desplazar al dron esclavo a su posición dentro de la formación. En este caso, *moveTo* emplea la posición objetivo, en coordenadas geográficas y sabiendo la altitud relativa. Sin embargo, como se verá más adelante, es posible emplear *moveTo* indicando al dron la velocidad a la que debe moverse en vez de la posición.

2.3. Simulación y resultados

Al comenzar una simulación en ArduSim, se abre el menú mostrado en la Figura 2.1. En este menú, en el apartado *Simulation Parameters*, se elige el número de drones que se van a simular, así como el protocolo. En este caso se van a simular dos drones siguiendo el protocolo *FollowMe*, de forma que se pueda estudiar fácilmente la distancia del dron esclavo respecto a su posición teórica dentro del enjambre.

En el resto de apartados se pueden modificar otros parámetros, como la batería de los drones, el viento u otros parámetros de rendimiento de la simulación, que se van a mantener con sus valores por defecto.

Una vez determinado el protocolo, en el siguiente menú (Figura 2.2) se pueden modificar ciertos parámetros del protocolo escogido (*FollowMe*). En el apartado *Ground master location* se puede escoger el lugar de inicio de la simulación, que en este caso es la Universitat Politècnica de València (Valencia, España). La formación empleada es *Linear*, de forma que los esclavos se situarán uno detrás de otro formando una línea. En este caso, como solo hay un esclavo, este se situará detrás del maestro.
La distancia mínima entre los drones en vuelo se ha mantenido en los 50 m que aparecen por defecto, por lo que si el esclavo volara de forma óptima, se mantendría a 50 m del maestro durante todo el vuelo. A continuación se comprobará que esto no sucede en el protocolo *FollowMe* original, de forma que se intentará solucionar en los nuevos protocolos propuestos en este trabajo.

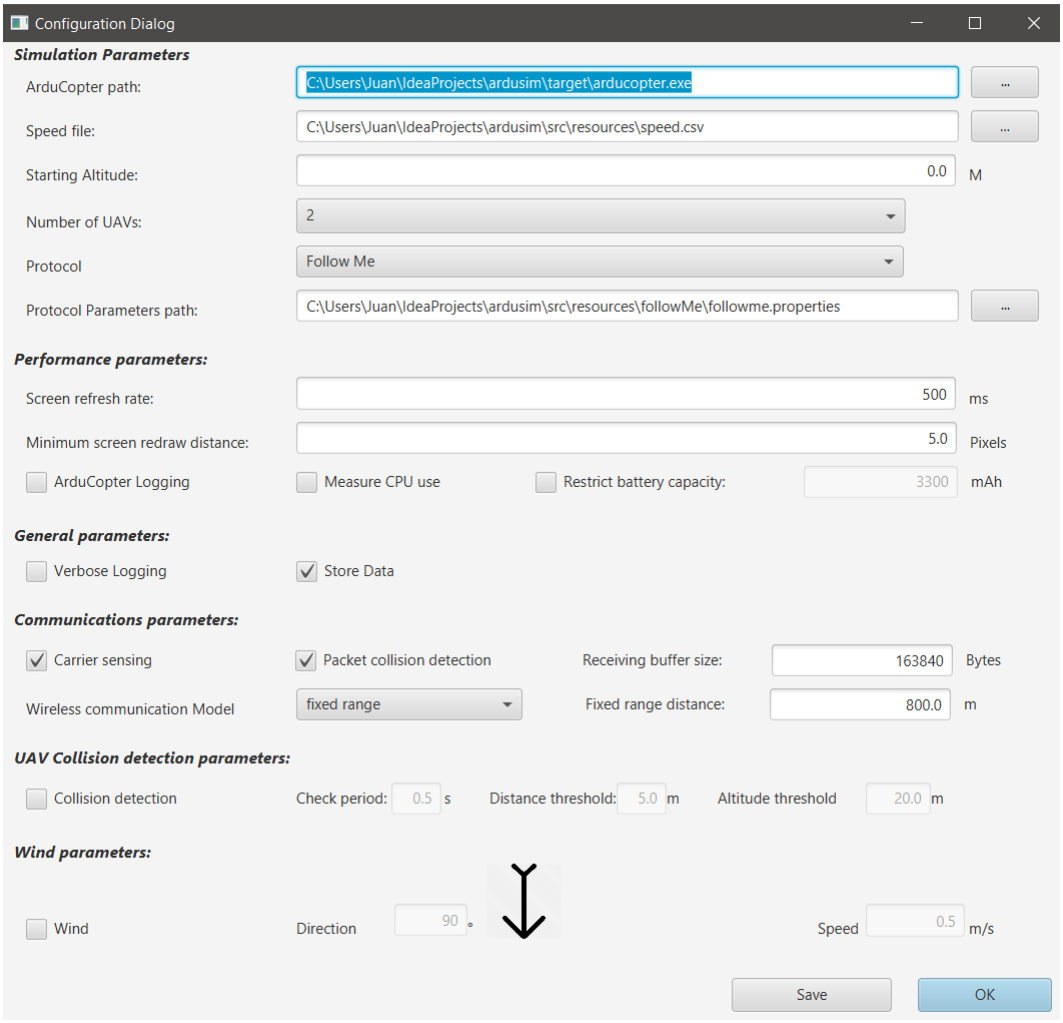

Figura 2.1: Primer menú de ArduSim.

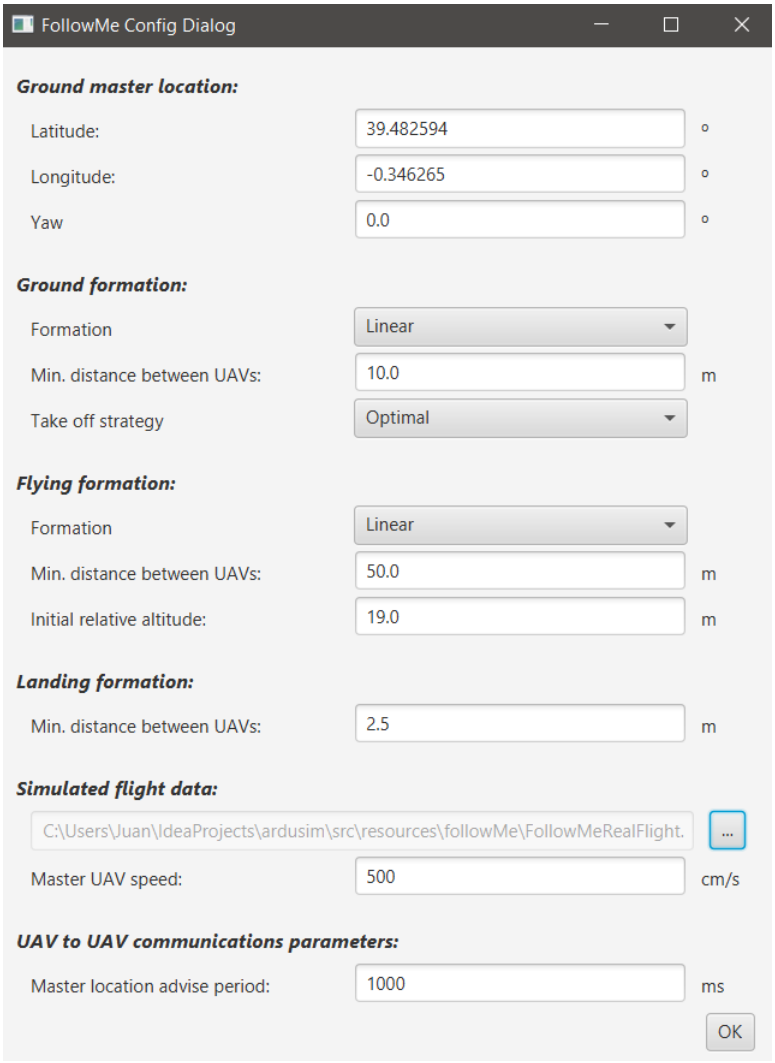

Figura 2.2: Segundo menú de ArduSim.

Respecto al apartado *Simulated flight data*, se emplean datos de un vuelo real. El archivo recoge las instrucciones que el mando del piloto enviaba al dron durante un vuelo real y las aplica en la simulación, de forma que el dron maestro simulado está controlado por un piloto de forma manual [12].

Tras superar este menú, se puede comenzar la simulación. ArduSim se estructura en tres partes, correspondientes a las Figuras 2.3, 2.4 y 2.5.

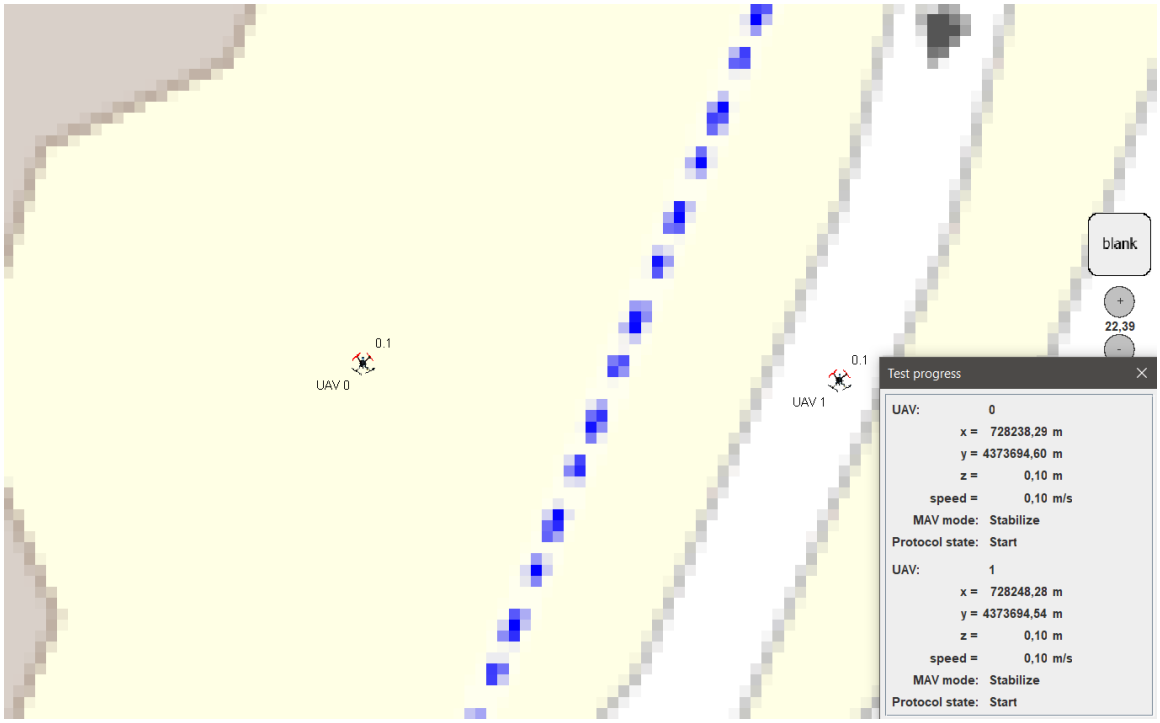

Figura 2.3: Disposición inicial en tierra de los drones.

En la Figura 2.3 se muestra el mapa donde se puede observar el vuelo de los drones del enjambre en tiempo real. En este caso, ambos drones se encuentran en tierra, puesto que todavía no se ha iniciado el experimento. ArduSim ofrece adapta el *zoom* del mapa para poder visualizar todos los drones automáticamente. También se puede apreciar una ventana con información relativa a la posición y a la velocidad de los dos drones que se van a simular.

```
Enabling Follow Me protocol...
UAV 1: Start
UAV 0: Start
Number of UAVs detected by master:1
Waiting for user interaction.
```

Figura 2.4: *Log* de ArduSim.

Por otro lado, en la Figura 2.4 se encuentra el *log* donde se muestra la información que se requiera durante la simulación del vuelo.

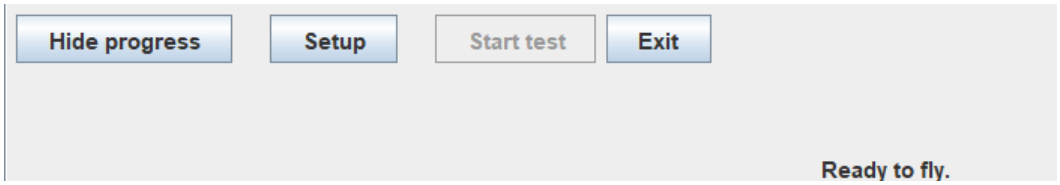

Figura 2.5: Botones de control de ArduSim.

Finalmente, ArduSim tiene un cuadro donde se encuentran los botones de control (Figura 2.5). Además, en este cuadro se muestra el estado actual de la simulación. En este caso, ambos drones están en tierra y listos para volar, de forma que el botón *Setup* aparece iluminado.

En la etapa de *Setup*, los drones esclavos vuelan para situarse en su lugar correspodiente dentro del enjambre, como se puede comprobar en la Figura 2.6. El UAV 1 es el esclavo, mientras que el UAV 0 es el maestro, que sigue en tierra esperando que el piloto lo haga despegar. El esclavo se ha situado detras del maestro a los 50 m indicados al iniciar el simulador. La línea azul representa el camino seguido por el esclavo, mientras que el camino que recorre el maestro se mostrará en negro.

Figura 2.6: Disposición de los drones depués de la etapa *Setup*.

Una vez el esclavo se ha situado en su posición incial, se puede comenzar el experimento (se ilumina el botón *Start test* de la Figura 2.5). El único cambio que se ha realizado

respecto al protocolo *FollowMe* original en este caso es la obtención de la distancia entre la posición real del esclavo y su posición ideal teórica (a 50 m), la obtención de la derrota real de los drones, así como la creación de un archivo *.csv* con cierta información relevante para estudiar y comparar los diferentes protocolos que se desarrollarán a continuación. El archivo se crea en las siguientes líneas de código:

```
// RESULTS OUTPUT
String resultsOutput = String.valueOf(timestamp) + ";" + String.valueOf(distance) +
        ";" + String.valueOf(error)  +";"+
        String.valueOf(mastDegCourse) +";" + String.valueOf(masterHeading)+
        ";"+ String.valueOf(DegCourse) +";"+ String.valueOf(copter.getHeading()) +";"+
        String.valueOf(masterLat) + ";" + String.valueOf(masterLon) + ";" +
        String.valueOf(Latitude) + ";" + String.valueOf(Longitude) + "\n";
try {
    File f = new File("C:\\Users\\Juan\\Documents\\TFG\\Results\\pruebas.csv");
    boolean writeHeader = !f.exists();
    FileWriter fr = new FileWriter(f, true);
    if (writeHeader) {
        fr.write("Tiempo;Distancia;Error;Master_course;Master_heading;Esclavo_course;" +
                "Esclavo_heading;Master_lat;Master_lon;Esclavo_lat;Esclavo_lon\n");
    }
    fr.write(resultsOutput);
    fr.close();
}   catch (IOException e) {
    e.printStackTrace();
}
```

En la Tabla 2.2 se muestran los datos que recoge el archivo *.csv* junto con una breve descipción.

Tabla 2.2: Datos recogidos en el archivo *.csv*

Dato	**Descripción**
Tiempo	*Timestamp* del dato guardado (*System.currentTimeMillis()*)
Distancia	Distancia entre el dron esclavo y el maestro en metros
Error	Distancia entre el dron esclavo y su posición deseada
Master_track	Derrota del dron maestro en grados
Master_heading	Rumbo magnético del dron maestro en radianes
Esclavo_track	Derrota del dron esclavo en grados
Esclavo_heading	Rumbo magnético del dron maestro en radianes
Master_lat	Latitud del dron maestro
Master_lon	Longitud del dron maestro
Esclavo_lat	Latitud del dron esclavo
Esclavo_lon	Longitud del dron esclavo

Por otro lado, el cálculo del error, es decir, de la distancia entre el esclavo y su posición deseada, se obtiene como se indica a continuación:

```
targetLocationUTM = takeOff.getFormationFlying().getLocation(
    takeOff.getFormationPosition(),masterLocation, masterHeading);
double error = targetLocationUTM.distance(copter.getLocationUTM());
```

Este error es el que se estudiará, de forma que se pueda reducir lo máximo posible con las nuevas versiones desarrolladas en este trabajo.

Estos cambios realizados sobre el protocolo original no afectan al comportamiento de los drones, por lo que al simular este protocolo se obtiene el resultado que se obtenía anteriormente y que se muestra en la Figura 2.7. Se trata de una captura de Google Earth, puesto que además del *.csv*, ArduSim también permite guardar las simulaciones para visualizarlas en Earth o AutoCAD.

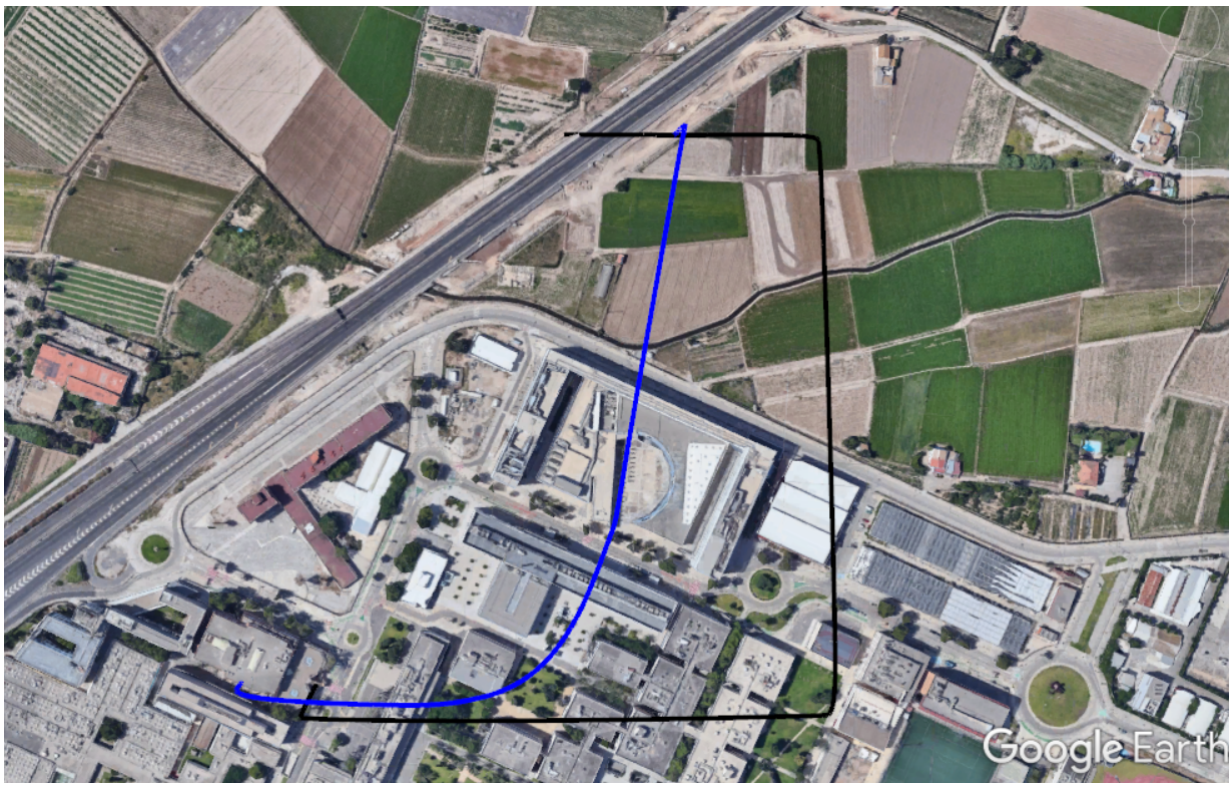

Figura 2.7: Resultado de la simulación con el protocolo *FollowMe* original.

Por otro lado, en la Figura 2.8 se observa una gráfica realizada en Matlab. Se ha creado un *script* en Matlab para realizar gráficas a partir de los datos recogidos en el *.csv*, de forma que se puedan obtener y comparar resultados. Este *script* se puede consultar en los Anexos, así como el resto del código desarrollado. Estas gráficas del error permiten estudiar de forma más fácil si los nuevos protocolos mejoran al *FollowMe* puesto que únicamente con el trazo de la trayectoria seguida no se puede apreciar el retardo con el que el dron esclavo alcanza las posiciones deseadas. En este caso, el error es bastante elevado, alcanzando los 300 metros.

En la Figura 2.8 también se pueden apreciar los cambios en la dirección del dron, que se producen en los segundos 43 y 80 y que provocan una disminución temporal del error.

Cabe destacar que estos giros los realiza el dron sin rotar, es decir, el dron siempre apunta hacia la misma dirección. Esto se traduce en que su rumbo o *heading* es prácticamente constante y en que la trayectoria seguida por el esclavo de forma ideal sería el representado en rojo en la Figura 2.9.
El dron esclavo debería estar a 50 m, y en el tramo vertical mantenerse a 50 m a la izquierda del maestro, tal y como se muestra en la Figura 2.9.

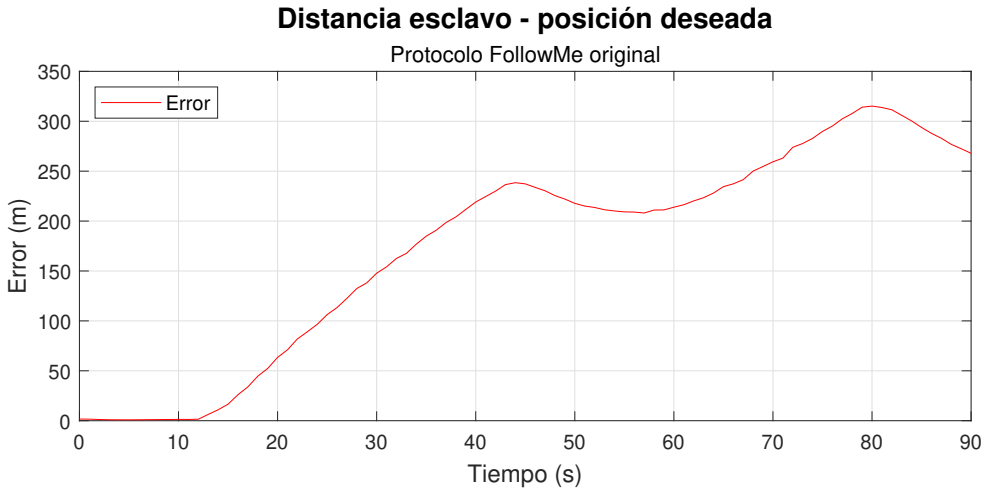

Figura 2.8: Distancia esclavo - posición deseada con el protocolo *FollowMe*.

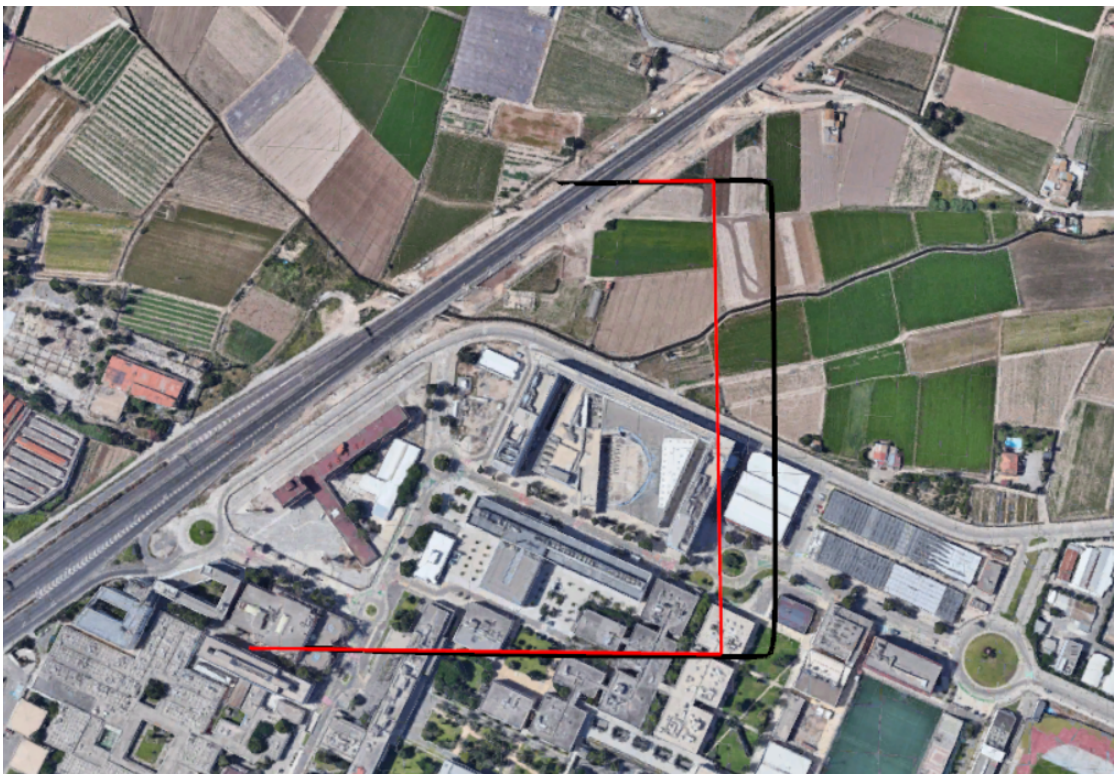

Figura 2.9: Trayectoria ideal del dron esclavo (rojo).

En conclusión, el protocolo *FollowMe* no consigue que los drones esclavos del enjambre mantengan la distancia indicada, como se ha comentado anteriormente. Por ello, se han desarrollado tres mejoras del protocolo que se expondrán a continuación.

Capítulo 3

Mejoras de protocolo *FollowMe*

3.1. Primera versión: predicción de la posición del dron maestro

En la primera propuesta de mejora del protocolo *FollowMe* se intenta solucionar una de las limitaciones principales que tienen los esclavos a la hora de seguir al maestro. Al ser el maestro un dron pilotado por un humano, los drones esclavos del enjambre no conocen las intenciones del piloto de antemano.

En esta primera versión, los drones esclavos estiman la posición futura del maestro, con la intención de poder anticiparse de algún modo al maestro. Esta idea parte de la propuesta realizada en el artículo *Automatic system supporting multicopter swarms with manual guidance* como futuro trabajo de cara a mejorar *FollowMe* [12].

La única información disponible por el momento es la velocidad del maestro, puesto que ArduSim no dispone por el momento de información sobre la aceleración de los drones. Esto supone una limitación importante, ya que la predicción de la posición empleará únicamente la velocidad.

La predicción de la posición se ha calculado en el *Listener Thread*, de forma que el maestro, en el *Talker Thread*, debe enviar su velocidad y la dirección en la que se mueve. Aquí se encuentra el primer problema. Como ya se ha comentado, el maestro recorre los diferentes tramos de su recorrido con un rumbo constante, por lo que el *heading* no se puede emplear en este caso.

ArduSim no dispone de una función que devuelva la derrota o *track* de los drones del enjambre, así que se ha implementado. En las siguientes líneas de código, se obtiene el *track* del maestro como el ángulo que forma la posición actual del dron y la posición anterior

respecto al eje y del sistema de coordenadas UTM:

```
// Track
prevLocation = here;
here = copter.getLocationUTM();
if (here.x != prevLocation.x && here.y != prevLocation.y) {
    track = Math.PI / 2 - Math.atan2(here.y - prevLocation.y, here.x - prevLocation.x);
    if (track < 0){
        track = track + 2*Math.PI;
    }
    else{
        if (track > 2*Math.PI){
            track = track - 2*Math.PI;
        }
    }
}
```

Este *track* se emplea para obtener la posición del maestro en el futuro, asumiendo que el piloto mantiene la velocidad y la derrota constantes. Para ello se obtienen las coordenadas x e y en el sistema UTM mediante el módulo de la velocidad y el coseno o el seno de la derrota, según corresponda. Los esclavos obtienen la predicción como se muestra a continuación:

```
// Prediction
Location2DUTM predictedPos;
double predictedT = 1;  // seconds
double predictedX = masterLocation.x + masterSpeed * Math.sin(masterTrack) * predictedT;
double predictedY = masterLocation.y + masterSpeed * Math.cos(masterTrack) * predictedT;
predictedPos = new Location2DUTM(predictedX, predictedY);
```

Se puede apreciar que aparece el parámetro *PredictedT*, que se trata del tiempo de la predicción. Se va a variar este parámetro para ver como afecta al comportamiento del enjambre.

Por otro lado, esta predicción de la posición se aplica a los drones esclavos empleando *copter.moveTo*, de igual forma que se hacía cuando el esclavo se movía a la posición actual del maestro.

Los resultados de la simulación con este protocolo se muestran en la Figura 3.1. Se puede apreciar como el error se ha reducido ligeramente, además de que un mayor tiempo de predicción también reduce el error. Sin embargo, este continúa siendo muy elevado.

Esto se debe a que, aunque el dron se dirige a una posición futura con cierta anticipación, no lo hace a mayor velocidad. Es decir, el esclavo no adapta su velocidad en función de lo lejos que se encuentre de la posición deseada.

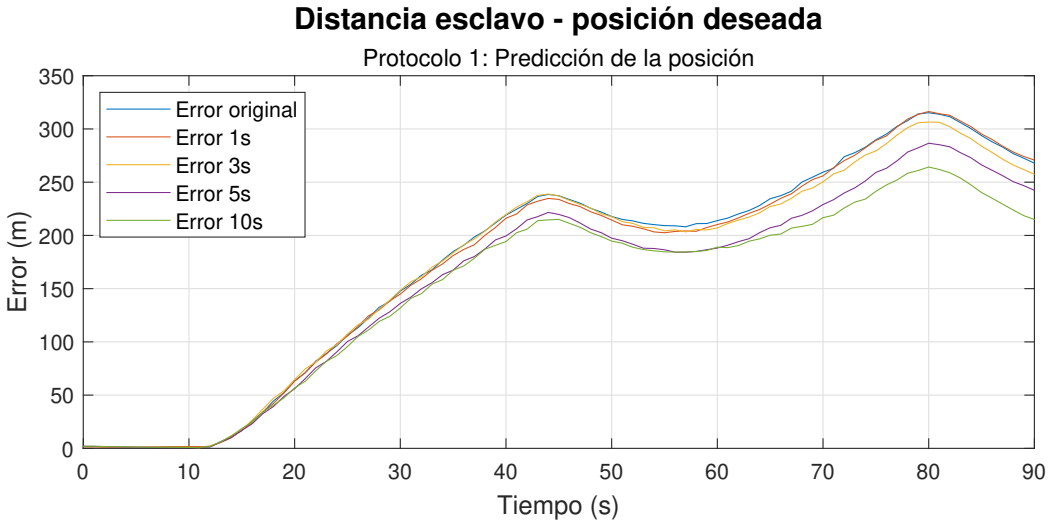

Figura 3.1: Comparativa del error según el tiempo de predicción. Protocolo 1.

Si se atiende al trazado recorrido por el esclavo en el caso del tiempo de predicción de 1 segundo (Figura 3.2) y en el caso de 10 segundos (Figura 3.3) se puede comprobar que no existe una diferencia significativa entre ambos casos ni entre el caso original de la Figura 2.7. De forma que se puede concluir que la primera versión no soluciona el problema del elevado retardo del dron esclavo.

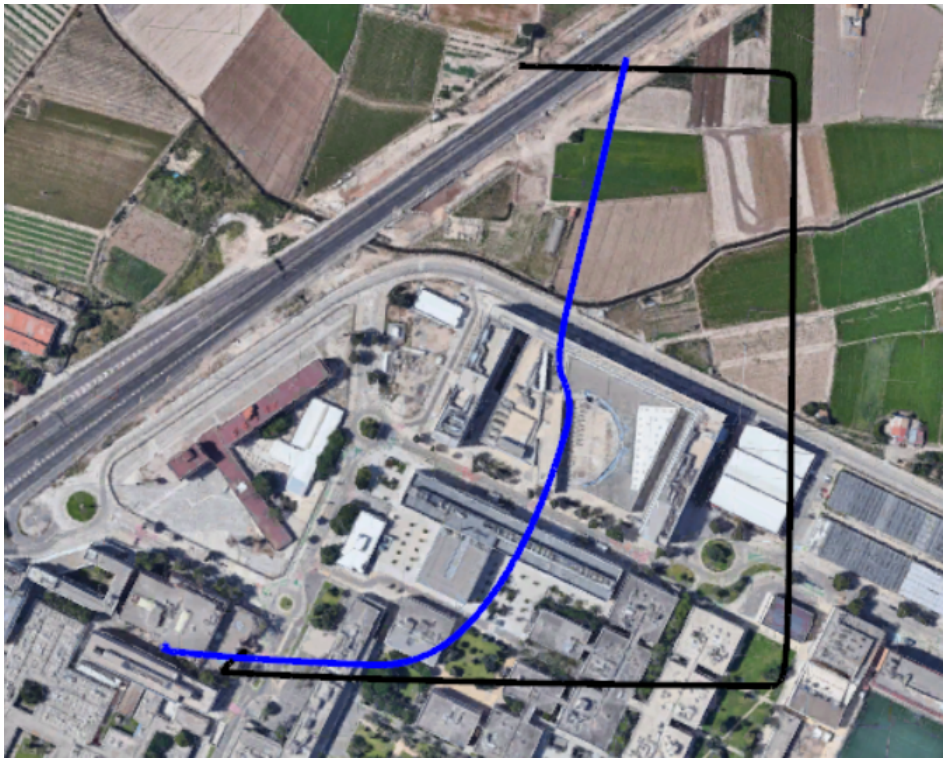

Figura 3.3: Trayectoria del esclavo con la primera versión y con 10 segundos de predicción.

Figura 3.2: Trayectoria del esclavo con la primera versión y con 1 segundo de predicción.

3.2. Segunda versión: estimación de la velocidad de los esclavos según la predicción de la posición del maestro

La función *copter.moveTo*, que se encarga de desplazar al dron de un lugar a otro, puede emplear velocidades en lugar de posiciones. La segunda versión desarrollada emplea esto para intentar solucionar el problema del retardo del esclavo, puesto que se le puede indicar una mayor velocidad al esclavo si el maestro se aleja, solucionando así la limitación de la primera versión propuesta.

En la descripción de *moveTo* aparece lo siguiente:

```
    /**
     * Move the UAV in an specific direction, given a speed vector.
     * <p>Method designed to update continuously the target location of the UAV.</p>
     * <p>The UAV must be in GUIDED flight mode.</p>
     * <p>This method uses the message SET_POSITION_TARGET_GLOBAL_INT, and doesn't wait
     * response from the flight controller.
     * Values are not applied immediately, but each time a message is received from the
     * flight controller.</p>
     * @param vx (m/s) target speed pointing north.
     * @param vy (m/s) target speed pointing east.
     * @param vz (m/s) target speed pointing down.
     */
    public void moveTo(double vx, double vy, double vz) {
        UAVParam.targetSpeed[numUAV].set(new float[]{(float)vx, (float)vy, (float)vz});
    }
```

Esto quiere decir que ArduSim permite controlar el movimiento de los drones esclavos si se le proporciona la velocidad siguiendo el sistema *North* - *East* - *Down* (NED), como el que se muestra en la Figura 3.4.

El código que se ha implementado en el *Listener Thread* para obtener las velocidades es el siguiente:

```
moveToLocation = takeOff.getFormationFlying().getLocation(takeOff.getFormationPosition(),
    predictedPos, masterHeading).getGeo();
moveToLocationUTM = moveToLocation.getUTM();

double vNorth = (moveToLocationUTM.y - copter.getLocationUTM().y)/predictedT;
double vEast = (moveToLocationUTM.x - copter.getLocationUTM().x)/predictedT;
double vDown = -(relAltitude - copter.getAltitudeRelative());

if (vDown < 0 && copter.getAltitudeRelative() < 10) {
    vDown = 0;
}
```

Las velocidades se obtienen de forma que el dron alcance la posicion predicha en el tiempo de predicción, es decir, se estima la posición del maestro dentro de los segundos determinados por *preditedT*. Posteriormente, según la formación del enjambre, se obtiene la posición en la que debería estar el esclavo si la predicción fuera correcta. Finalmente, se estiman las velocidades que debería tener el esclavo para alcanzar dicha posición en el tiempo especificado por *predictedT*.

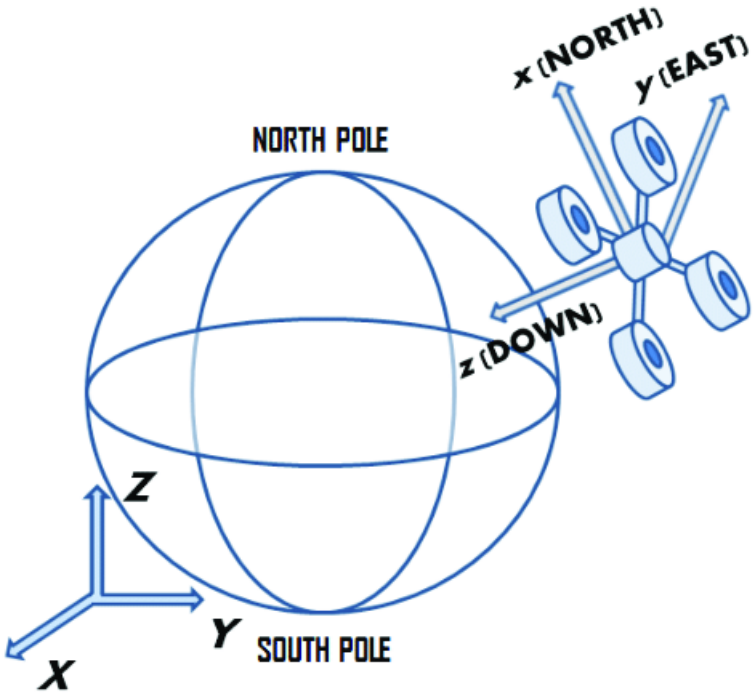

Figura 3.4: Sistema *North-East-Down* [14].

Esto se realiza en el plano Norte-Este, mientras que en el eje z, la velocidad vertical se ha obtenido de forma que el esclavo alcance la altitud del maestro en un segundo, ya que no se realiza una predicción de la altitud. También se ha comprobado que dicha velocidad no provoque una colisión del esclavo contra el suelo.

De igual forma que con el caso anterior, se ha simulado el comportamiento del enjambre siguiendo este protocolo para diferentes *PredictedT*, obteniendo los resultados mostrados en la Figura 3.5.

En la Figura 3.5 sí se aprecia una reducción significativa del error, mucho mayor que con la primera versión del protocolo. El error se mantiene reducido hasta el segundo giro del dron maestro, donde el error vuelve a crecer hasta niveles parecidos a los del protocolo original.

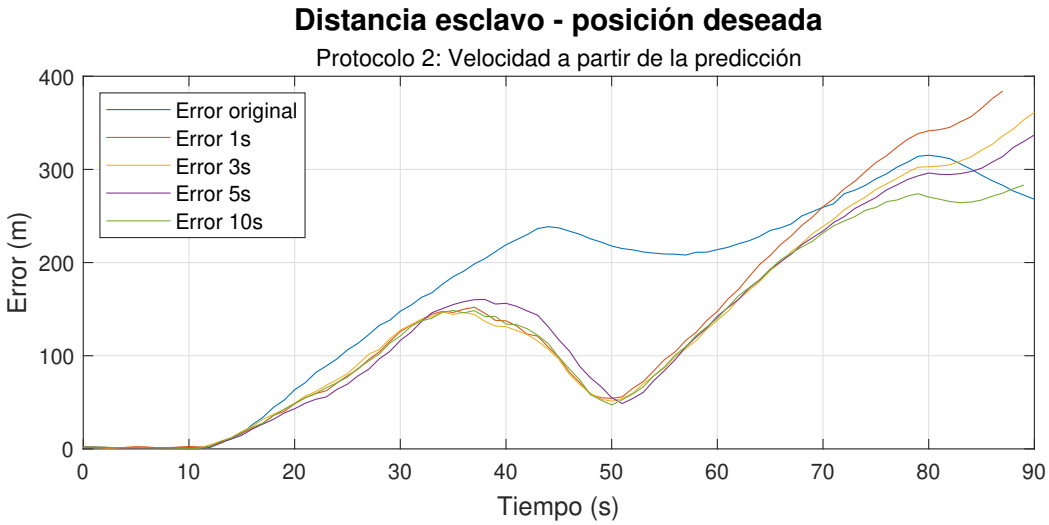

Figura 3.5: Comparativa del error según el tiempo de predicción. Protocolo 2.

Por otro lado, en las Figuras 3.6 y 3.7, se observa el camino recorrido por el esclavo para un *PredictedT* de 1 segundo y 10 segundos respectivamente. El uso de la velocidad en vez de la posición permite que el dron esclavo corrija el error. Sin embargo, la falta de un sistema que controle esta velocidad provoca que el esclavo se pase de largo en los giros. Además, si se atiende a la velocidad vertical, como se muestra en la Figura 3.8, se puede aperciar que el dron oscila, puesto que al alcanzar la posición deseada se pasa de largo.

Con este fallo se puede concluir que la segunda versión sigue sin solucionar los problemas del protocolo *FollowMe*, si bien el uso de velocidad en lugar de posición para indicar cómo debe moverse el esclavo sí permite reducir el error, a diferencia de la primera versión expuesta en este trabajo.

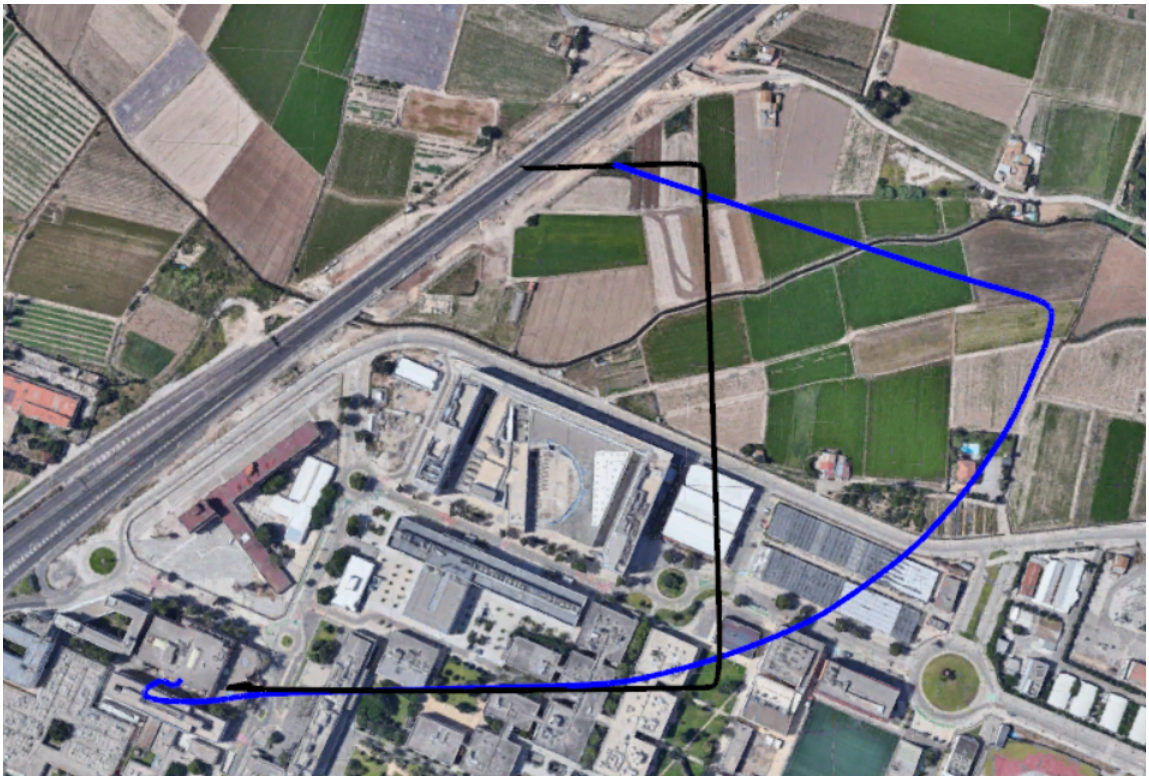

Figura 3.6: Trayectoria del esclavo con la segunda versión y con 10 segundos de predicción.

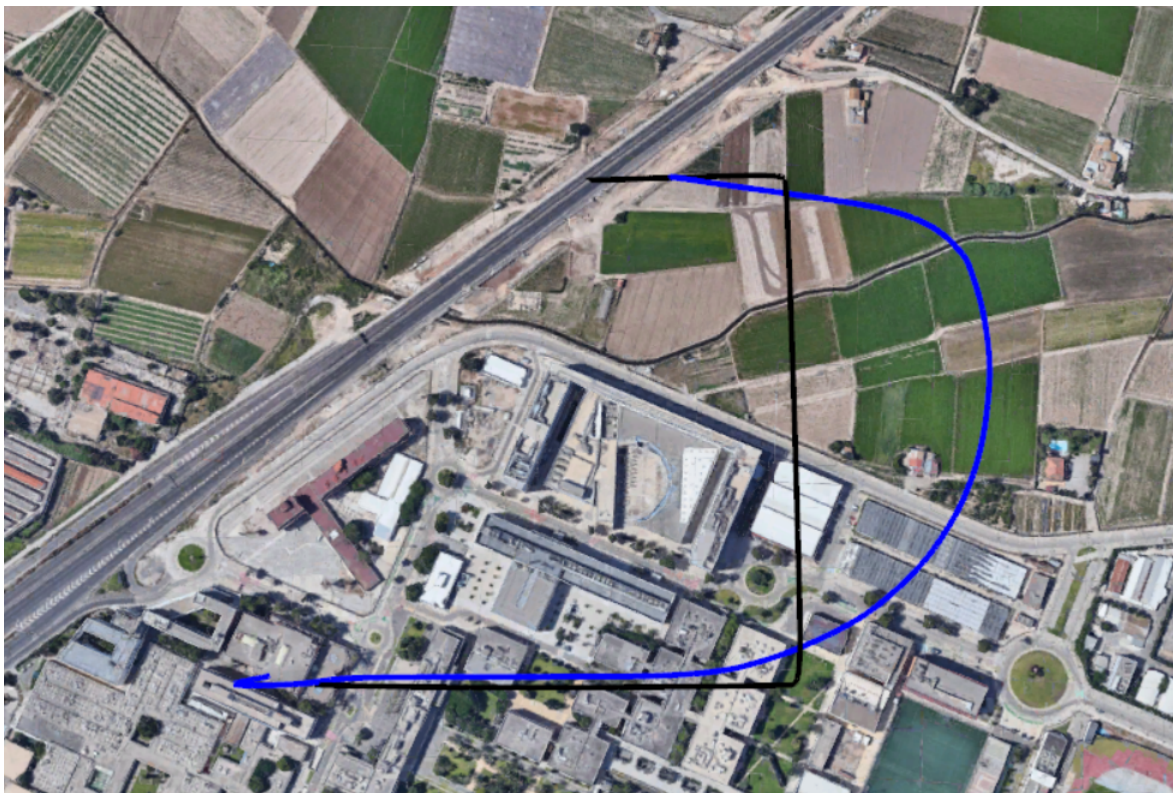

Figura 3.7: Trayectoria del esclavo con la segunda versión y con 10 segundos de predicción.

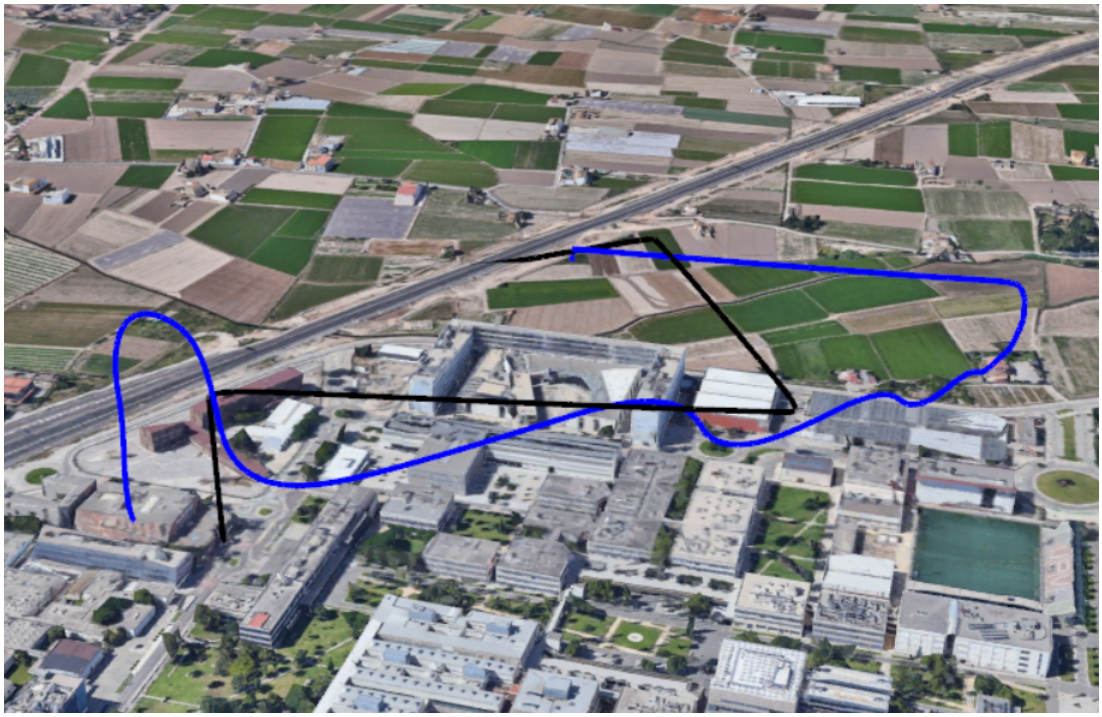

Figura 3.8: Oscilaciones del esclavo en la segunda versión del protocolo.

3.3. Tercera versión: controladores PID para la obtención de la velocidad de los esclavos

La tercera versión propuesta en este trabajo emplea tres controladores PID (Proporcional, Integral y Derivativo), de forma que la velocidad se pueda ajustar y evitar las oscilaciones. Se ha implementado un controlador PID para cada componente de la velocidad en el sistema NED, tal y como se muestra a continuación:

```
//vNorth
double errorNorth = targetLocationUTM.y - copter.getLocationUTM().y;
cumErrorNorth = cumErrorNorth + (errorNorth * elapsedTime);
double rateErrorNorth = (errorNorth - lastErrorNorth)/elapsedTime;
double vNorth = kpNorth * errorNorth + kdNorth * rateErrorNorth + kiNorth * cumErrorNorth;
```

En este caso se muestra el PID implementado para obtener la velocidad en dirección norte. Los otros dos controladores se han implementado de la misma forma, por lo que se va a explicar únicamente uno de ellos.

El controlador PID trata de ajustar la velocidad para acercar al dron a la posición de referencia, que en este caso es la posición teórica en la que debería estar el esclavo. Se pueden observar tres errores. En primer lugar el error actual *errorNorth*, que se trata del error en la posición en el eje y del sistema UTM. Este error se corrige con el valor proporcional

del controlador. En segundo lugar, el error acumulado en el tiempo *cumErrorNorth* es el que tiene en cuenta la parte integral del controlador. Por último, la tasa de cambio del error *rateErrorNorth* refleja cómo esta evolucionando el error y es el error tenido en cuenta por la parte derivativa del controlador.

Todo esto se repite tres veces para obtener las tres velocidades requeridas por *copter.moveTo*. Además, se deben tener en cuenta tres constantes que ponderan el peso de cada respuesta del controlador. Estas constantes se han seleccionado con la intención de reducir el error y mejorar el comportamiento de los drones del enjambre comparando sucesivas simulaciones.

Para ello, en primer lugar se realizan simulaciones manteniendo las constantes integral (ki) y derivativa (kd) a cero. Se ajusta la constante proporcional (kp) de forma que el esclavo siga al maestro. Posteriormente, se reduce la constante proporcional a la mitad y se ajusta la integral. En este caso, la constante integral solo generaba oscilaciones y no mejoraba el comportamiento, por lo que se han mantenido a cero para las tres componentes de la velocidad. Finalmente, se ajusta la constante derivativa hasta obtener una respuesta aceptable [15]. Cabe destacar que el ajuste se ha realizado empleando el único vuelo que se tenía disponible, por lo que es posible que se tenga que realizar un nuevo ajuste de las constantes del controlador para mejorar su comportamiento ante otras condiciones.

Este proceso se ha repetido tres veces, una para cada componente de la velocidad. Además, en cada simulación, las otras dos componentes que no se encontraban bajo estudio se han mantenido a cero con la intención de observar con mayor facilidad la respuesta del esclavo y la evolución de los tres errores expuestos anteriormente. Los valores que han ofrecido mejores resultados se muestran en la Tabla 3.1.

Tabla 3.1: Constantes de los controladores PID de la tercera versión.

	North	**East**	**Down**
Proporcional (kp)	0.2	0.2	0.15
Integral (ki)	0	0	0
Derivativo (kd)	0.29	0.29	0.15

Con estas constantes, se ha simulado el enjambre empleando como referencia la posición ideal actual del dron esclavo, así como una predicción de esta posición, obteniendo los resultados mostrados en la Figura 3.9. Se puede observar que el protocolo nuevo reduce en gran medida el error cuando se emplea como referencia la posición ideal del esclavo. Si se emplean predicciones de la posición del maestro como referencia, el error comienza a aumentar, como era de esperar, pues el controlador PID trata de ajustarse a una referencia que no es la real, aumentando el error.

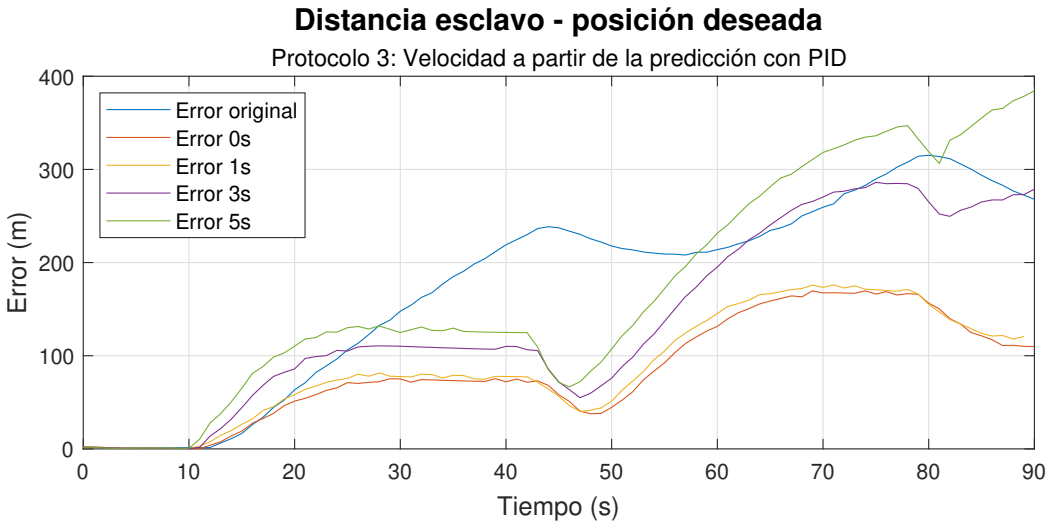

Figura 3.9: Comparativa del error según el tiempo de predicción. Protocolo 3.

Por otro lado, en la Figura 3.10 se puede observar la trayectoria del esclavo empleando la tercera versión del protocolo sin ninguna predicción, el cual ha demostrado ser el que reduce en mayor medida el error. La trayectoria se asemeja en mucha mayor medida a la del dron maestro, adapatándose mejor a las curvas y generando menos oscilaciones.

Figura 3.10: Trayectoria del esclavo con la tercera versión.

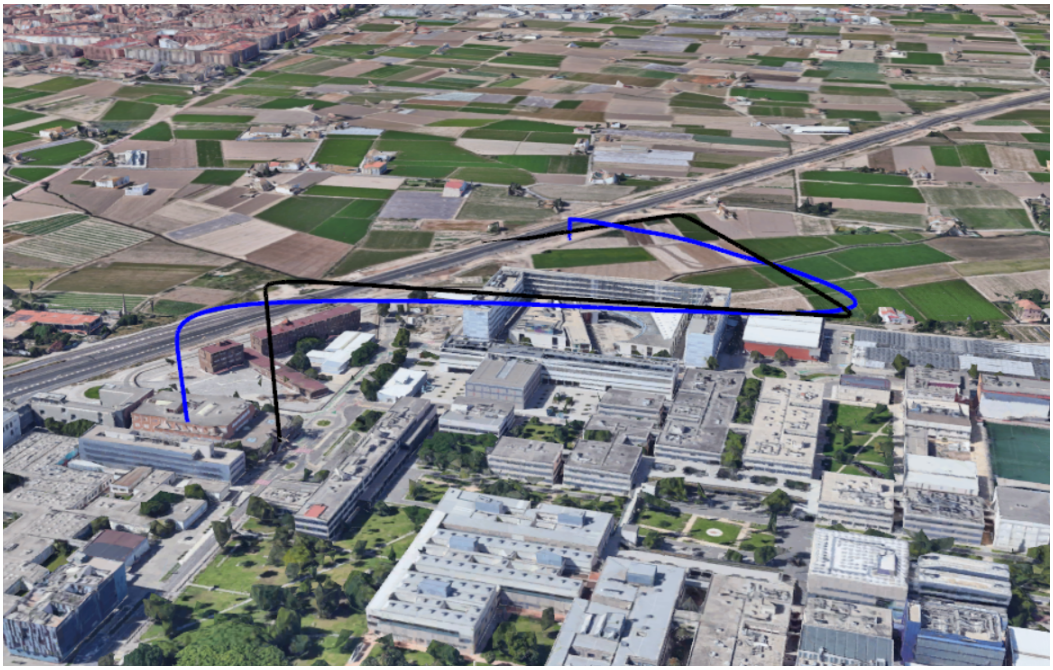

Figura 3.11: Variación de la altitud con la tercera versión.

Finalmente, en la Figura 3.11 se aprecia como las oscilaciones en la altitud han desaparecido y como el esclavo ajusta mucho mejor su altitud a la del dron maestro.

En definitiva, se puede concluir que la tercera versión supone una mejora considerable respecto a la versión original del *FollowMe* y respecto a las anteriores versiones de la propuesta de mejora.

Capítulo 4

Análisis de resultados y conclusiones

4.1. Análisis de resultados

Como ya se ha comentado a lo largo de este trabajo, la mejora del protocolo que ha ofrecido mejores resultados se trata de la tercera versión, empleando tres controladores PID, sin emplear ningún tipo de predicción.

En la Figura 4.1 se aprecia como el error se ha reducido durante todas las etapas del vuelo, incluidos los giros. Por otro lado, en la Figura 4.2, se puede comparar la trayectoria que seguía el esclavo empleando la versión original del protocolo con la trayectoria que sigue con el nuevo protocolo desarrollado. Se observa que el esclavo se ajusta más a la trayectoria del maestro, como consecuencia de haber reducido el error, mientras que en el caso original, el esclavo actuaba con mucho retraso ante los movimientos del maestro.

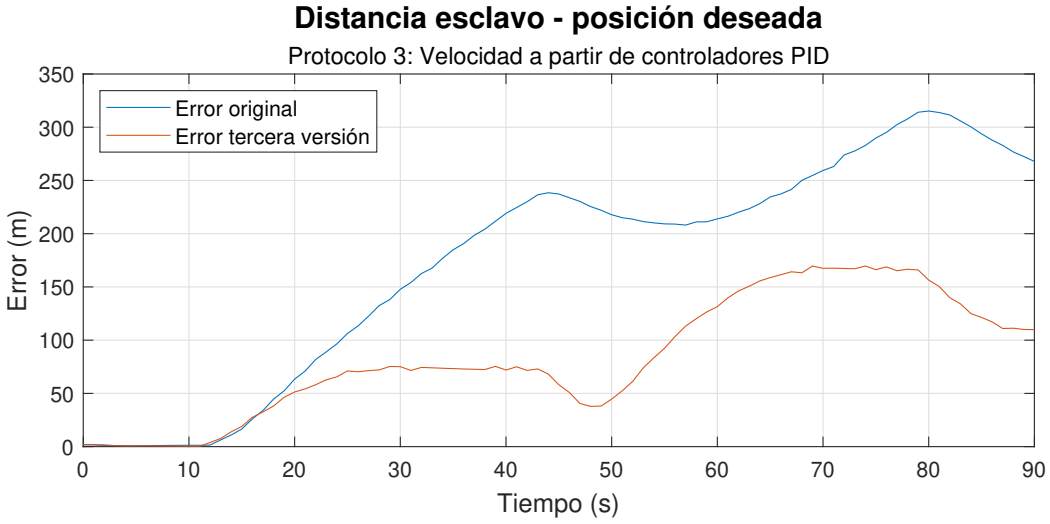

Figura 4.1: Comparativa del error entre el protocolo original y la nueva propuesta.

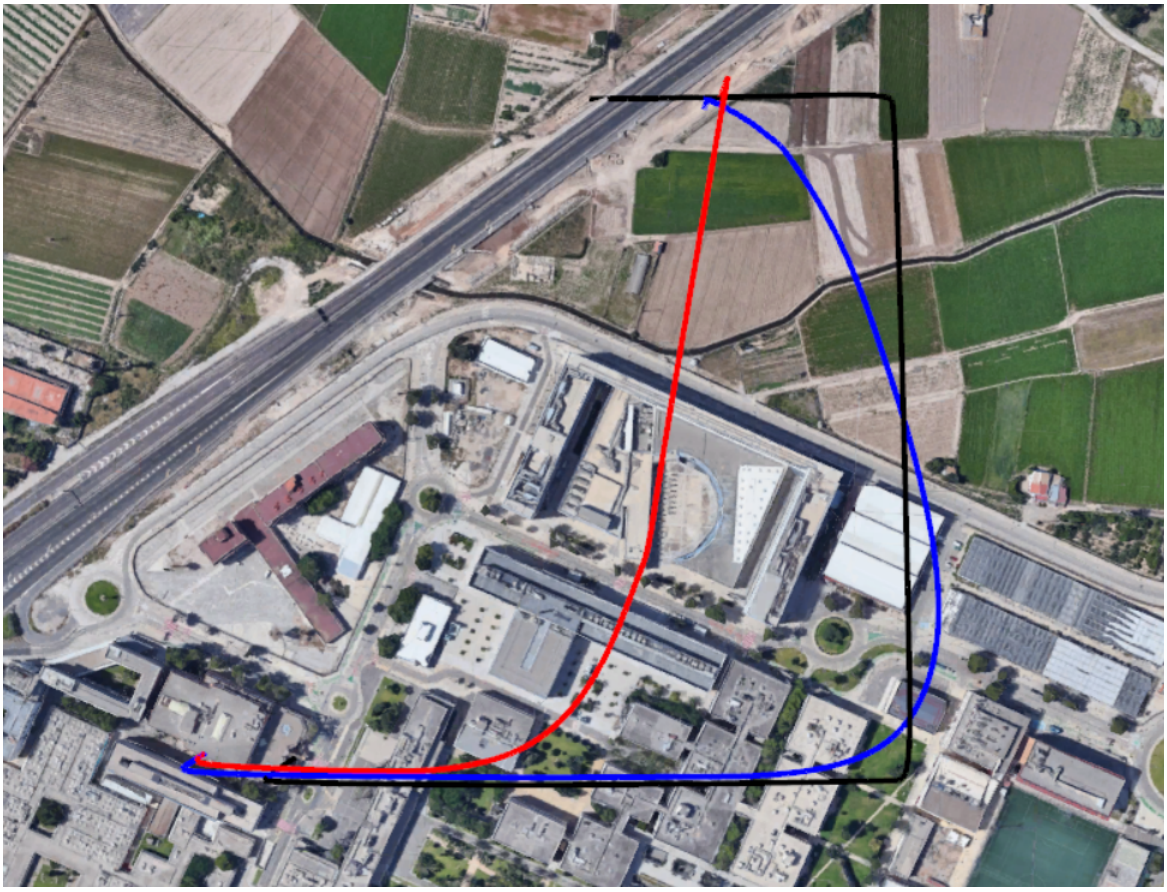

Figura 4.2: Comparativa entre las trayectorias del esclavo.

4.2. Conclusiones

El estudio de ciertas formas de predecir la posición del maestro puede ser de utilidad, sin embargo, existen ciertas limitaciones. La primera se trata de la propia naturaleza del dron maestro. Se trata de un dron pilotado por un humano, por lo que, si sus intenciones son desconocidas, es difícil implementar algún algoritmo de predicción que sirva en todos los casos y para todos los pilotos.

También se debe ser consciente de que este estudio se ha realizado con la única simulación del dron maestro que se disponía, por lo que no se ha podido demostrar que los nuevos protocolos ofrezcan resultados robustos en todas las situaciones.

Por otro lado, ArduSim no devuelve por el momento la aceleración del dron, por lo que las predicciones se basan en la velocidad actual del maestro. La falta de información de la aceleración es un limitante importante de la calidad de la predicción que puede ser corregido en el futuro, de forma que se pueda volver a intentar el desarrollo de un nuevo protocolo que emplee algoritmos de predicción de la posición con resultados favorables. Por

estas razones es por las que el uso de controladores PID ha ofrecido mejores resultados que los intentos de implementar algoritmos de predicción de la posición sencillos.

En conclusión, si bien la predicción de la posición mediante técnicas más avanzadas puede generar resultados favorables, en este caso no se ha alcanzado el objetivo de reducir el error de forma significativa empleando estas técnicas.

Por otro lado, el uso de controladores PID se trata de una solución efectiva de los problemas que presentaba el protocolo *FollowMe* original, tal y como se ha demostrado, por lo que puede ser un buen nuevo protocolo dentro de las opciones que ofrece ArduSim. El código de este protocolo puede encontrarse en el Anexo.

Bibliografía

[1] Wikipedia, la enciclopedia libre. *Vehículo aéreo no tripulado*. Recuperado de `https://es.wikipedia.org/wiki/Vehiculo_aereo_no_tripulado`. Accedido el 2 de septiembre de 2021.

[2] Wikipedia, the free encyclopedia. *Delivery drone*. Recuperado de `https://en.wikipedia.org/wiki/Delivery_drone`. Accedido el 2 de septiembre de 2021.

[3] Wikipedia, the free encyclopedia. *Amazon Prime Air*. Recuperado de `https://en.wikipedia.org/wiki/Amazon_Prime_Air`. Accedido el 2 de septiembre de 2021.

[4] Wikipedia, the free encyclopedia. *Swarm robotics - Drone swarming*. Recuperado de `https://en.wikipedia.org/wiki/Swarm_robotics#Drone_Swarming`. Accedido el 2 de septiembre de 2021.

[5] SofOS. Organización de conocimiento. *Enjambres de drones en la agricultura, usos y ventajas*. Recuperado de `http://www.sofoscorp.com/enjambres-drones-la-agricultura/`. Accedido el 2 de septiembre de 2021.

[6] BBC News. *Cómo los enjambres de drones cambiarán la estrategia de las guerras del futuro*. Recuperado de `https://www.bbc.com/mundo/noticias-47595525`. Accedido el 2 de septiembre de 2021.

[7] Rodríguez, Ana. iberfdrone. *Nuevas categorías de operaciones con drones*. Recuperado de `https://iberfdrone.es/nuevas-categorias-de-operaciones-con-drones/`. Accedido el 6 de septiembre de 2021.

[8] JARUS. *JARUS guidelines on Specific Operations Risk Assessment (SORA)*. Recuperado de `http://jarus-rpas.org/sites/jarus-rpas.org/files/jar_doc_06_jarus_sora_v2.0.pdf`. Accedido el 6 de septiembre de 2021.

[9] Fabra Collado, Francisco. *ArduSim. Readme*. Recuperado de `https://bitbucket.org/frafabco/ardusim/src/master/README.md`. Accedido el 3 de septiembre de 2021.

[10] MAVLink. *MAVLink Developer Guide*. Recuperado de `https://mavlink.io/en/`. Accedido el 3 de septiembre de 2021.

[11] Francisco Fabra, Carlos T. Calafate, Juan Carlos Cano, Pietro Manzoni, *ArduSim: Accurate and real-time multicopter simulation, Simulation Modelling Practice and Theory*,

Volume 87, 2018, Pages 170-190, ISSN 1569-190X, https://doi.org/10.1016/j.simpat.2018.06.009.

[12] Francisco Fabra, Willian Zamora, Joan Masanet, Carlos T. Calafate, Juan-Carlos Cano, Pietro Manzoni, *Automatic system supporting multicopter swarms with manual guidance, Computers Electrical Engineering*, Volume 74, 2019, Pages 413-428, ISSN 0045-7906, https://doi.org/10.1016/j.compeleceng.2019.01.026.

[13] Fabra Collado, Francisco. *ArduSim. Protocol development.* Recuperado de https://bitbucket.org/frafabco/ardusim/src/38bab8207e91d6b6ff9f92cd4d3547499d9eefe5/help/development.md. Accedido el 3 de septiembre de 2021.

[14] Giernacki, Wojciech Goslinski, Jaroslaw & Espinoza-Fraire, Tadeo. (2021). Mathematical Modeling of the Coaxial Quadrotor Dynamics for Its Attitude and Altitude Control. Energies. 14. 10.3390/en14051232.

[15] Wikipedia, la enciclopedia libre. *Controlador PID - Ajuste manual.* Recuperado de https://es.wikipedia.org/wiki/Controlador_PID#Ajuste_manual. Accedido el 3 de septiembre de 2021.

Parte II

PLIEGO DE CONDICIONES

Capítulo 1

Pliego de condiciones

1.1. Introducción

En este documento se van a exponer las condiciones de uso del simulador ArduSim, sobre el que se ha centrado el desarrollo de este trabajo. Se van a tratar tanto los requerimientos técnicos como las limitaciones de la licencia de uso. Finalmente, se va a explicar cómo se han cumplido dichas condiciones.

1.2. Requisitos técnicos

Los requisitos técnicos, tal como indica en el repositorio de ArduSim [1], son los siguientes:

Requisitos mínimos

- Intel Core i5 (versión de 4 núcleos)
- RAM 6 GB
- Microsoft Windows 7, Linux o MacOS
- Java SE Development Kit (JDK) 11 32 bits
- Cygwin y ImDisk Virtual Disk Driver (solo para Windows)

Requisitos recomendados

- Intel Core i7 (cuatro núcleos con Hyper-Threading)
- RAM 16 GB

- Linux (i.e. Ubuntu 16.04 o Ubuntu 18.04)
- Java SE Development Kit (JDK) 11 64 bits
- Cygwin y ImDisk Virtual Disk Driver (solo Windows)

También se indica que se recomienda el uso de ArduSim en Linux, puesto que el rendimiento se reduce en Windows. Es por esto que en un primer momento se intentó el uso de una máquina virtual con Ubuntu. Sin embargo, el ordenador no tenía capacidad para crear una máquina vitual que cumpliera con los requisitos mínimos, ya que sus especificaciones son las siguientes:

- Intel Core i5 (4 núcleos)
- nVidia GEforce GTX 950M
- Windows 10

Por tanto, empleando el ordenador con Windows se podrían cumplir los requisitos mínimos, pero no los recomendados, provocando un peor rendimiento del simulador y una mayor cantidad de tiempo empleado en las simulaciones.

Por otro lado, se tuvo que instalar Java SE Development Kit (JDK) 11, Cygwin y ImDisk Virtual Disk Driver para cumplir con todos los requerimientos necesarios para el correcto funcionamiento de ArduSim.

1.3. Licencia de uso

Como se indica en el *readme* de ArduSim [2], en la sección de *Citation*, se debe citar el repositorio y el artículo *ArduSim: Accurate and real-time multicopter simulation* cuando se emplee el simulador en algún proyecto. A lo largo de la Memoria se ha hecho referencia a ambos.

Por otro lado, en la sección *Copyright information* se indica que ArduSim está publicado con una licencia Apache 2.0.
Se trata de una licencia de software libre que permite que se modifique el código, por lo que habiando citado los anteriores documentos, tal y como indica el propietario, se ha cumplido con las condiciones de cara al uso de ArduSim.

Bibliografía

[1] Fabra Collado, Francisco. *ArduSim setup*. Recuperado de `https://bitbucket.org/frafabco/ardusim/src/38bab8207e91d6b6ff9f92cd4d3547499d9eefe5/help/setup.md`

[2] Fabra Collado, Francisco. *ArduSim. Readme*. Recuperado de `https://bitbucket.org/frafabco/ardusim/src/master/README.md`

[3] *APACHE LICENSE, VERSION 2.0*. Recuperado de `https://www.apache.org/licenses/LICENSE-2.0`

Parte III

PRESUPUESTO

Capítulo 1

Presupuesto

1.1. Introducción

El objetivo de este documento es exponer la estimación del coste de diseñar e implementar los protocolos desarrollados en este trabajo. Para ello, se van a tener en cuenta los materiales empleados así como las horas de trabajo. Se va a considerar que se contrata a un ingeniero para realizar el proyecto y que éste le dedica una cantidad de horas similares a las dedicadas por el alumno.

La divisa en la que se va a realizar este presupuesto es el Euro [€].

1.2. Costes unitarios

En primer lugar, se van a desglosar los costes unitarios del material, las licencias informáticas y el personal implicado. Posteriormente, se van a estimar los costes de una hora de trabajo.

Personal implicado

Para desarrollar los protocolos en ArduSim, como ya se ha explicado, se supone que se contrata a un ingeniero con un salario aproximado de 30 €/h.

Licencias informáticas

- **InteliJ**: InteliJ ha sido el entorno de desarrollo sobre el que se ha modificado el código de ArduSim. Es un entorno gratuito, por lo que el coste de su licencia es 0

€/h.

- **ArduSim**: como ya se ha visto, ArduSimes *software* libre, de forma que el coste de su uso es de 0 €/h.

- **Matlab**: el coste de una licencia anual para la Universitat Politècnica de València es de 250 euros al año, de forma que su coste por hora es de 0.029 €/h.

- **Microsoft Office 365**: Microsoft Teams se ha empleado en diferentes reuniones a distancia. Se encuentra dentro de la licencia Office 365 que ofrece la UPV. Esta licencia tiene un coste aproximado de 10 €/mes, de forma que su precio por hora estimado es 0.014 €/h.

- **Google Earth**: Google Earth Pro es un software gratuito, por tanto su coste es de 0 €/h.

Materiales

Puesto que la totalidad del trabajo se ha realizado mediante simulaciones en el ordenador, el único material empleado es el ordenador del alumno. Se va a suponer que el ingeniero emplea el mismo portátil. Su precio es de 700 €, con una antigüedad de 4 años, por lo que cada mes de uso ha supuesto 14.6 €. Puesto que el trabajo se ha realizado a lo largo de 6 meses, el precio de los materiales empleados es 87.5 €.

Consumo eléctrico

Para estimar el consumo eléctrico del ordenador durante las horas de trabajo, se ha considerado que el portátil empleado tiene un consumo específico de 0.11 kW [1]. Por otro lado, se ha considerado que el precio de la luz es el precio medio del día 31 de agosto de 2021, 0.22 €/kWh [2], de forma que una hora de uso del ordenador tiene un coste de 0.0242 €/h por consumo eléctrico.

1.2.1. Coste de una hora de desarrollo de código

Con todos los datos anteriores, se puede estimar el coste de una hora de desarrollo de código. Una hora de desarrollo implica una hora de trabajo del ingeniero, empleando InteliJ, así como el consumo eléctrico correspondiente. En la Tabla 1.1 se desglosa y se muestra el coste de una hora de desarrollo.

Tabla 1.1: Coste de una hora de desarrollo

Recurso	Coste unitario [€/h]
Ingeniero	30
InteliJ	0
Electricidad	0.0242
Total por hora	30.0242 €/h

1.2.2. Coste de una hora de simulación

Por otro lado, el coste de una hora de simulación con ArduSim se puede estimar de manera similar, como se muestra en la Tabla 1.2.

Tabla 1.2: Coste de una hora de simulación

Recurso	Coste unitario [€/h]
Ingeniero	30
InteliJ	0
ArduSim	0
Electricidad	0.0242
Total por hora	30.0242 €/h

1.2.3. Coste de una hora de comprobación de resultados

Para la comprobación de resultados se emplea Google Earth y Matlab, por lo que el coste de una hora de análisis de los resultados para comprobar que los protocolos desarrollados cumplen con los objetivos se muestra en la Tabla 1.3.

Tabla 1.3: Coste de una hora de comprobación de resultados

Recurso	Coste unitario [€/h]
Ingeniero	30
Matlab	0.029
Google Earth	0
Electricidad	0.0242
Total por hora	30.0532 €/h

1.2.4. Coste de una hora de reunión telemática

Las reuniones realizadas a lo largo del desarrollo de este proyecto se han efectuado a través de Microsoft Teams. Por tanto, el precio de una hora de reuniones es el que se expone en la Tabla 1.4.

Tabla 1.4: Coste de una hora de reunión telemática

Recurso	Coste unitario [€/h]
Ingeniero	30
Microsoft Teams	0.014
Electricidad	0.0242
Total por hora	30.0382 €/h

1.3. Presupuestos parciales

Una vez expuestos los costes unitarios, se va a obtener el presupuesto para cada etapa de trabajo.

1.3.1. Aprendizaje de uso de ArduSim

En esta etapa se recoge todo el proceso de aprendizaje, de forma que el ingeniero disponga del conocimiento y las herramientas necesarias para desarrollar el nuevo protocolo.

Para ello, se requieren 40 horas de aprendizaje del lenguaje de programación empleado (Java), 15 horas de reuniones explicativas del simulador, 25 horas de desarrollo de código de pruebas y 40 horas de simulaciones en las que se observe el resultado de dichas pruebas. En la Tabla 1.5 se exponen los costes de esta etapa.

Tabla 1.5: Coste del aprendizaje de uso de ArduSim

Trabajo	Tiempo [h]	Coste unitario [€/h]	Coste total [€]
Desarrollo	65	30.0242	1951.57
Simulaciones	40	30.0242	1200.968
Reuniones	15	30.0328	450.492
		Total	3603.03 €

1.3.2. Desarrollo de los protocolos

En la etapa de desarrollo de los protocolos se requieren unas 50 horas de desarrollo del código de dichos protocolos, además de otras 50 horas en las que se simulan y se prueban. Durante este proceso se comprueba que el código se ha desarrollado correctamente, se detectan y corrigen errores y se comprueban los resultados empleando herramientas como el *log* de ArduSim.

En la Tabla 1.6 se recogen los costes de esta parte del trabajo.

Tabla 1.6: Coste del desarrollo de los protocolos

Trabajo	Tiempo [h]	Coste unitario [€/h]	Coste total [€]
Desarrollo	50	30.0242	1501.21
Simulaciones	50	30.0242	1501.21
		Total	3002.42 €

1.3.3. Comprobación de resultados

Finalmente, una vez desarrollados los nuevos protocolos de ArduSim, se comprueba que estos cumplen los objetivos propuestos en este trabajo. Esto implica aproximadamente 20 horas de Matlab y Google Earth, donde se comprueban las gráficas del error así como las trayectorias de los drones del enjambre. En la Tabla 1.7 se exponen los costes de este trabajo.

Tabla 1.7: Coste de la comprobación de resultados

Trabajo	Tiempo [h]	Coste unitario [€/h]	Coste total [€]
Comprobación de resultados	20	30.0532	601.064
		Total	601.07 €

1.4. Presupuesto total

En la Tabla 1.8 se recoge el presupuesto total, una vez vistos los costes de cada etapa del desarrollo de este trabajo.

Tabla 1.8: Coste total

Trabajo	Coste [€]
Aprendizaje	3603.03
Desarrollo	3002.42
Comprobación	601.07
Materiales	87.5
Total	7294.02 €

En definitiva, el presupuesto total es: **7294.02 €**

Bibliografía

[1] Ecorresponsabilidad. *Uso del ordenador portátil vs ordenador sobremesa*. Recuperado de `http://www.ecorresponsabilidad.es/fichas/portatil.htm`. Accedido el 5 de septiembre de 2021.

[2] Tarifaluzahora. *Precio de la luz por horas*. Recuperado de `https://tarifaluzhora.es/?tarifa=pcb&fecha=31%2F08%2F2021`. Accedido el 5 de septiembre de 2021.

Parte IV

ANEXOS

Anexo I - Respuesta de AESA

Gmail Juan Robles <juanrroblesgomez@gmail.com>

RE: Envío de formulario desde: Consultas

1 mensaje

Buzón drones.aesa <drones.aesa@seguridadaerea.es> 7 de septiembre de 2021, 11:06
Para: "juanrroblesgomez@gmail.com" <juanrroblesgomez@gmail.com>

En referencia a la consulta realizada indicar:

- Las operaciones civiles con enjambre de UAS podrían realizarse al amparo de una autorización operacional emitida dentro de la categoría específica. Para solicitar dicha autorización deberán analizar los riesgos de acuerdo al artículo 11 del Re UE 2019/947 y sus medios aceptables de cumplimiento, AMcs (Ver como documentación complementaria Easy Access Rules de EASA https://www.easa.europa.eu/sites/default/files/dfu/Easy%20Access%20Rules%20for%20Unmanned%20Aircraft%20Systems.pdf). Ver https://www.seguridadaerea.gob.es/es/ambitos/drones/operaciones-uas-drones/operaciones-con-uas-drones---categoria-especifica

- Para llevar a cabo esta evaluación de riesgos los AMCs proponen la metodología SORA. En la evaluación deberán especificar el Concepto de Operación (VLOS o BVLOS, nivel de intervención del piloto, Altura, número de aeronaves, distancia mínima entre ellas, protocolo de comunicación, localización, si es zona terrestre controlada…) , analizar el riesgo en tierra y aire de su operación y las mitigaciones necesarias, y con ellos se obtendrán el nivel de riesgo (SAIL) asociado a la operación. En función del SAIL, el nivel de robustez de los objetivos de seguridad operacional (OSOs) será diferente, y por tanto los requisitos asociados.

- Adicionalmente el Real Decreto 1919/2009 de 11 de diciembre, por el que se regula la seguridad aeronáutica en las demostraciones aéreas civiles es de aplicación a toda demostración anunciada públicamente y abierta al público en general o de acceso restringido.

- Las disposiciones en materia de seguridad pública o restricciones al vuelo de UAS por motivo del lugar de operación contemplados en el Real Decreto 1036/2017, así como las reglas del aire aplicables a los UAS del Real Decreto 1180/2018 por el que se desarrolla el Reglamento del aire, seguirán siendo de aplicación a las operaciones con UAS. Ver https://www.seguridadaerea.gob.es/es/ambitos/drones/operaciones-uas-drones/vuelos-con-uas-drones-zonificacion

- Puede encontrar más información en https://www.seguridadaerea.gob.es/es/ambitos/drones

Un saludo

Buzón de Drones

División de Sistemas de Aeronaves No Tripuladas (UAS)

Tel +34 91 396 8000

E-mail: drones.aesa@seguridadaerea.es

Spanish Aviation Safety and Security Agency

www.seguridadaerea.gob.es

Paseo de la Castellana, 112 28046 Madrid

@AesaSpain

-----Mensaje original-----
De: aesa@seguridadaerea.es <aesa@seguridadaerea.es>
Enviado el: jueves, 24 de junio de 2021 11:41
Para: Buzón drones.aesa <drones.aesa@seguridadaerea.es>
Asunto: Envío de formulario desde: Consultas

Enviado el Jueves, Junio 24, 2021 - 11:41 Enviado por el usuario: Anónimo Submitted values are:

Nombre: Juan Robles

Organización:

Email: juanrroblesgomez@gmail.com

Repetir Email: juanrroblesgomez@gmail.com

Asunto: Drones. Operaciones/Tecnología

Drones. Operaciones/Tecnología: Aeronaves pilotadas por control remoto (RPA)

Descripción:

Buenos días:

Me llamo Juan y soy estudiante de ingeniería aeroespacial. Les escribo porque estoy realizando mi trabajo de fin de grado sobre enjambres de drones y me gustaría saber lo siguiente:

¿Se permite el vuelo en enjambre en algún caso? En caso de que no sea así, ¿se tiene alguna previsión de cuándo se permitirán o alguna idea de cómo será la normativa que los regule?

Por otro lado, ¿cuáles son las principles preocupaciones con los enjambres desde el punto de vista de la seguridad aérea?

Muchas gracias de antemano.

Un cordial saludo.

Anexo II - Talker Thread

```
package protocols.followme.logic;

import api.API;
import com.esotericsoftware.kryo.io.Output;
import es.upv.grc.mapper.Location2DGeo;
import es.upv.grc.mapper.Location2DUTM;
import protocols.followme.pojo.Message;
import main.api.ArduSim;
import main.api.Copter;
import main.api.GUI;
import main.api.communications.CommLink;

import java.util.Arrays;
import java.util.concurrent.atomic.AtomicInteger;

import static protocols.followme.pojo.State.*;

/* Developed by: Francisco Jos&eacute; Fabra Collado, from GRC research
group in Universitat Politecnica de Valencia (Valencia,
Spain). */

public class FollowMeTalkerThread extends Thread {

    private AtomicInteger currentState;

    private Copter copter;
    private GUI gui;
    private ArduSim ardusim;
    private CommLink link;
    private byte[] outBuffer;
    private Output output;
    private byte[] message;

    private long cicleTime;     // Cicle time used for sending messages
    private long waitingTime;   // (ms) Time to wait between two sent messages

    protected static volatile boolean protocolStarted = false;

    @SuppressWarnings("unused")
    private FollowMeTalkerThread() {}

    public FollowMeTalkerThread(int numUAV) {
        super(FollowMeText.TALKER_THREAD + numUAV);
        currentState = FollowMeParam.state[numUAV];
        this.ardusim = API.getArduSim();
        this.copter = API.getCopter(numUAV);
        this.gui = API.getGUI(numUAV);
        this.link = API.getCommLink(numUAV);
        this.outBuffer = new byte[CommLink.DATAGRAM_MAX_LENGTH];
        this.output = new Output(outBuffer);

        this.cicleTime = 0;
    }
```

```
    @Override
    public void run() {
        /* FOLLOWING PHASE */
        gui.logVerboseUAV(FollowMeText.MASTER_WAIT_ALTITUDE);
        while (!protocolStarted) {
            ardusim.sleep(FollowMeParam.STATE_CHANGE_TIMEOUT);
        }

        Location2DUTM here = copter.getLocationUTM();
        Location2DUTM prevLocation = copter.getLocationUTM();
        Location2DGeo hereGeo;

        double z, heading, speed;
        double track = 0;
        cicleTime = System.currentTimeMillis();
        while (currentState.get() == FOLLOWING) {
            // Track
            prevLocation = here;
            here = copter.getLocationUTM();

            hereGeo = copter.getLocationGeo();
            //gui.log("Lat: " + String.valueOf(hereGeo.latitude) + "grados");
            //gui.log("Long: " + String.valueOf(hereGeo.longitude) + "grados");

            if (here.x != prevLocation.x && here.y != prevLocation.y) {
                track = Math.PI/2-Math.atan2(here.y-prevLocation.y,here.x-prevLocation.x);
                if (track < 0){
                    track = track + 2*Math.PI;
                }
                else{
                    if (track > 2*Math.PI){
                        track = track - 2*Math.PI;
                    }
                }
            }


            //gui.log("Track: " + String.valueOf(track) + "rad");



            z = copter.getAltitudeRelative();
            heading = copter.getHeading();
            speed = copter.getSpeed();
            output.reset();
            output.writeShort(Message.I_AM_HERE);
//          output.writeLong(selfId);
            output.writeDouble(here.x);
            output.writeDouble(here.y);
            output.writeDouble(z);
            output.writeDouble(heading);
            output.writeDouble(speed);
            output.writeDouble(track);
            output.writeDouble(hereGeo.latitude);
            output.writeDouble(hereGeo.longitude);
            output.flush();
            message = Arrays.copyOf(outBuffer, output.position());
            link.sendBroadcastMessage(message);

            // Timer
            cicleTime = cicleTime + FollowMeParam.sendPeriod;
            waitingTime = cicleTime - System.currentTimeMillis();
            if (waitingTime > 0) {
                ardusim.sleep(waitingTime);
            }
        }
        while (currentState.get() < LANDING) {
            ardusim.sleep(FollowMeParam.STATE_CHANGE_TIMEOUT);
        }
```

```
        /** LANDING PHASE */
        gui.logVerboseUAV(FollowMeText.MASTER_SEND_LAND);
        output.reset();
        output.writeShort(Message.LAND);
        here = copter.getLocationUTM();
        output.writeDouble(here.x);
        output.writeDouble(here.y);
        output.writeDouble(copter.getHeading());
        output.flush();
        message = Arrays.copyOf(outBuffer, output.position());

        cicleTime = System.currentTimeMillis();
        while (currentState.get() == LANDING) {
            link.sendBroadcastMessage(message);

            // Timer
            cicleTime = cicleTime + FollowMeParam.SENDING_TIMEOUT;
            waitingTime = cicleTime - System.currentTimeMillis();
            if (waitingTime > 0) {
                ardusim.sleep(waitingTime);
            }
        }
        while (currentState.get() < FINISH) {
            ardusim.sleep(FollowMeParam.STATE_CHANGE_TIMEOUT);
        }

        /** FINISH PHASE */
        gui.logVerboseUAV(FollowMeText.TALKER_FINISHED);
    }


}
```

Anexo III - Listener Thread

```
package protocols.followme.logic;

import api.API;
import com.esotericsoftware.kryo.io.Input;
import es.upv.grc.mapper.*;
import main.api.formations.FlightFormation;
import protocols.followme.pojo.Message;
import main.Param;
import main.api.*;
import main.api.communications.CommLink;
import main.api.masterslavepattern.MasterSlaveHelper;
import main.api.masterslavepattern.discovery.DiscoveryProgressListener;
import main.api.masterslavepattern.safeTakeOff.SafeTakeOffContext;
import main.api.masterslavepattern.safeTakeOff.SafeTakeOffListener;
import main.sim.logic.SimParam;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Map;
import java.util.concurrent.atomic.AtomicInteger;

import static protocols.followme.pojo.State.*;

/* Developed by: Francisco Jos&eacute; Fabra Collado, from GRC research
group in Universitat Polit&egrave;cnica de Val&egrave;ncia (Valencia,
Spain). */

public class FollowMeListenerThread extends Thread {

    private AtomicInteger currentState;

    private int numUAV;
    private long selfId;
    private boolean isMaster;
    private Copter copter;
    private MasterSlaveHelper msHelper;
    private SafeTakeOffHelper takeOffHelper;
    private GUI gui;
    private ArduSim ardusim;
    double MinimumDistance;

    private CommLink link;
    byte[] inBuffer;
    Input input;

    @SuppressWarnings("unused")
    private FollowMeListenerThread() {}

    public FollowMeListenerThread(int numUAV) {
        super(FollowMeText.LISTENER_THREAD + numUAV);
        currentState = FollowMeParam.state[numUAV];
        this.numUAV = numUAV;
        this.copter = API.getCopter(numUAV);
```

```
        this.selfId = this.copter.getID();
        this.msHelper = this.copter.getMasterSlaveHelper();
        this.isMaster = this.msHelper.isMaster();
        this.takeOffHelper = this.copter.getSafeTakeOffHelper();
        this.gui = API.getGUI(numUAV);
        this.link = API.getCommLink(numUAV);
        this.inBuffer = new byte[CommLink.DATAGRAM_MAX_LENGTH];
        this.input = new Input(inBuffer);
        this.ardusim = API.getArduSim();
    }

    @Override
    public void run() {
        while (!ardusim.isAvailable()) {
            ardusim.sleep(FollowMeParam.STATE_CHANGE_TIMEOUT);
        }

        /** START PHASE */
        gui.logUAV(FollowMeText.START);
        // Let the master detect slaves until the setup button is pressed
        Map<Long, Location2DUTM> UAVsDetected = null;
        if (this.isMaster) {
            gui.logVerboseUAV(FollowMeText.SLAVE_START_LISTENER);
            final AtomicInteger totalDetected = new AtomicInteger();
            UAVsDetected = msHelper.DiscoverSlaves(new DiscoveryProgressListener() {

                @Override
                public boolean onProgressCheckActionPerformed(int numUAVs) {
                    // Just for logging purposes
                    if (numUAVs > totalDetected.get()) {
                        totalDetected.set(numUAVs);
                        gui.log(FollowMeText.MASTER_DETECTED_UAVS + numUAVs);
                    }
                    // We decide to continue when the setup button is pressed
                    return numUAVs == API.getArduSim().getNumUAVs() - 1;
                }
            });
        } else {
            gui.logVerboseUAV(FollowMeText.LISTENER_WAITING);
            msHelper.DiscoverMaster();
        }

        while(Param.simStatus != Param.SimulatorState.SETUP_IN_PROGRESS){
            ardusim.sleep(SimParam.SHORT_WAITING_TIME);
        }

        /* TAKE OFF PHASE */
        currentState.set(TAKE_OFF);
        gui.logUAV(FollowMeText.SETUP);
        gui.updateProtocolState(FollowMeText.SETUP);
        // 1. Synchronize master with slaves to get the takeoff sequence in
        //the take off context object
        SafeTakeOffContext takeOff;
        if (this.isMaster) {
            double formationYaw;
            if (ardusim.getArduSimRole() == ArduSim.MULTICOPTER) {
                formationYaw = copter.getHeading();
            } else {
                formationYaw = FollowMeParam.masterInitialYaw;
            }
            FlightFormation F =
            API.getFlightFormationTools().getFlyingFormation(UAVsDetected.size() + 1);
            takeOff = takeOffHelper.getMasterContext(UAVsDetected,F,
                    formationYaw, FollowMeParam.slavesStartingAltitude, true, true);
            MinimumDistance = F.getFormationMinimumDistance();
        } else {
            takeOff = takeOffHelper.getSlaveContext(false);
        }
        gui.logUAV("ready to takeOff");
```

```
// 2. Take off all the UAVs
takeOffHelper.start(takeOff, new SafeTakeOffListener() {

    @Override
    public void onCompleteActionPerformed() {
        currentState.set(SETUP_FINISHED);
    }
});
while (currentState.get() < SETUP_FINISHED) {
    ardusim.sleep(FollowMeParam.STATE_CHANGE_TIMEOUT);
}

/* SETUP FINISHED PHASE */
gui.logUAV(FollowMeText.SETUP_FINISHED);
gui.updateProtocolState(FollowMeText.SETUP_FINISHED);
gui.logVerboseUAV(FollowMeText.LISTENER_WAITING);
while (currentState.get() == SETUP_FINISHED) {
    ardusim.sleep(FollowMeParam.STATE_CHANGE_TIMEOUT);
    // Coordination with ArduSim
    if (ardusim.isExperimentInProgress()) {
        currentState.set(FOLLOWING);
    }
}

/* FOLLOWING PHASE */
gui.logUAV(FollowMeText.FOLLOWING);
gui.updateProtocolState(FollowMeText.FOLLOWING);
long waitingTime;

double currentTime = System.currentTimeMillis();
double prevTime = currentTime;

double lastErrorNorth = 0;
double lastErrorEast = 0;
double lastErrorDown = 0;

double cumErrorNorth = 0;
double cumErrorEast = 0;
double cumErrorDown = 0;

if (this.isMaster) {
    new FollowMeTalkerThread(numUAV).start();
    // Wait until the master UAV descends below a threshold (in the remote thread)
    while (currentState.get() == FOLLOWING) {
        ardusim.sleep(FollowMeParam.STATE_CHANGE_TIMEOUT);
    }
} else {
    gui.logVerboseUAV(FollowMeText.SLAVE_WAIT_ORDER_LISTENER);
    Location2DUTM masterLocation;
    Location2DGeo targetLocation = null;
    Location2DUTM targetLocationUTM = null;
    Location2DGeo moveToLocation = null;
    Location2DUTM moveToLocationUTM = null;
    Location3D targetLocationLanding = null;

    while (currentState.get() == FOLLOWING) {
        inBuffer = link.receiveMessage();
        if (inBuffer != null) {
            input.setBuffer(inBuffer);
            short type = input.readShort();

            if (type == Message.I_AM_HERE){
                // Master
                masterLocation = new
                    Location2DUTM(input.readDouble(), input.readDouble());
                double relAltitude = input.readDouble();
                double masterHeading = input.readDouble();
                double masterSpeed = input.readDouble();
                double masterTrack = input.readDouble();
```

```
                double masterLat = input.readDouble();
                double masterLon = input.readDouble();



                // Prediction
                Location2DUTM predictedPos;
                double predictedT = 0;  // seconds
                double predictedX = masterLocation.x +
                masterSpeed * Math.sin(masterTrack) * predictedT;
                double predictedY = masterLocation.y +
                masterSpeed * Math.cos(masterTrack) * predictedT;
                predictedPos = new Location2DUTM(predictedX, predictedY);



                // Distance
                Location3DUTM loc1 = new Location3DUTM(masterLocation,relAltitude);
                Location3DUTM loc2 = new
                    Location3DUTM(copter.getLocationUTM(),
                    copter.getAltitudeRelative());

                double distance = loc1.distance3D(loc2);

                // Track
                double mastDegTrack = masterTrack *180/Math.PI;
                double DegTrack = copter.getHeading()*180/Math.PI;

                // Location Geo
                Location2DGeo locGeo = copter.getLocationGeo();
                double Latitude = locGeo.latitude;
                double Longitude = locGeo.longitude;

                // Timestamp
                double timestamp = System.currentTimeMillis();

                // NEW VERSION

                targetLocationUTM =
                    takeOff.getFormationFlying().getLocation(
                    takeOff.getFormationPosition(), predictedPos, masterHeading);
                double error=targetLocationUTM.distance(copter.getLocationUTM());

            try {
                targetLocation =
                    takeOff.getFormationFlying().
                    getLocation(takeOff.getFormationPosition(),
                    masterLocation, masterHeading).getGeo();


                //PID
                currentTime = System.currentTimeMillis();
                double elapsedTime = (currentTime - prevTime)/1000;

                // Constants
                double kpNorth = 0.2;
                double kdNorth = 0.29;
                double kiNorth = 0;

                double kpEast = 0.2;
                double kdEast = 0.29;
                double kiEast = 0;

                double kpDown = 0.15;
                double kdDown = 0.15;
                double kiDown = 0;

                //vNorth
                double errorNorth=targetLocationUTM.y-copter.getLocationUTM().y;
```

```
                cumErrorNorth = cumErrorNorth + (errorNorth * elapsedTime);
                double rateErrorNorth = (errorNorth - lastErrorNorth)/elapsedTime;
                double vNorth = kpNorth * errorNorth + kdNorth * rateErrorNorth +
                    kiNorth * cumErrorNorth;
                //double vNorth = 0;

                lastErrorNorth = errorNorth;


                //vEast
                double errorEast=targetLocationUTM.x-copter.getLocationUTM().x;
                cumErrorEast = cumErrorEast + (errorEast * elapsedTime);
                double rateErrorEast = (errorEast - lastErrorEast)/elapsedTime;
                double vEast = kpEast * errorEast + kdEast * rateErrorEast +
                    kiEast * cumErrorEast;
                //double vEast = 0;

                lastErrorEast = errorEast;

                //vDown
                double errorDown=relAltitude-copter.getAltitudeRelative();
                cumErrorDown = cumErrorDown + (errorDown * elapsedTime);
                double rateErrorDown = (errorDown - lastErrorDown)/elapsedTime;
                double vDown = -(kpDown * errorDown + kdDown * rateErrorDown +
                    kiDown * cumErrorDown);
                //double vDown = 0;

                lastErrorDown = errorDown;

                prevTime = currentTime;

                // Results
                gui.log("V north: " + String.valueOf(vNorth) + " m/s");
                gui.log("V east: " + String.valueOf(vEast) + " m/s");
                gui.log("V down: " + String.valueOf(vDown) + " m/s");

                copter.moveTo(vNorth, vEast, vDown);

            } catch (LocationNotReadyException e) {
                gui.log(e.getMessage());
                e.printStackTrace();
                // Fatal error. It lands
                currentState.set(LANDING);
            }

/*
    // Initial version:

try {
    targetLocation = takeOff.getFormationFlying().
        getLocation(takeOff.getFormationPosition(), masterLocation,
        masterHeading).getGeo();
    copter.moveTo(new Location3DGeo(targetLocation, relAltitude));
} catch (LocationNotReadyException e) {
    gui.log(e.getMessage());
    e.printStackTrace();
    // Fatal error. It lands
    currentState.set(LANDING);
}*/


    // RESULTS OUTPUT
    String resultsOutput = String.valueOf(timestamp) + ";" +
    String.valueOf(distance) +
            ";" + String.valueOf(error)  +";" +
            String.valueOf(mastDegTrack) +
            ";" + String.valueOf(masterHeading) +
            ";" + String.valueOf(DegTrack) +
            ";" + String.valueOf(copter.getHeading()) +
```

```
                ";" + String.valueOf(masterLat) + ";" +
                String.valueOf(masterLon) + ";" +
                String.valueOf(Latitude) + ";" +
                String.valueOf(Longitude) + "\n";
        try {
            File f = new File("C:\\Users\\Juan\\Documents\\TFG\\Results
                \\results_v3_0.csv");
            boolean writeHeader = !f.exists();
            FileWriter fr = new FileWriter(f, true);
            if (writeHeader) {
                fr.write("Tiempo;Distancia;Error;Master_track;Master_heading;" +
                "Esclavo_track;Esclavo_heading;Master_lat;" +
                "Master_lon;Esclavo_lat;Esclavo_lon\n");
            }
            fr.write(resultsOutput);
            fr.close();
        }   catch (IOException e) {
            e.printStackTrace();
        }

            // GUI results
            //gui.log("Distance: " + String.valueOf(distance) + " m");
            //gui.log("Minimum distance: "+String.valueOf(MinimumDistance)+" m");
            //gui.log("Master track: " + String.valueOf(mastDegCourse) + "    ");
            //gui.log("Slave track: " + String.valueOf(DegCourse) + "    ");
            gui.log("Error: " + String.valueOf(error) + " m");
            }


            if (type == Message.LAND) {
                Location2DUTM centerUAVFinalLocation = new
                    Location2DUTM(input.readDouble(), input.readDouble());
                double yaw = input.readDouble();
                Location2DUTM landingLocationUTM = takeOff.getFormationLanding().
                    getLocation(takeOff.getFormationPosition(),
                    centerUAVFinalLocation, yaw);
                try {
                    targetLocationLanding = new Location3D(landingLocationUTM,
                        copter.getAltitudeRelative());
                    currentState.set(MOVE_TO_LAND);
                } catch (LocationNotReadyException e) {
                    gui.log(e.getMessage());
                    e.printStackTrace();
                    // Fatal error. It lands
                    currentState.set(LANDING);
                }
            }
        }
    }

    /** MOVE TO LAND PHASE */
    if (currentState.get() == MOVE_TO_LAND) {
        gui.logUAV(FollowMeText.MOVE_TO_LAND);
        gui.updateProtocolState(FollowMeText.MOVE_TO_LAND);
        gui.logVerboseUAV(FollowMeText.LISTENER_WAITING);
        MoveTo moveTo = copter.moveTo(targetLocationLanding, new MoveToListener(){

                    @Override
                    public void onFailure() {
                        gui.exit(FollowMeText.MOVE_ERROR + " " + selfId);
                    }

                    @Override
                    public void onCompleteActionPerformed() {
        // Nothing to do, as we wait until the target location is
        // reached with Thread.join()
                    }
                });
        moveTo.start();
```

```
                try {
                    moveTo.join();
                } catch (InterruptedException e) {}
                currentState.set(LANDING);
            }
        }

        /** LANDING PHASE */
        if (!copter.land()) {
            gui.exit(FollowMeText.LAND_ERROR + " " + selfId);
        }
        gui.logUAV(FollowMeText.LANDING);
        gui.updateProtocolState(FollowMeText.LANDING);
        gui.logVerboseUAV(FollowMeText.LISTENER_WAITING);
        long cicleTime = System.currentTimeMillis();
        while (currentState.get() == LANDING) {
            if(!copter.isFlying()) {
                currentState.set(FINISH);
            } else {
                cicleTime = cicleTime + FollowMeParam.LAND_CHECK_TIMEOUT;
                waitingTime = cicleTime - System.currentTimeMillis();
                if (waitingTime > 0) {
                    ardusim.sleep(waitingTime);
                }
            }
        }

        /** FINISH PHASE */
        gui.logUAV(FollowMeText.FINISH);
        gui.updateProtocolState(FollowMeText.FINISH);
        gui.logVerboseUAV(FollowMeText.LISTENER_FINISHED);
    }
```

Anexo IV - Matlab: Generación de los resultados

```
%% Juan R. Robles Gomez
% juarobgo@etsid.upv.es
% Ingenieria aeroespacial UPV 2021


%% Trabajo de fin de grado
% Se comprueba como evoluciona la distancia entre
% dos drones que siguen el protocolo Follow Me de
% ardusim. El maestro env a la predicci n de su
% posici n. Esta predicci n se estima
% multiplicando la velocidad del maestro por un
% tiempo.

disp('Juan Roberto Robles G mez');
disp('Comprobacion de la distancia con el nuevo protocolo FollowMe');
disp('Ingenieria aeroespacial UPV');
fprintf('\n');

%% Lectura del csv
datos_v0 = lectura_datos('Results/results_v0.csv');

datos_v1_1 = lectura_datos('Results/results_v1_1.csv');
datos_v1_3 = lectura_datos('Results/results_v1_3.csv');
datos_v1_5 = lectura_datos('Results/results_v1_5.csv');
datos_v1_10 = lectura_datos('Results/results_v1_10.csv');

datos_v2_1 = lectura_datos('Results/results_v2_1.csv');
datos_v2_3 = lectura_datos('Results/results_v2_3.csv');
datos_v2_5 = lectura_datos('Results/results_v2_5.csv');
datos_v2_10 = lectura_datos('Results/results_v2_10.csv');

datos_v3_0 = lectura_datos('Results/results_v3_0.csv');
datos_v3_1 = lectura_datos('Results/results_v3_1.csv');
datos_v3_3 = lectura_datos('Results/results_v3_3.csv');
datos_v3_5 = lectura_datos('Results/results_v3_5.csv');

%% Plots de resultados
figure;
plot(datos_v0.Tiempo, datos_v0.Error, 'g');
xlim([0 90]);
xlabel('Tiempo (s)', 'FontSize', 12);
ylabel('Error (m)', 'FontSize', 12);
title('Distancia esclavo - posici n deseada', 'FontSize', 14);
subtitle('Protocolo FollowMe original');
legend('Error', 'FontSize', 10);
grid on;


figure;
plot(datos_v0.Tiempo, datos_v0.Error, datos_v1_1.Tiempo,...
    datos_v1_1.Error, datos_v1_3.Tiempo, datos_v1_3.Error,...
```

```
    datos_v1_5.Tiempo, datos_v1_5.Error,...
    datos_v1_10.Tiempo, datos_v1_10.Error);
xlim([0 90]);
xlabel('Tiempo (s)', 'FontSize', 12);
ylabel('Error (m)', 'FontSize', 12);
title('Distancia esclavo - posici n deseada', 'FontSize', 14);
subtitle('Protocolo 1: Predicci n de la posici n');
legend('Error original', 'Error 1s', 'Error 3s', 'Error 5s', 'Error 10s', 'FontSize', 10);
grid on;


figure;
plot(datos_v0.Tiempo, datos_v0.Error, datos_v2_1.Tiempo,...
    datos_v2_1.Error, datos_v2_3.Tiempo, datos_v2_3.Error,...
    datos_v2_5.Tiempo, datos_v2_5.Error,...
    datos_v2_10.Tiempo, datos_v2_10.Error);
xlim([0 90]);
xlabel('Tiempo (s)', 'FontSize', 12);
ylabel('Error (m)', 'FontSize', 12);
title('Distancia esclavo - posici n deseada', 'FontSize', 14);
subtitle('Protocolo 2: Velocidad a partir de la predicci n');
legend('Error original', 'Error 1s', 'Error 3s', 'Error 5s', 'Error 10s', 'FontSize', 10);
grid on;

figure;
plot(datos_v0.Tiempo, datos_v0.Error, datos_v3_0.Tiempo,...
    datos_v3_0.Error, datos_v3_1.Tiempo, datos_v3_1.Error,...
    datos_v3_3.Tiempo, datos_v3_3.Error, ...
    datos_v3_5.Tiempo, datos_v3_5.Error);
xlim([0 90]);
xlabel('Tiempo (s)', 'FontSize', 12);
ylabel('Error (m)', 'FontSize', 12);
title('Distancia esclavo - posici n deseada', 'FontSize', 14);
subtitle('Protocolo 3: Velocidad a partir de la predicci n con PID');
legend('Error original', 'Error 0s', 'Error 1s', 'Error 3s', 'Error 5s', 'FontSize', 10);
grid on;

figure;
plot(datos_v0.Tiempo, datos_v0.Error, datos_v3_0.Tiempo,...
    datos_v3_0.Error);
xlim([0 90]);
xlabel('Tiempo (s)', 'FontSize', 12);
ylabel('Error (m)', 'FontSize', 12);
title('Distancia esclavo - posici n deseada', 'FontSize', 14);
subtitle('Protocolo 3: Velocidad a partir de controladores PID');
legend('Error original', 'Error tercera versi n', 'FontSize', 10);
grid on;
```

Anexo V - Matlab: Función *lectura_datos()*

```
function [datos] = lectura_datos(filename)
% datos = lectura_datos(filename)
% Devuelve un struct con los datos del csv

    data = readtable(filename);
    n_data = height(data);


    datos.Tiempo_abs = data.Tiempo';
    datos.Distancia = data.Distancia';
    datos.Error = data.Error';
    datos.Master_track = data.Master_track';
    datos.Master_heading = data.Master_heading'*180/pi;
    datos.Master_lat = data.Master_lat';
    datos.Master_lon = data.Master_lon';
    datos.Esclavo_track = data.Esclavo_track';
    datos.Esclavo_heading = data.Esclavo_heading'*180/pi;
    datos.Esclavo_lat = data.Esclavo_lat';
    datos.Esclavo_lon = data.Esclavo_lon';

    try
        datos.T_predic = data.T_predic';
    catch
    end

    for i = 1:n_data
        datos.Tiempo(i) = (datos.Tiempo_abs(i) - datos.Tiempo_abs(1))/1000;
    end

end
```