UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FIN DE GRADO

Procesado en tiempo real de rutas e incidencias aéreas para la gestión de drones comerciales

Autor:
Santiago Lozano Terol

Tutor:
Dr. Daniel Díaz-Sánchez

26 de septiembre de 2016

«*Nadie se ha ahogado jamás en su propio sudor.*»

Ann Landers

UNIVERSIDAD CARLOS III DE MADRID

Abstract

Grado en Ingeniería en Tecnologías de Telecomunicación

Procesado en tiempo real de rutas e incidencias aéreas para la gestión de drones comerciales

by Santiago Lozano Terol

Currently, we all know the concept of Unmanned Aerial Vehicles (UAV), although we refer to them colloquially as drones. See one of these small aircraft, which need no pilot on board, spraying a plantation, recording the traffic situation, flying over the scene of a concert or delivering a package are situations that, although recently seemed science fiction, start to assumme a strong note of reality. The advancement of technology allows these devices, historically linked to military operations, reach millions of homes and businesses around the world, helping people to perform tasks that were not possible before, or had a very high risk to be. As it has happened with other technologies at this times that we live in, what began as an eccentricity has become an advantageous option and, soon, will be an irreplaceable necessity.

Drones begin to be everywhere and, of course, begin to create problems. Among its infinite virtues are some capabilities that, for misconduct or sheer recklessness, can cause serious problems in airspace or threaten physical integrity and privacy of people.

Are we prepared to meet these challenges? Will we be prepared when drones that fly through the skies every day will be counted by millions? In this project these questions are addressed, posing the problem and trying to design a valid solution utilizing new Big Data analysis tools. In addition, a prototype showing a first implementation of this design it is presented and tested.

UNIVERSIDAD CARLOS III DE MADRID

Resumen

Grado en Ingeniería en Tecnologías de Telecomunicación

Procesado en tiempo real de rutas e incidencias aéreas para la gestión de drones comerciales

by Santiago Lozano Terol

Actualmente, todos conocemos el concepto de los vehículos aéreos no tripulados, aunque nos refiramos a ellos coloquialmente como drones. Ver a una de estas pequeñas aeronaves, que no necesitan de piloto a bordo, fumigando una plantación, grabando la situación del tráfico, sobrevolando el escenario de un concierto o repartiendo un paquete son situaciones que, aunque hace poco parecían de ciencia ficción, comienzan a coger claros tintes de realidad. El avance de la tecnología permite que estos aparatos, históricamente ligados a operaciones militares, lleguen a millones de hogares y empresas de todo el mundo, ayudando al ser humano a realizar tareas que antes no eran posibles, o tenían un riesgo muy elevado. Como ha ocurrido con otras tecnologías en este tiempo que nos ha tocado vivir, lo que comenzó siendo una excentricidad se ha convertido en una opción ventajosa y, dentro de poco, será una necesidad irremplazable.

Los drones comienzan a estar en todas partes y, evidentemente, comienzan a crear problemas. Entre sus infinitas virtudes se encuentran algunas capacidades que, por mala intención o por pura imprudencia, pueden provocar graves accidentes en el espacio aéreo o atentar contra la integridad física y la privacidad de las personas.

¿Estamos preparados para hacer frente a estos desafíos? ¿Lo estaremos cuando se cuenten por millones los drones que surquen los cielos cada día? En este proyecto se abordan estas preguntas, planteando el problema y tratando de diseñar una solución válida que utilice las tecnologías de análisis de Big Data existentes hoy en día. Además, se presenta y se pone a prueba un prototipo que muestra una primera implementación de este diseño.

Agradecimientos

A Dani, mi tutor, por su ayuda y sus consejos, más allá de este proyecto. Por sacar tiempo para mí cuando lo tenía difícil.

Muy especialmente a mis padres, porque a su sacrificio, su apoyo, su ejemplo y su cariño debo y deberé cada uno de los éxitos de mi vida. Espero que ésto empiece a devolveros todo lo que me habéis dado.

A mi familia, a todos mis tíos y primos, *Lozanos* y *Teroles* y a mis amigos y amigas, porque han sido cuatro años geniales y gran parte de eso se lo debo a ellos.

A mis hermanos, por la responsabilidad que conlleva saber que soy el mayor de vosotros. Con mayor o menor éxito, me he esforzado por ser un buen ejemplo, y estoy seguro de que eso me ha ayudado en todos los aspectos de mi vida, incluido el académico.

A mi abuelo Aurelio y a mi abuelo Pepe, así como a mis abuelas, Angelines e Isabel. Ellos aquí y ellas juntas desde otra parte, sé que presumen de nieto, y me llena de orgullo.

A todos mis compañeros a lo largo de la carrera, con especial cariño a Elena, Irene, Nacho y Víctor. Espero que estos años hayan sido solo el comienzo de una larga amistad. Gracias por ayudarme a llegar hasta aquí.

Por supuesto, a Jesús, que ha sido la persona con la que más tiempo he pasado estos cuatro años. Gracias por las aventuras y los buenos recuerdos. Lo hemos conseguido, y ha sido muy divertido.

A Tere. Por **todo** lo que cabe en una palabra y, a la vez, no cabría en todas estas páginas. Caería en un tópico si te dijera eso de que no hubiera sido posible sin ti. Y es que quizá lo hubiera sido, pero sin ti no tendría sentido.

Índice general

Índice de figuras

Índice de cuadros

List of Abbreviations

AESA	**A**gencia **E**statal de **S**eguridad **A**érea
ADS-B	**A**utomatic **D**ependant **S**urveillance **B**roadcast
CERN	**C**onseil **E**uropéen pour la **R**echerche **N**ucléaire
EASA	**E**uropean **A**viation **S**afety **A**gency
EE. UU.	**E**stados **U**nidos
FAA	**F**ederal **A**viation **A**gency
GPS	**G**lobal **P**ositioning **S**ystem
HDFS	**H**adoop **D**istributed **F**ile **S**ystem
ICAO	**I**nternational **C**ivil **A**viation **O**rganization
JVM	**J**ava **V**irtual **M**achine
LIDAR	**L**aser **I**maging **D**etection **A**nd **R**anging
SQL	**S**tructured **Q**uery **L**anguage
RDD	**R**esilient **D**istributed **D**atasets
UAT	**U**niversal **A**ccess **T**ransceiver
UAV	**U**nmaned **A**ircraft **V**ehicle

Capítulo 1

Introducción

1.1. Motivación del proyecto

A principios de este siglo los UAV (de las siglas de Vehículo Aéreo no Tripulado, en inglés: *Unmanned Aircraft Vehicle*) o drones eran aparatos de uso casi exclusivamente militar, con cierto interés para los aeromodelistas, pero desconocidos para el público en general. La gran revolución en el mercado de estas aeronaves surgió, probablemente, tras los atentados del 11 de septiembre de 2001 en los Estados Unidos y las posteriores guerras de Irak y Afganistán, donde los drones jugaron un papel fundamental en las estrategias de los EE. UU. y sus países aliados.

FIGURA 1.1: Dron *Predator*, desarrollado en los 90 por la Fuerza Aérea estadounidense

A finales de 2002, el ejército de EE. UU. contaba aproximadamente con 200 drones[1]. A día de hoy, el número supera los 11.000, y continúa en aumento. La demanda de esta tecnología por parte de las grandes potencias militares generó tal *boom* industrial y científico que no es de extrañar que las compañías comenzaran a interesarse por llevar estos avances al resto de la sociedad. Según estimaciones del Teal Group[2], la introducción de

los drones al plano civil podría generar a los fabricantes unas ganancias de 11.600 millones de dólares para el año 2023.

Así, del mismo modo que sucedió con otros grandes avances como la radio o el GPS, un invento que nació para revolucionar el entorno militar acabó transformando distintos ámbitos de la sociedad civil. Hoy, los drones han cambiado una gran cantidad de industrias, y las nuevas posibilidades que abren se aplican a investigación, conservación, agricultura, fotografía, salvamento, infraestructuras, rescate y búsqueda de víctimas, entre otras muchas cosas.

FIGURA 1.2: Dron realizando funciones agrícolas

Sin embargo, no todos son ventajas. La irrupción en masa de los drones provoca también una serie de problemas, en su mayoría relacionados con la falta de una regulación firme. Algunos de estos problemas podrían ser:

- **El espacio aéreo se ve amenazado:** Son cada vez más frecuentes las noticias en las que la seguridad de los pasajeros y la tripulación de algún avión comercial se ve amenazada por la aparición de drones en las inmediaciones de algunos aeropuertos cuando el avión se disponía aterrizar, siendo casos sonados los del aeropuerto Charles de Gaulle[3] en febrero de este año o, más recientemente, en agosto, en los aeropuertos de Múnich[4] y de Cornualles[5].
- **Peligro en zonas pobladas:** Aitor Goikoetxea, director de Drone School, donde se imparten clases para volar este tipo de apartos, dijo en una entrevista el año pasado: «Los drones no son juguetes. Son aparatos que pueden llegar a pesar 25 kilos y, si se utiliza de una manera que no es adecuada, puede causar una desgracia». Estas declaraciones llegaban después de que una vecina de Zestoa, en Guipúzcoa, sufriera gravísimas heridas en la cara tras recibir el golpe de un dron pilotado por un aficionado[6].
- **Amenaza para la privacidad:** Hoy en día existen drones, incluso de un tamaño muy reducido, al alcance del usuario medio y capaces de

sobrevolar zonas tomando fotografías y vídeos de una gran calidad. Patios, azoteas, colegios, recintos privados e incluso la seguridad del hogar se pueden ver vulneradas ahora debido a un mal uso de estas pequeñas aeronaves. Si, además, añadimos a esto los últimos avances en almacenamiento y procesamiento de imágenes, es evidente que la privacidad de los ciudadanos se encuentra ante un riesgo sin precedentes.

A día de hoy estos problemas, si bien no son despreciables, no han alcanzado ni mucho menos su máxima dimensión. Pero, en un momento en el que los drones se han convertido en el regalo favorito en cientos de miles de hogares y en la apuesta de muchas empresas de toda índole, cabe imaginarse un futuro en el que millones de estos aparatos surquen los cielos cada día. Los mecanismos actuales de localización y de prevención de colisiones para aeronaves no son aplicables para los drones actuales, debido principalmente a su reducido tamaño y a su gran maniobrabilidad. El control de este tipo de tráfico aéreo requerirá, por tanto, nueva y más sofisticada tecnología de la disponible actualmente.

1.2. Entorno social y económico

Así como la aviación tradicional tuvo sus raíces en la ciencia y la tecnología de surgió durante la era industrial, tras las transformaciones del siglo XIX y principios del XX, la aparición y la evolución de los drones tienen su origen en la Sociedad de la Información.

	2015
Infrastructure	45.2
Transport	13.0
Insurance	6.8
Media & Ent.	8.8
Telecommunication	6.3
Agriculture	32.4
Security	10.5
Mining	4.3
Total	127.3

FIGURA 1.3: Valor de las oportunidades de negocio generadas por los drones en miles de millones de dólares - Fuente: PwC

Su crecimiento y popularidad se deben a la confluencia de varios aspectos, entre los que podrían destacarse el cambio de estrategias militares a operaciones de inteligencia, reconocimiento y vigilancia, la proliferación de escenarios idóneos para su utilización y los grandes avances en miniaturización y autonomía de sistemas.

Como se ha comentado en la introducción, la revolución de los drones está transformando empresas e industrias de las más diversas índoles, desde la agricultura hasta el sector cinematográfico. De acuerdo con el informe realizado por PwC, *Clarity from above*[7], el incipiente mercado relacionado con el uso de los drones puede generar oportunidades de negocio por un valor total de más de 127.000 millones de dólares. Los sectores que, previsiblemente, más pueden beneficiarse del auge de estas aeronaves serían las infraestructuras, la agricultura y el transporte.

1.3. Objetivos

El objetivo principal de este proyecto es el estudio y el diseño de una posible solución a los desafíos que plantean la presente y futura irrupción en masa de los drones en el mercado.

Para cumplir este propósito, se diseñará e implementará un software que deberá cumplir otra serie de objetivos. Dicho software deberá ser capaz de:

1. Aprovechar de las ventajas de la computación distribuida y del Big Data, dado que trabajaría, previsiblemente, con una gran cantidad de datos provenientes de los drones en tiempo real. Para ello, dividir la vigilancia de distintas porciones del mapa entre los nodos de un clúster de máquinas.
2. Clasificar los drones según su finalidad de uso.
3. Delimitar zonas en las que sólo pueda circular un tipo concreto de dron.
4. Vigilar que los drones circulen por las zonas en las que lo tienen permitido, y avisar a las autoridades pertinentes en caso de que no lo hagan.
5. **Detectar** futuras colisiones entre los drones circulando por una misma porción de mapa, basándose en sus trayectorias. Ofrecer una solución para evitar dichas colisiones.
6. Diseñar e implementar una interfaz de usuario en la que:
   a) Pueda observarse en tiempo real el movimiento de los drones circulando por el mapa.
   b) Puedan distinguirse las distintas zonas que componen el mapa, así como diferenciarse los drones según su finalidad de uso.

c) Se vean delimitadas las porciones de mapa que controlará cada nodo del clúster que ejecute el software.

7. Abrir la puerta a futuras ampliaciones del proyecto.

1.4. Contenido de la memoria

Para ayudar en la lectura de este documento, se provee a continuación un breve resumen sobre cada uno de los capítulos que lo componen:

1.4.1. Capítulo 1: Introducción:

En la introducción se discute la motivación por la que ve la luz este proyecto y se explican brevemente los objetivos del mismo.

1.4.2. Capítulo 2: Estado del arte:

En este capítulo se presentarán los aspectos más importantes para comprender el sistema de vigilancia ADS-B, que es la base en la que se inspira este proyecto, así como el nacimiento del concepto de Big Data y los dos frameworks más importantes de Apache para tratar con este nuevo tipo de datos: Hadoop y Spark.

1.4.3. Capítulo 3: Diseño:

En el diseño se plantea el esquema de funcionamiento del sistema de control de drones y se describen brevemente los requisitos que éste debería cumplir.

1.4.4. Capítulo 4: Prototipo:

En este capítulo se describe el prototipo del sistema distribuido de control de drones que se ha programado. Para ello, se provee una descripción detallada de cada una de las clases que componen el programa.

1.4.5. Capítulo 5: Pruebas:

En este capítulo se describen las pruebas realizadas para comprobar el correcto funcionamiento del prototipo y se comentan los resultados.

1.4.6. Capítulo 6: Historia del proyecto:

En este capítulo se analizan los principales problemas que ha sufrido el proyecto, así como la solución a los mismos. Además, se presenta la planificación y el presupuesto del proyecto.

1.4.7. Capítulo 7: Conclusiones y trabajos futuros:

Este capítulo completa la memoria del proyecto, presentando las conclusiones que se extraen tanto del resultado como de la experiencia personal de realización del proyecto. Además, se comentan ligeramente sus posibilidades de mejora de cara al futuro.

1.4.8. Capítulo 8: Resumen extendido:

En este capítulo se presenta un resumen extendido de la memoria, destacando lo más importante en cuanto a la motivación y los objetivos del proyecto, el estado del arte, el diseño de una solución a los objetivos planteados y su prototipo, así como las conclusiones extraídas del proyecto.

1.4.9. Apéndice A: Introduction:

Ese apéndice consta de la traducción al inglés del capítulo de introducción al proyecto.

1.4.10. Apéndice B: Conclusions and future lines of work:

Este apéndice consta de la traducción al inglés del capítulo de conclusión del proyecto.

1.4.11. Apéndice C: Extended summary:

Este apéndice consta de la traducción del resumen extendido presentado en el capítulo 8.

1.4.12. Apéndice D: Código del proyecto:

Este apéndice indica la url donde puede encontrarse el código del prototipo.

Capítulo 2

Estado del arte

2.1. ADS-B

El 22 de diciembre del pasado 2015, la Organización de Aviación Civil Internacional (ICAO) informó de que el número total de pasajeros que volaron durante todo ese año se elevaba a 3.500.000 millones, un 6.4 % más que en 2014[8].

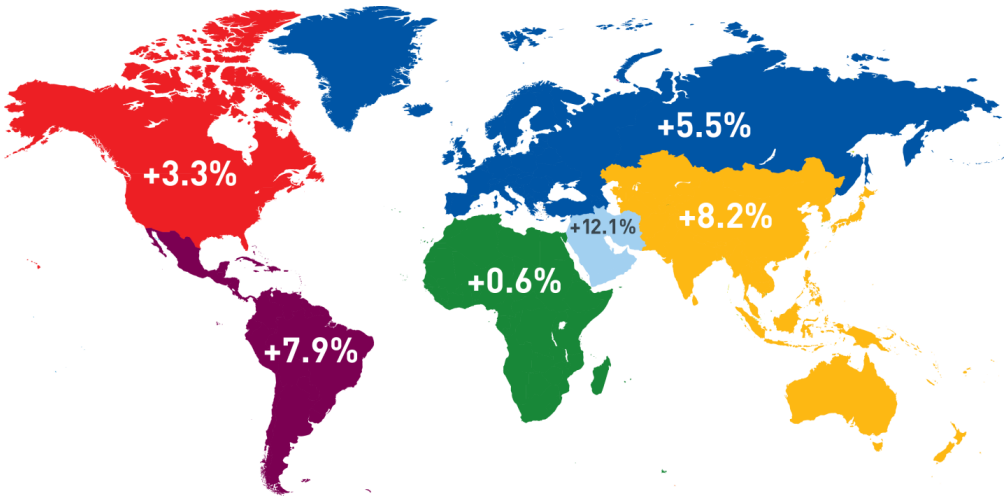

FIGURA 2.1: Aumento del tráfico aéreo de pasajeros en 2015

Se prevé que el tráfico aéreo continúe creciendo continuamente a lo largo de los próximo años. Existe, por tanto, una necesidad evidente de asegurar que los estándares de seguridad y de eficiencia se mantengan constantes o, incluso, aumenten. **ADS-B** llega para revolucionar la manera en la que se supervisa el espacio aéreo.

2.1.1. ¿Cómo funciona ADS-B?

Automatic Dependant Surveillance Broadcast (ADS-B) es un nuevo y preciso sistema de vigilancia basado en GPS llamado a sustituir a la tradicional vigilancia por radar.

Su filosofía es bastante sencilla: básicamente, cada aeronave transmite, de forma frecuente y regular a través de un transpondedor, parámetros como su posición o su velocidad a las estaciones de control en tierra y a otras aeronaves. Como su nombre indica, ADS-B tiene dos características principales: es **automática**, es decir, no requiere de la intervención del piloto para que los datos se transmitan, y es **dependiente**, en cuanto la información se genera en la propia aeronave y ésta depende del equipamiento de a bordo.

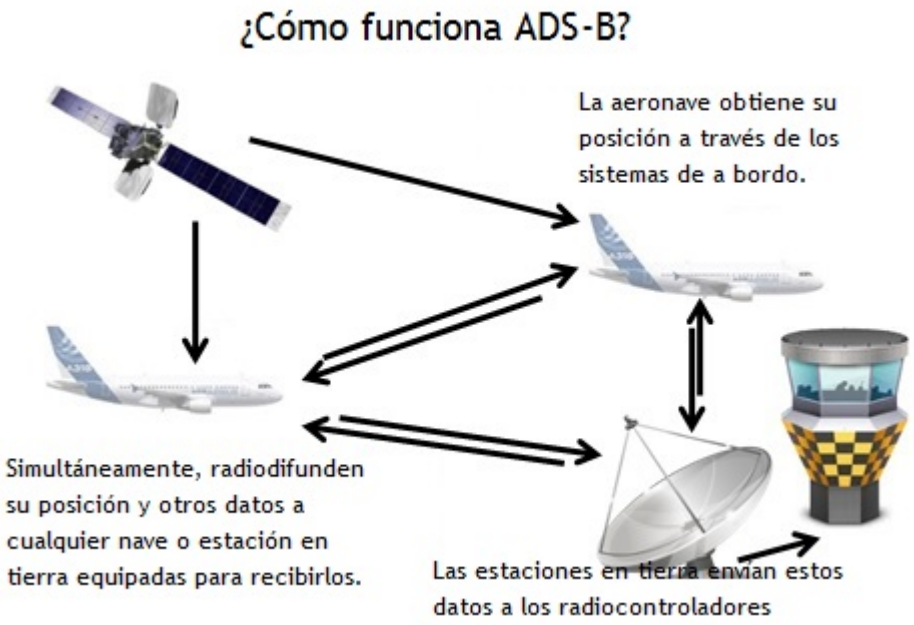

FIGURA 2.2: Funcionamiento esquemático de ADS-B

Desde la perspectiva de una aeronave, se pueden distinguir dos partes:

- **ADS-B OUT:** las señales ADS-B OUT son aquellas que se transmiten desde una aeronave a las estaciones en tierra y a las demás aeronaves que estén dentro de su rango de alcance. Esta señal puede incluir la posición de la aeronave, su velocidad, altitud, plan de vuelo y otros parámetros que son, normalmente, más precisos que aquellos obtenidos por radiodetección.
- **ADS-B IN:** las señales ADS-B IN son las señales entrantes, en una aeronave o en una estación de tierra, que proceden de otra aeronave. Para una aeronave, el ADS-B IN es opcional, aunque recomendable, pues supone para el piloto una fuente de información constante.

Los beneficios de ADS-B no se aplican solo al control del tráfico aéreo. Efectivamente, facilitará el trabajo de los controladores, pero también el piloto, la tripulación e incluso los pasajeros se verán beneficiados pues, entre otros, algunos de los beneficios que ofrecerá este nuevo sistema son:

- **Más seguridad**: como se ha mencionado previamente, los datos ofrecidos por ADS-B son sensiblemente más precisos que los obtenidos mediante un radar. Además, mientras el periodo de latencia de un radar varía generalmente entre los 4 y los 12 segundos, el transpondedor de una aeronave que utiliza ADS-B puede reportar su posición incluso más de una vez por segundo, ofreciendo información prácticamente en tiempo real.
- **Mayor capacidad en el espacio aéreo**: una consecuencia de la mayor precisión de ADS-B es que los estándares de separación que se aplican a día de hoy pueden ser reducidos, aumentando así el número de aeronaves que podrían surcar una determinada área del espacio aéreo. Uno de los principales objetivos de la **FAA** (Federal Aviation Agency) y la **EASA** (European Aviation Safety Agency) persigue que, mediante el uso de ADS-B, la capacidad del espacio aéreo se doble para el año 2030.
- **Mayor eficiencia**: ADS-B permitirá adoptar procedimientos más eficientes, al informar a la tripulación en cada momento del estado del tráfico o de las condiciones meteorológicas en los lugares donde se encuentren otras aeronaves, y al permitir a los pilotos de cada aeronave la comunicación directa entre ellos.
- **Mayor efectividad en costes**: existen dos modos en los que ADS-B permitiría ahorrar dinero:
  - Los primeros en beneficiarse del nuevo sistema serían los propios administradores. Las estaciones en tierra que reciben las señales de de los transpondedores de ADS-B son pequeñas y pueden ser instaladas en prácticamente cualquier lugar. Además, a diferencia de lo que ocurre con el radar, no necesitan de partes móviles, por lo que se ahorra también en mantenimiento y en la potencia necesaria para que operen. Por el precio de una estación de radar, se pueden comprar e instalar varias estaciones en tierra para ADS-B.
  - Un segundo y amplio grupo que se vería beneficiado económicamente sería el formado por las aerolíneas e, indirectamente, los pasajeros y los transportistas. Debido a la adopción de procedimientos más eficientes en el aire, se reducirán los tiempos de vuelo y el consumo de combustible. De hecho, según un estudio de Aireon, si la Región del Atlántico Norte utilizara vigilancia ADS-B, las aerolíneas podrían ahorrar hasta 125 millones de dólares en combustible. Esto se traduce también en una reducción en las emisiones de contaminantes, especialmente de dióxido de carbono y de los óxidos de nitrógeno.

ADS-B será obligatorio a partir de 2020 en la mayor parte del espacio aéreo americano controlado[9]. Concretamente, en:

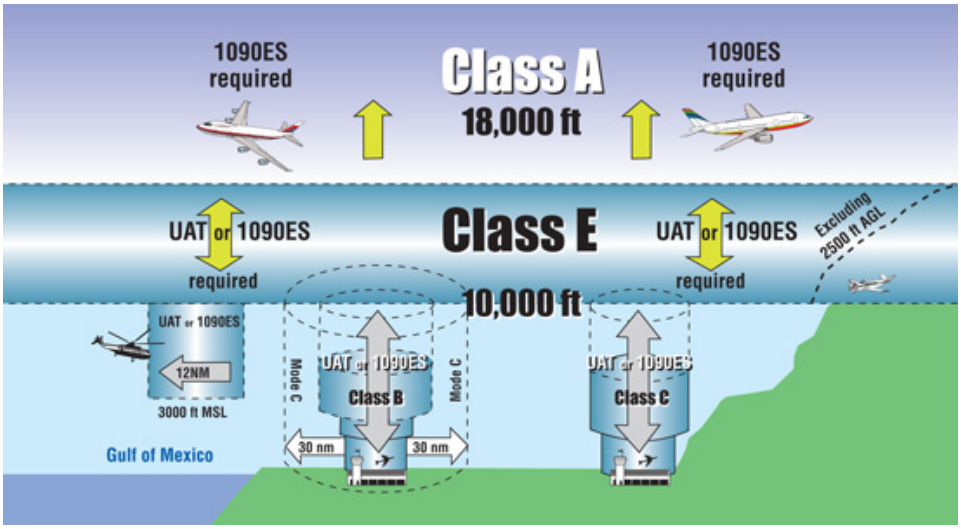

FIGURA 2.3: Zonas del espacio aéreo donde se requerirá ADS-B en Estados Unidos

- Espacio aéreo de clase A, B y C.
- Espacio aéreo de clase E, en aquellas áreas en las que la aeronave se encuentre sobrevolando cualquiera de los 48 estados americanos o el Distrito de Columbia a 10.000 pies o más **sobre el nivel del mar**, excluyendo el espacio aéreo a 2.500 pies o menos **sobre el nivel del terreno**.
- 30 millas náuticas, y desde el terreno hasta los 10.000 pies sobre el nivel del mar, en torno a ciertos aeropuertos muy transitados.
- Sobre el techo (y dentro de sus límites laterales) del espacio aéreo de clase B o C hasta los 10.000 pies de altura.
- Espacio aéreo de clase E sobre el Golfo de México, a 3.000 pies o más sobre el nivel de mar, a 12 millas náuticas de la costa de Estados Unidos o menos.

2.1.2. Equipamiento necesario

Existen dos dispositivos aprobados por la FAA para realizar la telecomunicación en ADS-B:

1. Un transpondedor modo S en la banda de 1090 MHz que posea señales espontáneas ampliadas (abreviado **1090ES**, por las siglas en inglés de *Extended Squitter*).
2. Un transceptor UAT (Universal Access Transceiver) en la banda de 978 MHz.

Estarán obligados a usar 1090ES todas las aeronaves que operen en el espacio aéreo de clase A (desde 18.000 pies sobre el nivel del mar hasta, incluyéndolo, el nivel de vuelo 600: 60.000 pies). Aquellas aeronaves que operen exclusivamente por debajo de 18.000 pies podrán usar tanto 1090ES como UAT.

2.1.3. ¿ADS-B en drones?

El mercado de los vehículos aéreos no tripulados o drones está en pleno apogeo. Las previsiones de crecimiento más optimistas de hace unos años se están quedando cortas. Se estima que el sector podría estar moviendo ya más de 10.000 millones de euros en todo el mundo y que esta cifra se triplicará para 2020. Además, según Isabel Maestre, directora general de la Agencia Estatal de la Seguridad Aérea, se prevé que el negocio de los drones representará el 10 % del mercado aeronáutico dentro de diez años.

FIGURA 2.4: En enero de 2015, el hallazgo de un dron en la Casa Blanca puso de manifiesto lo vulnerables que somos ante estas pequeñas aeronaves

Es un hecho, por tanto, que los drones están por todas partes y que prácticamente cualquier persona puede tener acceso a uno de ellos. Parece ser, también, que esta tendencia crecerá en el futuro, lo que no hace sino dar la razón a instituciones como la Asociación Internacional del Transporte Aéreo que, en febrero de este año, advirtió de que los drones civiles suponen cada vez más una 'amenaza real y creciente' para la seguridad de la aviación comercial[10]; o a la Agencia de Protección de Datos del Reino Unido, que el año pasado elaboró una guía sobre el uso de cámaras de vigilancia que contenía una sección exclusivamente dedicada a los drones[11]. Según esta guía, gracias a estas aeronaves, la vigilancia es mucho más efectiva a costa de ser potencialmente mucho más peligrosa para la privacidad.

Si a todo ésto le sumamos la intención, por parte de empresas como UPS, DHL o Amazon, de utilizar drones para repartir el correo en un futuro,

es evidente que la regulación y la vigilancia sobre los drones va a tener que ser más estricta en un futuro.

Hasta hace muy poco, no era plantable utilizar el sistema ADS-B en drones puesto que, para la gran mayoría, los transpondedores y las antenas utilizadas en este sistema son demasiado grandes y pesados. Sin embargo, en abril de este año, la empresa de electrónica uAvionix presentó el receptor ADS-B más pequeño y ligero jamás fabricado[12]. El llamado pingRX pesa 1.5 gramos y requiere de una centésima parte de la potencia que requiere un receptor ADS-B normal. Además, sus medidas son de tan sólo 32 mm x 15 mm x 3 mm.

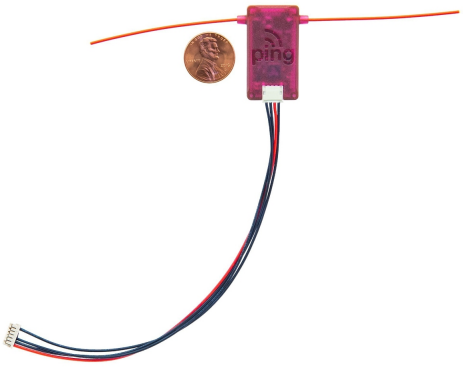

FIGURA 2.5: Receptor ADS-B pingRX

El pingRX, que es capaz de recibir información de otras aeronaves en las dos bandas aprobadas por la FAA (879 MHz y 1090 MHz) abre un nuevo escenario en el que cabe esperar la llegada de transmisores de similares dimensiones mediante los cuales, bien se podría extender el sistema de vigilancia ADS-B a los vehículos aéreos no tripulados, o bien podría desarrollarse otro sistema sincronizado con ADS-B, pero con un funcionamiento exclusivo y adaptado a los drones, para vigilarlos y para evitar posibles choques entre ellos. Un sistema de este tipo debería estar preparado para procesar, prácticamente en tiempo real, una gran cantidad de datos procedentes de un elevado número de drones. Dicho de otro modo: un sistema de estas características debería ser capaz de trabajar con Big Data.

2.2. Big Data

La cantidad de datos que se generan y se almacenan actualmente en el mundo es casi inconcebible. Cada año, los físicos que trabajan con el Gran

Colisionador de Hadrones del CERN deben filtrar los 30 petabytes (30 millones de gigabytes) que produce el acelerador de partículas y tratar de obtener información útil de estos datos [13]. Esta cifra, que parece increíble, se queda casi pequeña si hablamos de las de las más de 4 millones de publicaciones a las que los usuarios de Facebook dan un *"me gusta"* cada minuto o las 300 horas de vídeo que se suben a Youtube, **también cada 60 segundos**[14].

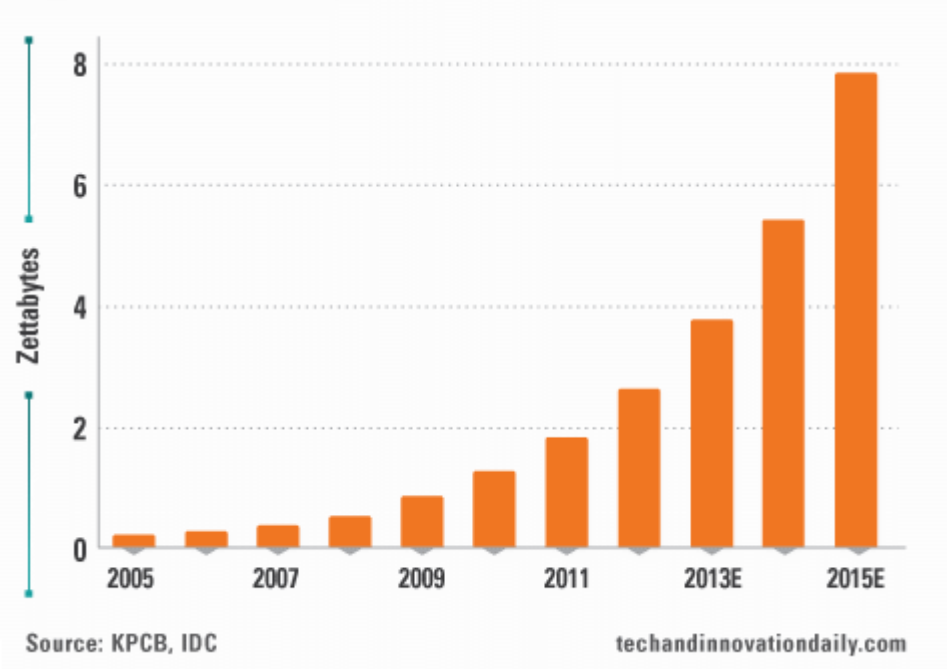

FIGURA 2.6: Crecimiento de la generación de datos en el mundo

Este crecimiento exponencial en la generación de datos ha sido acompañado también por un gran progreso en los sistemas y dispositivos capaces de almacenarlos; sin embargo, las técnicas utilizadas para extraer información útil de estos datos no se han desarrollado al mismo ritmo. Debido a esto y al bajo coste de almacenamiento, existen lo que muchos expertos denominan *tumbas de datos*, lugares donde residen grandes cantidades de potencial información sin explotar. Es en este contexto donde nace el concepto de **Big Data**.

2.2.1. ¿Qué es Big Data?

Lo primero que se debería saber es que Big Data es un término relativamente nuevo y en constante evolución que no tiene una sola definición. Sin embargo, se podría afirmar que es, al menos, cualquier conjunto de datos -tanto estructurados como semi-estructurados o sin estructura- con potencial de ser aprovechado para la extracción de información y cuyo volumen

o complejidad es tal que dicha extracción no puede abarcarse con los métodos tradicionales de análisis. El término se hizo popular a principios de este siglo, cuando el analista Doug Laney caracterizó Big Data mediante las 3 V's[15]:

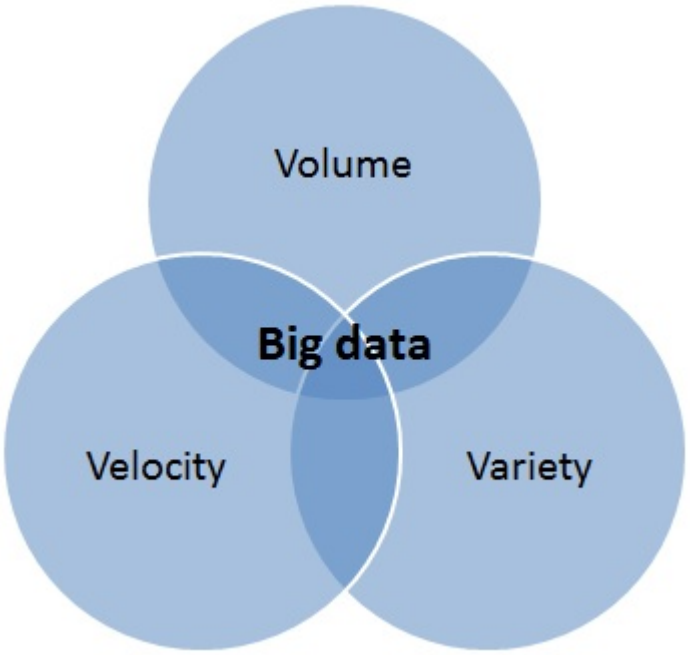

FIGURA 2.7: Las tres V's de Big Data

1. **Volumen:** Hace referencia al tamaño de los sets de datos a manejar. A día de hoy, tenemos más fuentes de información que nunca antes en el mundo: desde teléfonos móviles hasta ordenadores, sensores, equipos médicos y otros muchos dispositivos que, en definitiva, generan una enorme cantidad de datos.

2. **Velocidad:** Se refiere tanto a la frecuencia con la que se generan nuevos datos como, sobre todo, a la necesidad de dar respuesta a la información con rapidez. Hace años, las noticias de ayer se consideraban información reciente. Hoy, en las redes sociales, los mensajes de hace unos pocos minutos pueden no ser de interés para los usuarios, que los descartan para prestar atención a las nuevas actualizaciones. El movimiento de datos es prácticamente en tiempo real, y así debe ser su análisis.

3. **Variedad:** Se refiere a la naturaleza de la información a manejar. Muchas veces los datos no poseen formatos tradicionales ni están estructurados, sino que son recogidos en forma de imágenes, vídeos, pdf, SMS... El mundo real ofrece datos en innumerables formatos, y es el reto del Big Data el organizarlos y darles sentido.

Por tanto, aunque es tentador pensar en Big Data como simplemente una gran cantidad de datos tradicionales, es un error. Es más que eso. Big Data hace referencia a una gran cantidad de datos -en su mayoría desestructurados- que son generados de manera continua desde fuentes de información que hasta ahora no era posible explotar.

Es bastante evidente que esto abre un nuevo y gran abanico de posibilidades que tienen el potencial de transformar casi cualquier negocio. Sin embargo, como se ha comentado con anterioridad, las técnicas tradicionales de procesamiento y análisis de datos no son aplicables en la práctica a Big Data, serían demasiado costosas tanto en tiempo, como en equipamiento y dinero. Por tanto, ha tenido que nacer una nueva forma de interactuar con estos datos. Es en este momento cuando **MapReduce** ve la luz.

2.2.2. ¿Qué es MapReduce?

En 2004, para ayudar a digerir la gran cantidad de datos que producía diariamente, Google decidió que era hora de aprovechar la potencia de la computación distribuida. Nace entonces un conjunto de tecnologías y un paradigma de programación llamado MapReduce. De manera simplista, se podría decir que MapReduce aplica el viejo concepto de *"divide y vencerás"*: es mucho más rápido dividir una tarea grande en varios pedazos más pequeños y procesarlos **en paralelo**.

El término describe, en realidad, dos tareas diferentes y secuenciales: en primer lugar se realiza un mapeo (*Map*), en el cual se transforma un conjunto de datos en otro conjunto en el que los elementos individuales se dividen en tuplas (pares clave/valor). Posteriormente, se realiza una función de reducción (*Reduce*) que toma como entrada la salida del mapeo y convierte el conjunto de tuplas en un conjunto de tuplas más pequeño.

Como analogía, se podría pensar en la forma en la que se calculaba el censo de población en tiempos del Imperio Romano. En cada ciudad, se nombraba una persona encargada de contar el número de habitantes de dicha ciudad y, después, enviar el resultado a la capital. Entonces, estos resultados se reducían a un único número (la suma de los habitantes de todas las ciudades). Este mapeo de personas a ciudades, en paralelo, para después combinar los resultados (reducción), es mucho más eficiente que enviar una sola persona a contar todos los habitantes del imperio.

Del mismo modo, por primera vez un grupo de servidores básicos, cada uno de ellos diferente e independiente, podían trabajar en conjunto para un mismo fin, formando virtualmente un servidor de mucha mayor potencia.

Junto a MapReduce, Google desarrolló el *Google File System*, un innovador sistema de archivos distribuidos orientado a ser capaz de almacenar enormes cantidades de datos. Este sistema fue optimizado para satisfacer las necesidades de procesamiento de información de Google pero, como se

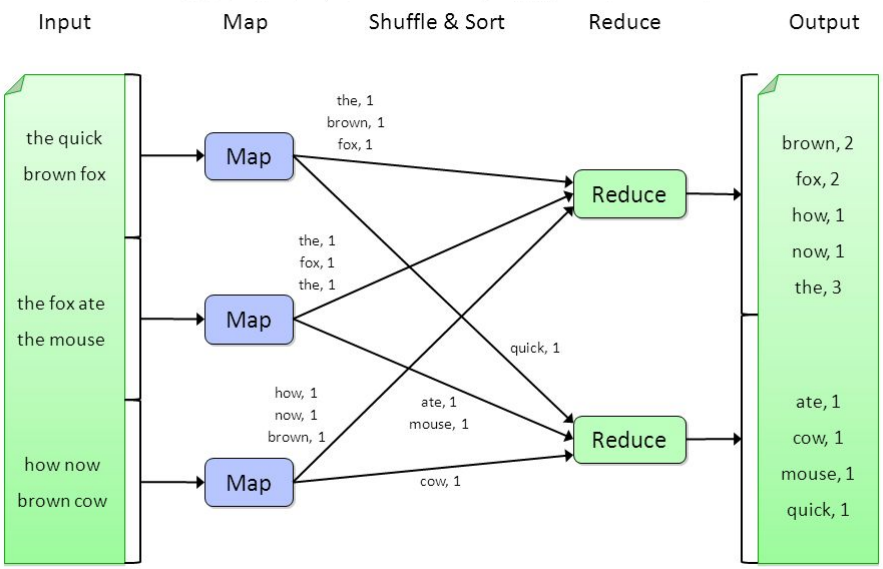

FIGURA 2.8: Diagrama del funcionamiento de MapReduce para un contador de palabras

describirá más adelante, sirvió de base para el nacimiento del sistema de archivos distribuidos de Hadoop (HDFS).

2.2.3. ¿Qué es Hadoop?

En realidad, el Hadoop original es un elefante de juguete. Más concretamente, el elefante de juguete del hijo de Doug Cutting. Doug, el creador de Hadoop, le dio ese nombre a su nueva iniciativa[16].

FIGURA 2.9: Logotipo de Hadoop

Apache Hadoop es un framework **de código abierto**, cimentado en los documentos de MapReduce y *Google File System*, que permite el almacenamiento y el procesamiento en paralelo de enormes cantidades de datos distribuidos en un conjunto de servidores. De este modo, Hadoop permite a cualquier usuario disfrutar de las bondades de MapReduce que, si bien supuso una revolución y una solución al problema de Big Data, originalmente requería de unos recursos tecnológicos y una estructura al alcance de muy pocos, aparte de Google.

De forma simplificada, Hadoop tiene dos partes principales: un framework que procesa los datos y un sistema de archivos distribuidos para almacenarlos (por defecto, el HDFS). En una base de datos normal, los usuarios pueden recoger y analizar los datos usando consultas (*queries*) basadas en los estándares SQL (*Structured Query Language*). Incluso en las bases de datos NoSQL se utilizan consultas, aunque éstas no tengan que estar sujetas al estándar. Pero Hadoop no es una base de datos: en lugar de consultas, utiliza MapReduce, que lanza una serie de aplicaciones -escritas en Java- mediante las cuales se recorren los datos y se extrae información de ellos.

Cuando se pregunta por información, el mapeo es realizado por el llamado *JobTracker*, que divide las tareas entre los diferentes nodos del clúster, denominados *TaskTrackers*. Una vez las tareas han sido terminadas, los conjuntos de datos resultante se reducen y se devuelve al nodo central.

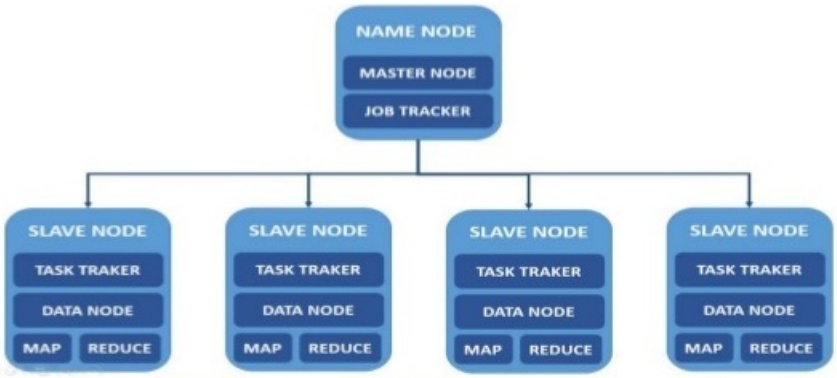

FIGURA 2.10: Arquitectura de Hadoop

El sistema de archivos HDFS funciona de forma similar. Existe un solo *NameNode*, que indica en qué máquinas (*DataNodes*) se almacenan los datos. Estos datos, divididos en bloques -generalmente de 128MB-, se replican de manera que estén distribuidos en diferentes *DataNodes*. En ésto reside una de las grandes fortalezas de Hadoop: añade redundancia de modo que, si uno de los nodos del clúster se cae, el sistema pueda ser funcionando con normalidad.

Pero no es su único punto fuerte, son numerosas las ventajas de este sistema. Por ejemplo: los datos y el software que se necesita para procesarlos se encuentran en la misma máquina, lo que acelera mucho las peticiones de información. Además, este hecho produce también que, cada vez se añade una máquina al clúster, el sistema gana tanto la capacidad del disco duro de esta nueva máquina, como su potencia de computación.

Por último, cabe destacar que Hadoop es completamente modular, lo que significa que puedes intercambiar cualquiera de sus componentes por

una herramienta de software distinta. Todo esto y muchas otras cosas convierten a Hadoop en una herramienta robusta, eficiente y, lo más importante, que pone la ciencia de los datos al alcance de todos.

2.2.4. ¿Qué es Spark?

Spark es la otra gran herramienta de Big Data de Apache Software Foundation. Su principal fortaleza reside en que, a diferencia de Hadoop, que tiende a escribir en disco los resultados de cada parte del procesamiento de datos, Spark fue optimizado para ejecutarse en memoria, lo que ayuda a obtener resultados mucho más rápidamente que sus competidores. Sus defensores claman que Spark puede ser, de hecho, hasta 100 veces más rápido cuando se ejecuta en memoria y hasta 10 veces más rápido cuando se procesa en disco[17], de manera similar a como lo hace el MapReduce de Hadoop.

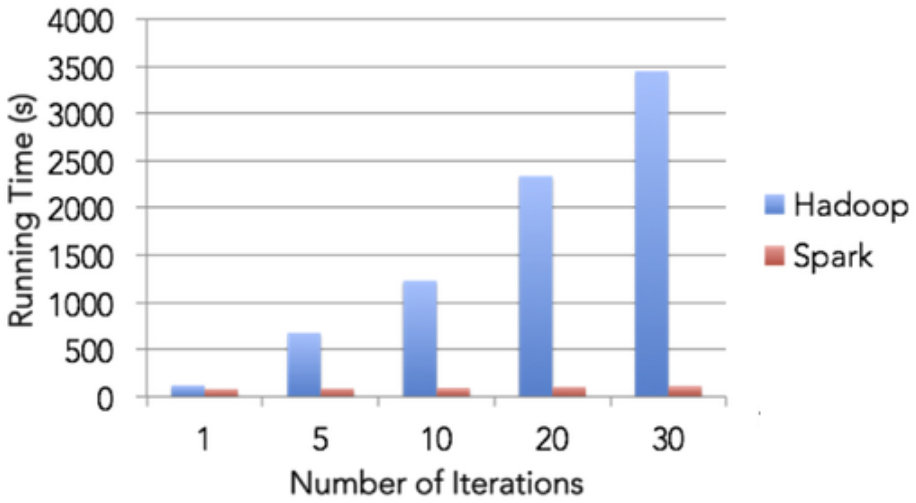

FIGURA 2.11: Comparación de la velocidad de Spark sobre Hadoop. Imagen de Cloudera

Muchos afirman que no debe ser visto como un competidor de Hadoop, sino como un complemento. Es cierto que Hadoop y Spark realizan trabajos distintos: mientras el primero funciona como una estructura de datos distribuida, Spark **no realiza funciones de distribución de almacenamiento**, sino que se trata simplemente de una herramienta de **procesamiento** de datos.

Pese a ello, es cierto que Spark es notablemente más rápido que Hadoop. En 2014, se utilizó Spark para ganar el concurso de Daytona Gray Sort Benchmark, procesando 100 terabytes de datos mediante un clúster de 206 máquinas en tan sólo 23 minutos, superando por mucho los 72 minutos del anterior ganador, que usó Hadoop y un clúster de 2.100 máquinas[18]. Esto, unido a otras ventajas como su simplicidad de uso o su más amplio

rango de lenguajes de programación aceptados -Java, Python, R, Scala...- han convertido a Spark en el proyecto de Big Data más activo en Apache.

Cabe preguntarse: ¿por qué ésto es así? ¿por qué es Spark tan rápido comparado con Hadoop? La lentitud en el intercambio de datos usando MapReduce se debe a la **replicación** y la **serialización** de los datos, así como las continuas operaciones de **lectura y escritura en disco**. Reconociendo este problema, los desarrolladores de Spark construyeron este nuevo framework alrededor del concepto de Resilient Distributed Datasets (**RDD**), que residen en memoria. En general tanto Hadoop como Spark pueden usarse para procesamiento por lotes: el primero en dos etapas (los ya descritos *map* y *reduce*) y el segundo en tantas como se necesiten (disponiendo además de una amplia caché). Pero, además, Spark no solo es capaz de procesar datos en lotes sino que, gracias a *Spark Streaming*, permite a su vez manejar datos en tiempo real al igual que hacen otras soluciones especializadas, como *Storm*. Esto puede hacerlo con o sin Hadoop y puede además convivir con otras soluciones.

2.2.5. Resilient Distributed Datasets

El documento original en el que nació el concepto de RDD es *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*, de Matei Zaharia et al[19]. Según los propios autores, la motivación tras los RDD era manejar dos tipos de aplicaciones que los frameworks existentes hasta el momento no utilizaban eficientemente:

- Algoritmos iterativos en aprendizaje máquina y computación de gráficos.
- Herramientas interactivas de minería de datos.

Formalmente, un **RDD** es un conjunto de elementos particionado, distribuido e inmutable (*read-only*). Directamente del propio nombre pueden sacarse varias conclusiones[20]:

- **Resilient** (elástico, resistente): RDD es tolerante a fallos, es capaz de recomponer partes perdidas o dañadas debido al fallo de algún nodo.
- **Distributed** (distribuido): los datos residen en varios nodos dentro de un clúster.
- **Dataset** (conjunto de datos): es una colección de datos (primitivos, tuplas, objetos...)

Además de estas características, incluidas en el nombre de RDD, se pueden destacar otras como:

- **En memoria**: los datos de un RDD se mantienen en memoria tanto (en tamaño y en tiempo) como sea posible.
- **Read-only**: un RDD es inmutable, no puede ser cambiado tras su creación, sólo transformado a un nuevo RDD.
- **Evaluación perezosa**: con un RDD estamos definiendo un flujo de información, pero no se ejecuta en el momento de la definición, sino cuando aplicamos una acción sobre éste.
- **Cacheable**: puedes almacenar todos los datos tanto en memoria o caché (por defecto) como en disco (no es la opción preferida, por tiempo de acceso).
- **Paralelización**: los datos son procesados en paralelo por los diferentes nodos.
- **Particionado**: los datos de un RDD están particionados y distribuidos entre los nodos del clúster.

De este modo, cualquier trabajo en Spark se expresa mediante las siguientes actividades:

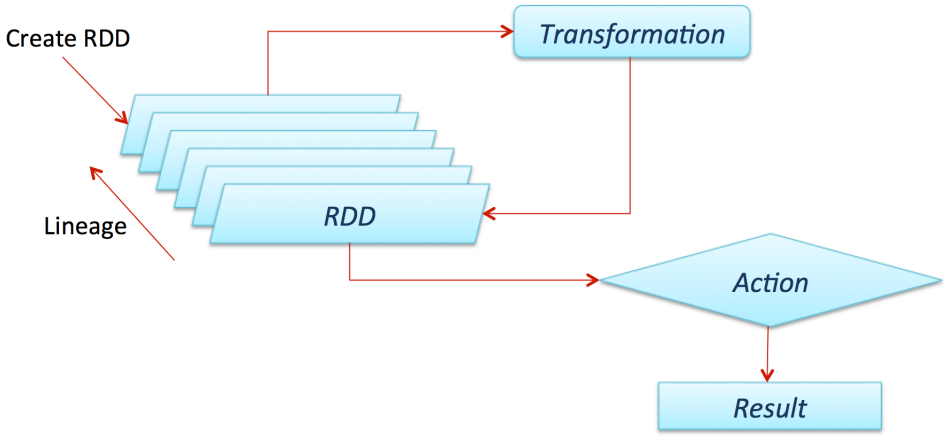

FIGURA 2.12: Diferencia entre transformaciones y acciones

- Creando RDDs: existen principalmente dos modos a través de los cuales se crea un RDD (aparte de las transformaciones, que se verán más adelante):
  - **Paralelizando** una colección en memoria (por ejemplo, una lista de números).
  - Referenciando datos en una **fuente de almacenamiento externa** como, por ejemplo, HDFS.
- Realizando transformaciones sobre un RDD para obtener otro nuevo. Algunas de estas transformaciones son:

- map(*function*): crea un nuevo RDD, resultado de aplicar la función *function* a cada elemento del RDD de entrada.
- filter(*function*): crea un nuevo RDD en el que se incluyen sólo los elementos del RDD de entrada que satisfagan la función *function*, la cual devuelve un booleano.

- Llevando a cabo operaciones (acciones) sobre los RDDs para obtener un resultado. Algunos ejemplos de acciones son:
  - count(): devuelve el número de elementos en el RDD.
  - take(*n*) devuelve un array con los primeros *n* elementos del RDD.
  - collect(): devuelve un array con todos los elementos del RDD.
  - saveAsTextFile(*fichero*): guarda el RDD en el fichero de salida *fichero*.

2.3. Teledetección

En 1858 el francés Gaspard-Félix Tournachon, más conocido como **Nadar**, mandó construir un globo aerostático llamado *El Gigante*, subió a él con su cámara fotográfica y realizó la primera fotografía aérea de la historia sobre los tejados de las casas del pueblo Petit-Becetre, en Francia. Pese a que la imagen no se conserva, este hecho revolucionó el mundo de la fotografía, acabaría revolucionando el ámbito militar -especialmente en la Primera Guerra Mundial- y se convirtió, probablemente, en el primer avance significativo en la disciplina que hoy conocemos como **teledetección**.

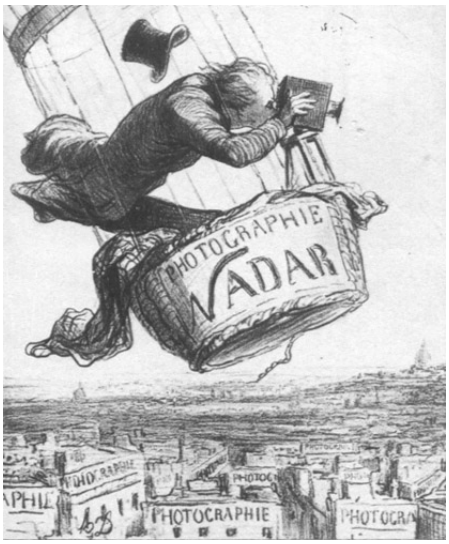

FIGURA 2.13: Litografía de Daumier. *"Nadar eleva la Fotografía a la altura del arte"*

2.3.1. ¿Qué es la teledetección?

A grandes rasgos, la teledetección es la adquisición de información acerca de un objeto o fenómeno mediante instrumentos de grabación o de escaneo en tiempo real a distancia, es decir, sin estar los instrumentos en contacto con el objeto.

Las técnicas de teledetección actuales pueden ser divididas en dos grupos: **teledetección pasiva y teledetección activa**:

- Los teledetectores pasivos detectan la radiación **natural** emitida o reflejada por el objeto observado (la luz solar reflejada es el ejemplo más común). Ejemplos de este tipo de teledetección son la fotografía o los infrarrojos.
- Los teledetectores activos, sin embargo, **emiten** energía hacia los objetos o áreas observadas y miden después la radiación reflejada por éstos. El radar, que utiliza ondas de radio, o el lidar, que utiliza lásers (haces de luz), son algunos de los ejemplos más conocidos de teledetección activa.

2.3.2. LIDAR

El **lidar** (Laser Imaging Detection And Ranging) es una tecnología de teledetección activa que funciona mediante la emisión de un haz láser pulsado hacia un objeto o entorno y la medición del tiempo que tarda en volver el reflejo hacia el emisor.

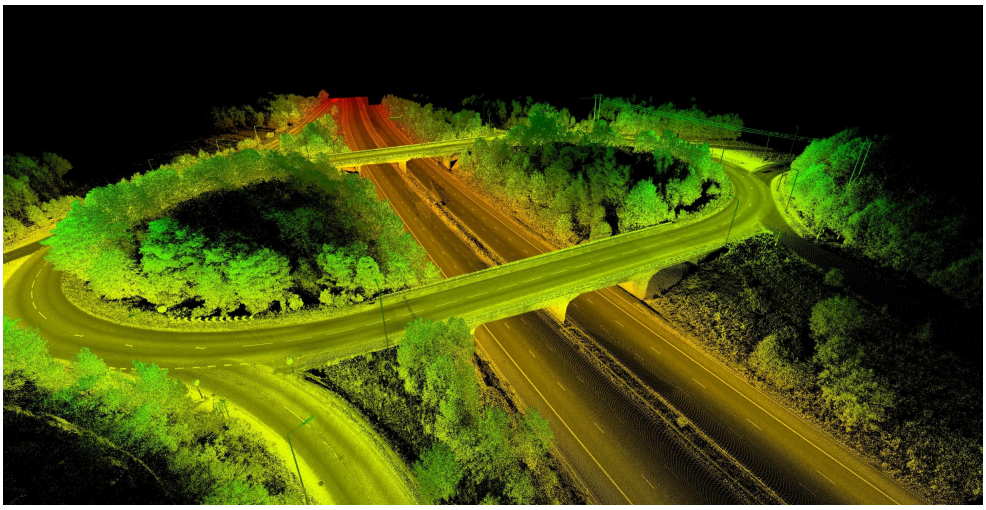

FIGURA 2.14: Ejemplo de imagen generada por lidar

Al igual que el radar, el lidar puede funcionar por la noche, bajo la lluvia, en cualquier posición en la que se encuentre fijo y en modo automático

o manual. Además es, en general, más económico, más fácil de transportar, manejar y mantener y notablemente más rápido que el radar.

El uso más común, extendido y apreciado del lidar a día de hoy es el orientado a la topografía, pues permite examinar el entorno con precisión y flexibilidad, ya que es capaz de sortear obstáculos como, por ejemplo, árboles. Sin embargo, otros usos del lidar son:[21]

- **Detección de velocidades:** por ejemplo, en las pistolas láser que los policías usan para calcular la velocidad de los vehículos.
- **Óptica adaptativa:** para corregir las perturbaciones más importantes que sufren las imágenes astronómicas debido a la atmósfera terrestre, lo cual es muy interesante porque equivale a observar el espacio **desde** el espacio.
- **Gestión forestal:** en la lucha contra incendios, la disponibilidad de un modelo preciso del tipo de combustible presente en cada punto del terreno es esencial para ser capaces de predecir el comportamiento del fuego con exactitud y poder así tomar decisiones para combatir el fuego.
- **Vehículos autónomos:** el lidar permite a muchos vehículos autónomos detectar y esquivar obstáculos en su entorno.
- **Geología, minería, Usos militares, vigilancia, etc.**

2.4. Marco regulatorio acerca de los drones

En julio de 2014, el Consejo de Ministros aprobó la primera regulación sobre la explotación de los drones con el *Real Decreto-ley 8/2014, de 4 de julio, de aprobación de medidas urgentes para el crecimiento, la competitividad y la eficiencia*.

Es importante aclarar que la ley sólo considera drones a las aeronaves no tripuladas que no superen en despegue un peso de 150 kilogramos. Para pesos superiores, el organismo regulador es la EASA (European Aviation Safety Agency) y se aplica una regulación especial más compleja en la que intervienen otras cuestiones relacionadas con el tráfico aéreo. Además, la regulación actual sólo se aplica a aquellos drones destinados a un uso profesional. Si el uso del dron es recreativo, no existe una regulación, aunque sí una serie de recomendaciones que hace la AESA (Agencia Estatal de Seguridad Aérea) hasta que llegue la anunciada ley para drones recreativos.

La ley distingue entre tres categorías de drones según su peso[22]:

- **Drones de peso inferior a 2kg:**
    1. No será necesario la inscripción en el registro de aeronaves ni el certificado de aeronavegabilidad.

2. La estación de control delimita la permitividad respecto al alcance visual y la distancia máxima de vuelo.
3. Altura máxima: 120 metros.
4. Necesario notificar el uso del dron.

- **Drones de peso inferior a 25kg:**

1. Necesaria la inscripción en el registro de aeronaves, el certificado de aeronavegabilidad además del carnet de piloto de drones.
2. El alcance visual permitido es el rango de visión del piloto, con un máximo de 500 metros.
3. Altura máxima: 120 metros.
4. Se requiere de una declaración de responsabilidad para hacer uso de la aeronave.

- **Drones de peso superior a 25kg:**

1. Necesaria la inscripción en el registro de aeronaves, el certificado de aeronavegabilidad además del carnet de piloto de drones.
2. El certificado de aeronavegabilidad determina la permitividad respecto al alcance visual y la distancia y altura máxima de vuelo.
3. Necesaria placa de identificación y matrícula del vehículo.
4. Se requiere registro de la del vehículo, así como una autorización de la AESA para poder volar la aeronave.

- **Para todos los drones:**

1. El dron requiere de una placa de identificación que incluya los datos del fabricante y de la empresa que esté operando con él.
2. Queda prohibido sobrevolar núcleos densamente poblados sin autorización expresa de la AESA.

Capítulo 3

Diseño

A continuación se presenta el diseño de una posible solución a los desafíos que plantean la presente y futura irrupción en masa de los drones en el mercado. Dicho diseño está basado en la tecnología de vigilancia ADS-B, descrita en el apartado 2.1. Por simplicidad, y ya que la altura a la que pueden volar los drones convencionales está muy limitada, se ha planteado una solución en dos dimensiones, en el plano cartesiano, despreciando la componente z, que equivaldría a la altura.

3.1. Arquitectura de red

No existe ningún motivo por el que la solución que se va a explicar en este capítulo esté limitada a funcionar con una arquitectura de red concreta. Dos posibilidades a tener en cuenta y que no están reñidas con la solución serían las siguientes:

- **Arquitectura A:** en este modelo el dron comunicaría sus parámetros únicamente a su piloto, como es habitual. Sería el piloto el encargado de enviar, mediante una red de comunicaciones, los parámetros de su aeronave al sistema de control, donde estaría el clúster de máquinas ejecutando el programa de vigilancia.

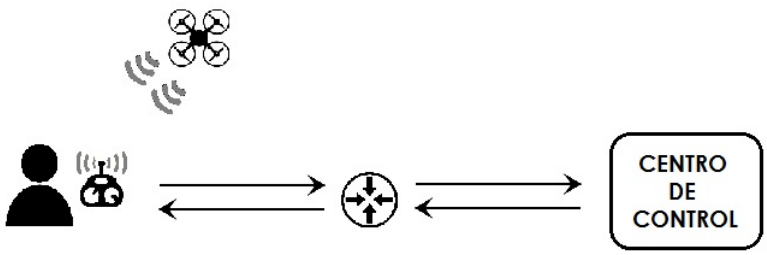

FIGURA 3.1: Arquitectura de red A

- **Arquitectura B:** en este modelo, el dron radiodifunde los datos tanto al piloto como al sistema de control. A diferencia de lo que ocurre con la arquitectura A, no es posible llevar este modelo a la práctica a día de hoy. Esto se debe, como se explicó en el apartado 2.1.3, a que no existen todavía transpondedores adecuados al tamaño y peso de un dron. Sin embargo, es digna de mención ya que es previsible que se convierta en una solución factible en un futuro cercano. Además, es la arquitectura óptima entre las dos, en cuanto a que no requiere de participación humana: los datos de cada dron son radiodifundidos al centro de control independientemente de la decisión del piloto.

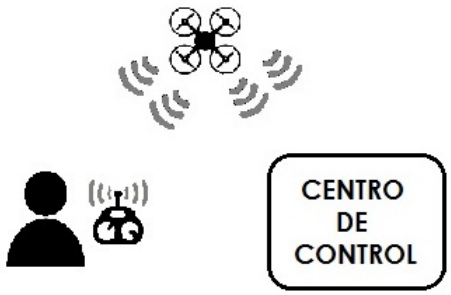

FIGURA 3.2: Arquitectura de red B

3.2. Formato de los datos

Puesto que no existe un estándar a la hora de definir los datos que describen el estado de un dron, en este diseño se ha asumido que dichos datos son enviados en forma de texto plano con el siguiente formato:

```
id, posX, posY, vel, destX, destY, tipo, priority
```

Donde:

- **`id:`** es el identificador del dron.
- **`posX:`** es la coordenada X de la posición actual del dron.
- **`posY:`** es la coordenada Y de la posición actual del dron.
- **`vel:`** es la velocidad del dron, en unidades por segundo.
- **`destX:`** es la coordenada X del punto al que se dirige el dron.
- **`destY:`** es la coordenada Y del punto al que se dirige el dron.

- **`tipo:`** es un entero que indica el tipo del dron. El número 1 significa que el dron es de ocio, el número 2 que es comercial y el número 3 indica que el dron ha sido detectado en una zona prohibida.
- **`tipo:`** es un entero que simboliza la prioridad que tiene un dron frente a otros a la hora de mantener su posición y/o velocidad cuando se ve involucrado en una colisión.

Los parámetros `destX` y `destY` son utilizados para calcular la orientación del dron y, mediante ella, las componentes de su velocidad. La existencia de estos parámetros es justificable si asumimos que los drones funcionan siguiendo un vuelo programado, correspondiendo dichos parámetros al siguiente punto de control al que se dirige el dron. Es razonable pensar que los drones comerciales funcionen de esta manera. Del mismo modo, resulta sensato asumir que los drones destinados a un uso recreativo no sigan un vuelo programado, sino que el piloto lo teledirija en tiempo real. En cualquier caso, no supone un gran problema, puesto que los drones que no sigan un vuelo programado deben comunicar de algún modo su orientación y/o las componentes de su velocidad, de modo que el programa contaría directamente con estos parámetros.

3.3. Estructura del mapa

De ahora en adelante, se llamará **mapa** a la totalidad del espacio bidimensional que vigila el sistema de control de drones. Un mapa está compuesto por muchas secciones más pequeñas denominadas **planos**, cada uno de los cuales está controlado por un nodo del sistema, de modo que cada nodo ejecuta el programa sobre los drones que se encuentren en su plano correspondiente.

Cada plano está dividido a su vez en pequeñas **celdas** de 1 píxel x 1 píxel. Estas celdas se usarán para prever posibles choques, puesto que el programa alertará si dos o más drones van a encontrarse en la misma celda en un mismo instante de tiempo.

3.4. División de un mapa por zonas

Ya que no todos los drones ni las motivaciones para su uso son iguales, en este diseño se plantea una diferenciación de las zonas en las que pueden circular. Así, se distinguirían tres zonas:

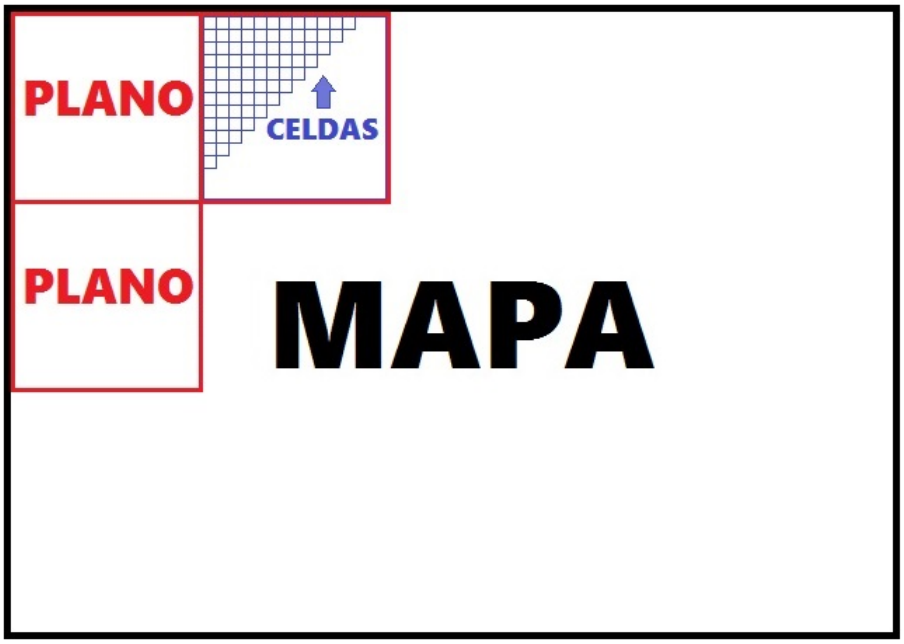

FIGURA 3.3: Diferencia entre mapa, plano y celda

3.4.1. Zona comercial

En las zonas comerciales podrían circular los drones cuyo uso esté exclusivamente destinado a servicios comerciales o de transporte. Algunos ejemplos de estas aeronaves son las utilizadas por la compañía de comercio electrónico Amazon o la empresa de paquetería alemana DHL, que recientemente han realizado varias pruebas para el reparto de paquetes utilizando drones autónomos.

Para este tipo de usos, y más a gran escala, los drones utilizados suelen ser autónomos y guiados por GPS. Una de las funciones del sistema informático encargado de vigilar el vuelo de estas aeronaves sería analizar sus trayectorias para evitar choques entre ellas. Además, tendría que vigilar que el dron se mantenga dentro de la zona destinada al uso comercial e, incluso, estrechar la vigilancia sobre él si, por su trayectoria, parece que podría abandonar la zona en el futuro más próximo.

De este modo, los drones autorizados a volar en zonas comerciales podrían ser identificados mediante, por ejemplo, una etiqueta que así lo indicara.

3.4.2. Zona de ocio

En las zonas de ocio podrían volar todos aquellos drones dirigidos por aficionados, cuyo único uso sea recreativo.

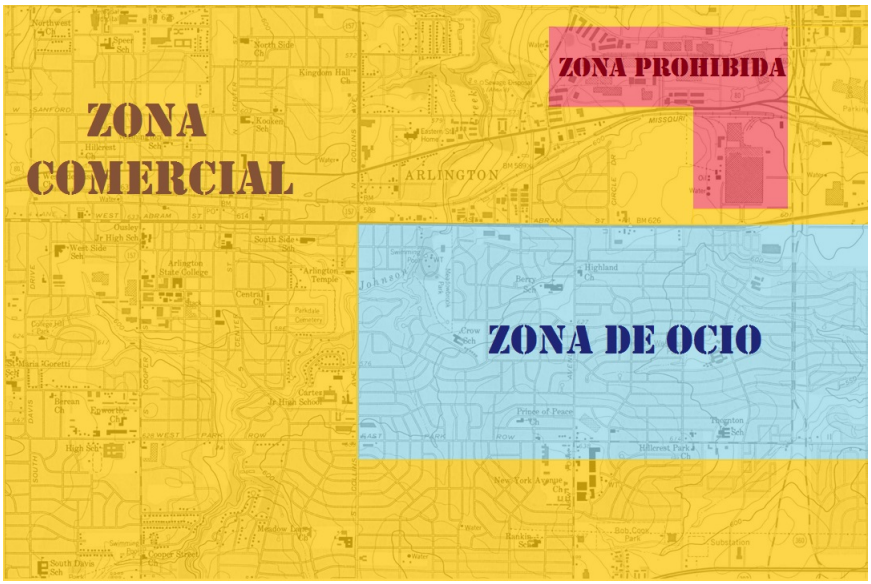

FIGURA 3.4: Ejemplo de división de un mapa por zonas

La normativa actual en la mayoría de los países que contemplan el vuelo de estos drones exige a los pilotos que nunca pierdan de vista el aparato, así como que éste circule debajo de una altura que varía entre los 50 y los 200 metros. Así pues, en este diseño no se contempla la previsión de posibles choques entre drones de ocio. Sí se analizará la posición y la trayectoria de estos drones para detectar o prever su salida de la zona delimitada para uso lúdico.

De la misma manera que en la zona comercial, un dron autorizado para volar en una zona de ocio podría ser identificado como tal por el programa mediante una etiqueta.

3.4.3. Zona prohibida

Las zonas prohibidas son aquellas que, por su sensibilidad o potencial peligrosidad, no podrá circular ningún dron. Ejemplos de estas zonas podrían ser lugares urbanizados y altamente poblados en los que se pondría en peligro la seguridad y la privacidad de la gente; recintos en los que puntualmente existe una gran aglomeración de personas, como estadios deportivos; centrales nucleares, etc.

Si bien la entrada de un dron en esta zona puede ser debida a un error humano o técnico de los sistemas de abordo, no es descartable la existencia de que drones malintencionados sobrevuelen estas zonas. Asumiendo, entonces, que algunos drones podrían no radiodifundir su posición para ser capaces de entrar en zonas prohibidas sin ser detectados por el sistema, debería existir en dichas zonas sistemas de teledetección que sean capaces de localizar intrusos en la zona como, por ejemplo, un sistema LIDAR como el descrito en el apartado 2.3.2.

FIGURA 3.5: Esquema del diseño

El sistema de detección, en cualquier caso, sería el encargado de comunicar al programa la posición y, de ser posible, otros parámetros como la velocidad del dron intruso. Al recibir la información del sistema de detección, el programa identificaría automáticamente al dron como un intruso, pudiendo así avisar a las autoridades correspondientes.

3.5. Ejecución en un clúster

Debido a la gran cantidad de información que el sistema manejaría y, sobre todo, a la necesidad de responder prácticamente en tiempo real, el programa debería utilizar un framework de Big Data como Apache Spark para distribuir el trabajo en un clúster.

De este modo, cada uno de los planos en los que se divide el mapa son asignados a un nodo del clúster. Estos planos pueden variar en tamaño, en función del número de drones que transiten el mismo en un momento dado. Por ejemplo, si existe un plano en el que transitan una gran cantidad de drones al mismo tiempo, éste puede dividirse en dos planos más pequeños, utilizando por tanto dos nodos para la vigilancia de la zona en la que antes trabajaba sólo uno.

Del mismo modo, puede aumentarse el tamaño de un plano en el caso de que tenga una concentración muy baja de drones, de manera que el

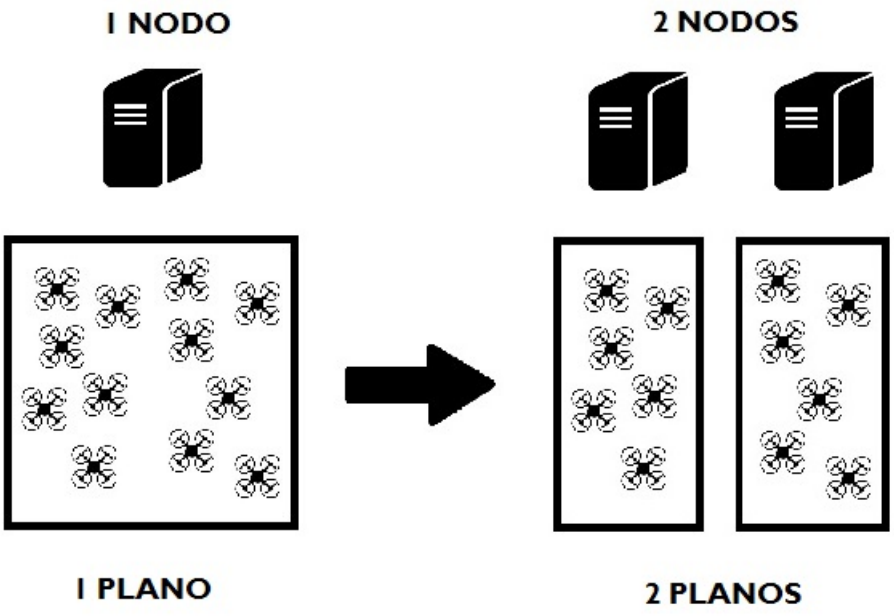

FIGURA 3.6: Ejemplo de división de un plano

nodo encargado de vigilar dicho plano libere de trabajo a otros nodos que cuentan con una mayor concentración de drones en su plano.

3.6. Centro de control de drones

El centro distribuido de control de drones sería el encargado de realizar todas las operaciones necesarias para velar por la seguridad del espacio aéreo que le corresponde. De acuerdo con lo explicado anteriormente, el mapa a vigilar se dividiría en secciones más pequeñas, de modo que éstas puedan ser vigiladas de manera independiente por los nodos del clúster. Las características que debería cumplir este sistema son las siguientes:

- Avisar a aquel dron que se haya salido o vaya a salirse de la zona en la que tiene permitido circular en próximos instantes de tiempo. Si procede, avisar a las autoridades competentes.
- Detectar con antelación choques entre drones, comprobando que dos o más de ellos nunca se vayan a encontrar en la misma celda en un mismo instante de tiempo.
- Tratar de ofrecer una solución para evitar la colisión. Para ello, se recomendaría un cambio de velocidad a uno o varios de los drones involucrados. Si esto no fuera solución suficiente, se calcularía una ruta

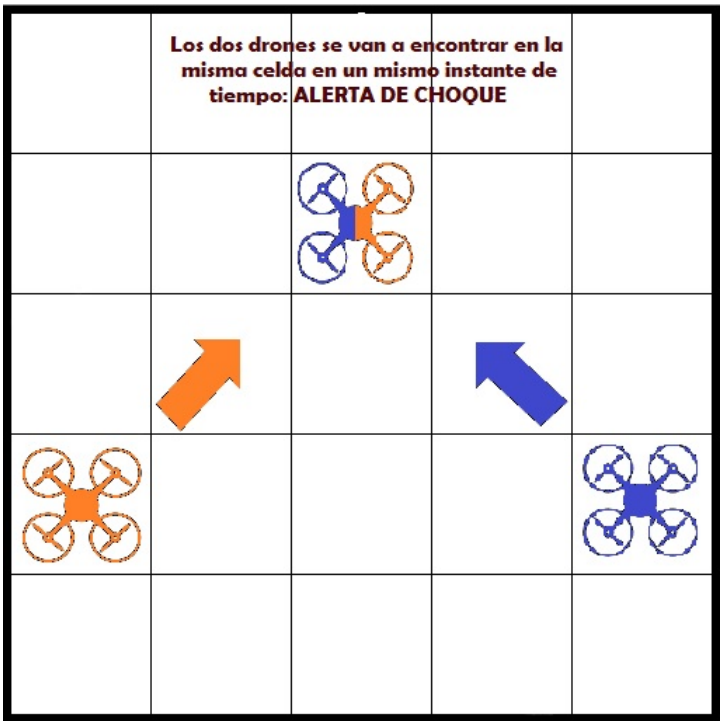

FIGURA 3.7: Esquema de un choque

alternativa para uno o varios de los drones involucrados. Cada dron contaría con un grado de prioridad, de cara a mantener su velocidad y trayectoria si se ve envuelto en una futura colisión.

- Avisar a las autoridades competentes si el sistema de vigilancia de zonas prohibidas detecta un dron intruso en dichas zonas.
- Ofrecer una interfaz gráfica en la que visualizar el tránsito de los drones por el mapa.
- Mantener a los pilotos de drones informados acerca de las otras aeronaves que se encuentran cerca de su dron, además de otra información potencialmente útil como temperatura ambiental, velocidad del viento, etc.

Capítulo 4

Prototipo

Por falta de recursos, tanto humanos como económicos, además de la limitación de tiempo para llevar a cabo este proyecto, el diseño planteado no puede llevarse a cabo en su totalidad. En su lugar, se ha desarrollado un prototipo que, se espera, demuestre a pequeña escala cómo funcionaría la idea descrita anteriormente.

Para este prototipo, se ha asumido que los datos son enviados por los drones en el formato especificado en el diseño, es decir:

```
id, posX, posY, vel, destX, destY, tipo
```

Estos datos (más detalles acerca del significado de cada uno en el apartado 3.2) son la fuente de información a la que accederán **todos** los nodos del clúster. Así, cada nodo busca los drones que se encuentran dentro del plano que tiene asignado, lo añade a su lista de drones y realiza con sus datos las operaciones necesarias.

4.1. Arquitectura

El prototipo del software consiste las siguientes clases programadas en Java:

4.1.1. Clase `Plane`

La clase `Plane` define un plano, y su estructura es la siguiente:

- **`startX`:** indica la componente X del punto inicial del plano.
- **`startY`:** indica la componente Y del punto inicial del plano.
- **`limitX`:** indica la componente X del punto final del plano.
- **`limitY`:** indica la componente Y del punto final del plano.

```
public class Plane implements Serializable {

    private static final long serialVersionUID = 6646626948046499013L;
    double startX;
    double startY;
    double limitX;
    double limitY;
    int numberofDrones;
    int numberofleisZ;
    int numberofcommZ;
    int numberofdZ;
    Dron [] dronesArray;
    commercialZone [] commZ;
    leisureZone [] leisZ;
    dangerZone [] dZ;
```

FIGURA 4.1: Estructura de la clase `Plane`

- **`numberofDrones:`** indica el número de drones que se encuentran situados dentro de las coordenadas del plano.
- **`numberofleisZ:`** indica el número de zonas de ocio que se encuentran enclavadas dentro del plano.
- **`numberofcommZ:`** indica el número de zonas comerciales que se encuentran enclavadas dentro del plano.
- **`numberofdZ:`** indica el número de zonas prohibidas que se encuentran enclavadas dentro del plano.
- **`dronesArray:`** colección en la que se almacena (en forma de objetos `Dron`) la información de todos los drones contenidos en el plano.
- **`commZ:`** colección en la que se almacena (en forma de objetos `commercialZone`) la información -puntos iniciales y finales- de las zonas comerciales enclavadas dentro del plano.
- **`leisZ:`** colección en la que se almacena (en forma de objetos `leisureZone`) la información -puntos iniciales y finales- de las zonas de ocio enclavadas dentro del plano.
- **`dZ:`** colección en la que se almacena (en forma de objetos `dangerZone`) la información -puntos iniciales y finales- de las zonas prohibidas enclavadas dentro del plano.

Además, la clase `Plane` cuenta con un constructor, en el que se pasan como parámetros las coordenadas iniciales y finales del plano, así como con 4 sencillos métodos:

- **`insertDron:`** recibe un objeto `Dron` como parámetro y lo añade a la colección de drones del plano.

```
public void insertDron(Dron newDron){
public void deleteDron(int id){
public void insertleisureZone(leisureZone newleisZ){
public void insertcommercialZone(commercialZone newcommZ){
public void insertdangerZone(dangerZone newdZ){
```

FIGURA 4.2: Métodos de la clase `Plane`

- **`deleteDron:`** recibe el identificador de un dron como parámetro, busca dicho dron en la colección del plano y, si lo encuentra, lo elimina.
- **`insertleisureZone:`** recibe un objeto `leisureZone` como parámetro y lo añade a la lista de zonas de ocio del plano.
- **`insertcommercialZone:`** recibe un objeto `commercialZone` como parámetro y lo añade a la lista de zonas comerciales del plano.
- **`insertdangerZone:`** recibe un objeto `dangerZone` como parámetro y lo añade a la lista de zonas prohibidas del plano.

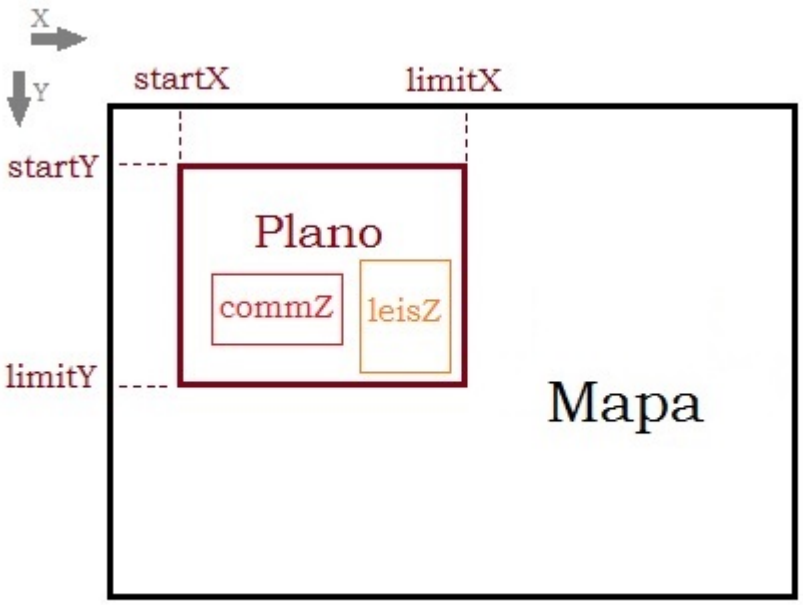

FIGURA 4.3: Representación gráfica de un plano

4.1.2. Clase `leisureZone`

La clase `leisureZone` define un plano en dos dimensiones en el que sólo podrán circular aquellos drones cuyo uso esté destinado al ocio. Como

se ha mencionado en el apartado 3.1.1, cada plano puede contener una o varias de estas zonas de ocio, en forma de objetos.

La estructura de la clase `leisureZone` es la siguiente:

```
public class leisureZone implements Serializable {

    private static final long serialVersionUID = 707249244983264113L;
    double startX;
    double startY;
    double endX;
    double endY;
```

FIGURA 4.4: Estructura de la clase `leisureZone`

- **`startX`:** indica la componente X del punto inicial de la zona de ocio.
- **`startY`:** indica la componente Y del punto inicial de la zona de ocio.
- **`endX`:** indica la componente X del punto final de la zona de ocio.
- **`endY`:** indica la componente Y del punto final de la zona de ocio.

La clase `leisureZone` contiene, además, un constructor en el que se reciben como parámetros dichas coordenadas iniciales y finales.

4.1.3. Clase `commercialZone`

La clase `commercialZone` define un plano en dos dimensiones en el que sólo podrán circular aquellos drones destinados a un uso comercial. Como se ha mencionado en el apartado 3.1.1, cada plano puede contener una o varias de estas zonas comerciales, en forma de objetos.

La estructura de la clase `commercialZone` es la siguiente:

```
public class commercialZone implements Serializable {

    private static final long serialVersionUID = 8889519062410570655L;
    double startX;
    double startY;
    double endX;
    double endY;
```

FIGURA 4.5: Estructura de la clase `commercialZone`

- **startX:** indica la componente X del punto inicial de la zona comercial.
- **startY:** indica la componente Y del punto inicial de la zona comercial.
- **endX:** indica la componente X del punto final de la zona comercial.
- **endY:** indica la componente Y del punto final de la zona comercial.

La clase `commercialZone` contiene, además, un constructor en el que se reciben como parámetros dichas coordenadas iniciales y finales.

4.1.4. Clase `dangerZone`

La clase `dangerZone` define un plano en dos dimensiones en el que, por su peligrosidad o sensibilidad, no podrá circular ningún dron. Como se ha mencionado en el apartado 3.1.1, cada plano puede contener una o varias de estas zonas prohibidas, en forma de objetos.

La estructura de la clase `dangerZone` es la siguiente:

```
public class dangerZone implements Serializable {

    private static final long serialVersionUID = 8515487212467421405L;
    double startX;
    double startY;
    double endX;
    double endY;
```

FIGURA 4.6: Estructura de la clase `dangerZone`

- **startX:** indica la componente X del punto inicial de la zona prohibida.
- **startY:** indica la componente Y del punto inicial de la zona prohibida.
- **endX:** indica la componente X del punto final de la zona prohibida.
- **endY:** indica la componente Y del punto final de la zona prohibida.

La clase `dangerZone` contiene, además, un constructor en el que se reciben como parámetros dichas coordenadas iniciales y finales.

4.1.5. Clase `Dron`

La clase `Dron` sirve para que el programa pueda manejar de manera eficiente toda la información relevante de un dron. Así, cuando el programa recibe el texto plano en el que los drones envían parámetros como su posición o su velocidad, entre otros, el programa almacena estos datos en un objeto `Dron` para posteriormente poder operar con ellos.

La estructura de la clase `Dron` es la siguiente:

```
public class Dron implements Serializable{

    private static final long serialVersionUID = -7026400437095171656L;
    double positionX;
    double positionY;
    int cellX;
    int cellY;
    int [] futureCellsX = new int [40];
    int [] futureCellsY = new int [40];
    double velocity;
    double velocityX;
    double velocityY;
    double destinationX;
    double destinationY;
    double theta;
    int id;
    int type;
    int priority;
```

FIGURA 4.7: Estructura de la clase `Dron`

- **`positionX`:** indica la componente X de la posición actual del dron.
- **`positionY`:** indica la componente Y de la posición actual del dron.

Como se ha mencionado al comienzo del apartado 3, cada plano se divide en pequeñas celdas de 1 unidad x 1 unidad. Dichas celdas quedan definidas por sus coordenadas (un número entero) en X y en Y. De este modo:

- **`cellX y cellY`:** indican la celda en la que se encuentra el dron.
- **`futureCellsX y futureCellsY`:** indican las celdas donde se encontrará un dron en los futuros 40 instantes de tiempo, si mantiene su dirección y velocidad actuales.
- **`velocity`:** indica la velocidad del dron, en unidades/segundo.

- **`velocityX:`** indica la velocidad a la que avanza el dron en la componente X.
- **`velocityY:`** indica la velocidad a la que avanza el dron en la componente Y.
- **`destinationX:`** indica la componente X del punto de destino del dron.
- **`destinationY:`** indica la componente Y del punto de destino del dron.
- **`theta:`** indica el valor del ángulo relativo entre el vector velocidad y su componente en X.
- **`id:`** indica el identificador del dron.
- **`type:`** indica el tipo del dron. El número 1 significa que el dron es de ocio, el número 2 que es comercial y el número 3 indica que el dron ha sido detectado en una zona prohibida.
- **`priority:`** indica la prioridad que tiene el dron frente a otro para mantener su velocidad, si se recomienda la variación de la misma a la hora de evitar un supuesto choque entre ambos.

La clase `Dron` posee un constructor que recibe como parámetros la posición del dron (en X e Y), su velocidad, punto de destino (en X e Y), identificador y tipo. En el constructor, por trigonometría, se calcula `theta` y, mediante ésta, se obtiene fácilmente la descomposición de la velocidad en X e Y. La celda en la que se encuentra el dron se define haciendo un *casting*, de `double` a entero, a la posición donde se encuentra el dron -resultando, por ejemplo, que la celda de un dron que se encuentre en la posición (X, Y) = (23'75, 198,32) será la celda (X, Y) = (23, 198)-. Por último, se calculan todos los valores de `futureCellsX` y `futureCellsY` usando la posición y velocidad (en X e Y) actuales del dron, así como el tiempo de discretizado, en milisegundos, que se quiera utilizar, llamado `discrTime`.

4.1.6. Clase `MyFrame`

La clase `MyFrame` es la primera de las tres clases que se utilizan para representar gráficamente de manera sencilla el movimiento de los drones sobre los distintos planos que componen el mapa completo.

Hereda de la clase de Swing (biblioteca gráfica) de Java `JFrame` y está orientada a construir un marco en el que luego introducir un panel. Para ello, se instancia un objeto `MyPanel` (que recibe como parámetro el nombre de la imagen de fondo del panel) y se añade este panel a un objeto `JScrollPane`, con el objetivo de poder hacer *scroll*. En el constructor se declaran los atributos básicos del marco, como el tamaño, o la visibilidad. Además, se añade el panel **panelPane**, objeto de la clase `MyPanel`.

La clase dispone también del método `move`, que a su vez llama al método `initializeRects`, de la clase `MyPanel`.

4.1.7. Clase `MyPanel`

La clase `MyPanel` contiene la mayor parte del código orientado a representar gráficamente el movimiento de los drones sobre los planos. Hereda de la clase de Swing (biblioteca gráfica) de Java `JPanel`, y consta de una colección (*array*) de rectángulos (cada uno de los cuales representará un dron) y de dos constructores:

- El primer constructor es el que se utiliza para instanciar un panel desde la clase `MyFrame`, y recibe como argumento el nombre de la imagen de fondo del panel.
- El segundo constructor se utiliza para instanciar de nuevo el panel, esta vez con el objeto Imagen como argumento.

Además, cuenta con las funciones:

- **`paintComponent`:** en la clase `MyPanel` se hereda del método `paintComponent` de `JPanel` y se añaden algunos factores, como un contexto gráfico de la clase `Graphics2D`. De acuerdo a la documentación Java: *"La clase Graphics2D extiende de la clase Graphics para proporcionar un control más sofisticado sobre la geometría, las transformaciones de coordinadas o el manejo del color. Es la clase fundamental para representar formas, texto e imágenes en dos dimensiones en la plataforma Java(tm)"*. Además, en el método `paintComponent` se dibujan tanto las líneas que servirán para distinguir el paso de un plano a otro, como los rectángulos que representan a los drones (recorriendo la colección de rectángulos previamente mencionada) y los identificadores de estos rectángulos, que ese corresponden con los identificadores de los drones a los que representan. Cada dron se pinta de un color diferente dependiendo de su tipo (de ocio, comercial o detectado en una zona prohibida).
- **`addRect`:** este método recibe como parámetros la componente X e Y del nuevo rectángulo que se quiere añadir a la colección, y le asigna como altura y anchura predeterminadas el valor 5.
- **`initializeRects`:** este método recibe como parámetro una lista de objetos `Dron`. Por cada dron de esta lista, borra la antigua colección de rectángulos y añade un nuevo rectángulo en la posición que indica la celda actual del dron.
- **`getPreferredSize`:** este método especifica el tamaño del panel.

4.1.8. Clase `MyRectangle`

La clase `MyRectangle` hereda de la clase de Swing (biblioteca gráfica) de Java `Rectangle`, y ha sido desarrollada para añadir un constructor en el que, además de las coordenadas, anchura y altura del rectángulo, se introduzca el identificador y el tipo de dron al que representa.

De esta manera, desde la clase `MyPanel`, cada dron puede ser dibujado, junto a su identificador, de distinto color según sea comercial, de ocio o detectado en una zona prohibida.

4.1.9. Clase `crearFicherosdeTiempo`

Como se ha explicado con anterioridad, en este prototipo se asume que los drones envían la información a la estación base donde se ejecuta el software en forma de texto plano, con el formato descrito en el apartado 3.2. Puesto que no se posee de drones reales y, además, que no existe un estándar para los datos enviados por los drones, se ha desarrollado la clase `crearFicherosdeTiempo`, cuya función es generar archivos de texto que correspondan a la posición, velocidad y demás parámetros de los drones en cada instante de tiempo, con el fin de simular el tráfico de datos que produciría un conjunto real de drones.

El valor de la posición y el destino de los drones es generado de manera aleatoria, con una distribución aproximadamente uniforme entre los valores iniciales y finales del mapa. Se asigna también de manera aleatoria a cada dron un número entero indicando su tipo. A los drones comerciales se les asigna una velocidad de 10 m/s mientras que, tanto a los drones recreativos como a los drones intrusos, se les asigna en la simulación una velocidad de 7 m/s. Por último, cabe mencionar que el tipo de cada dron (es decir, el atributo que designa si un dron está destinado a un uso comercial o recreativo o es un intruso) se decide en función de la posición inicial que ocupe en el mapa. De este modo, todos los drones cuya posición inicial se encuentre en una zona comercial, son designados desde el principio como drones comerciales. Lo mismo ocurre con las zonas de ocio y las zonas prohibidas.

4.1.10. Clase `Main`

La clase `Main` es la clase más extensa, en cuanto a líneas de código, y es dónde se ejecuta realmente el proyecto. Su estructura es la siguiente:

- **`discrTime:`** indica la medida con la que se discretiza el tiempo (esto es, cada cuánto tiempo se reciben datos de los drones para operar con ellos), en milisegundos.
- **`contador:`** es un número entero que se usa para llevar la cuenta de los planos cuando se informa de los drones que contiene cada uno.

```
private static final long serialVersionUID = 1100664403737529393L;
static double discrTime = 500;
static int contador = 0;
static double mapWidth = 800;
static double mapLength = 800;
```

FIGURA 4.8: Estructura de la clase `Main`

- **`mapWidth:`** indica el ancho del mapa en el que se va a representar gráficamente el movimiento de los drones. Se utiliza en el método GPStoXY.
- **`mapLength:`** indica el ancho del mapa en el que se va a representar gráficamente el movimiento de los drones. Se utiliza en el método GPStoXY.

Además, la clase `Main` consta de varias funciones. Téngase en cuenta si se va a analizar el código de las mismas lo enunciado en el apartado .....: un RDD es una colección distribuida entre los nodos de un clúster y es **inmutable** (lo cual tiene sentido, podría ser caótico que cada nodo pudiese cambiar el contenido del RDD cuando se intenta realizar una computación paralela). Lo que sí es posible es obtener un nuevo RDD basándose en uno anterior. Si se profundiza en los métodos descritos en las siguientes páginas, podrá observarse que se utilizan con frecuencia dos funciones aplicables a un `JavaRDD`:

- **`foreach(VoidFunction f):`** aplica la función `f` a todos los elementos de este RDD.
- **`map(Function(T, R)):` devuelve un nuevo RDD** resultado de aplicar la función `f` a todos los elementos de este RDD.

Podría decirse, por tanto, que `foreach` es una función sólo de lectura, mientras que mediante `map` podemos obtener un nuevo RDD resultado de aplicar al anterior las operaciones deseadas.

Los métodos de la clase `Main` son los siguientes:

- **`initializeMap:`** este método inicializa el mapa en el que se va a trabajar. Para ello, el procedimiento es el siguiente:
  1. Se definen todos los planos que componen el mapa y se almacenan en un `ArrayList`.
  2. Se añaden a estos planos las zonas de ocio, comerciales o prohibidas que el usuario considere oportunas.

  Para añadir las zonas, se extrae del `ArrayList` el plano que se desea modificar, se almacena este plano en una variable `Plane` auxiliar, la cual se modifica introduciendo la zona de ocio, comercial o prohibida

```
public static ArrayList<Plane> initializeMap1(){
public static Dron[] readDrones(String ruta) throws
public static boolean isInside(Plane plane, Dron dron){
public static void planeState(JavaRDD<Plane> planesRDD){
public static void printDronesInfo(JavaRDD<Plane> planesRDD){
public static JavaRDD<Plane> initializeDrones(JavaRDD<Plane> planesRDD,
public static void anticipateBadDron(JavaRDD<Plane> planesRDD){
public static void detectBadDron(JavaRDD<Plane> planesRDD){
public static void checkColissions(JavaRDD<Plane> planesRDD){
public static double [] GPStoXY (double latitude, double longitude,
public static double toDecimalDegrees(double degrees,
```

FIGURA 4.9: Métodos de la clase `Main`

deseada, para posteriormente introducir de nuevo este plano modificado en el `ArrayList`. El método devuelve un **RDD** (ver apartado 1.2.5.....) de planos, listo para ser distribuido entre los nodos del clúster.

- **`readDrones:`** este método es el encargado de leer los archivos de texto con los que los drones comunican su posición, velocidad y demás parámetros, así como de almacenar esta información en objetos `Dron`, para poder operar con ella internamente. Cabe mencionar que, en el formato asumido, cada línea del archivo de texto contiene todos los datos enviados por un dron, y cada uno de los datos está separado del anterior por una coma. Por ello, el procedimiento del método es el siguiente:

  1. Se lee el archivo de texto y cada línea se almacena en una colección de `String`.
  2. Cada uno de estos `String` (cada línea) se divide en varios `String` (cada dato) usando las comas como divisor.
  3. Con estos datos, se pueden instanciar los objetos `Dron` necesarios.

  Estos objetos se almacenan en una colección, que es la que devuelve el método.

- **`isInside:`** es un método auxiliar que recibe como parámetros un objeto `Plane` y un objeto `Dron`, comprueba si el dron se encuentra dentro de los límites de ese plano y devuelve un booleano que será **true**, si efectivamente el dron se encuentra en el plano, o **false**, si no es así.

- **`planeState:`** este método es útil para comprobar en qué plano se encuentran cada dron. Recibe como parámetro el RDD de planos y lo recorre, imprimiendo el número del plano y los drones que éste contiene.

- **`printDronesInfo:`** el objetivo de este método es imprimir la posición y la celda en la que se encuentra cada dron. Para ello:

1. Recorre el RDD de planos.
2. Para cada plano, recorre su colección de drones.
3. Para cada dron, imprime su posición y su celda.

- **`initializeDrones:`** recibe como parámetros el RDD de planos y la colección de drones que devuelve el método `readDrones`. El objetivo de este método es asignar cada dron al plano que le corresponda. Para ello:

  1. Recorre el RDD de planos.
  2. Para cada plano, borra su colección de drones.
  3. Se recorre la colección de drones recibida como parámetro. Se comprueba, por la posición de cada dron, si se encuentra dentro de los límites del plano actual. Si es así, se añade a la colección de drones del plano.

  De este modo, se soluciona el problema de cómo manejar el hecho de que un dron pase de un plano a otro. Constantemente se están borrando y volviendo a inicializar las colecciones de drones de cada plano.

- **`detectBadDron:`** recibe como parámetro el RDD de planos. El objetivo de este método es comprobar que cada dron esté en la zona que le corresponde según su tipo (de ocio o comercial), así como de que no haya ningún dron en una zona prohibida. Para ello, se utiliza un booleano `goodDron` que indicará si un dron está en una zona válida (goodDron valdrá `true`) o no (goodDron valdrá `false`, su valor por defecto). El procedimiento es el siguiente:

  1. Se recorre el RDD de planos.
  2. Para cada plano, se recorre su colección de drones.
  3. Se comprueba el tipo de cada dron. Según su tipo, se recorre la colección de zonas del plano para asegurarse de que el dron se encuentre dentro de alguna de estas zonas. Por ejemplo: si el dron es comercial, se recorre la lista de zonas comerciales del plano y se comprueba si el dron se encuentra dentro de los límites de alguna de las zonas. En caso afirmativo, `goodDron` se pone a `true`. Si se recorre toda la lista de zonas comerciales y si la posición del dron no se encuentra en ninguna de ellas, el dron estará fuera de sus límites y, por tanto, `goodDron` permanecerá en `false`.
  4. Se comprueba el estado de `goodDron` y se avisa si hay algún dron fuera de sus límites, indicando el identificador de este dron.

- **`anticipateBadDron:`** este método es similar en su ejecución a `detectBadDron`, con la diferencia de que, en vez de analizar la posición actual del dron para comprobar si éste se encuentra en una de sus zonas válidas, se comprueban sus posiciones futuras (mediante `futureCellsX` y `futureCellsY`) para advertir, con tiempo suficiente, de que el dron podría salirse de sus zonas válidas si mantiene la trayectoria que tiene hasta el momento.

- **`checkColissions:`** este método es el encargado de comprobar si se va a producir una colisión dentro de un plano. Para ello:

  1. Se recorre el RDD de planos.
  2. Para cada plano, se va a comprobar la posibilidad de choque en los siguientes **40 instantes de tiempo** (se recuerda que, en cada objeto Dron, existen dos atributos -`futureCellsX` y `futureCellsY`- que indican la celda en la que se va a encontrar el dron en los siguientes 40 instantes de tiempo).
  3. En cada instante de tiempo, se comprueba con dos bucles (para cada dron -primer bucle-, con todos los demás drones -segundo bucle-) que dos drones **diferentes** no vayan a estar en una misma celda durante el mismo instante de tiempo.

  Un choque puede ser de tres tipos diferentes:

  1. **Frontal** o casi-frontal: en un choque frontal, dos drones tienen la misma dirección, pero sentidos opuestos. El choque será frontal cuando el atributo `theta` de los dos drones difiera en pi (con una permisividad de 0.1).
  2. **Trasero** o casi-trasero: en un choque trasero, dos drones tienen la misma dirección y el mismo sentido. El choque será trasero cuando el atributo `theta` de los dos drones sea igual (con una permisividad de 0.1).
  3. **Otro:** en cualquier otro caso (como, por ejemplo, el de la figura), el choque no será ni trasero ni frontal.

  La forma de tratar estos choques será, por tanto, diferente:

  1. Un choque **trasero** podrá ser evitado si el dron que circula más rápido reduce su velocidad para ajustarla a la del dron más lento.
  2. En un choque **frontal**, la única solución es un cambio de trayectoria de, al menos, uno de los drones involucrados. Mediante el atributo `priority`, que indica la prioridad de cada dron, puede deducirse cuál es el dron más urgente (o con mayor prioridad), que será a su vez el dron que mantendrá la trayectoria, mientras el otro tendrá que desviarse.
  3. En cualquier otro choque, la solución puede pasar también por una variación en la velocidad de, al menos, uno de los drones involucrados. De nuevo, con el sistema de prioridades, puede mantener su velocidad el dron con mayor prioridad. En una primera aproximación a la solución de este problema, se estudia el caso de que el dron con menor prioridad disminuyera su velocidad en un 25 %, un 50 % y un 75 %. Si para alguno de los casos, el dron no colisiona con ningún otro dron de su zona, se recomienda dicho cambio de velocidad. En caso de que ninguna de las tres variaciones suponga una solución válida (bien porque no se solucione el choque tratado o porque con la nueva velocidad se produzca una colisión distinta), se recomienda un cambio de trayectoria.

- **`GPStoXY:`** este método está orientado a una futura ampliación del prototipo, en la que la información que transmiten los drones indica la posición de éstos en el sistema de coordenadas geográficas. Así, el método transforma un punto de coordenadas (longitud y latitud) en un píxel de un mapa (representado por sus valores en x e y). Para ello, recibe como parámetros la longitud y latitud del punto en cuestión, la anchura y altura en píxeles del mapa que vamos a representar, así como su longitud y latitud máximas y mínimas.

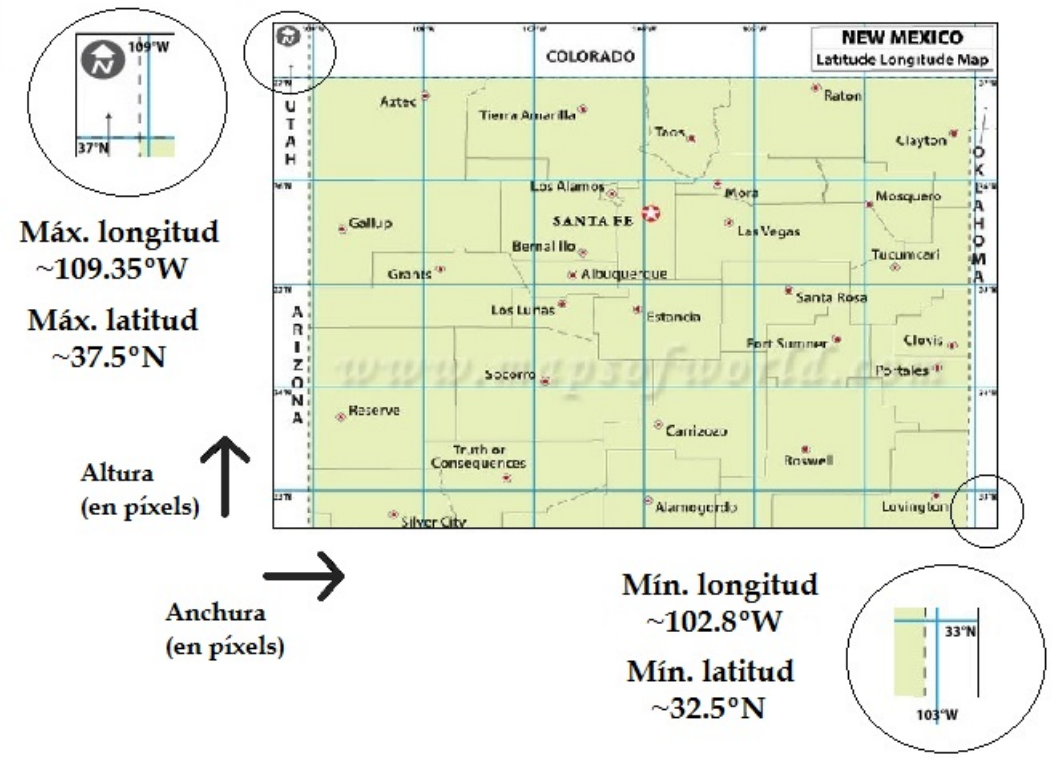

FIGURA 4.10: Medidas de un mapa

- **`toDecimalDegrees:`** este método transforma los grados expresados en el sistema sexagesimal en grados expresados en el sistema decimal.
- **`main:`** desde este método se produce realmente la ejecución del programa. El procedimiento es el siguiente:

  1. Se crea una colección en la que se almacenarán los nombres de los archivos a los que se accederá para obtener los parámetros simulados.
  2. Se crea un objeto `SparkConf`, el cual contiene la configuración para una aplicación *Spark* y se usa para establecer varios parámetros de *Spark* en forma de pares clave-valor. La llamada a `SparkConf()` carga unos valores predeterminados obtenidos de las propiedades del sistema y las rutas de las clases.

3. Se crea un `JavaSparkContext`, que no es más que una versión de `SparkContext` adaptada a Java, la cual devuelve `JavaRDDs` y trabaja con colecciones Java en vez de con colecciones Scala. El `SparkContext` representa la conexión a un clúster *Spark*, y se usa para crear RDDs, entre otras variables, en el clúster (sólo puede haber un `SparkContext` activo por cada **JVM** -Java Virtual Machine-).
4. Se llama a `initializeMap()` para inicializar el mapa deseado, y se recoge la lista de planos que devuelve el método.
5. Se paraleliza la lista de planos mencionada en el punto anterior, generando de esta manera el RDD con el que se trabajará en el clúster.
6. Se instancia el marco en el que se representarán gráficamente los drones simulados.
7. Para simular la llegada de datos cada `discrTime`, se pone a dormir el proceso durante los milisegundos indicados.

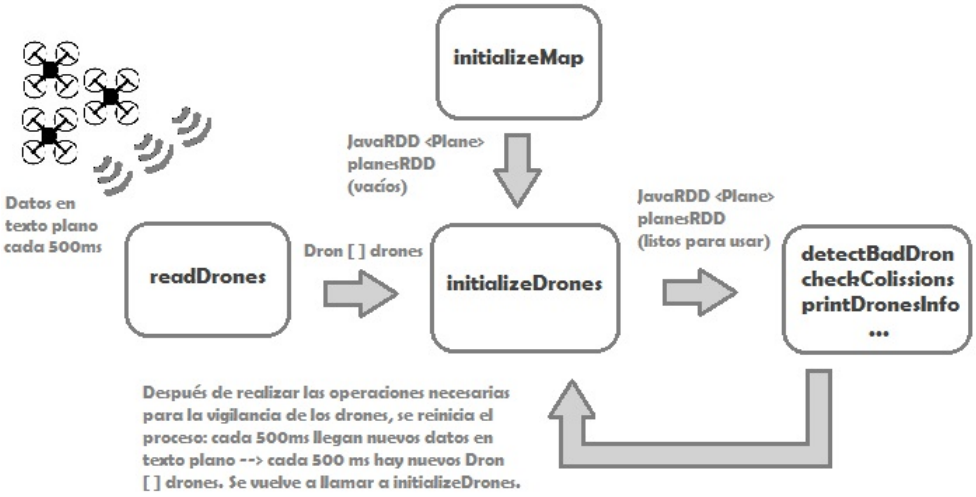

FIGURA 4.11: Funcionamiento esquemático de la clase `Main`

8. Se procede a la llamada en bucle de los métodos que vigilarán la correcta circulación de los drones, así como a `move`, que moverá los rectángulos que representan los drones gráficamente.
9. Se cierra el `JavaSparkContext`.

Capítulo 5

Pruebas

En este capítulo se realizarán algunas pruebas para comprobar el correcto funcionamiento del prototipo diseñado. Para ello, se simulará el vuelo de drones sobre dos mapas diferentes, variando para cada mapa el número de drones, sus posiciones iniciales y su punto de destino.

Para su distinción a simple vista, se han coloreado los distintos tipos de zonas en cada mapa de diferente color. Así, toda aquella zona destinada al vuelo de drones comerciales aparece representada en color naranja; las zonas en las que pueden volar drones de ocio, en color azul y, por último, las zonas en las que no puede volar ningún dron se distinguen por su color rojo. Como se mencionó en el capítulo 4, también los drones son pintados de distinto color según su tipo: marrón para los comerciales, azul para los de ocio y rojo para aquellos detectados por el sistema de vigilancia en zonas prohibidas.

Se recuerda, así mismo, que en este prototipo cada celda se corresponde con un píxel, y cada plano es gestionado por un nodo.

Por último, nótese que tras cada ejecución de la clase `crearFicherosdeTiempo`, los valores de posición, destino, tipo y velocidad de los drones cambiarán de manera aleatoria (leer apartado **4.1.9** para más información).

5.1. Primer mapa

Las características del primer mapa y las condiciones en las que se realizan las pruebas son:

- Número de planos: **16**
- Número de celdas por plano: **10.000** (100 x 100)
- Número total de celdas en el mapa: **160.000**
- Superficie de una celda: **100 metros cuadrados**
- Superficie total del mapa: **16 kilómetros cuadrados** (1.600 hectáreas)
- Número de drones simulados: **32**

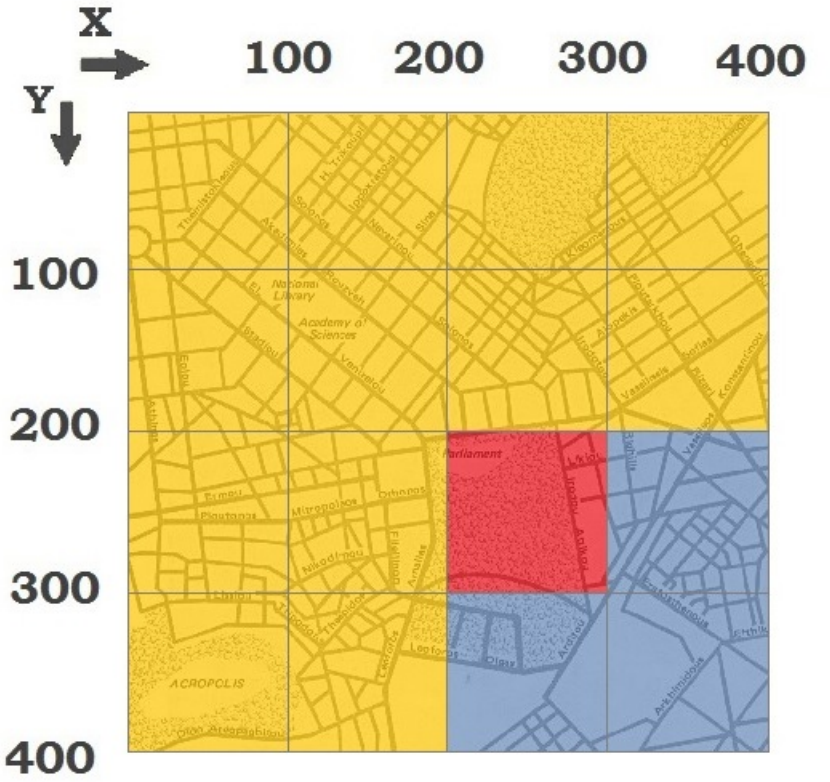

FIGURA 5.1: Primer mapa de las pruebas

- Tiempo de simulación: **30 segundos**.

Ya que los datos del estado inicial de los drones están inicializados al azar, durante los 30 segundos de duración del programa, la mayor parte de los drones no realizará ninguna acción que el sistema considere destacable (por ejemplo, salirse de su zona o formar parte de una futura colisión). Por ello, se han seleccionado algunos drones, cuyo comportamiento sí es destacable, en la siguiente tabla:

ID	Posición inicial (x, y)	Punto de destino (x, y)	Velocidad	Tipo	Prioridad
4	(263, 297)	(111, 380)	7 m/s	Intruso	0
5	(51, 229)	(307, 173)	10 m/s	Comercial	27
15	(326, 204)	(326, 85)	7 m/s	Ocio	0
26	(329, 187)	(248, 292)	10 m/s	Comercial	5
28	(51, 215)	(397, 298)	10 m/s	Comercial	3

CUADRO 5.1: Características iniciales de algunos drones relevantes del mapa 1

La pantalla de visualización gráfica de los drones tiene el siguiente aspecto en el instante de tiempo inicial:

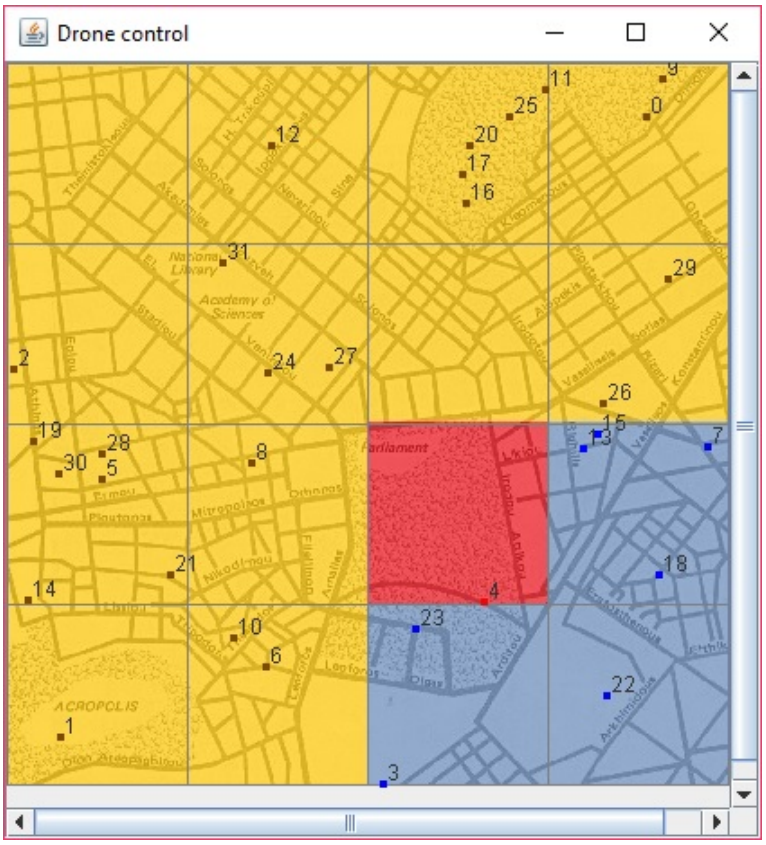

FIGURA 5.2: Mapa 1 en el instante inicial

Desde el instante inicial el programa detecta la presencia de un dron intruso, el cual tiene el identificador número 4:

```
HAY UN DRON INTRUSO (ID = 4) EN LA POSICIÓN X = 254.398786,
301.69673
```

Tras 6 segundos de simulación de programa, un dron de ocio abandona su zona para introducirse en una zona comercial, y así lo advierte el sistema:

```
HAY UN DRON DE OCIO (ID = 15) FUERA DE SUS ZONAS PERMITIDAS.
POSICIÓN: 326.0, 197.000001
```

Unos segundos después, en el instante t = 10.5 s, el sistema detecta que el dron 5 y el dron 28 van a colisionar 19.5 segundos más tarde. El programa calcula si el choque podría ser evitado reduciendo la velocidad del dron 28, que es el que tiene menor prioridad. Como en ese instante no encuentra solución, lanza el siguiente mensaje:

```
El dron con ID 28 y el dron con ID 5 van a colisionar
en 19.5 segundos

Se aconseja al dron 28 que cambie de trayectoria
```

Sin embargo, un instante de tiempo más tarde (t = 11 s), cuando todavía

quedan 19 segundos para que el choque tenga lugar, el programa sí encuentra una alternativa que, sin perjudicar al dron con mayor prioridad, evitaría el choque entre estos dos drones y con todos los demás del plano, y así lo comunica:

```
   El dron con ID 28 y el dron con ID 5 van a colisionar
en 19.0 segundos

   Se recomienda al dron 28 que reduzca su velocidad a 2.5
m/s para evitar el choque
```

Al no contar, por falta de medios, con pilotos y drones reales, el dron 28 no sigue la recomendación del programa, por lo que efectivamente colisionarán (ocuparán la misma celda en el mismo instante de tiempo) 19 segundos más tarde.

Por último, cabe mencionar también que el dron número 26, de tipo comercial, saldrá de su zona para adentrarse en una de ocio en el instante de tiempo t = 16.5 s:

```
   HAY UN DRON COMERCIAL (ID = 26) FUERA DE SUS ZONAS PERMITIDAS.
POSICIÓN: 318.6163412, 200.4602983
```

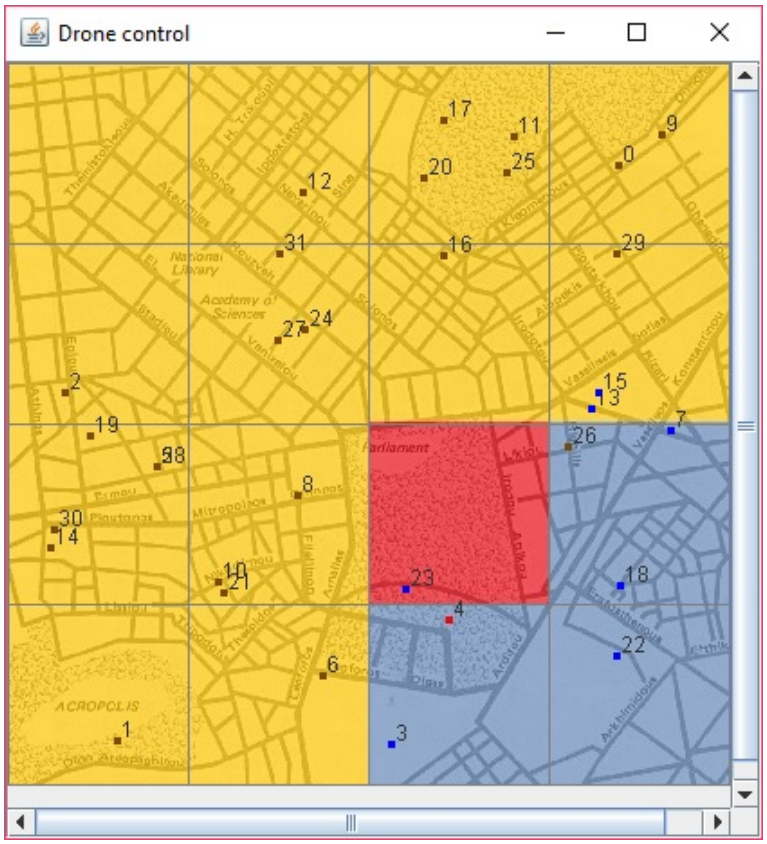

FIGURA 5.3: Mapa 1 tras la simulación

Las advertencias del programa son reflejadas en tiempo real en la pantalla de visualización gráfica de los drones, donde se puede ver cómo éstos se mueven por el mapa y, en el caso de los drones 15 y 26, entre otros, cómo se salen de sus zonas permitidas. Además, se observa que las advertencias del programa son coherentes con lo visualizado en el mapa y suceden en el instante correcto. Tras los 30 segundos de simulación, la pantalla de visualización gráfica tiene el aspecto indicado en la figura 5.3.

5.2. Segundo mapa

Las características del segundo mapa y las condiciones en las que se realizan las pruebas son las siguientes:

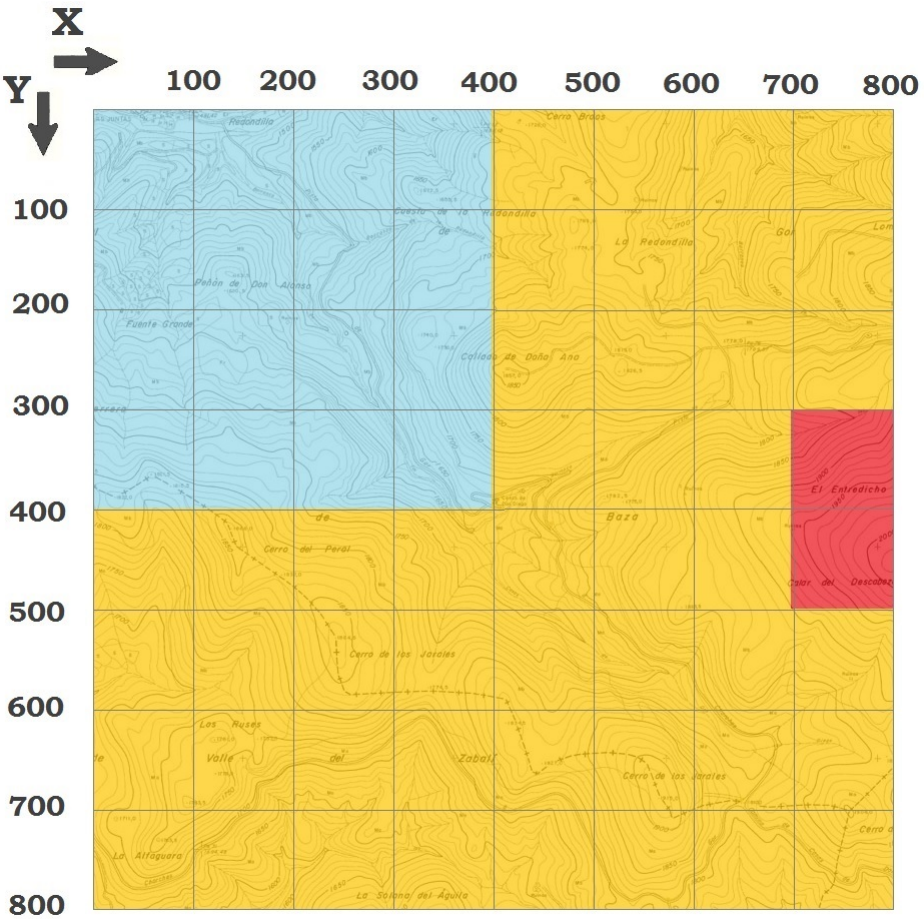

FIGURA 5.4: Segundo mapa de las pruebas

- Número de planos: **64**
- Número de celdas por plano: **10.000** (100 x 100)
- Número total de celdas en el mapa: **640.000**

- Superficie de una celda: **100 metros cuadrados**
- Superficie total del mapa: **64 kilómetros cuadrados** (6.400 hectáreas)
- Número de drones simulados: **128**
- Tiempo de simulación: 30 segundos.

De nuevo, como los drones están inicializados al azar, la mayor parte de ellos no interviene en ningún suceso que ayude a comprobar el correcto funcionamiento del programa. Por ello, se ha seleccionado un grupo de drones relevantes cuyas características iniciales se encuentran en la siguiente tabla:

ID	Posición inicial (x, y)	Punto de destino (x, y)	Velocidad	Tipo	Prioridad
4	(638, 30)	(766, 426)	10 m/s	Comercial	123
8	(646, 29)	(551, 776)	10 m/s	Comercial	119
64	(730, 426)	(225, 355)	7 m/s	Intruso	0
106	(170, 409)	(187, 51)	10 m/s	Comercial	21
124	(724, 511)	(264, 5)	10 m/s	Comercial	3

CUADRO 5.2: Posición inicial de algunos drones relevantes del mapa 2

La pantalla de visualización gráfica de los drones tiene, en el instante de tiempo inicial, el aspecto que se indica en la figura 5.5.

Desde el momento en el que se inicia la simulación, el programa detecta que el dron número 4 y el dron número 8 van a colisionar en 16 segundos. Al no encontrar solución, lanza el siguiente mensaje:

```
   El dron con ID 8 y el dron con ID 4 van a colisionar en
16.0 segundos

   El dron con ID 4 y el dron con ID 8 van a colisionar en
16.0 segundos

   Se aconseja al dron 8 que cambie de trayectoria
```

Al no contar, por falta de medios, con pilotos y drones reales, el dron 8 no sigue la recomendación del programa, por lo que efectivamente colisionarán (ocuparán la misma celda en el mismo instante de tiempo) 16 segundos más tarde.

Además, desde el instante inicial el programa advierte también de la presencia de drones intrusos en una zona prohibida. Entre estos drones se encuentra el número 64, que ha sido incluido en la lista de drones relevantes. El mensaje copiado a continuación avisa de la presencia de este dron, pero se reciben mensajes idénticos para el resto de los drones intrusos (con identificadores 11, 12, 18, 20, 92, 98 y 120).

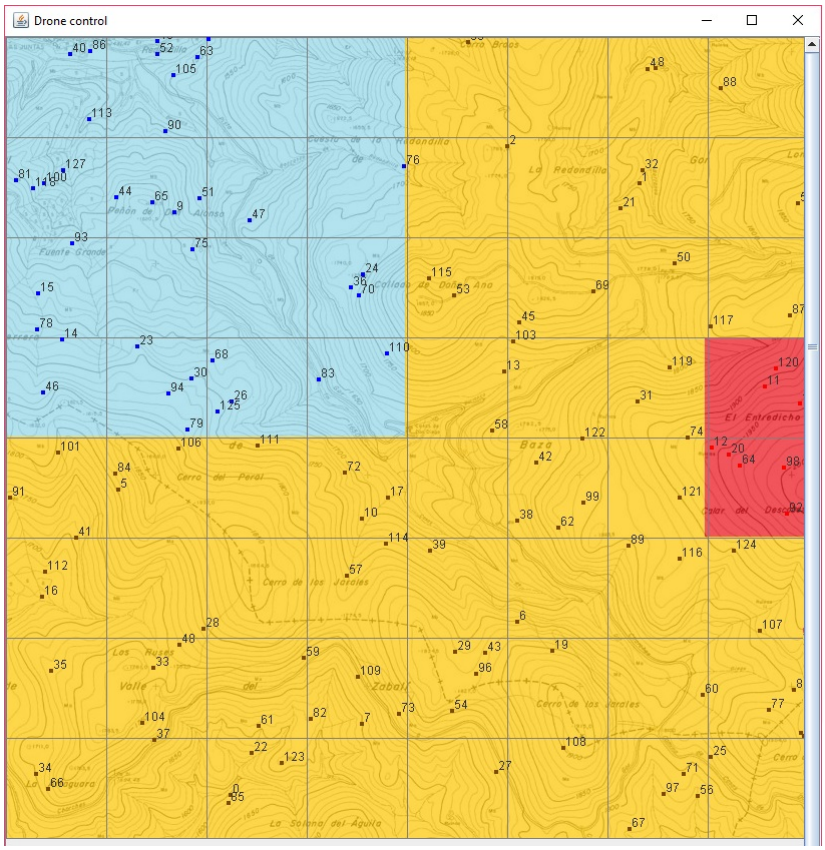

FIGURA 5.5: Mapa 2 en el instante inicial

```
HAY UN DRON INTRUSO (ID = 64) EN LA POSICIÓN X = 729.653408,
425.9512713
```

En el instante de tiempo t = 9.5 segundos, el programa detecta que el dron con identificador 106 se ha salido de la zona comercial y ha entrado en una zona de ocio.

```
HAY UN DRON COMERCIAL (ID = 106) FUERA DE SUS ZONAS PERMITIDAS.
POSICIÓN: 170.4506095, 399.5106928
```

Por último, unos segundos más tarde, en el instante t = 15.5 segundos, el dron 124 sale de su zona comercial para entrar en zona prohibida. Al ser un dron comercial, que radiodifunde su posición, no hay una dependencia del sistema de vigilancia de la zona prohibida para avisar del intruso: el programa detecta al recibir su posición que se encuentra en una zona prohibida, y así lo advierte.

```
HAY UN DRON COMERCIAL (ID = 124) FUERA DE SUS ZONAS PERMITIDAS.
POSICIÓN: 713.573571, 499.530928
```

Tras los 30 segundos de simulación, la pantalla de visualización gráfica tiene el siguiente aspecto:

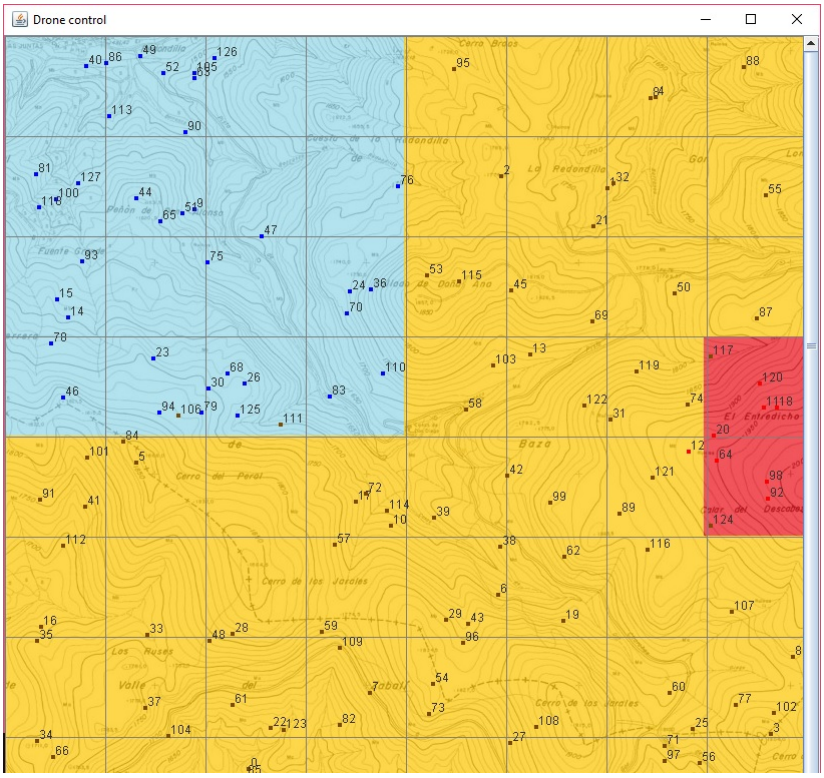

FIGURA 5.6: Mapa 2 tras la simulación

Capítulo 6

Historia del proyecto

En este capítulo se analizará el ritmo de realización del proyecto, es decir, la planificación del mismo, los problemas encontrados y las soluciones adoptadas ante estos problemas. Además, se detallará el coste del proyecto en cuanto a recursos humanos, de hardware y de software, para acabar obteniendo el presupuesto total.

6.1. Planificación

La planificación de un proyecto es esencial para poder aprovechar al máximo los recursos humanos y materiales disponibles y finalizar el proyecto en el tiempo previsto. Una buena planificación debe incluir un desglose de las actividades llevadas a cabo, así como la estimación del tiempo que se va a invertir en completarlas. Normalmente, también indica las personas designadas para ejecutar cada actividad pero, dado que este proyecto sólo cuenta con un desarrollador, ésto no se hará en la siguiente planificación.

Este proyecto tiene una duración estimada de 33 semanas y se realiza a tiempo parcial, dedicando el desarrollador un total de 20 horas a la semana a la ejecución del mismo. Por ello, para una mayor claridad, se indicará junto al desglose por días la duración de cada etapa en semanas.

En la tabla 6.1 se detallan las etapas de las que consta el proyecto junto a su duración. En la figura 6.1 se mostrará un diagrama de Gantt que, se espera, ayude a visualizar más fácilmente el contenido de la tabla.

Tarea	**Día de inicio**	**Día de finalización**	**Duración (semanas)**
Recopilación de bibliografía	08/02/2016	28/02/2016	3
Estudio de ADS-B	29/02/2016	6/03/2016	1
Instalación del software	7/03/2016	13/03/2016	1
Estudio de Apache Spark	14/03/2016	3/04/2016	3
Realización del diseño	4/04/2016	15/05/2016	6
Programación del prototipo	16/05/2016	10/07/2016	8
Planificación de las pruebas	11/07/2016	17/07/2016	1
Realización de las pruebas y análisis de resultados	18/07/2016	31/07/2016	2
Redacción de la memoria	1/08/2016	18/09/2016	7
Revisión de la memoria	19/09/2016	25/09/2016	1

CUADRO 6.1: Desglose por actividades para la planificación

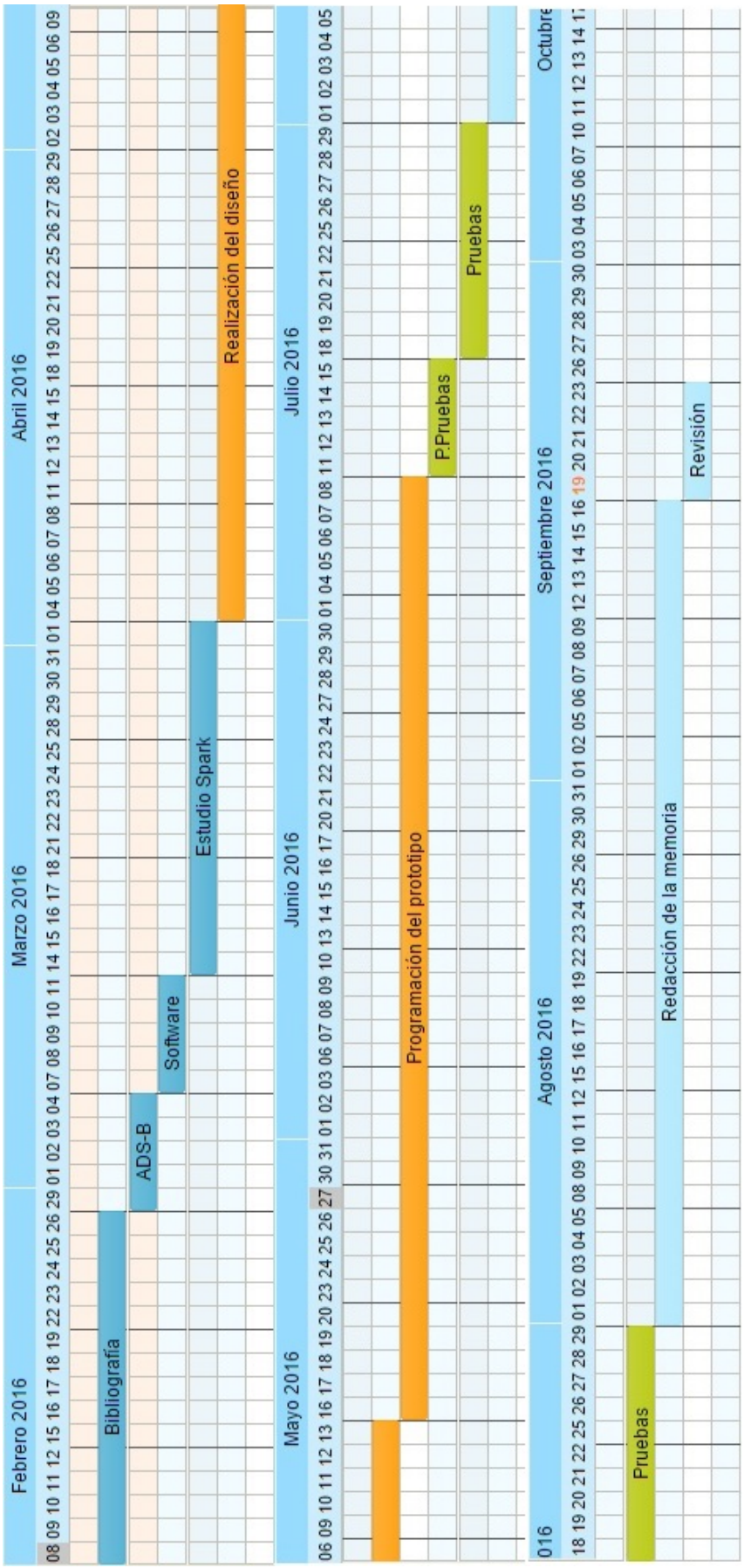

FIGURA 6.1: Diagrama de Gantt - Planificación del proyecto

6.2. Problemas encontrados

Durante la realización del prototipo y el diseño han surgido algunos problemas que han supuesto un obstáculo para el correcto desarrollo del proyecto. Entre ellos, cabría mencionar:

- **Ausencia de drones:** la ausencia de drones supuso un obstáculo en cuanto a que no se pudo contar con datos reales con los que probar el prototipo. No fue algo insalvable pues se diseñó un formato de datos (más detalles en el apartado 3.2), y se programó una clase cuyo único objetivo fuera generar archivos de texto con dicho formato (más detalles en el apartado 4.1.9), de manera que se consiguió simular el tráfico de datos entre los drones y el centro de control.
- **No existencia de transpondedores adaptados a drones:** en un principio, el único diseño de la arquitectura de red que se planteaba era aquel en el que el dron radiodifunde sus parámetros al centro de control por medio de un transpondedor. Sin embargo, como ya se discutió en el apartado 2.1.3, no existen por el momento transpondedores adecuados al tamaño y peso de estas pequeñas aeronaves. Al final, puesto que es factible que esto deje de ser un obstáculo en un futuro cercano, se optó por plantear dos modelos de arquitectura de red: el ya mencionado junto a otro que sí podría llevarse a cabo con la tecnología actual (más detalles en el apartado 3.1).

6.3. Recursos y presupuesto

En el presupuesto se incluyen todos los costes del proyecto, desde su inicio hasta su finalización. Para ello, se listarán primero los recursos utilizados:

Concepto	Tipo
Acer Aspire E5-522	Hardware
Windows 10 Home	Software
Apache Spark	Software
Eclipse Mars 2	Software

CUADRO 6.2: Recursos utilizados durante el proyecto

Tanto Apache Spark como Eclipse Mars 2 son herramientas de software gratuitas y de código abierto, por lo que no se les atribuye ningún coste. De este modo, los costes asignados a los recursos hardware y software quedan reflejados en la tabla 6.3.

En cuanto a los recursos humanos, como se ha mencionado con anterioridad, el número de horas trabajadas por el desarrollador cada semana

Concepto	Tipo	Coste (€)	Horas de vida	Horas dedicadas al proyecto	Coste total (€)
Acer Aspire E5-522	Hardware	369	6.480	660	37,58
Windows 10 Home	Software	94,95	6.480	660	9,67
TOTAL					**47,25**

CUADRO 6.3: Costes asociados a los recursos hardware y software

asciende a 20. Además, el supervisor del proyecto ha dedicado 2 horas cada semana a controlar y revisar el trabajo del desarrollador. La duración total del proyecto ha sido de 33 semanas. Si se asigna un coste de 20€/hora al desarrollador y de 25€/hora al supervisor, se pueden reflejar los costes asociados a los recursos humanos en la tabla 6.4.

Concepto	Coste (€/hora)	Horas dedicadas	Coste total (€)
Desarrollador	20	660	13.200
Supervisor	25	66	1.650
TOTAL			**14.850**

CUADRO 6.4: Costes asociados a los recursos humanos

Por último, se contemplan los gastos indirectos añadiendo el impuesto sobre el valor añadido en España (21 %).

Concepto	Coste (€)
Costes hardware y software	47,25
Costes de recursos humanos	14.850
Costes indirectos	3.128,42
TOTAL	**18.020,95**

CUADRO 6.5: Presupuesto total

Todo esto resulta en un coste total de **dieciocho mil veinte euros y noventa y cinco céntimos.**

Capítulo 7

Conclusiones y trabajos futuros

Este capítulo completa la memoria discutiendo las conclusiones que se extraen, tanto del resultado como de la experiencia personal tras la realización del proyecto. Además, se comentan ligeramente sus posibilidades de mejora de cara al futuro.

7.1. Conclusiones generales

En la introducción de este proyecto se ha tratado de evidenciar la importancia de la irrupción en masa de los drones en el ámbito civil, así como las consecuencias negativas que ello puede acarrear en un futuro cercano si a esta irrupción no le acompaña una evolución en las tecnologías de vigilancia aérea.

El principal objetivo de este proyecto era el de estudiar y diseñar una posible solución que permitiera vigilar con efectividad -y aprovechando las tecnologías de Big Data- el vuelo de los vehículos aéreos no tripulados. Esta solución estaría basada en la vigilancia ADS-B descrita en el capítulo 2, y debería permitir, además, detectar futuras colisiones entre los drones y ofrecer una solución a las mismas.

En líneas generales, puede decirse que el resultado es muy satisfactorio. El modelo diseñado es efectivo y realizable y, además, se ha conseguido desarrollar un prototipo usando el framework de Apache conocido como Spark. Este prototipo cumple con prácticamente todos los objetivos descritos en el apartado 1.2. En concreto:

- Permite dividir la carga de trabajo entre los diferentes nodos de un clúster, procesando la información de los drones en paralelo y siendo así capaz de manejar grandes cantidades de datos.
- Consigue clasificar los drones según su finalidad de uso.
- Divide el mapa en distintas zonas, en cada una de las cuales sólo pueden volar los drones de un determinado tipo. Consigue alertar al usuario si circula por una zona algún dron que no lo tiene permitido.

- Ofrece al usuario una interfaz gráfica sencilla donde visualizar las diferentes zonas del mapa y el movimiento de los drones a través de ellas distinguiendo, además, los drones de distintos tipos por colores.
- Puede detectar futuras colisiones entre dos o más drones, alertando de ello al usuario, e intenta ofrecer una solución.

Como aspecto mejorable, cabe destacar que debido a la falta de recursos, principalmente humanos y de tiempo, el algoritmo que ofrece solución a una futura colisión es algo tosco y, sin duda, superable. El programa calcula si podría evitarse el choque con la reducción de la velocidad de alguno de los drones involucrados. Sin embargo, no contempla otras opciones más complejas y, seguramente, más efectivas, como podría ser la posibilidad de que alguno de los drones varíe su trayectoria sin que cambie el punto al que se dirige.

Además, hubiera sido interesante ver funcionar el programa en un clúster real y con una carga de trabajo mucho mayor, para comprobar su rendimiento en unas condiciones más cercanas a la realidad. En cualquier caso, las pruebas demostraron que el prototipo, pese a sus limitaciones, funciona correctamente y en tiempo real. Esto es una gran noticia pues invita a pensar que, quizá con más tiempo y recursos materiales y humanos, la tecnología actual permite afrontar con firmeza los desafíos que nos presenta la vigilancia de la navegación aérea en el futuro más cercano.

7.2. Trabajos futuros

Pese a que los resultados del prototipo son satisfactorios, ya que cumplen la mayor parte de los objetivos planteados en la introducción, tiene también un importante margen de mejora.

Cabe mencionar que se han desarrollado algunos métodos orientados a facilitar algunos de estos futuros avances en el código. El método `anticipateBadDron` detecta cuando un dron va a salirse de la zona en la que tiene permitido circular en breves instantes de tiempo, si mantiene su trayectoria. Este método podría completarse, por tanto, para tomar medidas preventivas, como podría ser estrechar la vigilancia sobre dicho dron. Además, el método `GPStoXY` tampoco es utilizado actualmente en el prototipo, pero podría ser muy útil en el futuro. Está pensado para un escenario en el que los drones transmiten su posición en el sistema de coordenadas geográficas. Como se explica en el apartado 4.1.10, el método se encarga de transformar esas coordenadas geográficas en una celda o píxel del mapa en el que se va a representar gráficamente el vuelo de los drones.

Por último, sería interesante desarrollar o utilizar algún algoritmo de evitación de colisiones entre drones más eficaz que el utilizado en el prototipo actual.

7.3. Conclusiones personales

Personalmente, estoy muy contento con el desarrollo y el resultado de este proyecto. Durante estos meses, he tenido la oportunidad de investigar dos tecnologías que están llamadas a revolucionar el futuro, como son los vehículos aéreos no tripulados y el concepto de Big Data.

Estoy particularmente agradecido por haber podido profundizar en la programación utilizando Spark, que es un framework relativamente nuevo y sobre el que todavía hay mucho desconocimiento pero que, a su vez, es una de las herramientas más importantes del momento en cuanto al análisis de grandes cantidades de datos se refiere. Por supuesto, siempre hay cosas que aprender y mejorar, pero creo que esta experiencia ha sido realmente positiva y una gran introducción a un ámbito muy interesante de la ingeniería.

Capítulo 8

Resumen extendido

8.1. Introduction

A principios de este siglo, los UAV (del inglés *Unmanned Aircraft Vehicles*), más conocidos como drones, eran una tecnología de uso casi exclusivamente militar. Tras la guerra de Iraq y Afganistán, los drones comenzaron a ganar fama entre el público en general, y fue entonces cuando se empezó a plantear la posibilidad de aplicar estos avances a la esfera civil. Hoy en día, los UAV han revolucionado diversas áreas como la agricultura, la fotografía, el salvamento o la vigilancia, entre muchas otras. Estos avances han venido acompañados, por desgracia, de algunas desventajas. Algunas de las más importantes podrían ser:

- El espacio aéreo se ve amenazado: en los últimos meses son numerosos los casos en los que un avión ha estado cerca de colisionar con un dron, generalmente en las inmediaciones de un aeropuerto.
- Peligro en zonas pobladas: muchos de estos aparatos pueden superar los 25kg y poseen unas afiladas hélices, por lo que suponen un grave peligro en caso de caída sobre un ciudadano o una estructura sensible.
- Amenaza para la privacidad: su reducido tamaño, su gran maniobrabilidad y su capacidad para portar cámaras de alta definición los convierten en un espía perfecto.

A día de hoy estos problemas, aunque no son despreciables, no son todo lo graves que podrán llegar a ser en un futuro en el que, probablemente, millones de drones surquen los cielos cada día. Este proyecto nace con el objetivo de presentar una solución a todos los desafíos que plantea la irrupción en masa de los UAV en el ámbito civil. Por eso, se quiere diseñar un sistema software capaz de:

- Trabajar con Big Data, es decir, responder a grandes cantidades de datos prácticamente en tiempo real.
- Clasificar zonas en un mapa según el tipo de drones que puedan circular por ellas, y clasificar a los drones por tipos según las zonas en las que puedan circular.

- Vigilar que ningún dron circule por zonas en las que no lo tiene permitido.
- Detectar y evitar futuras colisiones entre drones.
- Diseñar una interfaz gráfica sencilla para el usuario.

8.2. Estado del arte

8.2.1. ADS-B

ADS-B es nuevo sistema de vigilancia aérea llamado a sustituir a la tradicional vigilancia por radar. Se basa en un concepto muy simple: en lugar de tratar de detectar una aeronave, la propia aeronave difundirá su posición y otros parámetros a todas las demás aeronaves y a los sistemas en tierra preparados para recibirlos. Esta difusión será automática, periódica y regular y se realizará mediante un transpondedor. Las ventajas de este sistema son numerosas, pues ofrece una mayor seguridad (la información es más precisa) a menor coste, más eficiencia y, consecuentemente, más capacidad en el espacio aéreo.

Su implementación está pensada para aeronaves como aviones o avionetas: no funcionaría en drones debido principalmente al peso y dimensiones de un transpondedor, considerables para un aparato pequeño como es un UAV. Sin embargo, ya hay empresas trabajando en conseguir transpondedores de un tamaño muy reducido y con un gasto en potencia muy bajo, por lo que es probable que en un futuro cercano podamos ver este sistema aplicado también a los drones.

8.2.2. Big Data y Apache Hadoop

Big Data es un término nuevo y en constante evolución, por lo que no es fácil de definir. Se podría que es, al menos, cualquier conjunto de datos con potencial de ser aprovechado para la extracción de información y cuyo volumen o complejidad es tal que dicha extracción no puede abarcarse con los métodos tradicionales de análisis. A principios de siglo, el analista Doug Laney caracterizó este concepto mediante las llamadas 3 V's:

- Volume: hace referencia al gran tamaño de estos conjunto de datos.
- Velocity: hace referencia a la rapidez con la que se generan estos datos y, sobre todo, a la necesidad de dar respuesta a estos datos con rapidez.
- Variety: se refiere a la naturaleza de los datos que se manejan. Muchas veces no poseen una estructura ni se recogen en los formatos tradicionales.

Para analizar Big Data, como se ha explicado anteriormente, no son útiles los sistemas tradicionales. A la vez que los expertos fueron asimilando este concepto, tuvieron que nacer nuevas formas de tratar con este tipo de datos. En 2004, Google presentó MapReduce, nacido en sus oficinas para digerir la gran cantidad de datos que generaban diariamente. Map Reduce es un paradigma de programación con una filosofía muy simple: un problema complejo puede ser solucionado de forma más eficiente dividiéndolo en varios problemas más sencillos y resolviéndolos en paralelo. El término describe dos tareas secuenciales: primero se realiza un mapeo en el que se transforma un conjunto de datos en otro en el que los elementos individuales se convierten en tuplas (clave/valor). Tras esto, se realiza una función de reducción para convertir el conjunto de tuplas en otro más pequeño.

De esta forma, se consigue aprovechar la computación distrubuida de modo que un conjunto de servidores básicos trabajan juntos para formar, virtualmente, un servidor de gran potencia. MapReduce funcionaba realmente bien, pero tenía el inconveniente de requerir de unos una estructura tecnológica al alcance de pocos. Para solucionar esto, nacía Hadoop, un framework de código abierto que permite al usuario medio disfrutar de las ventajas de MapReduce sin tener que contar con los recursos que posee Google.

De forma simplificada, *Hadoop* tiene dos partes principales: un framework que procesa los datos y un sistema de archivos distribuidos para almacenarlos (por defecto, el HDFS). El framework utiliza *MapReduce*, que lanza una serie de aplicaciones -escritas en Java- mediante las cuales se recorren los datos y se extrae información de ellos.

8.2.3. Apache Spark y los RDD

Hadoop funciona realmente bien y es, incluso, una solución óptima cuando no se requiere de una velocidad especialmente alta a la hora de analizar los datos. Sin embargo, en ciertas aplicaciones sí es necesario analizar y dar respuesta a los datos que recibes prácticamente en tiempo real. Spark es un framework, también de código abierto y también publicado por Apache Software Fundation, que puede llegar a ser hasta 100 veces más rápido que Hadoop cuando se ejecuta en memoria y hasta 10 veces más rápido cuando se procesa en disco[17]. En general, tanto Hadoop como Spark pueden ser usados para procesar información en lotes (el primero en dos etapas, map y reduce, y el segundo en tantas como se necesite, disponiendo de una amplia caché) pero, además, gracias a Spark Streaming, es capaz de dar analizar datos en tiempo real como hacen otras soluciones especializadas.

Spark está construido y trabaja sobre el concepto de los RDD (Resilient Distributed Datasets). Formalmente, un RDD es un conjunto de elementos particionado, distribuido e inmutable. Del nombre, podemos extraer:

- **Resilient** (elástico, resistente): RDD es tolerante a fallos, es capaz de recomponer partes perdidas o dañadas debido al fallo de algún nodo.

- **Distributed** (distribuido): los datos residen en varios nodos dentro de un clúster.
- **Dataset** (conjunto de datos): es una colección de datos (primitivos, tuplas, objetos...)

Así que se podría decir que un RDD es un conjunto de datos, los cuales se parten y se distribuyen y procesan en distintos nodos de un clúster. Cabe mencionar que estos datos son mantenidos en memoria (caché) siempre que se puede y tanto tiempo como sea posible. Además, los RDD son de evaluación perezosa: la creación de un RDD no se *'materaliza'* realmente hasta que aplicamos una acción sobre el mismo.

En definitiva, cualquier trabajo en Spark se expresa mediante las siguientes actividades:

- Definiendo un RDD, para lo cual podemos paralelizar una colección en memoria (por ejemplo, un archivo de texto) o referenciar datos de una fuente de almacenamiento externa, como HDFS.
- Realizando transformaciones sobre un RDD existente, lo cual produce otro RDD. Por ejemplo, en un mapeo, se aplica una función a cada elemento de un RDD y se obtiene un nuevo RDD con el resultado.
- Llevando a cabo acciones. Las acciones son operaciones que se aplican sobre un RDD para obtener un resultado. Por ejemplo, contar el número de elementos que contiene un RDD es una acción, y su resultado es un valor numérico.

8.2.4. LIDAR

El **lidar** (Laser Imaging Detection And Ranging) es una tecnología de teledetección (detección a distancia) que funciona mediante la emisión de un haz láser pulsado hacia un objeto o entorno y la medición del tiempo que tarda en volver el reflejo hacia el emisor. Funciona de manera similar al radar, pero con ondas de luz visible en lugar de ondas de radio. Además, presenta ciertas ventajas con respecto a éste, pues es notablemente más económico, rápido y fácil de transportar.

8.3. Diseño

A continuación, se describen las principales características del diseño del software planteado como solución a los desafíos que presenta la presente y futura irrupción de los drones en el mercado.

8.3.1. Arquitectura de red

No existe ningún motivo por el que el diseño tenga que estar reñido con una arquitectura de red concreta. Se presentan a continuación dos posibilidades:

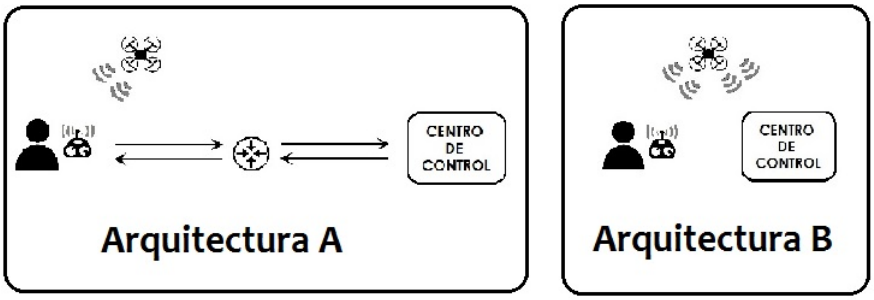

FIGURA 8.1: Posibles arquitecturas de red

- Arquitectura A: en este modelo, el dron comunica su posición al piloto, como es habitual. Es éste el encargado de hacer llegar los parámetros de su aeronave al centro de control.
- Arquitectura B: irrealizable en el momento, por la no existencia de transpondedores adecuados para drones, esta solución se convertirá en la mejor opción en un futuro cercano. El dron difunde sus parámetros a todos los dispositivos cercanos preparados para recibirlos.

8.3.2. Formato de los datos

Ya que no existe un estándar sobre los datos que informan de los parámetros del dron, se asume uno para este diseño:

```
id, posX, posY, vel, destX, destY, tipo, priority
```

En los que se representan, por orden, el identificador, la posición (en sus componentes cartesianas), la velocidad y el punto de destino (en sus componentes cartesianas) del dron, en caso de que exista, el tipo y la prioridad del dron.

8.3.3. Estructura y división del mapa

Para el funcionamiento interno del programa, de ahora en adelante, se llamará **mapa** a la totalidad del espacio bidimensional que vigila el sistema. Cada mapa se divide en varios **planos**, cada uno de los cuales es procesado

por un nodo del sistema. Además, cada plano se divide en pequeñas **celdas** de 1 píxel x 1 píxel, las cuales se usarán, entre otras cosas, para detectar choques, pues el programa no aceptará que dos o más drones se encuentren en la misma celda durante el mismo instante de tiempo.

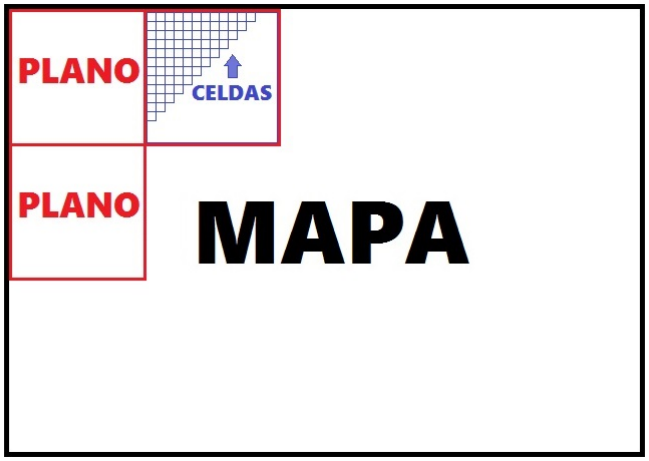

FIGURA 8.2: Diferencia entre mapa, plano y celda

Además, ya que no todos los drones ni las motivaciones para su uso son iguales, el mapa se divide en tres zonas, en cada una de las cuales sólo podrán volar los drones de un determinado tipo. De este modo, sólo los drones comerciales tendrán permitido circular por la **zona comercial**, sólo los drones de uso recreativo podrán circular por la **zona de ocio** y **ningún** dron podrá circular dentro de una **zona prohibida**. En las zonas prohibidas deberán existir sistemas de teledetección como el LIDAR descrito en la sección 8.3 de este resumen, para detectar posibles drones que, intencionadamente, estén manipulados para no comunicar sus parámetros.

8.3.4. Objetivos y ejecución

El programa deberá ser capaz de detectar si cualquier dron ha salido o va a salirse de sus zonas permitidas en breves instantes en caso de mantener su trayectoria. Por supuesto, también deberá avisar a las autoridades competentes si existe algún dron circulando en una zona prohibida.

Además, deberá detectar las posibles colisiones entre drones que circulen en un mismo plano, ofreciendo una solución a dicho choque siempre que sea posible. Para ello, se calculará si la reducción de la velocidad del dron con menor prioridad evitaría la colisión sin que esto incurra en problemas con otros drones del plano.

Por último, se deberá proveer de una interfaz gráfica sencilla para el usuario para visualizar el mapa, sus zonas y los drones que circulan por ellas.

La ejecución se llevará a cabo en un sistema distribuido que utilice Spark, en el cual cada plano es manejado por un nodo distinto de forma paralela. Estos planos pueden variar de tamaño, de manera que en las zonas con más concentración de drones puedan existir planos más pequeños de lo habitual, estando así la zona vigilada por un mayor número de nodos, mejorando de este modo la fluidez del programa.

8.4. Prototipo

Por falta de recursos, tanto humanos como económicos y materiales, además de la limitación de tiempo para llevar a cabo este proyecto, el diseño presentado no puede realizarse en su totalidad. En su lugar, se ha desarrollado un prototipo que, se espera, demuestre a pequeña escala como funcionaría la idea descrita.

Los datos enviados por los drones, en el formato descrito en el apartado 8.4.2, son la fuente de información a la que acceden **todos** los nodos del clúster. De este modo, cada nodo busca los drones que se encuentran dentro de los límites del plano que está vigilando, y realiza con sus datos las operaciones necesarias.

El prototipo cuenta con diferentes clases programadas en Java:

8.4.1. Clase `Plane`

La clase `Plane` define un plano. Para ello, indica sus límites en las coordenadas cartesianas, el número de drones que circulan dentro de las coordenadas del plano y el número de zonas comerciales, de ocio y prohibidas que coexisten dentro del plano. Además, almacena estos drones y estas zonas (en forma de objetos) en diferentes listas.

Por último, contiene los métodos necesarios para agregar y eliminar elementos a la lista de drones, así como los necesarios para agregar elementos a las listas de zonas.

8.4.2. Clases `leisureZone`, `commercialZone` y `dangerZone`

Estas clases definen, respectivamente, una zona de ocio, una zona comercial y una zona prohibida. Para ello, simplemente se indican los límites de estas zonas en sus componentes cartesianas.

8.4.3. Clase `Dron`

La clase `Dron` sirve para que el programa pueda manejar de forma eficiente los parámetros que envían los drones. Cuando se reciben, éstos son encapsulados en forma de objetos `Dron` para después operar con ellos.

Los atributos de la clase incluyen la posición del dron y la celda en la que se encuentra, su velocidad, su punto de destino y las celdas en las que se encontrará en los futuros 40 instantes de tiempo, si mantiene su velocidad y trayectoria, todo ello en las componentes cartesianas. Además, incluye atributos para señalar el identificador, el tipo y la prioridad que posee cada dron.

8.4.4. Clases `MyFrame`, `MyPanel` y `MyRectangle`

Estas tres clases heredan, respectivamente, de las clases de Swing (biblioteca gráfica) de Java `JFrame`, `JPanel` y `Rectangle`. Modifica el `paintComponent` de estas clases para diseñar el mapa sobre el que circularán los drones, el marco en el que se encuentra dicho mapa y los rectángulos que representan a cada uno de los drones. Se podrán distinguir los límites de cada plano y los distintos tipos de drones según su color, del mismo modo que ocurre con las zonas.

8.4.5. Clase `crearFicherosdeTiempo`

Esta clase se encarga de simular el tráfico de datos entre los drones y el sistema de control. Para ello, inicializa datos de drones al azar y crea archivos de texto con estos datos siguiendo el formato especificado en el apartado 8.4.2.

8.4.6. Clase `Main`

La clase `Main` es la más extensa y compleja, y es donde se ejecuta realmente el proyecto. Cuenta con diferentes atributos y métodos orientados a realizar las operaciones necesarias para la vigilancia de drones, como la detección de colisiones entre drones o la salida de algún dron de sus zonas permitidas. Para ello, almacena los distintos planos que componen el mapa en un RDD, de manera que estos planos sean distribuidos entre los nodos del clúster. Como los RDD son inmutables, cada vez que la colección de planos tiene que cambiar (cada vez que llegan nuevos datos de drones) se realiza un mapeo, con el que se obtiene un nuevo RDD que sobrescribe al anterior. Para una explicación más detallada acerca del funcionamiento de esta clase, se recomienda la lectura del apartado 4.1.10 del documento.

El funcionamiento esquemático de la clase es el indicado en la figura 8.3.

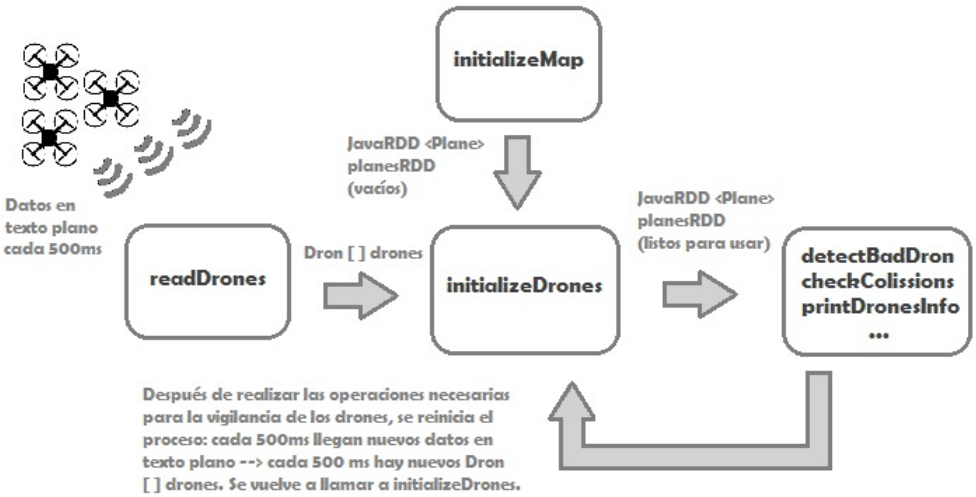

FIGURA 8.3: Funcionamiento esquemático de la clase `Main`

8.5. Conclusiones

Tras comprobar el correcto funcionamiento del prototipo en las pruebas (para más detalles, se recomienda consultar el capítulo 6) puede decirse que el resultado es muy satisfactorio. El modelo diseñado es efectivo y realizable, y satisface prácticamente todos los objetivos planteados en la introducción. Como aspecto mejorable y como línea de trabajo a seguir en un futuro, sería recomendable mejorar el algoritmo que ofrece solución a la colisión entre dos o más drones. Sin embargo, puede afirmarse que el programa, pese a sus limitaciones, funciona correctamente y en tiempo real. Esto es una gran noticia pues invita a pensar que, quizá con más tiempo y recursos materiales y humanos, la tecnología actual permite afrontar con firmeza los desafíos que nos presenta la vigilancia de la navegación aérea en el futuro más cercano.

Personalmente, estoy muy contento con el desarrollo del proyecto y, particularmente, por haber tenido la oportunidad de profundizar en la programación utilizando Spark, una de las herramientas más importantes del momento en cuanto a análisis de grandes cantidades de datos se refiere. Por supuesto, siempre hay cosas que aprender y mejorar, pero creo que esta experiencia ha sido realmente positiva y una gran introducción a ámbito muy interesante de la ingeniería.

Apéndice A

Introduction (traducción al inglés del capítulo 1)

A.1. Motivation of this project

Earlier this century UAVs (*Unmanned Aircraft Vehicle*) or drones were were an almost exclusively militar technology, with some interest for plane-modelers, but unknown to the public in general. The great revolution in the market for these aircraft came probably after the attacks of September 11, 2001 in the United States and the subsequent wars in Iraq and Afghanistan, where the drones played a key role in the strategies of the USA and its allies.

FIGURA A.1: Dron *Predator*, desarrollado en los 90 por la Fuerza Aérea estadounidense

In late 2002, the USA army had approximately 200 drones [1]. Today, the number exceeds 11,000 and continues to rise. The demand for this technology by the major military powers generated such an industrial and scientific *boom* that it was not surprising that companies began to be interested in

bringing these advances to the rest of society. According to Teal Group estimates [2], the introduction of drones in the civilian sphere could generate a profit of 11,600 million in 2023 to the manufacturers.

So, just as happened with other breakthroughs such as radio or GPS, an invention that was born to revolutionize the military environment ended up transforming various areas of civil society. Today, drones have changed a lot of industries, and they have opened new possibilities that apply to research, conservation, agriculture, photography, rescue, infrastructure, search and rescue of victims, among many other things.

FIGURA A.2: Dron realizando funciones agrícolas

However, not all are advantages. The mass emergence of drones also causes a number of problems, mostly related to the lack of strong regulation. Some of these problems could be:

- **The airspace is threatened:** there are increasingly frequent news in which the safety of passengers and crew of a commercial aircraft is threatened by the emergence of drones in the vicinity of some airports when the plane was about to land, hapenning in the Charles de Gaulle airport [3] in February this year, or more recently, in August, at the airports in Munich [4] and Cornwall **cornualles**
- **Danger in populated areas:** Aitor Goikoetxea, director of Drone School, where classes to fly this type of apparatus are taught, said in an interview last year: «Drones are not toys. They are devices that can reach 25 kilos and, if used in a way that is not appropriate, it can cause a disgrace». These statements came after a neighbor in Zestoa, Guipuzcoa, suffered very serious facial injuries after being hit by a drone piloted by an amateur [6].
- **Threat to privacy:** Today there are very small drones within the reach of the average user capable of flying over areas taking high quality pictures and videos. Yards, terraces, schools, private parcels and even home security can now be broken due to misuse of these small aircraft. Furthermore, if we add to this the latest advances in storage

and image processing, it is clear that the privacy of citizens is facing unprecedented risk.

Today these problems are not negligible, but they have not achieved its maximum dimension. But at a time when the drones have become the favorite christmas present in hundreds of thousands of homes and the favourite tool of many companies of all kinds, it can be imagined a future in which millions of these devices may sail the skies every day . The current mechanisms for locating aircraft and collision avoidance are not applicable for existing drones, mainly due to its small size and its maneuverability. The control of this type of air traffic recquire therefore new and more sophisticated technology than that currently available.

A.2. Social and economic environment

As well as traditional aviation had its roots in that science and technology that emerged during the industrial era, after the transformations of the nineteenth and early twentieth centuries, the emergence and development of drones have their origin in the Information Society.

	2015
Infrastructure	45.2
Transport	13.0
Insurance	6.8
Media & Ent.	8.8
Telecommunication	6.3
Agriculture	32.4
Security	10.5
Mining	4.3
Total	127.3

FIGURA A.3: Value of the bussiness oportunities generated by drones, in billion dollars - Source: PwC

Its growth and popularity are due to the confluence of several aspects, among which could be highlighted the change from military strategies to intelligence, reconnaissance and surveillance operations, the proliferation of suitable scenarios for the use of this technology and the great advances in miniaturization and autonomy of systems.

As mentioned in the introduction, the revolution of drones is transforming businesses and industries of the most various kinds, from agriculture

to the film sector. According to the report by PwC, *Clarity from above* [7], the emerging market related to the use of drones can generate business opportunities for a total value of more than 127 billion dollars. The sectors which, predictably, most can benefit from the rise of these aircraft would be infrastructure, agriculture and transport.

A.3. Objectives

The main objective of this project is the study and design of a possible solution to the challenges that present and future mass emergence of drones in the market poses.

To fulfill this purpose, some software it is designed and implemented that must meet another set of objectives. The software should be able to:

1. Take advantage of the benefits of distributed computing and Big Data, as it will work, predictably, with a lot of data from drones in real time. To do this, it must divide the monitoring of various portions of the map among the nodes of a cluster of machines.
2. Sort drones according to their intended use.
3. Delineate areas where only a particular type of drone can circulate.
4. Watch drones that circulate in the areas where they are allowed, and notify the relevant authorities if they do not.
5. **Detect** future collisions between drones circulating in the same portion of the map, based on their trajectories. Offer a solution to avoid such collisions.
6. Design and implement a user-friendly interface in which:
   a) The movement of drones circulating around the map can be observed in real time.
   b) The different areas that make up the map can be distinguished and drones can be differentiated according to their intended use.
   c) The portions of the map that each of the nodes are going to control can be distinguished.
7. Make way for future expansion of the project.

A.4. Contents of this report

To help in reading this document, a brief summary of each of the chapters that compose it are provided below:

A.4.1. Chapter 1: Introduction:

In the introduction it is discussed the motivation of this project and its objectives are briefly explained.

A.4.2. Chapter 2: State of the art:

In this chapter the most important aspects will be presented to understand the surveillance system ADS-B, which is the basis on which this project is inspired, as well as the birth of the concept of Big Data and the two most important frameworks of Apache to deal with this new type of data: Hadoop and Spark.

A.4.3. Chapter 3: Design:

In this chapter the scheme of operation of the control system of drones is introduced and the requirements that it should meet are briefly described.

A.4.4. Chapter 4: Prototype:

In this chapter the prototype of the distributed control system is described. Therefore, a detailed description of each of the classes that make up the program is provided.

A.4.5. Chapter 5: Testing:

In this chapter there are some tests to check the correct operation of the prototype and the results are described and discussed.

A.4.6. Chapter 6: Project history:

In this chapter the main problems of the project and the solution to them are remarked. In addition, planning and budget of the project is presented.

A.4.7. Chapter 7: Conclusions and future lines of work:

This chapter completes the project report, presenting the conclusions drawn from both the result and the personal experience of the project. Moreover, its potential for improvement in the future is slightly discussed.

A.4.8. Chapter 8: Extended summary:

This chapter provides an extended summary of the report, highlighting the most important in terms of motivation and objectives of the project, the state of the art, designing a solution to the objectives and its prototype, as well as the conclusions drawn of the project.

A.4.9. Appendix A: Introduction:

This appendix contains the english translation of the introductory chapter to the project.

A.4.10. Appendix B: Conclusions and future lines of work:

This appendix contains the english translation of the concluding chapter of the project.

A.4.11. Appendix C: Extended summary:

This appendix contains the english translation of the extended summary presented in Chapter 8.

A.4.12. Appendix D: Code of the prototype:

This appendix indicates the url where the code of the prototype can be found.

Apéndice B

Conclusions and future lines of work (traducción al inglés del capítulo 7)

This chapter completes the memory discussing the conclusions drawn from both the result and the personal experience after completion of the project. Moreover, their potential for improvement in the future is slightly discussed.

B.1. General conclusions

The introduction of this project has been conceibed to highlight the importance of the massive arrival of drones in civil matters as well as the negative consequences this may have in the near future if this irruption is not accompanied by an evolution in aerial surveillance technologies.

The main objective of this project was to study and design a possible solution that would allow effective monitoring flying unmanned aerial vehicles, taking advantage of the new Big Data technologies. This solution would be based on ADS-B surveillance, described in Chapter 2, and should allow also to detect future collisions between drones and offer a solution to them.

In general, we can say that the result is very satisfactory. The model is designed and works effectively and a prototype has been developed using the framework known as Apache Spark. This prototype meets virtually all the objectives described in paragraph 1.2. Specificly:

- It can divide the workload between different nodes in a cluster, processing information in parallel drones and thus being able to handle large amounts of data.
- It gets drones classified according to their purpose of use.

- It divides the map in different areas, in each of which can only fly the drones of a certain type. It alerts the user if a drone drives into an area that is forbidden for it.
- It offers the user a simple graphical interface where the different areas of the map and the movement of drones through them can be visualized, also distinguishing different types of drones by its color.
- It is able to detect future collisions between two or more drones, alerting it to the user, and it attempts to offer a solution.

As improvable aspect, it is worth noting that due to lack of resources, especially human resources and time, the algorithm that provides solution to a future collision is somewhat crude and certainly surmountable. The program calculates if the collision could be avoided by reducing the speed of any of the drones involved. However, it does not provide more complex and maybe more effective options, such as the possibility that any of the drones varies its path without changing the point to which it is addressed.

In addition, it would have been interesting to see how the program runs in a real cluster and with a much greater work load to check its performance in a closer-to-reality conditions. In any case, the evidence showed that the prototype, despite its limitations, works correctly and in real time. This is great news for us: to think that perhaps with more time and material and human resources, current technology allows firmly confront the challenges that monitoring air navigation brings in the nearest future.

B.2. Future lines of work

Although the results of the prototype are satisfactory, because they meet most of the goals outlined in the introduction, it has also a significant room for improvement.

It is noteworthy that some methods have been developed in order to facilitate some of these future advances in the code. The `anticipateBadDron` method detects when a drone is going to get out of the area that it is allowed to circulate in if it keeps its path, some time before it actually does. This method could be completed, therefore, to take preventive measures, such as might be keeping a closer eye on that drone. In addition, the `GPStoXY` method is not currently used in the prototype, but it could be very useful in the future. It is designed for a scenario in which the drones transmit their position in the geographic coordinate system. As explained in section 4.1.10, the method is responsible for transforming these geographic coordinates in a cell or a pixel the in map in which the flight of drones will be graphically represented.

Finally, it would be interesting to develop or use a collision avoidance algorithm between more effective than the one used in the current prototype.

B.3. Personal conclusions

Personally, I am very happy with the development and outcome of this project. During these months, I have had the opportunity to investigate two technologies that are destined to revolutionize the future, such as unmanned aerial vehicles and the concept of Big Data.

I am particularly grateful for having been able to develop my programming skills using Spark, which is a relatively new framework and on which there is still much to discover, but it is also one of the most important tools of the moment on the analysis of large amounts of data. Of course, there are always things to learn and improve, but I think this experience has been really positive and a great introduction to a very interesting field of engineering.

Apéndice C

Extended summary (traducción al inglés del capítulo 8)

C.1. Introduction

Earlier this century, UAVs (*Unmanned Aircraft Vehicles*), better known as drones, were technology almost exclusively for military use. After the war in Iraq and Afghanistan, drones began to gain fame among the general public, and that was when it began to raise the possibility of applying these advances to the civil sphere. Today, UAVs have revolutionized diverse areas as agriculture, photography, rescue or surveillance, among many others. These advances have been accompanied, unfortunately, by some disadvantages. Some of the most important would be:

- The airspace is threatened: in recent months there have been numerous cases in which a plane has been close to colliding with a drone, usually in the vicinity of an airport.
- Danger in populated areas: many of these devices can exceed 25kg and have a sharp propellers, which pose a serious danger in case of a fall on a citizen or a sensible structure.
- Threat to privacy: its small size, its maneuverability and its ability to carry high-definition cameras make them perfect spies.

Today these problems, although not negligible, are not as serious as they may become in a future in which, probably, millions of drones will sail the skies every day. This project was created with the aim of presenting a solution to all the challenges that mass emergence of UAVs in the civil sphere has brought. So it is desired to design a software system capable of:

- Working with Big Data, this is, respond to large amounts of data in near real time.
- Sort areas on a map depending on the type of drones that can move in them, and classify the drones by types depending on the area where they can circulate.

- Watch that no drone circulate through areas where it is not allowed.
- Detect and avoid future collisions between drones.
- Design a simple user-friendly graphical interface.

C.2. State of the art

C.2.1. ADS-B

ADS-B is new aerial surveillance system designed to replace traditional radar surveillance. It is based on a very simple concept: instead of trying to detect an aircraft, the aircraft itself will broadcast its position and other parameters to all other aircraft and ground systems prepared to receive them. This distribution will be automatic, periodic and regular basis and it will be made by a transponder. The advantages of this system are numerous, offering greater security (the information is more accurate) at a lower cost, it is more efficient and, consequently, it allows more capacity in the airspace.

Its implementation is intended for vehicles such as planes or light-aircraft: it would not work on drones mainly due to the weight and dimensions of a transponder, which is significant for a small device such as a UAV. However, there are already companies working on getting transponders of a very small size and a very low spending power, so it is likely that in the near future we can see this system also applied to the drones.

C.2.2. Big Data and Apache Hadoop

Big Data is a new term and it is constantly evolving, so it is not easy to define. It could be said that it is, at least, any dataset with potential to be exploited for the extraction of information, and with a volume or complexity is such that this extraction can not be covered with traditional methods of analysis. At the beginning of the century, the analyst Doug Laney characterized this concept by the so-called 3 V's:

- Volume: refers to the large size of the data set.
- Velocity: refers to the speed with which these data are generated and, above all, the need to respond quickly to these data.
- Variety: it refers to the nature of the data being handled. Often they do not have a structure and they cannot be found in traditional formats.

To analyze Big Data, as explained above, traditional systems are not useful. While experts were assimilating this concept new ways of dealing with this type of data had to be born. In 2004, Google introduced MapReduce, born in their offices to digest the large amount of data generated daily.

MapReduce is a programming paradigm with a very simple philosophy: a complex problem can be solved more efficiently by dividing it into several simpler problems and solving them in parallel. The term describes two sequential tasks: first a mapping in which a set of data turns into another in which the individual elements become tuples (key / value). After this, a reduction function is performed to convert the set of tuples in another smaller set.

In this way, you get to take advantage of distributed computing so that a basic set of servers are able to work together to virtually form a powerful server. MapReduce worked really well, but had the disadvantage of requiring a technological structure available only to few. To fix this, Hadoop was born: an open source framework that allows the average user to enjoy the benefits of MapReduce without having to rely on the resources owned by Google.

In a simplified form, *Hadoop* has two main parts: a framework that processes the data and a distributed system to store files (by default, HDFS). The framework uses *MapReduce*, launching a series of Java-written applications by which the data is run through and information is extracted from them.

C.2.3. Apache Spark and RDD

Hadoop works really well and is even an optimal solution when you do not require a particularly high speed when analyzing the data. However, in certain applications it is necessary to analyze and respond to data that you receive near real-time. Spark is a framework, also open source and published by the Apache Software Foundation, which can be up to 100 times faster than Hadoop when running in memory and up to 10 times faster when processed disk [17]. In general, both Hadoop and Spark can be used to process information in batches (the first one in two stages, map and reduced, and the second one in as many as needed, having also a large cache) but also, thanks to Spark Streaming, it is able to analyze data in real time as other specialized solutions do.

Spark is built on the concept of RDD (Distributed Resilient Datasets). Formally, a RDD is a set of elements which is partitioned, distributed and immutable. From the name, we can draw:

- **Resilient:** RDD is fault-tolerant, it is able to reconstruct missing or damaged parts due to the failure of some node.
- **Distributed:** Data resides on multiple nodes within a cluster.
- **Dataset:** It is a collection of data (primitive, tuples, objects...)

So you could say that an RDD is a set of data which is split and it is distributed and processed on different nodes of a cluster. It is noteworthy

that these data are kept in memory (cache) as long it is able and as long as possible. In addition, the RDD are lazy evaluated: the creation of an RDD is not *really 'done'* until we apply an action on it.

In short, any work in Spark is expressed by the following activities:

- Defining a RDD, for which we can parallelize a collection in memory (for example a text file) or reference data from an external storage such as HDFS.
- Performing transformations on an existing RDD, which produces another RDD. For example, a mapping function is applied to each element of a RDD and a new RDD and given with the result.
- Taking actions. Actions are operations that are applied to an RDD to get a result. For example, counting the number of elements contained in a RDD is an action, and the result is a numerical value.

C.2.4. LIDAR

The **LIDAR** (Laser Imaging Detection And Ranging) is a remote sensing technology that works by emitting a pulsed laser beam to an object or environment and measuring the time it takes to the reflection to return to the sender. It works similar to radar, but with visible light waves instead of radio waves. Furthermore, it has certain advantages with respect to radar, since it is considerably more economical, faster and easier to carry.

C.3. Diseño

The following chapter describes the main features of the design of the software proposed as a solution to the challenges that present and future emergence of drones in the market poses.

C.3.1. Network Architecture

There is no reason why this design has to be at odds with a particular network architecture. Two possibilities are presented below:

- Architecture A: In this model, the drone communicates its position to the pilot, as usual. He or she is in charge of getting the parameters of his or her aircraft to the control center.
- Architecture B: unfeasible at the time, by the nonexistence of transponders suitable for drones, this solution will become the best option in the near future. The drone broadcasts its parameters to all nearby devices equipped to receive them.

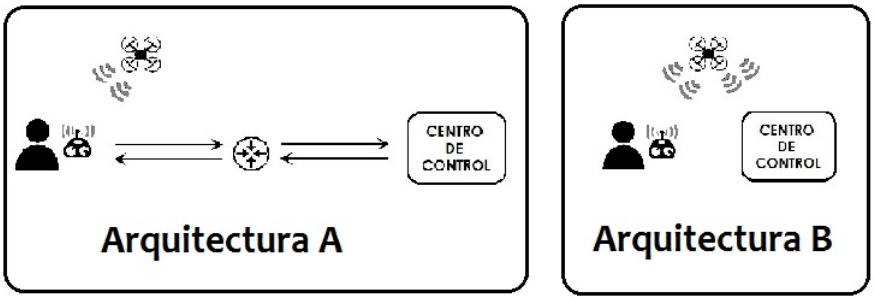

FIGURA C.1: Possible network architectures

C.3.2. Data format

Since there is no any standard on data reported by the drones, it is assumed one for this design:

```
id, posX, posY, vel, destX, destY, tipo, priority
```

In which are represented, in order, the identifier, the position (in its cartesian components), speed and destination (in its cartesian components) of the drone, if any, and the type and priority of the dron.

C.3.3. Structure of the map and map division

For the inner workings of the program, from now on, the entire two-dimensional space that the system monitors will be called **map**. Each map is divided into several **planes**, each of which is processed by a node in the system. In addition, each plane is divided into small textbfcells, 1 pixel x 1 pixel each, which will be used, among other things, to detect collisions, because the program will not accept two or more drones in the same cell during the same instant of time.

Moreover, since not all drones or the motivations for its use are the same, the map is divided into three areas, in each of which can only fly the drones of a certain type. Thus, only commercial drones will be allowed to run on the **commercial zones**, only the drones of recreational use may be moved within the **leisure zones** and **no** drone may travel within a **forbidden zone**. In the forbidden zone there should be some remote sensing systems such as LIDAR, described in Section 8.3 of this summary, to detect possible drones that are intentionally manipulated to withhold its parameters.

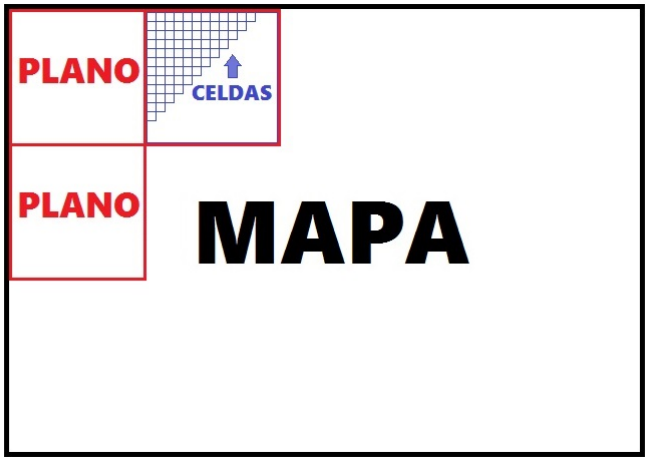

FIGURA C.2: Differences between map, plane and cell

C.3.4. Objetives and execution

The program should be able to detect if any drone has gone or will go out of their allowed zones in briefly if they maintain their trajectory. Of course, it must also notify the competent authorities if there is a drone circling in a prohibited area.

In addition, it should detect potential collisions between drones moving in the same plane, offering a solution to this collision whenever is possible. To do this, it will be calculated if the collision could be avoided with a reduction of the speed of the lower-priority drone, as long as this does not incur other problems with the drones of that plane.

Finally, it should provide a simple user-friendly graphical interface to display the map, the zones and the drones circulating in them.

The execution will take place in a distributed system using Spark, in which each plane is handled in parallel by a different node. These planes can vary in size, so that there can be smaller planes than usual in areas with the highest concentration of drones and the area can be monitored by a larger number of nodes, thereby improving the fluidity of the program.

C.4. Prototype

Due to the lack of resources, both human and financial, in addition to the limitation of time to carry out this project, the design presented can not be made in full. Instead, a prototype has been developed that, hopefully, will show on a small scale how this idea works.

Data sent by the drones, in the format described in paragraph 8.4.2, are the source of information to which **all** cluster nodes access. Thus, each node

searches the drones that are within the limits of the plane it is monitoring, and it performs the necessary operations.

The prototype has different classes programmed in Java:

C.4.1. Class `Plane`

The `Plane` class defines a plane. For this purpose, it indicates its limits in cartesian coordinates, the number of drones circulating within the coordinates of the plane and the number of commercial, leisure and prohibited zones that coexist within the plane. In addition, it stores these drones and these areas (as objects) in different lists.

Finally, it contains the necessary methods to add and remove items from the list of drones, as well as required to add items to the lists of areas.

C.4.2. Classes `leisureZone`, `commercialZone` and `Dangerzone`

These classes define, respectively, an leisure zone, a commercial zone and a forbidden zone. To do this, it simply indicates the boundaries of these areas in its cartesian components.

C.4.3. Class `Dron`

The class `Dron` serves to allow the program to efficiently manage the parameters sent by the drones. When received, they are encapsulated as objects `Dron` to operate with them.

The attributes of the class include the position of the drone and the cell that it is in, its speed, its destination and the cells in which it will be in the future 40 instants of time, if it maintains its speed and trajectory, all of it in the cartesian components. It also includes attributes to indicate the identifier, the type and the priority of every drone.

C.4.4. Classes `MyFrame`, `MyPanel` and `MyRectangle`

These three classes inherit, respectively, from Swing (graphics library) Java classes `JFrame`, `JPanel` and `Rectangle`. They modifie the `paintComponent` of these classes to design the map on which circulate the drones, the framework in which such a map is, and the rectangles representing each of the drones. The boundaries of each plane can be distinguished, and also the different types of drones differentiated by color, just as occurs with the zones.

C.4.5. Class `crearFicherosdeTiempo`

This class is responsible for simulating data traffic between drones and control system. To do so, it initializes random drones data and it creates text files with this data in the format specified in paragraph 8.4.2.

C.4.6. Class `Main`

The class `Main` is the most extensive and complex one, and is where the project is actually executed. It has different attributes and it is designed to make the necessary surveillance operations, such as detection of collisions between drones or ensuring that drones do not leave its allowed zones. To do this, it stores the various planes that make up the map in an RDD, so that these planes are distributed among the nodes in the cluster. As the RDD are immutable, every time the collection of planes has to change (each time new data arrives from the drones) a mapping is performed, with which a new RDD is obtained and it overwrites the previous one. For a more detailed explanation about the operation of this class, reading of paragraph 4.1.10 of this document is recommended.

The schematic operation of the class is shown in Figure 8.3.

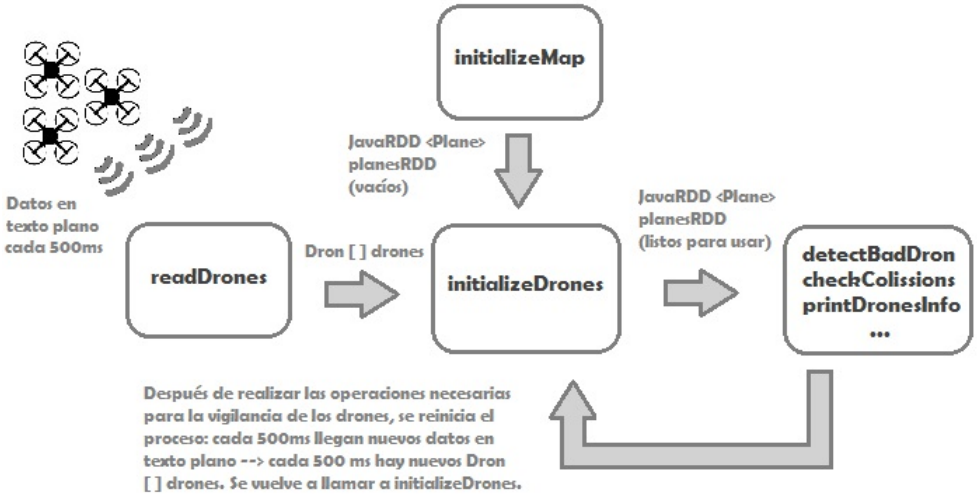

FIGURA C.3: Schematic operation of the class `Main`

C.5. Conclusions

After verifying the correct operation of prototype testing (for details, reading of chapter 6 is recommended) it can be said that the result is very satisfactory. The designed model is effective and feasible, and meets virtually all the goals outlined in the introduction. As improved a line to follow in

the future, it would be advisable to improve the algorithm that provides solution to the collision between two or more drones. However, it can be said that the program, despite its limitations, works correctly and in real time. This is great news for us to think that perhaps with more time and material and human resources, current technology allows us to firmly confront the challenges that air navigation brings in the nearest future.

Personally, I am very happy with the project and, in particular, for having had the opportunity to deepen programming using Spark, one of the most important tools of the moment in terms of analysis of large amounts of data. Of course, there are always things to learn and improve, but I think this experience has been really positive and a great introduction to a very interesting field of engineering.

Apéndice D

Código del prototipo

https://goo.gl/9Fxi2Z

Bibliografía

[1] K. Torres. (2015). Drones: Ventajas y desventajas en el mercado civil y militar, dirección: http://www.tecnolatinos.com/los-drones-ventajas-y-desventajas/ (visitado 15-08-2016).

[2] P. Finnegan. (2015). Uav production will total 93 billion dolars, dirección: http://goo.gl/WnCwVH (visitado 15-08-2016).

[3] Agencias. (2016). Un dron, a punto de chocar con un avión en un aeropuerto de parís, dirección: http://goo.gl/TIkORh (visitado 15-08-2016).

[4] Clarín. (2016). Un dron estuvo a punto de causar una tragedia aérea en munich, dirección: http://goo.gl/PKXMnI (visitado 15-08-2016).

[5] ABC. (2016). Un avión de pasajeros casi choca con un dron cuando se aproximaba al aeropuerto de cornualles, dirección: http://goo.gl/iYxisS (visitado 15-08-2016).

[6] I. Vázquez. (2015). Advierten de los peligros de pilotar drones en zonas urbanas, dirección: http://goo.gl/UQdRe3 (visitado 08-09-2016).

[7] PwC, *Clarity from above - pwc global report on the commercial applications of drone technology*, 2016.

[8] ICAO, *Continuing traffic growth and record airline profits highlight 2015 air transport results*, 2015.

[9] F. A. Administration, *Ads-b broadcast services*, 2014.

[10] Teknautas. (2016). Las aerolíneas avisan: Los drones se están convirtiendo en una amenaza de seguridad, dirección: https://goo.gl/XA6Uz5 (visitado 20-07-2016).

[11] R. G. y José Domínguez. (2015). Drones: ¿una amenaza para la privacidad?, dirección: https://goo.gl/pN5sgm (visitado 20-07-2016).

[12] PRNewswire. (2016). Uavionix introduces micro ads-b receiver for small drone collision avoidance, dirección: https://goo.gl/q3Um7T (visitado 08-07-2016).

[13] E. O. for Nuclear Research. (2016). Computing, dirección: http://home.cern/about/computing (visitado 15-07-2016).

[14] J. James. (2015). Data never sleeps 3.0, dirección: https://www.domo.com/blog/2015/08/data-never-sleeps-3-0/ (visitado 16-07-2016).

[15] P. Dave. (2013). Data – volume, velocity and variety, dirección: http://goo.gl/pt419s (visitado 16-07-2016).

[16] R. D. Schneider, *Hadoop for dummies*. John Wiley y Sons Canada, Ltd, 2012.

[17] A. Spark. (2016). Apache spark, dirección: http://spark.apache.org/ (visitado 28-07-2016).

[18] ——, (2016). Spark wins daytona gray sort 100tb benchmark, dirección: http://goo.gl/WulAh7 (visitado 18-07-2016).

[19] M. Z. et al, *Resilient distributed datasets*, 2012.

[20] J. Laskowski. (2016). Rdd - resilient distributed dataset, dirección: https://goo.gl/SxhF5e (visitado 18-07-2016).

[21] Wikipedia. (2016). Remote sensing, dirección: https://en.wikipedia.org/wiki/Remote_sensing (visitado 08-09-2016).

[22] A. Sánchez. (2016). Todo lo que necesitas saber para tener un drone en españa, dirección: https://hipertextual.com/2016/05/legislacion-drone-en-espana (visitado 10-09-2016).