Autonomía de vehículos "Follow me" en zonas aeroportuarias

Autonomous "Follow me" vehicles in airports

UNIVERSIDAD
COMPLUTENSE
MADRID

Grado en Ingeniería Informática
Facultad de Informática

Sergio Ramos Mesa
Director: Juan Carlos Fabero Jiménez

Madrid, 2021–2022

Índice general

Índice de figuras

Resumen

El objetivo del proyecto es mejorar el proceso de guiado de aeronaves en zonas aeroportuarias realizando el proceso de una manera autónoma y segura. A lo largo del presente trabajo se exponen las medidas de seguridad y adaptabilidad que se deben tomar en la implementación de un sistema que tome un dron que realice la tarea del guiado de aeronaves.

Se ha diseñado el software necesario que debe incluir el modelo, tanto de guiado del dron por las calles de rodaje, como de la acción de controlar las aeronaves que se encuentran a su alrededor.

El control de objetos alrededor del dron se ha realizado utilizando inteligencia artificial basada en visión por computador. Este sistema, además de calcular la posición de las aeronaves, será capaz de reconocer distancias a los mismos.

El sistema de guiado del dron está basado en waypoints. Para este sistema se ha creado una estructura de datos junto con un analizador léxico del lenguaje que pueda reconocer errores en los códigos que se cargarán para el movimiento.

En adición a todo el trabajo realizado se ha estudiado la viabilidad del proyecto ofreciendo soluciones a los contratiempos encontrados. También se ha incluido una lista de mejoras y cambios sugeridos para futuras versiones del proyecto.

Summary

This proyect aims to improve “Follow me vehicles” procedures assuring an autonomous and secure way to carry aircrafts. Along this proyect we will study the availability and adaptability measures that must be taken into account, as well as the necessary software that has to be implemented in the final model.

The guiadance system that allows the drone to move and the AI system that allows to control objects and distances have been studied too.

During the proyect, computer vision based AI solutions were studied. This solutions allow the drone to generate a behaviour to guide aircrafts and recognize their position and distance.

The drone movement system is based in a waypoint system which uses coordinates like Google Maps. To use it, a lexic language processor system has been created. Given an airport coordinate file, this system allows the drone to recognize errors in input data.

In addition to all the work done, project availability has been analyzed, giving solutions to encountered issues trying to guarantee the safety of the operations. We have added a changes list and improvements section at the end of this document.

Capítulo 1

Introducción

1.1. Estado de la cuestión

En la actualidad, solamente algunos aeropuertos en el mundo cuentan con un servicio de **ayuda al rodaje de las aeronave**. Este servicio se utiliza cuando las tripulaciones no están familiarizadas con la terminal ya sea por condiciones de visibilidad o trabajos de mantenimiento y en algunos casos por condiciones de seguridad durante el rodaje. Este servicio se conoce como vehículos «Follow me» o «Coches guía» y actualmente no está bien definido cuándo deben usarse estos vehículos y cuándo no.

Algunos pilotos reportan que en los aeropuertos más complicados, que suele ser cuando más hace falta este servicio, es donde generalmente no está implementado. Las aeronaves más modernas ya cuentan con sistemas de guiado en zonas aeroportuarias, aunque actualmente la gran mayoría de aeronaves no cuentan con este servicio.

Actualmente los vehículos «Follow me» utilizan combustibles fósiles para su movimiento y están tripulados por personal del aeropuerto cualificado para ello.

1.2. Motivación

Debido a la situación que presentan algunos grandes aeropuertos con respecto a los coches guía, se ha pensado en una **solución común que sea fácil de implementar** y que genere beneficios con respecto al sistema actual. El nuevo sistema debe integrar tanto a aeronaves modernas como antiguas.

Los vehículos utilizados actualmente en las operaciones de guiado de aeronave suponen un gasto de personal, combustible y reparaciones alto que no puede ser asumido por algunos aeropuertos. Se ha pensado en crear un sistema de guiado de vehículos basado en "waypoints" que sea capaz de automatizar el proceso.

Para automatizar el proceso de guiado de aeronaves se espera generar un comporta-

miento inteligente en los vehículos haciéndolos autónomos.

1.3. Objetivos

El proyecto trata de generar una ***autonomía en los vehículos guía*** dentro de zonas aeroportuarias, haciendo a su vez de ellos ***vehículos híbridos*** con bases de carga automáticas. Para ello, se generará un sistema de gestión de vehículos en un aeropuerto que espera generar mayor seguridad y eficiencia en zonas de tránsito aéreo.

La elección de vehículos buscará un enfoque autónomo. Estos vehículos pueden ***eliminar diferentes factores meteorológicos adversos***, como la niebla o la nieve haciendo el sistema de rodaje de aeronaves más eficiente y menos costoso. El uso de vehículos parcialmente eléctricos supondrá también beneficios económicos a medio-largo plazo.

En el proceso se estudiará la viabilidad sobre el uso de drones como vehículos guía y se contemplarán los problemas que puedan surgir de esta decisión de diseño. Este objetivo se basa en la premisa de una mejor visibilidad de los aparatos al poder volar a la altura de *cockpit* (cabina de mando).

Finalmente, junto a la propuesta sobre implementar un sistema de autonomía de vehículos, se espera generar un sistema de guiado basado en puntos de control que sea capaz de realizar un correcto movimiento de los vehículos por las zonas aeroportuarias. Este sistema debe ser muy adaptable a todos los tipos de aeropuertos y debe poder ser leído por los vehículos sin necesidad de operarios.

1.4. Estructura del trabajo

El proyecto contará de dos grandes bloques. El primero de ellos detallará el sistema y la infraestructura necesaria para implantar este sistema, estudiando el tipo de vehículo que se usará y las adaptaciones y sistemas que supondrá su uso. El segundo bloque genera un sistema de visión por computador capaz de reconocer y guiar aeronaves mediante el uso de sensores y cámaras instaladas en el vehículo.

Se estudiará la posibilidad de utilizar drones para poder maximizar la superficie libre para operaciones en tierra. Si esto no fuese posible, se pretende poder utilizar vehículos de tierra que se utilizarán además como ***medio de transporte*** para operarios cuando no se requiera su uso para aeronaves.

La mayor parte del trabajo realizado está basado en conocimientos adquiridos durante el curso del grado de Ingeniería Informática. También se han aplicado conocimientos adquiridos durante el estudio de procedimientos de control aéreo que realizo en mi tiempo libre, en la plataforma IVAO (1).

Introduction

1.5. State of affairs

Nowadays, only some airports in the world offer an aircraft taxi help service. This service is used when flight crews are not familiar with the terminal, either due to visibility conditions or maintenance work, and in some cases due to taxi safety conditions. This service is known as «Follow me» vehicles and, in fact, it is not well defined when they have to be used.

Some pilots report that this service is more unusual in bigger airports, where it is more neccesary. Newest aircrafts already have an airport guidance system implemented but, nowadays, there are a lot of *older* aircrafts which does not have this system implemented yet.

Presently, most of «Follow me» vehicles use petrol based fuel to work and they are manned by qualified staff.

1.6. Motivation

Due to the situation given by big airports respect «Follow me» vehicles, a smart and easy to implement solution has been though. This solution aims to generate benefits respect to the current system. The new system must integrate all aicraft types independently to its construction date.

Nowadays, using «Follow me» vehicles suppose personal, repair and fuel expenses that could be costly to some airports. A waypoint based system has been designed to automate this process.

In order to automate the aircraft guidance process, it is expected to generate intelligent behavior in the vehicles, making them autonomous.

1.7. Objectives

The objective of this project is to generate an autonomous behaviour in «Follow me» vehicles inside airport zones, making them hybrid using automatic charging areas. In order to do this, a vehicle management system will be designed. This system expects to generate better safety and efficiency conditions.

The vehicle choice will seek an autonomous approach. Flying vehicles can eliminate adverse weather factors (Eg: fog or snow) making the aircraft taxiing procedure efficient and cheaper. Using hybrid vehicles will also bring economic benefits in the medium to long term.

The project will study availability about the use of drones as «Follow me» vehicles. Problems about its implementation will be studied too. The objective of the project is to obtain a better visibility of them during taxi trying to fly drones at cockpit height.

Finally, next to the proposal to implement a vehicle autonomy system, it is expected to generate a waypoint based guiadance system that allows drones moving along airports. This system should be highly adaptable to all airport types and should be able to be read by drones without qualified staff.

1.8. Work structure

This project has two main blocks. The first one details neccesary infrastructure to implement the designed system, studying the type of vehicle used, its adjustments, and the airport modifications to allow it. The sencond block generates a computer vision system which allows drones to recognize aircrafts using installed sensors and cameras.

The posibility of using drones to maximize ground operations area will be studied. In case it is impossible, ground vehicles will be used, letting qualified staff use them for transport when they are not been in use.

Most of work done is based on adquiered knowledge during the Computer Science degree and during studing ground aereal control procedures. This procedures were adquiered by IVAO website (1).

Capítulo 2

Elección del vehículo

En una primera aproximación se ha pensado en la posibilidad de automatizar los vehículos terrestres, es decir, crear una adaptación para los vehículos existentes para que puedan realizar la tarea de guiado de aeronaves de manera autónoma. Estas adaptaciones suponen varios **problemas** a solventar dependiendo de si la automatización será total, creando vehículos totalmente autónomos, o si la automatización será parcial, necesitando así la contratación y organización de operarios que muevan los vehículos.

En el caso de tener una automatización completa usando vehículos terrestres, el sistema requeriría carriles especiales de circulación así como hardware específico de modificación de los vehículos actuales para su automatización.

A continuación estudiaremos una alternativa a los vehículos terrestres que suponen un menor coste de mantenimiento y combustible.

Como motivación a este capítulo, vamos a estudiar la posibilidad de utilizar **drones no tripulados** para realizar la tarea de guiado. Estos drones estarán completamente automatizados no requiriendo operarios para el movimiento de los mismos.

Resulta sencillo reconocer que los posibles problemas que puede tener el uso de drones como vehículos *"follow me"* son:

- Peso de los sistemas que debe portar.
- Duración de las baterías.
- Fuerzas referentes a las aeronaves.

Estos problemas los trataremos a lo largo de este capítulo. Se propondrán soluciones y modificaciones sobre el dron de referencia así como se expondrán las alternativas se han considerado.

Como ejemplo de dron que se podría usar para este trabajo hemos analizado las características del HYBRiX [1] que explicaremos a continuación:

[1] https://www.quaternium.com/hybrix20-rtf/

HYBRiX 2.1

Se trata de un dron medio, de 20kg de peso e híbrido (requiere combustibles fósiles) que ofrece una gran autonomía. Este dron es usado por varias delegaciones de policía local y salvamento en zonas costeras de la península española.

La empresa que fabrica estos drones es valenciana y tenemos la oportunidad de contactar con ellos si fuese necesario. En la figura 2.1 se muestra la ficha técnica del aparato. Esta ficha nos muestra las capacidades a las que podemos someterlo.

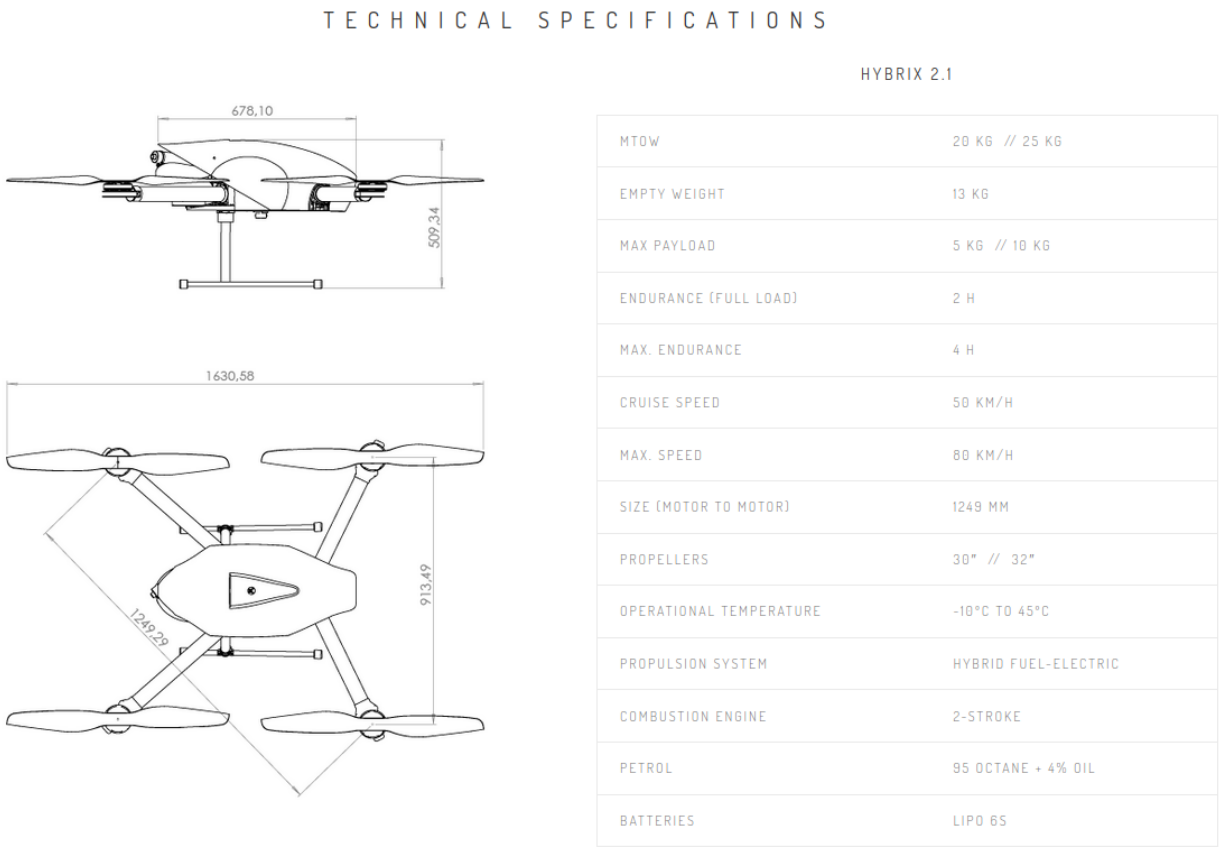


Figura 2.1: Especificaciones técnicas del HYBRiX 2.1

Las análisis sobre el uso de drones ha dado como resultado diferentes categorías que engloban las dificultades a las que se enfrenta su uso. Estas dificultades están sujetas a las siguientes categorías:

2.1. Componentes del sistema

En esta sección se estudiarán las modificaciones que se realizaran en el dron elegido y el peso que éstas puedan suponer en la operatividad del mismo. Las modificaciones están anidadas en cuatro categorías principales:

- Luces.

- Baterías.
- Placa controladora.
- Adaptaciones de la estructura.

Se exponen a continuación cada uno de los aparatados listados proponiendo las alternativas consideradas. Se estudiarán también dos de los factores de riesgo anteriormente mencionados, el peso de los componentes elegidos y las adaptaciones para el sistema de carga.

En lo que respecta al sistema de carga, HYBRiX puede cargar entre 5Kg de peso sin sufrir consecuencias en su velocidad máxima y 10Kg como carga máxima.

2.1.1. Luces

En una primera fase del análisis se estudió la posibilidad de utilizar luces separadas para el guiado de la aeronave y la seguridad del dron. Al estudiar los modelos de luces *beacon* se propone utilizar una misma luz para ambas tareas.

Las luces *beacon* permiten al dron hacerse visible ante otras aeronaves. La FAA[2] requiere que este tipo de luces estén bajo la norma FAR 107.29, que establece que deben ser visibles desde 3 millas terrestres de distancia y ser estroboscópicas a una velocidad de 40 a 100 ciclos por minuto.

Siguiendo estas indicaciones, se han tenido en cuenta varios modelos de luces *beacon* específicas para drones, como los modelos “Rugo R1S” y “D3060”. Estas opciones se han descartado ya que el foco de luz que emiten es demasiado tenue (75ft / 23m) para las operaciones a las que se espera que trabajen.

Otra opción más potente, pensada para uso en drones de exterior, es el modelo EXOLANDER Spotlight System[3]. Esta luz emite un foco de 15º de color blanco de 2300 lúmenes. Con tan solo 346 gramos es capaz de emitir luz a una distancia de 200ft (61m).

Finalmente se ha estudiado sobre el uso de luces V-16 utilizadas en vehículos de emergencia. Estas balizas emiten una luz en color amarillo auto visible a más de un kilometro de distancia (3000ft), que es emitida en un ángulo de 360 grados. Resultan tener muy bajo peso (130g) y su autonomía sería indefinida ya que el sistema de baterías del dron alimentaría este sistema.

2.1.2. Controlador

La placa que se usará para el controlador del sistema será una Raspberry PI[4] . Este modelo es necesario para el reconocimiento de imágenes que estudiaremos en futuros

[2] https://www.faa.gov/
[3] https://www.foxfury.com/product/exolander-spotlight-system-for-evo-ii/
[4] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

capítulos. Esta placa ronda los 50g de peso y tendrá que ser conectada con el resto de sistemas mediante cableado de puente, cuyo peso es irrelevante.

Se ha estudiado utilizar la alternativa ARDUINO UNO[5] como placa controladora del sistema pero finalmente se ha descartado.

TensorFlow Lite para microcontroladores se codificó en C++ 11 y requiere una plataforma de 32 bits. Se probó de manera exhaustiva con muchos procesadores basados en la arquitectura de la serie ARM Cortex-M y se trasladó a otras arquitecturas, como ESP32.

2.1.3. Cambios estructurales

Los cambios estructurales que se esperan realizar en el modelo son los siguientes:

- Tren de aterrizaje mejorado.
- Adaptación de baterías extraíbles y sistema de repostaje.
- Adaptación para las luces de navegación y la placa controladora.

El material sobre el que se trabajarán los cambios estructurales será fibra de carbono. Este material es el más duro y ligero que podemos encontrar para este tipo de tareas. Una placa de fibra de carbono más gruesa supone mayor rigidez y resistencia, pero también mayor peso. Las placas que debemos utilizar en nuestro chasis deben ser al menos de 4 mm de grosor. Ningún elemento adicional al dron base estará situado en los brazos del mismo. Todos los componentes se encontrarán en la base (centro).

Por motivos técnicos las baterías deberán colocarse en la parte inferior, pudiendo acceder a ellas con gran facilidad. Este dato no supone un cambio estructural en el aparto. El cambio de baterías y repostaje lo veremos en detalle en la siguiente sección.

El chasis debe incluir también una adaptación para una cámara y dos servomotores que sean capaces de moverla en dos ejes según el software diseñado. Esta adaptación añadirá dos servomotores de 240g de peso cada uno y una cámara. Hemos elegido como modelo la cámara “GoPro Hero7” debido a su buena calidad de imagen (1080p / 4k), sus materiales de construcción pensados para resistir todo tipo de clima y golpes, y su bajo peso, que tan sólo es de 116g.

Se deben incluir también dos cámaras que actuarán a modo de sensor de proximidad para objetos y aeronaves cercanas. Utilizaremos el mismo modelo de cámara que en el apartado anterior. Esta vez no se requiere de servomotores ni una adaptación especial más allá de su sujeción al chasis del dron.

Para medir las distancias con las aeronaves y otros objetos se utilizará un medidor láser; el modelo elegido es un controlador específico para Raspberry PI llamado B87A-b200416. Este láser cuenta con una precisión de +-3mm alcanzando distancias de hasta 100m. Este dispositivo debe ser cubierto ya que su temperatura de trabajo oscila entre los 0 - 40°c

[5] https://www.arduino.cc/

2.1.4. Baterías

La duración de las baterías dependen de cada dron en específico. Vamos a trabajar sobre la referencia del HYBRiX.

En este modelo y según el fabricante, la batería puede variar entre las 2h y las 4h, dependiendo del esfuerzo del dron en cada momento. Esta duración de batería es suficiente para hacer varios trayectos de taxi en cualquier aeropuerto.

Como el sistema que estamos estudiando es híbrido, una vez que se agote el tiempo de autonomía no basta con cargar las baterías del dron, sino que también hay que reabastecer su combustible. El repostaje de los drones se puede hacer de dos maneras:

1. **Operarios:** Se encargan de reabastecer los drones una vez se paran en sus bases. Esto puede ser peligroso si hay mucho tráfico en una base de carga.
2. **Reabastecimiento automático:** Trata de implementar un sistema que sea capaz de reabastecer de manera automática los drones dentro de un *box*, de esta manera se eliminará el factor humano y los riesgos que puede suponer dejar los aparatos en la superficie.

Se debe tener en cuenta que la batería no solo es consumida por los motores del dron, sino que también consumirán batería la placa controladora del sistema autónomo, los sensores y las luces adicionales.

Una opción para aumentar autonomía y la seguridad del dron es contar con dos baterías que trabajen en paralelo, una para trabajar con los motores y otra para los sistemas auxiliares. La placa podría controlar la carga de cada una de las baterías por separado y, en caso necesario, cargar la batería auxiliar con el sistema de combustibles fósiles, al igual que la batería principal. El peso de la batería auxiliar sería de 20Ah con un peso 400g. El modelo elegido es la Xiaomi Mi 50W Power Bank. Este modelo es superior al resto del mercado de baterías portátiles gracias a su gran capacidad de carga y la potencia de hasta 74W.

Queda pues definido que la duración de las baterías no es un problema en drones híbridos ya que en capítulos posteriores se estudiará un sistema de recarga de estos aparatos de manera automática, que se integrará en las zonas aeroportuarias y actuarán como puntos de espera.

2.2. Vientos y fuerzas

Como aclaración, aunque el guiado de aeronaves se haga desde la parte delantera (delante del morro), se deben tener en cuenta todas las fuerzas que estas aeronaves generan a la hora de mover los drones por los aeropuertos.

Existen dos factores que pueden suponer una fuerza sobre nuestros drones al volar cerca de aeronaves provocando posibles riesgos.

1. Factores meteorológicos

Por seguridad, el dron debe volar a una distancia considerable de la aeronave. Esta distancia elimina el problema de las rachas de **viento** (siempre que sean soportables) ya que el dron, junto a un giroscopio, es capaz de contrarrestar estas rachas sin desviarse de su camino. Como hemos estudiado en las especificaciones del fabricante del dron elegido, esto puede o no resultar un problema dependiendo de cada momento concreto por lo que la autoridad aeroportuaria deberá controlar este factor más allá de la seguridad que el sistema proporciona.

2. Factores mecánicos

Existen dos tipos principales de factores mecánicos que debemos tener en cuenta cerca de aeronaves. El primero de ellos de succión de gases y el segundo de expulsión de gases, llamado *jetblast*.

El cono de succión de los motores de las aeronaves no tiene alcance suficiente para succionar a un dron posicionado en la cabecera de la aeronave. Como hemos mencionado en los factores meteorológicos, nuestro dron, por seguridad, se posicionará varios metros alejado de la aeronave proporcionando mayor seguridad y asegurando el correcto funcionamiento del mismo.

La figura 2.2 ilustra el cono de succión del motor CFM56[6], utilizado en modelos Boeing 737. El cono de succión de este motor abarca un arco de 4,2 metros de radio.

Figura 2.2: Señalización de la zona peligro cerca de motores Boeing 737

Los gases de escape de los motores pueden suponer un problema si las aeronaves son grandes (**Medium** o **Heavy**).

El JetBlast (2) se define como el conjunto de peligros asociados a la fuerza de explo-

[6] https://www.cfmaeroengines.com/engines/cfm56/

sión generada por motor a reacción. Este peligro existe siempre que el motor esté en funcionamiento, pero aumenta y se extiende sobre un área mayor en configuraciones de alta potencia del motor al rodar, antes y durante el despegue, y durante la actividad de mantenimiento del motor. Esto puede suponer dos problemas a nuestro dron.

- Fuertes ráfagas de viento
- Gases a altas temperaturas

Este volumen de aire caliente que generan las aeronaves puede ser peligroso para otras aeronaves que circulen en las cercanías a la que lo genera. En la figura 2.3 se ilustra cómo la masa de aire caliente alcanza los 100kt (185,2 Km/h) en una extensión de hasta 500ft (150m).

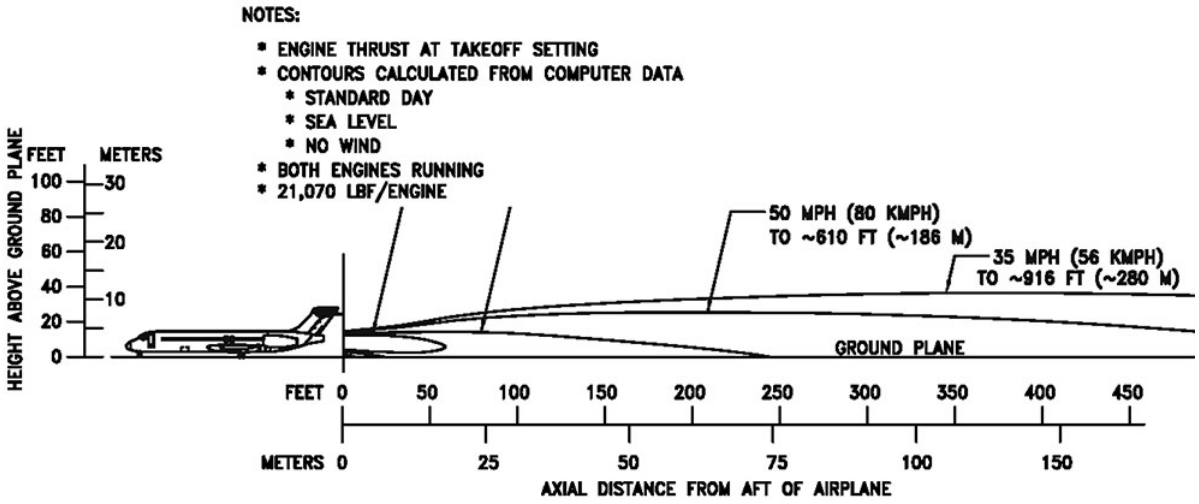


Figura 2.3: Jetblast producido por el Boeing 717

Se da por sentado que para nuestro dron la fuerza del **jetblast** supone grandes daños y problemas en la navegación. Por ello, se debe tener especial cuidado en aquellas zonas con una gran cantidad de tráfico. Se debe habilitar carriles especiales para la circulación de los drones entre partes de los aeropuertos. En cuanto a la fuerza de succión y compensación por rachas de viento, no son un factor a tener en cuenta siempre que estas rachas no superen la indicación del fabricante del dron.

Resumen

A modo resumen de este capítulo, consideramos que la utilización de drones es apta para la tarea propuesta.

Se ha propuesto un modelo de dron como referencia y se han realizado modificaciones sobre el mismo para adaptarlo a la tarea deseada.

En cuanto a los factores de riesgo

El análisis ha demostrado que las rachas de viento no suponen un problema en la navegación siempre que estas rachas sean asumibles por la velocidad que publica el fabricante del dron. Se debe tener especial cuidado con el **jetblast** siempre que los drones trabajen por las calles de rodaje.

El peso aproximado que debe soportar el dron en las modificaciones necesarias es de 1.5Kg. Este peso no supone una adversidad para la correcta operación del dron, que puede sorportar hasta 5Kg de peso.

La duración de las baterías puede resultar un problema por lo que habrá que tratar la carga de los drones y el transporte y movilización de estos entre bases de carga de manera segura. En el próximo capítulo estudiaremos algunos aspectos concretos sobre la carga de baterías.

En cuanto a las especificaciones técnicas

En la figura 2.4 se muestra la lista de modificaciones junto a sus datos técnicos. Aunque estas modificaciones están sujetas a cambios, nos permiten tener una idea aproximada del peso y precio de los componentes. Al conocer el peso podemos estimar el impacto que tendrán estos componentes en la velocidad final del dron.

Tipo	Nombre	Peso	Precio	Otro
Dron	HYBRiX2	20Kg	Desconocido[7]	Max. velocidad 80Km/h
Luz	V-16	160g	$\sim$ 15€	Color blanco o amarillo
Cámara	GoPro Hero7	116gx3	$\sim$ 240€ x 3	Vídeo: 4K a 60 fps.
Láser	B87A-b200416	13g	Desconocido	19200bps, 3Hz
Válvula	CAV-110 (1/8 NPT)	23g	$\sim$ 23€	60 PSI max
Controlador	Raspberry Pi 3	50g	$\sim$ 33€	40 pines + cámara USB
Cableado	Cable de puente	$\sim$ 50g	$\sim$ 10€	Diferentes longitudes
Batería auxiliar	Xiaomi Mi Power Bank	400g	40€	20Ah a 74W
Total Mods.	-	$\sim$ 1.5Kg	$>$ 700€	-

Figura 2.4: Especificaciones técnicas de los componentes añadidos al dron

Capítulo 3

Sistema de carga

La premisa para este capítulo es repartir bases de carga en las zonas aeroportuarias donde se puedan cargar los drones de manera segura. Este sistema complementa las modificaciones que se han realizado sobre el dron y permiten crear puntos de espera repartidos por el aeropuerto.

3.1. Problemas a tener en cuenta

Tras realizar un estudio al respecto, los problemas referentes a la batería que nos pueden surgir en nuestro sistema son:

- **Las dimensiones y el peso de las baterías**, que pueden suponer problemas de peso cuando se requiera mucha potencia (velocidad) en los mismos, o se trabaje en condiciones meteorológicas adversas.
- **Garantizar la seguridad e integridad de la batería y del depósito de combustible**, que, en caso de verse comprometidos por un fallo, podría llegar a explosionar o incendiarse provocando una situación de alto peligro.
- **Adaptar las baterías con métodos de extracción o inserción**, algo complicado cuando en la actualidad dichas baterías suponen no solo una fuente de energía para el vehículo sino también un elemento estructural muy importante del mismo.
- **La recarga del combustible en el modelo híbrido**, que se tiene que realizar en un entorno seguro debido al uso de líquidos combustibles.

3.2. La estación de carga (ERDG)

La Estación de Repostaje de Drones Guía (ERDG) consistirá en una **cabina cerrada en la que los drones podrán aterrizar** y en la que repostarán combustible y cambiarán

sus baterías. Estas bases de carga actuarán como puntos de espera mientras no se precisen los servicios de los drones.

La estación constará con una estructura metálica rígida en la que se adaptará el sistema NORIA (que veremos a continuación) junto con un sistema de abastecimiento de combustible, una plataforma de aterrizaje móvil y una tapa o parte superior móvil que podrá cerrarse o abrirse para proteger el interior de factores meteorológicos y haciendo el exterior de la caja más seguro durante el proceso de repostaje.

El proceso de repostaje se realizará en **dos fases**, una primera fase en la que se hará el cambio de baterías y una segunda fase en la que se realizará el repostaje de combustibles fósiles, minimizando así el peligro por incendio dentro de la caja.

3.2.1. NORIA

Sergio Sarasola Merino, en su trabajo de fin de grado “Sistema de reabastecimiento para drones” (3), propone un sistema sistema en forma octogonal que permitirá un almacenamiento circular de las baterías, llamado NORIA e ilustrado en la figura 3.1. Este sistema prevalece frente a otros por el sencillo acceso a las baterías que proporciona, así como un alto grado de compatibilidad.

También es destacable que este diseño, a diferencia de otros, permite que el actuador mecánico que se encargará de manipular las baterías solo tendrá que trabajar sobre un único plano, facilitando en gran medida su implementación.

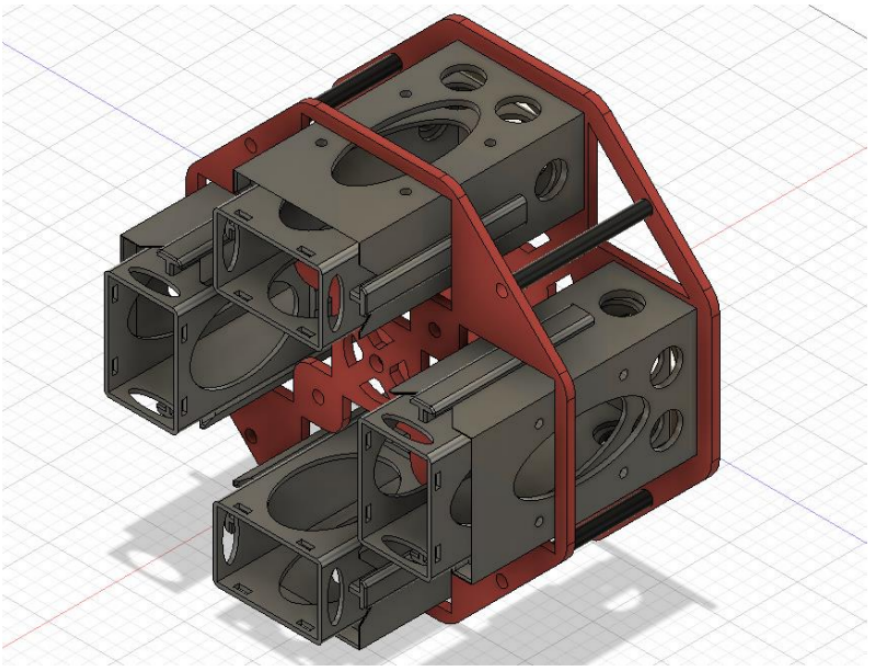


Figura 3.1: Sistema NORIA

Dicho conjunto mecánico albergará hasta 3 baterías de forma simultánea dejando siem-

pre uno de sus 4 espacios libre para la batería de un posible dron entrante. Este sistema ha sido diseñado para soportar una gran adaptabilidad pudiéndose incrementar el número de baterías incrementando simplemente el radio de la NORIA con respecto a su eje central.

3.2.2. Combustibles fósiles

En cuanto al repostaje de combustible fósiles, se propone crear un sistema similar al utilizado en las operaciones “*Aereal Air Refueling*” (AAR).

En las operaciones de AAR se transfiere el combustible entre las nodrizas y receptores mediante una manguera flexible y una cesta que contiene la válvula o una lanza rígida llamada “*boom*”. En este proyecto se pretende reutilizar este tipo de válvulas de no retorno de tal manera que cuando el dron se pose en el suelo, la lanza quede enganchada y cuando retome el vuelo, al ser controlado por un sistema centralizado (en nuestro caso utilizando una Raspberry PI), la válvula se cierre impidiendo el paso de combustible.

El repostaje de combustibles fósiles no se realizará al mismo tiempo que la recarga de las baterías eléctricas. Esta decisión de diseño aumentará el tiempo de repostaje pero aumentará la fiabilidad y seguridad del prototipo. Usando este método, primero se realizará el repostaje de combustible y posteriormente la recarga de baterías, pudiendo esta segunda fase dejarse sin finalizar. Al finalizar la operación se realizará una fase de limpieza de la cabina que la preparará para el siguiente dron.

3.2.3. Posicionamiento

En el proyecto de Sergio Sarasola (3) han observado que el error que más condiciona el buen funcionamiento de la estación es el comportamiento errático de los motores tipo servo que componen el sistema de sujeción de la batería durante su transporte. Esto hace que en algunas ocasiones la batería no esté asegurada en su posición nominal y provoca situaciones anómalas durante el proceso de inserción o retirada de la misma.

En nuestro proyecto este problema puede reducirse considerablemente ya que se pretende que el repostaje se haga dentro de una cabina cerrada por debajo del nivel del suelo, protegiendo todo el proceso de factores externos. Además, podemos contar con sensores que comuniquen si ha habido algún error en el proceso. Queda ilustrado en la figura 3.2 el funcionamiento interno del sistema.

Los sensores eliminarían el factor de riesgo que puede suponer una batería mal colocada. Funcionan de tal manera que un temporizador marca el tiempo de repostaje. Para comprobar posibles fugas de combustible se añadirá al *box* una sonda de detección de fugas por ultrasonidos, que en caso de estanqueidad de combustible, comunicaría el incidente al equipo técnico y dejaría inactiva la plataforma.

En la vista exterior, ilustrada en la figura 3.3, se puede apreciar cómo una vez el sensor de proximidad alerte de un dron en las cercanías del *box* y este se encuentre en estado libre, el cierre de seguridad se desbloquea abriendo la caja y subiendo la plataforma de

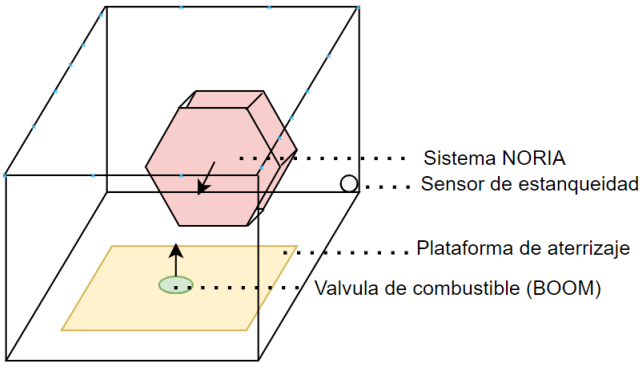


Figura 3.2: Interior del sistema BOX

aterrizaje. Una vez el dron quede aterrizado (encima de la válvula BOOM) la plataforma bajará y la caja quedará cerrada de manera estanca.

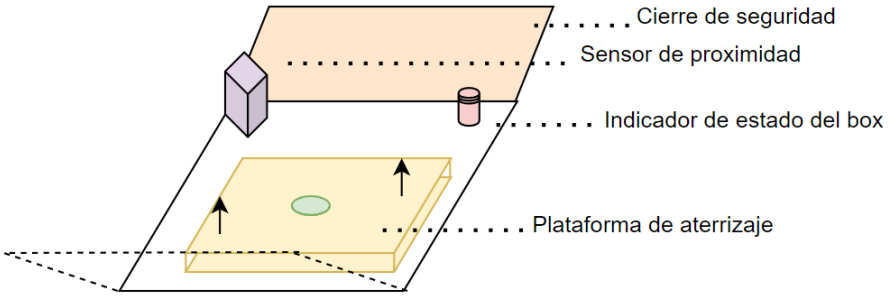


Figura 3.3: Exterior del sistema BOX

Capítulo 4

Sistema de guiado

El sistema de guiado de aviones se divide en dos grandes bloques.

El primer bloque se encarga del sistema de visión por computador que dará información acerca de la aeronave que se está guiando, así como de otros tráficos y factores externos que pudiesen suponer un riesgo. De ahora en adelante nos referiremos a este sistema como GUIa.

El segundo bloque consiste en un sistema mediante el cual los drones pueden guiarse por un aeropuerto mediante un método de entrada con datos del aeropuerto. A este sistema lo llamaremos MAPa, y se encargará de guiar el dron por el camino más corto y seguro posible desde un punto A a un punto B. Este sistema también controlará la interacción entre varios drones evitando la colisión y la molestia de estos en sus operaciones.

4.1. Tecnologías utilizadas

El entorno de programación elegido ha sido **Python** tanto para el sistema de IA como para el sistema de movimiento del prototipo. Para realizar el sistema de visión artificial nos ayudaremos de librerías específicas que nos permitan detectar en tiempo real varios objetos e identificarlos. Estas librerías son **TensorFlow** (4), **ImageAI** (5) y **OpenCV** (6).

Otras librerías auxiliares que utilizaremos durante este proyecto son las siguientes:

- **Imutils:** Librería de procesamiento de imágenes.
- **Numpy:** Librería utilizada para guardar datos en matrices.
- **Argparse:** Ayuda en la entrada por línea de comandos.
- **Otras:** CV2, Loggin, Time, OS..

Es muy importante saber en qué versiones se ha ejecutado el sistema de visión artificial ya que las dependencias de las funciones pueden variar entre versiones. Las versiones que se han utilizado durante el proyecto son las siguientes:

- Tensorflow (4) 2.8.0
- Ovencv (6) 4.5.5
- Tflite-model-maker (7) 0.3.4
- Tflite-support 0.3.1
- Imageai (5) 2.1.6
- Numpy (8) 1.21.5

4.2. Sistema GUIa

Para poder llevar a cabo la tarea de guiar aeronaves es necesario que éstas no nos pierdan de vista. Para ello, nuestro vehículo guía debe controlar la distancia a la que se encuentran las aeronaves próximas. Para realizar esta tarea vamos a implementar un sistema de visión por computador, compatible con la arquitectura **ARMx32** de Raspberry Pi, que reconozca las aeronaves y pueda medir la distancia a éstas.

Este sistema está diseñado para usar 3 cámaras, dos de ellas estáticas que ayudarán en el sistema de reconocimiento de objetos cercanos, y una móvil que será la encargada del guiado. El sistema de reconocimiento de distancias, usado solamente en la cámara encarga del guiado, se implementará mediante un láser que apuntará a la parte más relevante del avión guiado por una cámara. Explicaremos en detalle este mecanismo al final del capítulo.

4.2.1. Trabajo previo

En las primeras versiones del trabajo se creó un sistema de inteligencia artificial basado en ImageAI. Este sistema utilizaba el modelo **resnet50 coco best v2**[1].

Posteriormente, se creó otro sistema basado en Tensorflow. Este sistema utilizaba el modelo **ssd mobilenet v2**[2].

En este punto, nos dimos cuenta de la necesidad de creación de un modelo personalizado. Trabajando sobre la versión de Tensorflow, nos surgía el problema de la complejidad en el entrenamiento de un modelo y la falta de integración con modelos ARM.

En adición, la creación del modelo suponía gran cantidad de trabajo en pruebas además de no poder ser exportado a una versión *lite* por lo que al final no se utilizó este modelo.

[1]Modelo en GitHub: https://github.com/fizyr/keras-retinanet/releases
[2]SSD-Movilenet GitHub: https://github.com/saunack/MobileNetv2-SSD

Antes de volver a la primera opción se encontró una posibilidad que utilizaba Tensorflow en su versión Lite que, finalmente, ha sido la elegida para el trabajo.

4.2.2. Modelo final

Como hemos mencionado anteriormente, utilizaremos la **API de detección de objetos** de TensorFlow, concretamente la versión **Lite**, para la detección.

Tensorflow Lite (4) es un conjunto de herramientas que ayuda a los desarrolladores a ejecutar sus modelos en dispositivos incorporados, móviles o de IoT, y les permite implementar el aprendizaje automático integrado en el dispositivo. Nosotros lo usaremos por su simplicidad en la creación de modelos y ayudas por parte de la API. La única entrada que se le debe especificar al programa será el *dataset* creado. Lo veremos a continuación.

Las características clave de TL son:

- Optimizado para el aprendizaje automático integrado en el dispositivo. Aborda 5 limitaciones clave: latencia, privacidad, conectividad, tamaño y consumo de energía.
- Compatibilidad con múltiples plataformas, lo que incluye dispositivos iOS y Android, Linux incorporado y microcontroladores.
- Compatibilidad con diversos lenguajes, entre los que se incluyen Java, Swift, Objective-C, C++ y Python.
- Alto rendimiento, con aceleración de hardware y optimización de modelos.

El proceso que se ha seguido ha sido crear un conjunto de imágenes sobre las diferentes partes de las aeronaves para entrenar un modelo Tflite y usarlo en la detección de objetos. Veamos a continuación estos pasos detalladamente.

4.2.3. Creación de un Modelo

Como parte del reconocimiento de objetos, debemos tener en cuenta qué objetos ajenos al guiado de aeronaves están presentes cerca de la zona de operaciones. En caso de encontrarnos con algún vehículo o persona física cerca de la zona la operación de guiado deberá detenerse y dar alerta al control aéreo. Por simpleza y eficiencia del modelo generado no se han tenido en cuenta estos factores, reconociendo exclusivamente partes de las aeronaves, por lo que centraremos el capítulo en el reconocimiento exclusivo de aeronaves y sus partes.

El modelo deberá extenderse en futuras versiones para que reconozca las siguientes categorías de objetos:

- Vehículos (Coches, Motos, Autobuses, Helicópteros..)

- Personas
- Animales (terrestres grandes y voladores)
- Árboles y Farolas

El conjunto de datos generado ha sido creado a partir de imágenes de una biblioteca de imágenes de aviones online 4.1.

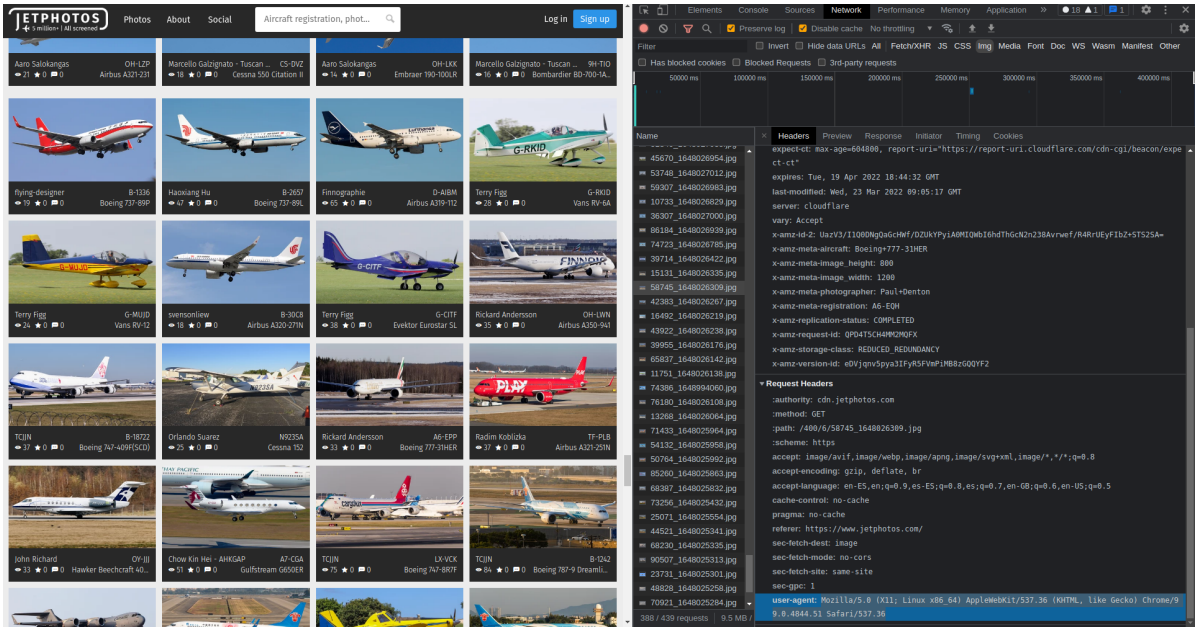


FIGURA 4.1: Referencia: https://www.jetphotos.com/

Para obtener las imágenes desde la web se ha creado un *script* en Bash[3] que realiza peticiones a dicha web obteniendo los enlaces a las imágenes para posteriormente descargarlas. Veamos el código en detalle:

```
1  #!/bin/bash
2
3  #Preparación de la carpeta para las imágenes
4  mkdir -p img
5  rm -f img/*
6
7  #Petición que obtiene las URLs sin formato
8  curl -s 'https://www.jetphotos.com/api/json/photos_internal.php?limit=2000
9     &offset=0'\
10   -H 'authority: www.jetphotos.com' \
11   -H 'pragma: no-cache' \
12   -H 'cache-control: no-cache' \
13   -H 'accept: */*' \
14   -H 'user-agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
15   (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36'  \
```

[3]Web de Bash https://linux.die.net/man/1/bash

```
  -H 'x-requested-with: XMLHttpRequest'  \
  -H 'sec-gpc: 1' \
  -H 'sec-fetch-site: same-origin'  \
  -H 'sec-fetch-mode: cors'  \
  -H 'sec-fetch-dest: empty'  \
  -H 'referer: https://www.jetphotos.com/'  \
  -H 'accept-language: en-ES,en;q=0.9,es-ES;q=0.8,es;q=0.7,en-GB;q=0.6,en-US;
     q=0.5' \
  -H 'cookie: -PRIVATE-' \
  --compressed | jq -r '.[].mediumURL' > urls.txt

#Prepara el formato para descargar las imágenes
cat urls.txt | sed -e 's#^//#https://#' -e 's#/400/#/full/#' > urls_fixed.txt

#Obtención de las imágenes a través del fichero de URLs
wget -P 'img/' -U 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36' -i urls_fixed.txt

#Bucle para renombrar las imágenes
counter=0
for file in $(ls img/*); do
  mv "$file" "img/$(printf '%04d' $counter).jpg"
  counter=$((counter+1))
done
```

El *script* realiza una primera petición a la web para obtener una lista de URL del conjunto de imágenes. Para poder realizar esto se debe especificar el *user-agent* en la petición ya que de otra manera la web bloquearía la conexión.

Una vez obtenemos la lista de URL de las imágenes realizamos un análisis (*parse*) para que tengan el formato que nos interesa. Posteriormente se realizan peticiones a la web obteniendo las imágenes. Finalmente se realiza un renombrado de todos los archivos.

En las figuras 4.2 y 4.3 se muestran imágenes ilustrativas tanto de la descarga de los archivos como del renombrado.

Se han obtenido un total de 2000 imágenes de las cuales se ha realizado una selección de **226 imágenes**. La distribución será del 60 % para entrenamiento (136 imágenes) y 40 % para pruebas (90 imágenes).

Para tomar estos valores se han tenido en cuenta dos factores: cuanto menor es el número de datos de entrenamiento, mayor será la varianza del sistema. Y cuanto menor sea el número de datos de prueba, mejor rendimiento tendrá la estadística. Lo que se pretende conseguir es que ninguna de las varianzas sea demasiado alta.

Es importante que al mejorar el modelo y añadir nuevas imágenes, éste **no genere *overfitting***.

FIGURA 4.2: Proceso de descarga de imágenes

Se dice que un modelo está sobre-ajustado (*overfitted*) cuando lo entrenamos con demasiados datos. Cuando un modelo se entrena con tantos datos, comienza a aprender del ruido y las entradas de datos inexactas en nuestro conjunto de datos.

Una vez seleccionadas las imágenes, para cada una, se debe seleccionar qué objetos se encuentran en él. Para ello utilizaremos la lista etiquetas (*labels*):

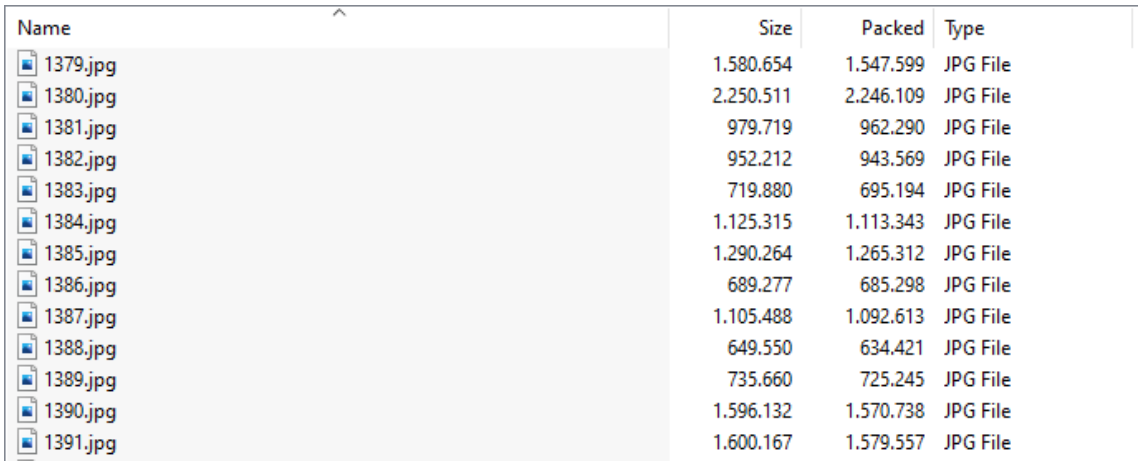
['aircraft','engine','cockpit','tail']

Hemos utilizado la herramienta **labelImg** (9) para realizar la categorización de las imágenes. LabelImg es una herramienta de código abierto para marcar objetos en imágenes. Para cada imagen devolverá un archivo `XML` que usaremos en la creación del modelo. En la figura 4.4 se ilustra el proceso que se debe seguir para conseguir el etiquetado.

Ahora que ya tenemos todo lo necesario podemos generar el modelo deseado. Para ello, utilizaremos un *script* de Google Colab **proporcionado por Tensorflow** (versión 2) para la creación de modelos TfLite. Este *script* (10) puede encontrarse en su página web y resulta una guía completa de cómo entrenar modelos TfLite utilizando un *dataset* personalizado.

Primero, y tras importar las librerías necesarias mencionadas anteriormente, cargamos el *dataset* que hemos creado.

```
#Imágenes seleccionadas para el entrenamiento
train_data = object_detector.DataLoader.from_pascal_voc(
    '/WorkingDirectory/images/train/jpg',
    '/WorkingDirectory/images/train/xml',
    ['aircraft','engine','cockpit','tail'])
```



Name	Size	Packed	Type
1379.jpg	1.580.654	1.547.599	JPG File
1380.jpg	2.250.511	2.246.109	JPG File
1381.jpg	979.719	962.290	JPG File
1382.jpg	952.212	943.569	JPG File
1383.jpg	719.880	695.194	JPG File
1384.jpg	1.125.315	1.113.343	JPG File
1385.jpg	1.290.264	1.265.312	JPG File
1386.jpg	689.277	685.298	JPG File
1387.jpg	1.105.488	1.092.613	JPG File
1388.jpg	649.550	634.421	JPG File
1389.jpg	735.660	725.245	JPG File
1390.jpg	1.596.132	1.570.738	JPG File
1391.jpg	1.600.167	1.579.557	JPG File

Figura 4.3: Proceso de renombrado de imágenes

```
6
7  #Imágenes seleccionadas para la validación
8  val_data = object_detector.DataLoader.from_pascal_voc(
9      '/WorkingDirectory/images/test/jpg',
10     '/WorkingDirectory/images/test/xml',
11     ['aircraft','engine','cockpit','tail'])
```

Una vez cargadas las imágenes debemos elegir la arquitectura que seguirá el modelo para poder entrenarlo. Para ello, Tensorflow proporciona una tabla con el rendimiento de las diferentes arquitecturas disponibles, como se puede ver en la tabla 4.1.

Arquitectura	Tamaño	Latencia	Precisión
EfficientDet-Lite0	4,4	146	25,69 %
EfficientDet-Lite1	5,8	259	30,55 %
EfficientDet-Lite2	7,2	396	33,97 %
EfficientDet-Lite3	11,4	716	37,70 %
EfficientDet-Lite4	19,9	1886	41,96 %

Tabla 4.1: Arquitecturas Tf-Lite

La latencia proporcionada está medida usando una Raspberry Pi 4 usando 4 hilos de CPU.

Tras hacer pruebas y comparar los resultados obtenidos en tiempo de ejecución, en nuestro modelo hemos decidido utilizar EfficientDet-Lite2.

Una vez cargada la arquitectura comenzamos a entrenar el modelo. A la función de la API encargada de esta tarea se le deben proporcionar los parámetros *Epoch*, que indicará el número de iteraciones que tendrá sobre el conjunto de imágenes y *batch_*size, que indica la cantidad de imágenes por paquete que se entrenará. También indicaremos al entrenador si queremos que realice el proceso con el modelo completo.

Finalmente evaluamos el modelo generado para comprobar cómo de buenos son nuestros

Figura 4.4: Proceso de etiquetado de imágenes

resultados

```
#Carga la arquitectura del modelo antes de entrenarlo
spec = model_spec.get('efficientdet_lite2')


#Entrena el modelo
model = object_detector.create(train_data,                  #Datos de entrenamiento
                               model_spec=spec,             #Arquitectura del modelo
                               batch_size=4,                #Tamaño de los paquetes
                               train_whole_model=True,      #Entrenamiento completo
                               epochs=20,                   #Iteraciones
                               validation_data=val_data)    #Datos de validación

#Evalua el modelo
model.evaluate(val_data)

#Exporta el modelo como archivo tflite
model.export(export_dir='.', tflite_filename='aircraft.tflite')
```

Los resultados de la evaluación han sido los siguientes:

Etiqueta	Precisión
AP_/aircraft	0,7150532
AP_/cockpit	0,21296522
AP_engine	0,5451518
AP_/tail	0,6062568

Cuanto más grande es el tamaño de la etiqueta para cada objeto, mayor es la precisión del modelo en encontrarlos. Esto significa que para que el modelo pueda reconocer correctamente el *cockpit* se necesita un mayor número de imágenes que para reconocer una aeronave al completo.

Como hemos comentado anteriormente, la precisión y eficiencia del modelo podría mejorarse considerablemente con un *dataset* generado específicamente para ello, con un mayor número de imágenes y entrenando por separado las diferentes clases de objetos.

4.2.4. Reconocimiento de Imágenes

Para el reconocimiento de imágenes se ha utilizado la estructura de Tensorflow Lite básica para el reconocimiento de objetos. En nuestro caso, el código ha sido obtenido del repositorio de GitHub de Evan Juras, *TensorFlow-Object-Detection-on-the-Raspberry-Pi* (11). Sobre este código, se han aplicado numerosas modificaciones para adaptar el reconocimiento genérico al reconocimiento de aeronaves.

Modificaciones

Se han creado clases específicas para la detección de imagen, vídeo y movimiento de los motores. Se ha reestructurado todo el código **adaptándolo a las necesidades** del proyecto y eliminando aquellas funciones que no sean estrictamente necesarias.

Se ha modificado el código original para la escritura de *logs* que contienen información acerca de los objetos detectados y la precisión con la que detectó cada uno de estos.

```
# Press any key to continue to next image, or press 'q' to quit
# Wait 'ESC' to quit or 'S' to save the log of objects
saved = False
k = cv2.waitKey(0)
while k != 27:
    k = cv2.waitKey(0)
    if (k == 83 or k == 115) and (not saved):
        # Used to know number os detected objects
        file_object = open(os.path.join(CWD_PATH, output_log_name), 'w')
        for i in range(len(scores)):
            if ((scores[i] > threshold) and (scores[i] <= 1.0)):
                file_object.write(self.labels[int(self.classes[i])])
                file_object.write(" : ")
                file_object.write(str(scores[i]))
```

```
                file_object.write("\n")
        file_object.close()

    print("Image saved successfully")
    saved = True

# Clean up
cv2.destroyAllWindows()
```

El código proporcionado se ha modificado para devolver por consola los elementos encontrados junto con una imagen que se renderiza en pantalla.

Se implementa un mínimo de probabilidad de acierto en la detección del 50 %. Esto significa que los objetos que detecte el modelo serán al menos del 50 % de confianza.

Se ha creado una función para la detección de las diferentes partes del avión en orden de preferencia en la detección. De esta manera, el modelo siempre escogerá la partes disponible que mejor le convenga para la detección.

```
def get_position(self, x_center, y_center):
    ret = ()
    best = ("", -1)

    #Iterate over detected objects
    for j in self.object_positions:
        object_name = self.labels[int(self.self.classes[j])]

        #Object classification by decrasing prevalence
        #Cockpit -> Engine -> Tail -> Aircraft -> None
        if object_name == "cockpit" and best[0] == "":
            best[0] = "cockpit"
            best[1] = j
        elif object_name == "engine" and best[0] != "cockpit"
        and best[0] != "engine":
            best[0] = "engine"
            best[1] = j
        elif object_name == "tail" and best[0] != "cockpit"
        and best[0] != "engine" and best[0] != "tail":
            best[0] = "tail"
            best[1] = j
        elif object_name == "aircraft" and best[0] != "cockpit"
        and best[0] != "engine" and best[0] != "tail"
        and best[0] != "aircraft":
            best[0] = "aircraft"
            best[1] = j
```

También se ha creado una función para la detección de la posición de las aeronaves con respecto al centro de la cámara. Esta función servirá en la detección de distancias con los objetos ya que será la encargada de indicar dónde deben moverse los motores.

```
#Given a "best" object, calculate its position
x_axis = ""; y_axis = ""
if best[0] != "":
    x_center = int( min(self.imH, (self.boxes[int(self.classes[best[1]])][2]
    * self.imH)) - max(1, (self.boxes[int(self.classes[best[1]])][0]
    * self.imH)))
    y_center = int(min(self.imW, (self.boxes[int(self.classes[best[1]])][3]
    * self.imW)) - max(1, (self.boxes[int(self.classes[best[1]])][1]
    * self.imW)))

    #Clasification
    if (x_center > (self.width/2)-50) and (x_center < (self.width/2)+50):
        x_axis = "CENTERED"
    elif x_center > (self.width/2)+50:
        x_axis = "RIGHT"
    else:
        x_axis = "LEFT"

    if (y_center > (self.height/2)-20) and (y_center < (self.height/2)+20):
        y_axis = "CENTERED"
    elif y_center > (self.height/2)+ 20:
        y_axis = "DOWN"
    else:
        y_axis = "UP"
else:
    print("No se ha encontrado ningún objeto")

return x_axis, y_axis
```

Veamos a continuación las imágenes de entrada 4.5 y salida 4.6 del programa junto con el log 4.7 creado para comprobar su funcionamiento.

4.2.5. Reconocimiento de vídeo en tiempo real

Esta parte del código utiliza la misma dinámica que la sección de código anterior. Adjuntamos el extracto de código en el Anexo I.

El número de fotogramas por segundo (FPS) al que hemos conseguido hacer funcional el modelo es bastante bajo en ordenador (Utilizando un sistema Windows). Se espera conseguir un mejor rendimiento tras su implementación en la RaspberryPi junto con

FIGURA 4.5: Entrada del progrma

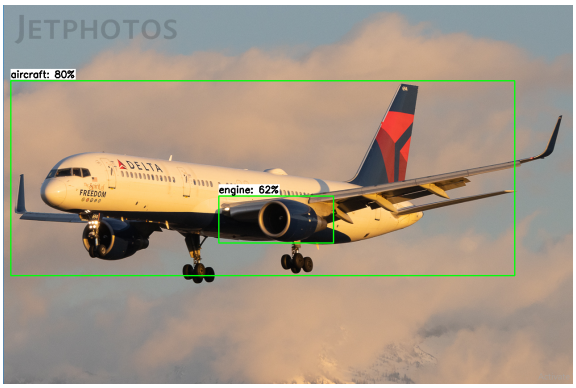


FIGURA 4.6: Salida del progrma

librerías específicas (USB Acelerator)[4].

Otro de los problemas encontrados es la sobrecarga de *frames* en espera de ser analizados. En el caso de la carga en Raspberry Pi existen funciones específicas en la que se puede indicar el número de fotogramas por segundos del vídeo capturado. En el caso de Windows, al utilizar la función *cv2.VideoCapture* esta funcionalidad no está disponible y, en consecuencia, el modelo tiene problemas para manejar la memoria y los hilos en los que trabaja.

El código generado se aprovechará de las mejoras realizadas sobre el módulo de imagen que hemos visto. Se contempla la alternativa de utilizar el sistema de imagen con un bucle que realice imágenes cada 3 segundos, de forma que el sistema no se sature.

4.2.6. Medida de distancias

En este proyecto hemos contemplado tres posibilidades a la hora de medir distancias con objetos:

- Utilizando una sola cámara y conociendo la anchura del objeto al que apuntamos.
- Utilizando dos cámaras y comparando el ángulo de visión entre ambas.

[4] https://coral.ai/

```
aircraft  :  80.859375 %
engine  :  62.890625 %
```

Figura 4.7: Salida del log del progrma

- Utilizando un medidor láser móvil junto a una cámara que apunten a un punto concreto del objeto.

Como hemos comentado al inicio del capítulo, se ha decidido utilizar un medidor láser guiado por una cámara que controle la posición de los objetos mediante visión por computador. Como hemos visto en la parte de reconocimiento de imágenes, el láser será guiado a una de las partes mas relevantes de un avión, que son, por orden de importancia:

1. Cockpit
2. Engine
3. Tail
4. Aircraft

Como se puede comprobar en las secciones anteriores, la clase generada contiene un método de detección de posición de un objeto. Esto nos sirve para, con una clase auxiliar en Python que nos permite mover stepper motors, guiar el láser junto con la cámara para centrar el objeto deseado.

Este código utiliza la librería AdvPiStepper[5]. Esta librería contiene los dirvers necesarios para mover motores stepper en una Raspberry Pi, usando la librería PIGPIO.

Veámos a continuación el código generado:

```
1  import advpistepper
2
3  class Stepper_Motors():
4      STEP = 10
5
6      #Stepper motor pins configuration
7      def __init__(self, dir_pin_1, dir_pin_2, step_pin_1, step_pin_2):
8          self.s1 = apis.AdvPiStepper(apis.DriverStepDirGeneric(
9              step_pin=step_pin_1, dir_pin=dir_pin_1))
10         self.s2 = apis.AdvPiStepper(apis.DriverStepDirGeneric(
11             step_pin=step_pin_1, dir_pin=dir_pin_1))
12
13     #Dir = 1:  FORWARD
14     #Dir = -1: BACKWARD
15
```

[5] https://github.com/innot/AdvPiStepper

```
    #Move the motor
    def move_stepper(self, id, dir):
        if id == "s1":
            self.s1.move(self.STEP*dir)
        elif id == "s2":
            self.s2.move(self.STEP*dir)
        else:
            print("There is no motor named ", id)

    def release_all(self):
        self.s1.release()
        self.s2.release()
```

El código que acabamos de ver debe ser controlado por un sistema **PID** de lazo cerrado para poder apuntar correctamente a los objetos.

La distancia a la que se deben mantener las aeronaves está relacionada con la velocidad a la que circulan.

4.3. Sistema MAPa

El sistema MAPa trata de reconocer una estructura de datos en la cual se presentan los datos de un aeropuerto concreto. Dado un documento en formato (.mp), el sistema del dron debe reconocer errores en el documento para, posteriormente, utilizar esos datos para moverse a lo largo del aeropuerto.

Para realizar el reconocimiento de errores en la estructura de datos utilizaremos un procesador del lenguaje realizando sus análisis léxico y sintáctico.

Se muestra a continuación un ejemplo de fichero de entrada (LEAS.mp)

```
LEAS
1204220900Z

COORDINATES
11:43.56648833002882, -6.046926704647996;
29:43.560656189796255, -6.0223108526660925;

A:43.5606089034433, -6.032428810223054;
B:43.55984178367669, -6.029205657042673;
C:43.56206140398197, -6.0317897115602195;
D1:43.56465491060213, -6.043565957606055;
D2:43.56492343822577, -6.041844445700936;

T01:43.564696725136166, -6.046390523319713;
```

```
T02:43.56399092264343, -6.04340436197854;
T03:43.56131610853287, -6.032118957500824;
T04:43.56055539290986, -6.028909027559575;
T05:43.55945465403994, -6.024253176643205

T1:43.56563785508267, -6.047408582647286;
T2:43.55979207998938, -6.022709051852848

PATHS
D:[D1,D2];
CN:[11,29];
CS:[29,11];
P-A29:[A,T03,T04,T05,T2];
P-B29:[A,T04,T05,T2];
P-A11:[A,T03,T02,T01,T1];
P-B11:[B,T04,T03,T02,T01,T1];
P-CA :[C,T03,A];
P-CB :[C,T03,T04,B];
P-DA :[D,T02,T03,A];
P-DA :[D,T02,T03,T04,B]

NOTAMS
//There is no active NOTAMS//
```

4.3.1. Analizador Léxico

Un analizador léxico es la primera fase de un compilador, consiste en un programa que recibe como entrada el código fuente de otro programa como secuencia de caracteres y produce una salida compuesta de símbolos (componentes léxicos).

La especificación del lenguaje dada incluye un conjunto de expresiones regulares que definen el léxico, es decir, que indican el conjunto de posibles secuencias de caracteres que definen un lexema.

Veamos la definición léxica de nuestra estructura de datos:

Definiciones léxicas

- airport: Cabecera e identificador del documento
- coordinates: Sección 1
- paths: Sección 2
- notams: Sección 3
- time: Hora a la que se actualizó el documento

- cId: ID de una coordenada
- pId: ID de un camino
- cAp: Corchete de apertura
- cCi: Corchete de cierre
- coma: Separador de coordenadas pc: Separador de instrucciones
- coord: Coordenada
- text: Texto plano, para los notams
- EOF: fin de archivo.

Especificación formal del léxico del lenguaje

Veamos a continuación la especificación formal utilizando expresiones regulares. Para ello diferenciaremos dos bloques. El primer bloque contendrá aquellas clases léxicas básicas. En un segundo bloque especificaremos aquellas clases que requieran de otras auxiliares (no básicas). Todo esto siguiendo las especificaciones en lenguaje no formal proporcionadas por el lenguaje.

- airport → letra letra letra letra
- coordinates → COORDINATES
- paths → PATHS
- notams → NOTAMS
- time → num num num num num num num num num num num letra
- cId → (num num | letra num num | letra letra num num | letra num)\ :
- pId → P \− (letra|num)* \ :
- cAp → \[
- cCi → \]
- coma → \,
- pc → \;
- coord → (-)?(num | num num|num num)\.num(num)*
- text → \/ \/ (letra | num |espacio)* \/ \/
- letra → [a-z, A-Z]
- num → [0-9]
- espacio → $\backslash n, \backslash t, \backslash r, \backslash b$,’ ’

Diseño de un analizador léxico para el lenguaje mediante un diagrama de transiciones.

Hemos utilizado el programa JFLAP (12) para realizar el diagrama de transiciones de la figura 4.8.

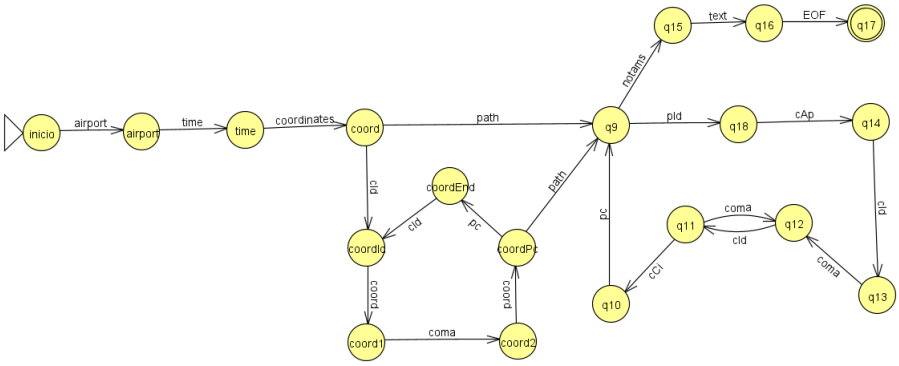


Figura 4.8: Autómata AFD de MAPa

El código se ha generado de manera automática incluyendo la especificación formal del lenguaje en un fichero de configuración (Analizador.flex) utilizando la herramienta JFlex (13).

```
1  package alex;
2
3  %%
4  %line
5  %column
6  %class AnalizadorLexicoMAP
7  %type  UnidadLexica
8  %unicode
9
10 %{
11   private ALexOperations ops;
12   public String lexema() {return yytext();}
13   public int fila() {return yyline+1;}
14   public int columna() {return yycolumn+1;}
15 %}
16
17 %eofval{
18   return ops.unidadEof();
19 %eofval}
20
21 %init{
22   ops = new ALexOperations(this);
23 %init}
24
```

```
letra  = ([A-Z]|[a-z])
num = [0-9]
espacio = \n|\t|\r|\b|[ ]
cAp = \[
cCi = \]
coma = \,
pc = \;
airport = {letra}{letra}{letra}{letra}
coordinates = COORDINATES
paths = PATHS
notams = NOTAMS
pId = P \- ({letra}|{num})* \:
cId = ({num}{num}|{letra}{num}{num}|{letra}{letra}{num}{num}|{letra}{num})[\:]?
coord = [\-]?({num}|{num}{num})\.{num}({num})*
text = \/\/({letra}|{num}|{espacio})* \/\/
time = {num}{num}{num}{num}{num}{num}{num}{num}{num}{num}{letra}

%%
{espacio}       {}
{airport}       {return ops.unidadAirport();}
{coordinates}   {return ops.unidadCoordinates();}
{paths}         {return ops.unidadPaths();}
{notams}        {return ops.unidadNotams();}
{time}          {return ops.unidadTime();}
{cId}           {return ops.unidadCid();}
{pId}           {return ops.unidadPid();}
{cAp}           {return ops.unidadCap();}
{cCi}           {return ops.unidadCci();}
{coma}          {return ops.unidadComa();}
{coord}         {return ops.unidadCoord();}
{text}          {return ops.unidadText();}
[^]             {ops.error();}
```

Los resultados obtenidos 4.9 tras el análisis es una lista de clases léxicas que utilizaremos en el analizador sintáctico para comprobar si el orden de las instrucciones es el correcto.

4.3.2. Analizador sintáctico

El analizador sintáctico (parser) se asegura de que el código se traduce correctamente al lenguaje deseado. Para la creación de este sistema utilizaremos la herramienta Proletool[6], que comprobará la eficacia y correcta generación de la especificación. La implementación en Java se realizará de manera manual.

[6] https://portal.esi.uclm.es/proletool

```
<terminated> MAPa [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  (3 may 2022 19:54:37 – 19:54:37) [pid: 5600]
[clase:AIRPORT,fila:1,col:1]
[clase:TIME,fila:2,col:1,lexema:1204220900Z]
[clase:COORDINATES,fila:4,col:1]
[clase:CID,fila:5,col:1,lexema:11:]
[clase:COORD,fila:5,col:4,lexema:43.56648833002882]
[clase:COMA,fila:5,col:21]
[clase:COORD,fila:5,col:23,lexema:-6.046926704647996]
[clase:CID,fila:6,col:1,lexema:29:]
[clase:COORD,fila:6,col:4,lexema:43.560656189796255]
[clase:COMA,fila:6,col:22]
[clase:COORD,fila:6,col:24,lexema:-6.0223108526660925]
[clase:CID,fila:8,col:1,lexema:A01:]
[clase:COORD,fila:8,col:5,lexema:43.5606089034433]
[clase:COMA,fila:8,col:21]
[clase:COORD,fila:8,col:23,lexema:-6.032428810223054]
[clase:CID,fila:9,col:1,lexema:B01:]
[clase:COORD,fila:9,col:5,lexema:43.55984178367669]
[clase:COMA,fila:9,col:22]
[clase:COORD,fila:9,col:24,lexema:-6.029205657042673]
[clase:CID,fila:10,col:1,lexema:C01:]
[clase:COORD,fila:10,col:5,lexema:43.56206140398197]
[clase:COMA,fila:10,col:22]
[clase:COORD,fila:10,col:24,lexema:-6.0317897115602195]
[clase:CID,fila:11,col:1,lexema:D01:]
[clase:COORD,fila:11,col:5,lexema:43.56465491060213]
[clase:COMA,fila:11,col:22]
[clase:COORD,fila:11,col:24,lexema:-6.043565957606055]
[clase:CID,fila:12,col:1,lexema:D02:]
[clase:COORD,fila:12,col:5,lexema:43.56492343822577]
[clase:COMA,fila:12,col:22]
```

Figura 4.9: Resultados de la ejecución del analizador léxico

Especificación sintáctica

Para realizar la especificación sintáctica primero debemos definir la tabla de prioridades de los operadores. En nuestro caso, al tratarse de una estructura de datos muy simple no se requiere. Trabajaremos directamente con las clases léxicas.

Definimos las clases léxicas en color verde.

Sp → S **EOF**
S → **AIRPORT TIME** C P N

C → **COORDINATES** COORDS
COORDS → COORDS **PC** CRD | CRD
CRD → **COORD COMA COORD**

P → **PATHS** PAS
PAS → PAS **PC** PTH | PTH
PTH → **PID CAP** CLIST **CCI**
CCI → **CID COMA** CLIST | **CID**

N → **NOTAMS TEXT**

Adaptación a implementación descendente

Para adaptar el modelo generado a una implementación descendente tenemos que eliminar el factor común y la recursión a izquierdas de las instrucciones.

Eliminar factor común:

CLIST := cid coma CLIST | cid	CLIST := cid CLIST2; CLIST2 := coma CLIST; CLIST2 := ;

Eliminar recursión a izquierdas:

COORDS := COORDS pc CRD | CRD	COORDS := CRD COORDS2; COORDS2 := pc CRD COORDS2; COORDS2 := ;
PAS := PAS pc PTH | PTH	PAS := PTH PAS2; PAS2 := pc PTH PAS2; PAS2 := ;

Una vez especificado y adaptado el lenguaje ya podemos comprobar que la especificación es correcta y proceder a la implementación.

```
1  grammar mapa_1
2  {
3  analysis LL1;
4  nonterminal S, C, P, N, COORDS, COORDS2, CRD, PAS, PAS2, PTH, CLIST, CLIST2;
5  terminal airport, time, coordinates, pc, coord, coma, paths, pid, cap, cid, cci,
6  notams, text;
7
8  S := airport time C P N;
9
10 C := coordinates COORDS;
11 COORDS := CRD COORDS2;
12 COORDS2 := pc CRD COORDS2;
13 COORDS2 := ;
14 CRD := coord coma coord;
15
16 P := paths PAS;
17 PAS := PTH PAS2;
18 PAS2 := pc PTH PAS2;
19 PAS2 := ;
20 PTH:=  pid cap CLIST cci;
21 CLIST := cid CLIST2;
22 CLIST2 := coma CLIST;
23 CLIST2 := ;
24
```

```
25 N := notams text;
26 }
```

Adjuntamos finalmente la clase AnalizadorLexico.java en el Anexo II. En el estracto de código se puede comprobar en código java la implementación que se ha seguido.

En la figura 4.10 se puede comprobar que programa devuelve un mensaje de confirmación en caso de que el lenguaje sea correcto. En caso de error devuelve la línea y posición en la que se encontró el error.

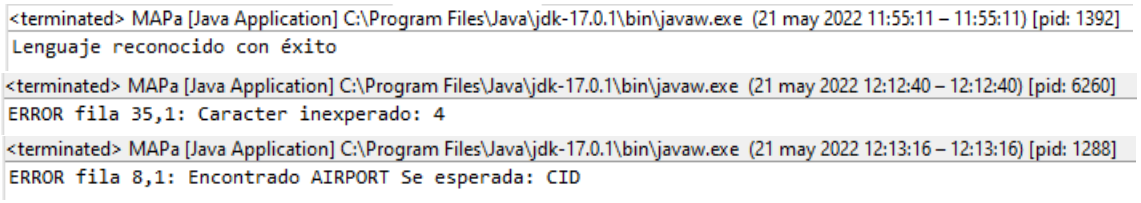


Figura 4.10: Resultado de la ejecución del analizador sintáctico

Demostramos así que se ha creado con éxito una estructura de datos capaz de guiar a cualquier vehículo por un aeropuerto siguiendo un sistema basado en puntos. Se ha creado un analizador capaz de reconocer dichas estructuras e informar de errores en su creación.

Capítulo 5

Modificaciones

Se proponen a continuación diferentes modificaciones para la ampliación del proyecto de cara a futuras versiones. Estas mejoras pueden afectar a la eficiencia, seguridad o ámbito del proyecto.

- Mejora de MAPa

  Se pretende automatizar el sistema MAPa para que este reconozca las puertas de embarque, y sepa moverse por los aeropuertos sin estar guiando un avión.

- Mejora del modelo IA

  Actualmente el modelo que utiliza el sistema GUIa sólamente reconoce aeronaves y sus partes. Se pretende mejorar este sistema para que reconozca otros tipos de aeronaves y objetos, así como personas y vehículos e implementar un comportamiento inteligente para poder actuar frente a situaciones en las que interacciones este tipo de objetos.

- Coorinación entre drones

  Implementar un sistema mediante el cual los drones reconozcan la presencia de otros drones para coordinar mejor las labores de guiado, evitar accidentes y mejorar los tiempos de espera. Este sistema deberá calcular el tiempo estimado que tardará el dron en llegar a cada cruce y, en caso de encontrarse con otro, evaluar posibles caminos alternativos.

- Distancia vs Velocidad

  En el modelo final se debe implementar un sistema mediante el cual la distancia que el dron toma con la aeronave que guía sea proporcional a la velocidad que esta lleve. Esto se tendrá también en cuenta para el tiempo de frenado requerido por la misma.

- Creacicón de un modelo

Utilizando una Raspberry PI 3 se busca crear un modelo de dron que sea capaz de realizar el comportamiento inteligente estudiado en este trabajo. Este comportamiento puede realizarse utilizando ROS.

Conclusiones

El trabajo realizado ha concluido en la posibilidad de utilizar drones autónomos para realizar la tarea de guiado de aeronaves en zonas aeroportuarias. Se ha analizado el uso de estos aparatos dando como resultado posibles problemas a los que se les ha buscado una solución. Estos posibles problemas son:

- Pesos
- Combustible y tiempos de taxi
- Vientos

Se ha diseñado un sistema de carga de drones (ERDG) que es capaz de reabastecer de manera automática y segura estos aparatos. Para aumentar la seguridad de este proceso se ha diseñado un sarcófago que se enterrará parcialmente y donde se realizará la operación.

Para el sistema de guiado, se ha diseñado un modelo de reconocimiento de objetos basado en Tensorflow (4) que es capaz de reconocer las diferentes partes de una aeronave. Este sistema ha resultado en un número considerable de problemas debido a la necesidad de utilizar un modelo tflite que sea capaz de ser cargado en sistemas con pocos recursos (como una RaspberryPI). Para probar el funcionamiento de este sistema se ha confeccionado junto a él un código que realiza el reconocimiento de objetos en una imagen y en un vídeo. Estos códigos serán fácilmente reutilizables para su uso con cámaras web.

Los resultados del experimento han sido un satisfactorio reconocimiento de aeronaves, teniendo que mejorar en sí el modelo para que aprenda mejor los objetos y reconozca con más velocidad. En cuanto al reconocimiento en tiempo real, la velocidad de reconocimiento es realmente baja, con una media de 0.2 fotogramas por segundo en una máquina Windows. Se espera que este sistema mejore con el uso de bibliotecas específicas en coordinación con el sistema específico RaspberryPI

Finalmente se ha creado una estructura de datos que recoja el sistema de guiado por un aeropuerto y que será leída por el dron. Para la correcta lectura de dicha estructura se ha creado un analizado léxico (utilizando jFlex) para la detección de errores.

En el siguiente capítulo se exponen las mejoras que se espera realizar en futuras versiones del proyecto ya que en la versión actual no se ha dispuesto de tiempo suficiente para la realización de sistemas que se consideran importantes. Por otro lado, tampoco se ha dispuesto de los materiales necesarios para la creación de un modelo.

Conclusions

The work done has concluded in the possibility of using autonomous drones to guiding aircraft in airport areas. The use of these machines has been analyzed, resulting in possible problems that were solved. These possible problems were:

- Weigths
- Fuel and taxi times
- Winds

Se ha diseñado un sistema de carga de drones (ERDG) que es capaz de reabastecer de manera automática y segura estos aparatos. Para aumentar la seguridad de este proceso se ha diseñado un sarcófago que se enterrará parcialmente y donde se realizará la operación.

Drone charge system (ERDG) has been designed. This system can recharge and refuel drones in a secure and automated way. To increase the security, a box (named sarcophagus) has been designed. This box, where the operations will be carried out, will be partially bured.

For the guidance system, an object recognition model based on Tensorflow has been designed. This system allow to recognize different parts of an aircraft. This system give us a lot of problems due to the need to use a tflite model, that is capable of being loaded on a few resoruces systems (such as a RaspberryPI). In order to test this system, a code that recognizes objects in an image and in a video has been created. These codes will be easily reusable using webcams.

The experiment results concluded in a satisfactory aircraft parts recognition. The model has to be improved to learn more objects in a better way, and to recognize objects faster. The recognition speed is really low using the model in real-time recognition (average 0.2 fps in a Windows machine). This system is expected to improve with the use of specific libraries in coordination with the specific RaspberryPI system.

Finally, a data structure has been created. This structure will be read by drones and collect the guidance system. For the correct reading of said structure, a lexical analyzer has been created (using jFlex) for error detection.

Anexo I

Detección de imagen

Extracto del código generado

```
######## Image Object Detection Using Tensorflow-trained Classifier #########

# Author: Sergio Ramos modification over Evan Juras code
# Surce code date: 15/01/18
# Modification date: 05/02/22

#Constants and PATHS decalrations
threshold = 0.5

CWD_PATH = os.getcwd()
MODEL_NAME = 'aircraft.tflite'
LABELMAP_NAME = 'labelmap.txt'

class AircraftDetector():
    def __init__(self):
        # Import TensorFlow libraries
        # Get model details
        # Load the label map

    def get_position(self, x_center, y_center):
        ret = ()
        best = ("", -1)

        #Iterate over detected objects
        for j in self.object_positions:
            object_name = self.labels[int(self.self.classes[j])]

            #Object classification by decrasing prevalence
            #Cockpit -> Engine -> Tail -> Aircraft -> None
            if object_name == "cockpit" and best[0] == "":
                best[0] = "cockpit"
```

```
32                  best[1] = j
33              elif object_name == "engine" and best[0] != "cockpit"
34              and best[0] != "engine":
35                  best[0] = "engine"
36                  best[1] = j
37              elif object_name == "tail" and best[0] != "cockpit"
38              and best[0] != "engine" and best[0] != "tail":
39                  best[0] = "tail"
40                  best[1] = j
41              elif object_name == "aircraft" and best[0] != "cockpit"
42              and best[0] != "engine" and best[0] != "tail"
43              and best[0] != "aircraft":
44                  best[0] = "aircraft"
45                  best[1] = j
46
47          #Given a "best" object, calculate its position
48          x_axis = ""; y_axis = ""
49          if best[0] != "":
50              #Calculate the center of the object
51              x_center = int( min(self.imH, (self.boxes[int(self.classes[best[1]])][2]
52              * self.imH)) - max(1, (self.boxes[int(self.classes[best[1]])][0]
53              * self.imH)))
54              y_center = int(min(self.imW, (self.boxes[int(self.classes[best[1]])][3]
55              * self.imW)) - max(1, (self.boxes[int(self.classes[best[1]])][1]
56              * self.imW)))
57
58              #Clasification in quadrants, it aplies a 20px margin
59              #to avoid PID problems
60              if (x_center > (self.width/2)-50) and (x_center < (self.width/2)+50):
61                  x_axis = "CENTERED"
62              elif x_center > (self.width/2)+50:
63                  x_axis = "RIGHT"
64              else:
65                  x_axis = "LEFT"
66
67              if (y_center > (self.height/2)-20) and (y_center < (self.height/2)+20):
68                  y_axis = "CENTERED"
69              elif y_center > (self.height/2)+ 20:
70                  y_axis = "DOWN"
71              else:
72                  y_axis = "UP"
73          else:
74              print("No se ha encontrado ningún objeto")
75
76
77          #We can connect output with another parts of the program
78          return x_axis, y_axis
79
```

```
80
81  ###################### This part of the code belongs Evan Juras ######################
82      def image_detection(self, output_log_name, input_image_name):
83          input_mean = 127.5
84          input_std = 127.5
85
86          # Check output layer name to determine if this model was created with TF2
87          # or TF1, because outputs are ordered differently for TF2 and TF1 models
88          outname = self.output_details[0]['name']
89
90          if ('StatefulPartitionedCall' in outname): # This is a TF2 model
91              boxes_idx, classes_idx, scores_idx = 1, 3, 0
92          else: # This is a TF1 model
93              boxes_idx, classes_idx, scores_idx = 0, 1, 2
94
95
96          # Load image and resize to expected shape
97          image = cv2.imread(input_image_name)
98          image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
99          self.imH, self.imW, _ = image.shape
100         image_resized = cv2.resize(image_rgb, (self.width, self.height))
101         input_data = np.expand_dims(image_resized, axis=0)
102
103         # Normalize pixel values if using a floating model
104         if self.floating_model:
105             input_data = (np.float32(input_data) - input_mean) / input_std
106
107         # Perform the actual detection by running the model with the image as input
108         self.interpreter.set_tensor(self.input_details[0]['index'],input_data)
109         self.interpreter.invoke()
110
111         # Retrieve detection results
112
113         # Bounding box coordinates of detected objects
114         self.boxes = self.interpreter.get_tensor(
115             self.output_details[boxes_idx]['index'])[0]
116         # Class index of detected objects
117         self.classes = self.interpreter.get_tensor(
118             self.output_details[classes_idx]['index'])[0]
119         # Confidence of detected objects
120         scores = self.interpreter.get_tensor(
121             self.output_details[scores_idx]['index'])[0]
122
123         # Loop over all detections and draw detection box if confidence is
124         #above minimum threshold
125         object_positions = np.array([],dtype=np.int8)
126         for i in range(len(scores)):
127             if ((scores[i] > threshold) and (scores[i] <= 1.0)):
```

```
                print(self.labels[int(self.classes[i])], scores[i] * 100, '%')

                # Get bounding box coordinates and draw box
                # self.interpreter can return coordinates that are outside of image
                # dimensions, need to force them to be within image using
                #max and min
                ymin = int(max(1,(self.boxes[i][0] * self.imH)))
                xmin = int(max(1,(self.boxes[i][1] * self.imW)))
                ymax = int(min(self.imH,(self.boxes[i][2] * self.imH)))
                xmax = int(min(self.imW,(self.boxes[i][3] * self.imW)))

                cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

                # Draw label
                # Look up object name from "labels" array using class index
                object_name = self.labels[int(self.classes[i])]
                # Example: 'person: 72%'
                label = '%s: %d%%' % (object_name, int(scores[i]*100))
                 # Get font size
                labelSize, baseLine = cv2.getTextSize(label,
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)
                # Make sure not to draw label too close to top of window
                label_ymin = max(ymin, labelSize[1] + 10)
                # Draw white box to put label text in
                cv2.rectangle(image,
                    (xmin, label_ymin-labelSize[1]-10),
                    (xmin+labelSize[0], label_ymin+baseLine-10),
                    (255, 255, 255), cv2.FILLED)
                # Draw label text
                cv2.putText(image, label, (xmin, label_ymin-7),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)

                object_positions = np.append(object_positions, int(i))
############################### End of Evan Juras part ################################


        # All the results have been drawn on the image, now display the image
        cv2.imshow('Object detector', image)

        # Press any key to continue to next image, or press 'q' to quit
        # Wait 'ESC' to quit or 'S' to save the log of objects
        saved = False
        k = cv2.waitKey(0)
        while k != 27:
            k = cv2.waitKey(0)
            if (k == 83 or k == 115) and (not saved):
                # Used to know number os detected objects
                file_object = open(os.path.join(CWD_PATH, output_log_name), 'w')
```

```
                for i in range(len(scores)):
                    if ((scores[i] > threshold) and (scores[i] <= 1.0)):
                        file_object.write(self.labels[int(self.classes[i])])
                        file_object.write(" : ")
                        file_object.write(str(scores[i]))
                        file_object.write("\n")
                file_object.close()

            print("Image saved successfully")
            saved = True

        # Clean up
        cv2.destroyAllWindows()
```

Detección de video

```
class VideoDetection():
    x_center = -1
    y_center = -1

    def __init__(self):
        # Import TensorFlow libraries
        # Get model details
        # Load the label map


####################### This part of the code belongs Evan Juras ######################
    def video_detector(self, video_path, log_path):
        input_mean = 127.5
        input_std = 127.5

        # Check output layer name to determine if this model was created with
        #TF2 or TF1
        outname = self.output_details[0]['name']

        if ('StatefulPartitionedCall' in outname):  # This is a TF2 model
            boxes_idx, classes_idx, scores_idx = 1, 3, 0
        else:  # This is a TF1 model
            boxes_idx, classes_idx, scores_idx = 0, 1, 2

        # Open video file
        video = cv2.VideoCapture(video_path)
        imW = video.get(cv2.CAP_PROP_FRAME_WIDTH)
        imH = video.get(cv2.CAP_PROP_FRAME_HEIGHT)
```

```
        log = ""
        num_frame = 1

        while (video.isOpened()):
            time.sleep(1)
            log += "Frame " + str(num_frame) + "\n"
            num_frame+=1

            ret, frame = video.read()


            if not ret:
                print('Reached the end of the video!')
                break
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frame_resized = cv2.resize(frame_rgb, (self.width, self.height))
            input_data = np.expand_dims(frame_resized, axis=0)

            # Normalize pixel values if using a floating model
            if self.floating_model:
                input_data = (np.float32(input_data) - input_mean) / input_std

            # Perform the actual detection by running the model with image as input
            self.interpreter.set_tensor(self.input_details[0]['index'], input_data)
            self.interpreter.invoke()

            # Bounding box coordinates of detected objects
            self.boxes = self.interpreter.get_tensor(
                self.output_details[boxes_idx]['index'])[0]
            # Class index of detected objects
            self.classes = self.interpreter.get_tensor(
                self.output_details[classes_idx]['index'])[0]
            # Confidence of detected objects
            scores = self.interpreter.get_tensor(
                self.output_details[scores_idx]['index'])[0]

            # Loop over all detections and draw detection box if confidence
            #is above minimum threshold
            for i in range(len(scores)):

                if ((scores[i] > threshold) and (scores[i] <= 1.0)):
                    #Sistema de log implementado por Sergio Ramos
                    log += self.labels[int(classes[i])] + ": " +
                            str(scores[i] * 100) + '%\n'
                    # Get bounding box coordinates and draw box
                    # Interpreter can return coordinates that are outside of image
                    #dimensions, need to force them to be within image using
```

```
                    #max() and min()
                    ymin = int(max(1, (boxes[i][0] * imH)))
                    xmin = int(max(1, (boxes[i][1] * imW)))
                    ymax = int(min(imH, (boxes[i][2] * imH)))
                    xmax = int(min(imW, (boxes[i][3] * imW)))

                    cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (10, 255, 0), 4)

                    # Draw label
                    # Look up object name from "labels" array using class index
                    object_name = self.labels[int(self.classes[i])]
                    # Example: 'person: 72%'
                    label = '%s: %d%%' % (object_name, int(scores[i]*100))
                     # Get font size
                    labelSize, baseLine = cv2.getTextSize(label,
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)
                    # Make sure not to draw label too close to top of window
                    label_ymin = max(ymin, labelSize[1] + 10)
                    # Draw white box to put label text in
                    cv2.rectangle(image,
                        (xmin, label_ymin-labelSize[1]-10),
                        (xmin+labelSize[0], label_ymin+baseLine-10),
                        (255, 255, 255), cv2.FILLED)
                    # Draw label text
                    cv2.putText(image, label, (xmin, label_ymin-7),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)


            #Show result
            cv2.imshow('Object detector', frame)

            # Press 'q' to quit -> Write log
            if cv2.waitKey(1) == ord('q'):
                file_object = open(os.path.join(CWD_PATH, log_path), 'w')
                file_object.write(log)
                break

        # Clean up
        video.release()
        cv2.destroyAllWindows()
```

Anexo II

Analizador sintáctico

```
1  public void Sp() {
2      S();
3      empareja(ClaseLexica.EOF);
4  }
5
6  private void S() {
7      switch(anticipo.clase()) {
8      case AIRPORT:
9          empareja(ClaseLexica.AIRPORT);
10         empareja(ClaseLexica.TIME);
11         C();
12         P();
13         N();
14         break;
15
16     default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
17                                      anticipo.clase(), ClaseLexica.AIRPORT);
18     }
19 }
20
21 private void C() {
22     switch(anticipo.clase()) {
23     case COORDINATES:
24         empareja(ClaseLexica.COORDINATES);
25         COORDS();
26         break;
27     default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
28                                      anticipo.clase(), ClaseLexica.COORDINATES);
29     }
30 }
31
32 private void COORDS() {
```

```
33      switch(anticipo.clase()) {
34      case CID:
35          CRD();
36          COORDS2();
37          break;
38      default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
39                                       anticipo.clase(), ClaseLexica.CID);
40      }
41  }
42
43  private void COORDS2() {
44      switch(anticipo.clase()) {
45      case PC:
46          empareja(ClaseLexica.PC);
47          CRD();
48          COORDS2();
49          break;
50      case PATHS:
51          break;
52      default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
53                                       anticipo.clase(), ClaseLexica.PC);
54      }
55  }
56
57  private void CRD() {
58      switch(anticipo.clase()) {
59      case CID:
60          empareja(ClaseLexica.CID);
61          empareja(ClaseLexica.COORD);
62          empareja(ClaseLexica.COMA);
63          empareja(ClaseLexica.COORD);
64          break;
65      default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
66                                       anticipo.clase(), ClaseLexica.CID);
67      }
68  }
69
70  //-----------------------------------------------------------------------------
71
72  private void P() {
73      switch(anticipo.clase()) {
74      case PATHS:
75          empareja(ClaseLexica.PATHS);
76          PAS();
77          break;
78      default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
79                                       anticipo.clase(), ClaseLexica.PATHS);
80      }
```

```
81  }
82
83  private void PAS() {
84      switch(anticipo.clase()) {
85      case PID:
86          PTH();
87          PAS2();
88          break;
89      default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
90                                              anticipo.clase(), ClaseLexica.PID);
91      }
92  }
93
94  private void PAS2() {
95      switch(anticipo.clase()) {
96      case PC:
97          empareja(ClaseLexica.PC);
98          PTH();
99          PAS2();
100         break;
101     case NOTAMS:
102         break;
103     default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
104                                         anticipo.clase(), ClaseLexica.PC);
105     }
106 }
107
108 private void PTH() {
109     switch(anticipo.clase()) {
110     case PID:
111         empareja(ClaseLexica.PID);
112         empareja(ClaseLexica.CAP);
113         CLIST();
114         empareja(ClaseLexica.CCI);
115         break;
116     default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
117                                         anticipo.clase(), ClaseLexica.PID);
118     }
119 }
120
121 private void CLIST() {
122     switch(anticipo.clase()) {
123     case CID:
124         empareja(ClaseLexica.CID);
125         CLIST2();
126         break;
127     default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
128                                         anticipo.clase(), ClaseLexica.CID);
```

```
129     }
130 }
131
132 private void CLIST2() {
133     switch(anticipo.clase()) {
134     case COMA:
135         empareja(ClaseLexica.COMA);
136         CLIST();
137         break;
138     case CCI:
139         break;
140     default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
141                                      anticipo.clase(), ClaseLexica.COMA);
142     }
143 }
144
145 //-------------------------------------------------------------------------------
146
147 private void N() {
148     switch(anticipo.clase()) {
149     case NOTAMS:
150         empareja(ClaseLexica.NOTAMS);
151         empareja(ClaseLexica.TEXT);
152         System.out.println("Lenguaje reconocido con éxito");
153         break;
154     default: errores.errorSintactico(anticipo.fila(), anticipo.columna(),
155                                      anticipo.clase(), ClaseLexica.NOTAMS);
156     }
157 }
158
159 //-------------------------------------------------------------------------------
160 private void empareja(ClaseLexica claseEsperada) {
161     if (anticipo.clase() == claseEsperada)
162         sigToken();
163     else errores.errorSintactico(anticipo.fila(), anticipo.columna(),
164                                  anticipo.clase(),claseEsperada);
165 }
166
167 private void sigToken() {
168     try {
169         anticipo = alex.yylex();
170     }
171     catch(IOException e) {
172         errores.errorFatal(e);
173     }
174 }
```

Bibliografía

[1] Rank Documentation - IVAO - International Virtual Aviation Organisation; 2020. [Online; accessed 3. May 2022]. Available from: https://mediawiki.ivao.aero/index.php?title=Rank_Documentation.

[2] Ground Jet Blast Hazard;. [Online; accessed 28. May 2022]. https://asrs.arc.nasa.gov/publications/directline/dl6_blast.htm.

[3] Sarasola Merino S. Sistema de reabastecimiento para drones [B.S. thesis]; 2019. https://e-archivo.uc3m.es/handle/10016/29734.

[4] TensorFlow;. [Online; accessed 28, May 2022]. https://www.tensorflow.org/.

[5] OlafenwaMoses. ImageAI; 2022. [Online; accessed 20. May 2022]. Available from: https://github.com/OlafenwaMoses/ImageAI.

[6] Home - OpenCV; 2022. [Online; accessed 20. May 2022]. Available from: https://opencv.org.

[7] TensorFlow Lite | AA para dispositivos móviles y perimetrales; 2022. [Online; accessed 15. May 2022]. Available from: https://www.tensorflow.org/lite.

[8] NumPy; 2022. [Online; accessed 20. May 2022]. Available from: https://numpy.org.

[9] tzutalin. labelImg. GitHub; 2018. https://github.com/tzutalin/labelImg.

[10] Detección de objetos con TensorFlow Lite Model Maker; 2022. [Online; accessed 4. Abr 2022]. Available from: https://www.tensorflow.org/lite/tutorials/model_maker_object_detection.

[11] EdjeElectronics. TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10. GitHub; 2020. https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi.

[12] JFLAP; 2022. [Online; accessed 4. May 2022]. Available from: https://www.jflap.org.

[13] Klein G. JFlex - JFlex The Fast Scanner Generator for Java; 2020. [Online; accessed 3. May 2022]. Available from: https://www.jflex.de.

Sergio Ramos Mesa
MAYO 2022
Ult. actualización 30 de mayo de 2022
LaTeX lic. LPPL & powered by CC-BY-NC-4.0