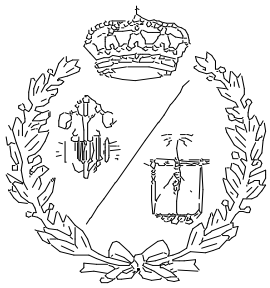
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Proyecto Fin de Máster

Planificación de trayectorias para el control autónomo de un cuadricóptero utilizando técnicas de visión artificial y aprendizaje profundo

(Path planning for the autonomous control of a quadropter aplying artifitial vision techniques and Deep Learning)

Para acceder al Título de

MÁSTER UNIVERSITARIO EN INGENIERIA INDUSTRIAL

Autor: Diego Álvaro Álvarez

Septiembre - 2021

RESUMEN

En este trabajo se abordará la problemática de la planificación de trayectorias y evasión de obstáculos en entornos dinámicos para vehículos autónomos no tripulados (UAV = unmanned aerial vehicle) en aplicaciones como transporte, mensajería, logística y operaciones de rescate.

Después de describir las herramientas y técnicas utilizadas, se desarrollará un prototipo sobre un cuadricóptero o drone (DJI Tello) que se programará para ser controlado manualmente mediante una interfaz de usuario desde un ordenador o poder moverse de forma autónoma introduciéndole las coordenadas del destino deseado, mientras construye un mapeado del entorno, registrando la trayectoria del drone y los obstáculos encontrados, así como la estimación de la posición y orientación de la aeronave mediante odometría.

Para la detección de obstáculos se implementará un modelo entrenado con aprendizaje profundo (Deep Learning) para reconocer en tiempo real, a partir de las imágenes que proporciona la cámara, distintos tipos de obstáculos como árboles, vallas, personas…

Para el movimiento autónomo se utilizará un algoritmo de distancia mínima para crear una ruta segura hasta el destino deseado evadiendo los obstáculos conocidos. Por tanto, cada vez que se encuentra un obstáculo nuevo, se actualizará el mapeado y, si es necesario, la ruta óptica hasta la localización objetivo.

Palabras clave: Planificación de trayectorias, evasión de obstáculos, DJI Tello, drone, robot aéreo autónomo, Deep Learning, D*Lite, detección de objetos, Red Neuronal Convolucional.

Todos los archivos (programas, videos, etc.) a los que se hacen referencia en este documento generados a lo largo del desarrollo de este trabajo se pueden consultar en el repositorio GitHub creado específicamente para este proyecto[1] [1].

[1] https://github.com/diegoalvaroalvarez/TFM2021UC-Diego-Alvaro

ABSTRACT

This project discusses the problematic of path planning and obstacle avoidance in unmanned autonomous vehicles (UAV) in dynamic environments in different application fields such as transportation, messenger service, logistics and rescue operations.

After describing what tools and techniques have been used, it will develop a prototype of a quadcopter or drone (DJI Tello) which will be programmed to be able to be controlled manually by an interface or to move autonomously just introducing the coordinates of a goal destination, while it is mapping the environment and keeping track of the path and obstacles. Also, it will estimate the current position and orientation of the aerial robot applying odometry techniques.

For the object detection task, it will use a model which has been trained applying Deep Learning techniques for recognizing in real time different types of obstacles such as trees, walls, people, etc. in images provided by the DJI Tello camera.

For the autonomous movement task, it will use a shortest path algorithm to elaborate a safe path to the goal destination avoiding known obstacles. Hence, every time it detects a new obstacle, it will map it and, if it is necessary, it will recalculate the optimal path to the goal location.

Keywords: Path planning, obstacle avoidance, DJI Tello, drone, auronomous aerial robot, Deep Learning, D*Lite, object detection, Convolutional Neuronal Network (CNN).

Every file developed for the purpose of this work could be found the GitHub repository created specifically for this project [1].

ÍNDICE

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

En la última década, el número de robots y sus aplicaciones prácticas han aumentado exponencialmente principalmente por su capacidad de hacer determinadas tareas de manera más sencilla o eficaz, entrando en un proceso de mejora continua. Hoy en día, es difícil pensar en un ámbito en la vida cotidiana donde no haya ninguna clase de robot. Siendo ésta una palabra un tanto abstracta que no solo abarca su caracterización popular de un ser humanoide metálico sino una definición que engloba un incontable número de artefactos, ya que, según la Real Academia Española, un robot es cualquier "máquina o ingenio electrónico programable que es capaz de manipular objetos y realizar diversas operaciones."

Especialmente, los robots móviles, aquellos capaces de moverse en un entorno y que no se encuentran fijados en una posición física, son realmente útiles en la industria. Por ejemplo, en campos tan divergentes como investigación, manufacturación, transporte y muchos más.

En este trabajo se abordará la problemática de la planificación de trayectorias y evasión de obstáculos para robots autónomos no tripulados en entornos dinámicos, así como el potencial de técnicas de inteligencia artificial como el aprendizaje profundo (Deep Learning) para un amplio rango de aplicaciones en la industria. Para ello, se hará una introducción a los conceptos y técnicas más importantes, posteriormente, se desarrollará un prototipo de un cuadricóptero capaz de solventar estas problemáticas y se estudiará su éxito de funcionamiento.

Esta memoria se encuentra dividida en 8 capítulos:

- Capítulo 1: Introducción.

  En el primer capítulo se explicarán brevemente las aplicaciones de los robots aéreos autónomos y del Deep Learning en la industria. Además, se establecerán los objetivos principales de este proyecto. Finalmente, se hará una breve descripción del robot aéreo o drone que se utilizará para este proyecto, el DJI Tello Drone.

- Capítulo 2: Deep Learning.

  En el segundo capítulo se introducirán los conceptos básicos de esta técnica, como qué es una red neuronal. Además, se explicará la estructura específica de red neuronal

utilizada en este proyecto, Red Neuronal Convolucional (CNN). También, se nombrarán las limitaciones principales del Deep Learning. Igualmente, se explicará la técnica de Machine Learning conocida como Transfer Learning. Finalmente, se describirán dos herramientas utilizadas para poder aplicar Deep Learning en este trabajo, TensorFlow y Google Colaboratory.

- Capítulo 3: Descripción detallada del modelo para detección de obstáculos.

En el tercer capítulo se describirá en profundidad tanto la estructura como el proceso de entrenamiento y pasos a seguir que se han llevado a cabo en el modelo entrenado con Deep Learning usado en este proyecto para la detección y clasificación de posibles obstáculos en tiempo real.

- Capítulo 4: Algoritmos de planificación de trayectorias y evasión de obstáculos.

En el cuarto capítulo se introducirán distintos algoritmos de planificación de trayectorias. Cada algoritmo que se cite estará fundamentado en el anterior, pero con innovaciones que se aproximen en mayor medida a las necesidades de este proyecto hasta llegar al algoritmo utilizado en este trabajo, el algoritmo D*Lite.

- Capítulo 5: Programa principal para el control autónomo del DJI Tello.

En el quinto capítulo se explicarán los distintos programas que se han ido elaborando para conseguir finalmente que el DJI Tello sea capaz de moverse autónomamente hasta un destino esquivando obstáculos y mapeando el entorno.

- Capítulo 6: Pruebas de evaluación del funcionamiento.

En el sexto capítulo se explicarán las pruebas que se van a llevar a cabo para comprobar la eficiencia del prototipo creado.

- Capítulo 7: Resultados de las pruebas

En este capítulo se comentarán los resultados obtenidos en las pruebas, así como cualquier conclusión o dato de interés obtenido en función de éstos.

- Capítulo 8: Conclusiones y trabajos posteriores.

En el último capítulo se enumerarán las conclusiones principales de este proyecto y se nombrarán algunas consideraciones a tener en cuenta para estudios futuros.

1.1 APLICACIONES EN LA INDUSTRIA

1.1.1 Robot aéreos autónomos en la industria

Actualmente, la utilización de robots móviles autónomos se encuentra en su pico más alto superándose año tras año y con una premisa de que va a continuar de igual manera por un largo periodo de tiempo. Sus principales ventajas competitivas son tanto la facilidad como la comodidad de su uso, además de su precisión y efectividad. El uso de este tipo de robots se puede apreciar en distintos campos de aplicación muy variados empezando por un robot aspirador doméstico, conocido popularmente como “Rumba” y continuando con robots más sofisticados utilizados en almacenes automatizados.

En este trabajo, se centrará la atención en aquellos utilizados para el transporte de mercancía, mensajería, logística o en operaciones de rescate y balance de daños. Más concretamente, se tratará de robots aéreos también conocidos como drones. A continuación, se van a describir algunos usos que se están dando a los robots autónomos aéreos en la industria.

Primero, en el sector de la agricultura una de sus principales funciones es la monitorización del estado de los cultivos, esta tarea se realiza mediante imágenes multiespectrales obtenidas a través de los sensores incorporados en las aeronaves, por ejemplo, sensores térmicos infrarrojos [2].

En el sector eléctrico, tienen gran aplicación en la revisión de líneas y torres eléctricas. De esta forma reducen el riesgo laboral de los operarios y no necesitan cortar el suministro eléctrico, grandes empresas como Iberdrola [3] o Red Eléctrica Española [4] realizan estas operaciones con drones para comprobar el estado de las líneas y nivel de corrosión y, establecer si necesitan ser reparadas o sustituidas. También, con sensores ultravioleta se puede detectar el efecto corona y así realizar un mantenimiento preventivo de las líneas. Esta

aplicación es utilizada de igual manera en huertas solares, para comprobar los paneles fotovoltaicos [5].

En el sector de transporte y logística existe un gran abanico de posibilidades para los drones. Grandes empresas como Amazon han realizado enormes inversiones en una red logística que les permita entregar rápidamente por medio de drones los productos adquiridos por sus clientes [6]. Las principales ventajas son una mayor rapidez de operación, así como una reducción de la huella ecológica en el proceso de transporte además de reducción de costes. Sin embargo, las legislaciones de los distintos países están frenando la implantación de esta técnica. Por otro lado, en la gestión de almacenes tienen una mayor libertad realizando tareas como gestión de inventario e inspeccionar y monitorizar el almacén, empresas como Danone o DHL utilizan drones en algunos de sus centros logísticos [7].

Los UAV son una herramienta que aporta grandes resultados para tareas de vigilancia y control con posibles aplicaciones en diferentes campos. Son capaces de realizar tareas demasiado peligrosas para un ser humano o en localizaciones a las que un ser humano no puede acceder. Un ejemplo donde se ha podido ver la utilización de las aeronaves ha sido en el accidente nuclear de Fukushima, Japón para obtener una vista precisa del interior del reactor nuclear [8]. Otras aplicaciones más generales son en operaciones de estudio y reducción de daños y rescate en desastres naturales, como en el terremoto de 2010 de Haití [9], o búsqueda de personas en zonas montañosas o nevadas aportando una opción más rápida y de menor coste que el utilizar un helicóptero [10]. También, son muy eficaces en tareas de vigilancia pudiendo establecer una patrulla de manera continua al usar varios drones sin dejar puntos muertos en ningún momento.

Como se puede ver, los drones son una herramienta con gran utilidad en un amplio rango de aplicaciones capaz de facilitar y mejorar las operaciones que se llevan a cabo en la industria.

1.1.2 Deep Learning en la industria

Los conceptos del Deep Learning, una de las metodologías de la Inteligencia Artificial, se explicarán con detalle en el siguiente capítulo. Esta técnica ha ganado popularidad en los últimos años debido a los resultados obtenidos y la gran variedad de usos que se le puede

dar. Puede ser implementada en prácticamente cualquier campo de la industria ya que la principal ventaja del Deep Learning es su capacidad de trabajar con una enorme cantidad de datos y parámetros y detectar patrones que el ser humano es incapaz.

Algunos ejemplos donde podemos encontrar su aplicación es en modelos previsivos como el propuesto para el servicio postal de Korea para la previsión de demanda en días festivos especiales como el Año Nuevo Lunar en donde crece bruscamente la demanda [11], gracias a esta técnica son capaces de cumplir con sus servicios como un día normal. Otra forma de aplicar estos modelos son para predecir y prevenir errores como, por ejemplo, en líneas de producción incluso con meses de antelación con una tasa de acierto de aproximadamente 91% [12]

Otra utilidad común para esta técnica es el reconocimiento de voz que se pueden encontrar en diversos aparatos mejorando así su funcionalidad, pero sin lugar a dudas, donde mayor progreso e implementación se está desarrollando en el uso de Deep Learning es en el reconocimiento de objetos. Desde su implementación en automóviles para identificar distintas señales de tráfico, otros vehículos o posibles colisiones y prevenirlas [13] hasta para detectar el agarre óptimo a un objeto incluso si este tiene gran plasticidad, de gran utilidad en robots autónomos de manufacturación [14].

En el siguiente estudio [15] donde se muestra cómo, aplicando un modelo entrenado con Deep Learning, éste es capaz de detectar hasta 9 enfermedades distintas en un tomate simplemente al estudiar una imagen de sus hojas. A pesar de no ser un ejemplo muy industrial, muestra perfectamente el gran potencial que tiene el Deep Learning.

Anteriormente, se han nombrado varios ejemplos de aplicaciones de Deep Learning en la industria en campos muy diferenciados, sin embargo, estos son unas de las incontables aplicaciones que se le puede dar para mejorar, optimizar e innovar tareas, desde las más básicas y simples hasta las de gran complejidad y precisión.

1.1.3 Robots autónomos que utilizan Deep Learning

En este apartado se mostrarán algunos ejemplos de robots autónomos que aplican un modelo entrenado con Deep Learning comenzando por, como se mencionó anteriormente,

aquellos utilizados para vigilancia que son capaces de detectar personas y/o llevar un control de aforo. También, aquellos cuya función es encontrar personas desaparecidas o que estén pidiendo ayuda, detectando y distinguiendo, por ejemplo, si la persona se encuentra con los brazos levantados o tirada en el suelo [16] o que, de manera similar, se pueden utilizar para búsquedas en el mar y otras operaciones de rescate donde detectar y distinguir objetos flotando en el agua, suele ser una tarea complicada [17].

Por otro lado, se pueden encontrar aquellos robots autónomos que utilizan el Deep Learning para planificación de trayectorias. Por ejemplo, un modelo que predice el gasto de batería y, en función de ésta, recalcula la ruta para optimizar los objetivos del vuelo o realizar operaciones para recargarse en vuelos largos [18]. También se pueden encontrar métodos que combinen algoritmos de distancia mínima con modelos que predicen trayectorias humanas reales para elaborar el camino óptimo hasta un destino, como el método desarrollado este mismo año Neural A* Search [19]. Otro ejemplo sería aquellos que utilizan un modelo para crear un mapa de transmisión que puede ser entendido como un mapa de profundidad relativa de las imágenes de la cámara del robot y con éste, determinar la dirección óptima para esquivar los obstáculos que se encuentren, como el usado para esquivar corales, peces y otros objetos en las profundidades del mar [20].

1.2 OBJETIVOS

Una vez realizada la introducción al contexto en el que se desarrolla este proyecto se describen en este apartado los objetivos planteados en este trabajo con el objetivo final de desarrollar un prototipo de un robot aéreo autónomo en entornos dinámicos con posible aplicación en tareas de transporte, mensajería, logística, etc.

En primer lugar, en este proyecto se pretende en construir un modelo entrenado con aprendizaje profundo que sea capaz de distinguir y encuadrar en una imagen distintos posibles obstáculos como, por ejemplo, podrían ser árboles, personas o vehículos entre otros.

El segundo será desarrollar un algoritmo de distancia mínima que, tomando como entradas un gráfico (a modo de representación del entorno), la posición inicial y la posición a la que se quiera llegar en dicho gráfico, sea capaz de establecer la ruta más corta y rápida. El algoritmo

tendría como salida una lista en orden de las siguientes posiciones a alcanzar hasta llegar a la meta deseada evadiendo los obstáculos recogidos en el gráfico. Además, este algoritmo, también será capaz de actualizar la ruta si el gráfico utilizado se modifica en algún momento del proceso.

El tercero será programar el DJI Tello de tal manera que se pueda controlar desde un ordenador, tanto a través de comandos, como con las propias teclas del ordenador. También, se desea obtener una representación visual, como un mapeado, que registre el entorno, los obstáculos, posición del drone y su orientación, además del recorrido de la aeronave durante el tiempo que esté siendo controlado.

El cuarto objetivo consistirá en, dada una trayectoria, el DJI Tello reciba los comandos de vuelo necesarios para poder seguir dicha trayectoria.

Finalmente, se tratará de implementar los cuatros objetivos anteriores en conjunto, de esta manera se conseguirá que el drone sea capaz de registrar obstáculos a través de las imágenes obtenidas por su cámara y el modelo entrenado, además utilizando el algoritmo de distancia mínima, la aeronave será capaz de trazar y seguir de forma autónoma la trayectoria óptima hasta el punto de destino que introduzca el usuario. También, durante este proceso mostrará en pantalla una representación gráfica del entorno para facilitar y aportar información al usuario el registro del entorno y las posiciones que ocupará el drone hasta llegar a la localización final.

El objetivo final, por tanto, es desarrollar una primera aproximación a un vehículo aéreo no tripulado (UAV = Unmanned Aerial Vehicle) que pueda ser utilizado en distintas aplicaciones donde se encuentre un entorno dinámico y se requiera detectar obstáculos y esquivar obstáculos (mensajería, el transporte, operaciones de rescate, etc.).

A esta aeronave se le dará unas coordenadas de destino y se dirigirá a ella sin necesidad de "feedback" con un operador, gracias a la cámara y al modelo entrenado que identificará obstáculos y recalculará la trayectoria para evadir dichos obstáculos de forma segura hasta la posición final. También, mostrará por la pantalla del usuario cualquier información relevante que pueda ayudar a obtener una idea clara del entorno y la situación en la que se encuentra el drone en cada momento.

Del mismo modo, el proyecto contempla incluir otras características que puedan ser de utilidad como poder elegir si controlar el robot aéreo manualmente o de forma autónoma, mensajes de información o capturar imágenes, rutas o mapeados de interés.

1.3 DJI TELLO DRONE

En este proyecto se utilizará el drone DJI Tello de la marca Ryze Tech. Esta aeronave es un pequeño cuadricóptero (4 rotores) de fácil accesibilidad con grandes posibilidades educativas debido a la facilidad de ser programado. Puede ser programado en diferentes lenguajes como Scratch, Python y Swift. En el caso de este proyecto ha utilizado Python.

A pesar de no ser un drone con la última tecnología en sensores, es una alternativa muy recomendada para gran variedad de proyectos por su bajo precio, en comparación con otros, modelos y su versatilidad. Existe un modelo superior de esta aeronave llamado DJI Tello EDU el cual permite programar varios drones en forma de enjambre para que vuelen en conjunto y sintonía, lo cual puede ser muy interesante para realizar tareas en las que un solo drone es incapaz de llevar a cabo y necesite ayuda de otros o para cumplir independientemente varias tareas teniendo en consideración el resto de las tareas que se estén realizando.

El DJI Tello cuenta con un sistema de posicionamiento visual, un controlador de vuelo, un sistema de transmisión de video, un sistema de propulsión, una IMU (Inertial Measurement Unit) y una batería de vuelo. También, dispone de un indicador de estado de la aeronave, que consiste en una luz que en función del color o patrón de parpadeo comunica el estado que se encuentra la aeronave (baja batería, cargando, encendiendo y realizando pruebas de autodiagnóstico, etc.)

A continuación, se muestran unas imágenes de la nave y su diagrama:

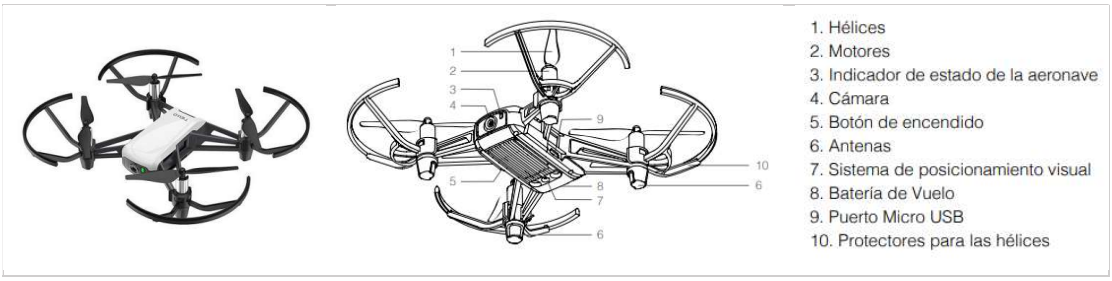


Figura 1. Diagrama del DJI Tello drone [21].

Una de las grandes ventajas de este drone es que puede ser fácilmente controlado manualmente con una aplicación para el móvil o con un control remoto compatible. Asimismo, dispone de distintos modos de vuelo inteligente que permiten realizar maniobras automáticamente como volteretas en 8 direcciones diferentes del espacio, "rebotar" mientras vuela o poder ser lanzado y que se mantenga estático flotando en el aire.

El sistema de posicionamiento visual ayuda a la aeronave a mantener su posición actual en el aire. Con la ayuda de este sistema, el DJI Tello puede volar en modo estacionario con mayor precisión, así como ser pilotado en interiores o exteriores en condiciones sin viento, ya que no tiene la potencia suficiente para contrarrestar el viento en la gran mayoría de los casos. Los componentes principales del sistema de posicionamiento visual son una cámara y un módulo infrarrojo 3D situado en la parte inferior del drone [21].

Algunas de las limitaciones de este drone son la carencia de sensores para geolocalizarse u otros sensores de proximidad con los que se podría estimar la posición de distintos obstáculos en el espacio o de la propia aeronave en sí. Algunos sensores que podrían ser de utilidad en este proyecto son GPS externos para conocer la posición de la aeronave en el espacio o sensores de proximidad tipo radar o tipo láser como los Lidar, popularmente utilizados en la industria para estimar distancias relativas entre puntos de interés en el entorno. Otra forma de obtener estimaciones de distancias sería a través de una cámara Kinect, que es un tipo de sensor compuesto por una cámara RGB-D, una cámara de infrarrojos y sensores de profundidad. Desgraciadamente, el Tello no dispone de ella. Asimismo, a pesar de tener un procesador incorporado, éste no es lo suficientemente potente para llevar a cabo las operaciones que se requieren en este proyecto con suficiente rapidez. Por ello, se utilizará una unidad computacional externa con mayor potencia como unidad principal de procesamiento.

Otras limitaciones de esta aeronave son la poca duración de su batería y su sencillez, ya que al ser un drone de bajo coste y complejidad, no está diseñado para ser volado por un extenso período de tiempo o en exteriores donde las condiciones meteorológicas como el viento o la lluvia podrían impedir un correcto manejo del mismo. La temperatura del ambiente en sí puede ser un factor limitante ya que puede sobrecalentar la batería de la aeronave e inhabilitarla hasta que se vuelva a enfriar y pueda volar de nuevo.

En la siguiente tabla se muestran algunas de las características principales de este robot aéreo:

Peso y dimensiones			
Ancho	92 mm	Profundidad	98 mm
Anchura	98 mm	Altura	41 mm
Peso (con batería)	80 g	Diámetro rotor	7,62 mm
Características de administración			
Despegue automático	SI	Aterrizaje automático	SI
Cámara			
Megapíxeles	5 MP	Máxima resolución de video	HD: 1280 x 720 30p
Máximo tamaño imagen	2592 x 1936	Ángulo de campo de visión (FOV)	82,6º
Formato de video soportado	720p	Formato de video compatible	MP4
Batería			
Tipo	LiPo	Capacidad	1100 mAh
Voltaje	3,8 V	Energía	4,18 Wh
Potencia de carga máx.	10 W	Tiempo de funcionamiento máx.	13 min
Otras características			
Número de rotores	4	Velocidad máxima	8 m/s
Distancia máxima	100 m	WiFi	SI
Microprocesador	SI	Frecuencia de banda	2,4 a 2,8 GHz
GPS	NO	Sensores de distancia	NO

Tabla 1. Caracerísticas principales del DJI Tello Drone [21].

2.- DEEP LEARNING

En este capítulo se introducirán los conceptos básicos de la técnica de Machine Learning llamada Deep Learning. Esta técnica será utilizada en este proyecto para crear un modelo capaz de distinguir y clasificar posibles obstáculos en las imágenes recibidas desde la cámara del DJI Tello en tiempo real.

El Deep Learning (Aprendizaje profundo) se encuentra dentro de las metodologías de Inteligencia Artificial (IA). La Inteligencia Artificial es un conjunto de técnicas y algoritmos que permite que una máquina imite el comportamiento humano.

El Deep Learning es una metodología que tiene el objetivo de lograr que la IA entrene sus algoritmos con una base de datos para ser capaz de desarrollar una labor en el futuro. Concretamente, el Deep Learning está basado en la estructura del cerebro humano. En términos de Deep Learning, esta estructura es conocida como Red Neuronal Artificial (ANN: Artificial Neural Network).

La principal diferencia entre Deep Learning y Machine Learning es que, en el caso de Machine Learning, cuando se entrena al modelo, hay que especificar los parámetros y características principales que diferencian los componentes de la base de datos. En cambio, en el Deep Learning, la red neuronal es la encargada de esta tarea. Esto conlleva una gran ventaja y una gran desventaja. La ventaja es que, utilizando Deep Learning, la IA puede ser capaz de captar patrones o detectar parámetros que el ser humano no puede, desarrollando una actuación más precisa y rápida. La desventaja consiste en que, para lograr esto, necesita una base de datos mayor para entrenar y todo lo que conlleva esta problemática (tiempo de entreno, potencia computacional…).

2.1 FUNCIONAMIENTO DE LAS REDES NEURONALES

Como se hs indicado en el apartado anterior, la red neuronal es la estructura sobre la que trabajan los algoritmos del Deep Learning. En este apartado, se explica cómo funcionan las redes neuronales.

Como su propio nombre indica, el elemento principal de las redes neuronales es la neurona. Una neurona es una simple unidad funcional que tiene una entrada y una salida. Es decir, en la propia neurona se lleva a cabo una operación matemática utilizando una entrada (X) que da como resultado una salida (Y).

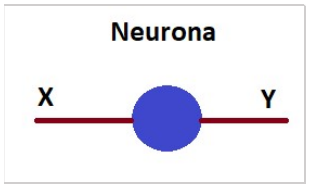


Figura 2. Representación de una neurona en una red neuronal.

Esta operación puede ser tan simple como añadir una constante o multiplicar por una constante (peso o coste entre neuronas), aunque suelen ser mucho más sofisticadas.

Una de las más comunes es una función sigmoide o también llamada, tangente hiperbólica, la cual para valores de entrada pequeños da como salida 0 y para valores de entrada grandes, sin importar cuánto de grandes sean, se obtiene como salida 1. Para valores intermedios, consiste en una función de activación que va de 0 a 1 de manera fluida. Muchas de las neuronas del ser humano funcionan de esta manera. Otra función de activación bastante utilizada es la función rampa unitaria (ReLU: Rectified linear unit), la cual la salida vale 0 hasta cierto valor de la entrada y a partir de ese valor la salida crece linealmente con la entrada. Estas son las más comunes y polivalentes, pero existen muchas otras que pueden ser mucho más apropiadas para casos más concretos [22].

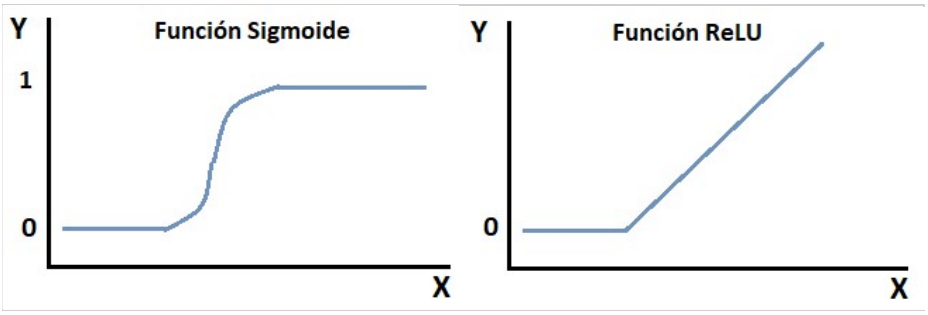


Figura 3. Ejemplos de funciones de activación.

Una red neuronal artificial (ANN) no es más que una asociación de neuronas en serie o paralelo o ambas para crear operaciones matemáticas más complicadas o laboriosas, generando capas de neuronas intermedias entre las capas de entrada y de salidas. Las neuronas que forman la red pueden tener distintas funciones de activación. Si se siguen añadiendo un mayor número de éstas capas intermedias, las cuales cada capa desarrolla un procesamiento consecuente, se obtiene una red neuronal profunda (DNN: Deep Neural Network) [22].

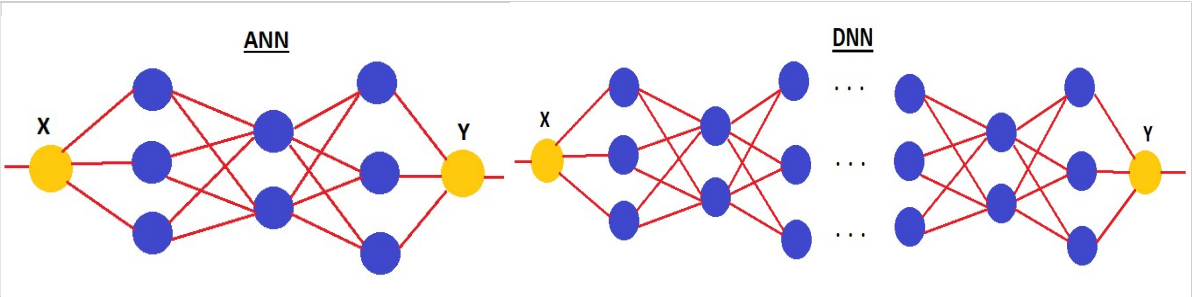


Figura 4. Representación de las conexiones neuronales en una red neuronal.

Al igual que las funciones de activación también existen un gran número de estructuras diferentes de DNN dependiendo de cómo están conectadas las distintas capas y la función que desarrolla cada una de ellas.

En este proyecto nos vamos a centrar en la estructura denominada red neuronal convolucional (CNN: Convolutional Neural Networks) que es la más utilizada en reconocimiento de imágenes.

A continuación, se muestran algunos de los distintos tipos de estructuras que existen:

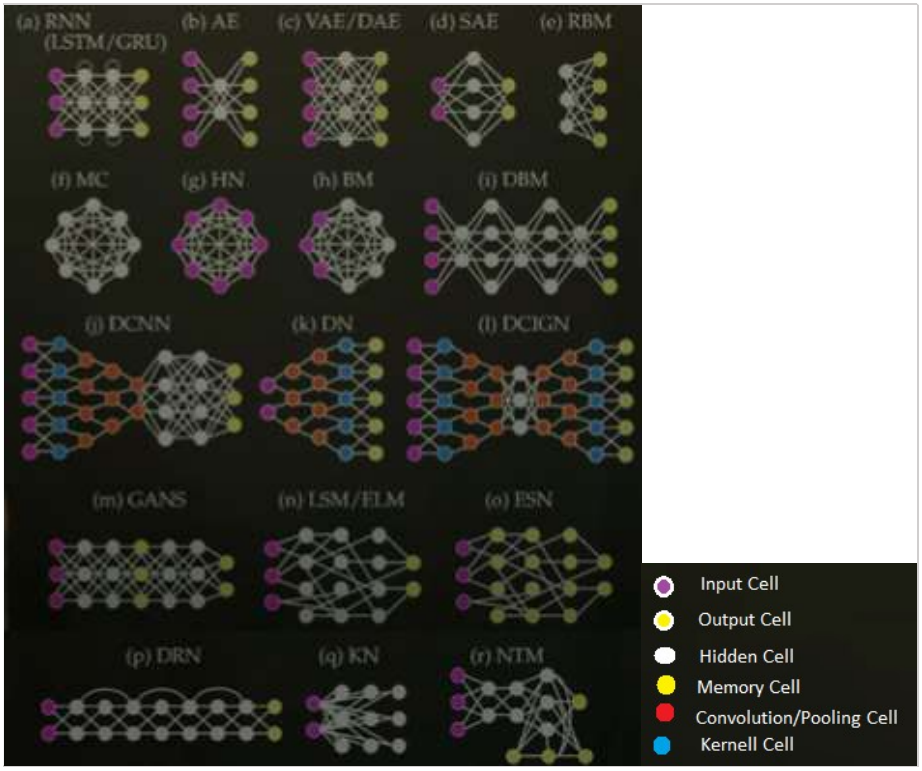


Figura 5. Tipos de estructuras de una ANN [22].

2.2 RED NEURONAL CONVOLUCIONAL (CNN)

El beneficio principal de utilizar una CNN es la considerable reducción de parámetros de la red permitiendo elaborar modelos más grandes y complejos, o con la misma aplicación que una red neuronal clásica, pero con una red más pequeña. Esta reducción se debe a la consideración de que las características a reconocer no dependen de su posición en el espacio. Es decir, si se trata de detectar ciertos objetos en una imagen como ocurre en este proyecto, no depende de si el objeto se encuentra arriba en una imagen o a la derecha o en cualquier punto de la imagen.

Cualquier CNN utilizada para el reconocimiento de objetos está compuesta por las siguientes capas (layer):

- Image input layer
- Convolutional layer
- Activation layer
- Pooling layer
- Fully connected layer
- Output layer

En algunos casos, se tendrá más de una capa de cada tipo a lo largo del modelo, tratándose así de modelos más profundos y complejos, lo cual no siempre es necesario u óptimo [23].

2.2.1 Image input layer

Esta capa estará formada por un número de neuronas igual al número de pixeles que tiene la imagen. También, habrá que tener en cuenta que si se trata de una imagen a color existirán tres matrices (tres canales) por pixel, una por cada intensidad del color del pixel, siendo estos el rojo, verde y azul. Los valores de cada matriz tendrán un valor entre 0 y 255 (2^8), sin embargo, normalmente antes de introducir la imagen al modelo se suele preprocesar de tal manera que se redimensionen el ancho y alto de la imagen y, además, se escalen los valores de cada pixel para que se encuentren entre 0 y 1 para facilitar posteriormente los cálculos en el resto de los procesos. Por ejemplo, si se trata de una imagen de entrada a color de 24

pixeles de ancho y 36 de alto, habría una totalidad de 24x36x3=2592 neuronas en la capa de entrada.

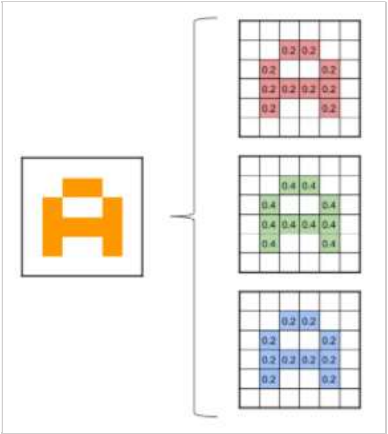


Figura 6. Los tres canales en una imagen a color [24].

2.2.2 Convolutional layer

Como el propio nombre indica, estas capas son las más distintivas y características de este tipo de estructuras. En ellas se realizarán las operaciones de convolución en dos o tres dimensiones. Estas consisten en tomar pequeñas matrices dentro de la matriz de entrada de la imagen e ir operando matemáticamente, generalmente con el producto escalar, con otra pequeña matriz llamada filtro kernel. Este filtro recorre todas las neuronas de entrada, de izquierda a derecha y de arriba a abajo, y genera una nueva matriz de salida que será la siguiente capa de neuronas ocultas. Realmente, no se tiene un solo filtro sino muchos de ellos por cada capa de convolución. Igualmente, se pueden tener distinto tipo de filtros a lo largo del modelo, siendo, los de las primeras capas de convolución, aquellos preparados para detectar características más generales como, por ejemplo, bordes y esquinas. Las siguientes capas tendrán filtros que detectarán características más abstractas. Si la imagen fuese a color, el filtro tendría profundidad 3 en vez de 1, por lo que habría tres matrices de salida que tras una operación matricial formarían una sola de salida, como si fuera solo un canal.

Siguiendo el ejemplo anterior, si tenemos 16 filtros, se tendrían 41472 neuronas en esta primera capa oculta. Se podría pensar que son una cantidad considerable de neuronas, pero hay que tener en cuenta que unas dimensiones de una imagen normalmente pasan los 640 pixeles, así la cantidad de neuronas aumenta considerablemente.

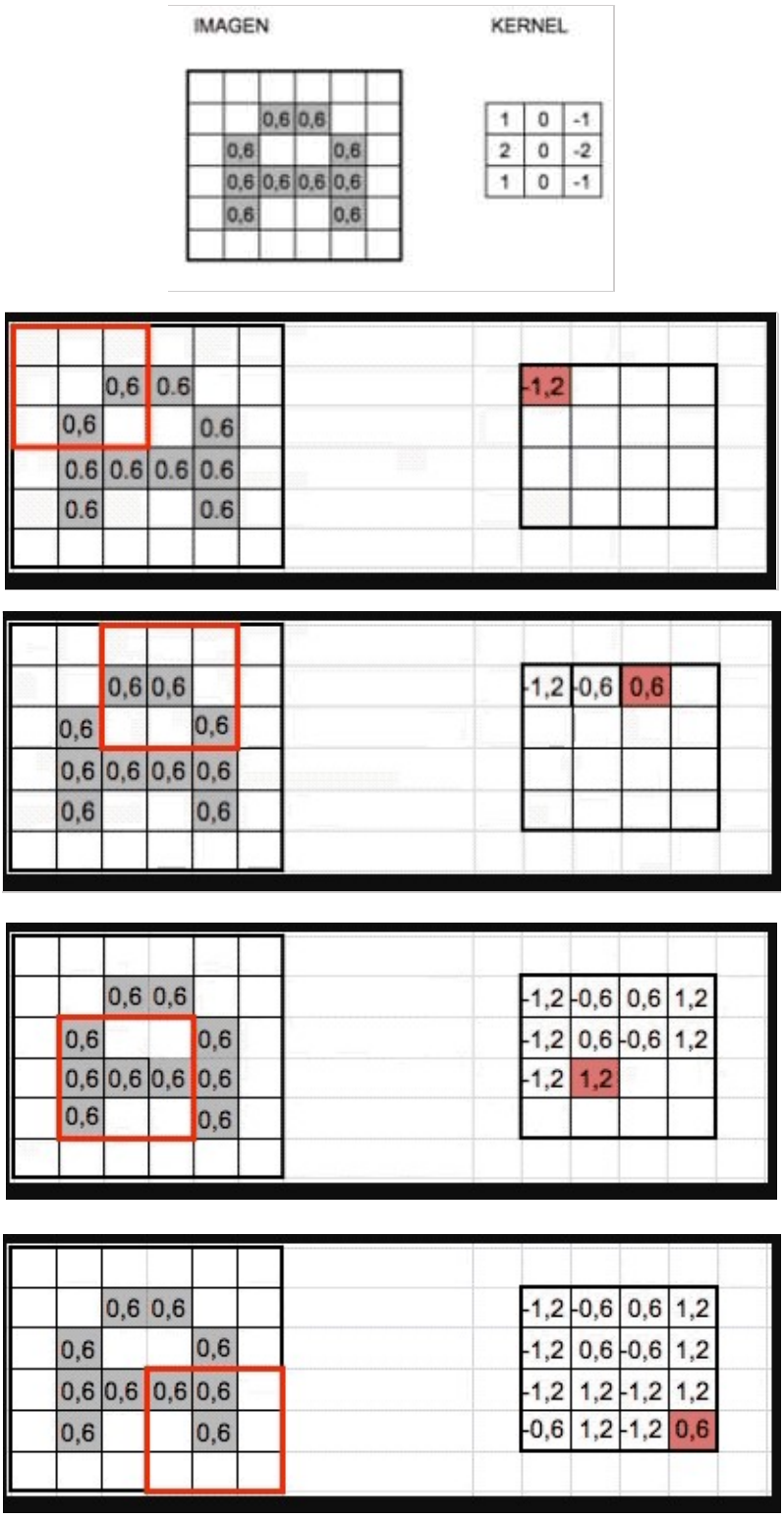


Figura 7. Funcionamiento de la convolución [24].

2.2.3 Activation layer

Esta capa consiste en aplicar las funciones de activación explicadas anteriormente para cada neurona. Principalmente para este tipo de redes neuronales se utilizan la función sigmoide y la ReLU.

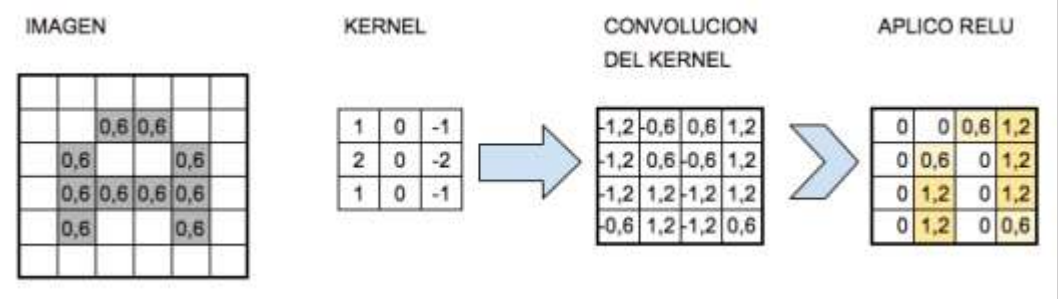


Figura 8. Ejemplo de convolución y activación [24].

2.2.4 Pooling layer

Como se ha mencionado, la cantidad de neuronas es un parámetro a tener en cuenta. En este tipo de capas lo que se va a realizar es una reducción de las dimensiones de la red. Para ello, se reducirá el tamaño de las imágenes filtradas, pero manteniendo las características más importantes que detectó cada filtro. El "pooling" o "subsampling" más usado es "Max-Pooling". Dependiendo de sus dimensiones (2x2, 3x3…) se estudiará un subgrupo de esas dimensiones y se preservará el valor mayor que se encuentre en el subgrupo, por ello, el nombre de "Max". Así, si se utiliza un filtro 2x2 la imagen resultante se reduce a la mitad. En el ejemplo anterior, quedarán 16 imágenes de 12x18 habiendo reducido de 41472 neuronas a 10368, un número considerablemente menor pero aun así manteniendo la información más importante para detectar los rasgos más característicos de la imagen.

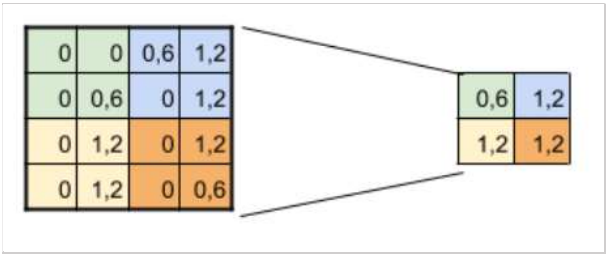


Figura 9. Pooling 2x2 [24].

2.2.5 Fully connected layer

Después de realizar varias etapas de convolución, activación y “pooling”, las dimensiones de la salida se habrán reducido, debido al “pooling”, mientras su profundidad habrá aumentado, debido al número de filtros. Esta salida será dimensionada a un vector el cual será la entrada para una ANN clásica.

2.2.6 Output layer

Finalmente, para problemas de clasificación, la capa de salida tendrá la misma dimensión que el número de clases que la CNN es capaz de identificar. Entonces, a partir de las funciones de activación de la última capa oculta, calculará la probabilidad para cada clase posible.

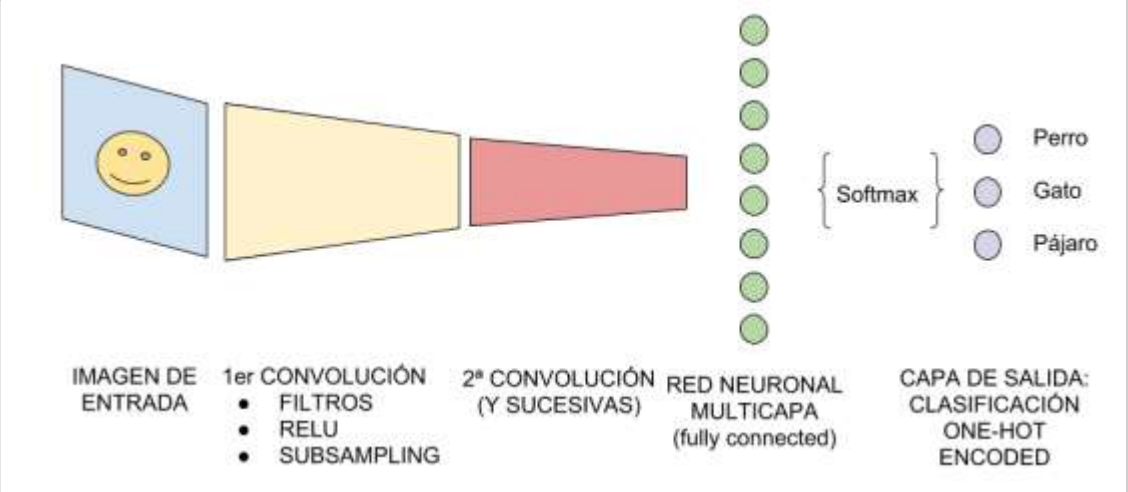


Figura 10. Ejemplo de arquitectura de una CNN [24].

2.3 LIMITACIONES DEL DEEP LEARNING Y LAS REDES NEURONALES

El Deep Learning, al ser una técnica relativamente nueva y poderosa, si no se contiene un conocimiento profundo sobre qué está pasando dentro del modelo puede llegar a ser una herramienta realmente frustrante y laboriosa con la que trabajar. En este apartado se van a tratar diferentes limitaciones o problemas comunes al trabajar con redes neuronales profundas, que requieren de un mayor desarrollo en los próximos años para mejorar y optimizar su uso.

El primer tema que se va a tratar es el sobreentrenamiento o “Overfitting”. Este problema consiste en entrenar al modelo de tal forma que es muy preciso y exacto con la base de datos utilizada para ser entrenado pero una vez se desea implementar en un entorno nuevo, su veracidad cae drásticamente. En ciertas situaciones, el “overfitting” no se considera un problema ya que los datos utilizados para entrenarlo son una representación con un grado de similitud casi exacto con los datos que se va a encontrar en su aplicación real. El Overfitting suele ocurrir cuando se tiene una base de datos tan amplia y rica en características de interés con un gran número de grados de libertad, lo cual es interesante para entrenar un modelo, pero puede no generalizar correctamente en el futuro. Por esto, es tan necesario e importante tener una base de datos con suficientes muestras y variedades, y realizar una validación cruzada de la misma. Esta validación cruzada consiste en una técnica que realiza una evaluación de los resultados de un análisis estadístico y garantiza que el subgrupo con el que se ha entrenado el modelo y el subgrupo complementario con el que se ha probado el modelo una vez entrenado son independientes [22]. Da una indicación de cómo de general o específico es el modelo, es decir, si se puede extrapolar a otros datos o si simplemente funciona satisfactoriamente con los datos con los que se ha entrenado, respectivamente.

Otra problemática con la que las redes neuronales, así como el Machine Learning en general, tienden a encontrarse es la capacidad para un modelo de ser generalizable, particularmente cuando se trata sobre leyes de física. Si a un modelo se le enseña trayectorias parabólicas de una bala de cañón, será capaz de predecir otras trayectorias parabólicas en el futuro, en cambio, si a un modelo se le enseña la Segunda Ley de Newton ($F = m \cdot a$), será difícil aplicarlo posteriormente a construir un cohete, por ejemplo. Hablando más en general, las redes neuronales son perfectas cuando se les enseña con lo que van a trabajar en el futuro, pero es bastante complicado extraer ese conocimiento generalizado para aplicarlo a algo diferente pero basado en la misma idea principal.

Relacionado con el punto anterior, se encuentra la interpretación y explicación dentro de un modelo. En ciertas ocasiones, estas redes neuronales profundas estarán elaboradas por una innumerable cantidad de parámetros o grados de libertad de tal manera que son difíciles de interpretar por lo que no se llega a entender lo que está haciendo realmente. Algunas estructuras son más fáciles de interpretar que otras, así como, dependiendo del

desarrollador, se tiene mayor facilidad para interpretar un tipo de red neuronal u otro. Esto encadena con la dificultad de explicar a otros desarrollares cómo los algoritmos están tomando decisiones o de entender en sí las decisiones que está tomando el modelo.

Por último, relacionado con aplicaciones más ingenieriles, un área que necesita un mayor desarrollo e investigación es conseguir incorporar leyes físicas que conocemos en los modelos. Como por ejemplo que la energía se conserva o la masa se conserva, la influencia de la simetría, etc. Esto podría llevar a modelos muchos más interesantes que usen esta información para ser más precisos o más rápidos o incluso aportar datos que descubran nuevas leyes físicas a los investigadores y desarrolladores [25].

Todas estas limitaciones son aplicables a cualquier método de Machine Learning pero especialmente para aquellos que usan redes neuronales ya que ganan gran importancia a la hora de desarrollar la estructura interna de las mismas.

2.4 TRANSFER LEARNING

En este proyecto se va a utilizar un método de Machine Learning llamado Transfer Learning, aunque este método no sea exclusivo de Deep Learning, tiene gran popularidad en esta rama debido a la gran cantidad de recursos necesarios para desarrollar un modelo entrenado con Deep Learning. Este método consiste en reutilizar un modelo ya desarrollado para una determinada tarea en otra tarea relacionada. Las principales ventajas de este método son una reducción de tiempo y una mejora de rendimiento.

En Transfer Learning, primero, se utilizará una red neuronal basada en una determinada base de datos para una tarea específica, después, se transferirá lo aprendido sobre las características más significativas a una segunda red con otra base de datos y otra misión. La forma de transferir lo aprendido consiste en transferir las relaciones entre neuronas que principalmente son los costes entre ellas que se han ido reajustando durante el proceso de entrenamiento. Para que este proceso funcione, las características descubiertas por la red base tienen que ser generales para que se puedan aplicar tanto a la red base como a la nueva. Este tipo de Transfer Learning utilizado en Deep Learning se conoce como transferencia inductiva [26].

Este tipo de técnica es comúnmente usada en modelos predictivos que utilizan una imagen como entrada, ya sea una fotografía o un video. Normalmente, se utiliza un modelo pre-entrenado (así es como se llama a los modelos base) con una gran y compleja misión de clasificación de imágenes como, por ejemplo, la competición de ImageNet, donde se evaluaban algoritmos capaces de clasificar en más de 1000 clases distintas. Los modelos prometedores de este tipo de competiciones se suelen hacer públicos una vez se haya desarrollado la versión final con una licencia para poder ser reusados. Estos modelos pueden tomar días incluso semanas en ser entrenados [27]. Otros ejemplos de este tipo de modelos son Oxford VGG, Google Inception y Microsoft ResNet, existiendo varias versiones de cada uno. También, se pueden encontrar modelos combinando varios como Inception-ResNet.

Muchas instituciones de investigación tienen disponibles modelos para el público. El modelo pre-entrenado puede ser utilizado como punto de partida del nuevo modelo, puede usarse la totalidad del modelo o simplemente alguna parte, además, si es necesario, puede ser modificado para cumplir de una manera más precisa con las especificaciones del nuevo modelo. Cuando se utilizan redes neuronales convolucionales, las primeras capas suelen ser las que captan características más genéricas y las más profundas aquellas enfocadas en características más específicas, por ello, en ocasiones es de interés simplemente utilizar las primeras capas del modelo pre-entrenado [28].

2.5 TENSORFLOW Y GOOGLE COLAB

En este proyecto se va utilizar TensorFlow para desarrollar y entrenar el modelo que se utilizará posteriormente para detectar obstáculos. Además, se utilizará Google Colab como entorno para entrenar el modelo de detección de objetos.

TensorFlow, desarrollada por Google Brain Team y abierta al público por primera vez en 2015, es una plataforma de código abierto de extremo a extremo para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que permite que los investigadores innoven con el aprendizaje automático y los desarrolladores creen e implementen aplicaciones con tecnología de aprendizaje automático fácilmente. TensorFlow ofrece una colección de flujos de trabajo para desarrollar y entrenar modelos mediante Python o JavaScript, y poder implementarlos con facilidad en la nube, de

forma local, en el navegador o en algún dispositivo móvil [29]. La página web oficial de esta plataforma cuenta con foros, blogs, tutoriales y otros recursos para ayudar a la comunidad de manera colaborativa y así poder sacar el máximo partido a TensorFlow. Además, cuenta con una interfaz de programación de aplicaciones (API) llamada Keras la cual es de alto nivel y permite una sencilla iniciación y desarrollo en TensorFlow y el aprendizaje automático.

Se ha elegido TensorFlow ya que es considerada la plataforma de Deep Learning más importante del mundo por su flexibilidad y gran comunidad de desarrolladores que la posicionan como la herramienta líder en el sector del Aprendizaje Profundo. Esto se puede ver reflejado en el gran número de grandes empresas multinacionales que lo utilizan para alguno de sus procesos. Entre ellas se encuentran *Coca-Cola*, en el reconocimiento de comprobantes de compra en dispositivos móviles, *GE Healthcare*, en una red neuronal que identifica anatomías en resonancias magnéticas del cerebro, o *Paypal*, en la detección de fraudes, entre muchas otras firmas [30].

Google Colaboratory, también llamado Google Colab, es un producto de Google Research. Esta herramienta permite ejecutar y programar en Python en el navegador. Es ideal para aplicarlo en proyectos de aprendizaje automático, análisis de datos y educación. Más técnicamente, Colab es un servicio de notebook alojado de Jupyter cuyas principales ventajas son el acceso gratuito a recursos computacionales, incluidas GPUs (unidadades de procesamiento de gráficos), su facilidad para compartir contenido fácilmente y que no requiere configuración previa [31]. También, cuenta con versiones de pago Colab Pro y Colab Pro+ para aquellos casos en los que se necesite un mayor margen de libertad que la versión gratuita no permite.

Google Colaboratory permite almacenar sus notebooks en Google Drive o GitHub, también se pueden compartir de igual manera que las Hojas de Cálculo o los Documentos de Google. El código se ejecuta en una máquina virtual exclusiva para la cuenta del usuario y se borran cuando están inactivas por un tiempo prolongado con una vida útil máxima de unas 12 horas y una memoria RAM máxima de en torno a 16 GB, más que suficiente para la gran mayoría de casos. Sin embargo, estas limitaciones varían con el tiempo dependiendo de la demanda y otros factores con el objetivo de poder seguir brindando recursos gratuitos. Colab no publica estos límites ya que pueden cambiar con rapidez. Las GPU disponibles también varían

con el tiempo por las mismas razones, pero generalmente incluyen K80, T4, P4 y P100 de NVIDIA, las cuales son reconocidas por su efectividad en la comunidad. No obstante, los usuarios no pueden elegir de forma predeterminada a qué tipo de GPU se van a conectar de forma gratuita, aquellos que busquen un acceso más confiable a las más veloces pueden optar por la opción de pago Colab Pro [31].

Por todas estas ventajas, se va a utilizar esta herramienta como entorno para entrenar el modelo del proyecto. En el apartado 3.2 de este documento, se explica con mayor detalle los pasos a seguir a modo de tutorial de igual manera que se han seguido para realizar este proyecto.

3.- EXPLICACIÓN DETALLADA DEL MODELO PARA LA DETECCIÓN DE OBJETOS

En este apartado se desarrolla cómo ha sido entrenado el modelo que se utilizará para la detección y clasificación de posibles obstáculos en las imágenes recibidas a través de la cámara del DJI Tello en tiempo real. Además, se explicarán sus características y cómo ha sido el proceso de entrenamiento, paso a paso.

Para este modelo se utilizarán algunos de los recursos que proporciona TensorFlow en su repositorio GitHub. Se utilizarán varios ficheros de Python de código abierto.

Para entrenar el modelo, se utilizará una base de datos de imágenes creada específicamente para este proyecto. En cada imagen habrá etiquetado uno o varios obstáculos con la clase de obstáculo que les corresponde. Esto quiere decir que, previamente, las imágenes han tenido que ser etiquetadas una a una. Lo que significa que el modelo será de tipo supervisado ya que han introducido los datos de tal forma que el algoritmo genera datos de salida esperados (las diferentes clases de clasificación), se guía al algoritmo para diferenciar las salidas y se comparan con las entradas para comprobar si ha acertado o no.

Las imágenes se dividirán en dos grupos, uno para entrenar el modelo en sí y otro para verificar cómo de preciso es. En cada iteración del entrenamiento, el modelo estudiará el primer grupo de imágenes para buscar patrones, características destacables, etc. (representado por los pesos y cálculos computacionales entre las conexiones de las neuronas) que ayuden a identificar y distinguir las distintas clases entre sí, después se probará cómo de acertado ha sido el estudio con el segundo grupo de imágenes. La precisión se estimará con una medida de error calculada con una función de pérdida, por tanto, cuanto menor sea este valor con mayor acierto estará prediciendo el modelo.

Existen varias funciones de pérdida siendo las más comúnmente usadas máxima probabilidad y entropía cruzada. Para problemas de clasificación con más de dos clases, se utiliza entropía cruzada, también conocida como pérdida logarítmica H(P,Q), por lo tanto, ésta es la que se usará para este proyecto. Básicamente, esta función compara el valor probabilístico real (P(x)) de un objeto clasificado con la probabilidad que calcula el modelo para ese objeto (Q(x)) y penaliza en función de la magnitud de la diferencia entre ellas. Esto se realizará con cada

objeto clasificado y finalmente se calculará la media de estos valores que será el valor de pérdida con el que se evalúa el rendimiento del modelo. Como la penalización es logarítmica, para diferencias pequeñas el valor de pérdida será pequeño, pero para grandes diferencias este valor será mucho mayor [32]. Al haberse etiquetado manualmente las imágenes, el valor P(x) corresponderá solamente con 0 o con 1.

$$H(P,Q) = P(x) * \log(Q(x))$$

En la siguiente imagen se muestra una guía como referencia para evaluar cómo de buenas son las predicciones en función del valor de pérdida calculado con la función de entropía cruzada (Cross-entropy).

- **Cross-Entropy = 0.00**: Perfect probabilities.
- **Cross-Entropy < 0.02**: Great probabilities.
- **Cross-Entropy < 0.05**: On the right track.
- **Cross-Entropy < 0.20**: Fine.
- **Cross-Entropy > 0.30**: Not great.
- **Cross-Entropy > 1.00**: Terrible.
- **Cross-Entropy > 2.00** Something is broken.

Figura 11. Guía para interpretar el valor de pérdida calculado con la función de entropía cruzada [32].

En cada iteración, los pesos y cálculos computacionales se irán reajustando partiendo de los pesos y cálculos computacionales entre las neuronas de la anterior iteración con el objetivo de mejorar el rendimiento del modelo. Esto se verá reflejado en el decrecimiento del valor de pérdida.

3.1 CARACTERÍSTICAS

Las características principales del modelo utilizado en este proyecto y su entrenamiento se explican en este apartado.

Primero, se ha entrenado para que sea capaz de distinguir 8 clases distintas:

1. *"Poste"*: Englobando desde farolas, estructura de los semáforos, y otros tipos de postes.
2. *"Valla"*: Incluyendo cualquier tipo de valla de madera, verja de alambre, etc.

3. *"Cartel"*: Considerando como tal cualquier señal de tráfico o cartel publicitario entre otros.
4. *"Arbol"*: Englobando tanto árboles como setos.
5. *"Vehiculo"*: Coches, motos, furgonetas, autobuses…
6. *"Semaforo"*: Distintos tipos de semáforos centrándose en la parte de elementos luminosos.
7. "*Persona*"
8. *"Muro"*: Refiriéndose así a muros de piedras, paredes, ventanas, etc.

Estas clasificaciones se han elegido debido a que son los obstáculos más comunes y globales que se podría encontrar el DJI Tello al ser volado en el exterior.

La base de datos con la que se ha entrenado se puede encontrar en el repositorio GitHub del proyecto [1] en la carpeta "Preparation/images", cuenta con un total de 115 imágenes enumeradas del 1 a 115 tomadas personalmente y de internet sin copyright. En cada foto habrá al menos un obstáculo clasificado y algunas podrán contener hasta un máximo de 8 obstáculos, con un total entre todas las imágenes de 387 obstáculos. Entre estas 115 fotos, el 80% (92 imágenes) han sido utilizadas para entrenar el modelo y el 20% (23 imágenes) se han utilizado para verificar y calcular el correcto funcionamiento del modelo durante el proceso de entrenamiento. Las imágenes usadas para verificar han sido elegidas aleatoriamente, en la siguiente imagen se muestran las que han sido seleccionadas:

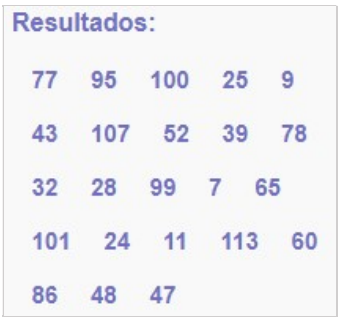


Figura 12. Imágenes seleccionadas para verificar el modelo.

En la siguiente tabla, se muestra cuántos obstáculos hay por clasificación y su distribución entre las imágenes utilizadas para entrenar y aquellas utilizadas para verificar el modelo.

En la tabla del Anexo II. Clasificación imágenes utilizadas como base de datos. se encuentran los obstáculos desglosados de cada imagen.

Distribución de obstáculos									
	Persona	Muro	Vehiculo	Arbol	Poste	Cartel	Valla	Semaforo	Total
Total	29	36	51	58	93	53	40	27	387
Train	18	29	37	45	83	45	32	24	313
Test	11	7	14	13	10	8	8	3	74
%Train	62,1	80,6	72,6	77,6	89,2	84,9	80,0	88,9	80,9
%Test	37,9	19,4	27,4	22,4	10,8	15,1	20,0	11,1	19,1

Figura 13. Distribución de obstáculos.

Como se puede ver, la distribución entre las categorías no es exactamente 80/20 aunque en cuanto al número total de obstáculos sí lo sea prácticamente.

Como se explicó en el apartado 2.4 se va a utilizar un modelo pre-entrenado. TensorFlow dispone de varios de estos modelos, abiertos al público con distintas características en función de la tarea que se desee desarrollar, como modelos aplicados específicamente en dispositivos móviles. Para este proyecto se va a utilizar un modelo pre-entrenado con la base de datos COCO (Common Objects in Context), la cual contiene más de 200.000 imágenes etiquetadas en 91 categorías diferentes [33]. Se ha elegido este tipo de modelo porque entre estas categorías se encuentran personas, distintos tipos de vehículos, semáforos, señales de STOP y otros elementos que se pueden encontrar comúnmente en el exterior. Es decir, el contexto es muy parecido. Coinciden varias de las clases que se desean reconocer en este proyecto y las imágenes de esta base de datos son similares a las utilizadas en este proyecto.

La información de los modelos pre-entrenados que se utilizará para entrenar el nuevo modelo se archivan en archivos CONFIG. Existen modelos con dos tipos de salida, recuadros y máscaras, estas últimas envuelven la forma del objeto a identificar por lo que se han descartado ya que requieren de un considerable mayor poder computacional. Dentro de los archivos CONFIG que proporciona TensorFlow se ha elegido "faster_rcnn_inception_v2_coco.config" debido a la comparación de su velocidad de

procesamiento y precisión medido por el parámetro COCO mAP, este valor indicará mayor precisión cuanto mayor sea.

Modelos entrenados con COCO		
Nombre	Velocidad (ms)	COCO mAP
ssd_resnet_50_fpn_coco	76	35
ssd_inception_v2_coco	42	24
faster_rcnn_inception_v2_coco	58	28
faster_rcnn_resnet50_coco	89	30
rfcn_resnet101_coco	92	30
faster_rcnn_resnet101_coco	106	32
faster_rcnn_inception_resnet_v2_atrous_coco	620	37
faster_rcnn_nas	1833	43

Figura 14. Tabla comparativa de modelos pre-entrenados con COCO [34].

Este modelo, como el propio nombre indica, tendrá una estructura del tipo Faster R-CNN (Faster Region Based-Convolutional Neural Network) la cual es un tipo de red neuronal convolucional, como las mencionadas en 2.2, más concretamente la estructura específica será Inception v2, también mencionada en el apartado 2.4. Faster R-CNN es una versión mejorada de Fast R-CNN la cual a su vez es una versión mejorada de R-CNN. Entendiendo versión mejorada aquella que cuenta con una mayor precisión y menor necesidad computacional.

Las principales diferencias entre una Faster R-CNN y una red neuronal convolucional clásica son [35], [36]:

1. El modelo R-CNN está divido en tres etapas: Proposición de regiones, CNN y clasificador.
2. La proposición de regiones (de ahí proviene “Region Based”) significa que de una imagen se generan en torno a 2000 regiones de esta imagen de diferente tamaño y ratios y se estudiarán independientemente en una CNN hasta la última capa que se clasificará como fondo de imagen o una de las clases a identificar.

Figura 15. Ejemplo de regiones propuestas [37].

3. En Fast RCNN, se añade una capa llamada ROI Pooling que extrae vectores de características de igual longitud. Esto hace que se hagan cálculos con todas las regiones propuestas en vez de independientemente.
4. En Fast R-CNN, la red se modifica para solo tener una etapa.

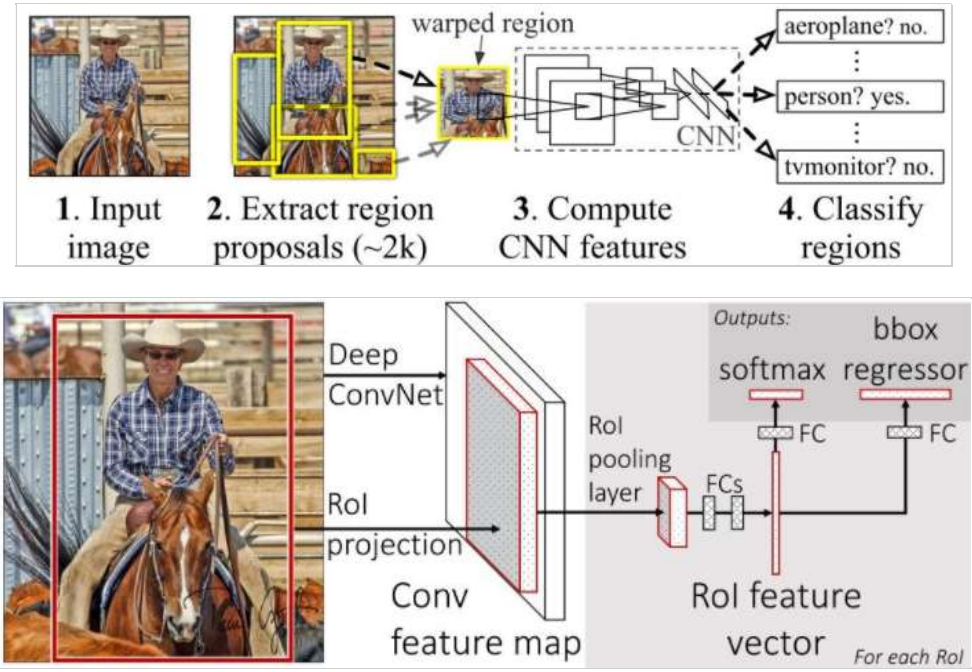


Figura 16. Comparación de las estructuras de R-CNN y Fast R-CNN [35].

5. En Faster R-CNN, se añade una RPN (Region Proposal Network) la cual ayuda a la red neuronal (Fast R-CNN) a elegir mejores propuestas de regiones.

6. En Faster R-CNN, en vez de usar pirámides de imágenes (varios ejemplares de la imagen, pero a distinta escala) o pirámides de filtros (muchos filtros con distintos tamaños), se introduce el concepto de "anchor boxes". Estas "cajas" se usarán como referencia para una determinada escala y ratios, por tanto, cada región será asignada a una "anchor box" y así poder detectar objetos a diferentes escalas y ratios. Es decir, realizan la misma función que lo que sustituyen, pero de una forma más optimizada.
7. En Faster R-CNN, las distintas convoluciones se realizan tanto en la RPN como en la R-CNN reduciendo así el tiempo computacional.

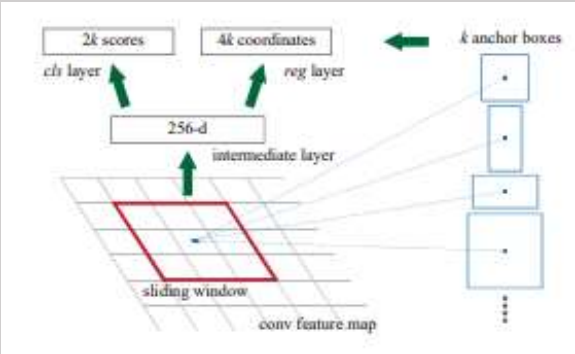


Figura 17. Region Proposal Network (RPN) [36].

Como se ha mencionado, la estructura de la red neuronal será Inception v2. Ésta es la segunda versión de las 4 que existen de la originaria propuesta en septiembre de 2014. La segunda y tercera versión se publicaron en el mismo documento en diciembre de 2015 y la más moderna en agosto de 2016. Inception v2 cuenta con 42 capas y más de 6 millones de neuronas en la totalidad de su red [38]. La estructura de la red se muestra a continuación:

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figura 18. Capas de Inception v2 [38].

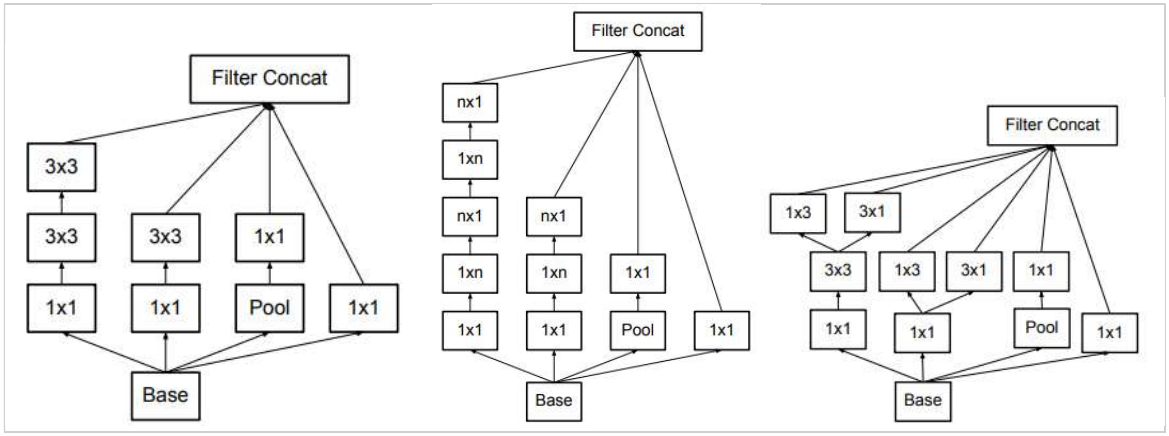


Figura 19. De izquierda a derecha: Figure 5, Figure 6 y Figure 7, referenciadas en la Figura 18 (n=7) [38].

"Stride" es un parámetro que significa el número de pixeles que se va a mover el filtro cada vez que realice una operación como en la Figura 7, en esta estructura se utilizan kernel de dimensión 2x2 [38].

"Convolutional Padding" es una técnica utilizada para que, al realizar una convolución, la imagen final mantenga el mismo tamaño, de esta forma se puedan extraer de mejor manera parámetros característicos de menor nivel. Esta técnica consiste en añadir pixeles extra alrededor de la imagen original. En Inception v2 solo se utiliza esta técnica en la tercera capa de convolución. Más concretamente, se utiliza "Zero padding" que significa que los pixeles añadidos tendrán un valor de 0 [38].

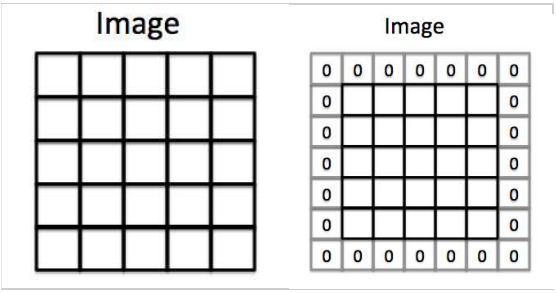


Figura 20. Ejemplo al usar "Zero Padding" = 1 [39].

La única diferencia de las capas mostradas de Inception v2 con las del modelo de este proyecto es que en la última capa la entrada será 1x1x8 ya que solo se tienen 8 clases diferentes en las que clasificar. Por tanto, la estructura a utilizar para este proyecto constará de un total de 42 capas divididas en 32 capas de convolución, 5 de "pooling", 3 de concatenación de filtros, 1 de linearización y 1 de clasificación.

3.2 ENTRENAMIENTO DEL MODELO

En este apartado se van a describir los pasos a realizar para entrenar el modelo. Estos pasos son los que se han seguido para llevar a cabo el entrenamiento del modelo de detección de objetos usado en este proyecto.

Primero habrá que descargar la carpeta “Preparation” del repositorio GitHub del proyecto [1]. En esta carpeta se encuentran 7 ficheros y una carpeta llamada “images” que contiene otras dos carpetas “train” y “test” que a su vez contienen las imágenes con las que se va a entrenar y verificar el modelo, respectivamente. También, incliuyen los documentos XML correspondientes a cada imagen donde almacenan las coordenadas y clasificación de cada elemento que ha sido etiquetado.

En este proyecto, para etiquetar, se ha utilizado el programa LabelImg, un programa gratuito y de uso simple. Una vez abierto este programa, se elegirá el directorio donde se encuentran las imágenes a etiquetar pulsando “Open Dir” y donde se desean guardar los ficheros creados por este programa con “Change Save Dir”. Después, en cada imagen con la herramienta “Create RectBox”, se marcará con un rectángulo que englobe lo que se quiere etiquetar y se elegirá el nombre de la etiqueta que se le desea dar, las coordenadas y la clasificación de cada rectángulo se guardarán en un documento XLM. Sin embargo, para este proyecto no se desea que se reconozca todos los obstáculos de la imagen, inclusive los que se encuentran bastante lejos, sino solo los más próximos ya que el DJI Tello, al no disponer de sensores de distancia, se requiere que solo sea capaz de detectar a través de su cámara los posibles obstáculos inminentes en su camino. Por ello, solo se etiquetarán en las imágenes aquellos más cercanos para que así el modelo solo los identifique cuando se encuentran cerca del drone y pueda chocarse con ellos. Esto puede no ser óptimo para entrenar el modelo ya que le confundirá en varias ocasiones, además, puede que al etiquetar todas las imágenes no haya habido coherencia con la distancia máxima para etiquetar un obstáculo o no.

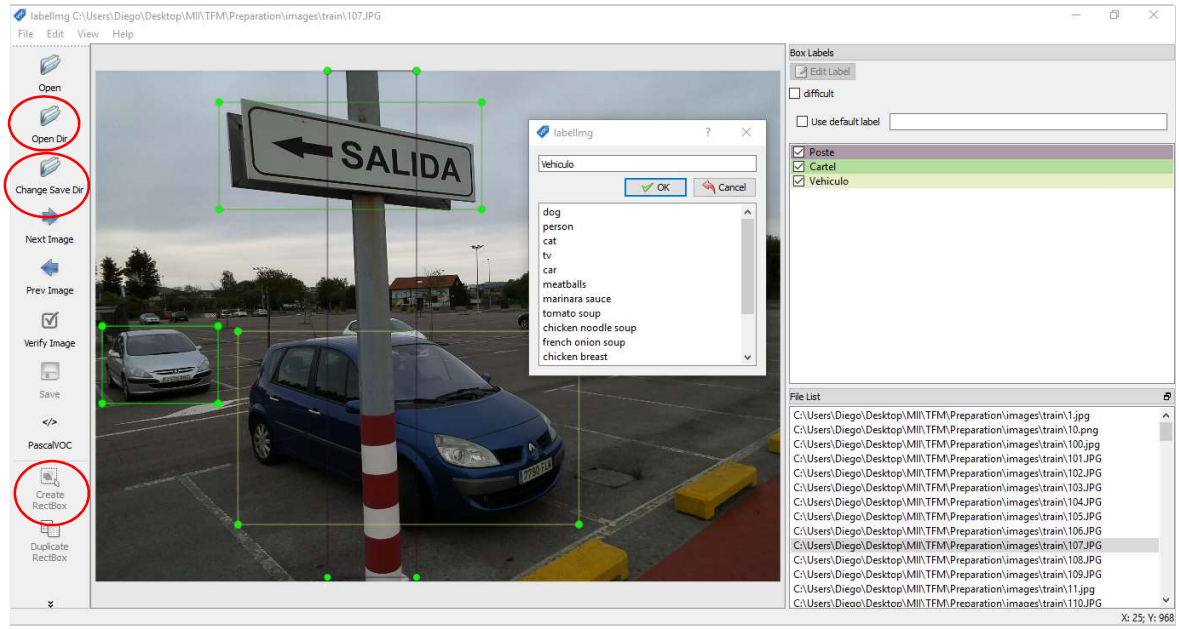


Figura 21. Etiquetación de una imagen usando el programa labelImg.

Entre el resto de ficheros de la carpeta “Preparation”, se encuentran “xml_to_csv.py”, “Object_detection_image.py” y “Object_detection_webcam.py” estos programas han sido aportados por el repositorio de GitHub “EdjeElectronic” [40] cuyo autor es Evan Juras, Ingeniero Hardware y Especialista en Visión Artificial. En este repositorio se pueden encontrar distintos tutoriales educativos para iniciarse en la visión artificial.

El primer archivo de Python corresponde con un programa para convertir documentos XML en archivos CSV. Los otros dos son programas para probar el funcionamiento del modelo una vez entrenado, ambos se utilizarán posteriormente para verificar el desempeño del modelo entrenado para este proyecto. Estos programas están basados en un tutorial de Tensorflow [41] que muestra los principios básicos para utilizar un modelo entrenado con Deep Learning. Para este proyecto, se han actualizado las funciones obsoletas que se utilizaban y se ha variado el código cuanto haya sido necesario para que funcionen correctamente.

Los archivos restantes “faster_rcnn_inception_v2_coco.config”, “labelmap.pbtxt”, “train.py” y “generate_tfrecord.py” son archivos que se pueden encontrar entre los recursos disponibles del repositorio de GitHub de TensorFlow. Sin embargo, no se pueden descargar directamente desde el Google Colab como se va a hacer con otros archivos, ya que hay que modificarlos primero. Las modificaciones consistirán principalmente en introducir el número

de clases y su nombre en los distintos archivos, para que al entrenar sepa cuáles son las salidas que tiene que dar. También, habrá que cambiar los directorios de los archivos externos que se van a utilizar para que sean coherentes y las funciones que estén obsoletas o desactualizadas.

En el documento "labelmap.pbtxt", se asociará cada clase "name" con un número "id". Estos nombres y valores tienen que coincidir con los que se usarán en "generate_tfrecord.py".

```
item {
  id: 1
  name: 'Poste'
}
item {
  id: 2
  name: 'Valla'
}

item {
  id: 3
  name: 'Cartel'
}

item {
  id: 4
  name: 'Arbol'
}

item {
  id: 5
  name: 'Vehiculo'
}
```

```
def class_text_to_int(row_label):
    if row_label == 'Poste':
        return 1
    elif row_label == 'Valla':
        return 2
    elif row_label == 'Cartel':
        return 3
    elif row_label == 'Arbol':
        return 4
    elif row_label == 'Vehiculo':
        return 5
    elif row_label == 'Semaforo':
        return 6
    elif row_label == 'Persona':
        return 7
    elif row_label == 'Muro':
        return 8
    else:
        return 0
```

Figura 22. Definición de clases en "labelmap.pbtxt" y "generate_tfrecord.py".

En el fichero de configuración, se ha cambiado el número de clases a 8, las correspondientes a este proyecto y en "eval_config" el número de muestras se ha cambiado el valor por 23 ya que corresponde con el número de imágenes que se van a utilizar para verificar el modelo.

faster_rcnn_inception_v2_coco: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
model {
  faster_rcnn {
    num_classes: 8
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
```

faster_rcnn_inception_v2_coco: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
    input_path: "train.record"
  }
  label_map_path: "labelmap.pbtxt"
}

eval_config: {
  num_examples: 23
  # Note: The below line limits the
  # Remove the below line to evaluat
  # max_evals: 10
}
```

Figura 23. Cambios mencionados en "faster_rcnn_inception_v2_coco.config".

Para poder empezar a entrenar el modelo, se comprimirá la carpeta “Preparation” y se subirá al Google Drive del usuario. Después, se descargará el documento “Entrenamiento_del_Modelo.ipynb” (archivo compatible para utilizarse en Google Colaboratory), el cual se encuentra en el respositorio GitHub del proyecto [1], se subirá al Drive y se abrirá directamente desde Google Colaboratory o desde el mismo Drive. Una vez abierto, simplemente habrá que ejecutar cada celda de código por orden.

Antes de comenzar, habrá que comprobar que se encuentra habilitada la GPU (Graphic Processing Unit) en el entorno de Google Colaboratory ya que es necesaria para estudiar las imágenes.

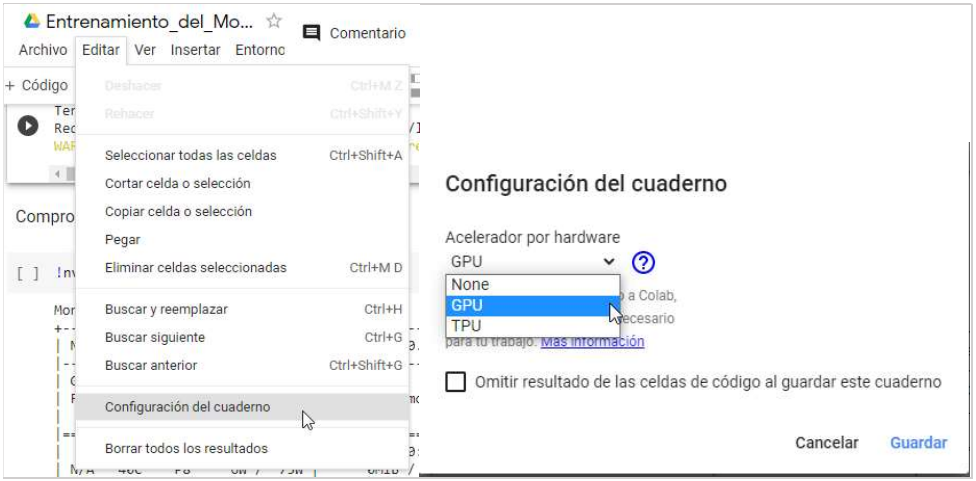


Figura 24. Configuración de Google Colab para utilizar una GPU.

Primero se instalarán TensorFlow y NumPy ya que se usarán a lo largo del proceso. Si se desea, se puede comprobar qué GPU aporta Google Colab para ser utilizada durante el proceso. En el caso de este proyecto se utilizó Tesla K80.

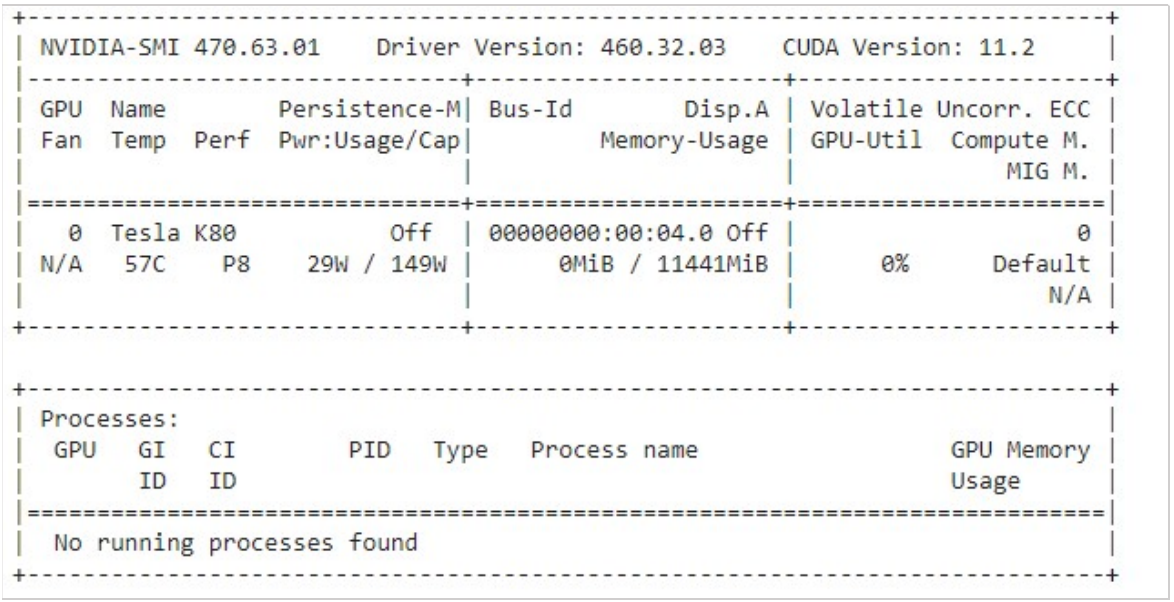


Figura 25. Salida en Colab sobre la GPU utilizada en el proyecto.

Después, se conectará el Google Colaboratory con el Google Drive para permitir que se puedan escribir o coger archivos en Drive desde Colab. Para ello, se clicará en la URL que sale por pantalla tras ejecutar la celda correspondiente, se copiará el código y se introducirá en el recuadro debajo de éste. A continuación, se descargarán dentro de la memoria de Colab el repositorio de GitHub de Tensorflow para poder utilizar los recursos que posibilitan. Dentro de estos recursos se encuentra una carpeta comprimida "faster_rcnn_inception_v2_coco_2018_01_28.tar.gz" que contiene los ficheros Python necesarios para guardar el modelo a lo largo del proceso de entrenamiento, por si se desea pausar y para poder guardarse una vez haya acabado. El siguiente paso será coger la carpeta comprimida "Preparation" del Drive al Colab y descomprimirla. Una vez, descomprimida desde el directorio creado de esta carpeta en el Colab se utilizará el programa de Python "xml_to_csv.py" para convertir los documentos XLM de las imágenes a CSV y posteriormente, con el programa "generate_tfrecord.py", se creará un archivo RECORD que tendrá formato de una tabla con las coordenadas y la clasificación de cada recuadro que engloba un obstáculo. Si se desea, ejecutando la siguiente celda, se puede comprobar cuánto tiempo queda a la GPU para verificar que hay tiempo suficiente para que el modelo sea entrenado y no vaya a haber ningún problema.

Por último, al ejecutar las siguientes dos celdas se comenzará con el entrenamiento del modelo.

Por pantalla, nos irá indicando cómo va evolucionando en modelo.

```
INFO:tensorflow:Recording summary at step 0.
I0911 13:08:50.574235 140162763986688 supervisor.py:1050] Recording summary at step 0.
INFO:tensorflow:global step 1: loss = 3.3661 (23.287 sec/step)
I0911 13:08:55.060641 140166261311360 learning.py:512] global step 1: loss = 3.3661 (23.287 sec/step)
INFO:tensorflow:global step 2: loss = 2.7426 (3.197 sec/step)
I0911 13:08:58.822782 140166261311360 learning.py:512] global step 2: loss = 2.7426 (3.197 sec/step)
INFO:tensorflow:global step 3: loss = 1.9198 (0.321 sec/step)
I0911 13:08:59.145382 140166261311360 learning.py:512] global step 3: loss = 1.9198 (0.321 sec/step)
INFO:tensorflow:global step 4: loss = 1.4817 (3.273 sec/step)
I0911 13:09:02.419842 140166261311360 learning.py:512] global step 4: loss = 1.4817 (3.273 sec/step)
INFO:tensorflow:global step 5: loss = 1.2201 (3.357 sec/step)
I0911 13:09:05.778589 140166261311360 learning.py:512] global step 5: loss = 1.2201 (3.357 sec/step)
INFO:tensorflow:global step 6: loss = 1.2474 (0.300 sec/step)
I0911 13:09:06.080973 140166261311360 learning.py:512] global step 6: loss = 1.2474 (0.300 sec/step)
INFO:tensorflow:global step 7: loss = 1.1397 (0.322 sec/step)
I0911 13:09:06.404882 140166261311360 learning.py:512] global step 7: loss = 1.1397 (0.322 sec/step)
INFO:tensorflow:global step 8: loss = 3.4136 (4.551 sec/step)
I0911 13:09:10.957122 140166261311360 learning.py:512] global step 8: loss = 3.4136 (4.551 sec/step)
INFO:tensorflow:global step 9: loss = 1.3479 (0.306 sec/step)
I0911 13:09:11.265143 140166261311360 learning.py:512] global step 9: loss = 1.3479 (0.306 sec/step)
INFO:tensorflow:global step 10: loss = 2.0025 (0.333 sec/step)
I0911 13:09:11.600367 140166261311360 learning.py:512] global step 10: loss = 2.0025 (0.333 sec/step)
```

Figura 26. Primeras iteraciones al entrenar el modelo del proyecto.

Se muestra cuánto tiempo ha tomado cada “global step”, siendo éste el número de veces que el modelo ha pasado por todas las imágenes de entrenamiento intentando aprenderse los parámetros más característicos para clasificar las imágenes y todas las imágenes de verificación para comprobar cómo de preciso es el modelo. Su precisión se representa con “loss” que corresponde con la pérdida o error que se está cometiendo. Aunque se trate de minimizar el error, es incoherente esperar una pérdida de 0. Se buscará conseguir que el valor de pérdida se encuentra consistentemente por debajo de 0,05 esto corresponde con una probabilidad del 95% de que el modelo prediga correctamente la clasificación de obstáculos. Lógicamente, cuánto más complejo sea el modelo que se quiere desarrollar, mayor tiempo necesitará para llegar a este punto, si es que llega. También hay que tener en cuenta la importancia de la base de datos utilizada, cuanto mayor y más compleja sea más tardará.

En la siguiente imagen, se muestra la pantalla de salida tras 15 minutos de entrenamiento, se puede apreciar cómo ha variado la pérdida desde valores entre 1,1 a 3,3 a valores entre 0,1 y 0,7 aproximadamente.

```
I0911 13:35:18.178996 139649059567488 learning.py:512] global step 2246: loss = 0.3610 (0.302 sec/step)
INFO:tensorflow:global step 2247: loss = 0.6539 (0.310 sec/step)
I0911 13:35:18.490469 139649059567488 learning.py:512] global step 2247: loss = 0.6539 (0.310 sec/step)
INFO:tensorflow:global step 2248: loss = 0.1585 (0.314 sec/step)
I0911 13:35:18.806025 139649059567488 learning.py:512] global step 2248: loss = 0.1585 (0.314 sec/step)
INFO:tensorflow:global step 2249: loss = 0.2663 (0.354 sec/step)
I0911 13:35:19.162350 139649059567488 learning.py:512] global step 2249: loss = 0.2663 (0.354 sec/step)
INFO:tensorflow:global step 2250: loss = 0.1872 (0.311 sec/step)
I0911 13:35:19.475504 139649059567488 learning.py:512] global step 2250: loss = 0.1872 (0.311 sec/step)
INFO:tensorflow:global step 2251: loss = 0.2746 (0.301 sec/step)
I0911 13:35:19.781126 139649059567488 learning.py:512] global step 2251: loss = 0.2746 (0.301 sec/step)
INFO:tensorflow:global step 2252: loss = 0.2433 (0.310 sec/step)
I0911 13:35:20.094027 139649059567488 learning.py:512] global step 2252: loss = 0.2433 (0.310 sec/step)
INFO:tensorflow:global step 2253: loss = 0.3471 (0.287 sec/step)
I0911 13:35:20.383600 139649059567488 learning.py:512] global step 2253: loss = 0.3471 (0.287 sec/step)
INFO:tensorflow:global step 2254: loss = 0.4607 (0.328 sec/step)
I0911 13:35:20.715265 139649059567488 learning.py:512] global step 2254: loss = 0.4607 (0.328 sec/step)
INFO:tensorflow:global step 2255: loss = 0.2899 (0.309 sec/step)
I0911 13:35:21.026721 139649059567488 learning.py:512] global step 2255: loss = 0.2899 (0.309 sec/step)
INFO:tensorflow:global step 2256: loss = 0.3875 (0.315 sec/step)
I0911 13:35:21.343672 139649059567488 learning.py:512] global step 2256: loss = 0.3875 (0.315 sec/step)
```

Figura 27. Iteraciones a los 15 minutos del modelo del proyecto.

Cada cierto tiempo, se guardará automáticamente una copia del proceso por si se desea pausar el entrenamiento y retomarlo partiendo desde el último punto guardado o si se desea pasarlo al Drive una vez finalizado, como máximo se guardarán las 5 últimas copias. Esto habrá que tenerlo en cuenta si se desea guardar el modelo en diferentes puntos del entrenamiento. Se pueden encontrar los distintos puntos de guardado en la carpeta "training" dentro de la carpeta "model-masters/research/objetdetection" descargada del repositorio de TensorFlow, con el nombre model.ckpt-XXXXX siendo XXXXX el número del "global step" en el que se ha hecho la copia.

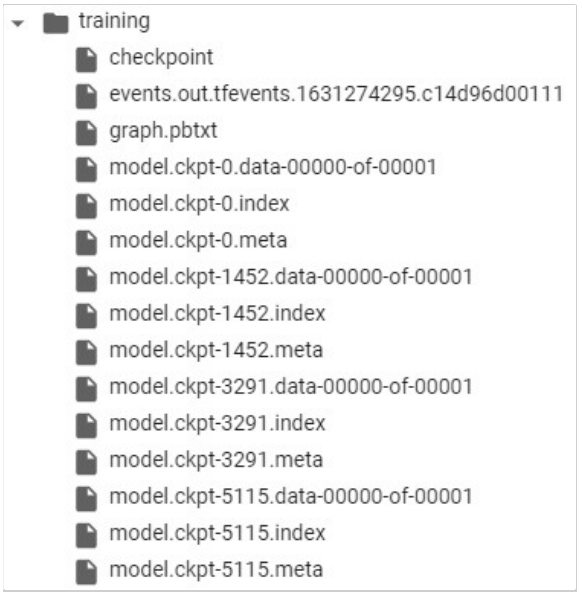


Figura 28. Carpeta "training" con las copias guardadas del modelo del proyecto a los 31 minutos.

Para guardar el modelo, se parará la celda que ejecuta el entrenamiento y se iniciará la siguiente. En ésta, habrá que cambiar en el código donde pone model.ckpt- XXXXX, por los números de la copia que deseemos guardar. Finalmente, se guardarán dos carpetas comprimidas correspondientes al modelo entrenado en el Google Drive, la primera será "model_graph.zip" el cual contendrá archivos que almacenan las operaciones y relaciones de TensorFlow entre las neuronas y la segunda "model-master.zip" contendrá los archivos necesarios para utilizar el modelo entrenado. Si se van a guardar distintas copias, habrá que modificar el nombre con el que se van a guardar para no sobrescribir las anteriores.

```
INFO:tensorflow:global step 4884: loss = 0.0746 (0.307 sec/step)
I0911 13:49:52.837943 139649059567488 learning.py:512] global step 4884: loss = 0.0746 (0.307 sec/step)
INFO:tensorflow:global step 4885: loss = 0.2040 (0.364 sec/step)
I0911 13:49:53.204072 139649059567488 learning.py:512] global step 4885: loss = 0.2040 (0.364 sec/step)
INFO:tensorflow:global step 4886: loss = 0.2934 (0.329 sec/step)
I0911 13:49:53.536610 139649059567488 learning.py:512] global step 4886: loss = 0.2934 (0.329 sec/step)
INFO:tensorflow:global step 4887: loss = 0.2320 (0.299 sec/step)
I0911 13:49:53.837251 139649059567488 learning.py:512] global step 4887: loss = 0.2320 (0.299 sec/step)
INFO:tensorflow:global step 4888: loss = 0.1398 (0.294 sec/step)
I0911 13:49:54.134302 139649059567488 learning.py:512] global step 4888: loss = 0.1398 (0.294 sec/step)
INFO:tensorflow:global step 4889: loss = 0.2125 (0.313 sec/step)
I0911 13:49:54.449737 139649059567488 learning.py:512] global step 4889: loss = 0.2125 (0.313 sec/step)
INFO:tensorflow:global step 4890: loss = 0.3141 (0.304 sec/step)
I0911 13:49:54.755634 139649059567488 learning.py:512] global step 4890: loss = 0.3141 (0.304 sec/step)
INFO:tensorflow:global step 4891: loss = 0.2403 (0.349 sec/step)
I0911 13:49:55.106397 139649059567488 learning.py:512] global step 4891: loss = 0.2403 (0.349 sec/step)
INFO:tensorflow:global step 4892: loss = 0.2486 (0.278 sec/step)
I0911 13:49:55.386531 139649059567488 learning.py:512] global step 4892: loss = 0.2486 (0.278 sec/step)
INFO:tensorflow:global step 4893: loss = 0.0375 (0.369 sec/step)
I0911 13:49:55.757413 139649059567488 learning.py:512] global step 4893: loss = 0.0375 (0.369 sec/step)
INFO:tensorflow:global step 4894: loss = 0.2151 (0.311 sec/step)
I0911 13:49:56.070414 139649059567488 learning.py:512] global step 4894: loss = 0.2151 (0.311 sec/step)
```

Figura 29.Iteraciones tras 30 minutos del modelo del proyecto.

```
INFO:tensorflow:global step 10401: loss = 0.1383 (0.338 sec/step)
I0911 14:20:15.923698 139649059567488 learning.py:512] global step 10401: loss = 0.1383 (0.338 sec/step)
INFO:tensorflow:global step 10402: loss = 0.0733 (0.369 sec/step)
I0911 14:20:16.294073 139649059567488 learning.py:512] global step 10402: loss = 0.0733 (0.369 sec/step)
INFO:tensorflow:global step 10403: loss = 0.0994 (0.318 sec/step)
I0911 14:20:16.614478 139649059567488 learning.py:512] global step 10403: loss = 0.0994 (0.318 sec/step)
INFO:tensorflow:global step 10404: loss = 0.0753 (0.332 sec/step)
I0911 14:20:16.948033 139649059567488 learning.py:512] global step 10404: loss = 0.0753 (0.332 sec/step)
INFO:tensorflow:global step 10405: loss = 0.1916 (0.301 sec/step)
I0911 14:20:17.250883 139649059567488 learning.py:512] global step 10405: loss = 0.1916 (0.301 sec/step)
INFO:tensorflow:global step 10406: loss = 0.0592 (0.328 sec/step)
I0911 14:20:17.580531 139649059567488 learning.py:512] global step 10406: loss = 0.0592 (0.328 sec/step)
INFO:tensorflow:global step 10407: loss = 0.1371 (0.318 sec/step)
I0911 14:20:17.900126 139649059567488 learning.py:512] global step 10407: loss = 0.1371 (0.318 sec/step)
INFO:tensorflow:global step 10408: loss = 0.1502 (0.308 sec/step)
I0911 14:20:18.209966 139649059567488 learning.py:512] global step 10408: loss = 0.1502 (0.308 sec/step)
INFO:tensorflow:global step 10409: loss = 0.0350 (0.334 sec/step)
I0911 14:20:18.545603 139649059567488 learning.py:512] global step 10409: loss = 0.0350 (0.334 sec/step)
INFO:tensorflow:global step 10410: loss = 0.1414 (0.328 sec/step)
I0911 14:20:18.875720 139649059567488 learning.py:512] global step 10410: loss = 0.1414 (0.328 sec/step)
INFO:tensorflow:global step 10411: loss = 0.1176 (0.347 sec/step)
I0911 14:20:19.224147 139649059567488 learning.py:512] global step 10411: loss = 0.1176 (0.347 sec/step)
```

Figura 30. Iteraciones tras 1 hora del modelo del proyecto.

```
INFO:tensorflow:global step 15417: loss = 0.0913 (3.227 sec/step)
I0911 15:04:15.776060 139825492436864 learning.py:512] global step 15417: loss = 0.0913 (3.227 sec/step)
INFO:tensorflow:global step 15418: loss = 0.0297 (0.326 sec/step)
I0911 15:04:16.104312 139825492436864 learning.py:512] global step 15418: loss = 0.0297 (0.326 sec/step)
INFO:tensorflow:global step 15419: loss = 0.0385 (0.322 sec/step)
I0911 15:04:16.427724 139825492436864 learning.py:512] global step 15419: loss = 0.0385 (0.322 sec/step)
INFO:tensorflow:global step 15420: loss = 0.0161 (0.343 sec/step)
I0911 15:04:16.772700 139825492436864 learning.py:512] global step 15420: loss = 0.0161 (0.343 sec/step)
INFO:tensorflow:global step 15421: loss = 0.0490 (0.342 sec/step)
I0911 15:04:17.117154 139825492436864 learning.py:512] global step 15421: loss = 0.0490 (0.342 sec/step)
INFO:tensorflow:global step 15422: loss = 0.0614 (0.303 sec/step)
I0911 15:04:17.422701 139825492436864 learning.py:512] global step 15422: loss = 0.0614 (0.303 sec/step)
INFO:tensorflow:global step 15423: loss = 0.0117 (0.345 sec/step)
I0911 15:04:17.769618 139825492436864 learning.py:512] global step 15423: loss = 0.0117 (0.345 sec/step)
INFO:tensorflow:global step 15424: loss = 0.0732 (0.338 sec/step)
I0911 15:04:18.109131 139825492436864 learning.py:512] global step 15424: loss = 0.0732 (0.338 sec/step)
INFO:tensorflow:global step 15425: loss = 0.0710 (0.318 sec/step)
I0911 15:04:18.429002 139825492436864 learning.py:512] global step 15425: loss = 0.0710 (0.318 sec/step)
INFO:tensorflow:global step 15426: loss = 0.0549 (0.311 sec/step)
I0911 15:04:18.741753 139825492436864 learning.py:512] global step 15426: loss = 0.0549 (0.311 sec/step)
INFO:tensorflow:global step 15427: loss = 0.0735 (0.300 sec/step)
I0911 15:04:19.043581 139825492436864 learning.py:512] global step 15427: loss = 0.0735 (0.300 sec/step)
```

Figura 31. Iteraciones tras 1 hora y 30 minutos del modelo del proyecto.

```
INFO:tensorflow:global step 20127: loss = 0.0119 (0.307 sec/step)
I0911 15:36:59.314820 140066386454400 learning.py:512] global step 20127: loss = 0.0119 (0.307 sec/step)
INFO:tensorflow:global step 20128: loss = 0.0565 (0.317 sec/step)
I0911 15:36:59.633511 140066386454400 learning.py:512] global step 20128: loss = 0.0565 (0.317 sec/step)
INFO:tensorflow:global step 20129: loss = 0.0375 (0.314 sec/step)
I0911 15:36:59.949322 140066386454400 learning.py:512] global step 20129: loss = 0.0375 (0.314 sec/step)
INFO:tensorflow:global step 20130: loss = 0.0797 (0.300 sec/step)
I0911 15:37:00.251440 140066386454400 learning.py:512] global step 20130: loss = 0.0797 (0.300 sec/step)
INFO:tensorflow:global step 20131: loss = 0.0072 (0.327 sec/step)
I0911 15:37:00.580372 140066386454400 learning.py:512] global step 20131: loss = 0.0072 (0.327 sec/step)
INFO:tensorflow:global step 20132: loss = 0.1020 (0.369 sec/step)
I0911 15:37:00.951077 140066386454400 learning.py:512] global step 20132: loss = 0.1020 (0.369 sec/step)
INFO:tensorflow:global step 20133: loss = 0.0192 (0.311 sec/step)
I0911 15:37:01.264502 140066386454400 learning.py:512] global step 20133: loss = 0.0192 (0.311 sec/step)
INFO:tensorflow:global step 20134: loss = 0.2061 (0.312 sec/step)
I0911 15:37:01.578792 140066386454400 learning.py:512] global step 20134: loss = 0.2061 (0.312 sec/step)
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
I0911 15:37:01.656275 140062872426240 supervisor.py:1117] Saving checkpoint to path training/model.ckpt
INFO:tensorflow:global step 20135: loss = 0.0454 (0.908 sec/step)
I0911 15:37:02.505880 140066386454400 learning.py:512] global step 20135: loss = 0.0454 (0.908 sec/step)
INFO:tensorflow:global step 20136: loss = 0.0300 (0.808 sec/step)
I0911 15:37:03.332909 140066386454400 learning.py:512] global step 20136: loss = 0.0300 (0.808 sec/step)
INFO:tensorflow:Recording summary at step 20136.
I0911 15:37:03.339537 140062855640832 supervisor.py:1050] Recording summary at step 20136.
INFO:tensorflow:global step 20137: loss = 0.0349 (0.508 sec/step)
I0911 15:37:03.856122 140066386454400 learning.py:512] global step 20137: loss = 0.0349 (0.508 sec/step)
INFO:tensorflow:global step 20138: loss = 0.0333 (0.438 sec/step)
I0911 15:37:04.298102 140066386454400 learning.py:512] global step 20138: loss = 0.0333 (0.438 sec/step)
INFO:tensorflow:global step 20139: loss = 0.0596 (0.337 sec/step)
```

Figura 32. Iteraciones tras 2 horas y 5 minutos del modelo del proyecto.

```
INFO:tensorflow:global step 27189: loss = 0.0536 (0.335 sec/step)
I0911 16:26:45.863098 139944346978176 learning.py:512] global step 27189: loss = 0.0536 (0.335 sec/step)
INFO:tensorflow:global step 27190: loss = 0.0410 (0.306 sec/step)
I0911 16:26:46.171482 139944346978176 learning.py:512] global step 27190: loss = 0.0410 (0.306 sec/step)
INFO:tensorflow:global step 27191: loss = 0.0170 (0.331 sec/step)
I0911 16:26:46.504390 139944346978176 learning.py:512] global step 27191: loss = 0.0170 (0.331 sec/step)
INFO:tensorflow:global step 27192: loss = 0.0114 (0.370 sec/step)
I0911 16:26:46.876196 139944346978176 learning.py:512] global step 27192: loss = 0.0114 (0.370 sec/step)
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
I0911 16:26:47.044705 139940832462592 supervisor.py:1117] Saving checkpoint to path training/model.ckpt
INFO:tensorflow:global step 27193: loss = 0.0304 (0.525 sec/step)
I0911 16:26:47.403656 139944346978176 learning.py:512] global step 27193: loss = 0.0304 (0.525 sec/step)
INFO:tensorflow:global step 27194: loss = 0.0186 (0.574 sec/step)
I0911 16:26:47.999740 139944346978176 learning.py:512] global step 27194: loss = 0.0186 (0.574 sec/step)
INFO:tensorflow:global step 27195: loss = 0.1342 (0.608 sec/step)
I0911 16:26:48.837736 139944346978176 learning.py:512] global step 27195: loss = 0.1342 (0.608 sec/step)
INFO:tensorflow:Recording summary at step 27195.
I0911 16:26:48.838253 139940815677184 supervisor.py:1050] Recording summary at step 27195.
INFO:tensorflow:global step 27196: loss = 0.0112 (0.469 sec/step)
I0911 16:26:49.355425 139944346978176 learning.py:512] global step 27196: loss = 0.0112 (0.469 sec/step)
INFO:tensorflow:global step 27197: loss = 0.1020 (0.372 sec/step)
I0911 16:26:49.729706 139944346978176 learning.py:512] global step 27197: loss = 0.1020 (0.372 sec/step)
INFO:tensorflow:global step 27198: loss = 0.1231 (0.335 sec/step)
I0911 16:26:50.066988 139944346978176 learning.py:512] global step 27198: loss = 0.1231 (0.335 sec/step)
INFO:tensorflow:global step 27199: loss = 0.0470 (0.342 sec/step)
I0911 16:26:50.410933 139944346978176 learning.py:512] global step 27199: loss = 0.0470 (0.342 sec/step)
INFO:tensorflow:global step 27200: loss = 0.0098 (0.306 sec/step)
```

Figura 33. Iteraciones tras 2 horas y 50 minutos del modelo del proyecto.

En las anteriores imágenes se muestran extractos del entrenamiento del modelo que se utilizará en el proyecto. Se puede apreciar que la reducción del valor de pérdida es más acentuada al comienzo del entrenamiento y a medida que va entrenando necesita más tiempo para variar de la misma manera hasta llegar a un punto donde prácticamente se estabiliza como se podría considerar al comparar entre la Figura 32 y Figura 33. Entre ellas transcurren 45 minutos, pero en ambas se puede ver que el valor de pérdida varía aproximadamente entre 0,01 y 0,1 por lo que no hay un gran progreso. También, se puede observar que en varios “global step” seguidos el valor de pérdida se encuentra controlado por debajo de 0,05 y en el siguiente “global step” pasa a un valor superior a 0,1 esto se debe a que como se mencionó anteriormente, la función de entropía cruzada cuando hay un gran error al clasificar hace que este valor aumente considerablemente, desajustando el control que se había conseguido en los pasos anteriores. Por esto, al no notar una mejora considerable tras 45 minutos y que aproximadamente 3 horas para una base de datos relativamente pequeña es más que suficiente, se paró el entrenamiento en este punto y se guardó el modelo para ser utilizado como el modelo entrenado con Deep Learning para reconocimiento de obstáculos de este proyecto. Este modelo ha sido entrenado por aproximadamente 2 horas y 50 minutos, habiendo estudiado la base de datos una totalidad de 27192 veces correspondiendo un valor de pérdida para ese punto de 0,0114.

```
INFO:tensorflow:global step 27192: loss = 0.0114 (0.370 sec/step)
I0911 16:26:46.876196 139944346978176 learning.py:512] global step 27192: loss = 0.0114 (0.370 sec/step)
```

Figura 34. Punto de entrenamiento en el que se guarda el modelo utilizado en este proyecto.

Durante la elaboración del proyecto, se ha realizado el proceso de entrenamiento varias veces por diversas razones, siendo 8 horas la mayor duración de uno de estos procesos. En todas ellas las características del proceso coincidían con las mostradas anteriormente. Sin embargo, es imposible que el mismo modelo entrenado en dos ocasiones diferentes obtenga los mismos resultados. En algunos casos, al entrenar el modelo, tras cierto tiempo de obtener valores de pérdida entre 0,01 y 0,1, este valor empeoraba hasta valores superiores de los que se obtuvieron al inicio del proceso. Esto guarda relación con el desajuste que se crea al cometer un gran error y como se realimenta el modelo con este desajuste. Esto puede ocurrir en distintos puntos del entrenamiento sin relación aparente, por ejemplo, en la elaboración de este proyecto se manifestó tanto a las 2 horas y media en un caso como tras las 6 horas en otro.

Cabe destacar que en Google Colab se puede ver en cada momento la memoria que está utilizando. La primera vez que se intentó entrenar el modelo para este proyecto se saturaba a los pocos segundos de iniciarse llegando hasta el límite de capacidad de memoria RAM (sobre 16 GB). Esto fue debido a que algunas imágenes de la base de datos original fueron tomadas con un móvil con una cámara de alta calidad por lo que ocupaban una gran capacidad. Una vez eliminadas, el modelo podía ser entrenado sin problema, siendo la memoria RAM de alrededor 2,5 GB. Por ello, se recomienda no utilizar imágenes con un tamaño superior a entorno 2000 KB [42].

Figura 35. Capacidad utilizada tras 1h 22 min en el entrenamiento del modelo del proyecto.

3.3 RESULTADOS

Para comprobar el funcionamiento del modelo se pueden utilizar los ficheros de Python de Evan Juras mencionados anteriormente. Para ello, habrá que descargarse las carpetas comprimidas “model-master.zip” y “model_graph.zip” del Google Drive y descomprimirlas. Los únicos ficheros que se van a utilizar, y tienen que estar en la misma carpeta para que funcionen correctamente, son la carpeta “new_graph”, que se encuentra en “model_graph”, la carpeta “object_detection, que se encuentra en “models-master\content\models-master\research\”, las carpetas “utils”, “training” y “Preparation”, que se encuentran en “models-master\content\models-master\research\object_detection”, y los ficheros de Python de Evan Juras. A continuación, se verificarán en estos ficheros que el número de clases es el correcto y que las direcciones entre los ficheros son coherentes. Por último, simplemente habrá que ejecutar el programa que se desee.

A continuación, se muestran 4 resultados al ejecutar el programa “Object_detection_image.py” con el modelo entrenado y con distintas imágenes utilizadas para verificar el modelo. Para elegir qué imagen se va a estudiar, habrá que modificar la línea 33 del código de dicho programa.

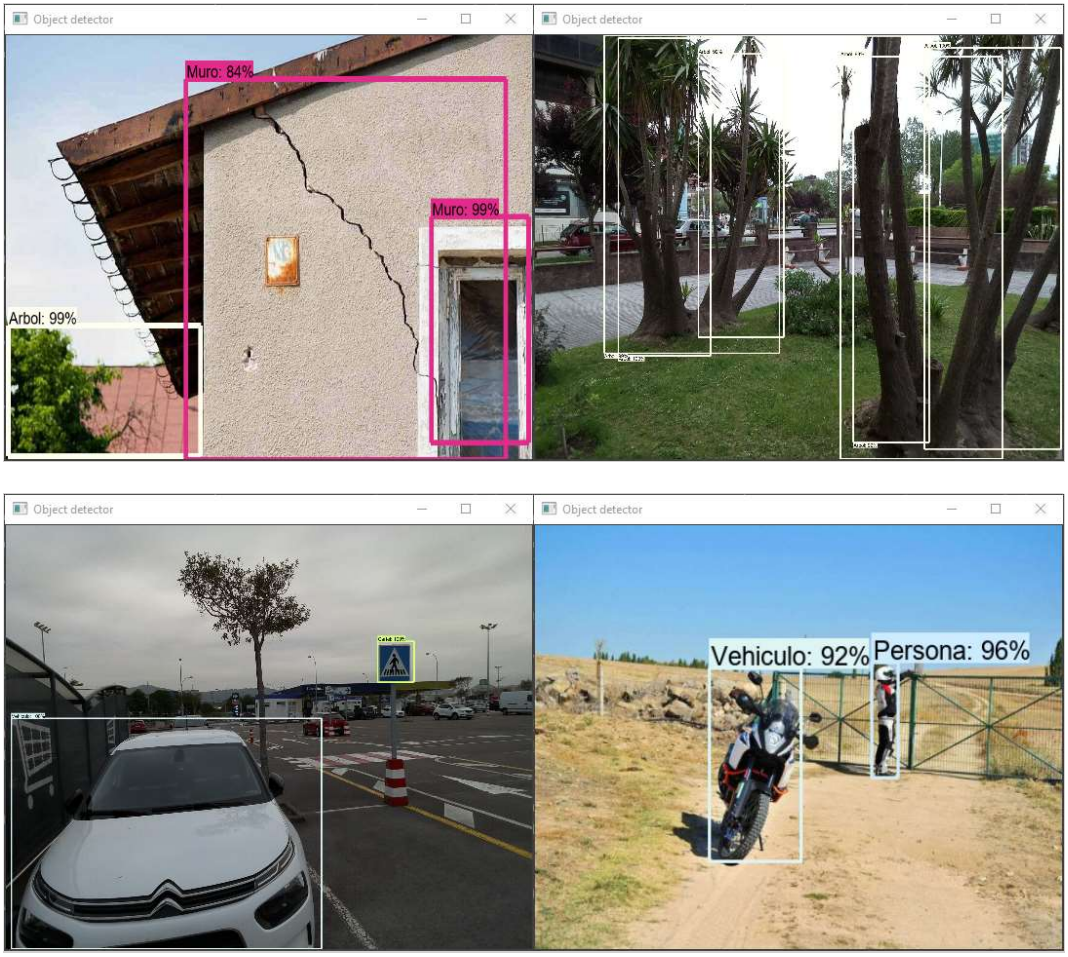


Figura 36. Detección de obstáculos con el modelo del proyecto correspondientes imágenes de la base de datos para verificar el entrenamiento. Imágenes por orden "95", "48", "52" y "100".

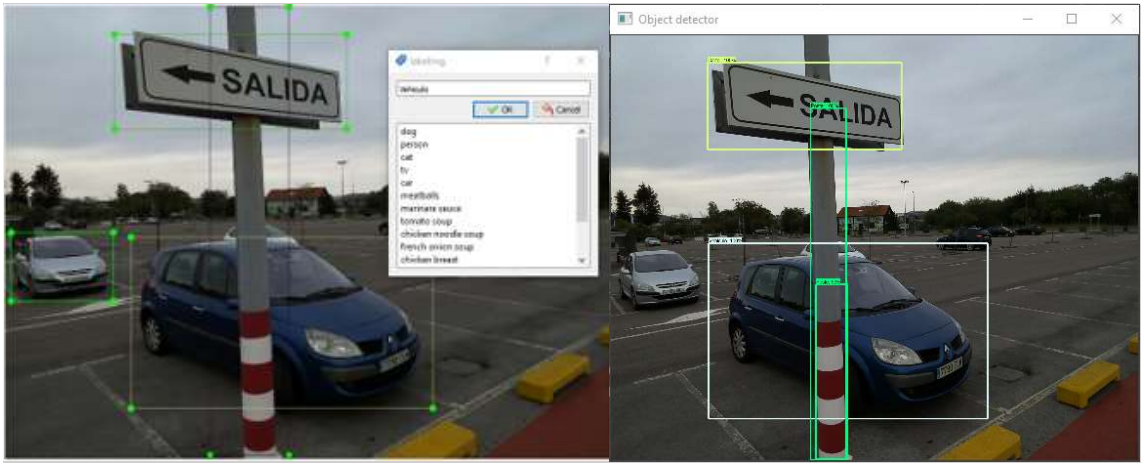


Figura 37. Comparación clasificación manual (Izquierda) con clasificación por el modelo del proyecto (Derecha) de la imagen "107".

También, en las siguientes imágenes se muestran las predicciones del modelo del proyecto con fotografías independientes a la base de datos con la que se entrenó y en un entorno nuevo a los de la base de datos. Estas imágenes se pueden encontrar en "Extras" del repositorio del proyecto [1] con el nombre de "Prueba1.jpg", "Prueba2.jpg" y "Prueba3.jpg"

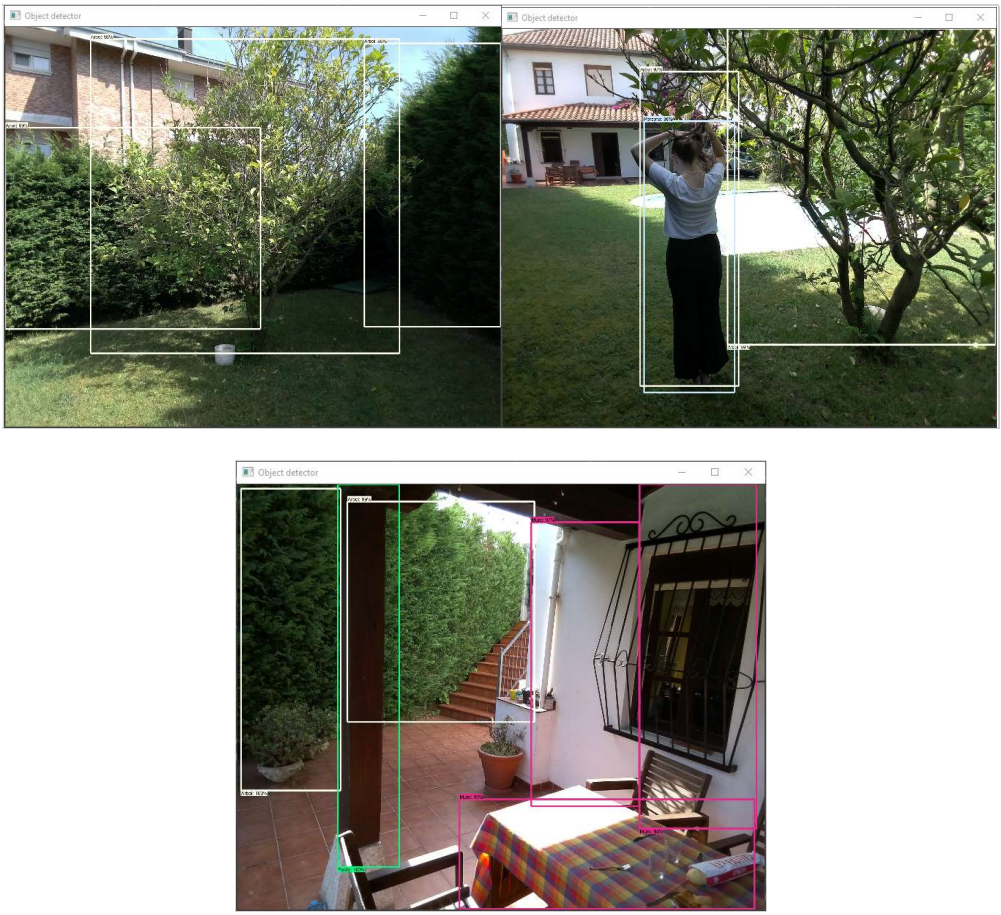


Figura 38. Predicciones del modelo con imágenes independientes a las usadas para ser entrenado.

Como se puede apreciar, en ocasiones clasifica un obstáculo más de una vez o no clasifica un posible obstáculo relativamente próximo. A pesar de ello, se considera que el modelo detecta objetos con una precisión más que notable en la gran mayoría de casos. Además, todos los posibles obstáculos más problemáticos de cada imagen son identificados satisfactoriamente.

3.4 RESUMEN

En la siguiente tabla se muestra un resumen de las características del modelo entrenado con Deep Learning para identificar obstáculos que se usará en este proyecto.

Modelo entrenado con Deep Learning para este proyecto			
Clases a identificar	8	Tiempo entrenado	2 horas 50 minutos
"global step" del entrenamiento	27192	Pérdida para ese "global step"	0,0114
Imágenes usadas para entrenar	93	Imágenes usadas para verificación	22
Obstáculos para entrenar	313	Obstáculos para verificación	74
Imágenes totales	115	Obstáculos totales	387
Tipo de Red Neuronal	Faster R-CNN	Estructura de Red Neuronal	Inception v2
Capas de Red Neuronal	42	Neuronas de Red Neuronal	Más de 6 millones
Capas de Convolución	32	Capas de Pooling	5
Capas de Linearización	1	Capas de Clasificación	1
Capas de Concatenación de Filtros	3	Dimensión Kernel	2x2

Tabla 2. Resumen de las características del modelo de este proyecto.

4.- ALGORITMOS DE PLANIFICACIÓN DE TRAYECTORIAS Y EVASIÓN DE OBSTÁCULOS

En esta sección se presentarán y desarrollarán distintos tipos de algoritmos utilizados comúnmente para la planificación de trayectorias y evasión de obstáculos. Concretamente, se tratarán cinco algoritmos diferentes: el algoritmo de Dijkstra, el algoritmo A*, el algoritmo D*, el algoritmo LPA* y, finalmente, el algoritmo que se empleará en este proyecto, D*Lite.

La finalidad de estos algoritmos es elaborar una secuencia de acciones a desarrollar para alcanzar un objetivo deseado. En el caso de aplicación en planificación de trayectorias, consistirá en un plan de acciones para mover al robot de un punto a otro. En gran medida, se tratará de un problema geométrico para seguir una trayectoria que esquive los obstáculos conocidos en el entorno.

Para ello, el entorno será definido como una malla con cierto número de nodos o vértices, siendo cada nodo una posición en el plano. Los arcos que unen los nodos pueden llevar consigo un peso o coste, por ejemplo, la distancia física entre ellos. El objetivo, por tanto, es construir un camino desde el nodo inicial o de salida al nodo de destino. Sin embargo, puede haber más de una trayectoria posible, incluso infinitas, aunque normalmente la de mayor interés es aquella más corta. Por eso, estos algoritmos a través de cálculos numéricos establecerán la ruta más corta, la que menos peso o coste lleve asociada, desde el punto inicial al punto final en el plano.

La idea principal es, utilizando uno de estos algoritmos, obtener una secuencia de nodos de la malla que extrapolándolo al mundo físico se conviertan en los puntos por donde tiene que pasar el drone DJI Tello de forma fluida hasta llegar a la localización meta deseada esquivando los obstáculos conocidos en cada momento gracias al modelo entrenado con Deep Learning.

4.1 ALGORITMO DE DIJKSTRA

El algoritmo de Dijkstra fue publicado por primera vez en 1959. Este algoritmo calcula el coste mínimo entre un nodo o vértice y sus vecinos. Desde el nodo origen hasta el nodo final irá sumando el coste mínimo entre un nodo y su vecino, pudiéndose rectificar el coste mínimo hasta un nodo en iteraciones siguientes. El coste del nodo origen será 0 y el valor del resto

en su inicio será un valor infinito relativo, estos valores se irán recalculando a lo largo del proceso del algoritmo. Finalmente, se obtendrá una secuencia de nodos desde el origen hasta el final que llevará asociada el coste mínimo de todas las posibles.

Una de las limitaciones de este algoritmo es la potencia computacional que requiere. Siendo V el número de nodos de un gráfico y E el número de arcos entre nodos, el poder computacional crece cuadráticamente con el número de nodos $O(|V|^2)$. Existe una variante de este algoritmo que reduce la complejidad computacional de una forma que crece de una forma más lenta $O((|E|+|V|)\cdot log(|V|))$, esta variante está basada en una modificación de la estructura de la lista de nodos llamada cola de prioridad utilizando montículos de Fibonacci. La cola de prioridad es una lista donde almacena conjuntamente el nodo y su prioridad o valor clave ("key" en inglés), en caso de que haya un empate en el valor clave, el orden se elige arbitrariamente. En computación, los montículos ("heap" en inglés) son una estructura del tipo árbol que se utilizan cuando se desea insertar o borrar un nodo en la cola de prioridad. Además, para que funcione correctamente los costes entre nodos no pueden ser negativos [43].

A pesar de su demanda computacional, este algoritmo es utilizado en bastantes escenarios simples y no tan simples como enrutamiento de aviones y tráfico aéreo [44].

El pseudocódigo se muestra a continuación, donde la prioridad será el valor *n.distancia*:

Pseudocódigo

Inputs: *Gráfico* ***G*** *con n nodos y costes* $\boldsymbol{c(n_i \rightarrow n_j)}$ *no negativos entre nodos, nodo inicial* ***v*** *y nodo final* ***w***

Output: *Lista* ***D*** *que almacena la secuencia de nodos hasta el nodo final con el menor coste posible*

Inicialización:

v.distancia = 0

n.distancia = ∞ para n =! v

Crear lista vacía D = 0

Q = Lista de prioridad, añadir cada nodo de G a Q

Loop principal:

While Q no esté vacía:

Extraer no con menor n.distancia de Q → u

If u = w

break

For cada nodo n vecino de u:

If n.distancia > u.distancia+c(u→n)

n.distancia= u.distancia+c(u→n)

añadir n a D si no lo está ya

En la siguiente imagen se puede ver un ejemplo que explica de manera gráfica cómo funciona el algoritmo. En este caso, muestra el camino más corto para cada nodo “y”, “t”, “x” y “z”, además del coste mínimo desde el nodo inicial “s” hasta cada nodo.

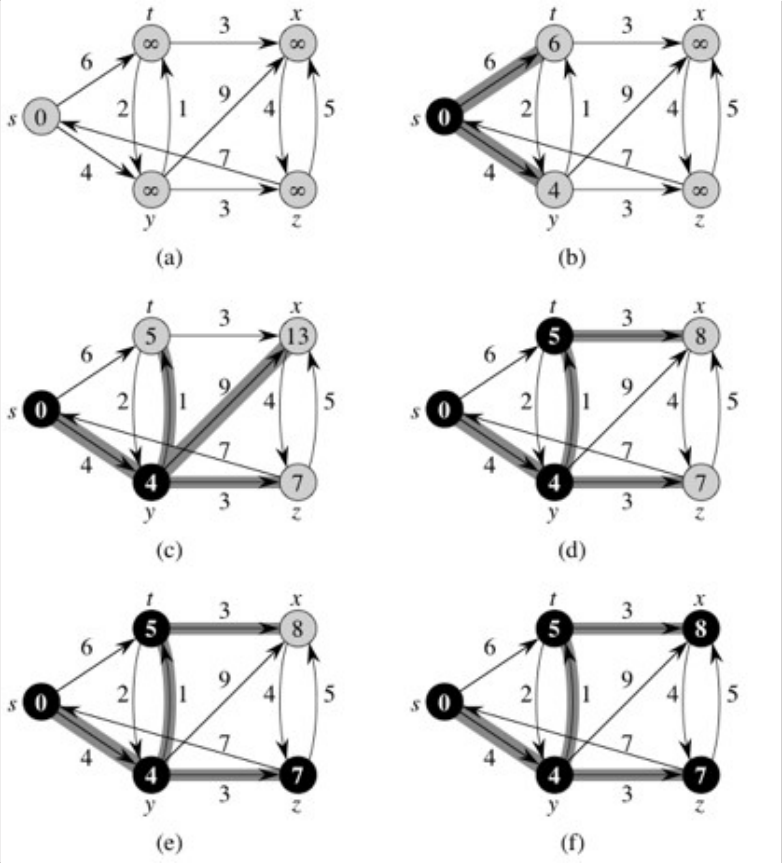


Figura 39. Ejemplo del funcionamiento del algoritmo de Djiskstra [45].

4.2 ALGORITMO A*

Este algoritmo fue desarrollado en 1968 y es uno de los más populares en planificación de rutas ya que es bastante flexible y se puede utilizar en una gran variedad de contextos. El algoritmo A* o "A-star" toma como referencia el algoritmo anterior, sin embargo, elude las limitaciones de éste y optimiza el tiempo computacional necesario debido a su simplicidad, lo que representa su mayor ventaja. El algoritmo de Dijsktra prácticamente estudia todos o casi todos los vértices del gráfico, lo que puede suponer un problema cuando se tienen un gran número de ellos. En cambio, el algoritmo A* utiliza una función heurística para "guiar" la trayectoria hacia el destino, reduciendo el número de nodos que estudia y, con ello, el tiempo necesario para evaluar el camino más corto. En cualquier vértice, si un tramo a otro vértice tiene mayor coste que otro segmento de ruta ya encontrado, abandona el tramo de mayor coste y se dirige al tramo de menor coste, este proceso continúa hasta alcanzar el vértice de meta deseado.

Como el nodo final es conocido, se puede estimar la distancia entre el nodo inicial y el final a través de la distancia heurística. Puede haber casos en los que no consiga obtener el camino más corto, ya que sea necesario empezar visitando un nodo que se encuentre en la dirección contraria al nodo final. En el peor de los casos, el desempeño de este algoritmo será prácticamente el mismo que el de Djikstra.

La terminología que se usa normalmente al tratar estos temas es, $g(n)$ siendo el coste de la trayectoria del punto de origen a cualquier vértice n y $h(n)$ la estimación heurística de coste del vértice n al punto de meta. En cada iteración del bucle principal del algoritmo, examinará el nodo n que tengo el menor $f(n)=g(n)+h(n)$ añadiéndoles a una lista que finalmente será la secuencia de nodos a seguir desde el inicio al destino. Por tanto, $f(n)$ es la actual aproximación al camino más corto a la meta. Este valor además será el valor de prioridad utilizado en la cola de prioridad. Debido a esto, el algoritmo de Djikstra podría considerarse un caso particular del algoritmo A* donde $h(n) = 0$ para cualquier valor de n [46].

Cuánto más precisa sea la estimación heurística, con mayor rapidez y precisión será desarrollada la trayectoria más corta. Las estimaciones heurísticas que se suelen utilizar más comúnmente en una malla plana son las siguientes:

- **Distancia Euclídea**

$$h(x_n, y_n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2}$$

- **Distancia Manhattan**

$$h(x_n, y_n) = |x_n - x_g| + |y_n - y_g|$$

Este algoritmo es utilizado tanto en videojuegos como en planificación de trayectorias para robot autónomos en entornos conocidos.

A continuación, se muestra el pseudocódigo de este algoritmo:

Pseudocódigo

Inputs: *Gráfico* ***G*** *con n nodos y costes* ***c(n_i→n_j)*** *no negativos entre nodos, nodo inicial* ***v*** *y nodo final* ***w***

Output: *Lista* ***D*** *que almacena la secuencia de nodos hasta el nodo final con el menor coste posible*

Inicialización:

f(n)= ∞ ; g(n) = ∞ para cada n =! v

Crear lista vacía D = 0

g(v) = 0; f(v) = h(v,w), añadir v a D

Q = Lista de prioridad, añadir cada nodo de G a Q

Loop principal:

While Q no esté vacía:

    Extraer nodo con mínimo f(n) de Q → u

    If u = w

        Break

    For cada nodo n vecino de u:

        If g(n) > g(u) + c(u→n):

            g(n) = g(u) + c(u→n)

            f(n) = g(n) + h(n,w)

            añadir n a la lista D si no lo está ya

En la siguiente imagen se puede ver una comparación de los dos algoritmos anteriores para un mismo caso. La celda verde es el nodo de origen, la celda amarilla es el nodo de destino, las celdas rojas son los nodos ya estudiados y los nodos azules son los siguientes posibles nodos a estudiar.

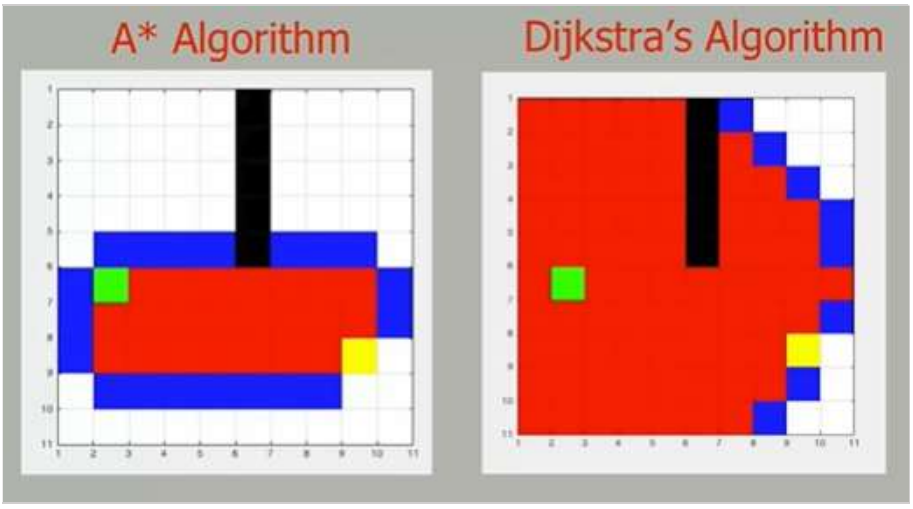


Figura 40. Comparación entre el algoritmo A y el algoritmo de Dijkstra [47].*

Como se puede apreciar, el algoritmo A* estudia un número de nodos considerablemente menor que el algoritmo Dijkstra y, por ello, utiliza menos potencia computacional ya que tarda menos tiempo en obtener los recursos necesarios para desarrollar el camino más corto entre el nodo de partida y el nodo objetivo.

4.3 ALGORITMO D*

El algoritmo D* o "D-star" se publicó por primera vez en 1994 por A. Stentz. Como se ha visto, los algoritmos citados anteriormente tienen gran utilidad y aplicación, pero no consideran la problemática de un entorno poco conocido o cambiante. Aquí es donde entra el algoritmo D*, también conocido como algoritmo A* Dinámico. Este tipo de algoritmos como el D* son llamados de cálculo inverso ya que almacena información de los cálculos iniciales que se tendrán en cuenta para recalcular la ruta una vez que se produce un cambio en el entorno [48].

Este algoritmo permite que el coste entre nodos pueda variar a lo largo del proceso resolutivo replanteando la trayectoria. Inicialmente utiliza el algoritmo de Dijkstra para planear la ruta, pero una vez se varían los costes entre nodos, necesita muy poco poder computacional para recalcular la ruta ya que en general la mayoría de costes se mantienen constantes y los cambios locales solo influyen en los costes de los nodos vecinos del que se encuentran, permitiendo así, variar lo mínimo posible la trayectoria inicial. Por ello, las principales ventajas y, las razones por las que se ha decidido utilizar este algoritmo, son su nivel de optimización y eficiencia en tiempo real en entornos en los que se desarrollan cambios dinámicos o de los que se dispone de poca información en un inicio.

La idea principal de este algoritmo es que en el cálculo inicial computa el coste mínimo de cada nodo hasta el nodo final. Por tanto, se puede obtener el camino más corto yendo a través de los nodos con menor coste. Una vez se encuentra con un nodo en el camino que ha variado, el algoritmo recalcula los costes y actualiza el camino si es necesario. Como ya se han obtenido los costes mínimos con anterioridad, no es necesario que recalcule todos los nodos otra vez sino solo los del entorno del nodo que ha variado su coste. Una de las grandes desventajas de este algoritmo es que el cálculo inicial toma un tiempo considerable. Existen modificaciones posteriores de este algoritmo para hacerlo más rápido y eficaz como el algoritmo D* Enfocado del propio A. Stentz [49].

4.4 ALGORITMO LPA*

El algoritmo LPA* o “Lifelong Planning A star” (2001) es una versión mejorada del algoritmo A* mencionado anteriormente. La principal diferencia es que este algoritmo sí tiene en cuenta si el coste cambia entre nodos, concretamente si aumenta o decrece, por lo que se puede utilizar en entornos dinámicos. En cambio, cada vez que un nodo varíe y se quiera obtener una nueva ruta, habría que usar de nuevo este algoritmo, lo cual no es para nada óptimo. Por eso mismo, los autores de dicho algoritmo S. Koenig y M. Likhachev desarrollaron un nuevo algoritmo en 2005 que solucionó esta problemática, este algoritmo es conocido como D*Lite del cual se hablará en el siguiente punto.

Entendiendo como predecesor de un nodo n todo aquel nodo que llegue a dicho nodo n y sucesor cualquier nodo que provenga del nodo n, todos los nodos excepto el nodo inicial y el

nodo final tendrán antecesores y sucesores. Este algoritmo añade el valor *rhs(n)* como una aproximación futura de la distancia al nodo inicial basada en los valores *g(n)* de los predecesores *n'* del nodo.

$$rhs(n) = \begin{cases} 0 & si\ n = nodo\ inicial \\ \min\big(g(n') + c(n' \to n)\big) & en\ cualquier\ otro\ caso \end{cases}$$

Cuando *rhs(n) = g(n)* significará que el nodo es localmente consistente, es decir, no ha habido cambios en los costes de los predecesores. Sin embargo, cuando eso no ocurra, se convertirá en un nodo localmente inconsistente que será colocado en la lista de prioridad Q para ser reevaluado.

Si el primer nodo de la lista tiene un *rhs(n) < g(n),* denominado sobreconsistente, el valor de *g(n)* se iguala al valor de *rhs(n)* haciéndolo consistente y, entonces, se elimina de la lista. Si *rhs(n) > g(n)* convierte el valor de *g(n)* en infinito convirtiéndolo en sobreconsistente o localmente consistente y, entonces, eliminado de la lista. Ahora que se han cambiado los valores de *g(n)* se actualizarán los valores de *rhs* de los sucesores y se reevaluará la ruta, este proceso continuará hasta alcanzar el nodo final y este sea localmente consistente [50].

En este algoritmo, el valor clave de la cola de prioridad consistirá en un vector de dos componentes los cuales se muestran a continuación:

$$k(n) = [k_1(n); k_2(n)]$$

$$k(n) = \begin{cases} k_1(n) = \min(g(n), rhs(n) + h\big(n, n_{objetivo}\big)) \\ k_2(n) = \min(g(n), rhs(n)) \end{cases}$$

El primer valor corresponde con los valores de f, $f(n) = g(n) + h(n, n_{objetivo})$, usados de igual manera en el algoritmo A*. El segundo componente corresponde con los valores g de A*. El primer término será la comparación para determinar el orden en la cola de prioridad y en caso de empate, se comparará el segundo término del valor clave para el que tenga un valor menor, adquiera una mayor prioridad [50].

A continuación, se muestra el pseudocódigo del algoritmo LPA*:

Pseudocódigo

Inputs: *Gráfico* ***G*** *con n nodos y costes* $\boldsymbol{c(n_i \to n_j)}$ *no negativos entre nodos, nodo inicial* ***v*** *y nodo final* ***w***

Output: *Lista* ***D*** *que almacena la secuencia de nodos hasta el nodo final con el menor coste posible*

Función Inicialización():

rhs(n) = g(n) = ∞ para cada n =! v

rhs(v)= g(v) = 0

Crear lista vacía D = 0

añadir v a D

Q = Lista de prioridad, añadir v a Q con su prioridad

Función CaminoMasCorto():

While menor prioridad de Q < prioridad de w OR rhs(w) != g(w):

Extraer nodo con menor prioridad de Q → u

If g(u) > rhs(u)

g(u) = rhs(u)

For cada nodo n' sucesor de u:

ActualizarNodo(n')

Else

g(u) = ∞

For cada nodo n' sucesor de u:

ActualizarNodo(n')

Función AcztualizarNodo():

If u =! v

rhs(u) = min(g(n)+c(n→u)), siendo n un predecesor

Añadir u a D si no lo está ya

If u está en Q

Eliminar de la lista de prioridad Q

If g(u) =! rhs(u)

Meter u en Q con su prioridad

Función MAIN()

Inicializar()

While true:

CaminoMasCorto()

"Esperar a cambios en los costes"

Para todos los arcos entre nodos (n,n') que hayan cambiado su coste:

Actualizar coste c(n→n')

ActualizarNodo(n')

4.5 ALGORITMO D*LITE

Este algoritmo, a pesar de su nombre, no está basado en el algoritmo D*, sino en el algoritmo LPA*. Recibe este nombre ya que desarrolla el mismo comportamiento que D* pero de forma más simple y en un menor número de líneas de código, además su rendimiento es incluso mejor que el de D* o D* Enfocado. Por ello, actualmente la mayoría de sistemas actuales se basan en este algoritmo. Un ejemplo con relevancia considerable son los prototipos de sistema de navegación utilizados en los vehículos espaciales Opportunity y Spirit en Marte, ambos desarrollados en la Carnegie Mellon University [51].

Al basarse en el algoritmo LPA*, se trata de un algoritmo heurístico pero la principal diferencia es que cuando un coste cambia, no tiene en cuenta si crece o decrece o si este nodo está cerca o lejos del nodo actual. Los costes iniciales cambian a infinito cuando el robot descubre que ese nodo no se puede atravesar. Otra diferencia es que LPA* estudia los nodos desde el inicial hasta el final y, por ello, los valores *g(n)* se estiman desde la distancia al inicio, en cambio, D*Lite estudia desde el nodo final hasta el inicial y, por tanto, los valores *g(n)* se estiman en función de la distancia al nodo final. Es decir, el pseudocódigo es prácticamente el mismo que en el algoritmo LPA* pero intercambiando el nodo inicial por el nodo final y los costes entre nodos de sucesores y predecesores en sentido contrario. Además, se añade la variable k_m, que representa el cambio en el coste en un tramo, por lo que tiene que ser añadida en el primer término del valor clave de la cola de prioridad, por tanto, la prioridad queda de la siguiente forma [50]:

$$k(n) = \begin{cases} k_1(n) = \min(g(n), rhs(n) + h(n_{inicial}, n) + k_m) \\ k_2(n) = \min(g(n), rhs(n)) \end{cases}$$

A continuación, se muestra el pseudocódigo de una versión optimizada del algoritmo D*Lite:

Pseudocódigo

Inputs: *Gráfico* ***G*** *con n nodos y costes* ***c(n_i→n_j)*** *no negativos entre nodos, nodo inicial* ***v*** *y nodo final* ***w***

Output: *Lista* ***D*** *que almacena la secuencia de nodos hasta el nodo final con el menor coste posible*

Función Inicialización():

rhs(n) = g(n) = ∞ para cada n =! v

rhs(w) = 0

k_m = 0

Crear lista vacía D = 0

añadir w a D

Q = Lista de prioridad, añadir w a Q con la prioridad [h(v,w); 0]

Función CaminoMasCorto():

```
While menor prioridad de Q < prioridad de w OR  rhs(v ) !=  g(v):
        Extraer nodo con mínima prioridad de Q → u
        k_antiguo = menor prioridad de Q
        k_nuevo = prioridad de u
        If k_antiguo < k_nuevo
                Meter u dentro de Q con su prioridad
        Elif g(u) > rhs(u)
                g(u) = rhs(u)
        Quitar u de Q
                For cada nodo n predecesor de u:
                        If n != w
                                rhs(n) = min( rhs(n), c(n→u) + g(u ) )
        ActualizarNodo(n)
        Else
                g_antiguo = g(u)
        g(u) = ∞
```

For cada nodo n predecesor de u:

If rhs(n) = c(n→u) + $g_{antiguo}$ AND n != w

rhs(n) = min(rhs(n), c(n→n') + g(n')) siendo n' un succesor de n

ActualizarNodo(n)

Función AcztualizarNodo(u):

If g(u) =! rhs(u) AND u se encuentra en Q

Actualizar u y su prioridad en Q

Elif g(u) =! rhs(u) AND u no se encuentra en Q

añadir u y su prioridad en Q

Elif g(u) = rhs(u) AND u se encuentra en Q

Eliminar u de Q

Función MAIN()

$n_{antiguo} = n_{inicial}$

Inicializar()

CaminoMasCorto()

While $n_{inicial}$!= w

if g($n_{inicial}$) = ∞ no existe camino posible

$n_{inicial}$ = arg min (c($n_{inicial}$→n') + g(n'), siendo n' un sucesor de $n_{inicial}$

Moverse a $n_{inicial}$

"Escanear gráfico por si ha habido cambios en los costes"

"Si hay cambios"

km = km + h($n_{antiguo}$,n_{nuevo})

$n_{antiguo}$ = n_{nuevo}

For todos los arcos entre nodos (n,n') que hayan cambiado su coste:

Actualizar coste c(n→n')

$c_{antiguo}$ = c(n→n')

If $c_{antiguo}$ > c(n→n') AND n != w

rhs(n) = min(rhs(n), c(n→n') + g(n'))

Elif rhs(n) = c_{old} + g(n')

rhs(n) = min(rhs(n), c(n→n'') + g(n'')), siendo n'' un sucesor de n

ActualizarNodo(n)

ComputeShortestPath()

5.- PROGRAMA PRINCIPAL PARA EL CONTROL AUTÓNOMO DEL DJI TELLO

En este apartado, se encuentra el proceso que se ha seguido para realizar el código final en el lenguaje de programación Python que se ha utilizado para que el DJI Tello sea capaz de moverse autónomamente hasta un destino esquivando obstáculos detectados por el modelo entrenando con Deep Learning mientras mapea el entorno.

Los programas mencionados se encuentran por orden según cómo se han desarrollado a lo largo del proyecto. Primero, se quiso controlar la aeronave desde un ordenador, a continuación, se deseó además mapear tanto el entorno como la propia aeronave y su posición y trayectoria, después se tuvo por objetivo aplicar el modelo entrenado con Deep Learning desde la cámara del drone a tiempo real, la siguiente meta fue aplicar el algoritmo D*Lite para elaborar una trayectoria hasta un destino deseado esquivando los obstáculos conocidos y los que se descubran por el camino y, por último, con el objetivo final de este proyecto, se implementaron todos logros anteriores para conseguir una primera aproximación a una aeronave autónoma capaz de viajar de un punto a otro esquivando tanto obstáculos conocidos como desconocidos utilizando simplemente su cámara para detectarlos.

5.1 CONTROL DEL DRONE DESDE UN ORDENADOR

Como se ha mencionado anteriormente, una de las ventajas más atractivas para usar el DJI Tello es su facilidad para ser programado con fines educacionales. La aeronave utiliza un puerto Wi-Fi UDP para permitir a los usuarios controlarlo a través de comandos de texto, ya sea con un ordenador, móvil, tablet, etc. Esta forma de control es denominada modo Tello SDK o simplemente SDK. Igualmente, por tema de seguridad, si se está volando el drone y no recibe ningún comando en 15 segundos, éste aterrizará automáticamente. Existen tres tipos de comandos en este modo:

- Comandos “Control” (xxx)
    - Devuelve “ok” si el comando se ejecuta satisfactoriamente.
    - Devuelve “error” o un código informativo si es insatisfactorio.

- Comandos “Read” (xxx?)
    - Devuelve el estado en ese instante de un subparámetro (altura, batería, etc.)
- Comandos “Set” (intentarán establecer un valor en un subparámetro)
    - Devuelve “ok” si el comando se ejecuta satisfactoriamente.
    - Devuelve “error” o un código informativo si es insatisfactorio.

Los comandos que se pueden utilizar en este modo son los siguientes:

Comandos “Control”		
Comando	**Descripción**	**Respuestas posibles**
Command	Iniciar modo SDK	ok error
Takeoff	Despegar automáticamente	ok error
Land	Aterrizar automáticamente	ok error
Streamon	Habilitar cámara	ok error
Streamoff	Deshabilitar cámara	ok Error
Emergency	Detener todos los motores inmediatamente	ok error
up *x*	Vuela hacia arriba una distancia de *x* centímetros *x*: 20-500	ok error
down *x*	Vuela hacia abajo una distancia de *x* centímetros *x*: 20-500	ok error
left *x*	Vuela hacia la izquierda una distancia de *x* centímetros *x*: 20-500	ok error
right *x*	Vuela hacia la derecha una distancia de *x* centímetros *x*: 20-500	ok error

forward *x*	Vuela hacia delante una distancia de *x* centímetros *x*: 20-500	ok error
back *x*	Vuela hacia atrás una distancia de *x* centímetros *x*: 20-500	ok error
cw *x*	Gira *x* grados en sentido horario *x*: 1-3600	Ok error
ccw *x*	Gira *x* grados en sentido antihorario *x*: 1-3600	ok error
flip *x*	Da una voltereta hacia *x* l (izquierda) r (derecha) f (adelante) b (atrás)	Ok error
go *x y z speed*	Vuela a *x y z* con una velocidad *speed* (cm/s) *x*: 20-500 *y*: 20-500 *z*: 20-500 *speed*: 10-100	ok error
curve *x1 y1 z1 x2 y2 z2 speed*	Vuela haciendo una curva definida por las dos coordenadas dadas a una velocidad *speed* (cm/s) *x1, x2*: 20-500 *y1, y2*: 20-500 *z1, z2*: 20-500 *speed*: 10-60 *x/y/z* no pueden estar entre -20 y 20 al mismo tiempo	Ok error
Comandos "Set"		
Comando	**Descripción**	**Respuestas posibles**
speed *x*	Establecer velocidad a *x* cm/s *x*=10-100	ok error

rc *a b c d*	Moverse a velocidad *a b c d* *a*: izquierda/derecha (-100~100) *b*: delante/atrás (-100~100) *c*: arriba/abajo (-100~100) *d*: giro (-100~100)	ok error
wifi ssid pass	Establecer la clave SSID Wi-Fi	Ok Error
Comandos "Read"		
Comando	**Descripción**	**Respuestas posibles**
speed?	Obtener velocidad actual (cm/s)	*x*: 1-100
baterry?	Obtener porcentaje de batería actual	*x*: 0-100
time?	Obtener tiempo actual de vuelo (s)	Time
height?	Obtener altura actual (cm)	*x*: 0-3000
temp?	Obtener temperatura del drone (ºC)	*x*: 0-90
attitude?	Obtener ángulos "pitch", "roll", "yaw" de la IMU	pitch roll yaw
baro?	Obtener valor del barómetro	*X*
acceleration?	Obtener aceleración angular de la IMU (0,001g)	*x y z*
tof?	Obtener distancia del TOF (cm)	*x*: 30-1000
wifi?	Obtener Wi-Fi SNR	Snr

Tabla 3. Comandos disponibles para controlar el DJI Tello [52].

Se describe a continuación los pasos a seguir para iniciar la comunicación entre el dispositivo de control y el DJJ Tello y poder mandar los comandos.

Primero, hay conectarse por Wi-Fi asociado al Tello con el dispositivo a utilizar, en este caso un PC. Siendo la IP del Tello 192.168.10.1 y su UDP PORT:8889.

En segundo lugar, establecer un cliente UDP para mandar y recibir mensajes del drone a través del mismo puerto. A continuación, mandar el comando "command" al Tello por el puerto UDP 8889 para iniciar el modo SDK comandos y así poder utilizar cualquiera de los otros comandos. Después, para recibir el estado del Tello, habrá que iniciar un servidor UPC

en el PC y esperar al mensaje de IP 0.0.0.0 vía puerto UDP 8890. En este momento, para poder habilitar la cámara del Tello, se iniciará un servidor UDP y esperar el mensaje de IP 0.0.0.0 a través del puerto UDP 11111. Una vez recibido el mensaje, habrá que mandar el comando "streamon" [52].

Para usuarios que no estén familiarizados con esta clase de programación, el simple hecho de conectarse al DJI Tello y habilitar su cámara puede ser un obstáculo importante, ya que no entienden o no saben seguir los pasos anteriormente citados. Por ello, se han desarrollado diversas librerías en un amplio rango de lenguajes de programación para poder manejar y elaborar programas de una forma más simple e intuitiva. En este proyecto, al usar Python, se ha usado la librería o módulo "djitellopy" versión 2.3 desarrollada principalmente por Damià Fuentes y Jacob Löw, que se puede encontrar en el repositorio GitHub del primer autor nombrado [53]. Esta librería es de código abierto con licencia MIT y nos permite conectarnos al DJI Tello además de controlarlo y mandarle indicaciones con una simple línea de código utilizando una de las funciones de dicha librería, sin necesidad de entender en profundidad cómo funciona la comunicación entre el PC y el Tello.

En la siguiente tabla, se muestran las funciones más importantes utilizadas en este proyecto y una breve descripción sobre su utilidad.

Función	Descripción
me = Tello()	Conectarse al DJI Tello
me.connect()	Activar el modo SDK
me.get_battery()	Obtener porcentaje de batería actual
me.streamon()	Habilitar la cámara
me.land()	Aterrizar automáticamente
me.Takeoff()	Despegar automáticamente
me.end()	Desconectarse del DJI Tello de forma segura
me.send_rc_control(*x, y, z, yaw*)	Establecer unas velocidades lineales *x, y* y *z* (cm/s) y una velocidad angular *yaw (grados/s)*
me.rotate_clockwise(*x*)	Rotar en sentido horario *x* grados

Función	Descripción
me.rotate_counter_clockwise(*x*)	Rotar en sentido antihorario *x* grados
me.get_frame_read().frame	Obtener el frame actual de la cámara

Tabla 4. Funciones de la librería "djitellopy".

La primera línea a llamar es "me = Tello()" donde "me" es un nombre arbitrario el cual irá asociado al Tello que se usará para referenciar a qué aeronave se van a enviar los comandos. Estas funciones utilizan otras funciones secundarias para realizar la conexión entre PC y drone y poder enviar y recibir comandos de texto. Así, el usuario con una sola línea del programa de Python es capaz de realizar una comunicación satisfactoria de una forma más sencilla.

Sin embargo, es de interés en este proyecto poder controlar al DJI Tello en tiempo real en función de los estímulos que recibe el usuario. Para ello, se elaborará un programa con el cual, cuando se produzcan determinadas situaciones o factores desencadenantes, se llamará a una o varias de las funciones de la librería "djitellopy" para que el drone actúe en consecuencia. Más concretamente, se utilizarán las teclas del teclado como actuadores desencadenantes. Dependiendo que tecla el usuario presione, la aeronave realizará una acción u otra.

De igual manera, se utilizará con el DJI Tello una librería de Python para que, con solamente una función, se pueda enviar y recibir mensajes con información sobre qué tecla del teclado se ha pulsado y en qué momento. Esta librería será "pygame", también de código abierto con licencia LGPL, que tiene como finalidad principal estar enfocada en la creación de videojuegos en dos dimensiones de una manera sencilla. Los creadores de esta librería, que cuenta con millones de descargas, tienen una página web donde se puede encontrar información práctica e interesante además de tutoriales, preguntas frecuentes y explicaciones sobre posibles aplicaciones de dicha librería [54].

El módulo de esta parte se encuentra en el fichero de Python "Manual1.py" que se encuentra en el Anexo I, aunque la versión final contendrá modificaciones, como se explicarán más adelante ya que consta con diversas consideraciones para el mapeado.

En este fichero, primero se encuentra la importación de las librerías “pygame”, “djitello” y “cv2”, esta última consiste en una interface simplificada de la gran librería de código abierto “OpenCV”. Esta librería está diseñada para su uso en visión computacional, “machine learning” y procesamiento de imágenes entre otros. Actualmente, es una pieza clave en operaciones en tiempo real lo que hace que gane importancia cada vez más en los sistemas de hoy en día.

El fichero “Manual1.py” constará de 3 funciones y un “loop” principal donde se llamarán a dichas funciones. La primera función es “init()” que iniciará el módulo de “pygame” y mostrará por pantalla una pestaña negra de este módulo en la cual el usuario tendrá que estar para que cuando presione una tecla, este módulo lo perciba. Después se encuentra la función “getKey(*keyName*)”, está función tendrá como salida “*True*” o “*False*” en función de si se presiona una tecla y esta tecla coincide con “*keyName*”, si es así, devolverá “True”.

Por último, la función “getkeyboardinput()” tendrá como salida una lista de 4 valores correspondientes a la velocidad en la dirección izquierda y derecha del drone (*lr*), en la dirección delante y atrás (*fb*), en la dirección arriba y abajo (*ud*) y la velocidad angular (*yv*). También, dentro de esta función, se definirán los valores de velocidad lineal y angular con los que se quiere que se mueva el drone (*speed, aspeed*). Dependiendo de qué tecla se pulse, los valores de *fb, lr, ud* tendrán el valor positivo o negativo de *speed* (moverse en dirección positiva o negativa de los ejes *x, y, z* del DJI Tello) y el valor de *yv* obtendrá el valor positivo o negativo de *aspeed* (giro horario o antihorario). En la siguiente tabla, se muestran las teclas del teclado que llevan asociadas una acción en el drone, una breve descripción de dicha función y los valores de salida en cada caso distintos a 0 o las funciones del módulo “djitellopy” a las que llaman.

Funciones de las teclas		
Tecla	**Descripción**	**Salidas**
LEFT	Moverse a la izquierda	lr = -speed
RIGHT	Moverse a la derecha	lr = speed
UP	Moverse hacia enfrente	fb = speed

Funciones de las teclas		
Tecla	**Descripción**	**Salidas**
DOWN	Moverse hacia atrás	fb = -speed
w	Moverse hacia arriba	ud = speed
s	Moverse hacia abajo	ud = -speed
a	Girar en sentido horario	yv=aspeed
d	Girar en sentido antihorario	yv = -aspeed
l	Aterrizar automáticamente	me.land()
t	Despegar automáticamente	me.takeoff()
c	Habilitar o deshabilitar la cámara	me.streamon() me.streamoff()
p	Tomar una fotografía con la cámara	
b	Obtener porcentaje de batería	
q	Desconectarse del drone de forma segura	me.end()

Tabla 5. Funciones de las teclas en el programa "Manual1.py".

Posteriormente, los valores *lr, fb, ud* y yv serán utilizados en el "loop" principal del programa "Manual1.py". Después de definir las funciones anteriormente descritas, se llamará a la función "init()" para iniciar "pygame" y se conectará con el DJI Tello habilitando el modo SDK con las funciones ya descritas anteriormente. A continuación, se entrará al "loop" principal que funcionará hasta queel programa detecta algún error o por el drone (algo que no debería ocurrir) o cuando se presione la tecla "q" cuando se desee terminar de controlar el drone. En este "loop" se almacenarán los valores de la función "getKeyboardInput()" en la variable "vals" y, posteriormente, los valores se usarán en la función del módulo "djitellopy" "send_rc_control()" para indicar al drone las velocidades y las direcciones en las que tiene que moverse. Finalmente, se usará la función "sleep(*x*)". Esta función básica que consiste en que el programa tenga que esperar *x* segundos hasta leer la siguiente línea de código. De esta forma, el PC y el drone tendrán tiempo de mandar y recibir mensajes sin saturarse o sin perder ningún mensaje. Con un valor de *x*=0,05 es más que suficiente.

Otra ventaja de este "loop" es que, aunque no se pulse a ninguna tecla, los valores de salida *lr, fb, ud* y *yv* serán 0 los 4, pero estos mensajes se enviarán igualmente al drone, por lo que no se desconectará, como se explicó anteriormente, al pasar 15 segundos sin pulsar ninguna tecla porque sí estará recibiendo comandos durante esos 15 segundos.

Para visualizar la cámara, se habilitará desde un inicio con "me.streamon()" o si se ha desactivado podrá volver a activarse al presionar la tecla "c". Después, en el "loop" principal, se utilizará la función y "me.get_frame_read().frame" para obtener un frame individualmente en cada momento de la cámara y posteriormente, con las funciones "cv2.resize(*frame, (x,y)*)" y "cv2.imshow*('título',frame*)", se reajustarán las dimensiones del frame y se mostrará por pantalla en una pantalla con el nombre de "título" que en este caso será "DJI Tello", respectivamente. Por último, se usará "cv2.waitKey(*x*)" para introducir un retraso de *x* segundos. Si no se mete esta función, la pestaña que muestra la imagen se cerraría tan rápidamente que el usuario no tendría tiempo ni para verla por pantalla.

5.2 ESTIMACIÓN DE LA POSICIÓN Y LA ORIENTACIÓN

Debido a la carencia de sensores adecuados del DJI Tello, se utilizarán técnicas de odometría para la estimación de la posición y orientación de la aeronave en cada momento. La odometría consiste en estimar la localización relativa respecto a su posición inicial de un robot mediante algoritmos y modelos matemáticos que utilizan como datos información proveniente de los sensores de dicho robot. Estos modelos se basan normalmente en ecuaciones matemáticas no muy complejas por lo que el coste computacional suele ser más bajo que cuando se utilizan otros sistemas de posicionamiento más sofisticados. Un problema considerable de la odometría es que, al tratarse de una estimación, se produce una acumulación de errores a lo largo del tiempo. Comúnmente, se utilizan sensores laser para obtener la posición relativa del robot en función del obstáculo que detecta el láser. También, una gran corriente que ha ganado popularidad en los últimos años es la odometría visual. Ésta consiste en utilizar la información captada por una o varias cámaras del robot, ya sea forma, color o distancia. Esta tecnología surge en los años 80 tras un estudio de Hans Moravec, investigador de la Universidad de Stanford, siendo desarrolados los mayores avances de esta técnica por la NASA para su implantación en un rover para la misión en Marte

en 2004 [55]. Por supuesto, cuánto mayor número de sensores o muestras tenga el robot, con mayor precisión podrá ser obtenida la estimación.

En este caso, utilizaremos tanto la velocidad lineal como la angular y el tiempo de vuelo para estimar la posición y orientación en el plano del DJI Tello.

Es de interés que el drone solo pueda moverse en la malla virtual creada para representar el entorno, de esta manera, coincidirá en mayor medida la posición real con su nodo correspondiente en dicha malla. Por ejemplo, si partimos del origen (0, 0) y mandamos una orden al Tello de moverse a una velocidad en la dirección positiva del eje *y* de 10 cm/s por 2 segundos, una vez finalizada, nos encontraremos en el nodo (0, 2) y, por tanto, habrá avanzado 20 cm hacia delante. De igual manera, ocurriría con la velocidad angular, si rotamos con una velocidad de 45º/s por dos segundos, pasaría de tener una orientación de 0º a 90º, siendo el sentido horario el utilizado como referencia de sentido positivo. Ahora bien, en el caso que se desee mover tanto linealmente como angularmente, para hallar la posición final habrá que tener en cuenta las siguientes consideraciones:

$$\begin{cases} \varphi = \varphi_0 + \omega \cdot t \\ x = x_0 + v_y \cdot t \cdot \cos(\varphi) + v_x \cdot t \cdot \sin(\varphi) \\ y = y_0 + v_y \cdot t \cdot \sin(\varphi) + v_x \cdot t \cdot \cos(\varphi) \end{cases}$$

φ *ángulo final*	x, *coordenada x final*	v_y, *velocidad rectilínia en el eje y*
φ_0, *ángulo inicial*	x_0, *coordenada x inicial*	v_x, *velocidad rectilínia en el eje x*
ω *velocidad angular*	y, *coordenada y final*	t, *tiempo transcurrido*
	y_0, *coordenada y inicial*	

5.3 MAPEADO DEL ENTORNO

El mapeado consistirá en mostrar por pantalla una representación gráfica del entorno para poder mostrar claramente cualquier información de interés, como en qué posición se encuentra la aeronave y los obstáculos, así como la trayectoria realizada por el drone y la trayectoria a realizar hasta el destino deseado.

Para esta parte, se trabajará con el fichero de Python "Manual2.py" el cual es una versión modificada del desarrollado en el apartado 5.1. Como referencia, se utilizará la explicación del apartado anterior para estimar la posición y orientación del DJI Tello.

En este proyecto se utilizará una malla cuadrada de 500x500 como representación del entorno, es decir, una totalidad de 250.000 nodos. Esta malla abarcará una distancia real de 25,0 m en cada dirección positiva y negativa de los ejes *x* e *y* del plano. Siendo el origen, el punto central de la malla, el nodo (x=250, y=250).

El sistema de referencia del programa y el que utilice el usuario será diferente, el programa no utiliza nodos en direcciones negativas, por lo que el origen se situará en el nodo de más arriba a la izquierda siendo las direcciones positivas de los x e y hacia la derecha y hacia abajo, respectivamente. En cambio, el usuario utilizará un sistema de referencia en el que el origen (0, 0) se encuentre en el centro de la imagen y las direcciones positivas de los ejes x e y sean hacia la derecha y hacia arriba, respectivamente. Por tanto, el nodo (500, 500) en el sistema del programa consistirá en el nodo (250, -250) para el sistema del usuario, por ejemplo. En la siguiente imagen, se muestra una aclaración gráfica de lo mencionado anteriormente, donde *a)* es la referencia del programa y *b)* es la del usuario.

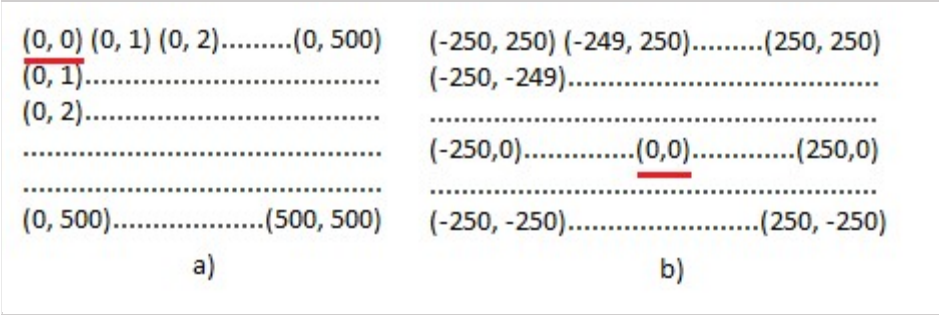


Figura 41. Sistema de referencia del ordenador (izquierda) y sistema de referencia del usuario (derecha).

Para el mapeado se utilizará la librería "NumPy" la cual es una librería fundamental de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos. Da soporte para vectores y matrices grandes multidimensionales. Concretamente, se utilizará una matriz de dimensiones 500x500x3 (x, y, color BGR). El BRG es una representación de 24 bits, los 8 bits (valores desde 0 a 255, valores de base hexadecimal) de dirección inferior son azules (Blue->B), los 8 bits siguientes son rojos(Red>R)

y los 8 bits de dirección superior son verdes (Green->G). Es decir, cada nodo llevará asociado un color, siendo el color base por defecto negro, en BGR (0,0,0).

Utilizando de igual manera que para la visualización de la cámara, se utilizará la función "cv2.imshow()" con la matriz del mapeado para mostrar por pantalla una representación del entorno en 2D en una pestaña independiente con el nombre de "Mapping".

Para dibujar tanto la localización del DJI Tello como su trayectoria se utilizará una nueva función llamada "drawpoints(*img, points, pos, yaw*)", está función utilizará como entradas la matriz utilizada como mapeado "img" y la modificará de tal forma que los puntos por lo que ha pasado el drone, es decir, la trayectoria "points", los convertirá en círculos blanco, además con la información de la posición y orientación actual "pos" y "yaw" variará algunos valores de la matriz para que cuando se muestre por pantalla, aparezca un símbolo de estrella azul a modo de representación del drone y un texto en el que se muestra la posición y orientación del drone en el plano.

Para estimar la posición de la aeronave habrá que modificar la función "getKeyboardInput()", añadirá ahora como salidas "x", "y" y "yaw", siendo éstas la estimación de la posición (pos=(x,*y*)) y la estimación del ángulo del Tello, respectivamente. La modificación se realizará con el objetivo de aplicar los principios de odometría explicados anteriormente. Primero de todo, a partir de probar el DJI Tello, se puede apreciar cómo no es realmente preciso, es decir, la velocidad que establecemos en "speed" o "aspeed" y el intervalo que vuela, el DJI Tello debería moverse una cierta distancia según la función distancia = velocidad·tiempo, sin embargo, la distancia que realmente vuela no es exactamente la misma. Esto puede ser debido a diversas razones como la necesidad de acelerar o desacelerar, tiempo que tarda el programar en dar un "loop" o simplemente la inexactitud de la aeronave. Para contrarrestar esta discordancia, se utilizarán dos parámetros diferentes para representar las velocidades lineales y angulares, "fspeed" y "fSpeed", "aspeed" y "aSpeed", siendo los primeros en cada caso, parámetros empíricos obtenidos a través de prueba y error para que la distancia teórica de "fSpeed"·"interval" ("dInterval") coincida con la distancia física real que se mueve el drone al recibir los parámetros de "fspeed" en la función "getkeyboardinput()", de igual manera con "aSpeed" y "aspeed". El valor de "interval" tendrá que ser el mismo que el utilizado en la

función “sleep(x)” después de mandar los parámetros al DJI Tello para que sea coherente la estimación.

Por tanto, para estimar la posición utilizaremos los parámetros “dInterval” y “aInterval” siendo, respectivamente, la distancia lineal teórica que se ha movido el drone y la distancia angular teórica que ha girado. Uno de estos valores se almacenarán en un nuevo parámetro de “getkeyboardinput()” llamado “d” dependiendo de si se quiere mover o rotar, además se utilizará otro nuevo parámetro llamado “a” que servirá para determinar si corresponde con la dirección positiva o negativa de los ejes del plano para el caso de traslación. Estos valores se utilizarán para actualizar los valores cartesianos de “x”, “y” y “yaw” que representan la posición y orientación del DJI Tello. En esta función se utilizarán los valores actuales del drone y se les sumará los nuevos valores en función de qué tecla ha sido pulsada para obtener la nueva posición, basándose en la técnica de odometría explicada en el anterior apartado y como se muestra a continuación.

$$\begin{cases} yaw = yaw + aInterval, -180º < yaw < 180º \\ a = a + yaw \\ x = x + d \cdot \cos(a) \\ y = y + d \cdot \sin(a) \end{cases}$$

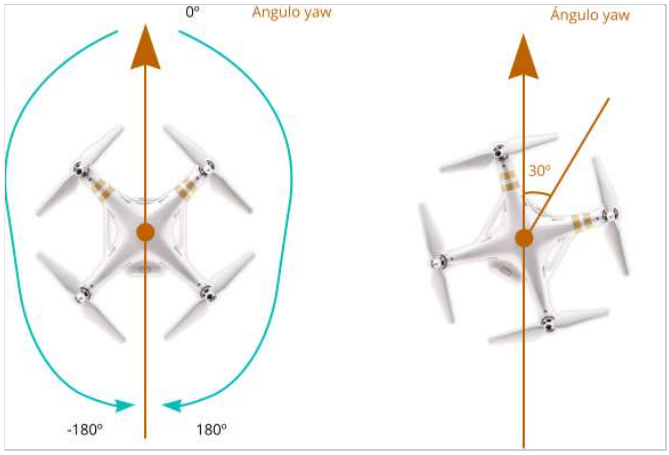


Figura 42. Referencia tomada para el valor del ángulo "yaw".

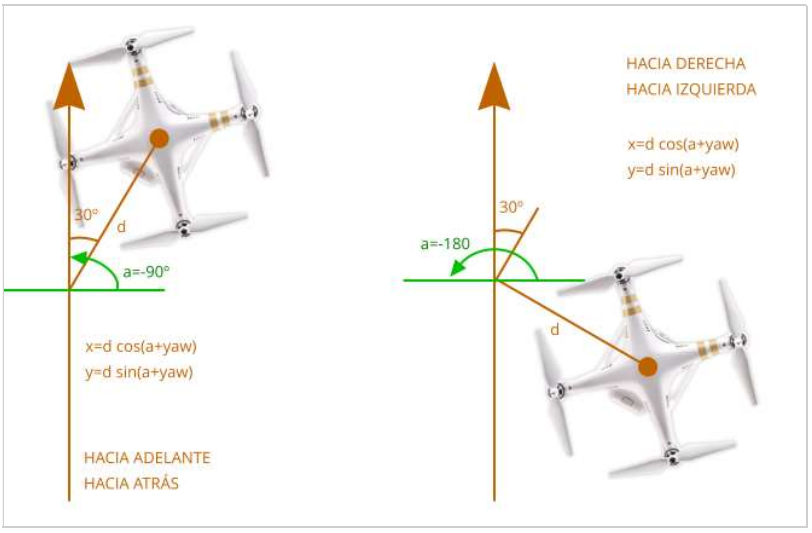


Figura 43. Valores de "a" en función de su movimiento.

Ahora que se está controlando la posición del drone, es de interés utilizar información de anteriores vuelos para continuar mapeando en función de éstos y así poder continuar con una misión inicial sin necesidad de manipular el drone de ninguna forma. Por ello, cada vez que se inicie el programa “Manual2.py” se preguntará por pantalla al usuario si desea reiniciar todos los valores o si desea conservar los anteriores. Por defecto, si se inicia por primera vez o si se desea partir desde cero, en el mapeado se nos mostrará simplemente el DJI Tello en mitad de la pestaña en la posición (0,0) y con una orientación de 0º. Sin embargo, si se utiliza información anterior a modo de continuación de la misión, en el mapeado se mostrará la posición y orientación final del proceso anterior además de la trayectoria realizada en el anterior vuelo. Esta aplicación puede ser de gran utilidad, de hecho, en este proyecto será indispensable por una problemática que se explicará posteriormente. Para esta función se guardarán archivos NPY (formato compatible con la librería “NumPy” en una carpeta llamada “datos”. Estos archivos se reescribirán cada vez que se desconecte manualmente del drone para poder guardar la última posición del DJI Tello e información del mapeado, por si se requiere continuar desde ese punto la siguiente vez que se utilice el programa.

En el video “PruebaManual2“, disponible en la carpeta comprimida “OtrosVideos” del GitHub del proyecto [1] o en el siguiente enlace[2], se muestra un ejemplo del funcionamiento de este programa.

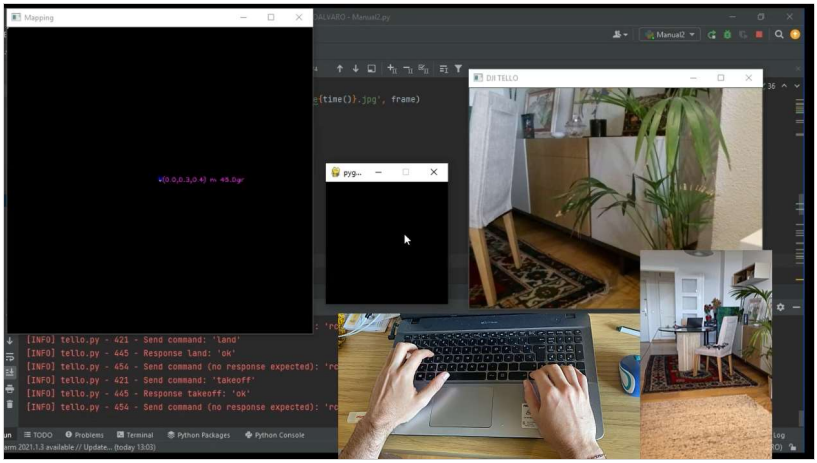


Figura 44. Fotograma del video "PruebaManual2".

5.4 DETECCIÓN DE OBSTÁCULOS

En este apartado, se explica cómo se ha implementado el modelo entrenado con Deep Learning en un programa de Python capaz de recibir las imágenes de la cámara del DJI Tello en tiempo real y en éstas detectar y clasificar las diferentes clases de obstáculos.

La programación del módulo que permite la detección de obstáculos se incluye en el fichero “ReconocimientoObjetos.py”. La parte correspondiente al uso del modelo toma como base el fichero mencionado en el apartado 3.2 “Object_detection_webcam.py”, cuya función es aplicar el modelo entrenado a través de las imágenes recibidas en tiempo real por la cámara web del ordenador. Se modificará este código para que en vez utilizar las imágenes de la cámara web se conecte al DJI Tello y reciba las imágenes de su cámara.

Este programa, utiliza dos ficheros Python proporcionados por TensorFlow que se encuentran en la carpeta “utils” dentro de la carpeta “object_detection”. Estos ficheros son “label_map_util.py” y “visualization_utils.py”, y contienen funciones que facilitarán la

[2] https://unicancloud-my.sharepoint.com/:v:/g/personal/daa770_alumnos_unican_es/ETmtORqzFddLnruiYKVlSz8BWQtAd-FPSfuihjzEsDGisw?e=oNn4Jd

importación y el uso de las distintas clases y la representación gráfica por pantalla de los obstáculos reconocidos, sus correspondientes recuadros y probabilidad como se explica a continuación.

En la línea 103 del código de "ReconocimientoObjetos.py" se encuentra la variable "min_score_thresh" la cual nos indica la probabilidad mínima que se necesita para mostrar los recuadros que engloban los objetos por pantalla. Es decir, el propio modelo establece diferentes obstáculos posibles y les asocia una probabilidad que indica cómo de seguro está de esa clasificación. En el proyecto se ha establecido una probabilidad de 0,85 para que solo muestre los obstáculos identificados con mayor certeza.

Para este proyecto, para un correcto mapeado es importante conocer qué obstáculo se encuentra delante de la aeronave. En esta parte simplemente se mostrará por pantalla qué tipo de obstáculo detecta delante de la aeronave a través del "id" que le corresponde. Por sencillez, optimización del tiempo en procesar el código y dimensiones del drone, se va a considerar como obstáculo simplemente aquel que se encuentre justo enfrente del drone a la misma altura, es decir, justo en el centro de la imagen. Para saber que se encuentra en esa posición se ha modificado la función "visualize_boxes_and_labels_on_image_array(...)" perteneciente a "visualization_utils.py". Esta función es la que principalmente permite que al visualizarse la imagen de la cámara del DJI Tello por pantalla, se muestren tanto los recuadros que envuelven los obstáculos, su clase y su probabilidad. En esta función, al definir "min_score_thresh", se establece que solo aparezcan aquellos obstáculos detectado por el modelo con una probabilidad superior a éste valor. Como simplemente se requiere estudiar si estos se encuentran delante del drone o no, en la sección del código (a partir de línea 1191 de "visualization_utils.py") donde se verifica que la probabilidad es mayor se añade una condición nueva.

Siendo "xmin", "xmax", "ymin" e "ymax" los límites mínimos y máximos del eje x y del eje y del límite de la caja o recuadro que engloba el obstáculo, la condición se cumplirá cuando se cumplan las siguientes condiciones a la vez: "xmin" se encuentra a la izquierda del centro de la imagen, "xmax" se encuentra a la derecha del centro, "ymax" se encuentra arriba del centro y "ymin" por debajo del centro. De esta forma solo identificará aquellos obstáculos que tengan alguna parte justo delante del DJI Tello, es decir, en el centro de la imagen.

El “id” del obstáculo, el número identificativo de cada clase que se encuentra delante al cumplir está condición se almacenará en la variable “enfrente” que se añadirá como salida de esta función para poder utilizarlo posteriormente. Como puede ocurrir que haya más de un obstáculo delante que cumpla la condición, la variable “enfrente” será de tipo “lista” para almacenar todas las clases distintas de obstáculos que se detectan.

En el video “PruebaReconocimientoObjetos“, disponible en “VideosPruebas” en el GitHub del proyecto [1] o en el siguiente enlace[3], se muestra un ejemplo del funcionamiento de este programa, las imágenes provienen de la cámara del DJI Tello movido manualmente, no mientras vuela.

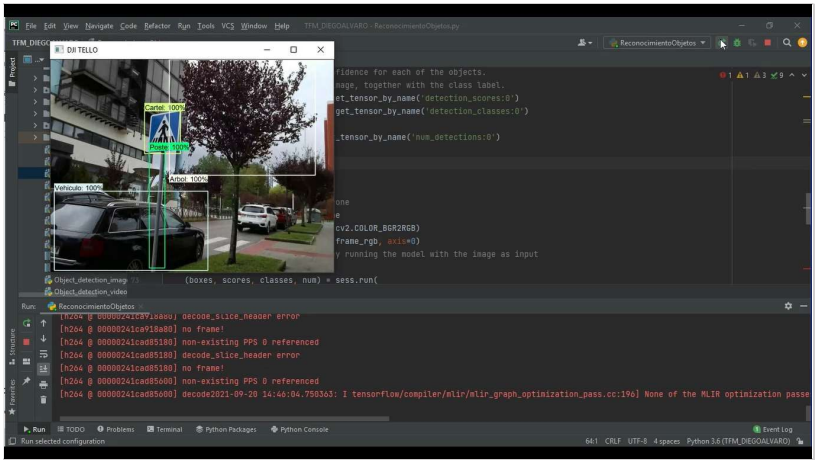


Figura 45. Fotograma del video "PruebaReconomientoObjetos".

Como se aprecia en el video, la retransmisión por cámara no es continua. Esto es debido a que todas las operaciones se realizan en la misma iteración del bucle y tiene que esperar hasta la siguiente iteración para enseñar el siguiente frame y, como el poder computacional no es muy potente, tarda un tiempo notable en realizar los cálculos de detección de objetos en cada iteración.

[3] https://unicancloud-my.sharepoint.com/:v:/g/personal/daa770_alumnos_unican_es/Eciwe27V0lxOo6mY29cy5sABpJAJWE3XFLltb0QmPl-_4w

5.5 PROGRAMA QUE REALICE EL ALGORITMO D*LITE

Para esta tarea se utilizará el programa “PruebaDstarlite.py” cuyo objetivo es mostrar por pantalla el proceso de funcionamiento del algoritmo D*Lite para poder aplicarlo posteriormente al DJI Tello. Se introducirán unas coordenadas de destino y se representará lo que sería el drone y cómo va avanzando, además de la trayectoria hasta el destino y cómo varia al encontrarse un obstáculo, representado en naranja. Este programa se ha basado en uno con un objetivo similar que se puede encontrar en un repositorio GitHub [56] de Kristoffer Rakstad, ingeniero de software.

El funcionamiento de “PruebaDstarlite.py” consiste en utilizar una matriz (“map”) que represente los pixeles de la malla y su valor, 255 si son obstáculos y 0 si están vacías.

Primero, se preguntará por el destino deseado. Después con la clase “DStarLite” se establecerán el tamaño de la malla, así como el origen y destino de la trayectoria. A continuación, realizará un bucle donde se usará la función “dstar.move_and_replan(…)” para indicar en qué punto de la malla se encuentra y así recalcular la trayectoria hasta el destino. Los nuevos obstáculos se habrán identificado anteriormente con la función “slam.rescan(…)” que identificará un obstáculo nuevo si se encuentra igual o menor número de nodos que un parámetro prestablecido llamado “view_range”. Este proceso se repetirá hasta llegar al destino deseado.

En el video “PruebaDStarLite”, que se puede encontrar en el repositorio GitHub del proyecto [1] o en el siguiente enlace[4], se muestra un par de ejemplos al ejecutar el programa “PruebaDstarlite.py” con unos obstáculos (en naranja) que son desconocidos para el algoritmo D*Lite en un inicio.

[4] https://unicancloud-my.sharepoint.com/:v:/g/personal/daa770_alumnos_unican_es/Edgoc9rSG31Lijz95rdyKcwBmwwZIcB_0JIKtOopWM4Jag?e=HYB8JB

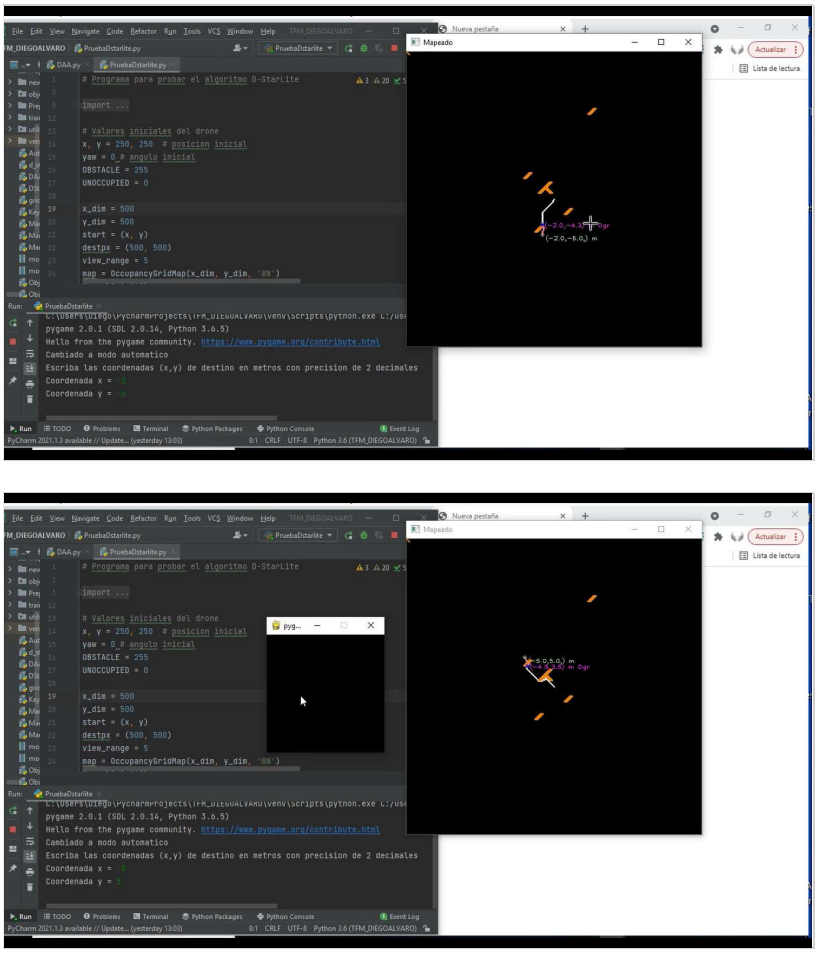


Figura 46. Fotogramas del video "PruebaDStarLite".

5.6 SEGUIMIENTO DE LA TRAYECTORIA

Para que el DJI Tello sea capaz de moverse autónomamente debe poder seguir una trayectoria dada sin necesidad de recibir comandos de un operador. Como se explicó anteriormente, la salida del algoritmo D*Lite será una lista con las coordenadas a seguir desde un punto a su destino. Por tanto, es necesario que el DJI Tello sea capaz de seguir una trayectoria representada ésta con una lista de puntos por los que tiene que pasar. Para realizar esta tarea, se desarrolló el programa “PruebaSeguirTrayectoria.py”.

Este programa contará con las funciones ya explicadas para controlar el drone de la librería “djitellopy” además de las funciones de “Manual0.py” para utilizar el teclado durante el código. Además, contará con dos funciones nuevas “signo(*num*)” y “angulodegiro(*pos0,pos1,posref*)”. La primera de ellas es una función simple que se utilizará

dentro de la segunda. La primera función devolverá un valor de 1 si el signo de *num* es positivo y un valor de 0 en el resto de los casos. La segunda función devolverá el ángulo (entre 0 y 180º) que forma el vector de *pos0* a *posref* con el vector de *posref* a *pos1*, también devolverá el sentido en el que hay que girar el primer vector para llegar al segundo. De esta forma, se podrá implementar en la función "me.send_rc_control(…)" para que gire dicho ángulo y en el sentido correcto cuando sea necesario. En la siguiente imagen se muestra las comprobaciones que hace para calcular el ángulo a girar (angle1) y en función de que el ángulo (angle2) sea mayor o menor de 90º en qué sentido girar.

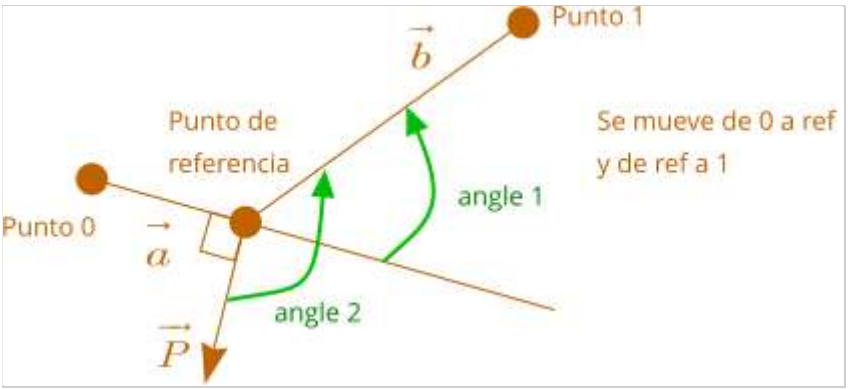


Figura 47. Ángulos que calcula la función "angulodegiro(...)".

El programa "PruebaSeguirTrayectoria.py" consiste, de manera resumida, en introducir una trayectoria "path" y utilizar un bucle para que en cada iteración avance en la posición de la trayectoria. En cada iteración, calculará el ángulo que tiene que girar y en qué sentido para ir desde la posición en la que se encuentra a la siguiente posición. Después girará utilizando "me.send_rc_control(…)" para orientarse hacia la siguiente posición y avanzará hasta ella utilizando nuevamente "me.send_rc_control(…)". La función "angulodegiro(…)" necesita tres parámetros para funcionar, por lo que cuando se encuentre en el anteúltimo punto de la trayectoria no podría calcular en ángulo para llegar al destino final. Por ello, durante la iteración y una vez se haya movido la aeronave, se guardará la posición anterior de esta en el parámetro "*lastpos*" y será el que se utilice dentro de la función como entrada "*pos0*".

En el video "PruebaSeguimientoTrayectoria", que se puede encontrar en el repositorio GitHub del proyecto [1] o en el siguiente enlace[5], se muestra un ejemplo de este programa en funcionamiento. La trayectoria a seguir se muestra en la siguiente imagen.

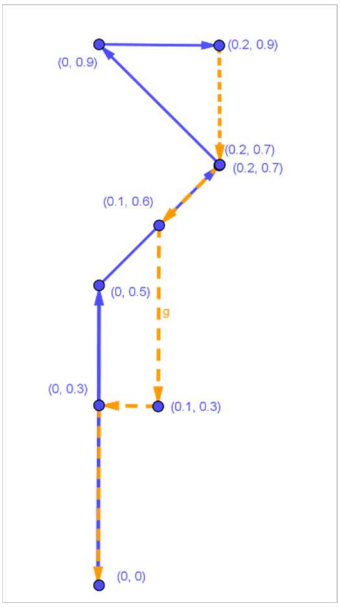


Figura 48. Trayectoria realizada en el video "PruebaSeguimientoTrayectoria" por el DJI Tello.

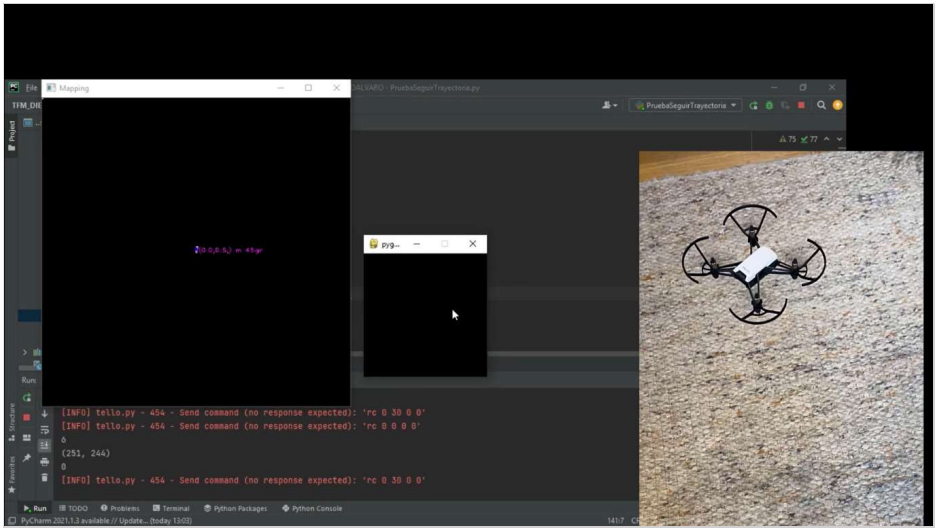


Figura 49. Fotograma del video "PruebaSeguimientoTrayectoria" después de girar 45º en el punto (0, 0,5).

[5] https://unicancloud-my.sharepoint.com/:v:/g/personal/daa770_alumnos_unican_es/EScwxQ71uwpKokFk2FMy1aMBFhmiToZ3IyTIZ3iqGb0uJA

5.7 IMPLEMENTACIÓN CONJUNTA

En este apartado se explica el código final utilizado para la realización de este proyecto. Este código englobará varios programas de Python que realizan conjuntamente todas las funciones explicadas en los apartados anteriores. Estos programas de Python serán “DAA.py”, “Manual0.py” y “d_star_lite”.

El fichero de Python “DAA.py” es el código principal, en este se llevarán a cabo tanto el control del DJI Tello, el reconocimiento de obstáculos y el mapeado. En este programa se podrá elegir cómo controlar el drone, teniendo la posibilidad de hacerlo manualmente a través de las teclas del teclado o de forma autónoma introduciéndole unas coordenadas de destino. Para controlarlo manualmente y estimar la posición de la aeronave, se utilizará el programa “Manual0.py” una variación del programa anteriormente “Manual1.py”. La variación básicamente consiste en que el control para mover la aeronave se realizará desde “DAA.py” así como mostrar por pantalla todo lo relacionado con el mapeado. En el fichero “Manual0.py” únicamente contendrá las funciones para habilitar el control con las teclas y los cálculos necesarios para obtener los parámetros a utilizar en las funciones de la librería “djitellopy” y para estimar la posición y orientación del DJI Tello en cada momento cuando se encuentre controlado manualmente.

Además, en “DAA.py” se encontrará prácticamente todo el código de “ReconocimientoObjetos.py” para poder realizar la tarea de detección de obstáculos y mostrarlos por pantalla.

En “DAA.py” se añadirá una nueva función para dibujar los obstáculos, esta función es “drawobstacles(…)” basada en los mismos principios que “drawpoints(…)”. En la siguiente tabla se muestran cómo se van a mapear cada tipo de obstáculo.

Mapeado de obstáculos		
Clase	**Forma**	**Color**
Poste	Cuadrado	Azul claro
Valla	Cuadrado	Amarillo
Cartel	Cruz	Morado
Arbol	Triangulo punta hacia arriba	Verde
Vehiculo	Cuadrado	Rojo
Semaforo	Cruz	Gris
Persona	Triangulo punta hacia abajo	Rosa
Muro	Rombo	Naranja
Contorno obstáculo	Punto	Amarillo claro

Tabla 6. Representación de cada obstáculo en el mapeado.

La diferencia con lo indicado anteriormente, es que ahora habrá que posicionar los obstáculos tanto en el mapeado como en el algoritmo D*Lite. Como se mencionó en el apartado 1.3, el DJI Tello no cuenta con ningún sensor de distancia por lo que no se pueden estimar realmente a la distancia que se encuentra cada obstáculo respecto al drone. Entonces a la hora de estimar la posición de un obstáculo, se considerará que, cuando sea detectado, se encuentre a una distancia de dos metros de la aeronave en la dirección en la que se encuentra orientada en ese momento. Además, por precaución, se considerará que el obstáculo tiene unas dimensiones de 30x30 cm, para dar margen entre la trayectoria del DJI Tello y la posición de ese obstáculo. Por esta razón, como se explicó en el apartado 3.2, en el modelo de detección de obstáculos solo se identificaron como tal aquellos que se encontraban próximos, en torno a una distancia de 2 metros. Para que así se ajusté en lo máximo posible la distancia máxima de detección del modelo con la distancia a la que se va a estimar la posición del obstáculo en el mapeado.

El funcionamiento del programa “DAA.py” consistirá básicamente en una primera parte de iniciación de módulos y valores iniciales (posición, obstáculos, etc.), después se conectará al DJI Tello y a continuación entrará en un bucle. En este bucle, habrá dos ramas y una parte

conjunta a ambas. Las ramas consistirán en si se quiere controlar el drone de forma manual con las teclas del teclado de forma similar a “Manual0.py” y el resto de sus versiones, o si se desea que la aeronave se mueva autónomamente utilizando los conceptos de “PruebaSeguirTrayectoria.py” y “PruebaDstarlite.py”. Se podrá cambiar entre los modos durante el vuelo mediante la tecla del teclado “m”. En la parte conjunta del bucle, se desarrollará el reconocimiento de obstáculos de igual manera que en “ReconocimientoObjetos.py”. También, se desarrollará el mapeado con las funciones “drawpoints(…)” y “drawobstacles(…)”. Además, utilizando la función “getKey(…)” habrá distintas acciones en función de la tecla que se pulse, como por ejemplo, hacer fotografías, guardar el mapeado, cambiar el modo de vuelo, etc.

En el siguiente apartado se mostrará en mayor detalle el funcionamiento de todos los programas envueltos en este proyecto, así como sus funciones principales y parámetros.

5.8 RESUMEN

En los siguientes gráficos se muestran las principales funciones y parámetros que se utilizan en cada programa así como diagramas de flujo para esclarecer el proceso en la función “getkeyboardinput()” y el programa “DAA.py"

DAA.py

Funciones:

convdist(posicion, modo):
"Dependiendo del modo convierte unas coordenads (posicion) en metros para el sistema de referencia del usuario a pixeles para el sistema de referencia del ordenador, o viceversa"

signo(num):
"Devuelve True si el signo de num es positivo. En otro caso, devuelve False"

angulodegiro(pos0, pos1, posref):
"Devuelve el ángulo angle y el sentido de giro (sentido) entre el vector pos0 a posref con el vector posref a pos1"

drawobstacles(img, obstaculos):
"Dibuja en el mapeado (img) los obstáculos en función de a que clase pertenezca además de un contorno en cada uno de margen de error"

drawpoints(img, points, pos, angulo, modo):
"Dibuja en el mapeado (img) la trayectoria (points) además del drone y sus coordenadas en su posición (pos) y orientación (angulo) actual. También dibuja el destino y sus coordenadas. Además si el modo se encuentra en automático, también dibuja la trayectoria a seguir hasta el destino (path)"

visualize_boxes_and_labels_on_image_array(image, ...):
"Dibuja en la imagén de la cámara(img) los recuadros que envuelven los obstáculos detectados así como la probabilidad que estima para cada uno. Además, devuelve una lista, enfrente, con los "id" que pertenecen a los obstáculos que se encuentren en mitad de la imagen"

D*Lite: d_star_lite.py, grid.py, utils_d_star.py

Funciones:

OccupancyGridMap (xdim, ydim):
"Establece las dimensiones de la malla que representa el mapeado en función de xdim y ydim"

SLAM (map, view range):
"Establece en la malla (map) a cuantos nodos (view_range) se detecte un obstáculo para recalcular la ruta"

DStarLite(map, s_start, s_goal):
"Establece en la malla (map) el nodo inicial (s_start) y el nodo objetivo (s_goal)"

move_and_replan(robot_position):
"En función de la posición del robot (robot_position) calcula y da como salida los valores de la trayectoria (path), y los valores de g y rhs para cada nodo de acuerdo a las características de map"

set_obstacle(pos):
"Establece un obstáculo en la posición pos de la malla"

rescan(pos):
"Evalúa si se encuentra un obstáculo dentro del view_range para la posición pos y actualiza el mapa map donde se encuentra el obstáculo"

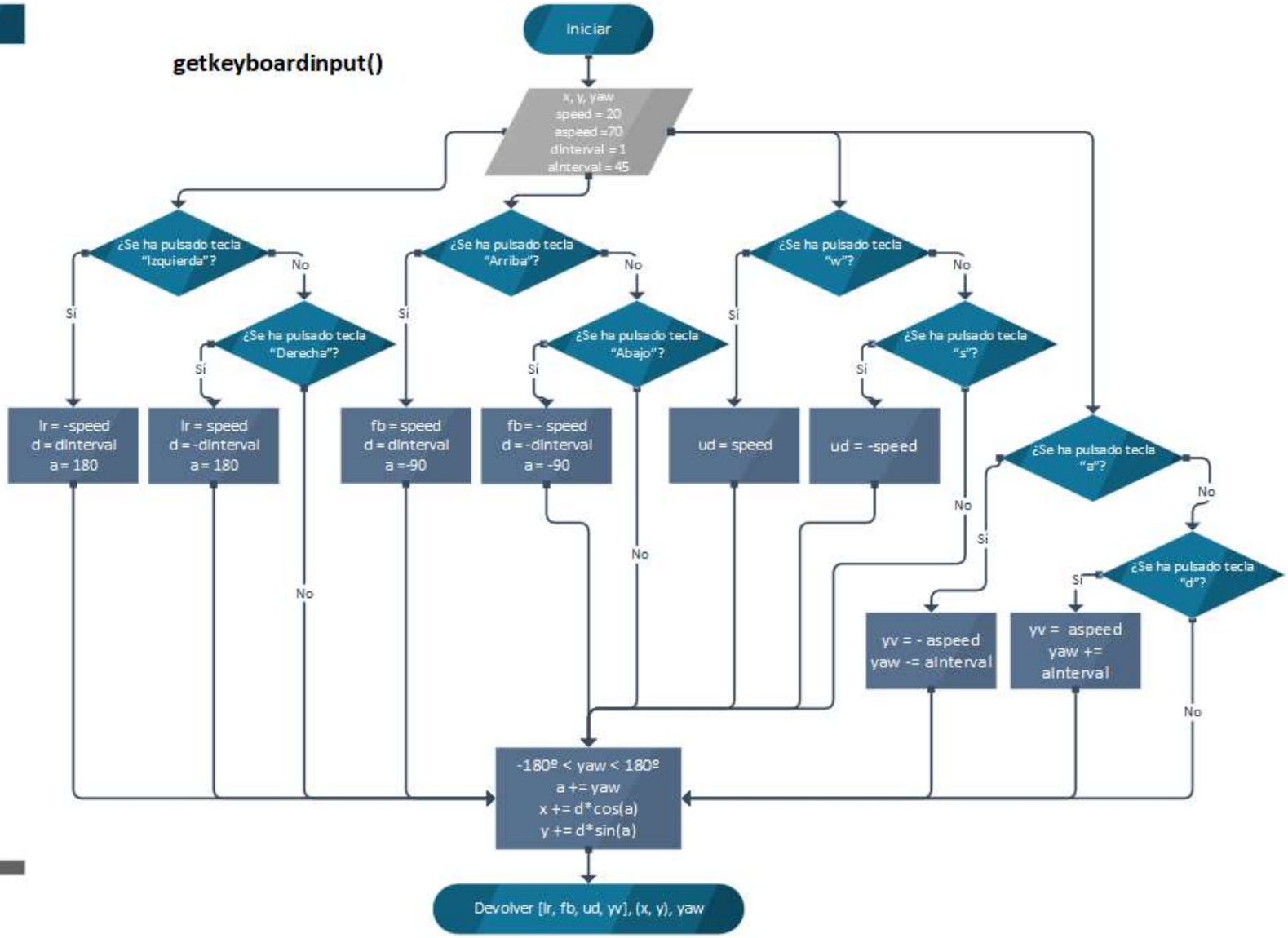
Manual0.py

Funciones:

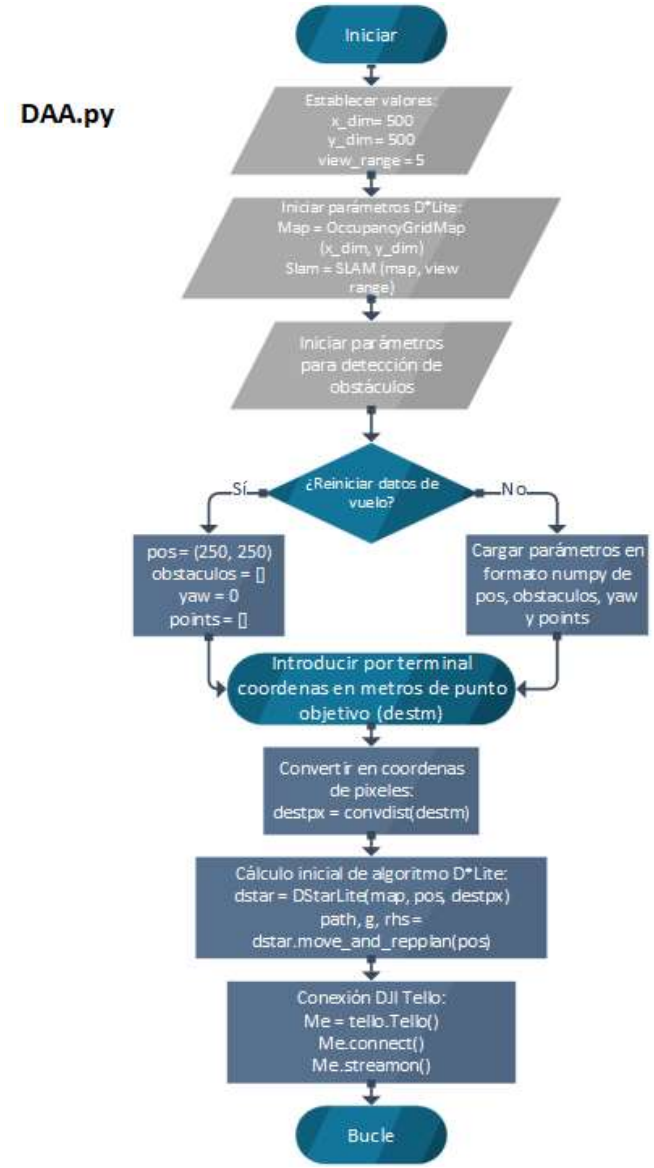
Init():
"Inicia el modulo Pygame"

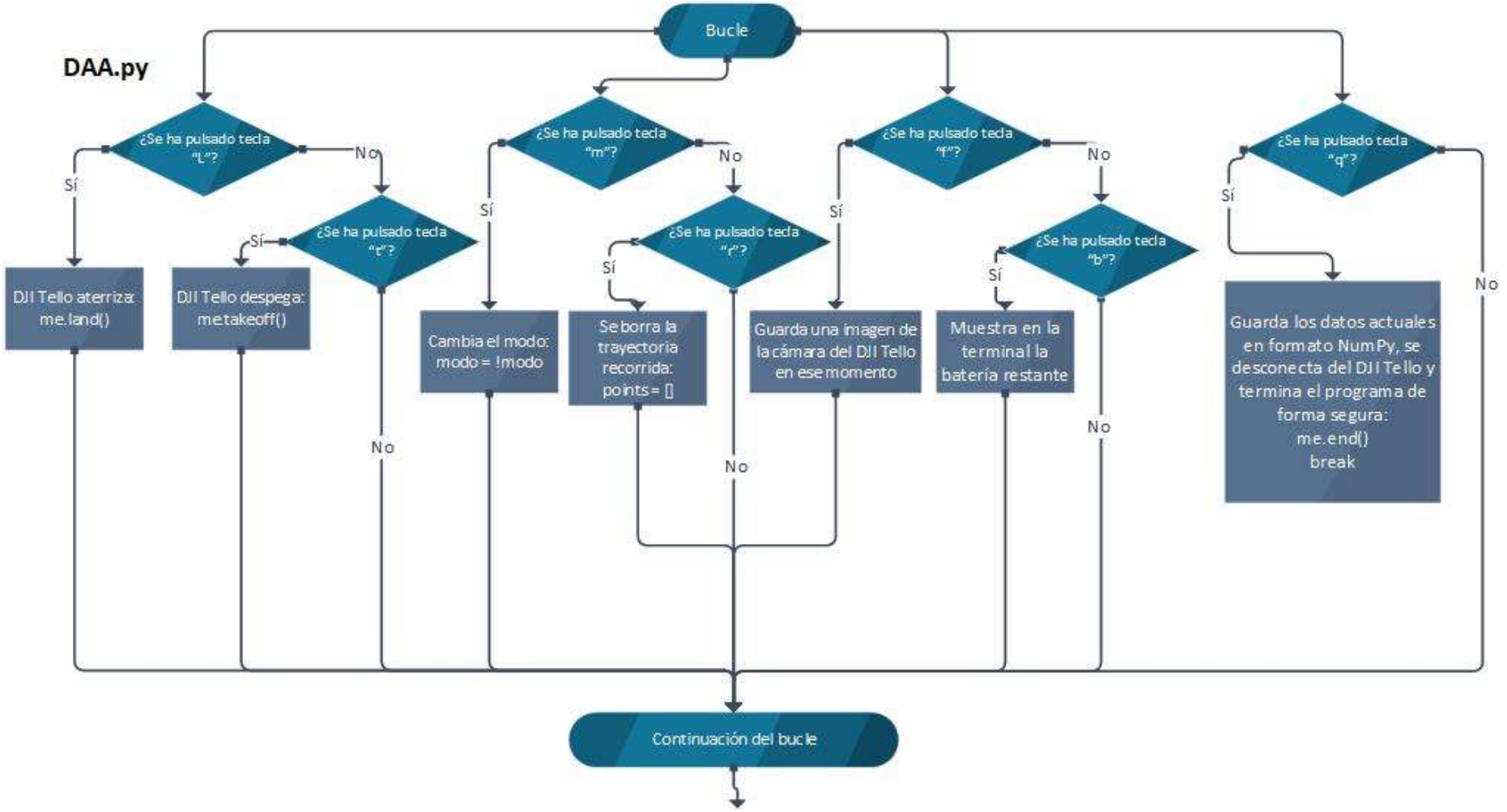
getKey(keyName):
"Devuelve True si se pulsa la tecla keyName. En otro caso, devuelve False"

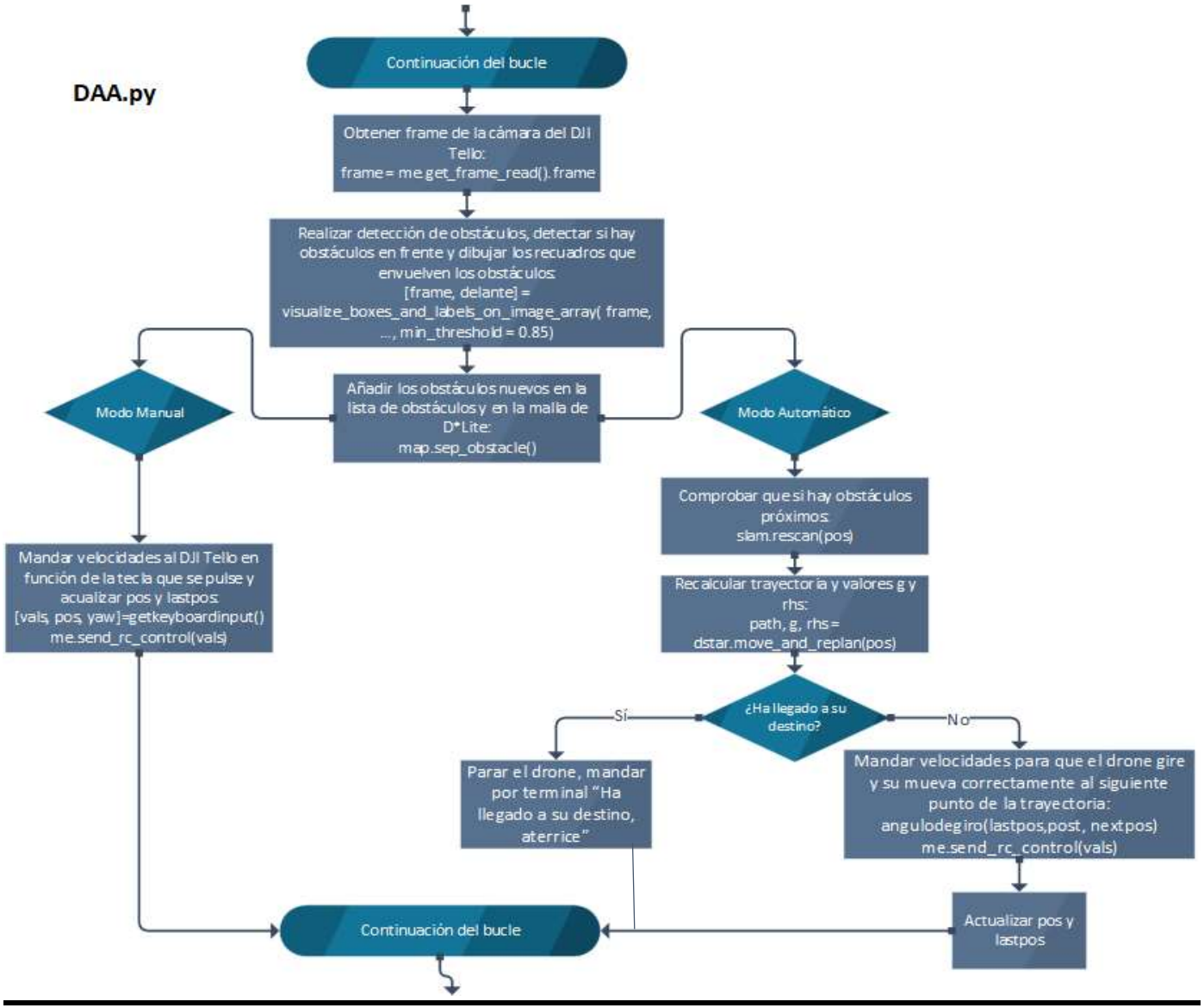
getkeyboardinput():
"Función principal de este fichero que se encarga del control del DJI Tello y el mapeado cuando se encuentra en modo manual"
Salidas:
[lr, fb, ud, yv], (x,y), yaw
[lr, fb, ud, yv] = velocidades izquierda-derecha, delante-atrás, arriba-abajo y angular
(x, y) = Nueva posición del DJI Tello al moverse
yaw = Nueva orientación del DJI Tello al moverse

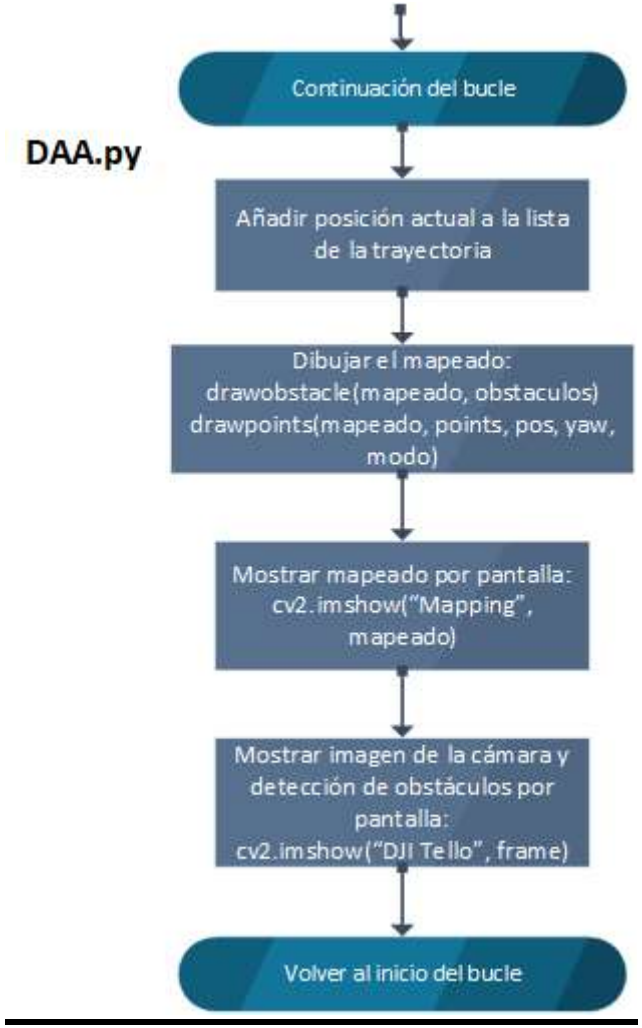


DAA.py


Iniciar
Establecer valores:
x_dim= 500
y_dim= 500
view_range = 5
Iniciar parametros D*Lite:
Map = OccupancyGridMap
(x_dim, y_dim)
Slam = SLAM (map, view
range)
Iniciar parámetros
para detección de
obstáculos
¿Reiniciar datos de
vuelo?
Sí
No
pos = (250, 250)
obstaculos = []
yaw = 0
points = []
Cargar parámetros en
formato numpy de
pos, obstaculos, yaw
y points
Introducir por terminal
coordenas en metros de punto
objetivo (destm)
Convertir en coordenas
de pixeles:
destpx = convdist(destm)
Cálculo inicial de algoritmo D*Lite:
dstar = DStarLite(map, pos, destpx)
path, g, rhs =
dstar.move_and_replan(pos)
Conexión DJI Tello:
Me = tello.Tello()
Me.connect()
Me.streamon()
Bucle


DAA.py
Bucle
¿Se ha pulsado tecla "L"?
Sí
DJI Tello aterriza: me.land()
No
¿Se ha pulsado tecla "t"?
Sí
DJI Tello despega: me.takeoff()
No
¿Se ha pulsado tecla "m"?
Sí
Cambia el modo: modo = !modo
No
¿Se ha pulsado tecla "r"?
Sí
Se borra la trayectoria recorrida: points = []
No
¿Se ha pulsado tecla "f"?
Sí
Guarda una imagen de la cámara del DJI Tello en ese momento
No
¿Se ha pulsado tecla "b"?
Sí
Muestra en la terminal la batería restante
No
¿Se ha pulsado tecla "q"?
Sí
Guarda los datos actuales en formato NumPy, se desconecta del DJI Tello y termina el programa de forma segura: me.end() break
No
Continuación del bucle


DAA.py
Continuación del bucle
Obtener frame de la cámara del DJI Tello:
frame = me.get_frame_read().frame
Realizar detección de obstáculos, detectar si hay obstáculos en frente y dibujar los recuadros que envuelven los obstáculos:
[frame, delante] = visualize_boxes_and_labels_on_image_array(frame, ..., min_threshold = 0.85)
Modo Manual
Añadir los obstáculos nuevos en la lista de obstáculos y en la malla de D*Lite:
map.sep_obstacle()
Modo Automático
Mandar velocidades al DJI Tello en función de la tecla que se pulse y acualizar pos y lastpos:
[vals, pos, yaw]=getkeyboardinput()
me.send_rc_control(vals)
Comprobar que si hay obstáculos próximos:
slam.rescan(pos)
Recalcular trayectoria y valores g y rhs:
path, g, rhs = dstar.move_and_replan(pos)
¿Ha llegado a su destino?
Sí
No
Parar el drone, mandar por terminal "Ha llegado a su destino, aterrice"
Mandar velocidades para que el drone gire y su mueva correctamente al siguiente punto de la trayectoria:
angulodegiro(lastpos,post, nextpos)
me.send_rc_control(vals)
Actualizar pos y lastpos
Continuación del bucle


DAA.py
Continuación del bucle
Añadir posición actual a la lista de la trayectoria
Dibujar el mapeado:
drawobstacle(mapeado, obstaculos)
drawpoints(mapeado, points, pos, yaw, modo)
Mostrar mapeado por pantalla:
cv2.imshow("Mapping", mapeado)
Mostrar imagen de la cámara y detección de obstáculos por pantalla:
cv2.imshow("DJI Tello", frame)
Volver al inicio del bucle

6.- PRUEBAS DE EVALUACIÓN DEL FUNCIONAMIENTO

Para evaluar el funcionamiento de la evasión de obstáculos y la planificación de trayectorias, se realizarán múltiples pruebas en tres localizaciones distintas a lo largo de varios días, estas localizaciones son el Parque Atlántico de Las Llamas, el parking del Centro Comercial Valle Real y el Parque De La Vaca, todas ellas en la localidad de Santander. Para el correcto funcionamiento del DJI Tello se tuvo en consideración que durante las pruebas las condiciones climatológicas fueran idóneas, sin lluvia ni viento.

Las pruebas consistirán en realizar un vuelo hasta un punto destino, desde despegue hasta aterrizaje, y después, volver al punto de salida otra vez. No se podrá realizar la ida y vuelta sin aterrizar y volver a ejecutar el programa principal, porque al realizar las operaciones del algoritmo D*Lite al inicio del vuelo de vuelta se perdería la señal con el DJI Tello. En estas pruebas se estudiará el número de vuelos realizados con existo, causa en caso contrario, error entre posición final deseada y posición final alcanzada y número de obstáculos esquivados.

Para realizar las pruebas, en la mayoría de los casos, se elegía aleatoriamente el punto de destino para que así hubiera gran variedad de distancias y se obtuvieran resultados representativos. Entonces, antes de iniciar el vuelo, se colocaba el DJI Tello en un punto de origen donde hasta ese destino hubiera al menos un obstáculo. Además, en los vuelos de vuelta como la trayectoria esquivaba desde un inicio los obstáculos encontrados en la ida, se utilizaba una persona para añadir al menos un obstáculo a la vuelta.

En la siguiente tabla se muestra el número de pruebas que se han realizado por localización.

Número de Pruebas	
Centro Comercial Valle Real	7
Parque Atlántico Las Llamas	6
Parque De La Vaca	9
Total	22

Tabla 7. Número de pruebas realizadas por localización.

7.- RESULTADOS DE LAS PRUEBAS

En este apartado se van a exponer los resultados y se sacarán conclusiones en función de estos. Los resultados se pueden encontrar desglosados por cada vuelo en el Anexo III Resultados de las pruebas y en el Excel "ResultadosPruebas" que se encuentra en la carpeta "Extras" del GitHub del proyecto [1].

Primero se comenzará estudiando el porcentaje de éxito de los vuelos y las causas en los casos contrarios. En la siguiente tabla se muestra cuántos vuelos fueron satisfactorios y cuántos insatisfactorios por localización y en la totalidad de los vuelos del estudio del proyecto.

Vuelos Satisfactorios/insatisfactorios	
Centro Comercial Valle Real	7/2
Parque Atlántico Las Llamas	6/2
Parque De La Vaca	9/2
Total	16/6

Tabla 8. Número de vuelos satisfactorios/insatisfactorios.

En porcentajes se obtienen los resultados siguientes, expresados en la Figura 50.

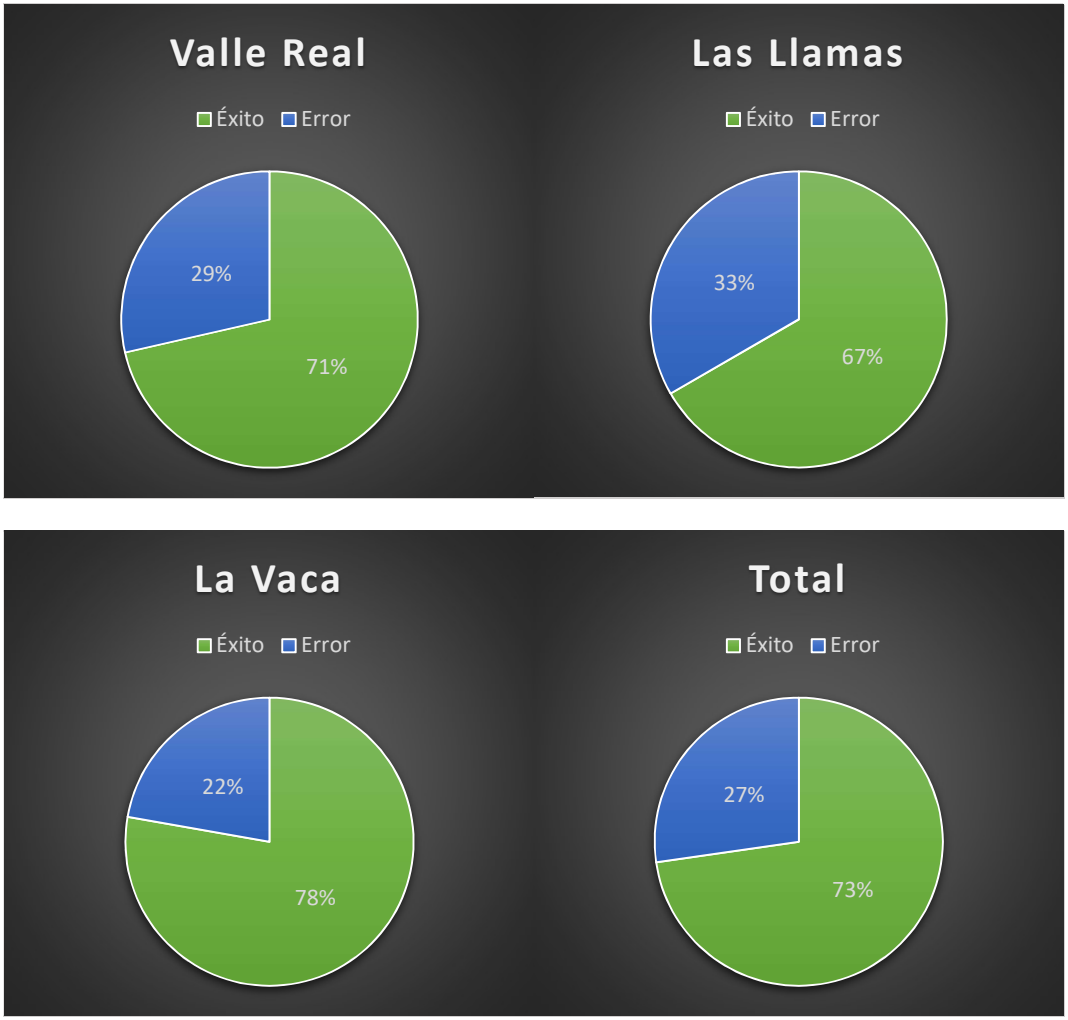


Figura 50. Porcentaje de éxito en los vuelos de las pruebas.

Como se puede apreciar, aproximadamente 3 de cada 4 vuelos consiguen llegar a un destino y volver al punto de salida, es decir, realizar un vuelo completo.

A modo de análisis en mayor profundidad, las diferentes causas de vuelos no completados se muestran a continuación:

- Valle Real
    - Prueba nº6: Se perdió el control del DJI Tello porque había demasiado viento.
    - Prueba nº7: La batería del DJI Tello se agotó en mitad de la prueba.

- Las Llamas
    - Prueba nº4: El programa dio error ya que el punto de destino coincidió con la posición de un obstáculo.
    - Prueba nº5: Se chocó con la copa de un árbol. Identificó que había un árbol, pero el ancho de la copa era mayor que el estimado por lo que el nodo de la malla que correspondía no estaba contado como obstáculo.
- La Vaca
    - Prueba nº2: Se perdió la conexión con el DJI Tello ya que tomó demasiado tiempo en recalcular la ruta al encontrarse con un obstáculo.
    - Prueba nº6: Se chocó con un obstáculo que identificó correctamente, pero, al iniciarse el vuelo, éste se encontraba muy próximo y, por tanto, se estimó erróneamente su posición.

A partir de estos resultados, podemos concluir que, en líneas generales, es una primera aproximación satisfactoria ya que el DJI Tello es capaz de completar aproximadamente el 75% de los vuelos, un porcentaje próximo al que se deseaba obtener al realizar este proyecto. Además, la mayoría de las causas por las que no es capaz están relacionadas con las limitaciones del DJI Tello, no con los conceptos generales en los que están basados tanto el modelo de detección de objetos como el de planificación de trayectorias. Esto se desarrollará con mayor detalle en el capítulo 8.

Durante la elaboración de estas pruebas, se pudo percibir cómo influenciaba la distancia al destino y cuánto tiempo tardaba en recalcular la ruta cuando se encontraba un obstáculo al principio del recorrido. Si la distancia era lejana pero el primer obstáculo que se encontraba estaba próximo al destino, no había ningún problema, en cambio, si el obstáculo se encontraba al principio de la trayectoria tardaba una considerable cantidad de tiempo en recalcular la nueva trayectoria llegando incluso a perder la conexión con el DJI Tello porque no se le han enviado comandos en más de 15 segundos. Esto se verificó posteriormente al realizar pruebas simuladas con el programa “PruebaDstarlite.py”. La principal razón se encuentra en la cantidad de costes entre nodos que tiene que recalcular como se explicó en el apartado 4.5, como se encuentra lejos del destino el número de nodos a recalcular es considerablemente grande por lo que tarda un mayor tiempo en hacerlo. Debido a este

ensayo, el cual fue uno de los primeros en realizarse, se tuvo en consideración para los siguientes y se concluyó que, para las distancias límites del mapeado (25 metros), el programa creado para este proyecto no era posible utilizarse. Por ello, para el resto de pruebas se eligió un destino que se encontrara más próximo al punto de partida.

Otro punto a estudiar ha sido los errores de posición (en metros) entre la posición final deseada y a la que llegaba el DJI Tello, con una precisión de ±0,05 metros. En este punto, se han tenido en cuenta tanto el fallo en el eje *x* como en el eje *y*, además de la distancia total entre la posición objetivo y la posición final real tanto de ida como de vuelta. El valor positivo o negativo corresponde con los sentidos positivos o negativos de los ejes, por tanto, si en el vuelo de ida estos valores son negativos significa que no ha llegado a esa posición, pero, si el vuelo es de vuelta, significaría que ha sobrepasado esa posición. A continuación, se muestran diversos diagramas de cajas y bigotes con los resultados obtenidos. Los datos utilizados son solamente los de los vuelos completos tanto de ida como de vuelta.

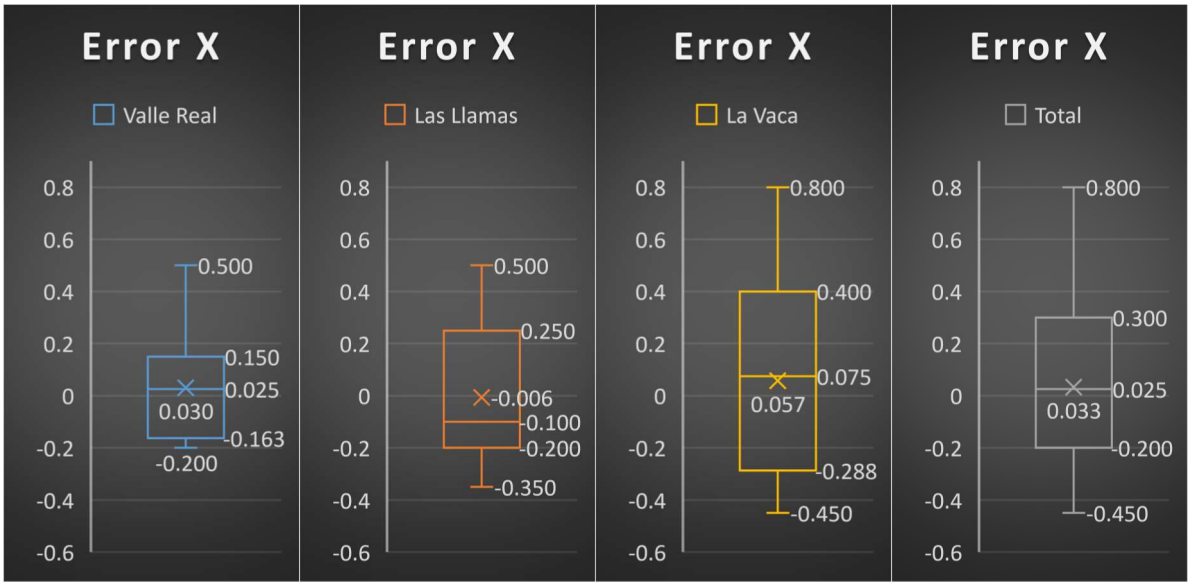


Figura 51. Error en dirección X (metros).

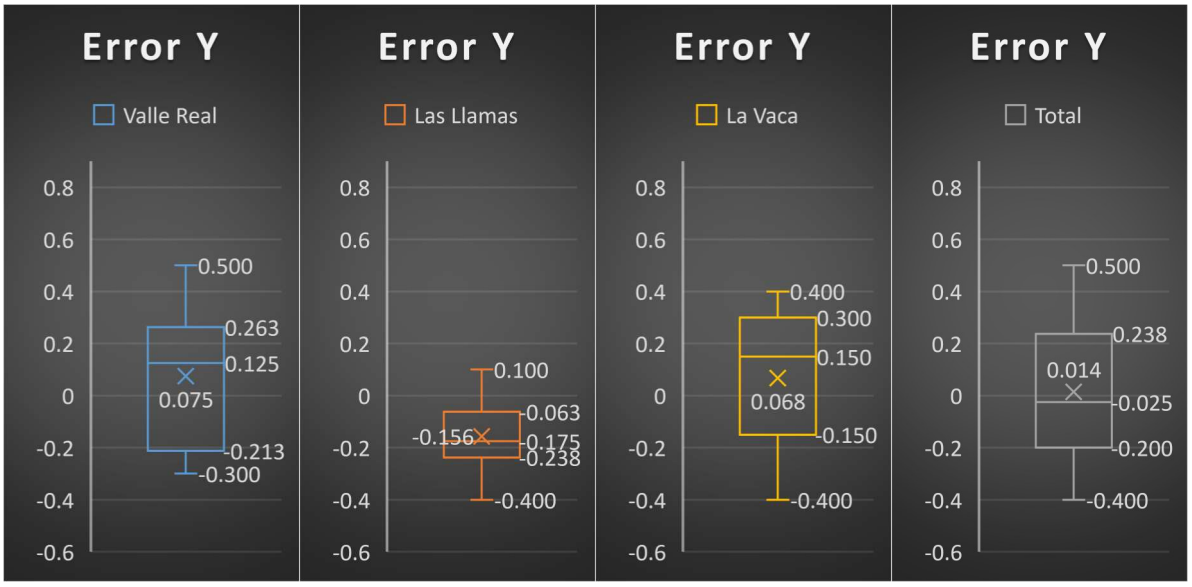


Figura 52. Error en dirección Y (metros).

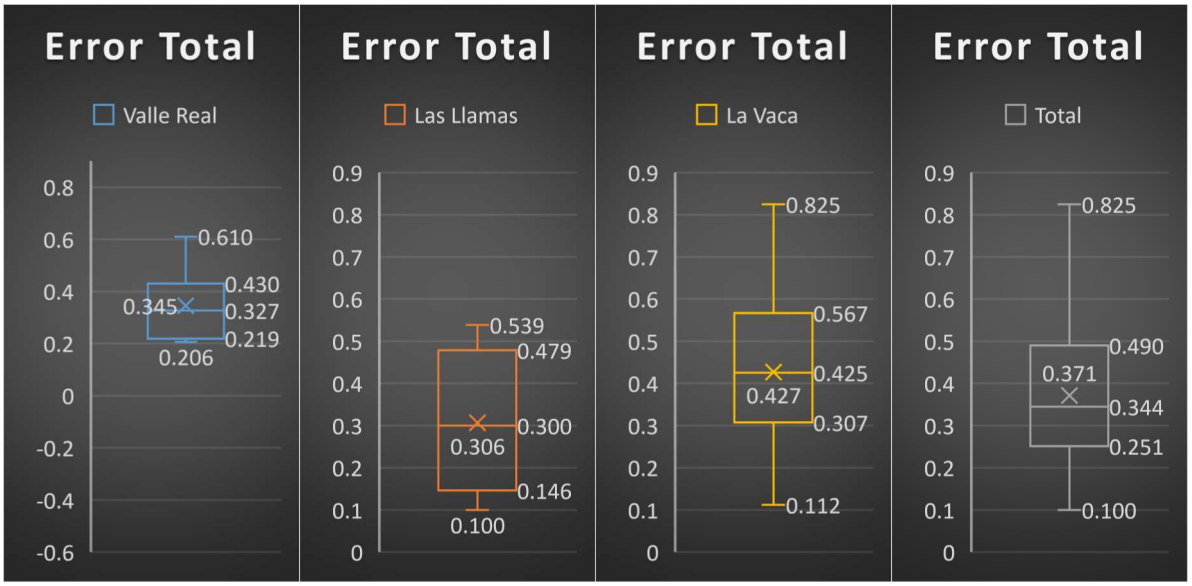


Figura 53. Distancia de error total (metros).

De estos gráficos se puede sacar como valores más característicos la media y el rango de error, los cuales corresponde con 0,025 con valores entre -0.5 y 0.8, 0.014 con valores entre -0,45 y 0,5 y 0.371 con valores entre 0.1 y 0.825, para el error en el eje *x*, el error en el eje *y* y el error total, respectivamente.

De estos resultados se puede concluir que la gran mayoría de distancia errada se encontraba entre 0.25 y 0.49 metros, esto correspondería con un error entre 1% y 2% aproximadamente de la dimensión total de 25 metros, sin embargo, la media de la distancia en línea recta desde el origen al destino es de 7.5 metros es decir un error de entre 3.3% y 6.6%.

A pesar del resultado satisfactorio en cuanto a error, a lo largo de las pruebas, se podía percibir cómo la aeronave no permanecía en su posición o se movía como se tenía que mover. Es difícil establecer la causa única de este comportamiento ya que se puede ver influenciado por diversos factores, como el viento, el sistema de autoposicionamiento del DJI Tello o el propio programa en sí. Es por esto, que estos resultados se consideran ilustrativos, pero no concluyentes.

Por el otro lado, de acuerdo a los resultados, se puede concluir con cierta seguridad que el número de obstáculos influye en el error final. En la gran mayoría de los casos si se comparan los errores de ida y de vuelta, se puede apreciar como los de vuelta son menores. Esto se considera que tiene relación con la cantidad de obstáculos que se encuentra durante el vuelo,

ya que en la mayoría de pruebas en el vuelo de vuelta el drone tenía que esquivar un número menor de obstáculos porque ya esquivaba desde un inicio los que se había encontrado en el vuelo de ida. La razón a la que el error sea menor se da en la cantidad de veces que tiene que recalcular la ruta y la cantidad de movimientos innecesarios (que no le acercan al destino) que tiene que realizar. En los siguientes gráficos se encuentra la comparación de error entre ida y vuelta.

Figura 54. Comparación errores ida y vuelta.

Se puede apreciar como todos los valores más característicos son menores, lo cual difícilmente podría ser una coincidencia ya que se han llevado a cabo prácticamente en las mismas circunstancias que en los vuelos de ida exceptuando la cantidad de obstáculos.

8.- CONCLUSIONES Y TRABAJOS POSTERIORES

A lo largo de este proyecto se han tratado temas tan actuales y con tanto potencial en la industria, como robots móviles autónomos y Deep Learning. En este proyecto se deseaba darle un enfoque conjunto y novedoso en comparación a los mostrados en la introducción de esta memoria. El objetivo principal era elaborar una primera aproximación a un robot aéreo no tripulado que pudiera realizar tareas en entornos dinámicos relacionadas con el transporte, logística, mensajería o rescates en entornos donde el ser humano no puede acceder. La única herramienta para detectar obstáculos, y así elaborar una trayectoria segura, era la cámara del DJI Tello y por ello este proyecto es tan particular.

Para conseguir llegar a la implementación definitiva se fueron persiguiendo hitos menores para acercarse paso a paso al objetivo final.

En primer lugar, se elaboró un modelo de red neuronal convolucional con una estructura de red del tipo Inception v2. Este modelo fue entrenado con 115 imágenes aplicando la técnica de Deep Learning para ser capaz de detectar 8 clases distintas de posibles obstáculos en una imagen.

Después, se estudiaron diversos algoritmos de mínima distancia para comprobar su posible utilización como herramienta a utilizar para la planificación de trayectorias. De este análisis, se concluyó que el algoritmo D*Lite era la mejor opción por su capacidad de recalcular la ruta durante la trayectoria sin tener que volver aplicar el algoritmo desde cero y minimizando así los cálculos necesarios para ello.

El siguiente paso consistió en desarrollar distintos programas para controlar el DJI Tello desde el ordenador, ya fuera mediante las teclas del ordenador o dándole una trayectoria a seguir. Igualmente, se planteó una propuesta de mapeado para aportar información durante el vuelo del drone que pudiera ser de interés. Se utilizaron los conceptos de odometría para estimar la posición y orientación de la aeronave cuando se estuviera controlando mediante el teclado.

A continuación, se desarrolló el programa “DAA.py” que utilizaba conjuntamente el modelo entrenado, el control del DJI Tello, el mapeado y la planificación de trayectorias aplicando el algoritmo D*Lite.

Por último, se realizaron diversas pruebas y se evaluaron los resultados para verificar cuánto de satisfactorio había sido la implementación este proyecto en el que se ha conseguido una aproximación a un robot aéreo autónomo no tripulado utilizando técnicas de Deep Learning.

Los resultados de este proyecto han superado las expectativas que se tenían al inicio del mismo debido a la originalidad del tema abordado y al escaso conocimiento que se tenía sobre las técnicas utilizadas, únicamente se reducía a dos cursos introductorios. Finalmente, se ha conseguido desarrollar una aeronave con diversas aplicaciones que es capaz de detectar y esquivar por sí mismo obstáculos u otros objetos de interés utilizando simplemente su cámara, realizando vuelos de ida y vuelta satisfactorios en un 73% de las ocasiones ensayadas, un valor más que positivo para una primera aproximación.

A lo largo del desarrollo de este proyecto, se han encontrado dificultades que han limitado el potencial de este proyecto y que podrían considerarse como base para el desarrollo de estudios posteriores. Entre ellas, principalmente, se encuentran aquellas relacionadas con el drone utilizado, el DJI Tello, ya que, debido a sus características, como es una batería de simplemente 15 minutos o la carencia de sensores de distancia o de geolocalización por ejemplo, es poco recomendable utilizarlo para vuelos realmente largos o que necesiten una gran precisión en cuanto a la estimación de posición de obstáculos y del propio drone. Por ello, el uso de las bases de este proyecto en un drone más sofisticado y con mayor potencia sería de interés para estudiar realmente su posibilidad de aplicación.

El poder computacional también ha sido una limitación a tener en cuenta. Ya que al realizar todas las operaciones desde una sola unidad y con una potencia tan reducida, la velocidad y fluidez del vuelo, así como la retransmisión de la cámara no era la ideal. Por ello, sería de interés utilizar una unidad computacional más potente o varias unidades para realizar los cálculos, por ejemplo, en una unidad se podrían realizar los cálculos relacionados con el algoritmo D*Lite o con el reconocimiento de objetos y en otra el resto, o incluso más de dos unidades. La gran mayoría de drones modernos y sofisticados cuentan con una CPU relativamente potente en el propio drone.

También cabe destacar que la base de imágenes con la que se entrenó el modelo no era muy extensa y la etiquetación no fue óptima ya que solo se etiquetó aquellos obstáculos próximos,

lo cual ha podido confundir al modelo ya que no tiene perspectiva de profundidad. Sería interesante comprobar nuevamente los resultados utilizando un modelo entrenado con una base de datos mayor y más rica.

Por otro lado, en este proyecto se ha estudiado la aplicación en exteriores, sin embargo, cambiando las clases a identificar en el modelo entrenado podría estudiarse su aplicación en interiores para tareas de logística.

Por lo anteriormente nombrado, se considera que la introducción de estas mejoras en los recursos utilizados podría ser un campo a explorar posteriormente al iniciado con este proyecto para desarrollar un prototipo más competitivo y eficiente.

BIBLIOGRAFÍA

[1] D. Alvaro, «Repositorio GitHub: TFM2021UC-Diego Alvaro,» 2021. [En línea]. Available: https://github.com/diegoalvaroalvarez/TFM2021UC.

[2] ICT-AGRI, «ICT and robotics for sustainable agriculture,» 2016.

[3] IDEA INGENIERIA, «Revisión de estructuras eléctricas con drones para IBERDROLA,» Mayo 2018. [En línea]. Available: https://ideaingenieria.es/project/revision-estructuras-electricas-drones-iberdrola/. [Último acceso: Septiembre 2021].

[4] Red Eléctrica de España, «Red Eléctrica apuesta por el uso de drones en la inspección y mantenimiento de líneas eléctricas,» 24 Enero 2018. [En línea]. Available: https://www.ree.es/es/sala-de-prensa/notas-de-prensa/2018/01/red-electrica-apuesta-por-el-uso-de-drones-en-la-inspeccion-y-mantenimiento-lineas-electricas. [Último acceso: Septiembre 2021].

[5] L. E. Ituarte, S. L. Martines y E. E. Tarifa, «Monitoreo en plantas fotovoltaicas: una revisión de técnicas y métodos utilizando imágenes termográficas,» 2019.

[6] FEW (AFP, EFE), «Estados Unidos autoriza a Amazon a utilizar drones para entregar paquetes,» *El Mundo,* 1 Septiembre 2020.

[7] A. Mendoza, «Tres empresas que usan drones para inventario,» 12 Octubre 2020. [En línea]. Available: https://thelogisticsworld.com/innovacion/3-empresas-que-se-usan-drones-para-inventarios/.

[8] T. Alonso, «rones, robots y coches conectados: esta es la tecnología con la que están limpiando Fukushima,» Lenovo, 21 Septiembre 2018. [En línea]. Available: https://www.bloglenovo.es/tecnologia-para-limpiar-fukushima/. [Último acceso: Septiembre 2021].

[9] CartONG, «Using High-resolution Imagery to Support the Post-earthquake Census in Port-au-Prince, Haiti,» 24 Julio 2014. [En línea]. Available: https://reliefweb.int/report/haiti/drones-humanitarian-action-case-study-no7-using-high-resolution-imagery-support-post. [Último acceso: Septiembre 2021].

[10] Y. Karaca, M. Cicek y O. Tatli, «The potential use of unmanned aircraft systems (drones) in mountain search and rescue operations,» *American Journal of Emergency Medicine,* vol. 36, nº 4, 2017.

[11] L. Munkhdalai, K. H. Park, E. Batbaatar, N. Theera-Umpon, Ryu y K. H, «Deep Learning-Based Demand Forecasting for Korean Postal Delivery Service,» *IEEE Access,* vol. 8, pp. 188135-188145, 2020.

[12] Oasys, «Deep Learning: aplicaciones reales en la Industria 4.0,» 16 Mayo 2021. [En línea]. Available: https://oasys-sw.com/deep-learning-aplicaciones-reales-industria-4-0/. [Último acceso: Septiembre 2021].

[13] H. Wang, Y. Yu, Y. Cai, X. Chen, L. Chen, Liu y Q, «A Comparative Study of State-of-the-Art Deep Learning Algorithms for Vehicle Detection,» *IEEE Intelligent Transportation Systems Magazine,* vol. 11, nº 2, pp. 82-95, 2019.

[14] I. Lenz, «DEEP LEARNING FOR ROBOTICS,» Cornell University, 2016.

[15] M. Brahimi, B. K. Kamel y A. Moussaoui, «Deep Learning for Tomato Diseases: Classification and Symptoms Visualization,» *Applied Artificial Intelligence,* vol. 31, nº 4, pp. 1-17, 2017.

[16] D. Bozic-Stulic, Z. Marusic y S. Gotovac, «Deep Learning Approach in Aerial Imagery for Supporting Land Search and Rescue Missions,» *International Journal of Computer Vision,* vol. 127, pp. 1256-1278, 2019.

[17] J. Gasienica-Jozkowy, M. Knapik y B. Cyganek, «An ensemble deep learning method with optimized weights for drone-based water rescue and surveillance,» *Integrated Computer-Aided Engineering,* vol. 28, nº 3, pp. 221-235, 2021.

[18] R. Alyassi, M. Khonji, S. Chi-Kin, K. Elbassioni y C. Tseng, «Autonomous Recharging and Flight Mission Planning for Battery-operated Autonomous Drones.,» arXiv e-prints, 2017.

[19] R. Yonetani, T. Taniai, M. Barekatain, M. Nishimura y A. Kanezaki, «Path Planning using Neural A* Search,» de *Proceedings of the 38th International Conference on Machine Learning, PMLR*, 2021.

[20] J. O. Gaya, L. T. Gonçalves, A. C. Duarte, B. Zanchetta, P. Drews y S. S. C. Botelho, «Vision-Based Obstacle Avoidance Using Deep Learning,» de *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*, 2016.

[21] Ryze, «Tello. Manual del usuario.,» 2018.

[22] S. L. Brunton y J. N. Kutz, Data-Driven Science and Engineering: Machine Learning, Dynamical Systems and Control, Cambridge: Cambridge Unversity Press, 2019.

[23] T. Haas, C. Schubert, M. Eickhoff y H. Pfeifer, «BubCNN: Bubble detection using Faster RCNN and shape regression network,» *Chemical Engineering Science,* vol. 216, nº 115467, 2020.

[24] J. I. Bagnato, Aprende Machine Learning en Español: Teoría + Práctica Python, 2020.

[25] N. Sünderhayf, O. Brock, W. Schreirer, R. Hadsell, D. Fox y J. Leitner, «The limits and potentials of deep learning for robotics,» *The International Journal of Robotics Research,* vol. 37, nº 4-5, pp. 405-420, 2018.

[26] E. Soria, J. D. Martin, M. Martinez, R. Magdalena y A. Serrano, «Chapter 11: Transfer Learning,» de *Handbook of Research on Machine Learning Applications*, Information Science Reference, 2009.

[27] O. Russakovsky, J. Deng, H. Su y e. al., «ImageNet Large Scale Visual Recognition Challenge,» *International Journal of Computer Vision,* nº 115, pp. 211-252, 2015.

[28] I. Goodfellow, Y. Bengio y A. Courville, Deep Learning (Adaptive Computation and Machine Learning series), The MIT Press, 2016.

[29] Google Brain Team, «TensorFlow. ¿Por qué TensorFlow?,» [En línea]. Available: https://www.tensorflow.org/.

[30] Google Brain Team, «TensorFlow. Casos de éxito.,» [En línea]. Available: https://www.tensorflow.org/about/case-studies?hl=es-419.

[31] Google, «Colaboratory. Preguntas frecuentes.,» [En línea]. Available: https://research.google.com/colaboratory/faq.html.

[32] K. P. Murphy, Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.

[33] «What is COCO?,» Microsoft, [En línea]. Available: https://cocodataset.org/#home. [Último acceso: Septiembre 2021].

[34] «Repositorio GitHub: TensorFlow 1 Detection Model Zoo.,» TensorFlow, [En línea]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md#snapshot-serengeti-camera-trap-trained-models. [Último acceso: Septiembre 2021].

[35] R. Girshick, «Fast R-CNN,» de *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[36] S. Ren, K. He, R. Girshick y J. Sun, «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,» vol. 39, nº 6, pp. 1137-1149, 2017.

[37] R. Balasubramanian, «Region — Based Convolutional Neural Network (RCNN),» [En línea]. Available: https://medium.com/analytics-vidhya/region-based-convolutional-neural-network-rcnn-b68ada0db871. [Último acceso: Septiembre 2021].

[38] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens y Z. Wojna, «Rethinking the Inception Architecture for Computer Vision,» de *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[39] I. Goodfellow, Y. Bengio y A. Courville, «Chapter 9: Convolutional Networks,» de *Deep Learning*, The MIT Press, 2016.

[40] E. Juras, «Repositorio GitHub: Edje Electronics,» [En línea]. Available: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10. [Último acceso: Mayo 2021].

[41] TensorFlow, «Repositorio GitHub: Object Detection Tutorial,» [En línea]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/colab_tutorials/object_detection_tutorial.ipynb. [Último acceso: Septiembre 2021].

[42] J. Torres, Deep Learning: Introducción práctica con Keras (primera parte), WHAT THIS SPACE, 2018.

[43] J. Sneyer, T. Schrijvers y B. Demoen, «Dijkstra's Algorithm with Fibonacci Heaps: An Executable Description in CHR,» de *20th Workshop on Logic Programming*, Vienna, Austria, 2006.

[44] G. Sánchez y V. M. Lozano, «Algoritmo de Dijkstra. Un tutorial interactivo.,» VII Jornadas de Enseñanza Universitaria de la Informático (JENUI 2001) , 2012.

[45] S. Rohaut y F. Ebel, Algoritmia: Tecnicas Fundamentales de Programación, ENI, 2019.

[46] A. Babinec, M. Kajan, M. Florek, T. Fico, L. Jurisica y F. Duchon, «Path Planning with Modified a Star Algorithm for a Mobile Robot,» de *Procedia Engineering*, vol. 96, Elsevier, 2014, pp. 59-69.

[47] Universidad de Pensylvania, «Robotics: Computational Motion Planning: A* Algorithm,» [En línea]. Available: https://www.coursera.org/learn/robotics-motion-planning/lecture/Vv9fL/1-4-a-algorithm. [Último acceso: Agosto 2021].

[48] A. Stentz, «Optimal and efficient path planning for partially-known environments,» de *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994.

[49] A. Stentz, «The focussed D* algorithm for real-time replanning,» *IJCAI,* vol. 95, pp. 1652-1659, 1995.

[50] S. Koenig y M. Likhachev, «Fast replaning for navigation in unknown terrain,» *IEEE Transaction on Robotics,* vol. 21, nº 3, pp. 354-363, 2005.

[51] M. M. Whiston, «NASA's Mars Rover,» *The Spartan,* 19 febrero 2017.

[52] Ryze Robotics, «Tello SDK. 1.3.0.0».

[53] D. Fuentes y J. Löw, «Repositorio Github: DJITelloPy,» [En línea]. Available: https://github.com/damiafuentes/DJITelloPy. [Último acceso: Marzo 2021].

[54] «Pygame,» [En línea]. Available: https://www.pygame.org/. [Último acceso: agosto 2021].

[55] S. Lacroix, A. Mallet, R. Chatia y L. Gallo, «Rover self localization in planetary-like environment,» de *Proc. Int. Symp. Articial Intell. Robot. Autom. Sp.*, 1999, pp. 433-440.

[56] K. Rakstad, «Repositorio GitHub: Dstar-lite-pathplanner,» [En línea]. Available: https://github.com/Sollimann/Dstar-lite-pathplanner. [Último acceso: Marzo 2021].

ANEXO I. CÓDIGO UTILIZADO.

En este apartado se encuentra el código de los tres programas principales utilizados, "Manual0.py", "d_star_lite.py" y "DAA.py". El resto del código utilizado se puede encontrar en el repositorio GitHub del proyecto [1].

• Manual.py

```
"import pygame
from time import sleep
import math

def init():

    pygame.init()
    win = pygame.display.set_mode((200, 200))

def getKey(keyName):
    ans = False

    for eve in pygame.event.get(): pass

    keyInput = pygame.key.get_pressed()

    myKey = getattr(pygame, 'K_{}'.format(keyName))

    # print('K_{}'.format(keyName))

    if keyInput[myKey]:

        ans = True

    pygame.display.update()

    return ans

######## PARAMETROS EMPIRICOS ###########

fSpeed = 40 / 10  # Forward Speed in cm/s   (20 cm/s)
aSpeed = 1800 / 10  # Angular Speed Degrees/s  (45 d/s)
interval = 0.25
dInterval = fSpeed * interval
aInterval = aSpeed * interval

################################################

x, y = 250, 250
a = 0
yaw = 0

init()
```

```
def getkeyboardinput():
    lr, fb, ud, yv = 0, 0, 0, 0  # left-right forward-backward up-down yaw-
velocity

    speed = 20  # cm/s
    aspeed = 70  # degrees/s 45 gira  cada vez

    global x, y, yaw, a
    d = 0

    if getKey('LEFT'):

        lr = -speed
        d = dInterval
        a = 180

    elif getKey('RIGHT'):

        lr = speed
        d = -dInterval
        a = 180

    if getKey('UP'):

        fb = speed
        d = dInterval
        a = -90

    elif getKey('DOWN'):

        fb = -speed
        d = -dInterval
        a = -90

    if getKey('w'):

        ud = speed

    elif getKey('s'):

        ud = -speed

    if getKey('a'):

        yv = -aspeed
        yaw -= aInterval

    elif getKey('d'):

        yv = aspeed
        yaw += aInterval

    sleep(interval)

    if yaw > 180:
        yaw = yaw - 360 * (yaw // 180)
    elif yaw < -180:
        yaw = yaw + 360 * (-yaw // 180)
```

```
    a += yaw

    x += int(d * math.cos(math.radians(a)))

    y += int(d * math.sin(math.radians(a)))

    return [lr, fb, ud, yv], (x, y), yaw ”
```

- **d_star_lite.py**

```
“from priority_queue import PriorityQueue, Priority
from grid import OccupancyGridMap
import numpy as np
from utils_d_star import heuristic, Vertex, Vertices
from typing import Dict, List

obs = np.load('datos/obs.npy')
if obs == 1:
    obstaculos = np.load('datos/obstaculos.py')

OBSTACLE = 255
UNOCCUPIED = 0


class DStarLite:
    def __init__(self, map: OccupancyGridMap, s_start: (int, int), s_goal:
(int, int)):
        """
        :param map: the ground truth map of the environment provided by gui
        :param s_start: start location
        :param s_goal: end location
        """
        self.new_edges_and_old_costs = None

        # algorithm start
        self.s_start = s_start
        self.s_goal = s_goal
        self.s_last = s_start
        self.k_m = 0  # accumulation
        self.U = PriorityQueue()
        self.rhs = np.ones((map.x_dim, map.y_dim)) * np.inf
        self.g = self.rhs.copy()

        self.sensed_map = OccupancyGridMap(x_dim=map.x_dim,
                                           y_dim=map.y_dim,
                                           exploration_setting='8N')

        if obs == 1:
            for i in range(2,len(obstaculos)):
                for j in range(-3, 3):
                    for p in range(-3, 3):
                        self.sensed_map.set_obstacle(obstaculos[i][0]+j,
obstaculos[i][1]+p)
```

```
        self.rhs[self.s_goal] = 0
        self.U.insert(self.s_goal, Priority(heuristic(self.s_start,
self.s_goal), 0))

    def calculate_key(self, s: (int, int)):
        """
        :param s: the vertex we want to calculate key
        :return: Priority class of the two keys
        """

        k1 = min(self.g[s], self.rhs[s]) + heuristic(self.s_start, s) +
self.k_m
        k2 = min(self.g[s], self.rhs[s])

        return Priority(k1, k2)

    def c(self, u: (int, int), v: (int, int)) -> float:
        """
        calcuclate the cost between nodes
        :param u: from vertex
        :param v: to vertex
        :return: euclidean distance to traverse. inf if obstacle in path
        """
        if not self.sensed_map.is_unoccupied(u) or not
self.sensed_map.is_unoccupied(v):
            return float('inf')
        else:
            return heuristic(u, v)

    def contain(self, u: (int, int)) -> (int, int):
        return u in self.U.vertices_in_heap

    def update_vertex(self, u: (int, int)):
        if self.g[u] != self.rhs[u] and self.contain(u):
            self.U.update(u, self.calculate_key(u))
        elif self.g[u] != self.rhs[u] and not self.contain(u):
            self.U.insert(u, self.calculate_key(u))
        elif self.g[u] == self.rhs[u] and self.contain(u):
            self.U.remove(u)

    def compute_shortest_path(self):
        while self.U.top_key() < self.calculate_key(self.s_start) or
self.rhs[self.s_start] > self.g[self.s_start]:
            u = self.U.top()
            k_old = self.U.top_key()
            k_new = self.calculate_key(u)

            if k_old < k_new:
                self.U.update(u, k_new)
            elif self.g[u] > self.rhs[u]:
                self.g[u] = self.rhs[u]
                self.U.remove(u)
                pred = self.sensed_map.succ(vertex=u)
                for s in pred:
                    if s != self.s_goal:
                        self.rhs[s] = min(self.rhs[s], self.c(s, u) +
self.g[u])
                    self.update_vertex(s)
```

```
            else:
                self.g_old = self.g[u]
                self.g[u] = float('inf')
                pred = self.sensed_map.succ(vertex=u)
                pred.append(u)
                for s in pred:
                    if self.rhs[s] == self.c(s, u) + self.g_old:
                        if s != self.s_goal:
                            min_s = float('inf')
                            succ = self.sensed_map.succ(vertex=s)
                            for s_ in succ:
                                temp = self.c(s, s_) + self.g[s_]
                                if min_s > temp:
                                    min_s = temp
                            self.rhs[s] = min_s
                        self.update_vertex(u)

    def rescan(self) -> Vertices:

        new_edges_and_old_costs = self.new_edges_and_old_costs
        self.new_edges_and_old_costs = None
        return new_edges_and_old_costs

    def move_and_replan(self, robot_position: (int, int)):
        path = [robot_position]
        self.s_start = robot_position
        self.s_last = self.s_start
        self.compute_shortest_path()

        while self.s_start != self.s_goal:
            assert (self.rhs[self.s_start] != float('inf')), "There is no
known path!"

            succ = self.sensed_map.succ(self.s_start,
avoid_obstacles=False)
            min_s = float('inf')
            arg_min = None
            for s_ in succ:
                temp = self.c(self.s_start, s_) + self.g[s_]
                if temp < min_s:
                    min_s = temp
                    arg_min = s_


            self.s_start = arg_min
            path.append(self.s_start)
            # scan graph for changed costs
            changed_edges_with_old_cost = self.rescan()
            #print("len path: {}".format(len(path)))
            # if any edge costs changed
            if changed_edges_with_old_cost:
                self.k_m += heuristic(self.s_last, self.s_start)
                self.s_last = self.s_start
                # for all directed edges (u,v) with changed edge costs
                vertices = changed_edges_with_old_cost.vertices
                for vertex in vertices:
                    v = vertex.pos
                    succ_v = vertex.edges_and_c_old
                    for u, c_old in succ_v.items():
```

```
                        c_new = self.c(u, v)
                        if c_old > c_new:
                            if u != self.s_goal:
                                self.rhs[u] = min(self.rhs[u], self.c(u, v)
+ self.g[v])
                        elif self.rhs[u] == c_old + self.g[v]:
                            if u != self.s_goal:
                                min_s = float('inf')
                                succ_u = self.sensed_map.succ(vertex=u)
                                for s_ in succ_u:
                                    temp = self.c(u, s_) + self.g[s_]
                                    if min_s > temp:
                                        min_s = temp
                                self.rhs[u] = min_s
                            self.update_vertex(u)
                self.compute_shortest_path()
            #  print("path found!")
            return path, self.g, self.rhs "
```

- **DAA.py**

```
"from djitellopy import tello

import Manual0 as man
import math
import os
import numpy as np
import sys
import time
import tensorflow as tf
import cv2

from grid import OccupancyGridMap, SLAM

# Valores iniales del drone
p0=[(0,0)]
p1=(0, 0, 4), (0, 0, 5)
OBSTACLE = 255
UNOCCUPIED = 0
x_dim = 500
y_dim = 500

view_range = 5
map = OccupancyGridMap(x_dim, y_dim, '8N')
vals = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
points = [(0, 0), (0, 0)]

i = 0
k = 0

slam = SLAM(map=map, view_range=view_range)
new_observation = {"pos": None, "type": None}
modo = False
manual = False
automatico = True
delante = []
```

```
def convdist(posicion, modo=0):
    """
    :param posicion: Coordenadas de un punto del mapa
    :param modo: si 0 pasa de m a px, si 1 pasa de px a m
    :return: Coordenadas convertidas
    """
    nposicion = []
    if modo == 0:
        nposicion.append(int(posicion[0]*100/10+250))
        nposicion.append(int(round(250-posicion[1] * 100 / 10)))

    elif modo == 1:
        nposicion.append(round((posicion[0]-250)*10/100, 2))
        nposicion.append(round(-(posicion[1]-250)*10/100, 2))
    npos=(nposicion[0], nposicion[1])
    return npos

def signo(num):
    """
    :param num: número a identificar si positivo o negativo
    :return: 1 si el signo es positivo, 0 si es negativo
    """
    if num < 0:
        return 0
    else:
        return 1

def angulodegiro(pos0, pos1, posref):
    """
    Calculo del angulo de giro entre dos puntos respecto a un punto de
referencia
    Devuelve el angulo entre 0-180 y el sentido horario-antihorario
    """
    ax = posref[0] - pos0[0]  # a vector pos0aposderef
    ay = posref[1] - pos0[1]
    bx = pos1[0] - posref[0] # b vector de posrefapos1
    by = pos1[1] - posref[1]
    if ax == 0:
        px = -ay
        py = 0
    elif ay == 0:
        px = 0
        py = -ax
    elif ax != -ay:
        py = -ax / (ax+ay)  # p vector perpendicular a vector a
        px = py + 1
    else:
        py = ax/ay
        px = 2

    # Get dot product of pos0 and pos1.
    _dot = (ax * bx) + (ay * by)
    _dot2 = (px * bx) + (py * by)
    # Get magnitude of pos0 and pos1.
    _magP = math.sqrt(px**2 + py**2)
    _magB = math.sqrt(bx**2 + by**2)
    _magA = math.sqrt(ax ** 2 + ay ** 2)
    _rad = math.acos(_dot / (_magA * _magB))
```

```
    _rad2 = math.acos(_dot2 / (_magP * _magB))
    # Angulo en grados.
    angle = (_rad * 180) / math.pi  # angulo entre pos 0 y pos 1
    angle2 = (_rad2 * 180) / math.pi  # angulo entre perpendicular a
pos0posref y pos1
    round(angle)
    if pos0 == pos1:
        angle = 180
        sentido = 'horario'
    elif 0 < angle2 <= 90 or angle2==180 or (angle2==0 and (bx<0 or by<0)):
        sentido = 'horario'
        angle = angle
    else:
        sentido = 'antihorario'
        angle = angle

    return int(angle), sentido

def drawobstacles(img, obstaculos):
    # Cambiar colores y formato de cada cosa
    for l in range(2, len(obstaculos)):
        for j in range(-1, 1):
            for p in range(-1, 1):
                cv2.circle(img, (obstaculos[l][0]+j, obstaculos[l][1]+p),
1, (152, 255, 255), cv2.FILLED)  # amarillo claro
    for w in range(2, len(obstaculos)):
        if obstaculos[w][2] == 4:  # Arbol
            cv2.drawMarker(img, (obstaculos[w][0], obstaculos[w][1]), (57,
143, 0), cv2.MARKER_TRIANGLE_UP, 4, 1) # verde oscuro
        elif obstaculos[w][2] == 1:  # Poste
            cv2.drawMarker(img, (obstaculos[w][0], obstaculos[w][1]), (140,
0, 0), cv2.MARKER_SQUARE, 4, 1)  # azul clarito
        elif obstaculos[w][2] == 2:  # Valla
            cv2.drawMarker(img, (obstaculos[w][0], obstaculos[w][1]), (0,
233, 255), cv2.MARKER_SQUARE, 4, 1)  # amarillo
        elif obstaculos[w][2] == 3:  # Cartel
            cv2.drawMarker(img, (obstaculos[w][0], obstaculos[w][1]), (39,
14, 34), cv2.MARKER_CROSS, 4, 1)  # morado
        elif obstaculos[w][2] == 5:  # Vehiculo
            cv2.drawMarker(img, (obstaculos[w][0], obstaculos[w][1]), (0,
0, 240), cv2.MARKER_SQUARE, 4, 1)  # rojo
        elif obstaculos[w][2] == 7:  # Persona
            cv2.drawMarker(img, (obstaculos[w][0], obstaculos[w][1]), (203,
142, 255), cv2.MARKER_TRIANGLE_DOWN, 4, 1)  # rosa
        elif obstaculos[w][2] == 8:  # Muro
            cv2.drawMarker(img, (obstaculos[w][0], obstaculos[w][1]),  (0,
128, 255), cv2.MARKER_DIAMOND, 4, 2)   # naranja
        elif obstaculos[w][2] == 6:  # Semaforo
            cv2.drawMarker(img, (obstaculos[w][0], obstaculos[w][1]), (155,
155, 155), cv2.MARKER_TILTED_CROSS, 4, 1)  # gris  ,

def drawpoints(img, points, pos, angulo=0.0, modo = 0):
    global path, destpx, destm
    for point in points:

        cv2.circle(img, point, 1, (240, 240, 240), cv2.FILLED)  # bgr color
circulos blancos
```

```
    cv2.drawMarker(mapeado, destpx, (150, 230, 150), cv2.MARKER_DIAMOND, 6,
1) # rombo verde
    cv2.putText(mapeado, f'({round(destm[0], 2)},{round(destm[1], 2)},) m
',
                (destpx[0] + 5, destpx[1] + 10), cv2.FONT_HERSHEY_PLAIN,
0.75, (150, 230, 150), 1)

    cv2.putText(img, f'({round((points[-1][0] - 250)/10, 2)},{round((-
points[-1][1] + 250)/10, 2)},{me.get_height()/100}) m {angulo}gr',
                (points[-1][0] + 3, points[-1][1] + 5),
cv2.FONT_HERSHEY_PLAIN, 0.75, (255, 0, 255), 1)
    if modo == 1:
        for point in path:
            cv2.circle(img, point, 1, (130, 130, 240), cv2.FILLED)
#circulos rojo claro
    cv2.drawMarker(img, pos, (255, 0, 0), cv2.MARKER_STAR, 6, 1) # Estrella
azul

# This is needed since the notebook is stored in the object_detection
folder.
sys.path.append("..")

# Import utilites
from utils import label_map_util
from utils import visualization_utils as vis_util

# Name of the directory containing the object detection module we're using
MODEL_NAME = 'new_graph'
# Grab path to current working directory
CWD_PATH = os.getcwd()
# Path to frozen detection graph .pb file, which contains the model that is
used for object detection.
PATH_TO_CKPT = os.path.join(CWD_PATH, MODEL_NAME,
'frozen_inference_graph.pb')
# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH, 'Preparation', 'labelmap.pbtxt')
# Number of classes the object detector can identify
NUM_CLASSES = 8
# Load the label map.
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES,

use_display_name=True)
category_index = label_map_util.create_category_index(categories)
# Load the Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

    sess = tf.compat.v1.Session(graph=detection_graph)


# Define input and output tensors (i.e. data) for the object detection
classifier
```

```
# Input tensor is the image
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
# Output tensors are the detection boxes, scores, and classes
# Each box represents a part of the image where a particular object was
detected
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
# Each score represents level of confidence for each of the objects.
# The score is shown on the result image, together with the class label.
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes =
detection_graph.get_tensor_by_name('detection_classes:0')
# Number of objects detected
num_detections = detection_graph.get_tensor_by_name('num_detections:0')
# Comienzo del programa
nuevo = str(input("Reiniciar datos de vuelo? (SI/NO): "))
if nuevo == "SI":
    pos = (250, 250)
    obstaculos = [(0, 0, 4), (0, 0, 5)]  # los dos primeros no se toman en
cuenta y es para no repetir obstaculos
    yaw = 0
    points = [(0, 0)]
    np.save('datos/obs.npy', 0)

else:
    obstaculos = np.load('datos/obstaculos.npy')
    for y in range(0,len(obstaculos)):
        p1.append((obstaculos[y][0], obstaculos[y][1]),obstaculos[y][2])
    obstaculos = p1
    pos = np.load('datos/posicion.npy')
    x = pos[0]
    y = pos[1]
    pos = (x, y)
    yaw = np.load('datos/angulo.npy')
    points = np.load('datos/recorrido.npy')
    for z in range(len(points)):
        p0.append((points[z][0], points[z][1]))
    points = p0
    lastpos = np.load('datos/lastposicion.npy')
    lastpos=(lastpos[0], lastpos[1])
    np.save('datos/obs.npy', 1)

from d_star_lite import DStarLite

print("Escriba las coordenadas (x,y) de destino en metros con precision de
0.1 m")
dx = float(input("Coordenada x = "))
dy = float(input("Coordenada y = "))
destm = (dx, dy)
destpx = convdist(destm)
dstar = DStarLite(map, pos, destpx)
path, g, rhs = dstar.move_and_replan(robot_position=pos)

man.init()

me = tello.Tello()
me.connect()
print(me.get_battery())
me.streamon()
```

```
while True:


    if man.getKey('l'):

        me.land()
        time.sleep(2)

    elif man.getKey('t'):

        me.takeoff()
        time.sleep(1)

    if man.getKey('m'):

        if modo == 0:
            i = 0
        modo = not modo
        time.sleep(0.25)

    elif man.getKey('r'):

        points = [(0, 0), (0, 0)]

    if man.getKey('f'):

        cv2.imwrite(f'Fotografias/fotografiadrone{time.time()}.jpg', frame)
        time.sleep(0.2)

    elif man.getKey('b'):

        print(me.get_battery())

    if man.getKey('q'):

        print(me.get_distance_tof())
        cv2.imwrite(f'Mapeado/mapeadofinal{time.time()}.jpg', mapeado)
        time.sleep(0.1)
        obstaculos = np.unique(obstaculos, axis=0)

        for j in range(2, len(obstaculos)):

            if obstaculos[j-k][3]==[5] or obstaculos[j-k][3]==[7]:
                del obstaculos[j-k]
                k+=1

        np.save('datos/posicion.npy', pos)
        np.save('datos/lastposicion.npy', lastpos)
        np.save('datos/angulo.npy', yaw)
        np.save('datos/recorrido.npy', points)
        np.save('datos/obstaculos.npy', obstaculos)

        print('¡Hasta la proxima')

        me.end()
        time.sleep(1)

        break
```

```
    mapeado = np.zeros((500, 500, 3), np.uint8)

    # Deteccion de objetos con el drone
    frame = me.get_frame_read().frame
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame_expanded = np.expand_dims(frame_rgb, axis=0)
     # Perform the actual detection by running the model with the image as
input

    (boxes, scores, classes, num) = sess.run(
            [detection_boxes, detection_scores, detection_classes,
num_detections],
            feed_dict={image_tensor: frame_expanded})
    # Draw the results of the detection (aka 'visulaize the results')
    [frame, delante] = vis_util.visualize_boxes_and_labels_on_image_array(
            frame,
            np.squeeze(boxes),
            np.squeeze(classes).astype(np.int32),
            np.squeeze(scores),
            category_index,
            use_normalized_coordinates=True,
            line_thickness=3,
            min_score_thresh=0.85)

    # Definir obstáculos nuevos en la lista de obstáculos
    if len(delante) > 0 and (pos[0] + round(20*math.sin(yaw)), pos[1] -
round(20*math.cos(yaw)), delante[0]) != obstaculos[-len(delante)]:
        for j in range(-3, 3):
            for p in range(-3, 3):
                map.set_obstacle((pos[0] + round(20*math.sin(yaw)) + p,
pos[1] - round(20*math.cos(yaw)) + j))
        for c in range(len(delante)):
            obstaculos.append((pos[0] + round(20*math.sin(yaw)), pos[1] -
round(20*math.cos(yaw)), delante[c]))

    if modo == manual:

        if pos != lastpos:
            lastpos = pos
        [vals, pos, yaw] = man.getkeyboardinput()
        me.send_rc_control(vals[0], vals[1], vals[2], vals[3])
        time.sleep(0.25)

    elif modo == automatico:

        if i == 0:
            path, g, rhs = dstar.move_and_replan(robot_position=pos)
            nextpos = path[1]
            anggiro, angsent = angulodegiro((pos[0]-1, pos[1]-1), path[1],
pos)
            i=1

        else:
            if new_observation is not None:
                old_map = map
                slam.set_ground_truth_map(gt_map=map)

            if pos != lastpos:
```

```
                lastpos = pos
                # slam
                new_edges_and_old_costs, slam_map =
slam.rescan(global_position=pos)

                dstar.new_edges_and_old_costs = new_edges_and_old_costs
                dstar.sensed_map = slam_map

                # d star
                path, g, rhs = dstar.move_and_replan(robot_position=pos)
                c2 = len(path)
                # print(path)

            i = i + 1
            if i % 50 == 0:
                obstaculos = np.unique(obstaculos, axis=0)

            #print(i)

            if len(path) == 1 or 0:
                print("Ha llegado a su destino, aterrice")
                me.send_rc_control(0, 0, 0, 0)
                time.sleep(0.25)
            else:
                pos = path[1]
                nextpos = destpx
                if len(path) > 2:
                    nextpos = path[2]


                anggiro, angsent = angulodegiro(lastpos, nextpos, pos)

                if anggiro != 0:

                    if angsent == 'horario':
                        yaw = yaw + anggiro
                        me.send_rc_control(0, 0, 0, 100)
                        time.sleep(anggiro/100)
                        me.send_rc_control(0, 30, 0, 0)
                        time.sleep(0.45)

                    else:
                        yaw = yaw - anggiro
                        me.send_rc_control(0, 0, 0, -100)
                        time.sleep(anggiro/100)
                        me.send_rc_control(0, 30, 0, 0)
                        time.sleep(0.45)

                    if yaw > 180:
                        yaw = yaw - 360 * (yaw // 180)
                    elif yaw < -180:
                        yaw = yaw + 360 * (-yaw // 180)

                else:
                    me.send_rc_control(0, 30, 0, 0)
                    time.sleep(0.33)
                me.send_rc_control(0, 0, 0, 0)
                time.sleep(0.2)
```

```
        if points[-1][0] != pos[0] or points[-1][1] != pos[1]:
            points.append(pos)

        # Mapeado
        #print(obstaculos)

        drawobstacles(mapeado, obstaculos)
        drawpoints(mapeado, points, pos, yaw, modo)
        cv2.imshow('Mapping', mapeado)

        # Camara drone
        frame = cv2.resize(frame, (480, 360))
        cv2.imshow("DJI TELLO", frame)

        cv2.waitKey(1)

    # Clean up
    cv2.destroyAllWindows()”
```

ANEXO II. CLASIFICACIÓN IMÁGENES UTILIZADAS COMO BASE DE DATOS.

FOTOS	Persona	Muro	Vehiculo	Arbol	Poste	Cartel	Valla	Semaforo	total/foto
1		1							1
2			2	1	1			2	6
3	1				1		2		4
4					1	1			2
5					2		2		4
6					1	1		1	3
7						1			1
8	1		1						2
9	3	1			1	2			7
10							3		3
11				2			1		3
12		1							1
13		1							1
14	2	2							4
15					6	1			7
16				2	1		2		5
17			2		1	2			5
18		2	1						3
19				1	3	1			5
20	3		2	1	1			1	8
21					2	2			4
22		1							1
23			1	1	1				3
24							3		3
25				3	1				4
26							1		1

FOTOS	Persona	Muro	Vehiculo	Arbol	Poste	Cartel	Valla	Semaforo	total/foto
27	3				1	1		2	7
28	1		1	2		1			5
29	1			6			1		8
30				1	1				2
31					1				1
32		1	1			1			3
33	1		1		2			1	5
34		1		1	1	2			5
35			1	1			1		3
36		1			1	1	1		4
37			1		2		1	3	7
38				1	2			2	5
39		2		1					3
40		1		4			1		6
41		1			1				2
42	1				1	1			3
43							1		1
44			1		3	2			6
45		2	1						3
46				2	1			1	4
47					1			1	2
48				2					2
49				1	1				2
50		1		1		1	2		5
51				1	1			1	3
52			1	1	2	1			5
53		2	1						3

FOTOS	Persona	Muro	Vehiculo	Arbol	Poste	Cartel	Valla	Semaforo	total/foto
54		1			2		1		4
55			1						1
56		1	1						2
57					2	1			3
58						1		1	2
59				1			2		3
60					1	1			2
61					2	1		1	4
62					1	1	1		3
63			1						1
64					2	1			3
65	1	2	1		1				5
66				1	1	3		1	6
67				2					2
68		1		1	1			1	4
69					1	1		2	4
70				1			1		2
71					1				1
72		1			1	1	1		4
73			2	1					3
74					2	2			4
75			1						1
76					1	1			2
77	1				1			2	4
78			3						3
79			1						1
80	2								2

FOTOS	Persona	Muro	Vehiculo	Arbol	Poste	Cartel	Valla	Semaforo	total/foto
81					1				1
82					1	2			3
83			1						1
84	1		1		1				3
85				1	1	1	2		5
86	4		2						6
87		1	1	2			1		5
88			1		1		1		3
89				1	1				2
90	1		1		3			2	7
91					1	1			2
92		1		2					3
93	1								1
94		1							1
95		1		1					2
96					1	2		2	5
97			1		1	1	1		4
98							3		3
99							2		2
100	1		1				1		3
101			1						1
102			1	1			1		3
103				1	1	2			4
104				2	1	1			4
105					2	1			3
106					2	2			4
107			2		1	1			4

FOTOS	Persona	Muro	Vehiculo	Arbol	Poste	Cartel	Valla	Semaforo	total/foto
108			1		4	1			6
109			1	1	2	1			5
110		2							2
111			2		1				3
112			1	1					2
113			1	1	1				3
114		3	1			1			5
115			2	1	1				4
TOTAL	29	36	51	58	93	53	40	27	387
TOTAL TEST	11	7	14	13	10	8	8	3	74
Total Practica	18	29	37	45	83	45	32	24	313
%Text	37.93	19.44	27.45	22.41	10.75	15.09	20.00	11.11	19.12
%Practica	62.07	80.56	72.55	77.59	89.25	84.91	80.00	88.89	80.88

ANEXO III. RESULTADOS DE LAS PRUEBAS.

Valle Real	Ida					Vuelta				Ida	Vuelta
	Destino	Error X	Error Y	E. Total	Obstáculos	Error X	Error Y	E. Total	Obstáculos	Media	Media
1	(5,5)	0.1	0.2	0.22	2	-0.2	0.2	0.28	0	**ErrorX**	**ErrorX**
2	(0,3)	0.15	-0.2	0.25	1	0.15	0.05	0.16	0	0.19	0.13
3	(-4.4,-10.1)	0.5	0.5	0.71	2	0	-0.3	0.30	1	0.23	0.22
4	(-11.5, 5.5)	-0.15	0	0.15	3	-0.2	0.3	0.36	2	**E. Tot**	**E. Tot**
5	(-1,6.9)	0.05	0.25	0.25	2	-0.1	-0.25	0.27	0	0.31713291	0.27415399
6	(-17,-16.9)									10	3
7	(-10,-10)									Ida	Vuelta
Las Llamas	Ida					Vuelta				Media	Media
	Destino	Error X	Error Y	E. Total	Obstáculos	Error X	Error Y	E. Total	Obstáculos	**ErrorX**	**ErrorX**
1	(5,5)	-0.2	-0.2	0.28	1	-0.2	-0.15	0.25	1	0.25	0.2125
2	(0,3)	0.1	-0.05	0.11	1	0	-0.1	0.10	0	**ErrorY**	**ErrorY**
3	(-10.5,2.9)	-0.2	-0.25	0.32	3	-0.35	-0.4	0.53	2	0.175	0.1875
4	(-2.6,10.1)									**E. Tot**	**E. Tot**
5	(-15.2,-23.3)									0.20888647	0.19962251
6	(10,10)	0.5	-0.2	0.54	3	0.3	0.1	0.32	2	**Obs**	**Obs**
La Vaca	Ida					Vuelta				8	5
	Destino	Error X	Error Y	Total	Obstáculos	Error X	Error Y	Total	Obstáculos	Ida	Vuelta
1	(5,5)	-0.25	-0.1	0.27	2	0.35	0.3	0.46	1	Media	Media
2	(0,3)									**ErrorX**	**ErrorX**
3	(8.3,-13.4)	0.4	-0.4	0.57	4	-0.4	0.3	0.50	2	0.39285714	0.29285714
4	(9.9,4.2)	-0.45	0.35	0.57	3	0.4	-0.05	0.40	1	**E. Tot**	**E. Tot**
5	(0,20)	-0.25	-0.3	0.39	3	-0.45	0.4	0.60	2	0.48393354	0.370523
6	(22,22)									**Obs**	**Obs**
7	(-4.6,2.8)	-0.2	0.25	0.32	1	0.05	0.1	0.11	0	17	7
8	(-17, 5.2)	0.8	0.2	0.82	2	0.3	-0.15	0.34	0		
9	(10,10)	0.4	0.2	0.45	2	0.1	-0.15	0.18	1		