

Grado Universitario en Ingeniería Informática
2021-2022

Trabajo Fin de Grado

Detección y geolocalización de objetos automática desde drones

Pablo de Alba Martínez

Tutor

Daniel Amigo Herrero

Leganés, 2022



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento – No Comercial – Sin Obra Derivada**

DEDICATORIA

Esta dedicatoria está destinada a todas las personas que han hecho posible, mediante su apoyo, la realización de este documento, que ha supuesto uno de los retos más complicados a los que he tenido que enfrentarme en mi vida.

A mis padres y a mi hermana, que han sido un pilar indispensable en mi vida. A mis padres por permitirme estudiar esta carrera, aguantando mis desvaríos sobre programación cuando no son capaces de entenderlo. A mi hermana, por haberme enseñado con cuatro años lo que era una consola, lo que inició mi creciente interés por la tecnología.

A mis amigos, tanto los que he hecho durante este grado, como los amigos de toda la vida. Los que he hecho durante la carrera me han hecho disfrutar de los mejores años de mi vida, son amistades para toda la vida forjadas durante muy poco tiempo. A mis amigos de toda la vida, gracias por estar ahí, aunque hayamos perdido las viejas costumbres.

Por supuesto, este documento también está dedicado a mi tutor, Daniel, cuya ayuda y tutelaje han sido imprescindibles para la realización del mismo.

Y en general, este trabajo va dedicado a todas las personas que me rodean en el día a día, que me han sabido apoyar en mis mejores momentos, pero sobre todo en los peores.

Muchas gracias.

RESUMEN

En los últimos años la tecnología ha avanzado en gran medida hasta el punto de que pequeños dispositivos como **drones** tienen un gran abanico de usos. Esto permite pensar posibles nuevos usos para los drones, de manera que faciliten el trabajo de otras personas o incluso, en algunos casos, lo sustituya. Por otro lado, existen muchas maneras de mapear calles, tanto con personas capturando imágenes o coches que a medida que las recorren, van capturando todos los detalles de estas. No se puede olvidar tampoco el creciente interés en **inteligencias artificiales**, cada vez más avanzadas, que transforman tareas que antes eran impensables, como por ejemplo el reconocimiento de objetos en imágenes, en tareas mucho más sencillas.

De estos datos surge la idea de este proyecto: ser capaces de utilizar drones para capturar imágenes de calles o carreteras y utilizar una IA para detectar las señales capturadas.

Este proyecto permite la ejecución de **misiones autónomas** de drones en un **entorno simulado** mediante AirSim en el motor gráfico de Unreal. El proyecto está pensado para obtener imágenes durante la ejecución de la misión y obtener datos de telemetría desde los distintos sensores como, por ejemplo, la posición del dron, qué cámara ha sacado la imagen, en qué momento se ha sacado la imagen o las propiedades de la cámara. Esta información será utilizada para que un **modelo** (previamente entrenado para detectar señales de tráfico) sea capaz de detectar las señales captadas en las imágenes y, gracias a la información obtenida por los **sensores**, ser capaz de identificarla y aproximar su posición.

El desarrollo de este proyecto se ha realizado en su totalidad en el lenguaje de programación de Python.

El resultado final consiste en un conjunto de imágenes con las detecciones de las señales y un conjunto de archivos JSON con la información de cada una de las señales detectadas. Este proyecto sienta las bases de un sistema de localización de señales en un entorno simulado, de manera que en un futuro esto se pueda llevar a un entorno real.

El desarrollo de este proyecto se ha dividido en fases independientes que han ido variando durante el tiempo hasta obtener el producto final, que se explicará en este documento. La primera meta fue obtener un modelo que detecte señales de tráfico de manera consistente; la segunda ser capaz de crear una misión autónoma en la que el dron realice un recorrido y capture imágenes durante el mismo y obtener la telemetría asociada. La última meta es juntar las dos metas anteriores y poder clasificar las señales detectadas por el modelo de la primera meta en las imágenes capturadas por misión autónoma gracias a la telemetría proporcionada por los sensores del dron.

Palabras clave: Dron, inteligencia artificial, misión autónoma, entorno simulado, modelo, sensores.

ABSTRACT

In recent years, technology has advanced greatly to the point where small devices such as **drones** have a wide range of uses. This allows us to think of possible new uses for drones, so that they can facilitate the work of other people or even, in some cases, replace it. On the other hand, there are many ways of mapping streets, either with people capturing images or cars that, as they drive along a street, capture all the details of it. Nor can we forget the growing interest in increasingly advanced **artificial intelligences**, which allow tasks that were previously unthinkable, such as the recognition of objects in images, to become much simpler tasks.

From this data comes the idea of this project: to be able to use drones to capture images of streets or roads and use an AI to detect the captured signals.

This project enables the execution of **autonomous drone missions** in a **simulated environment** using AirSim in the Unreal graphics engine. The project is designed to obtain images during the execution of the mission and to obtain telemetry data from the different sensors such as the position of the drone, which camera has taken the image, at what time the image was taken or the properties of the camera. This information will be used so that a **model** (previously trained to detect traffic signals) will be able to detect the signals captured in the images and, thanks to the information obtained by the **sensors**, be able to identify it and approximate its position.

The development of this project will be carried out entirely in the Python programming language.

The final result would be a set of images with the signal detections and a set of JSON files with the information of each of the detected signals. This project lays the foundations for a signal localisation system in a simulated environment, so that in the future this can be taken to a real environment.

The development of this project has been divided into independent phases that have varied over time to obtain the final product that will be explained in this document. The first goal was to be able to have a model that detects traffic signals consistently; the second to be able to create an autonomous mission in which the drone performs a route and captures images during it, and also obtain the associated telemetry. The last goal is to put the two previous goals together and to be able to classify the signals detected by the model of the first goal in the images captured by the autonomous mission thanks to the telemetry provided by the drone's sensors.

Keywords: Drone, artificial intelligence, autonomous drone mission, simulated environment, model, sensors.

ÍNDICE DE CONTENIDOS

DEDICATORIA	ii
RESUMEN	iii
ABSTRACT	iv
ÍNDICE DE CONTENIDOS	v
Índice de figuras	viii
Índice de tablas	ix
Índice de ilustraciones	x
1 INTRODUCCIÓN	1
1.1 Visión general	1
1.2 Motivación	1
1.3 Objetivos	2
1.4 Estructura del documento	3
2 Estado del arte	4
2.1 Drones	4
2.1.1 Clasificación	4
2.1.2 Componentes	6
2.1.3 Controlador de vuelo	8
2.2 Simulación de drones	9
2.2.1 AirSim	9
2.2.2 CARLA	9
2.3 Motor Gráfico	9
2.3.1 Unreal Engine	10
2.3.2 Unity	10
2.3.3 Source Engine	11
2.4 IA	11
2.4.1 Computer visión	12
2.4.2 Detectron	13
2.5 Entorno de desarrollo	13
2.5.1 Sistema operativo	13
2.5.2 Lenguaje de programación	14
2.5.3 Google Colaboratory	15
2.6 Aplicaciones Similares	15
2.6.1 Mapillary	15

2.6.2	Google Street View	16
3	Entorno socioeconómico y legal.....	17
3.1	Impacto socioeconómico	17
3.2	Marco regulador.....	17
4	Diseño de la solución.....	19
4.1	Arquitectura del sistema	19
4.1.1	Mi Misión	20
4.1.2	PX4.....	20
4.1.3	MavSDK.....	20
4.1.4	AirSim	21
4.1.5	Unreal	21
4.1.6	Detectron	21
4.2	TrafficSignDetection	21
4.2.1	exampleSettings.json	22
4.2.2	mainNoGUI.py	24
4.2.3	missionHandler.py	26
4.2.4	trafficSign.py	26
4.2.5	mapTools.py	28
4.3	Sección de Google Colaboratory	29
5	Planificación	33
5.1	Planificación	33
5.1.1	Ejecución del modelo de planificación.....	33
5.1.2	Diagrama de Gantt.....	34
5.2	Requisitos / casos de uso	37
5.2.1	Requisitos de usuario.....	38
5.2.2	Requisitos de software funcionales	41
5.2.3	Requisitos de software no funcionales	43
5.2.4	Casos de uso	44
5.2.5	Matriz de trazabilidad.....	46
5.3	Presupuesto	46
5.3.1	Costes directos.....	46
5.3.2	Costes indirectos.....	47
5.3.3	Costes finales.....	47
5.3.4	Oferta del proyecto	48
6	Implementación / Resultados.....	49
6.1	Implementación	49

6.2	Resultados.....	63
7	Conclusiones.....	65
7.1	Líneas futuras.....	66
7.1.1	Realizar pruebas en entornos reales.....	66
7.1.2	Mejorar el sistema de detección	66
7.1.3	Localización con precisión	66
7.2	Problemas.....	66
8	Bibliografía / Referencias.....	68

Índice de figuras

Figura 1: Partes de un Dron.....	6
Figura 2: Diagrama de componentes Ubuntu	19
Figura 3: Diagrama de componentes Windows.....	20
Figura 4: Diagrama de componentes Colaboratory	20
Figura 5: Posición Cámaras dron	22
Figura 6: Procedimiento de checkpoints	28
Figura 7: Diagrama detección señales	31
Figura 8: Diagrama de Gantt parte 1	35
Figura 9: Diagrama de Gantt parte 2	36
Figura 10: Recorrido del Dron Implementación.....	50
Figura 11: Señales de Mapillary en la zona de ejemplo	51
Figura 12 - A1: Components Ubuntu	75
Figura 13 - A2: Components Windows	76
Figura 14 - A3: Components Colaboratory	76

Índice de tablas

Tabla 1: Plantilla de especificación de requisitos.....	38
Tabla 2: Requisito de usuario RU-01	38
Tabla 3: Requisito de usuario RU-02	39
Tabla 4: Requisito de usuario RU-03	39
Tabla 5: Requisito de usuario RU-04	39
Tabla 6: Requisito de usuario RU-05	40
Tabla 7: Requisito de Usuario RU-06	40
Tabla 8: Requisito de Usuario RU-07	40
Tabla 9: Requisito de software funcional RF-01	41
Tabla 10: Requisito de software funcional RF-02.....	41
Tabla 11: Requisito de software funcional RF-03.....	41
Tabla 12: Requisito de software funcional RF-04.....	42
Tabla 13: Requisito de software funcional RF-05.....	42
Tabla 14: Requisito de software funcional RF-06.....	42
Tabla 15: Requisito de software no funcional RNF-01	43
Tabla 16: Requisito de software no funcional RNF-02.....	43
Tabla 17: Requisito de software no funcional RNF-03.....	43
Tabla 18: Requisito de software no funcional RNF-04.....	44
Tabla 19: Requisito de software no funcional RNF-05.....	44
Tabla 20: Plantilla de casos de uso	44
Tabla 21: Caso de uso CU-01	45
Tabla 22: Caso de uso CU-02.....	45
Tabla 23: Matriz de trazabilidad.....	46
Tabla 24: Resumen del proyecto	46
Tabla 25: Costes de trabajadores	47
Tabla 26: Costes de equipo y software.....	47
Tabla 27: Costes finales.....	47
Tabla 28: Desglose de la oferta	48
Tabla 29 A-1: User Requirement UR-03.....	79
Tabla 30 A-2: Functional Software Requirement FR-04	79
Tabla 31 A-3: Non-Functional Software Requirement NFR-02	79
Tabla 32 A-4: User case UC-01	80
Tabla 33: Offer breakdown.....	80

Índice de ilustraciones

Ilustración 1: Piloto automático PX4	8
Ilustración 2: Ejemplo de Opciones de cámara	23
Ilustración 3: Ejemplo de configuración del dron	25
Ilustración 4: Procedimiento de missionHandler.py si la misión es trafficSign.....	26
Ilustración 5: Librerías e instalaciones necesarias para utilizar Detectron.....	29
Ilustración 6: Ajustes de entrenamiento utilizados para el modelo	30
Ilustración 7: Señales implementación	51
Ilustración 8: Pasos inicio Unreal	52
Ilustración 9: Configuración de launch.json.....	53
Ilustración 10: Mensajes de conexión satisfactoria	53
Ilustración 11: Vuelo del dron desde AirSim 1	53
Ilustración 12: Vuelo del dron desde AirSim 2	54
Ilustración 13: Ejemplo imagen capturada por el dron.....	54
Ilustración 14: Collage de imágenes capturadas por dron	55
Ilustración 15: Conexión con AirSim desde mission_handler.py	55
Ilustración 16: Ejemplo de recorrido del dron desde mission_handler.py	56
Ilustración 17: Mensajes al finalizar la misión	56
Ilustración 18: Ruta a la carpeta con las imagenes de salida.....	56
Ilustración 19: Ubicación por defecto de los proyectos de Colab en drive	57
Ilustración 20: Mensaje de instalación satisfactoria	57
Ilustración 21: Distribución Dataset	58
Ilustración 22: Distribución imágenes	58
Ilustración 23: Ejemplo de clase.....	58
Ilustración 24: Cuantificación del muestreo	58
Ilustración 25: Ejemplo imagen entrenamiento	59
Ilustración 26: Distribución de Instancias	59
Ilustración 27: Inicio del entrenamiento	60
Ilustración 28: Final del entrenamiento	60
Ilustración 29: Ejemplo de validación	60
Ilustración 30: Ejemplo de detección de una señal.....	61
Ilustración 31: Ejemplo señal detectada múltiples veces	61
Ilustración 32: Comparación antes/después de la detección	62
Ilustración 33: JSONs de las señales detectadas	64
Ilustración 34 A-1: Successful connection message.....	81
Ilustración 35 A-2: Flight from the drone	81
Ilustración 36A-3: Image captured from the drone	82
Ilustración 37 A-4: Collage of images captured by the drone at the checkpoint.....	82
Ilustración 38A-5: Successful installation message	83
Ilustración 39 A-6: Dataset distribution	83
Ilustración 40 A-7: Training Sample Image	83
Ilustración 41 A-8: Start of training	84
Ilustración 42 A-9: End of training.....	84
Ilustración 43 A-10: Comparison before/after detection	85
Ilustración 44 A-11: Signal detected multiple times example.....	86

1 INTRODUCCIÓN

En este apartado se realizará una visión general del proyecto, incluyendo las motivaciones del mismo y los objetivos que se esperan cumplir con la realización del mismo.

1.1 Visión general

Durante los últimos años se ha incrementado el número de tareas automatizadas mediante herramientas no controladas por el hombre, desde multas de tráfico hasta reconocimientos faciales que nos ofrecen una gran cantidad de beneficios. La creación y uso de inteligencias artificiales nos permiten no solo facilitar el trabajo de un ser humano, sino también automatizar procesos que serían mucho más lentos de intervenir la mano humana.

Uno de los principales usos que se dan a las inteligencias artificiales (IA) es el reconocimiento y procesamiento de imágenes. La IA recibe de entrada una imagen sin procesar, y de salida obtiene una gran cantidad de información. En función del tipo de IA puedes obtener muchos datos acerca de esta imagen.

Por otro lado, tenemos el auge en el uso de los drones, que han pasado de ser dispositivos caros, difíciles de usar y de acceso muy limitado, a ser dispositivos de acceso mucho más sencillo, baratos y, mediante el uso de herramientas como PX4 o mavSDK, mucho más sencillos de configurar y usar. Estos permiten acceder a zonas en las que otro vehículo o incluso personas no puedan acceder.

De esta idea nace este proyecto: mediante la captura de imágenes en tiempo real y el posterior análisis de estas, podemos obtener todas las señales de tráfico de una carretera concreta y estimar la posición de cada una de ellas, de manera que se pueden determinar multitud de datos, como si la posición es la correcta o su visibilidad es buena, entre otros. La principal intención de este proyecto es diseñar una herramienta que permita, con ayuda de un dron (pilotado por una persona o no), capturar imágenes de una ruta determinada y posteriormente analizarlas con ayuda de una inteligencia artificial para determinar qué tipo de señal es y en qué posición geográfica ha sido detectada. Este proyecto se realizará en un entorno simulado (por razones que se explicarán en la sección del estado del arte), pero la intención de futuro es que estas herramientas se puedan probar en un entorno real.

1.2 Motivación

En la última década el uso de los drones se ha extendido enormemente en todos los ámbitos, ya que suelen ser fáciles de manejar y tienen una gran cantidad de funcionalidades. También se puede observar que existen multitud de vehículos que te permiten capturar imágenes de distintas zonas de las calles para mapear (como hace Google Maps) o para tener información de las calles de todo el mundo (como hace Mapillary). Estos dos ejemplos se realizan siempre desde coches o motos, necesitando al menos una persona (el conductor, por ejemplo). Sin embargo, si se pudieran utilizar drones para recorrer las calles de manera automática no sería necesario tener una persona controlando el dron. Además, también se facilitaría el acceso a carreteras con recorridos que podrían ser peligrosos para un ser humano.

Desde el punto de vista económico, los drones han visto su inversión enormemente incrementadas en España, con más de 1400 millones de euros destinados por el ministerio de industria[1], o en empresas privadas, cuya inversión en 2021 llegó a superar los 7000 millones de dólares[2].

Todo esto genera la posibilidad de crear un programa que permita realizar misiones autónomas con drones, de manera que no se pongan en peligro vidas humanas y abaraten los costes (Pues los drones son relativamente baratos si los comparamos con coches o motos).

También me gustaría comentar mis motivaciones personales para este proyecto, y es que a lo largo del grado se han ido proponiendo distintos proyectos con el fin de mejorar la aptitud en la programación. Este proyecto supone un reto de programación e investigación muy superior al propuesto durante los cursos, y supone un reto de aprendizaje muy atractivo para mí, que especializado en la rama de computadores ha implicado la adquisición de gran cantidad conocimientos nuevos en muy poco tiempo, como el funcionamiento de modelos, simulaciones y programar en un lenguaje que apenas he utilizado durante el grado (Python).

1.3 Objetivos

El principal objetivo de este proyecto consiste en ser capaz de generar un modelo de detección de señales de tráfico funcional que permita detectarlas en imágenes generadas en una simulación de AirSim y ser capaz de clasificar cada una de las señales. La parte más importante de este objetivo es la de detectar las señales, siendo menos prioritario clasificarlas correctamente, aunque no opcional.

Otros objetivos de este proyecto son los siguientes:

- Ser capaz de realizar una misión de dron autónoma. El objetivo de esa misión será realizar un recorrido entre dos puntos, dividido en puntos de control, capturando imágenes desde el dron.
- Utilizar un mapa que simule un entorno real con señales de tráfico lo más realistas posibles. Para ello se utilizará AirSim mediante Unreal.
- Ser capaz de tener información constante del estado de la misión autónoma. Para ello se generará un log durante la misión, además de poder visualizar el recorrido del dron en todo momento.
- Obtener las imágenes resultantes de la detección de señales, para tener una perspectiva más visual de las mismas. No solo utilizaremos los JSON con las señales detectadas, sino que obtendremos las imágenes de la misión con las señales detectadas remarcadas en recuadros.

1.4 Estructura del documento

2 Estado del arte: Este apartado trata de explicar los conceptos previos necesarios para entender en su totalidad el contenido de este proyecto.

3 Entorno socioeconómico y legal: Este apartado trata del impacto social y económico que podría tener la implementación de este proyecto, además del marco regulador del que se ciñe.

4 Diseño de la solución: Este apartado trata del diseño propuesto para este proyecto.

5 Planificación: Este apartado especifica todos los detalles técnicos del proyecto, incluyendo la planificación de tareas, la especificación de requisitos y el presupuesto.

6 Implementación / Resultados: Este apartado trata de las instrucciones para implementar el proyecto, su relación con los casos de uso y un análisis de los resultados.

7 Conclusiones: Este apartado trata de las conclusiones del proyecto, si ha cumplido los objetivos, líneas de trabajo futuras y problemas ocasionados durante el desarrollo.

2 Estado del arte

Para poder entender el desarrollo de este proyecto será necesario conocer algunos conceptos. Entre estos conceptos básicos se encuentran qué es un dron, sus principales partes y los distintos programas que usaremos a lo largo del proyecto, además de compararlos brevemente con sus alternativas más conocidas. También se hablará brevemente del entorno de desarrollo utilizado para el desarrollo de este proyecto.

2.1 Drones

Un vehículo aéreo no tripulado (VANT) o UAV (unmanned aerial vehicle) en inglés, conocido comúnmente por todo el mundo como dron (drone en inglés), es una aeronave que vuela sin tripulación y ejerce sus funciones de manera remota, suele ser reutilizable y es capaz de mantener de manera autónoma un nivel de vuelo constante y sostenido. Su diseño es muy variado en forma, tamaño, configuración y características.

Su origen es militar, nació como un intento de blanco aéreo de entrenamiento que se controlaba por radio desde tierra durante la Primera Guerra Mundial (1914-1918) y, más tarde, como blancos para entrenar a los cañones antiaéreos durante la segunda Guerra Mundial (1939-1945). Solo en el último tramo del siglo XX se podría considerar que los drones son plenamente autónomos y su uso civil se remonta a principios del siglo XXI, siendo solo en los últimos años cuando su uso se ha normalizado enormemente en el ámbito civil[3].

2.1.1 Clasificación

Los drones se pueden clasificar de muchas maneras en función de diversos parámetros, como pueden ser su tipo de uso, el número de brazos, por su propia definición, por la altitud máxima en la que opera o incluso su peso. Estas son las clasificaciones más comunes para los drones.

2.1.1.1 Por tipo de uso

Es la clasificación más abierta, en función de su tipo de uso podemos clasificar por:

- Drones de uso militar: Son los más complejos y evolucionados. Se utilizan para multitud de tareas, como abastecimiento en zonas de difícil acceso, ataques aéreos, localización de objetivos, escaneo de grandes terrenos o blancos de entrenamiento.
- Drones de uso civil: Normalmente son menos complejos, mucho más baratos y accesibles. Al igual que los de uso militar tienen una gran cantidad de funciones, la mayoría de ellas están relacionadas con el ocio, permitiendo llegar a zonas de difícil acceso, facilitan la realización de sesiones fotográficas o vídeos, etc.

2.1.1.2 Por peso

Los drones se pueden clasificar en función de su peso:

- Nanodrones: Aquellos drones cuyo peso es inferior a 250 gramos
- Microvehículos aéreos (MAV): Drones que oscilan entre 250 gramos y 2 kilogramos
- Vehículos aéreos no tripulados en miniatura (SUAV): Desde los 2 kilogramos hasta los 25 kilogramos.
- Drones medianos: desde los 25 kilogramos hasta los 150 kilogramos.
- Drones Grandes: de uso casi exclusivamente militar, son aquellos drones que superan los 150 kilogramos de peso.

2.1.1.3 Por su altitud

Además de su peso, los drones se pueden clasificar por la altitud máxima a la que pueden volar:

- Handheld (Drones de mano): Pueden volar hasta 600 metros de altitud, con una media de 2 kilómetros de alcance en vuelo.
- Close: Alcanzan los 1500 metros de altitud y tienen un alcance de hasta 10 kilómetros.
- NATO: Aproximadamente 3000 metros de altitud, con un alcance de 50 kilómetros.
- Tactical: Alcanzan aproximadamente 3500 metros de altitud con un alcance de hasta 160 kilómetros.
- MALE (Medium altitude, Long endurance): Llegan hasta los 9000 metros de altitud con 200 kilómetros de alcance.
- HALE (High altitude, Long endurance): Altitud similar al MALE, pero con mayor alcance.
- HYPERSONIC: Son capaces de alcanzar o superar la velocidad del sonido y de alcanzar una altitud suborbital (unos 15000 metros de altitud) con un alcance de más de 200 kilómetros.
- ORBITAL: Drones capaces de realizar órbitas bajas terrestres.
- CIS lunar: Capaces de realizar trayectos entre la luna y la Tierra.

2.1.1.4 Por número de brazos

Esta clasificación es muy común entre los drones de uso civil, y es quizá la mejor forma de clasificarlos dentro de ese ámbito. La clasificación es la siguiente:

- Cuadricóptero: Drones compuestos de cuatro brazos y cuatro motores, uno por cada brazo. Son los más habituales y extendidos en el mercado.
- Hexacópteros: Son drones de seis brazos y seis motores. Son los más comunes en el ámbito profesional por su estabilidad y seguridad, pues pueden perder un motor y aterrizar sin mayor dificultad.
- Ocotocópteros: Ocho brazos y ocho motores componen este tipo de dron, por lo que son más estables, pero considerablemente más pesados.
- Coaxiales: No tienen un número fijo de brazos, pero poseen dos motores por cada brazo.

Si tuviéramos que clasificar el dron que se va a usar en este proyecto, sería un dron de uso civil, MAV, Handheld y Cuadricóptero.

2.1.2 Componentes

Teniendo en cuenta el tipo de dron que se va a usar en este proyecto, estas son las partes más destacables de un dron:

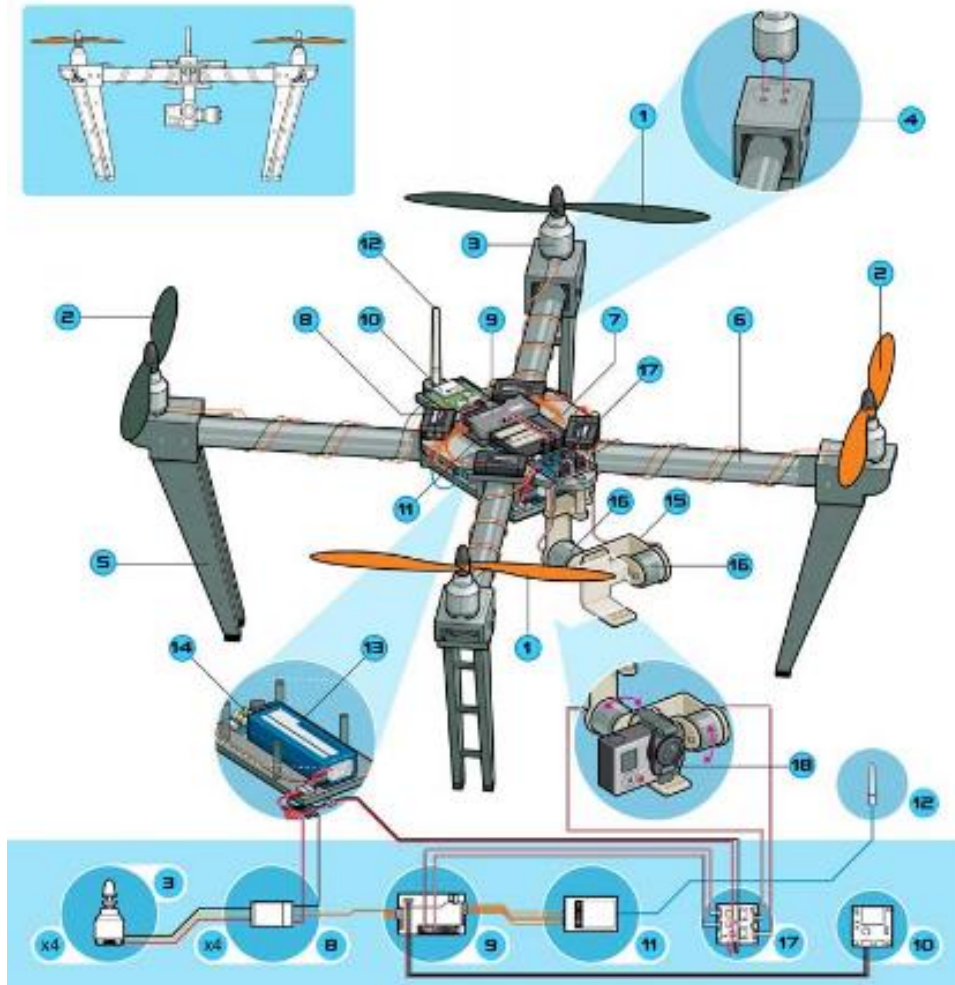


Figura 1: Partes de un Dron

Como se puede observar en la Figura 1 existen un gran número de elementos en el dron, todos numerados en la imagen y explicados a continuación:

1. Hélices estándar: Son las hélices que giran en sentido contrario a las agujas del reloj. Normalmente están hechas de plástico, si bien las de mejor calidad son de fibra de carbono.
2. Hélices inversas: Son las hélices que giran en el sentido de las agujas del reloj, suelen ser del mismo material que las hélices estándar y se suelen colocar perpendicularmente (Ver Figura 1).
3. Motores: En este caso los motores son sin escobilla, en función del dron pueden tener escobilla o no. Por lo general los motores sin escobilla son más fiables, eficientes y silenciosos, por lo que normalmente se preferirán esos.

4. Soporte del motor: Permite conectar el motor con los brazos del dron. En algunas ocasiones, el soporte suele estar integrado con el propio motor o con su brazo correspondiente. Cuando es una pieza independiente, es más sencillo de sustituir en caso de defecto o rotura.
5. Tren de aterrizaje: Permiten crear espacio entre la parte principal del cuerpo del dron y el suelo. Los trenes de aterrizaje más comunes son los que se encuentran debajo de los brazos del dron (Como se puede observar en Figura 1) o debajo del cuerpo principal del dron.
6. Brazos: Permiten conectar los motores con el cuerpo principal del dron. Pueden formar parte del cuerpo principal o ser una pieza independiente. Cuantos más brazos (y por lo tanto más motores) más estable es el dron, y es más sencillo de manejar cuando pierde un motor. Su tamaño también es muy influyente en el diseño final de un dron, pues cuanto más largo sea el brazo mayor estabilidad tendrá el dron.
7. Cuerpo central del dron: Es la parte central del dron, en ella se conectan todos los brazos y además alberga la mayor parte de la electrónica, las baterías, procesadores y, si existen, cámaras y sensores. Normalmente es la parte más importante de un dron, ya que si se daña, condiciona seriamente su funcionamiento.
8. Variadores: Conocidos en inglés como Electronic Speed Controllers (ESC) son una parte cuya función es convertir la energía de una batería de corriente continua en corriente alterna trifásica para alimentar los motores. En la mayoría de los drones suele tener uno conectado a cada motor y se suelen encontrar en el cuerpo central.
9. Controlador de vuelo: Es una placa que permite interpretar los datos recibidos mediante los distintos elementos del dron. Permite realizar la mayoría de las funciones del dron, como podrían ser regular la velocidad, la dirección, activación de cámaras, etc.
10. Módulo GPS: Permite obtener información sobre la ubicación geográfica del dron. No todos los drones tienen este componente, sin embargo cualquier dron medianamente moderno lo posee.
11. Receptor: Parte que permite obtener señales de radio, normalmente las que implican el control del dron. Como mínimo un dron necesita cuatro canales, uno para cada dirección (adelante, atrás, izquierda, derecha).
12. Antena: Permite recibir y emitir señales de radiofrecuencia. Normalmente está acoplada al receptor y son fácilmente reemplazables.
13. Batería: Sin este componente, el dron no puede funcionar. Normalmente son de polímero de litio.
14. Controlador de la batería: Facilita información del estado de la batería, de manera que el usuario puede conocer y estimar el tiempo restante de vuelo.
15. Gimbal: También conocido como cardán, es una parte opcional de los drones que permite acoplar al mismo cámaras o sensores. Es una pieza que permite crear una estabilización sobre los tres ejes, de manera que se puedan capturar imágenes de forma más sencilla.
16. Motor del gimbal: Permite al gimbal realizar los movimientos mecánicos para estabilizar el elemento acoplado en cualquiera de los tres ejes.
17. Unidad de control del gimbal: Permite controlar el gimbal como si fuera un servomotor.

18. Elementos extra: En este caso es una cámara, pero pueden ser perfectamente sensores.

2.1.3 Controlador de vuelo

Para poder utilizar y controlar un dron es necesario utilizar un controlador de vuelo que permita conectar al usuario con el dron y realizar acciones. Estos son algunos ejemplos de los distintos controladores de vuelo disponibles:

2.1.3.1 Ardupilot

Ardupilot[6] es un controlador de vuelo de código abierto que permite la creación de software de control autónomo de vehículos. Permite el uso de drones multirrotor, helicópteros, Rovers, barcos y submarinos.

2.1.3.2 PX4

PX4[4] es un piloto automático de código abierto orientado a aviones autónomos de bajo costo. Permite funcionar un gran número de drones, desde drones de carga y de carreras hasta drones de uso civil.

En la siguiente imagen podemos observar el aspecto de un controlador de vuelo PX4:

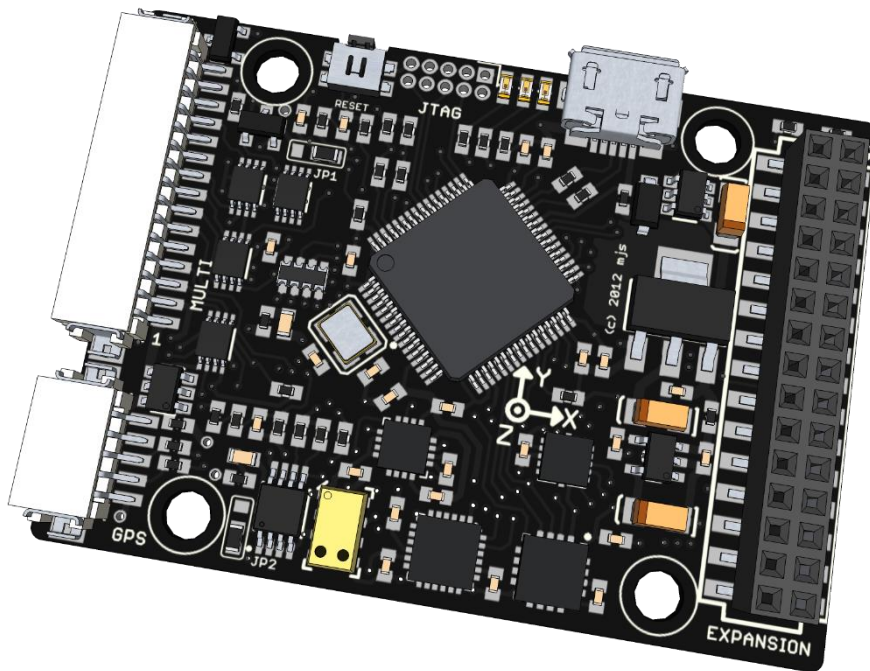


Ilustración 1: Piloto automático PX4

2.1.3.3 MavSDK

MavSDK[5] es un conjunto de librerías para varios lenguajes de programación para drones, cámaras o vehículos de tierra. Tiene una API simple para manejar uno o varios vehículos de forma simultánea, y se pueden utilizar tanto en el software de un dron como en un ordenador desde tierra.

En este proyecto se usará PX4 para controlar el vuelo del dron y también MavSDK para controlar la cámara. Tanto PX4 como MavSDK tienen una documentación muy buena, bien organizada y simple, por lo que esa es la razón por la que se han usado en detrimento de Ardupilot, que tiene una comunidad más dividida. Además, PX4 tiene una gran compatibilidad con AirSim, lo que permitirá realizar simulaciones más realistas y que sean más parecidas al entorno real.

2.2 Simulación de drones

Uno de los principales objetivos de este proyecto es crear un entorno simulado lo más realista posible, para poder utilizar el mismo software en un entorno real. Para ello necesitamos un simulador que permita acercar la simulación lo máximo posible a la realidad. En este apartado trataremos todos los elementos que permiten tener una simulación lo más realista posible.

2.2.1 AirSim

AirSim[7] es un simulador de drones, coches y más dispositivos hecho para Unreal Engine (también cuenta con una versión experimental para Unity). Es de código abierto, multiplataforma y tiene soporte para controladores de vuelo como PX4 o Ardupilot e incluso soporte para simulaciones realistas con PX4.

AirSim destaca por su facilidad tanto de instalación como de uso. Además, cuenta con una documentación muy interesante que explica paso a paso cómo poner en marcha un proyecto de manera muy sencilla.

2.2.2 CARLA

Carla[8] es un simulador de sistemas autónomos que permite el desarrollo, entrenamiento y validación de estos. Es de código abierto y tiene una gran cantidad de elementos digitales como entornos urbanos, edificios o vehículos, creados para ser usados de manera gratuita.

Su plataforma de simulación tiene mucha flexibilidad de cara a sensores, condiciones medioambientales, generación de mapas y mucho más.

Aunque CARLA tenga una buena base para trabajar, la opción elegida para este proyecto es AirSim principalmente por su compatibilidad con PX4, ya que facilitaría bastante el pasar a un entorno real cuando las simulaciones lo permitan.

2.3 Motor Gráfico

Debido a que en este proyecto se va a utilizar AirSim, se va a necesitar un motor gráfico que permita simular correctamente todas las físicas y generar los entornos adecuados para que AirSim funcione correctamente.

Un motor gráfico[9], también conocido como motor de juego, consiste en una serie de rutinas de programación que permiten el diseño, creación y funcionamiento de (normalmente) un videojuego, pero también entornos de simulación o renderizados.

Incluye funcionalidades para renderizar en 2D y 3D, simular físicas (gravedad y colisiones principalmente), animaciones, sonidos y mucho más.

Existen una gran variedad de motores gráficos como pueden ser Unreal Engine, Unity, Source 2 Engine, CryEngine, Frostbite y mucho más.

En este apartado vamos a hablar principalmente de tres:

2.3.1 Unreal Engine

Motor gráfico creado por Epic en 1998, de uso libre desde 2015 (siempre que el proyecto resultante sea de código libre). Escrito sobre C++, Unreal presenta un alto grado de portabilidad y es una de las herramientas más utilizadas para el desarrollo de videojuegos[10].

Existen cinco grandes versiones de este motor gráfico:

1. Unreal Engine 1: Primera versión, lanzada en 1998, integraba renderizado, detección de colisiones, IA, opciones de red y manipulación de archivos de sistema. Para su época se consideraba un motor bastante completo.
2. Unreal Engine 2: Lanzado en 2002, reescribió completamente el motor de renderizado, además de incluir el SDK de Karma physics.
3. Unreal Engine 3: Lanzado en 2006, reescribe de nuevo su motor de renderizado para soportar técnicas como HDRR (High-Dinamic-Range Rendering), mapeados y sombras dinámicas. Además sustituye Karma por PhysX y añade FaceFX para generar animaciones faciales.
4. Unreal Engine 4: Lanzado oficialmente en 2015, fue el primer motor de Epic que estaba disponible de manera gratuita, de manera que Epic obtendría un 5% de los proyectos que se comercialicen, siempre y cuando los beneficios del mismo sean de más de 3000 dólares por trimestre. Es la versión actual y la más extendida, pues el acceso a la misma es tan simple como acceder a la Epic Games Store (tienda de videojuegos de Epic, disponible tanto para MacOS como para Microsoft Windows) y descargarlo.
5. Unreal Engine 5: Es aún una versión de prueba, en Early Access (Acceso anticipado) desde mayo de 2021 y con su primera versión experimental lanzada en mayo de 2022. Es una mejora del motor que permite generar entornos tan detallados que parecen reales, con unas físicas casi perfectas y que destaca por el manejo de partículas.

2.3.2 Unity

Motor gráfico creado por Unity Technologies, lanzado en mayo de 2005, está disponible en Microsoft Windows, MacOS y Linux. En su origen fue lanzado como un motor de desarrollo para sistemas MacOS, y a partir de Unity 3 (lanzado en 2010) tiene herramientas para el resto de los sistemas operativos. Actualmente se encuentra en la versión 5 (desde 2015) y no es un motor completamente gratuito[11].

Sus licencias permiten usarlo de manera gratuita de manera individual, con planes de pago anuales para equipos y empresas.

2.3.3 Source Engine

Es un motor gráfico desarrollado por Valve Corporation para las plataformas Microsoft Windows, Mac OS, GNU/Linux, Xbox y PlayStation. Debutó en 2004 de la mano del famoso videojuego Counter-Strike: Source.

La ideología de desarrollo de Valve con Source[12] consiste en crear un motor que evolucione poco a poco mientras la tecnología avanza, de manera que se actualice de manera constante, sin cambios bruscos entre versiones. Además se tiene en cuenta que está directamente ligada a Steam (plataforma de videojuegos propiedad de Valve), que permite que las actualizaciones del motor se descarguen automáticamente y todos los jugadores tengan un acceso rápido a las nuevas versiones.

Source ha tenido dos grandes versiones: Source y Source 2. Esta última salió en 2015 y es una versión considerablemente más potente que su predecesora.

Este motor gráfico es gratuito para el desarrollo de videojuegos, principalmente para mods (modificaciones de un juego no oficiales), dando acceso a su SDK de manera gratuita siempre que se adquirieran cualquiera de sus productos que lo utilicen vía Steam, como puede ser Team Fortress 2, Half-Life 2 o Counter-Strike.

Debido a su compatibilidad con AirSim, la opción elegida en este proyecto es Unreal Engine, en concreto su versión 4.27.

2.4 IA

Para este proyecto se van a necesitar varios procesos que deben ser ejecutados de forma automática. Estos procesos son principalmente el reconocimiento de señales y el procesamiento de imágenes, para lo cual será necesario el uso de Inteligencias Artificiales (IA).

La inteligencia artificial es la disciplina que intenta replicar y desarrollar la inteligencia y sus procesos implícitos a través de computadores. No se puede hacer una definición completa de la inteligencia artificial, pero se basa en cuatro puntos: Una IA es un sistema que piensa como un humano, actúa como un humano, piensa racionalmente y actúa racionalmente.

La Inteligencia artificial abarca muchos subcampos, entre los que destacan el aprendizaje y la percepción.

Tal como dicen Stuart J. Russell y Peter Norvig, se podrían diferenciar cuatro tipos de inteligencia artificial:

1. Sistemas que piensan como humanos: Tratan de emular el pensamiento humano mediante la automatización de actividades como toma de decisiones, resolución de problemas o aprendizaje. Un ejemplo de este tipo serían las redes de neuronas artificiales.
2. Sistemas que actúan como humanos: Son aquellos que tratan de emular el comportamiento humano. Un caso de este tipo es la robótica, que pretende que los computadores realicen tareas igual o mejor que un ser humano.

3. Sistemas que piensan racionalmente: Son aquellos que mediante la lógica pretenden imitar el pensamiento racional del ser humano. Un ejemplo de este tipo de IA serían los sistemas expertos. Estos consisten en sistemas informáticos que emulan el razonamiento, actuando tal y como lo haría un experto en cualquier área de conocimiento.
4. Sistemas que actúan racionalmente: Son aquellos que tratan de emular de forma racional el comportamiento humano. Un ejemplo son los agentes inteligentes. Estos permiten percibir su entorno, procesar esa percepción y responder en consecuencia a ella, tratando de maximizar el resultado esperado.

Estos son los elementos de IA que se van a utilizar en este proyecto:

2.4.1 Computer visión

Computer Vision o visión artificial es un subcampo de machine learning que enseña a los ordenadores a “ver” y entender el contenido de imágenes digitales. Las aplicaciones que este subcampo puede aportar son muy numerosas, como por ejemplo: Automoción (reconocimiento de señales), seguridad (reconocimiento facial), salud (reconocimiento de tumores) o Banca (reconocimiento de datos).

Existen multitud de herramientas que permiten usar computer vision:

2.4.1.1 OpenCV

OpenCV[13] es una biblioteca open source de computer vision desarrollada inicialmente por Intel. Significa Open Computer Vision. Destaca por ser de código abierto bajo licencia BSD (permite su uso libre para propósitos comerciales y de investigación), es multiplataforma para los sistemas GNU/Linux, Mac OS, Microsoft Windows y Android y tiene una documentación completa y concisa.

Totalmente desarrollada en C++, orientada a objetos, tiene una API principal en C++ pero con conectores a otros lenguajes como Python, Java, Matlab, Octave o Javascript.

Surgió en 1999, pero su primera versión salió en versión alfa en el año 2000. Intel fue la encargada de sus 2 primeras versiones oficiales en 2006 (1.0) y 2009 (2.0). En agosto de 2012 delegó el proyecto a la fundación OpenCV.org, una fundación sin ánimo de lucro que mantiene su desarrollo y sitio web desde entonces.

2.4.1.2 SimpleCV

SimpleCV[14] es una herramienta de código abierto para el desarrollo de aplicaciones de visión artificial, que permite tener acceso a librerías como OpenCV sin la necesidad de aprender en profundidad cómo funciona.

En un principio era una alternativa útil a OpenCV, pero debido a la facilidad de uso actual de OpenCV en la mayoría de sistemas, no es necesariamente la mejor herramienta de uso.

2.4.1.3 Google Cloud Vision API

Es una herramienta en la nube de Google que permite analizar imágenes y extraer información valiosa para comprender su contenido. Proporciona una interfaz RESTful

que facilita la tarea de desarrollar algoritmos de procesamiento de imágenes. Tiene diferentes características orientadas a la detección de distintos tipos de rasgos. Por ejemplo: Face Detection permite reconocer caras humanas, Landmark detection reconoce estructuras populares o Text detection extrae texto de imágenes[15].

Su principal desventaja es que el tamaño de las imágenes no debe superar los 4MB, que en la actualidad puede llegar a ser muy poco tamaño para una imagen de calidad estándar.

La opción elegida para la realización de este proyecto ha sido OpenCV. Aunque en primera instancia lo lógico sería utilizar cualquiera de las otras dos opciones, la más sencilla era OpenCV. En este proyecto no se utilizará mucho esta herramienta más que para el procesamiento de imágenes, con varias líneas de código muy sencillas bastante bien explicadas en la documentación, por lo tanto, no sería necesario el uso de SimpleCV. Además, la calidad de las imágenes a utilizar podría provocar que no todas las imágenes se limiten a pesar menos de 4MB como pide Google Cloud Vision API. De esta manera, la única opción válida era OpenCV.

2.4.2 Detectron

Detectron[16] es una plataforma de código abierto enfocada en la detección de objetos y su segmentación. Ha sido desarrollada por Facebook AI Research (FAIR) para dar soporte a la implementación y evaluación de visión artificial. Incluye la implementación de múltiples algoritmos de detección de objetos como Mask R-CNN, RetinaNet, RPN y muchos más.

Utiliza herramientas como Pytorch (Framework de código abierto para Deep learning), CUDA (API de computación paralela de NVIDIA) o OpenCV (ya explicada en el apartado 2.4.1.1).

El proyecto de Detectron comenzó en 2008 y ha pasado por muchas versiones, siendo mucho más accesible e intuitivo desde 2019.

2.5 Entorno de desarrollo

Para entender el desarrollo del proyecto, es necesario hablar de todos los elementos que conforman el entorno de desarrollo del proyecto, desde el sistema operativo recomendable hasta el lenguaje de programación.

2.5.1 Sistema operativo

En el desarrollo de este proyecto se van a utilizar dos sistemas operativos: Windows y Linux bajo Windows Subsystem for Linux (WSL). La razón de esta decisión es simplemente por la comodidad del desarrollo, pues algunos elementos del desarrollo como Unreal están optimizados para su funcionamiento en Windows, mientras que otros como PX4 o Mavsdk están más optimizados para Linux. Usando WSL podremos facilitar la comunicación entre el sistema Windows y el sistema Linux y usar ambos simultáneamente.

2.5.1.1 Windows

Windows es un sistema operativo de ordenadores (y durante un tiempo un sistema operativo de dispositivos móviles en los Windows Phone) creada y desarrollada por

Microsoft. Históricamente ha sido el sistema operativo dominante del mercado mundial de ordenadores, llegando a tener aproximadamente un 70% de la cuota de mercado[17].

Actualmente se encuentra la versión Windows 11, introducida de manera oficial en octubre de 2021 como una actualización gratuita en todos los equipos con Windows 10 y con los requisitos necesarios para actualizarlo.

Sin embargo, la versión en la que se ha realizado este proyecto es Windows 10, lanzada en julio de 2015 como una actualización gratuita para aquellos equipos con Windows 8 con requisitos para ser actualizados (al igual que harán con Windows 11). La razón para esta decisión es que en el momento del inicio del desarrollo de este proyecto, aunque ya existía Windows 11 de manera oficial, la versión no era excesivamente estable y sabiendo que Windows 10 era perfectamente estable, no era necesario actualizar, aunque este proyecto se podría ejecutar perfectamente en un sistema con Windows 11.

2.5.1.2 Linux

Linux es un sistema operativo compuesto de software libre y código abierto. Este sistema operativo surge de las contribuciones de distintos proyectos de software. Aunque se utilice el término Linux, realmente Linux es solo el núcleo del sistema operativo, ya que el sistema completo está formado por compiladores y entornos de ejecución. Linux tiene muchas distribuciones como Debian, Ubuntu, Manjaro o muchas más[18].

En este proyecto se usará Ubuntu 20.04 mediante la aplicación de la Windows Store Ubuntu 20.04 LTS. Además, se utilizará Windows Subsystem for Linux (WSL)[19]. WSL permite ejecutar entornos de Linux directamente desde Windows sin la necesidad de utilizar máquinas virtuales o tener configuraciones de arranque dual. Es muy útil, pues permite compartir archivos entre ambos sistemas de manera muy sencilla y evita sobrecargar el sistema al usar dos sistemas operativos al completo simultáneamente.

2.5.2 Lenguaje de programación

Este proyecto se va a realizar en el lenguaje operativo de Python. Además se usará el editor de código de Visual Studio Code [20](creado por Microsoft) ya que permite programar en WSL de manera que se pueda tener ejecutando en una terminal de Visual Studio el subsistema de Linux que se usa en el proyecto gracias a las distintas extensiones que permite instalar de manera gratuita.

2.5.2.1 Python

Python[21] es un lenguaje de programación multiparadigma utilizado para desarrollar aplicaciones de todo tipo. Soporta parcialmente la orientación a objetos y es un lenguaje multiplataforma.

Gestionado y administrado por Python Software Foundation, es un lenguaje de código abierto bajo la licencia Python Software Foundation License. Surge a finales de los ochenta por Guido van Rossum en los Países Bajos. Su versión más longeva y conocida fue Python 2.7, aunque fue discontinuada en enero de 2020 y su versión actual con más soporte es Python 3.6, que será además la empleada en el desarrollo de este proyecto.

2.5.3 Google Colaboratory

Google Colaboratory [22], más conocido como Colab, es un producto de Google Research que permite la ejecución de código dentro de un navegador. Entre sus principales ventajas se encuentra que no requiere ningún tipo de instalación previa y permite el acceso a recursos computacionales de gran potencia, incluyendo GPUs.

Colab tiene varios modelos de negocio: una versión gratuita con almacenamiento muy limitado sin recursos garantizados y con un uso máximo de 12 horas del entorno; otra llamada Colab Pro que por 9.25 euros al mes permite un mayor almacenamiento, garantizando el acceso a un entorno con GPU si fuera necesario (además de ser GPUs más potentes) y hasta 24 horas de uso; y por último Colab Pro+ que por 42.25 euros al mes permite ejecutar procesos en segundo plano con las GPUs más rápidas, con un almacenamiento aún mayor y durante aún más tiempo.

La razón de utilizar Colab en este proyecto es simple: mi ordenador no dispone de una gráfica de NVIDIA, que son gráficas que permiten utilizar CUDA, un programa que permite realizar el entrenamiento de un modelo de Detectron y sin el cual no es posible. Por lo tanto he buscado una manera de poder utilizar Detectron sin la necesidad de comprarme un ordenador nuevo, o cambiar la gráfica de mi ordenador actual. Ha sido necesario utilizar Google Colab Pro debido al tamaño del dataset de señales de tráfico (aproximadamente 45 GB) del que hablaremos en el apartado de diseño e implementación. Además, Google Colab permite acoplar Google Drive al entorno, de manera que no necesitamos subir los 45GB desde mi ordenador cada vez que queramos entrenar el modelo.

2.6 Aplicaciones Similares

Este proyecto no es el primero en tratar la detección de señales mediante el uso de modelos. Existen algunas aplicaciones que realizan una función similar. En este subapartado se hablará brevemente de ellas, explicando los puntos en común con este proyecto y las principales diferencias.

2.6.1 Mapillary

Mapillary [23] es una plataforma que permite acceder a imágenes de las calles de todo el mundo de manera sencilla y permite usar esa información para distintos usos como mapeado, detecciones o análisis. Proporciona acceso a diversos datasets con gran cantidad de imágenes, como por ejemplo señales de tráfico, coches y mucho más.

Los principales puntos en común con este proyecto son que permite tener un mapeado de las calles de una zona y obtiene en bases de datos la información de posición de señales y sentidos de las carreteras (este proyecto se centra en la detección de señales únicamente).

Las diferencias son más claras. Mapillary utiliza diversos elementos para mapear las calles (personas, coches...) pero nunca usa drones. Además, Mapillary es más completa en el sentido de que tiene bases de datos no sólo de señales, sino de pasos de cebra, cruces, coches y mucho más.

Mapillary ha sido una base importante de este proyecto, pues la idea del mismo surge de intentar mejorar el comportamiento de esta aplicación a la hora de mapear. Además, el dataset utilizado para el entrenamiento del modelo ha sido obtenido de la página de Mapillary [24] y ha sido extremadamente útil para el desarrollo del proyecto.

2.6.2 Google Street View

Google Street View [25] es una representación virtual del entorno de Google Maps formada por millones de imágenes panorámicas. Permite visualizar prácticamente cualquier lugar del mundo con una vista panorámica.

El principal punto en común con este proyecto es la intención de generar vistas panorámicas del entorno que rodea a las cámaras.

Sin embargo, de ahí también radican las principales diferencias, como el uso de coches en vez de drones, o la cantidad de cámaras que llega a usar un coche de Google (pueden llegar a superar por bastante las 6 cámaras que se usarán para este proyecto).

Google Street view ha sido una aplicación en la que se ha basado este proyecto, pues la idea de utilizar imágenes panorámicas en lugar de capturar imágenes de una o dos cámaras deriva de esta aplicación. En este proyecto se va a intentar simular imágenes panorámicas mediante el uso de diversas cámaras utilizando 6 cámaras (como se puede observar en la Figura 5: Posición Cámaras dron).

3 Entorno socioeconómico y legal

En este apartado se va a tratar el posible impacto tanto social como económico que tendría la implementación de este proyecto. Además, se va a comentar también el marco regulador bajo el cual este proyecto se basa.

3.1 Impacto socioeconómico

Este proyecto puede tener distintas implicaciones si es llevado a la realidad. En primer lugar, este proyecto podría mejorar enormemente la seguridad de las carreteras, pues permitiría realizar amplios barridos de carreteras en los que, al detectar una señal o no, se podría valorar el estado de esta y si necesita mantenimiento. También permitiría llegar por vía aérea a carreteras de difícil acceso, como carreteras de montaña, y evitaría poner en peligro vidas humanas.

Desde el punto de vista puramente económico los drones no son excesivamente caros y montar un par de drones adaptados para esta tarea podría ser muy beneficioso en rentabilidad. Además, posiblemente crearía más empleos de los que destruiría al automatizar esta tarea, pues sería necesario como mínimo una persona dedicada al mantenimiento del dron, otra para el control y mantenimiento del software y una más para la valoración del estado de las señales tras las misiones.

Aunque también hay que tener en cuenta que en las ciudades el posible rendimiento de este proyecto sea menor, pues la normativa española de drones, de la que se hablará en el siguiente subapartado, no permite el vuelo de drones en zonas urbanas si el usuario se encuentra a más de 500 metros, por lo que no sería posible realizar grandes barridos de distancia pero sí permitiría un mapeado más exhaustivo de las señales y su posición, quizá para determinar la posible utilidad de las mismas (por ejemplo, al detectar que dos señales iguales se encuentran muy cerca, igual se podría valorar eliminar una de ellas).

3.2 Marco regulador

La normativa española de drones se basa en la normativa Europea de drones, actualmente aplicando los Reglamentos Europeos RE 2019/947 y RD 2019/945 disponible en el BOE[26]. Entre las medidas respecto al uso de drones destacan las siguientes:

- Las aeronaves no tripuladas. Independientemente de su masa, pueden utilizarse en el mismo espacio aéreo que las aeronaves tripuladas.
- Los operadores de aeronaves no tripuladas deben registrarse si utilizan aeronaves no tripuladas dotadas de un sensor que pueda captar datos personales, teniendo en cuenta el riesgo que ello supone para la privacidad y la protección de dichos datos. Este registro no será necesario si la aeronave no tripulada se considera un juguete según la Directiva 2009/48/CE del Parlamento Europeo y del Consejo, sobre la seguridad de los juguetes.
- Los operadores y los pilotos a distancia de UAS deben asegurarse de estar adecuadamente informados de las normas de la Unión y nacionales aplicables a las operaciones previstas, especialmente respecto a seguridad, protección, privacidad, protección de datos, responsabilidad, seguros y protección del medio ambiente.

- Determinadas zonas pueden ser sensibles a algunos tipos de operaciones de UAS, por lo que los Estados miembros pueden establecer normas nacionales para someter a ciertas condiciones la realización de operaciones con UAS en esas zonas.

En España, el organismo encargado de velar para que se cumplan las normas de aviación civil es la Agencia Estatal de Seguridad Aérea (AESA)[27]. Esta se encarga de supervisar, inspeccionar y ordenar el transporte aéreo, navegación aérea y seguridad aeroportuaria, evalúa los riesgos de la seguridad de transporte y tiene potestad sancionadora. Está adscrita a la Secretaría de Transportes, Movilidad y Agenda Urbana.

Para poder volar un dron existe una normativa que se debe cumplir independientemente de su uso (recreativo o profesional), estos son sus puntos más básicos:

- Todo usuario que pretenda volar un dron debe registrarse como operador en la sede electrónica de AESA y obtener un número de operador según la normativa europea, que debe incluirse en el dron de forma visible.
- Para volar un dron se debe tener un mínimo de formación acreditable en función de la categoría operacional en la que se opere. La formación y examen correspondiente es accesible desde la web de AESA de manera telemática y gratuita.
- Se debe de disponer de manera obligatoria de un seguro de responsabilidad civil, tanto para vuelos recreativos como profesionales.
- El vuelo de drones está sujeto a reglas generales de operación condicionada, como peso del dron, presencia de otras personas o cercanía a edificios.
- Existen limitaciones al vuelo de drones en ciertos lugares, entre las que se encuentran cercanía a aeródromos, bases militares o hospitales.

Este proyecto también respeta el Reglamento General de Protección de Datos, un reglamento a nivel europeo relativo a la protección de personas físicas respecto a sus datos personales y libre circulación de sus datos aplicado en España desde diciembre de 2018, concretamente el Reglamento 2016/679 de la UE. Este BOE se conoce bajo la Ley Orgánica 3/2018 del 5 de diciembre[28].

Concretamente este proyecto, aun estando diseñado para una simulación, se puede aplicar al mundo real y por lo tanto al capturar las imágenes pueden aparecer personas, por lo que se compromete a tratar las imágenes para preservar la privacidad de las posibles personas que aparezcan en ellas acorde al Reglamento 2016/679 anteriormente nombrado.

Por último indicar que las licencias de uso de todos los programas y software usados para el desarrollo de este proyecto. Todos los programas y software tienen una licencia open source para desarrollo personal siempre que no se comercie con ello, y al ser un proyecto de investigación universitario cumple con la normativa de uso open source.

4 Diseño de la solución

En este apartado se va a discutir el diseño propuesto para el proyecto mediante un diagrama de flujo que muestre los distintos componentes del diseño. Además, se explicarán todas las clases que componen este diseño, acompañado de imágenes del código para facilitar el entendimiento del mismo.

4.1 Arquitectura del sistema

En este subapartado se va a proponer un diagrama de flujo para cada sección del proyecto que explica cómo se conectan los distintos elementos del mismo, junto con una breve explicación de la función de los mismos y como se conectan entre ellos.

Para facilitar la comprensión, las tres secciones son las siguientes:

Resaltado en gris se encuentran los elementos que han sido producidos por mí, en blanco los programas auxiliares o pasos intermedios.

Parte de Ubuntu:

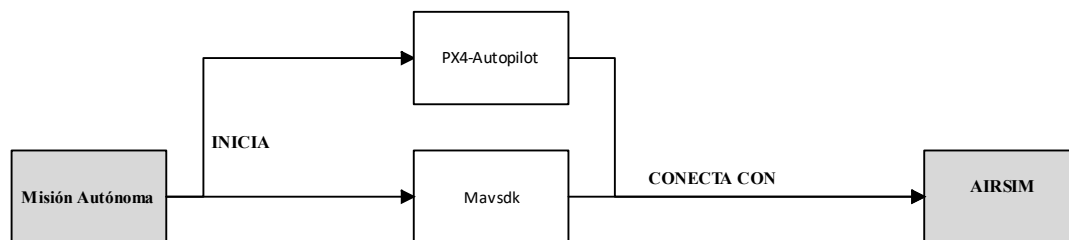


Figura 2: Diagrama de componentes Ubuntu

Parte de Windows:

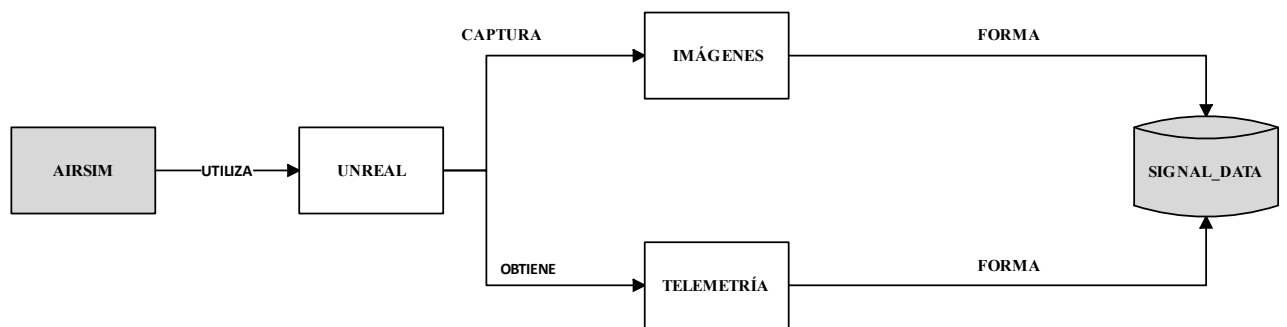


Figura 3: Diagrama de componentes Windows

Parte de Google Colab:

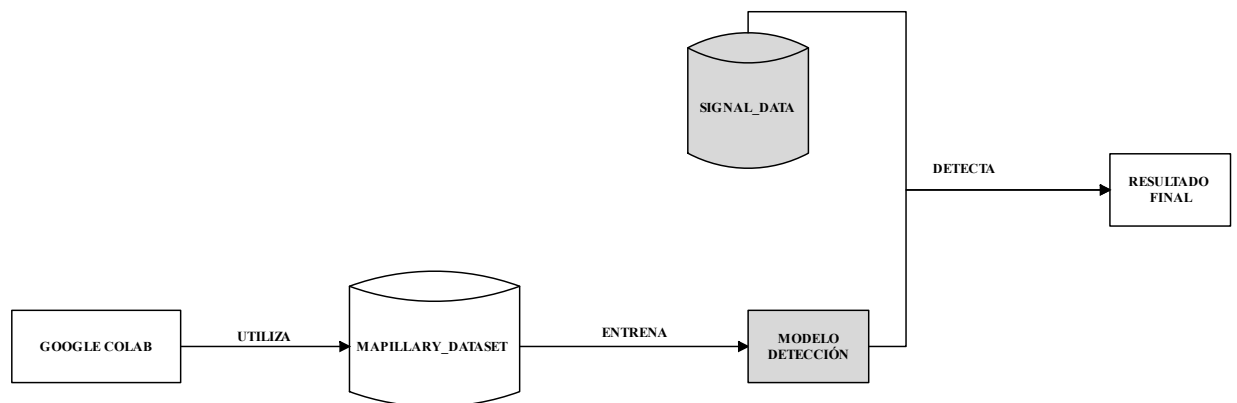


Figura 4: Diagrama de componentes Colaboratory

4.1.1 Mi Misión

Este componente es la misión que realizará el dron. Para este proyecto he realizado una misión que consiste en recorrer una carretera con un dron de forma autónoma en la que se va deteniendo y capturando imágenes en ciertos puntos. Debe recorrer una carretera con una o más señales de tráfico para su posterior análisis. Permite iniciar PX4 y MavSDK.

4.1.2 PX4

Piloto automático que debe recibir de la misión los datos de los puntos por los que debe pasar el dron durante su recorrido. Se conecta a AirSim gracias a MavSDK.

4.1.3 MavSDK

Permite realizar la conexión entre el sistema y el dron y su conexión con el piloto automático. Se conecta a AirSim junto con PX4.

4.1.4 AirSim

Es el simulador de drones utilizado para el proyecto, permite generar drones funcionales y que sean controlados mediante PX4. Recibe los datos tanto de la misión como del controlador de vuelo y realiza la simulación.

4.1.5 Unreal

Motor gráfico que permite que AirSim realice la simulación. Tiene un entorno creado en el que se desarrolla la simulación de AirSim.

4.1.6 Detectron

Recibe las imágenes generadas por la simulación en AirSim y las procesa para detectar las señales que haya en las mismas. Previamente se debe entrenar un modelo para su detección.

4.2 TrafficSignDetection

En este subapartado se va a hablar de todas aquellas clases que componen la misión diseñada, incluyendo también el fichero JSON que permite obtener las características concretas del dron de la simulación.

Mi misión autónoma consiste en recorrer desde el punto A hasta el punto B de una carretera, deteniéndose en determinados puntos de control o checkpoints, capturando una imagen en cada uno de ellos con las cámaras equipadas en el dron. Para intentar imitar la realidad lo máximo posible, se ha equipado al dron con un total de 6 cámaras que intentarán cubrir la totalidad de los alrededores del dron con el menor número de puntos ciegos. En la simulación de AirSim no es posible mostrar las cámaras físicamente, por lo que para explicar la distribución de las cámaras voy a disponer de una imagen de un dron genérico similar al usado en la simulación, marcando la posición de las cámaras y la visión que proporcionan las mismas:

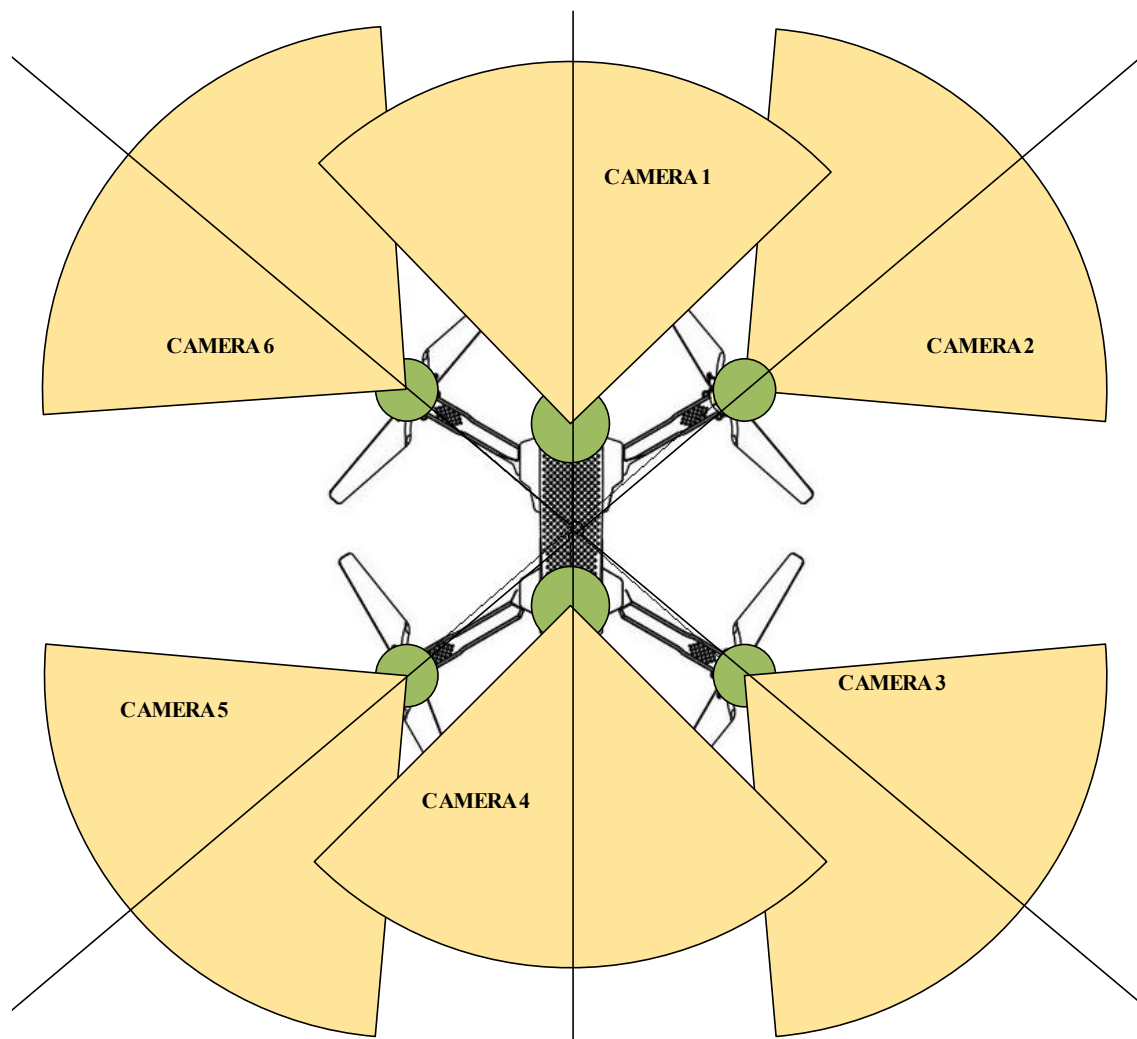


Figura 5: Posición Cámaras dron

Como se puede observar en la Figura 5: Posición Cámaras dron, hay 6 cámaras distribuidas por todo el dron; una en la parte frontal, otra en la trasera y una en cada uno de los brazos. De esta manera se genera una visión casi de la totalidad de los alrededores del dron.

La misión consta de varias clases. A continuación, se explicarán una a una todas ellas, incidiendo en los fragmentos de código más relevantes con imágenes.

4.2.1 exampleSettings.json

Este fichero JSON contiene los atributos básicos del dron que se va a usar en la simulación. Los atributos más relevantes son los siguientes:

SimMode es un atributo que permite definir el tipo de vehículo que se va a usar en la simulación, en este caso al ser un dron cuadricóptero, el atributo será Multirrotor.

Vehicles es un atributo que contiene una lista con todos los vehículos que se van a usar en la simulación, en este caso solo será un dron, al que llamaremos Drone1.

Drone1 es el nombre del dron, y su campo contiene toda la información necesaria del mismo. Entre los campos más importantes se encuentra el número de cámaras y las propiedades de estas (se pueden destacar posición respecto al dron, rotación, inclinación u opciones de captura), la IP del controlador, el puerto por el que se comunica (normalmente TCP sobre UDP), una lista de sensores o su posición de origen. Un ejemplo de opciones del dron se puede encontrar en Ilustración 2.

```
"Camera1": {  
  "CaptureSettings": [  
    {  
      "AutoExposureBias": 0.0,  
      "AutoExposureMaxBrightness": 0.64,  
      "AutoExposureMinBrightness": 0.03,  
      "AutoExposureSpeed": 100.0,  
      "FOV_Degrees": 90.0,  
      "Height": 1080.0,  
      "MotionBlurAmount": 0.0,  
      "OrthoWidth": 5.12,  
      "ProjectionMode": "perspective",  
      "TargetGamma": 1.0,  
      "Width": 1920.0  
    }  
  ],  
  "Gimbal": {  
    "Pitch": 0.0,  
    "Roll": 0.0,  
    "Stabilization": 0.0,  
    "Yaw": 0.0  
  },  
  "NoiseSettings": [  
    {  
      "Enabled": false,  
      "HorzDistortionContrib": 1.0,  
      "HorzDistortionStrength": 0.0,  
      "HorzNoiseLinesContrib": 1.0,  
      "HorzNoiseLinesDensityXY": 0.5,  
      "HorzNoiseLinesDensityY": 0.01,  
      "HorzWaveContrib": 0.03,  
      "HorzWaveScreenSize": 1.0,  
      "HorzWaveStrength": 0.08,  
      "HorzWaveVertSize": 1.0,  
      "ImageType": 0.0,  
      "RandContrib": 0.2,  
      "RandDensity": 2.0,  
      "RandSize": 500.0,  
      "RandSpeed": 100000.0  
    }  
  ],  
  "Pitch": 0.0,  
  "Roll": 0.0,  
  "X": 0.0,  
  "Y": 0.0,  
  "Yaw": 0.0,  
  "Z": -2.0  
},
```

Ilustración 2: Ejemplo de Opciones de cámara

4.2.2 **mainNoGUI.py**

Esta clase se ha diseñado de manera que pueda funcionar como la clase principal de cualquier misión autónoma diseñada para drones, tanto para múltiples drones simultáneos, como para uno solo.

En primer lugar, se define el tipo de misión que queremos ejecutar. A continuación, hay un elif con las distintas misiones implementadas. Cada misión tiene sus propios campos, pero todas tienen varios campos en común, estos son origin, dest, arrUAVs y una variable Drone por cada uno de los drones participantes en la misión.

Origin es una variable en la que se encuentra el path a exampleSettings.json, que es un archivo JSON con los datos del dron necesarios para que AirSim simule un dron con las características que necesitamos.

Dest es el path a la ubicación del archivo settings.json de AirSim. Gracias a estas dos variables podemos utilizar la librería shutil para copiar nuestras settings en AirSim de manera automática antes de cada ejecución, de forma que no tenga que hacerse manualmente cada vez que se cambia algo.

arrUAVs es un array con el nombre de cada dron participante en la misión, este array comienza vacío.

Después están las variables Drones. Al ser solo un dron, solo existirá la variable Drone1, que contiene todos los datos necesarios del Dron a utilizar, dispuesta en distintos campos. General contiene los datos básicos del dron: nameDrone para su nombre, missionDrone para el nombre de la misión a ejecutar, Latitude para la latitud inicial del dron y Longitude para la longitud; Mission contiene los datos relativos a la misión del dron, OverlapPercent para la precisión de la distancia a los puntos de control, RelativeAltitude para la altura sobre el nivel del suelo a la que se posiciona, PointsLat son los puntos de inicio y final de la Latitud, PointsLon son los puntos de inicio y final de la Longitud y por último el campo Cameras contiene la información de todas las cámaras de las que dispone el dron.

```
Drone1 = {
  'General':{
    'nameDrone': 'Drone1',
    'missionDrone': 'trafficSign',
    'Latitude' : 40.54479102105834,
    'Longitude': -4.012444057860788,
  },
  'Mission':[
    'OverlapPercent' : 15,
    'RelativeAltitude' : 2,

    'PointsLat': [40.542550, 40.543686],
    'PointsLon': [-4.010950, -4.010950],

  ],
  'Cameras':{
    'Camera1':{
      'General':{
        'CameraName': 'Camera1',
      },
      'Capture':{
        'FOV_Degrees': 90,
      }
    },
    'Camera2':{
      'General':{
        'CameraName': 'Camera2',
      },
      'Capture':{
        'FOV_Degrees': 90,
      }
    },
    'Camera3':{
      'General':{
        'CameraName': 'Camera3',
      },
      'Capture':{
        'FOV_Degrees': 90,
      }
    },
    'Camera4':{
      'General':{

```

Ilustración 3: Ejemplo de configuración del dron

A continuación, comienza la preparación para comenzar a ejecutar la misión. En primer lugar se crea un fichero en el que se guardarán todos los logs. Si este ya existe, se elimina su contenido.

Después abre el fichero `px4_mavsdk_multiple.sh`, que es un script que conecta PX4 y MavSDK a AirSim. Una vez se ha conectado y ha recibido el mensaje de All running se asigna el Home position del dron en función de los datos establecidos en cada uno de los drones y lanza `missionHandler`, clase de la que hablaremos a continuación y que se ocupa de controlar el desarrollo de la misión.

4.2.3 missionHandler.py

Esta clase se ocupa de conectar correctamente con AirSim y preparar la ejecución de la misión.

Para ello primero espera la respuesta de la conexión de MavSDK y PX4, para a continuación crear un log llamado misión_handler.log. Este log contiene los datos más relevantes de la misión detallados tales como la distancia hasta el siguiente punto de control, el momento en el que captura la imagen y cuándo termina la misión. Una vez creado, utilizará el parámetro del nombre de la misión obtenido de la clase anterior para preparar la ejecución de la misión.

En este caso la misión será trafficSign y los preparativos que hará serán principalmente actualizar la posición de inicio de la misión, establecer el margen de distancia en los puntos de control, extraer los distintos sensores del dron para la misión (en este caso las cámaras), determinar el número de drones que van a ejecutar la misión y calcular los distintos puntos de control que va a tener la misión mediante la clase mapTools, de la que se hablará más adelante.

```
class DroneGeneral:
    def __init__(self, missionDrone):
        # Se extraen los parámetros del array inicial
        OverlapPercent = float(drone['Mission']['OverlapPercent']) / 100
        RelativeAltitude = float(drone['Mission']['RelativeAltitude'])
        # El área para la ortomosaico
        PointsLat = drone['Mission']['PointsLat']
        PointsLon = drone['Mission']['PointsLon']
        positions = []
        for j, pointlat in enumerate(PointsLat):
            positions.append([PointsLat[j], PointsLon[j]])
        # Se extraen los sensores
        droneName = drone['General']['nameDrone']
        camerasName = []
        lidarName = []
        for j, cameraName in enumerate(drone['Cameras']):
            camerasName.append(drone['Cameras'][j]['CameraName'])
        cameraFOV = float(drone['Cameras'][0]['Capture']['FOV_Degrees'])
        for j, lidarName in enumerate(drone['Lidar']):
            lidarName.append(lidarName)
        # Se calcula cuantos drones van a hacer esta misión, para repartir
        numUAVsTrafficSign = 0
        posThisUAVTrafficSign = 0
        for j, droneAux in enumerate(self.arrUAVs):
            if droneAux['General']['missionDrone'] == 'trafficSign':
                numUAVsTrafficSign = numUAVsTrafficSign + 1
                if i == j:
                    posThisUAVTrafficSign = numUAVsTrafficSign - 1
            else:
                continue
        # Se calculan los puntos en donde realizar fotos
        lat = float(drone['General']['Latitude'])
        lon = float(drone['General']['Longitude'])
        mTools = mapTools()
        mTools.groundpoints2checkpoints(positions=positions, drone_pos=[lat, lon], overlap=OverlapPercent, relative_altitude=RelativeAltitude, fov=cameraFOV, arrUAVs=numUAVs)
        setpoints = mTools.checkpoints(posThisUAVTrafficSign) # Brinda los checkpoints de cada drone
        # Envía la partición (y su trayectoria correspondiente) de este dron a JS, para pintarlo
        if self.usingDjango == True:
            message = {
                'type': 'messageToJS',
                'message': 'trafficSign_data',
                'drone_id': i+1,
                'geo_checkpoints': mTools.geo_checkpoints[posThisUAVTrafficSign],
                'geocoor_square': mTools.geocoor_square,
            }
            await self.channel_layer.group_send(self.groupName, message)
        # Inicializa la misión
        mission = trafficSign(drone_id = i+1, droneName=droneName, RelativeAltitude=RelativeAltitude, camerasName=camerasName, setpoints= setpoints)
```

Ilustración 4: Procedimiento de missionHandler.py si la misión es trafficSign

Por último, inicializa la misión llamando al main de la clase trafficSign, que será la próxima clase de la que se va a hablar. En caso de que hubiera varios drones o varias misiones a ejecutar, existe un bucle que permite lanzar cada misión en un thread distinto.

4.2.4 trafficSign.py

Esta clase es la que tiene el código correspondiente a la misión. En primer lugar obtiene los atributos pasados por parámetro en el init además de inicializar el log misión_handler.log para esta clase.

El código referente a la misión es el método run, que a su vez llama a tres métodos:

Initialize realiza la conexión con el dron, para ello se enlaza con AirSim y MavSDK, comprueba que MavSDK está inicializado correctamente, inicializa la telemetría del dron y lo arma. Una vez ha realizado estas comprobaciones establece la posición inicial y va a ella.

Mission va recorriendo los distintos checkpoints y captura una imagen por cámara en cada uno de ellos. Para ello coge la latitud y longitud del checkpoint y manda el dron a esa posición. Mediante un bucle comprueba la distancia hasta el checkpoint y cuando es lo suficientemente pequeña, se detiene y estabiliza el dron. Una vez estabilizado realiza la imagen con el método snapshot_with_metadata. Este método pide a cada cámara que capture una imagen con el método de AirSim simGetImages, que permite realizar varias llamadas a ImageRequest a la vez. ImageRequest permite obtener una imagen de lo que está viendo la cámara en ese momento. Una vez tiene todas las imágenes, guarda en ella la fecha, latitud y longitud y las convierte en metadatos de la imagen. A continuación, guarda la imagen con un nombre concreto, que consiste en el nombre del dron seguido de la fecha y por último la cámara que ha capturado esa imagen (un ejemplo de nombre podría ser DroneX_AAAADDMM_CameraX.jpg). Una vez que ha completado de hacer y guardar las imágenes en este checkpoint, asigna el siguiente y comienza el mismo proceso hacia el siguiente checkpoint.

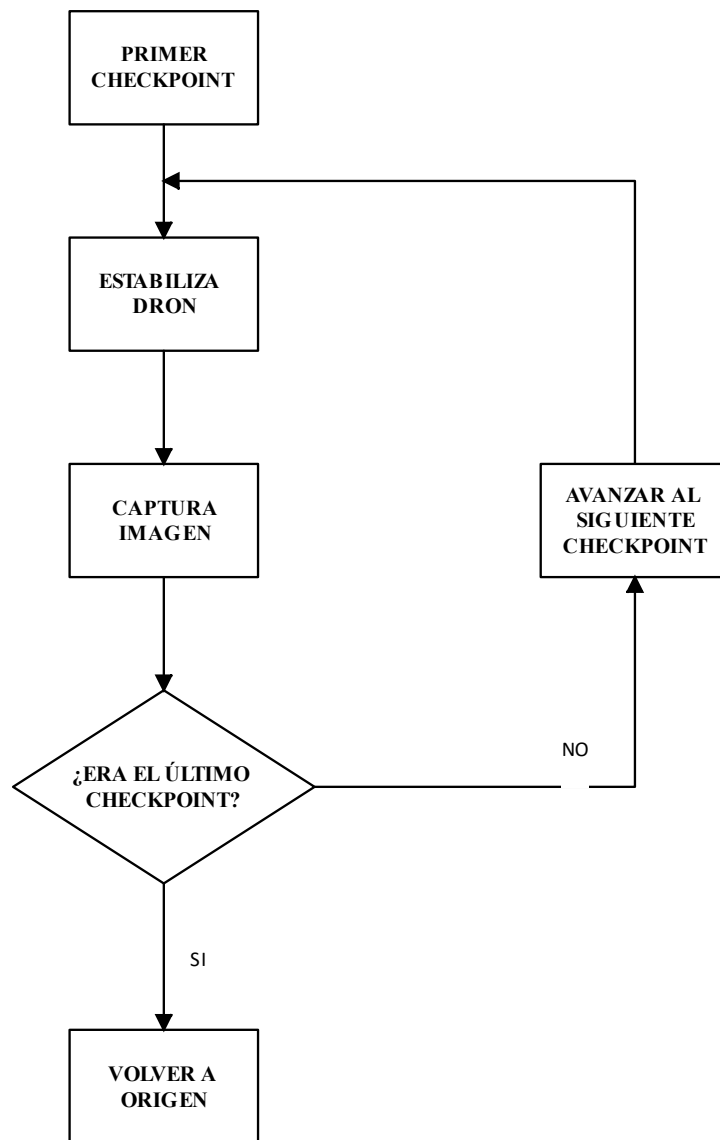


Figura 6: Procedimiento de checkpoints

Ending es el método que se ejecuta una vez ha llegado al último checkpoint. Este método hace que el dron vuelva a su posición de origen, aterrice y por último cierra la telemetría y la conexión con MavSDK.


4.2.5 mapTools.py

Esta clase es la que traza los checkpoints a recorrer por el dron para su misión. La manera en la que funciona MavSDK y PX4 en AirSim hace que, aunque tengamos coordenadas de latitud y longitud de la posición en el mundo real, estos necesitan convertir esas coordenadas a coordenadas locales en el eje XYZ. Para ello utiliza el método `geoposSquare`. Una vez obtenidas las coordenadas locales, utiliza los datos de las cámaras para determinar los checkpoints necesarios para cubrir toda la distancia (también se puede sustituir este paso creando a mano los checkpoints o utilizando un array de checkpoints obtenido de otra manera).

4.3 Sección de Google Colaboratory

Como ya se comentó en el estado del arte, hay una parte de este proyecto que se ha realizado en Google Colaboratory, Concretamente todo lo relativo al uso del software de Detectron. En este subapartado se explicará todo lo referente al entrenamiento y uso del modelo de Detectron utilizado para analizar las imágenes obtenidas desde AirSim y posteriormente su geolocalización.

Al ser un entorno de ejecución alojado, cada vez que se inicie hay que instalar aquellos elementos que se van a usar. Por lo tanto, la primera parte del código consiste en instalar las dependencias necesarias y Detectron.



```
!python -m pip install pyyaml==5.1
!python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'
import torch, detectron2
!nvcc --version
TORCH_VERSION = ".".join(torch.__version__.split(".")[0:2])
CUDA_VERSION = torch.__version__.split("+")[-1]
print("torch: ", TORCH_VERSION, "; cuda: ", CUDA_VERSION)
print("detectron2:", detectron2.__version__)
```

Ilustración 5: Librerías e instalaciones necesarias para utilizar Detectron

Después de instalar Detectron, hay que preparar el dataset de señales de tráfico que vamos a usar para el modelo. El dataset elegido es el que proporciona Mapillary [24]. Este contiene una gran cantidad de imágenes con señales de tráfico con anotaciones y una cantidad algo menor de imágenes destinadas a la evaluación y validación de los modelos (todas las imágenes que tengan su archivo JSON con las anotaciones en la carpeta train serán las destinadas al entrenamiento del modelo y aquellas que tenga el JSON en la carpeta val serán aquellas destinadas a la validación). El dataset se encuentra comprimido en una carpeta de Google drive debido a su tamaño (aproximadamente 45Gb) y para poder usarlo se debe montar el drive en el entorno y descomprimir el mismo en la carpeta de destino, que se llamará mapillary.

Una vez descomprimido se creará una variable global llamada MAPILLIS_ROOT que será utilizado siempre que se vaya a acceder al dataset de manera más sencilla.

A continuación, se va a crear el metadatacatalog, esto es una recopilación de todas las posibles opciones que se va a enseñar a detectar en este modelo, es decir, se va a recopilar cada una de las señales que se encuentran en el dataset y ponerlas en un archivo para que cada vez que el modelo detecte algo se asegure que está entre las señales de este archivo. Para ello en primer lugar se recorren todas las imágenes y se van contando las distintas señales que encuentran, de momento sin apuntarlas, para saber cuántas hay. En este caso hay 400 señales distintas.

Después se construye el índice de categorías (una por señal) y mediante el método `get_mapillis_dicts` se construye el metadatacatalog, además de guardarlo como JSON para su consulta en cualquier momento.

Una vez se tiene esto, ya se puede comenzar el entrenamiento del modelo. Para este entrenamiento se va a utilizar el entrenamiento por defecto de un modelo[29], usando el `model_zoo` de `retinanet_R_50_FPN_1x` y con el dataset cargado anteriormente.

```
[ ] from detectron2.engine import DefaultTrainer
    from detectron2.config import get_cfg
    from detectron2 import model_zoo
    import os

    # https://detectron2.readthedocs.io/en/latest/tutorials/datasets.html#update-the-config-for-new-datasets
    cfg = get_cfg()
    cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/retinanet_R_50_FPN_1x.yaml"))
    cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/retinanet_R_50_FPN_1x.yaml")
    cfg.DATASETS.TRAIN = ("mtsd_train",)
    cfg.DATALOADER.NUM_WORKERS = 1
    cfg.SOLVER.IMS_PER_BATCH = 1
    cfg.SOLVER.MAX_ITER = num_train_examples
    cfg.SOLVER.BASE_LR = 0.00025
    cfg.SOLVER.STEPS = [] # do not decay learning rate
    cfg.MODEL.RETINANET.NUM_CLASSES = len(class_labels)

    os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
    trainer = DefaultTrainer(cfg)
    trainer.resume_or_load(resume=False)
    trainer.train()
```

Ilustración 6: Ajustes de entrenamiento utilizados para el modelo

Cuando se tenga el modelo entrenado, se puede continuar con la detección de las señales de las imágenes capturadas en la simulación. Primero se crean un par de variables, una con el path al directorio con las imágenes y otra con el path al directorio en la que guardaremos la salida. Después hay que abrir el archivo JSON con todas las posibles señales creado anteriormente y recorrer el directorio con las imágenes. A cada imagen le pasaremos el modelo y se producirán las imágenes de salida con las detecciones. A cada señal detectada se le realizará un proceso para intentar determinar si esa señal ya ha sido detectada anteriormente. Este proceso se puede realizar de manera más automática utilizando algún algoritmo, pero para este proyecto se ha optado por una clasificación más manual realizada de la siguiente manera.

Cada señal diferente detectada va a tener un JSON asociado con los datos necesarios para su geolocalización. Si al detectar una señal ya existe un JSON con el mismo nombre de esa señal, se evaluará si esta puede ser una señal nueva, o es otra detección de una señal ya existente. Teniendo en cuenta que hay un total de seis imágenes en cada checkpoint, se van a tener cuenta conjuntos de seis imágenes para realizar las detecciones. A cada señal detectada en una tanda de imágenes se la va a considerar que no puede estar en ninguna del resto de imágenes de la misma tanda, y se va a guardar su nombre en una lista. Al cambiar de tanda de imágenes la lista se convertirá en la lista de señales de la tanda anterior y, por lo tanto, cualquier señal cuyo nombre esté en la lista, que aparezca detectada en la siguiente tanda debería ser una señal ya existente por lo que en lugar de crear un JSON nuevo con la información de la señal, se añadirán los datos de la imagen de la misma al JSON de la señal ya existente. Este razonamiento se ha hecho asumiendo que no deberían existir dos señales iguales en pocos metros de distancia. Esto podría generar falsos positivos al existir calles con carriles en los dos sentidos como se verá en el apartado 6.1 Implementación, pero es el único razonamiento que se puede usar si no se van a usar algoritmos complejos.

Para facilitar el entendimiento de este razonamiento se va a utilizar el siguiente diagrama de flujo:

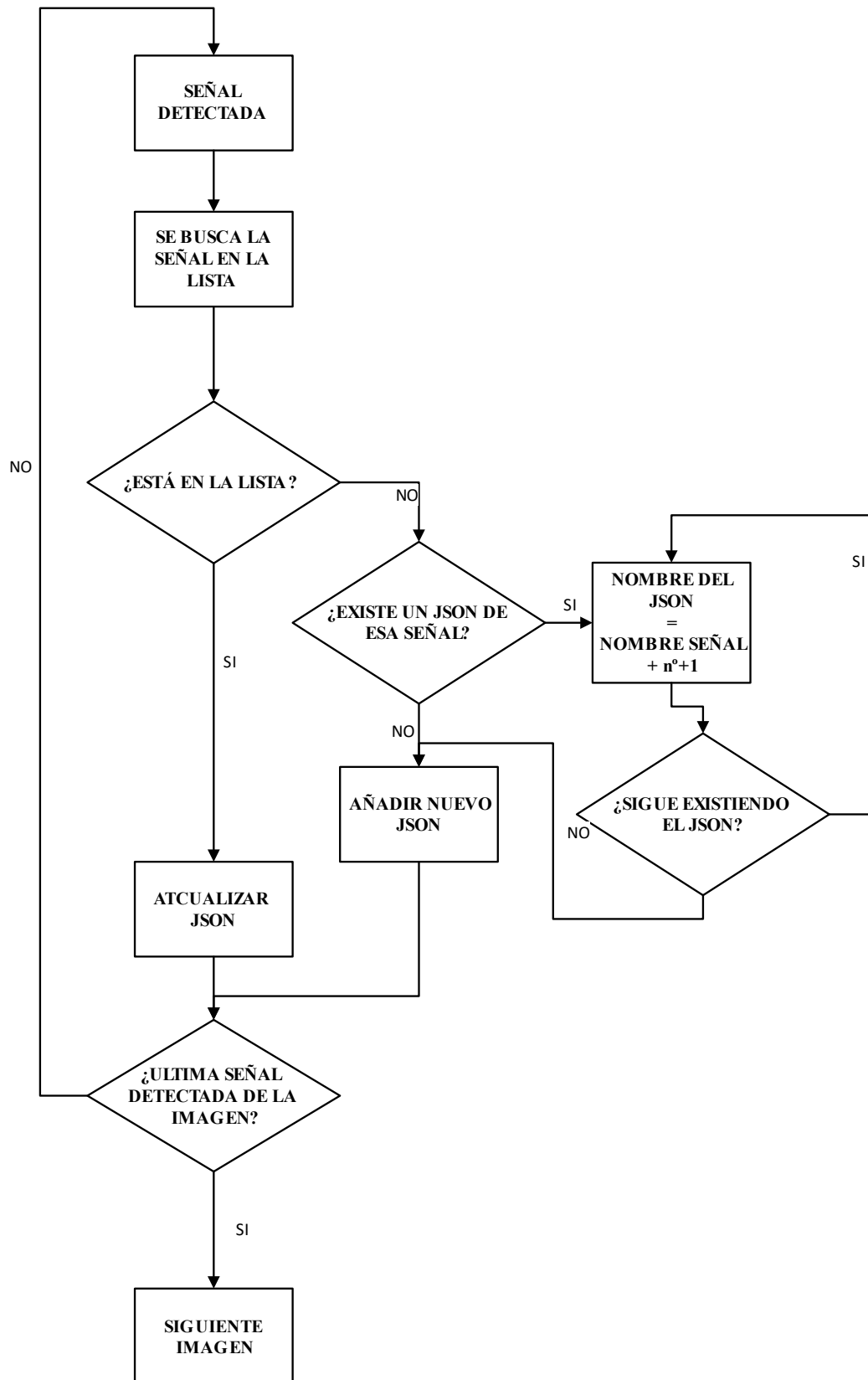


Figura 7: Diagrama detección señales

De esta forma tendremos por salida del proceso de detección una imagen con las señales detectadas por cada imagen analizada y un JSON con los datos de geolocalización para cada una de las señales detectadas, ya haya sido detectada una vez o más (en cuyo caso tendrá un campo de datos por cada detección). Con esto tendríamos una lista de las distintas señales y una posición aproximada de la misma sacada de la posición del dron en el momento de la detección.

5 Planificación

En este apartado se van a tratar todos los aspectos relacionados con la parte más profesional del proyecto, es decir: la planificación del proyecto, los requisitos del proyecto, los casos de uso asociados a ellos y por último el presupuesto del proyecto.

5.1 Planificación

Para el desarrollo del proyecto se han valorado dos modelos de planificación diferentes. En primer lugar se encuentra la metodología del “modelo de prototipos”[30], que consiste en un diseño rápido que representa las características principales del programa que el usuario puede ver y utilizar. De esta manera se puede probar y dar opinión de aspectos como usabilidad, utilidad o rendimiento. Este prototipo se puede modificar en cualquier momento y los resultados de las pruebas deben de anotarse para ayudar al desarrollo del producto final.

En segundo lugar se encuentra la metodología de “modelo en espiral”[31], que se basa dividir las tareas como iterativas en lugar de lineales, de forma que cada fase se realiza varias veces en forma de espiral. De esta forma el proyecto se acerca lentamente al objetivo, pero minimizando el riesgo de fracaso gracias a controles regulares.

Por último, se ha propuesto un “modelo en cascada”[32] que consiste en dividir los procesos en distintas fases del proyecto, pero al contrario que en el “modelo en espiral”, las tareas se ejecutan una sola vez.

Mientras el “modelo de prototipos” tiene grandes ventajas en proyectos de desarrollo que están basados en pruebas constantes (como este caso), la magnitud del proyecto podría ralentizar gravemente al proyecto el buscar diferentes prototipos simultáneos, y el “modelo en cascada” suponía una visión a largo plazo del proyecto, sin opción a revisión de algunos elementos de este. Por lo tanto la mejor opción posible es “modelo en espiral”, que permite realizar las diferentes fases de un modelo de planificación con cada una de las tareas del proyecto, algo que viene muy bien en un proyecto que tiene tareas tan independientes entre sí como este, ya que tenemos por un lado la parte de AirSim y la simulación para obtener las imágenes y por otro lado el entrenamiento y ejecución del modelo para la detección de las señales.

5.1.1 Ejecución del modelo de planificación

La ejecución de este modelo se basa en seguir ciclos de cuatro etapas para cada una de las fases del proyecto, que son las siguientes:

- **Planificación:** Recoger requisitos de usuario y comprobar el alcance de la fase del proyecto. Una vez analizados se establecen los objetivos a cumplir.
- **Análisis:** Analizar los requisitos e identificar riesgos que puedan surgir durante el desarrollo.
- **Implementación:** Desarrollo del programa y de las pruebas para comprobar y validar su correcto funcionamiento.

- **Evaluación:** Comprobar si los riesgos establecidos han sido solventados y si los objetivos se han cumplido antes de pasar al siguiente ciclo del proyecto.

-

El proyecto está compuesto de cuatro fases, las cuales son las siguientes:

- **Preparación del entorno:** En esta fase se prepara el entorno de desarrollo del proyecto, configurando tanto la máquina Linux bajo WSL, el entorno de Unreal con AirSim y la conexión de Linux con Windows.
- **Entrenamiento y validación del modelo de detección:** En esta fase se investiga el mejor método de obtener un modelo para la detección de señales y su funcionamiento.
- **Diseño e implementación de la misión autónoma:** En esta fase se diseña, se desarrolla y se implementa la misión autónoma que ejecutará el dron para obtener las imágenes con las señales en la simulación.
- **Diseño de la clasificación de las señales:** En esta fase se diseñará la ejecución de la detección de las señales y su aproximación de la ubicación.

5.1.2 Diagrama de Gantt

El diagrama de Gantt[33] mostrado a continuación muestra las actividades realizadas durante el proyecto junto con el tiempo dedicado a cada una de ellas. La duración del proyecto comienza el 31 de enero de 2022 y concluye el 7 de septiembre de 2022. El proyecto se extiende en un total de 32 semanas en las que se han trabajado una media de 18 horas por semana, dando lugar a un total de 576 horas.

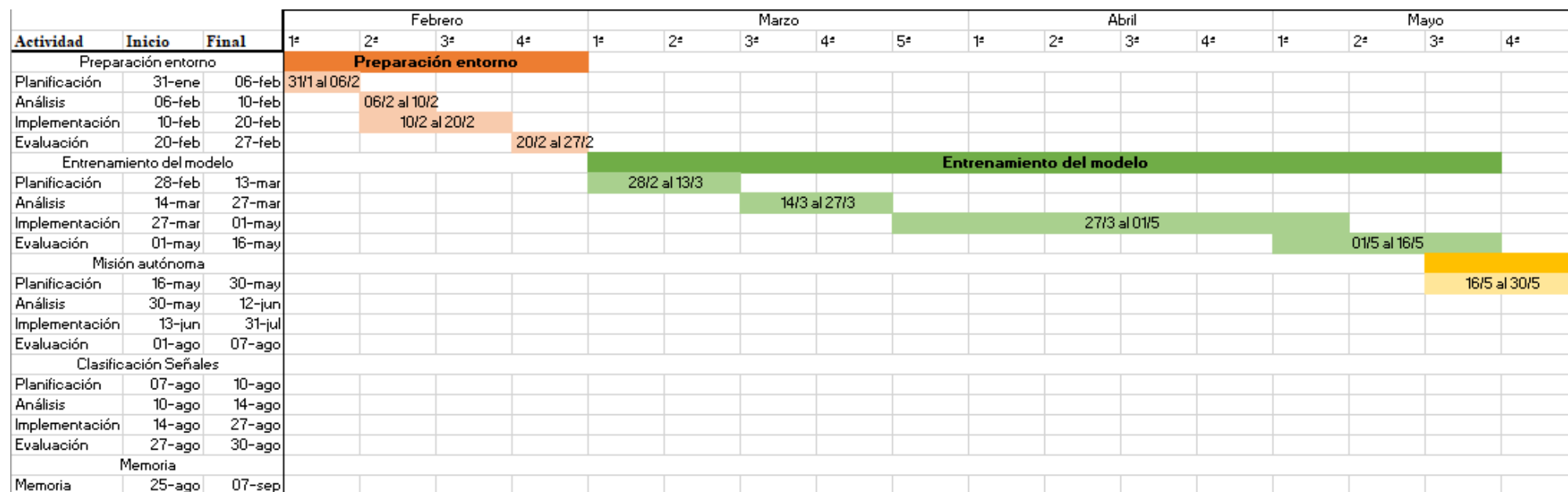


Figura 8: Diagrama de Gantt parte 1

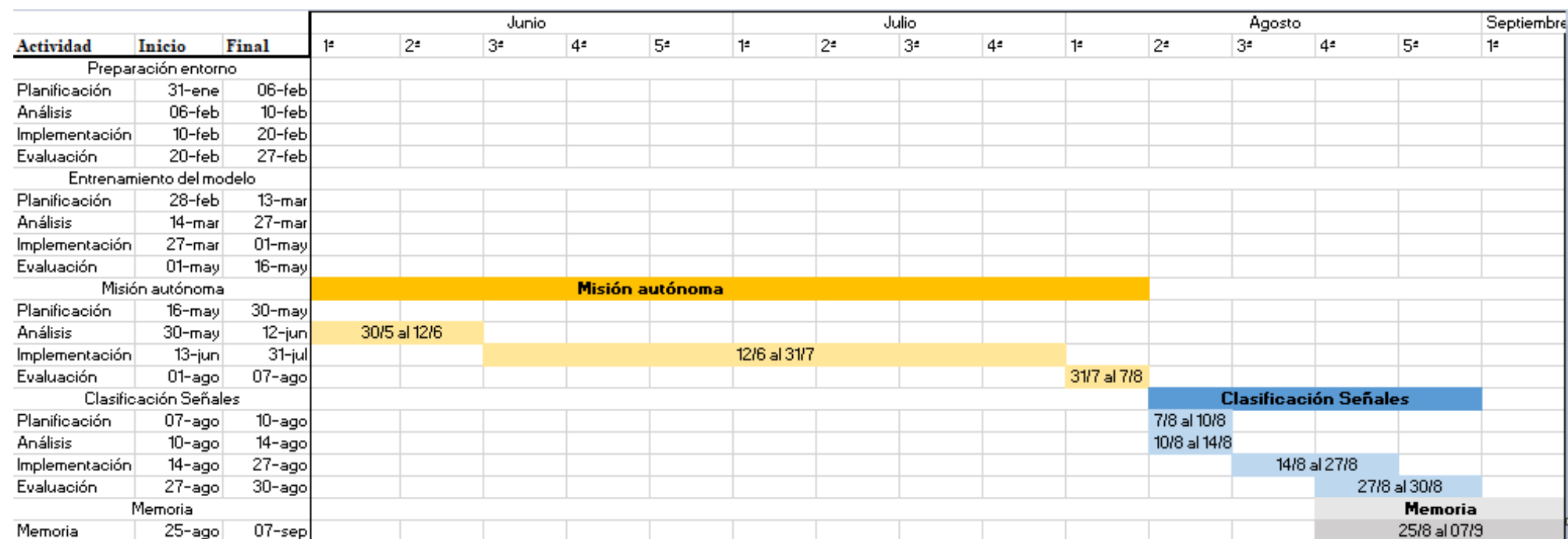


Figura 9: Diagrama de Gantt parte 2

El desglose por horas es el siguiente, asumiendo la media de 18 horas semanales:

- **Preparación del Entorno:**
 - **Planificación:** 18 horas
 - **Análisis:** 10 horas
 - **Implementación:** 26 horas
 - **Evaluación:** 18 horas
- **Entrenamiento del Modelo:**
 - **Planificación:** 36 horas
 - **Análisis:** 36 horas
 - **Implementación:** 106 horas
 - **Evaluación:** 54 horas
- **Misión Autónoma:**
 - **Planificación:** 36 horas
 - **Análisis:** 36 horas
 - **Implementación:** 106 horas
 - **Evaluación:** 18 horas
- **Clasificación de Señales:**
 - **Planificación:** 9 horas
 - **Análisis:** 9 horas
 - **Implementación:** 36 horas
 - **Evaluación:** 22 horas
- **Memoria:** 50 horas

5.2 Requisitos / casos de uso

En este subapartado se van a explicar los requisitos del proyecto, cuya definición sigue las convenciones especificadas y establecidas por el estándar IEEE [34], que indica que la especificación debe ser:

- **Correcta:** Los requisitos deben reflejar un comportamiento que el sistema debe cumplir.
- **No ambigua:** Los requisitos definidos solo pueden tener una interpretación.
- **Completa:** El conjunto de requisitos definidos deben definir el funcionamiento del sistema.
- **Consistente:** Los requisitos no deben contradecirse entre ellos.
- **Clasificables:** Los requisitos deben de estar clasificados por su importancia y estabilidad.
- **Verificable:** Los requisitos deben de tener un proceso que permita su verificación.
- **Modificable:** Los requisitos deben de poder ser modificables de manera sencilla, completa y consistente.
- **Trazable:** Los requisitos deben ser trazables desde su origen y fáciles de trazar desde cualquier momento.

Se usarán tres tipos de requisitos que permitirán definir el comportamiento del sistema. Los tipos de requisitos son los siguientes:

- **Requisitos de usuario:** Son aquellos requisitos que recogen las necesidades que el usuario expresa para alcanzar un objetivo.
- **Requisitos de software funcionales:** Son los que definen los servicios que debe de tener el sistema.
- **Requisitos de software no funcionales:** Son los que definen las restricciones del sistema.
-

La plantilla utilizada para definir los requisitos es la siguiente:

Tabla 1: Plantilla de especificación de requisitos

Número de requisito	Número del requisito. Seguirá el formato RX-YY donde X será el tipo de requisito (usuario, funcional o no funcional) e Y será el número.
Nombre del requisito	Nombre del requisito
Prioridad del requisito	Necesidad del requisito para su cumplimiento.
Estado del requisito	Posibilidad del requisito de cambiar a lo largo del desarrollo.
Descripción del requisito	Descripción breve del requisito

5.2.1 Requisitos de usuario

Tabla 2: Requisito de usuario RU-01

Número de requisito	RU-01
Nombre del requisito	Simulación con AirSim
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El sistema debe ser capaz de iniciar una simulación en Unreal con AirSim

Tabla 3: Requisito de usuario RU-02

Número de requisito	RU-02
Nombre del requisito	Vista del dron
Prioridad del requisito	Alta
Estado del requisito	Invariable
Descripción del requisito	El sistema debe de ser capaz de proporcionar una visualización del recorrido del dron

Tabla 4: Requisito de usuario RU-03

Número de requisito	RU-03
Nombre del requisito	Tecnología de programación
Prioridad del requisito	Alta
Estado del requisito	Invariable
Descripción del requisito	El código de la misión y del modelo de detección será programado usando Python

Tabla 5: Requisito de usuario RU-04

Número de requisito	RU-04
Nombre del requisito	Sistema Operativo
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El sistema debe de poder usarse en Windows 10 junto con Ubuntu 20.04 LTS mediante WSL.

Tabla 6: Requisito de usuario RU-05

Número de requisito	RU-05
Nombre del requisito	Salida de la misión
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	La misión debe devolver correctamente las imágenes obtenidas por la cámara y su JSON asociado en el directorio determinado.

Tabla 7: Requisito de Usuario RU-06

Número de requisito	RU-06
Nombre del requisito	Entrada detección señales
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	Para el entrenamiento del modelo y la detección de señales el programa debe de recibir el dataset de señales

Tabla 8: Requisito de Usuario RU-07

Número de requisito	RU-07
Nombre del requisito	Salida detección
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El programa de entrenamiento y detección debe devolver correctamente las imágenes con las señales detectadas y los JSON asociados a cada una de las señales.

5.2.2 Requisitos de software funcionales

Tabla 9: Requisito de software funcional RF-01

Número de requisito	RF-01
Nombre del requisito	Sensores del dron
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El dron debe de poder determinar su ubicación en todo momento y sacar instantáneas de cualquiera de las cámaras.

Tabla 10: Requisito de software funcional RF-02

Número de requisito	RF-02
Nombre del requisito	Conexión con AirSim
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El programa de la misión debe ser capaz de conectarse correctamente con AirSim.

Tabla 11: Requisito de software funcional RF-03

Número de requisito	RF-03
Nombre del requisito	Estado de la misión
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El sistema debe de poder mostrar información del estado de la misión durante el transcurso de esta.

Tabla 12: Requisito de software funcional RF-04

Número de requisito	RF-04
Nombre del requisito	Información de la imagen
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El sistema debe de poder guardar en un archivo JSON la información relativa a una imagen capturada desde el dron.

Tabla 13: Requisito de software funcional RF-05

Número de requisito	RF-05
Nombre del requisito	Acceso al dataset de señales
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El sistema debe de poder acceder al dataset de señales de tráfico correctamente.

Tabla 14: Requisito de software funcional RF-06

Número de requisito	RF-06
Nombre del requisito	Salida imagen con detecciones
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El sistema debe de poder mostrar por pantalla las imágenes resultantes del modelo de detección de señales.

5.2.3 Requisitos de software no funcionales

Tabla 15: Requisito de software no funcional RNF-01

Número de requisito	RNF-01
Nombre del requisito	Lenguaje de programación
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El proyecto será desarrollado usando el lenguaje de programación Python y debe ser compatible con la versión 3.6.

Tabla 16: Requisito de software no funcional RNF-02

Número de requisito	RNF-02
Nombre del requisito	Compatibilidad con Ubuntu
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El proyecto debe de poder ejecutarse en un sistema Ubuntu con la versión 20.04+.

Tabla 17: Requisito de software no funcional RNF-03

Número de requisito	RNF-03
Nombre del requisito	Dependencias necesarias
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	El sistema utilizado para entrenar el modelo de detección debe tener instaladas las dependencias determinadas en el apartado 4.3.

Tabla 18: Requisito de software no funcional RNF-04

Número de requisito	RNF-04
Nombre del requisito	Versión Unreal
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	La versión de Unreal debe ser la 4.27+.

Tabla 19: Requisito de software no funcional RNF-05

Número de requisito	RNF-05
Nombre del requisito	Compatibilidad versión AirSim
Prioridad del requisito	Alta
Estado del requisito	Invariante
Descripción del requisito	La versión de AirSim debe de ser compatible con la versión 4.27 de Unreal.

5.2.4 Casos de uso

Los casos de uso propuestos son las ejecuciones del proyecto. A continuación se detalla la explicación de los casos de uso mediante la siguiente plantilla:

Tabla 20: Plantilla de casos de uso

Identificador	Número del caso de uso. Sigue el formato CU-XX donde XX es el número del caso de uso.
Nombre del caso de uso	Nombre del caso de uso que se va a hablar
Personal	Tipo de personal destinado para el caso de uso.
Objetivos	Objetivos del caso de uso
Entorno del caso de uso	Elementos del entorno que se deben cumplir para la ejecución del caso de uso.
Salida	Resultados obtenidos tras la finalización del caso de uso
Guía	Serie de pasos para la realización del caso de uso.

Tabla 21: Caso de uso CU-01

Identificador	CU-01
Nombre del caso de uso	Ejecución misión autónoma.
Personal	Usuario.
Objetivos	Ejecutar la misión autónoma y obtener los datos de salida necesarios.
Entorno del caso de uso	El usuario debe de iniciar el proyecto de Unreal con el mapa e introducir los datos del dron que se va a usar para simular.
Salida	Una vez el dron finalice su misión se obtendrá una carpeta con todas las imágenes capturadas por el dron y los archivos JSON con la información de estas.
Guía	<ul style="list-style-type: none"> - El usuario inicia el proyecto de Unreal - El usuario determina las características del dron - El usuario inicia el programa - Se espera a que la misión concluya - El usuario puede acceder a las imágenes y JSON obtenidos durante la misión.

Tabla 22: Caso de uso CU-02

Identificador	CU-02
Nombre del caso de uso	Entrenamiento y ejecución del modelo de detección
Personal	Usuario.
Objetivos	Entrenar el modelo y ejecución de este con las imágenes obtenidas de la misión.
Entorno del caso de uso	El usuario debe de iniciar Google Colab, tener acceso al dataset de entrenamiento y las imágenes capturadas por el dron durante la misión.
Salida	Una vez finalice el proceso, se tendrá un modelo entrenado y una carpeta con las detecciones de las señales y un JSON con la información de la señal por cada señal detectada
Guía	<ul style="list-style-type: none"> - El usuario inicia Google Colab. - El usuario determina la ubicación del dataset y las imágenes de la misión. - El usuario ejecuta el proceso de entrenamiento. - Se espera a que el entrenamiento concluya. - El usuario ejecuta la detección de señales con el modelo entrenado sobre las imágenes de la simulación. - El usuario puede acceder a las imágenes con las detecciones y a los archivos JSON con las distintas señales y los datos de las detecciones.

5.2.5 Matriz de trazabilidad

Una vez establecidos todos los requisitos se puede establecer una matriz de trazabilidad en la que se puede observar la relación entre los requisitos de usuario y los de software. Una buena especificación de requisitos garantiza que todos los requisitos de usuario tienen asociados como mínimo uno de software.

Tabla 23: Matriz de trazabilidad

Requisitos	RU-01	RU-02	RU-03	RU-04	RU-05	RU-06	RU-07
RF-01	●	✓	●	●	✓	●	●
RF-02	✓	●	●	●	●	●	●
RF-03	●	✓	●	●	●	●	●
RF-04	●	●	●	●	✓	●	●
RF-05	●	●	●	●	●	✓	●
RF-06	●	●	●	●	●	●	✓
RNF-01	●	●	✓	●	●	●	●
RNF-02	●	●	●	✓	●	●	●
RNF-03	●	●	✓	●	●	●	●
RNF-04	✓	●	●	●	●	●	●
RNF-05	✓	●	●	●	●	●	●

5.3 Presupuesto

En este subapartado se va a tratar todos los detalles relacionados con el presupuesto del proyecto. En primer lugar, una tabla con las características del proyecto y su coste final, después se desglosarán todos los costes.

Tabla 24: Resumen del proyecto

Título	Detección y geolocalización de objetos automática desde drones.
Autor	Pablo de Alba Martínez
Inicio	31 de enero de 2022
Finalización	7 de septiembre de 2022
Transcurrido	32 semanas
Presupuesto Final	15491,42 €

5.3.1 Costes directos

Los costes directos se desglosan de los trabajadores que han participado en el desarrollo, Los diferentes roles del proyecto se han asignado de la siguiente manera:

- Pablo de Alba Martínez: Planificador, analista, desarrollador y encargado de pruebas.

Para determinar el coste por hora de cada rol, se ha calculado el salario medio de cada profesión[35] dividido entre las semanas de un año (52), teniendo en cuenta una jornada de 40 semanales. No se incluirán las horas dedicadas a la redacción de esta memoria del proyecto.

Tabla 25: Costes de trabajadores

Función	Coste por hora (€)	Horas	Total (€)
Planificador	12,98	99	1285,02
Analista	14,90	91	1355,90
Desarrollador	14,42	274	3951,08
Encargado de pruebas	12,59	112	1410,08
Total			8002,08 €

Costes del equipo y software:

Tabla 26: Costes de equipo y software

Concepto	Coste por unidad (€)	Unidades	Total (€)
PC	900 €	1	900
Google Colab Pro	9,25 €	8 meses	74
Almacenamiento Google Drive (100GB)	1,99 €	8 meses	15,92
Total			989,92 €

Características del ordenador utilizado:

- PC por piezas, Ryzen 5 3600, Radeon RX 480, 16 GB RAM, 500 GB SSD M2 WD BLACK, 2TB HDD BARRACUDA.

5.3.2 Costes indirectos

Los costes indirectos suponen aproximadamente un 18% de los costes directos e incluyen los costes de electricidad, agua, acceso a internet o dietas.

5.3.3 Costes finales

El desglose de costes finales se mostrará en la siguiente tabla:

Tabla 27: Costes finales

Concepto	Coste (€)
Coste de trabajadores	8002,08
Coste de equipo y software	989,92
Costes directos	8992,00
Costes indirectos (18%)	1618,56
Total	10610,56 €

5.3.4 Oferta del proyecto

La oferta propuesta incluirá un margen de riesgos e imprevistos de un 10%, unos beneficios del 10% y el Impuesto de Valor Añadido (IVA) cuyo valor en España es de un 21% [36].

La siguiente tabla muestra el desglose de la oferta:

Tabla 28: Desglose de la oferta

Concepto	Coste (€)	Coste acumulado (€)
Costes finales	10610,56	10610,56
Margen de imprevistos	1061,06	11671,62
Margen de beneficios	1591,58	13263,20
IVA	2228,22	15491,42
Total		15491,42

Una vez aplicados todos los conceptos el coste final del proyecto será de **15491,42 €** (quince mil cuatrocientos noventa y uno euros con cuarenta y dos céntimos).

6 Implementación / Resultados

En este apartado se va a hablar de la implementación del proyecto y un breve análisis de los resultados obtenidos.

6.1 Implementación

En este apartado se va a explicar los distintos pasos para poder ejecutar el proyecto de forma correcta y obtener los resultados esperados. Además, se va a relacionar cada caso de uso con la implementación.

En primer lugar, hay que determinar los requisitos del sistema en el que se ejecute la simulación de AirSim y los requisitos de la licencia de Google Colab Pro necesaria para el entrenamiento del modelo de detección.

Requisitos del sistema:

- Sistema Operativo: Windows 10 o Windows 11 con acceso a la Microsoft Store y a la aplicación Ubuntu LTS 20.04.
- Aplicaciones necesarias: Visual Studio Code, Windows Subsystem for Linux, Unreal 4.27+ (versión 4.27 o más), AirSim, MavSDK, PX4 autopilot y Python/Anaconda.
- Librerías necesarias: Channels, numpy, opencv y pymap3d.
- Internet: Para poder ejecutar la parte de Google Colab es necesaria una conexión estable a internet.
- Mapa de Cesium: Será necesario tener un mapa de Cesium compatible con tu versión de AirSim.

Todas las librerías necesarias se pueden obtener mediante pip/conda (en función de si se esta usando Python o la distribución de Anaconda/Conda) menos OpenCV que se obtiene mediante apt-get.

Una vez el sistema está listo para la ejecución se procede a ajustar los datos del dron en mainNoGUI.py. Como se explica en el apartado 4.2.2 se modifica el Dron deseado con los datos que se quieran obtener (Los campos que se suelen modificar son Latitude y Longitude, que es la posición inicial del dron, y PointsLat y PointsLon, que son los puntos de latitud y longitud de inicio y final del dron). Opcionalmente se puede cambiar el array de checkpoints de la clase maptools.py si se tiene un array con los puntos concretos por los que se desea que pase el dron.

La ejecución de la simulación implica la ejecución de la misión autónoma.

La misión autónoma está diseñada para realizar un recorrido determinado en la configuración del dron en mainNoGUI.py. Durante el recorrido, el dron se parará en ciertos puntos de control, o checkpoints, en los que captura una imagen con cada una de sus cámaras. Una vez el dron recorre el último checkpoint volverá a su punto de origen y dará por finalizada la misión.

En el ejemplo determinado para la implementación se ha configurado para que recorra una calle del campus de Colmenarejo de la Universidad Carlos III de Madrid. En la siguiente imagen se puede ver el punto de inicio, el lugar en el que da la vuelta el dron y el lugar final.



Figura 10: Recorrido del Dron Implementación.

Para colocar las señales de tráfico en el entorno de simulación se han seguido las notas del dataset de Mapillary [37] y en la siguiente imagen se pueden ver las diferentes señales que se han añadido a la simulación (tendrá un tick ✓ si la señal está representada y una X si no lo está):

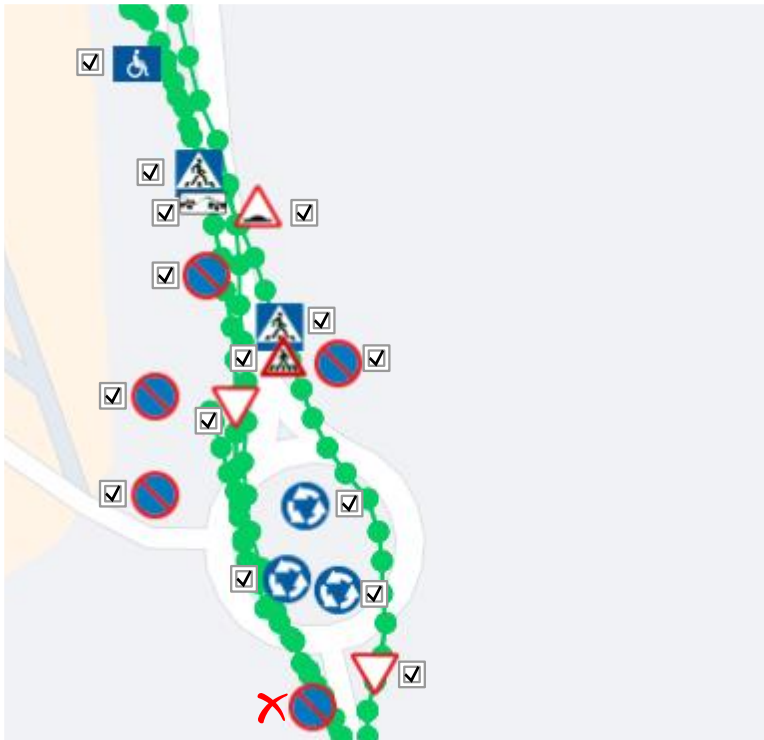


Figura 11: Señales de Mapillary en la zona de ejemplo

La señal de prohibido aparcar que no está en la simulación se debe a que al estar de espaldas al recorrido del dron, nunca se iba a detectar, por lo que se ha optado por no ponerla en la simulación.

En la siguiente imagen se pueden observar las señales que existen en el mapa simulado por AirSim (Esta imagen ha sido sacada desde el entorno de simulación de Unreal):



Ilustración 7: Señales implementación

Si se cuentan las distintas señales que se pueden ver en esta imagen se obtienen los siguientes datos:

- En el interior de la rotonda se encuentran 3 señales de rotonda.
- En el carril derecho se encuentran una señal de prohibido aparcar, una señal de cuidado paso de cebra, otra de aviso de paso de cebra y al fondo un aviso de peligro de elevación del asfalto.
- En el carril izquierdo (señales que se ven de espaldas por la posición de la cámara) se encuentran un aviso de parking de minusválidos, una señal de aviso de paso de cebra, una señal de aviso de grúa, un prohibido aparcar y una señal de ceda el paso.
- En una de las salidas de la rotonda se encuentra una señal de ceda el paso.

El total de señales es de trece señales que corresponden a un total de ocho señales distintas.

La ejecución de la simulación permite probar el caso de uso explicado en Tabla 21: Caso de uso CU-01 se han de seguir los siguientes pasos:

1. Iniciar Unreal con el proyecto en el que se encuentre el mapa
Para ello se debe ir a la tienda de Epic Games Launcher, que abrirá la Epic Games Stores, una vez allí seleccionar el apartado de Unreal Engine e iniciar la versión correspondiente (en este caso la 4.27.2). Una vez dentro seleccionar el proyecto deseado.

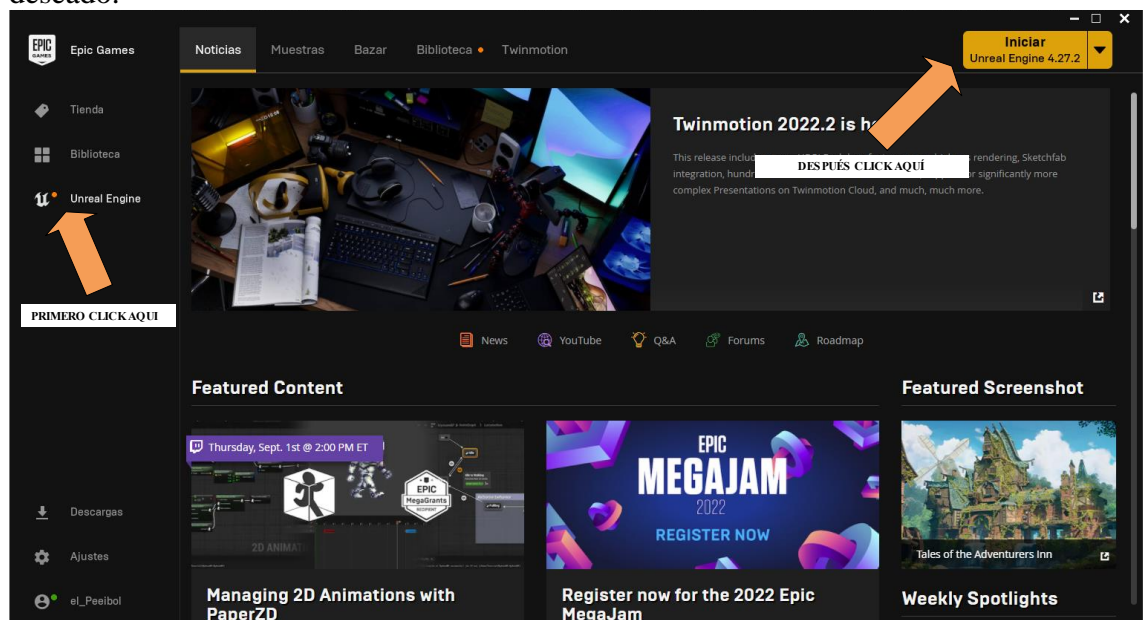


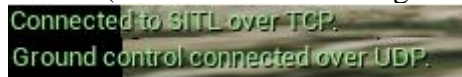
Ilustración 8: Pasos inicio Unreal

2. Iniciar el programa con la clase mainNoGUI.py
Se puede configurar un archivo launch.json que lance el programa automáticamente con un click desde Visual Studio, de forma más sencilla. Este es la información que habría que poner el mismo:

```
{  
  "name": "mainTrafficSign",  
  "type": "python",  
  "request": "launch",  
  "program": "${workspaceFolder}/droneSim/mainNoGUI.py",  
  "console": "integratedTerminal",  
  "args": [  
    "trafficSign"  
  ],  
}
```

Ilustración 9: Configuración de launch.json

3. Esperar a que se realice la conexión de MavSDK y PX4 con AirSim.
Cuando la conexión se ha establecido, aparecerán los siguientes mensajes en Unreal (No necesariamente seguidos):



Connected to SITL over TCP.
Ground control connected over UDP.

Ilustración 10: Mensajes de conexión satisfactoria

4. Observar desde la pantalla de Unreal como el dron realiza la misión
En las siguientes imágenes se puede observar diversas imágenes que muestran el recorrido del dron:

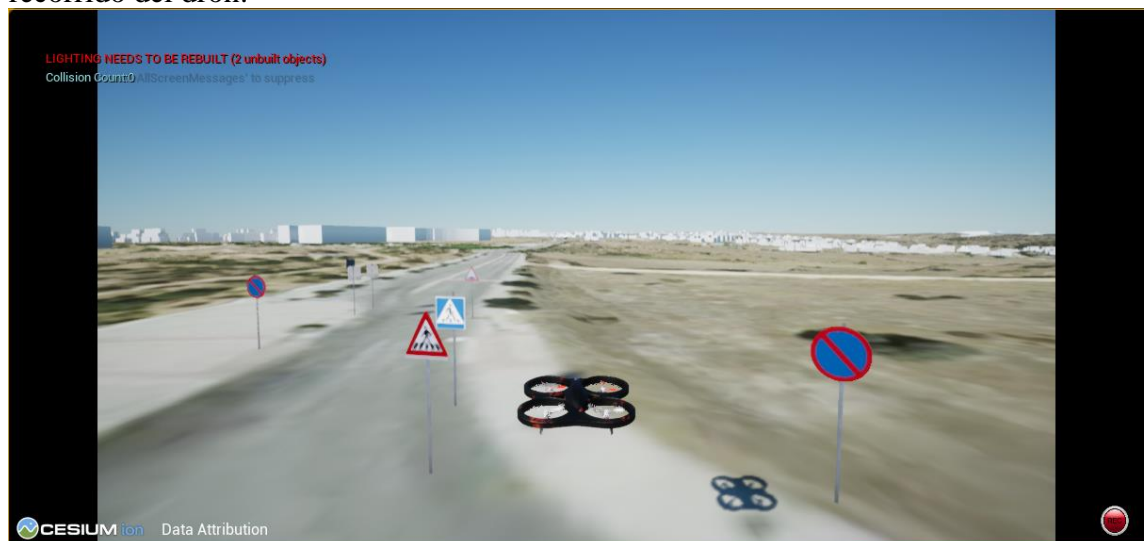


Ilustración 11: Vuelo del dron desde AirSim 1



Ilustración 12: Vuelo del dron desde AirSim 2

Y en las siguientes imágenes se pueden observar algunas imágenes captadas por el dron:



Ilustración 13: Ejemplo imagen capturada por el dron

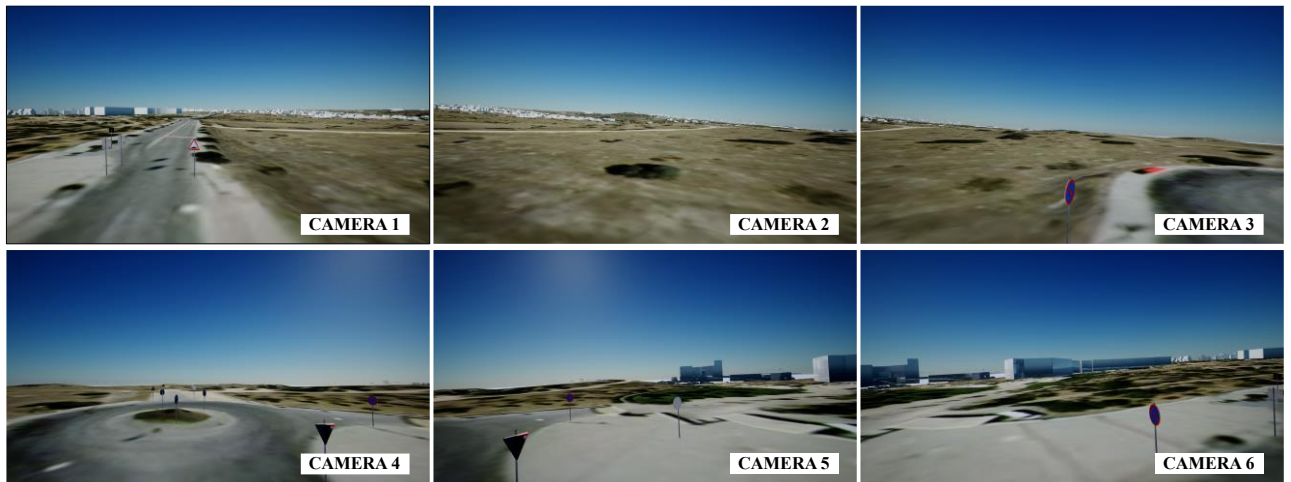


Ilustración 14: Collage de imágenes capturadas por dron

Esta última ilustración representa todas las imágenes capturadas por un dron en un checkpoint concreto.

5. Opcionalmente se puede consultar el log de mission_handler.log que detalla los pasos por puntos de control. Este se encuentra en droneSim/logs
El mission_handler.log tiene los datos de la conexión con AirSim y después muestra los distintos pasos por los checkpoints:

```
INFO:root:MissionHandler Starting mission handler
INFO:root:MissionHandler X Starting missions
INFO:root:1 Starting run
INFO:root:1 Initializing drone
INFO:root:1 X Connecting with AirSim
INFO:root:1 ✓ Connecting with AirSim
INFO:root:1 X Connecting with MavSDK
INFO:root:1 MavSDK port: 14540
INFO:root:1 ✓ Connecting with MavSDK
INFO:root:1 ✓ MavSDK global_position is ok
INFO:root:1 X Setting variables
INFO:root:1 ✓ Setting variables
INFO:root:1 X Arming drone
INFO:root:1 ✓ Arming drone
INFO:root:1 X Setting initial point
INFO:root:1 X Starting offboard mode
INFO:root:1 ✓ Starting offboard mode
INFO:root:1 Starting mission
```

Ilustración 15: Conexión con AirSim desde mission_handler.py

```
INFO:root:1 ✓ Completed setpoint no. 5
INFO:root:1 📦 Move to next setpoint: no.6: X: 0.0, Y: 20.0
INFO:root:1 📦 Current distance: 3.8277116694873334
INFO:root:1 📦 Current distance: 3.5385073260622555
INFO:root:1 📦 Current distance: 2.661432284726255
INFO:root:1 📦 Current distance: 1.5230435410494085
INFO:root:1 ✓ Completed setpoint no. 6
INFO:root:1 📦 Move to next setpoint: no.7: X: 0.0, Y: 24.0
INFO:root:1 📦 Current distance: 3.840276217887207
INFO:root:1 📦 Current distance: 3.557519496164683
INFO:root:1 📦 Current distance: 2.6591417181120485
INFO:root:1 📦 Current distance: 1.5258267442588307
INFO:root:1 ✓ Completed setpoint no. 7
INFO:root:1 📦 Move to next setpoint: no.8: X: 0.0, Y: 28.0
INFO:root:1 📦 Current distance: 3.9540399166400806
INFO:root:1 📦 Current distance: 3.7189367350668894
INFO:root:1 📦 Current distance: 2.8381583600269606
INFO:root:1 📦 Current distance: 1.5614466001033056
INFO:root:1 ✓ Completed setpoint no. 8
INFO:root:1 📦 Move to next setpoint: no.9: X: 0.0, Y: 32.0
```

Ilustración 16: Ejemplo de recorrido del dron desde mission_handler.py

6. Cuando el dron termina la misión volverá al punto de origen y aterrizará.
En la siguiente imagen se mostrarán los mensajes que se imprimen en AirSim cuando el dron finaliza la misión y posteriormente llega al punto de origen:

```
RTL: landing at home position.
RTL: climb to 920 m (6 m above destination)
RTL: land at destination
RTL: return at 920 m (6 m above destination)
RTL HOME activated
```

Ilustración 17: Mensajes al finalizar la misión

7. En la carpeta droneSim/logs/instance1/sensors se encuentran las imágenes y los JSON con sus datos.

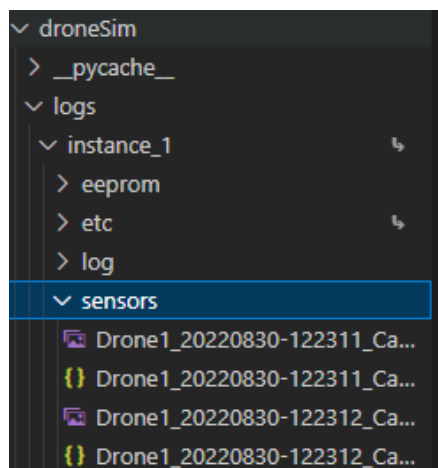


Ilustración 18: Ruta a la carpeta con las imágenes de salida

Una vez tenemos esto, podemos realizar la implementación de la parte realizada en Google Colab, que muestra el caso de uso especificado en Tabla 22: Caso de uso CU-02.

Google Colab funciona por fragmentos de código, por lo que para la ejecución de este programa se deben ejecutar uno a uno en orden. Google Colab permite la ejecución sucesiva de los mismo de forma automática si se prefiere.

Los pasos realizados son los siguientes:

1. Abrimos el proyecto de Google Colab
Para ello entramos en Google Colaboratory y seleccionamos el proyecto deseado, o bien es posible entrar desde el propio drive seleccionándolo en su carpeta correspondiente (por defecto se encuentra en la carpeta Colab Notebooks).

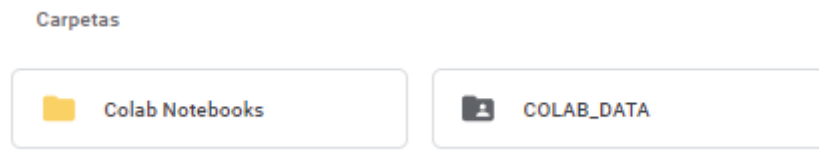


Ilustración 19: Ubicación por defecto de los proyectos de Colab en drive

2. Ejecutamos los fragmentos de código que instalan las dependencias necesarias que no están instaladas en el entorno.
Las dependencias no incluidas en el entorno de desarrollo de Colab son Pyyaml y el propio Detectron. Este último requiere alguna dependencia más a instalar, pero se instalarán automáticamente al instalar Detectron.
Si todo funciona como debería y se completa la instalación, se imprimirá el siguiente mensaje en la terminal:

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:09:46_PDT_2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.TC455_06.29190527_0
torch: 1.12 ; cuda: cu113
detectron2: 0.6
```

Ilustración 20: Mensaje de instalación satisfactoria

3. Ejecutamos los fragmentos de código para montar el drive y descomprimir el dataset.
Para ello iniciamos sesión en la cuenta de Google en la que tengamos el dataset comprimido y los descomprimos en el directorio deseado.
El dataset utilizado es el dataset de Mapillary, que consta de una gran cantidad de imágenes con distintas señales de tráfico (más de 10000), cada una de ellas tiene un JSON asociado en la carpeta de annotations, con la información de cada una de las señales existentes. En la carpeta splits se encuentran tres archivos .txt que contiene el nombre de las imágenes destinadas a cada uno de los propósitos: test.txt para las imágenes de prueba, train.txt para las imágenes de entrenamiento y val.txt para las imágenes de validación.

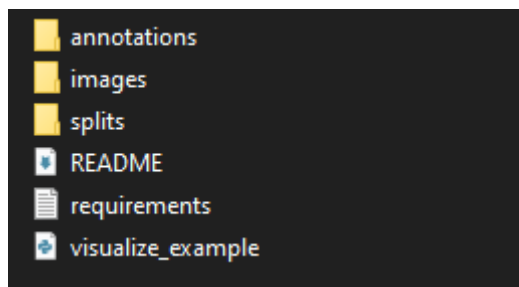


Ilustración 21: Distribución Dataset



Ilustración 22: Distribución imágenes

4. Ejecutamos los fragmentos de código para preparar el entrenamiento. Estos fragmentos se basan en preparar el dataset descomprimido para un entrenamiento, para ello primero lista el número de clases (señales) distintas que está preparado para entrenar, después cuantifica el número de muestras que hay para el entrenamiento (en este caso imágenes con anotaciones que estén en el txt de train) y por último hace un muestreo de ejemplo de las imágenes a entrenar.

```
N. Classes: 400  
E.g. regulatory--end-of-buses-only--g1
```

Ilustración 23: Ejemplo de clase

En este ejemplo se pueden observar que hay 400 clases (señales) distintas en el dataset y un ejemplo de señal.

```
num_train_examples = len(get_mapillis_dicts("train"))  
num_train_examples  
  
25956
```

Ilustración 24: Cuantificación del muestreo

Este ejemplo muestra cómo se realiza la cuantificación de imágenes para el entrenamiento.



Ilustración 25: Ejemplo imagen entrenamiento

Este es un ejemplo de una imagen que se va a usar para el entrenamiento del modelo.

5. Ejecutamos el fragmento de código que realiza el entrenamiento.
Tal y como se muestra en el apartado 4.3 Sección de Google Colaboratory se realiza el entrenamiento del modelo. Este proceso es un proceso largo que puede llegar a tomar más de 3 horas. Este entrenamiento realiza una iteración por cada imagen disponible.

```
[08/27 11:19:47 d2.data.build]: Removed 0 images with no usable annotations. 3813 images left.
[08/27 11:19:47 d2.data.build]: Distribution of instances among all 400 categories:
```

category	#instances	category	#instances	category	#instances
regulatory--..	2	regulatory--..	159	regulatory--..	11
warning--ho..	12	information..	0	information..	13
warning--ro..	7	regulatory--..	13	regulatory--..	7
warning--tr..	3	warning--do..	5	regulatory--..	25
regulatory--..	6	information..	11	information..	20
regulatory--..	37	warning--bi..	11	information..	8
regulatory--..	59	warning--cr..	22	warning--ho..	8
complementa..	30	regulatory--..	7	regulatory--..	8
information..	0	warning--bi..	7	warning--pe..	138
warning--ro..	16	regulatory--..	12	warning--tu..	6
complementa..	47	warning--ro..	12	regulatory--..	25
complementa..	9	warning--ro..	7	information..	27
warning--fa..	3	complementa..	40	regulatory--..	9

Ilustración 26: Distribución de Instancias

Esta imagen muestra las distintas instancias en las que se encuentra cada señal en el dataset. Al ser 400 categorías diferentes, no voy a incluir una imagen de cada una de ellas, pero con esta imagen es posible hacerse una idea de cuantas veces aparece cada señal a lo largo del dataset.


```
[08/27 11:20:02 d2.engine.train_loop]: Starting training from iteration 0
/usr/local/lib/python3.7/dist-packages/torch/functional.py:478: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered int
return VF.meshgrid(tensors, **kwargs) # type: ignore[attr-defined]
[08/27 11:20:11 d2.utils.events]: eta: 3:07:01 iter: 19 total_loss: 2.808 loss_cls: 2.5 loss_box_reg: 0.1519 time: 0.5070 data_time: 0.1021 lr: 4.9953e-05 max_mem: 5334M
[08/27 11:20:30 d2.utils.events]: eta: 3:08:23 iter: 39 total_loss: 3.611 loss_cls: 3.25 loss_box_reg: 0.3011 time: 0.4078 data_time: 0.2339 lr: 9.9902e-05 max_mem: 5333M
[08/27 11:20:39 d2.utils.events]: eta: 3:07:11 iter: 59 total_loss: 3.845 loss_cls: 3.444 loss_box_reg: 0.1964 time: 0.4839 data_time: 0.2595 lr: 1.4985e-05 max_mem: 5333M
[08/27 11:20:47 d2.utils.events]: eta: 2:59:14 iter: 79 total_loss: 3.669 loss_cls: 3.157 loss_box_reg: 0.3698 time: 0.4524 data_time: 0.1409 lr: 1.998e-05 max_mem: 5333M
[08/27 11:20:55 d2.utils.events]: eta: 2:59:06 iter: 99 total_loss: 3.22 loss_cls: 2.826 loss_box_reg: 0.3045 time: 0.4427 data_time: 0.1910 lr: 2.4975e-05 max_mem: 5333M
[08/27 11:21:04 d2.utils.events]: eta: 3:04:39 iter: 119 total_loss: 3.398 loss_cls: 3.077 loss_box_reg: 0.1405 time: 0.4430 data_time: 0.2170 lr: 2.997e-05 max_mem: 5333M
[08/27 11:21:13 d2.utils.events]: eta: 3:05:23 iter: 139 total_loss: 3.076 loss_cls: 2.653 loss_box_reg: 0.392 time: 0.4433 data_time: 0.2283 lr: 3.4965e-05 max_mem: 5333M
[08/27 11:21:20 d2.utils.events]: eta: 2:59:20 iter: 159 total_loss: 3.348 loss_cls: 3.109 loss_box_reg: 0.2279 time: 0.4325 data_time: 0.1269 lr: 3.996e-05 max_mem: 5333M
[08/27 11:21:28 d2.utils.events]: eta: 2:58:32 iter: 179 total_loss: 3.246 loss_cls: 2.909 loss_box_reg: 0.4175 time: 0.4318 data_time: 0.2211 lr: 4.4955e-05 max_mem: 5333M
[08/27 11:21:37 d2.utils.events]: eta: 2:59:56 iter: 199 total_loss: 3.156 loss_cls: 2.970 loss_box_reg: 0.1488 time: 0.4331 data_time: 0.2262 lr: 4.995e-05 max_mem: 5333M
[08/27 11:21:46 d2.utils.events]: eta: 2:58:55 iter: 219 total_loss: 3.367 loss_cls: 3.077 loss_box_reg: 0.2295 time: 0.4313 data_time: 0.2063 lr: 5.4945e-05 max_mem: 5333M
[08/27 11:21:56 d2.utils.events]: eta: 3:02:50 iter: 239 total_loss: 4.042 loss_cls: 3.413 loss_box_reg: 0.3222 time: 0.4385 data_time: 0.2894 lr: 5.994e-05 max_mem: 5333M
[08/27 11:22:04 d2.utils.events]: eta: 3:01:26 iter: 259 total_loss: 3.247 loss_cls: 2.757 loss_box_reg: 0.3311 time: 0.4360 data_time: 0.1814 lr: 6.4935e-05 max_mem: 5333M
[08/27 11:22:12 d2.utils.events]: eta: 2:57:46 iter: 279 total_loss: 3.523 loss_cls: 3.105 loss_box_reg: 0.3371 time: 0.4319 data_time: 0.1701 lr: 6.993e-05 max_mem: 5333M
[08/27 11:22:19 d2.utils.events]: eta: 2:56:26 iter: 299 total_loss: 2.915 loss_cls: 2.793 loss_box_reg: 0.2139 time: 0.4290 data_time: 0.1681 lr: 7.4925e-05 max_mem: 5333M
```

Ilustración 27: Inicio del entrenamiento

En esta imagen se puede ver el inicio del entrenamiento. Aunque aparezca un warning, es controlado y el entrenamiento se desarrollará con normalidad. Se puede comprobar que el tiempo estimado es de alrededor de 3 horas.

```
[08/27 14:19:05 d2.utils.events]: eta: 0:00:38 iter: 25899 total_loss: 0.8155 loss_cls: 0.582 loss_box_reg: 0.1879 time: 0.4139 data_time: 0.1121 lr: 0.00025 max_mem: 5531M
[08/27 14:19:12 d2.utils.events]: eta: 0:00:30 iter: 25879 total_loss: 0.5167 loss_cls: 0.3663 loss_box_reg: 0.1226 time: 0.4139 data_time: 0.1582 lr: 0.00025 max_mem: 5531M
[08/27 14:19:20 d2.utils.events]: eta: 0:00:22 iter: 25899 total_loss: 0.4614 loss_cls: 0.3324 loss_box_reg: 0.1333 time: 0.4139 data_time: 0.1406 lr: 0.00025 max_mem: 5531M
[08/27 14:19:28 d2.utils.events]: eta: 0:00:14 iter: 25919 total_loss: 1.306 loss_cls: 0.7935 loss_box_reg: 0.2872 time: 0.4139 data_time: 0.1922 lr: 0.00025 max_mem: 5531M
[08/27 14:19:36 d2.utils.events]: eta: 0:00:06 iter: 25939 total_loss: 0.7883 loss_cls: 0.5866 loss_box_reg: 0.1751 time: 0.4139 data_time: 0.2044 lr: 0.00025 max_mem: 5531M
[08/27 14:19:44 d2.engine.hooks]: eta: 0:00:00 iter: 25955 total_loss: 0.9498 loss_cls: 0.5728 loss_box_reg: 0.1791 time: 0.4139 data_time: 0.1786 lr: 0.00025 max_mem: 5531M
[08/27 14:19:44 d2.engine.hooks]: Overall training speed: 25954 iterations in 2:59:01 (0.4139 s / it)
```

Ilustración 28: Final del entrenamiento

Una vez ha concluido el entrenamiento, obtendremos el modelo, llamado model_final.pth, que será utilizado para las detecciones de las señales en las imágenes obtenidas durante la simulación.

6. Ejecutamos los fragmentos de código que validan el entrenamiento. Estos fragmentos comprueban mediante imágenes que se encuentran en el txt de val (y por lo tanto no han sido utilizados en el entrenamiento del modelo) si el modelo ha sido correctamente entrenado. Aunque el modelo reconocerá correctamente la mayoría de las señales, nunca se podrá garantizar un 100% de eficiencia.



Ilustración 29: Ejemplo de validación

Como se puede observar en esta imagen, ha detectado cuatro señales (habiendo cuatro señales en la imagen), sin embargo, una de ellas no ha sido capaz de

encuadrarla (la señal de peligro) y ha identificado el keep-right (la azul) como keep-left. Esto es normal que ocurra con señales poco comunes, pues el dataset está compuesto en su mayoría por señales comunes como ceda el paso, prohibidos, etc.

Además, se puede observar una lista de tensor con 4 elementos, cada uno de estos elementos es el nombre de una de las señales detectadas, siendo estas los dos ceda el paso, el keep-right y por último un peligro.

7. Ejecutamos el fragmento de código que detecta señales y las identifica. Tal y como se explica en el apartado 4.3 Sección de Google Colaboratory cada imagen se pasa por el modelo, y por cada señal detectada se establece si es una señal ya existente o no, en cuyo caso se crea un JSON con la información de la detección.



Ilustración 30: Ejemplo de detección de una señal

```
[
  {
    "camera": 2,
    "tiempo": "20220827-121932",
    "latitudDron": 40.5448613,
    "longitudDron": -4.0124445
  },
  {
    "camera": 3,
    "tiempo": "20220827-121940",
    "latitudDron": 40.5448926,
    "longitudDron": -4.0124446
  },
  {
    "camera": 6,
    "tiempo": "20220827-121940",
    "latitudDron": 40.5448926,
    "longitudDron": -4.0124446
  }
]
```

Ilustración 31: Ejemplo señal detectada múltiples veces

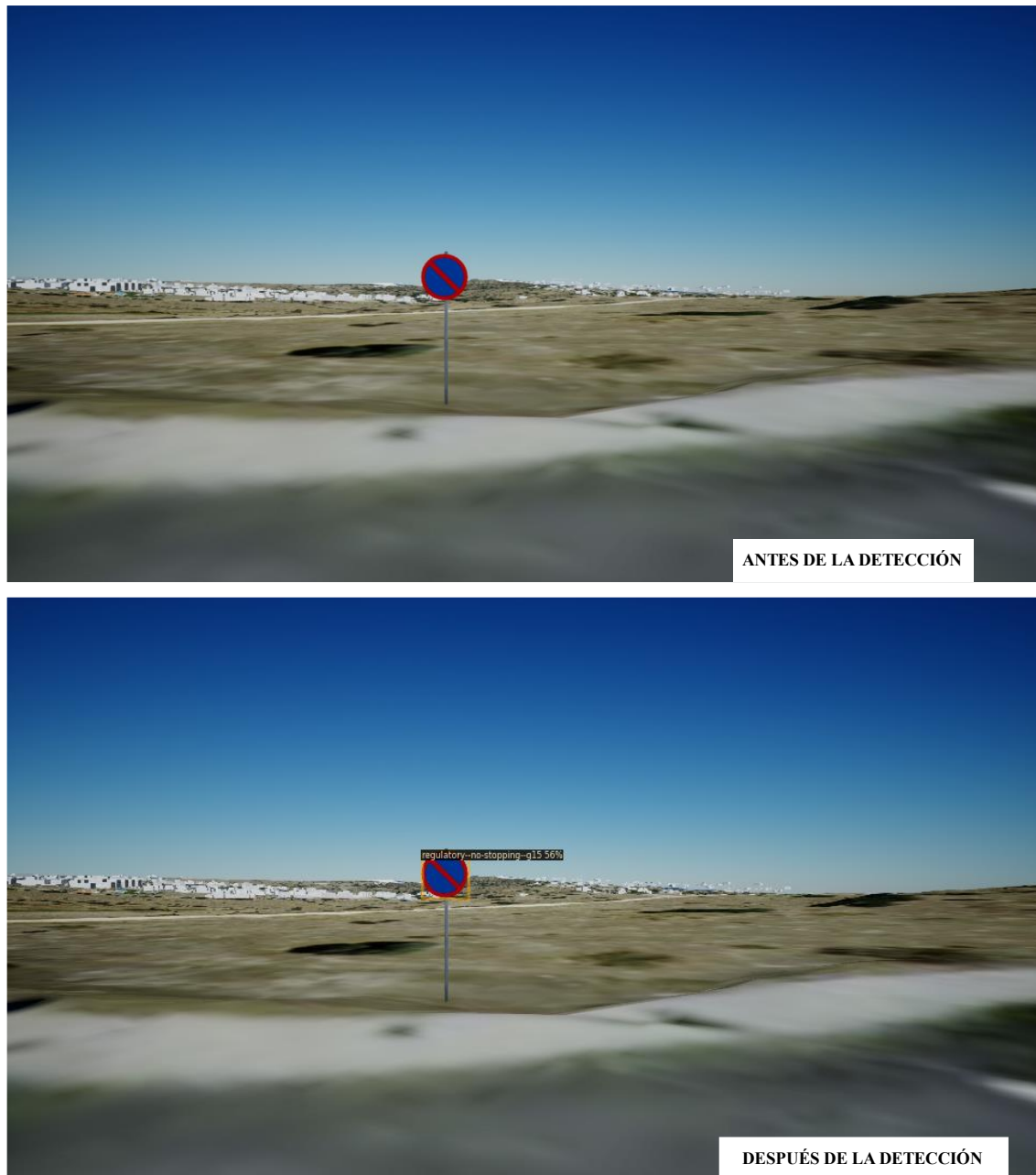


Ilustración 32: Comparación antes/después de la detección

En esta imagen podemos comprobar como es el antes y el después de las imágenes cuando son pasadas por el modelo de detección. Esta imagen será guardada en la carpeta de salida.

8. Ejecutamos el fragmento de código que copia la salida del fragmento anterior en Google drive.
Este fragmento simplemente copia la carpeta de salida al Google drive personal, de manera que no se pierda cuando se cierre el entorno de desarrollo de Google Colab.

6.2 Resultados

Los resultados obtenidos los podemos dividir en dos partes, al igual que podemos considerar que el proyecto se divide en dos partes.

En primer lugar, tenemos los resultados de la ejecución de la misión autónoma en AirSim. Es decir, tenemos un conjunto de seis imágenes capturadas por cada una de las distintas cámaras del dron por cada uno de los puntos de control o checkpoints establecidos en la misión autónoma. Además, tendremos un JSON por cada una de las imágenes con los datos de telemetría de la cámara.

Por otro lado, tenemos los resultados de la ejecución del modelo previamente entrenado en Google Colaboratoy. Estos son una imagen por cada imagen de entrada, pero con las detecciones de las señales, y además un JSON asociado a cada una de las señales.

Los resultados de la primera parte no dan mucho lugar a interpretación pues son datos intermedios cuya función es servir de datos de entrada para la segunda parte del proyecto.

Sin embargo, los resultados de la segunda parte dan para más interpretación.

Obviando las imágenes con las señales detectadas, que no dejan de ser una representación visual de las detecciones de las señales, tenemos un total de 20 archivos JSON con cada una de las señales detectadas. Teniendo en cuenta las 13 señales que existen en la simulación, podemos deducir que se han producido falsos positivos. Vamos a analizar las señales obtenidas en la detección:

- En primer lugar, tenemos dos señales de aviso de paso de cebr, que coinciden con las esperadas, pues existe un aviso de paso de cebr para cada carril.
- En segundo lugar, tenemos un aviso de parada de bus, que no existe en la simulación, al ser una señal poco común ha sido confundida con la señal de aviso de grúa, probablemente por la forma de la señal (ambas son rectangulares).
- A continuación, tenemos tres señales de prohibido el paso, estas señales se corresponden al prohibido aparcar, que confunde con bastante regularidad, esto se explicará cuando se haya hablado de todas las señales.
- Después tenemos una señal de prohibido girar a la izquierda.
- El siguiente conjunto de señales son las señales de prohibido aparcar, que ha detectado dos, existiendo dos, por lo que han sido el número esperado.
- A continuación, se encuentran cuatro señales de prohibido parar el vehículo, al igual que en el caso de prohibido el paso, se explicará al concluir el análisis de señales.
- La siguiente señal detectada es un aviso de rotonda, que al existir tres en la simulación, indica que no ha sido posible detectar dos. Sin embargo, esto es esperable, pues dos de las tres señales de aviso de rotonda están en de espaldas al recorrido del dron y estaban preparadas para no ser detectadas y ser capaz de comprobar si el modelo detecta la parte trasera de las señales como una señal en sí. Demostrando que el modelo funciona correctamente en ese aspecto.
- Después tenemos dos señales de ceda el paso, que concuerdan con los datos de la simulación, pues, aunque no se ve en la Ilustración 7: Señales implementación

existe un segundo ceda el paso, en el carril izquierdo, justo en una salida de la rotonda.

- A continuación, tenemos una señal de peligro, el modelo no ha sido capaz de detectar que señal de peligro es, pero se corresponde a la señal de aviso de paso de cebra.
- El siguiente conjunto es de una señal de aviso de elevación del asfalto, que coincide con la que existe en la simulación.
- En penúltimo lugar tenemos un aviso de obras, que es una señal que no existe en la simulación, pero probablemente se haya confundido con la señal de peligro de elevación del asfalto.
- Por último, tenemos una señal de peligro rotonda, que no coincide con ninguna en la simulación, posiblemente confundida con otra.

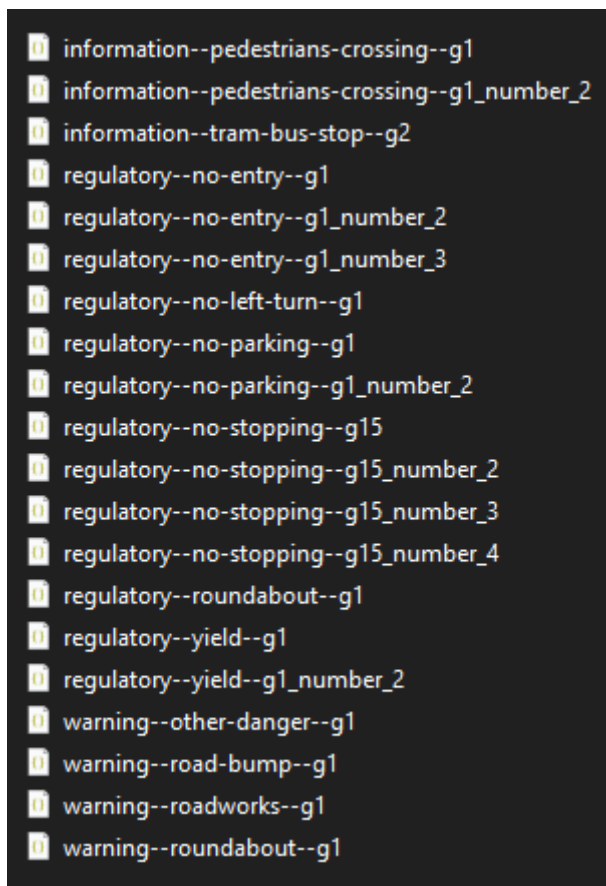


Ilustración 33: JSONs de las señales detectadas

En esta imagen se puede comprobar la estructura de los JSONs de las señales detectadas.

Una vez hemos analizado todas las detecciones de señales únicas podemos sacar las siguientes conclusiones:

- Se han detectado todas las señales menos la de parking de minusválidos y se ha confundido el aviso de grúa con el aviso de parada de bus.
- Hay algunas señales en la simulación que se han identificado como varias señales distintas en la detección. La razón principal de este problema es que las señales de la simulación no dejan de ser señales simuladas, no son 100% perfectas, ya sea

en forma, color o ajuste (entendiendo ajuste como encuadrar el contenido de la señal en la señal en sí). Esto provoca que el modelo detecte patrones inusuales a la señal en cuestión y sea capaz de detectar múltiples señales en una misma. Posiblemente en un entorno real esto ocurra muy pocas veces.

- Ha detectado 7 de las 8 señales diferentes que había en la simulación, lo que le otorga un 87.5% de precisión, entendiendo precisión como la proporción de señales correctamente identificadas
- Ha detectado 20 señales en total de 13 señales totales que existían en la simulación, lo que le da un 65% de efectividad, entendiendo efectividad como la proporción de señales detectadas respecto al total de señales existentes.

7 Conclusiones

En este apartado se va a valorar el resultado obtenido en el proyecto, si se han cumplido los objetivos establecidos en 1.3 Objetivos, cuáles son los posibles trabajos futuros y una pequeña lista con los diferentes problemas que han surgido durante el desarrollo de este proyecto.

El principal objetivo de este proyecto era conseguir un modelo de detección de señales de tráfico y poder identificar cada una de ellas y clasificarlas. Este objetivo se ha cumplido en gran medida, pues, aunque el modelo no es perfecto, es capaz de detectar señales reales y señales simuladas y además las clasifica correctamente. Si bien es cierto que el método de clasificación no usa algoritmos, si no suposiciones lógicas, funciona correctamente y permite identificar las señales.

El siguiente objetivo establecido fue la creación de una misión autónoma. Este objetivo se ha cumplido, pues se ha obtenido una misión autónoma capaz de recorrer una distancia y capturar imágenes durante su recorrido mediante una serie de cámaras distribuidas de forma que se captura toda la vista desde el dron. Además, la misión está preparada para actuar con más de un dron si fuera necesario.

El siguiente objetivo establecido fue utilizar un mapa que simule la realidad. Este objetivo se ha cumplido también, pues gracias a Cesium es posible obtener un mapa simulado de una zona real que existe en el mundo.

El siguiente objetivo establecido fue obtener información de manera constante sobre el estado de la misión autónoma. Este objetivo está relacionado con el segundo objetivo y se ha cumplido de manera satisfactoria, porque desde la misión se pueden ver las imágenes que se están capturando, además de existir el `mission_handler.log` que permite obtener información del recorrido que está realizando y el momento en el que captura las imágenes.

El último objetivo establecido fue obtener la salida de las detecciones de imágenes. Este objetivo también se ha cumplido, pues durante la detección, el programa devuelve las imágenes con las detecciones, de manera que se puede ver de manera visual que señales se han detectado.

Para concluir, se podría decir que se han cumplido todos los objetivos propuestos para el proyecto, pero siendo realistas, hay que notar que el título del proyecto es Detección y Geolocalización de objetos automática desde drones y la geolocalización realizada es muy primitiva. Esto se debe principalmente a la falta de tiempo, pues un proyecto de este calibre es difícil de completar en un periodo de realización de TFG, y el resultado ha obligado a ser menos preciso en ese campo. Sin embargo, el resultado de este TFG es satisfactorio y abre líneas de trabajo muy interesantes para el futuro.

7.1 Líneas futuras

Una vez concluido el desarrollo del proyecto, se pueden proponer una serie de distintos trabajos futuros que mejorarían la implementación de este proyecto y permitirían tener un mejor sistema de detección.

7.1.1 Realizar pruebas en entornos reales

El primer trabajo futuro es obvio, poder pasar de un entorno simulado a probar esto en calles reales. Esto supondría conseguir todas las licencias necesarias y tomar precauciones adicionales con el dron. Posiblemente no se pueda volar el dron a la altura que se establece en la misión autónoma, pues probablemente ralentizaría el tráfico de manera notable.

7.1.2 Mejorar el sistema de detección

Actualmente el modelo está entrenado para detectar señales de tráfico. Se podría mejorar el sistema extendiéndolo al resto de elementos de una carretera, como pueden ser semáforos, luces u otros vehículos. Existen datasets en Mapillary que permiten plantear esta línea de trabajo futuro.

7.1.3 Localización con precisión

La manera en la que este proyecto localiza la posición de las señales se basa en aproximar la posición desde la ubicación del dron cuando captura una imagen en la que se detecta dicha señal. Mediante algoritmos de posicionamiento se podría aproximar con mucha más precisión la posición de la señal. El proyecto está diseñado para iniciar esta línea de trabajo, pues la telemetría que se obtiene de las imágenes incluye datos necesarios para estos algoritmos como por ejemplo la rotación de la cámara, o el grado de estabilidad en el momento de la imagen.

7.2 Problemas

El principal problema a la hora de la realización de este proyecto fue el encontrar un programa o librería que permitiera entrenar un modelo en cualquier tipo de ordenador. La mayoría de los sistemas de entrenamiento de modelos tienen altas restricciones a la hora de su uso, incluso Detectron (herramienta utilizada finalmente) tiene una gran restricción, que es la necesidad de CUDA para su uso. Esto implicaba que cualquier sistema con una GPU que no fuera de la marca de NVIDIA no pudiera ejecutar el entrenamiento de un modelo (Mi GPU era de AMD por lo que me ha tocado improvisar). Esto se solucionó gracias a Google Colab, que de manera gratuita permite ejecutar código, aunque finalmente se tuvo que recurrir a Colab Pro debido al tamaño del dataset de entrenamiento.

Otro problema, aunque mucho menor, han sido las falsas detecciones de señales en el entorno simulado. Esto ocurre porque el modelo está entrenado para detectar señales reales, y las simuladas tienen patrones que pueden no coincidir con las reales. Por ejemplo, hay señales que no están perfectamente encuadradas, o señales que por los píxeles pierden un poco de calidad. Realmente no hay una solución que sea 100% efectiva contra este problema, pero se ha intentado minimizar el impacto que este problema puede ocasionar en un caso de uso.

8 Bibliografía / Referencias

- [1] «El Ministerio de Industria anticipa más de 1.400 millones de euros en drones de guerra», *20bits*, 4 de mayo de 2022. <https://www.20minutos.es/tecnologia/actualidad/el-ministerio-de-interior-anticipa-mas-de-1-400-millones-de-euros-en-drones-de-guerra-4994776/> (accedido 5 de septiembre de 2022).
- [2] Infodron, «El sector dron triplica su inversión en 2021 con 7.000 millones de dólares», *Infodron*. <https://www.infodron.es/texto-diario/mostrar/3529015/sector-dron-triplica-inversion-2021-7000-millones-dolares> (accedido 5 de septiembre de 2022).
- [3] «¿Qué es Dron? » Su Definición y Significado [2022]», *Concepto de - Definición de*. <https://conceptodefinicion.de/dron/> (accedido 22 de febrero de 2022).
- [4] «Open Source Autopilot for Drones», *PX4 Autopilot*. <https://px4.io/> (accedido 5 de septiembre de 2022).
- [5] «Introduction · MAVSDK Guide». <https://mavsdk.mavlink.io/main/en/index.html> (accedido 5 de septiembre de 2022).
- [6] ArduPilot, «ArduPilot», *ArduPilot.org*. <https://ardupilot.org> (accedido 5 de septiembre de 2022).
- [7] «Home - AirSim». <https://microsoft.github.io/AirSim/> (accedido 24 de febrero de 2022).
- [8] C. Team, «CARLA», *CARLA Simulator*. <http://carla.org/> (accedido 5 de septiembre de 2022).
- [9] «Motor de videojuego», *Wikipedia, la enciclopedia libre*. 22 de enero de 2022. Accedido: 24 de febrero de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Motor_de_videojuego&oldid=141146960
- [10] A. Soloaga, «Unreal Engine, qué es y para qué sirve», *El Blog de Akademus*, 19 de julio de 2019. <https://www.akademus.es/blog/emprendedores/unreal-engine-que-es-y-para-que-sirve/> (accedido 24 de febrero de 2022).
- [11] U. Technologies, «Plataforma de desarrollo en tiempo real de Unity | Motor de VR y AR en 3D y 2D». <https://unity.com/es> (accedido 5 de septiembre de 2022).
- [12] «Source - Valve Developer Community». <https://developer.valvesoftware.com/wiki/Source> (accedido 5 de septiembre de 2022).
- [13] «Home», *OpenCV*. <https://opencv.org/> (accedido 5 de septiembre de 2022).
- [14] «SimpleCV». <http://simplecv.org/> (accedido 5 de septiembre de 2022).
- [15] «Google Cloud Vision API - Davinci Group». <https://www.ackstorm.com/google-cloud-vision-api/> (accedido 5 de septiembre de 2022).
- [16] «Detectron». Meta Research, 5 de septiembre de 2022. Accedido: 5 de septiembre de 2022. [En línea]. Disponible en: <https://github.com/facebookresearch/Detectron>
- [17] «Microsoft Windows», *Wikipedia, la enciclopedia libre*. 23 de febrero de 2022. Accedido: 24 de febrero de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Microsoft_Windows&oldid=141875308
- [18] «¿Qué es Linux? » Su Definición y Significado 2021», *Concepto de - Definición de*. <https://conceptodefinicion.de/linux/> (accedido 24 de febrero de 2022).
- [19] craigloewen-msft, «Qué es el Subsistema de Windows para Linux». <https://docs.microsoft.com/es-es/windows/wsl/about> (accedido 24 de febrero de 2022).

- [20] «Documentation for Visual Studio Code». <https://code.visualstudio.com/docs> (accedido 24 de febrero de 2022).
- [21] «History and License — Python 3.10.2 documentation». <https://docs.python.org/3/license.html> (accedido 22 de febrero de 2022).
- [22] «Google Colaboratory», <https://colab.research.google.com/>, 1 de agosto de 2022. <https://colab.research.google.com/> (accedido 1 de septiembre de 2022).
- [23] Mapillary, «The street-level imagery platform that scales and automates mapping», *Mapillary*. <https://mapillary.com> (accedido 1 de septiembre de 2022).
- [24] «Mapillary». <https://www.mapillary.com/dataset/trafficsign> (accedido 2 de septiembre de 2022).
- [25] «Descubre Street View y añade tus propias imágenes a Google Maps.», *Google Maps Street View*. <https://www.google.com/intl/es/streetview/> (accedido 2 de septiembre de 2022).
- [26] *Reglamento de Ejecución (UE) 2019/947 de la Comisión, de 24 de mayo de 2019, relativo a las normas y los procedimientos aplicables a la utilización de aeronaves no tripuladas (Texto pertinente a efectos del EEE.)*, vol. 152. 2019. Accedido: 4 de septiembre de 2022. [En línea]. Disponible en: http://data.europa.eu/eli/reg_impl/2019/947/oj/spa
- [27] «Drones | AESA-Agencia Estatal de Seguridad Aérea - Ministerio de Fomento». <https://www.seguridadaerea.gob.es/es/ambitos/drones> (accedido 1 de septiembre de 2022).
- [28] «BOE.es - BOE-A-2018-16673 Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.» <https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673> (accedido 1 de septiembre de 2022).
- [29] «Google Colaboratory». https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5?hl=es (accedido 4 de septiembre de 2022).
- [30] «Modelo de prototipos: ¿qué es y cuáles son sus etapas? | Blog | Hosting Plus España», *Hosting Plus*, 6 de julio de 2021. <https://www.hostingplus.com.es/blog/modelo-de-prototipos-que-es-y-cuales-son-sus-etapas/> (accedido 29 de agosto de 2022).
- [31] «Modelo en espiral: el modelo para la gestión de riesgos en el desarrollo de software», *IONOS Startupguide*. <https://www.ionos.es/startupguide/productividad/modelo-en-espiral/> (accedido 29 de agosto de 2022).
- [32] «El modelo en cascada: desarrollo secuencial de software», *IONOS Digital Guide*. <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-modelo-en-cascada/> (accedido 29 de agosto de 2022).
- [33] «¿Qué es y para qué sirve un diagrama de Gantt?», *España*. <https://www.teamleader.es/blog/diagrama-de-gantt> (accedido 6 de septiembre de 2022).
- [34] «Especificación de Requisitos Software según el estándar IEEE 830 - FdIwiki ELP». https://wikis.fdi.ucm.es/ELP/Especificaci%C3%B3n_de_Requisitos_Software_seg%C3%BA_n_el_est%C3%A1ndar_IEEE_830 (accedido 5 de septiembre de 2022).
- [35] «Jobted - Descubra 200.000+ Trabajos y Ofertas de Empleo en España». <https://www.jobted.es> (accedido 29 de agosto de 2022).

- [36] «BOE.es - BOE-A-1992-28740 Ley 37/1992, de 28 de diciembre, del Impuesto sobre el Valor Añadido.» <https://www.boe.es/buscar/act.php?id=BOE-A-1992-28740> (accedido 29 de agosto de 2022).
- [37] «Mapillary». <https://www.mapillary.com/app/?lat=40.54273127728956&lng=-4.010857863663887&z=17.873472299974356&trafficSign%5B%5D=all> (accedido 5 de septiembre de 2022).

Anexo A. Extended abstract

In this section, I will give a brief summary of the project. This summary will contain a small portion of the content of each section. I will try to include the most relevant parts of each section so that you can understand the purpose of this project just by reading this annex.

A.1 Introduction

In recent years, the number of tasks automated by non-human-controlled devices, such as traffic regulation or facial recognition, has increased. One of the main uses of artificial intelligence (AI) is image recognition and processing. In addition, the use of drones has increased significantly since their price and ease of use has increased considerably. The idea of this project arises from the possibility of taking images captured by a drone in a simulated environment and analysing them by using a model afterwards.

The main motivation for this project is to be able to make an autonomous mission that performs a pre-established route and takes images during this route with a drone without the help of human hands. This makes it possible to reach areas that are difficult to access without putting human lives at risk.

The main objective of this project is to be able to generate a traffic signal detection model that can detect traffic signals in a simulated environment using AirSim and then identify each of them.

Other objectives include the following:

- Being able to generate an autonomous mission.
- Use a simulated map with road signs as accurately as possible.
- Be able to have information related to the mission at every moment.
- Obtain resulting images of the final signal detection, to have a more visual perspective of the results.

A.2 State of the art

This section will explain the previous concepts necessary to understand the development of the project.

A.2.1 Drones

A drone[3] is an unmaned aerial vehicle (UAV), an airship who flies without tripulation and does its tasks remotely, normally is reusable and is capable of mantain flight level constantly by an autonomous way.

Drones can be classified by many ways depending of diverse parameters. This are the most common sorting of drones:

By type of use:

- Military use drones: These are the most complex and evolved drones, can be used to supply tasks, aerial strikes, tracking, scanning or as a practice target.
- Civilian use drones: These are less complex and more cheap and accesible to everyone. Their common uses are fotografic sesions, video making or leisure related.

By weight:

- Nanodrones: Weight less tan 250g.
- Aerial Microvehicles: Weight between 250g and 2kg.
- Miniature unmaned vehicles: From 2kg to 25kg.
- Medium drones: From 25kg to 150kg.
- Big drones: Almost exclusevy military used, these are drones who weight more than 150kg.

By altitude:

- Handheld: Can fly until 600 meters of altitude, with an average 2km of flight scope.
- Close: Until 1500 meters of altitude, and an average of 10km of flight scope.
- NATO: 3000 meters of altitude and 50km of flight scope.
- Tactical: 3500 meters of altitude and 160km of flight scope.
- Medium altitude, Long endurance (MALE): 9000 meters of altitude, 200km of flight scope.
- High altitude, Long endurance (HALE): Similar to MALE, but with more scope.
- Hypersonic: Can get over supersonic speed, with 15000 meters of altitude and more tan 200km of flight scope.
- Orbital: Can make low orbital routes.
- CIS lunar: Can travel from Earth to the Moon.

By number of arms:

- Cuadricopter: 4 arms and 4 engines, one for each arm. Its use is more extended than the rest.
- Hexacopter: 6 arms and 6 engines.
- Octocopter: 8 arms and 8 engines.
- Coaxial: Doesn't have a certain number of arms, but has two engines per arm.

The drone used for this proyect will be a civilian used, MAV, Handheld cuadricopter.

A.2.2 Flight Controller

A flight controller will be needed to allow the conection between dron and user. These are the flight controllers used for this project:

- PX4: Open source code autonomus flight controller that allows to use several drones at once, with a variety of drone types compatibility[4].
- MavSDK: Set of several libraries of several programming languages for drones or terrestrial vehicles. Has an easy-to-use API that allows several simultaneous drone use[5].

A simulator that can aproach reality will be needed. AirSim is the selected software. AirSim[7] is a drone simulator made for Unreal Engine (also has a beta version for Unity). Is an open source, multiplatform simulator that has support for flight controllers as PX4 or MavSDK.

A.2.3 Game Engine

AirSim needs a game engine that can simulate physics and generate enviroments to make it been able to perform correctly.

A game engine[9] is a set of programming routines that allows design, creation and working of (in almost all cases) a game, but also simulated enviroments or renders. It includes functionalities to render in 2D or 3D, simulate physics (like gravity), sounds, animations and much more.

The Game Engine selected for this proyect is Unreal Engine, developed by Epic, writen in C++, has the advantage of being free if your project hasnt enough renew.

A.2.4 AI

Artificial Intelligence will be needed to make this project posible.

The selected AI elements will be:

- OpenCV: Open source computer vision originally developed by Intel. Has a multiplatform compatibility and has a very complete documentation[13].
- Detectron: Open source platform focused on object detection on images. Uses pytorch, CUDA and OpenCV to work[16].
-

A.2.5 Development Enviroment

This will be the development enviroment:

- Operative system: This projec will use two different OS, Windows and Linux under Windows Subsystem for Linux. This will allow it to comunicate with PX4 and MavSDK that need to run in a Linux enviroment and AirSim (with Unreal) that can run in Windows.
- Programming lenguaje: In this project Python[21] will be used as programming lenguaje. Visual Studio Code will be used too, as it allows to use WSL while programming.
- Google Colaboratory: It allows the execution of code without no previous set up. It will allow to use Detectron efectively. Colab has several monetization models, being Colab Pro are the selected for this project[22].

A.2.6 Similar Applications

This isn't the first project to use signal detection by using models. There are some applications that are similar to this project.

- Mapillary: Platform that allows access to images from around the world and allows to use that info for several purposes. It also grants access to several dataset with a lot of information like traffic signs, cars and much more[23].
- Google Street view: Its a visual representation of Google Maps with a lot of panoramic images. It allows to visualize almost every corner of Earth[25].

A.3 Socioeconomic and legal environment

This section will discuss the possible social and economic impact of the implementation of this project. In addition, the regulatory framework under which this project is based will also be discussed.

A.3.1 Socioeconomic Impact

This project can have different applications if it is brought to reality. Firstly, it can improve road safety by allowing large road sweeps or reaching roads that are difficult to access, making it possible to establish the status of traffic signs depending on whether they are detected or not.

From a purely economic point of view, drones designed for this purpose are not excessively expensive and can be very cost-effective. It would probably not destroy more jobs than it could create.

A.3.2 Legal environment

Spanish drone regulations are based on European regulations, currently SR 2019/947 and RD 2019/945 [26]. Its measures include:

- Unmanned aircraft, regardless of their mass, may be used in the same airspace as manned aircraft.
- Unmanned aircraft operators must register if they use aircraft equipped with a sensor that can capture personal data. If the aircraft is considered a toy, this step is not necessary.
- UAS operators must be informed of EU and national rules applicable to planned operations, especially on safety, security, privacy, data protection, liability, insurance and environmental protection.
- Certain areas may be sensitive to some types of UAS operations, so member states may establish additional rules in those areas.

In Spain, the body in charge of ensuring compliance with civil aviation regulations is the State Agency for Air Safety (AESA)[27].

In order to fly a drone, there are regulations that must be complied with regardless of its use, the basic points of which are:

- Any user who intends to fly a drone must register at AESA's electronic headquarters and obtain an operator number that must be visibly displayed on the drone.
- To fly a drone, a minimum amount of accredited training is required. The corresponding training and examination can be obtained free of charge on the AESA website (telematically).
- You must have civil liability insurance.
- Drone flight is subject to general rules of conditional operation, such as the weight of the drone, the presence of other people or proximity to buildings.
- There are additional restrictions on flying in certain places, such as near airfields, military bases or hospitals.

This project also complies with the general data protection regulation, which is a regulation at European level concerning the protection of individuals with regard to their personal data. This project specifically complies with EU Regulation 2016/779, known as Organic Law 3/2018 of 5 December [28].

All programs and software used for the development of this project are licensed as open source for personal development, as long as they are not commercialised. As this is a university research project, it complies with the rules of use.

A.4 Solution design

In this section, the proposed design of the project will be discussed by means of a flowchart showing the different components of the design. In addition, all the classes that make up this design will be explained, accompanied by images of the code to facilitate the understanding of the design.

A.4.1 System Architecture

To facilitate the understanding of this section, three diagrams have been used with the different sections of the project:

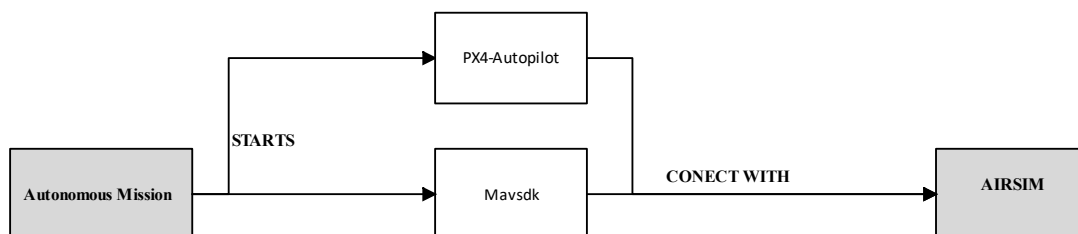


Figura 12 - A1: Components Ubuntu

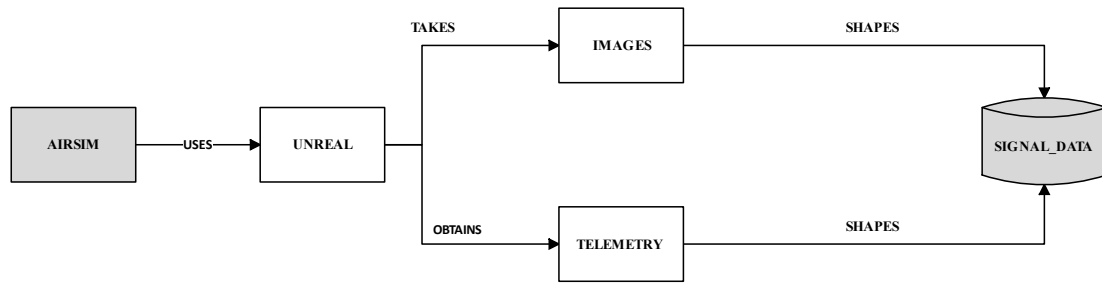


Figura 13 - A2: Components Windows

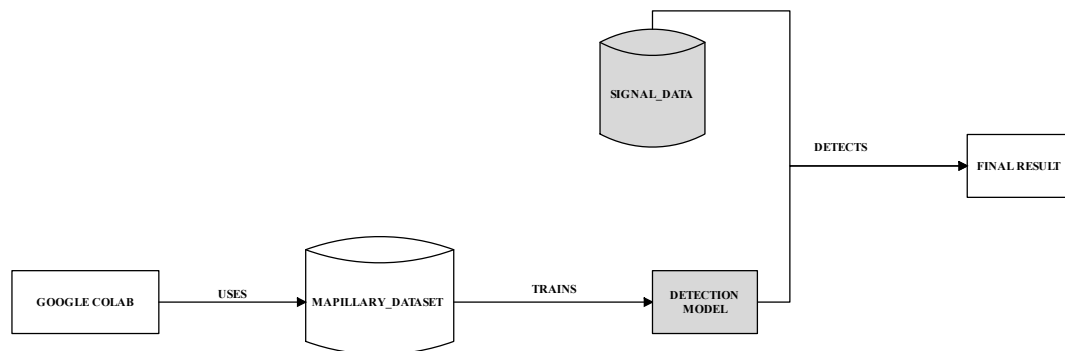


Figura 14 - A3: Components Colaboratory

A.4.2 TrafficSignDetection

In this subsection, we will discuss the classes that make up the autonomous mission, as well as the JSON file that allows us to obtain the specific characteristics of the simulation.

A.4.2.1 exampleSettings.json

This file contains the basic attributes of the drone to be used in the simulation, among them: SimMode which indicates the type of vehicle to be used (multirotor in this case), Vehicles which contains a list of vehicles (only one) and each of the drones defined in Vehicles.

A.4.2.2 mainNoGUI.py

This class is the main class of the autonomous mission. It defines the type of mission, defines the drones that are to execute the mission and prepares the execution of the mission. Finally it launches missionHandler.py which will be in charge of controlling the performance of the mission.

A.4.2.3 missionHandler.py

This class connects the mission to the AirSim environment and launches its execution. It also creates a log where all the information about the drone during the mission is stored. It launches the autonomous mission in question, which in this case is trafficSign.

A.4.2.4 trafficSign.py

This class has the code corresponding to the mission. During the mission the drone stops at various checkpoints set in maptools.py. At each checkpoint it takes an image for each camera and once it reaches the last checkpoint it returns to its point of origin.

A.4.2.5 mapTools.py

Allows you to plot the checkpoints that the drone will traverse during the mission.

A.4.3 Google Colaboratory Section

In order to carry out this project, Google Colaboratory has been used, specifically the part related to the detection software. This subsection will explain the procedure carried out to obtain a traffic sign detection model.

As it is a hosted execution environment, each time it is started it will be necessary to install the elements that are going to be used.

Once installed, the Mapillary traffic sign dataset will be used to train the model.

Then the dataset is prepared and the training is performed and finally the trained model is used to detect the signals from the images obtained from the AirSim simulation.

A.5 Planning

This section deals with the more professional part of the project.

A.5.1 Planning

The planning model used for the planning of this project is the "spiral model"[31], which allows the different phases of a project to be carried out in each of the tasks.

The implementation of the planning model is based on following four-step cycles, which are as follows:

- **Planning:** Gather user requirements and check the scope of the phase.
- **Analysis:** Analysing the requirements and identifying risks that may arise.
- **Implementation:** Development of the programme and required testing.
- **Evaluation:** Check whether the risks have been addressed and the objectives have been met to move on from the cycle.

The project consists of four phases:

- **Environment preparation:** The development environment is prepared, configuring both the Linux machine under WSL and the Unreal environment with AirSim.
- **Training and validation of the detection model:** In this phase, the best method of obtaining a model for signal detection and its operation is investigated.
- **Design and implementation of the autonomous mission:** The autonomous mission that the drone will execute to obtain the images is designed, developed and implemented.

- **Signal classification design:** The signal detection and location approach will be designed.

The project duration starts on 31 January 2022 and ends on 7 September 2022. It spans 32 weeks in which an average of 18 hours per week has been worked, for a total of 576 hours.

The hours breakdown of the Gantt chart[33] assuming an average of 18 hours per week:

- **Environment Preparation:**
 - **Planning:** 18 hours.
 - **Analysis:** 10 hours.
 - **Implementation:** 26 hours.
 - **Evaluation:** 18 hours.
- **Training and validation of the detection model:**
 - **Planning:** 36 hours.
 - **Analysis:** 36 hours.
 - **Implementation:** 106 hours.
 - **Evaluation:** 54 hours.
- **Design and implementation of the autonomous mission:**
 - **Planning:** 36 hours.
 - **Analysis:** 36 hours.
 - **Implementation:** 106 hours.
 - **Evaluation:** 18 hours.
- **Signal classification design:**
 - **Planning:** 9 hours.
 - **Analysis:** 9 hours.
 - **Implementation:** 36 hours.
 - **Evaluation:** 22 hours.
- **Memoria:** 50 horas.

A.5.2 Requirements / User cases

The project requirements follow the conventions specified and established by the IEEE standard[34].

Three types of requirements will be used (as this is a summary of the project, an example of each type of requirement will be included):

- **User requirement:** Those that capture the needs expressed by the user to achieve an objective.

Tabla 29 A-1: User Requirement UR-03

Requirement number	UR-03
Requirement name	Programming technology
Requirement priority	High
Requirement status	Invariable
Requirement description	The mission and detection model code will be programmed using Python.

- **Functional software requirements:** These define the services that the system must have.

Tabla 30 A-2: Functional Software Requirement FR-04

Requirement number	FR-04
Requirement name	Image information
Requirement priority	High
Requirement status	Invariable
Requirement description	The system must be able to store in a JSON file the information related to an image captured from the drone.

- **Non-functional software requirements:** These define the restrictions of the system.

Tabla 31 A-3: Non-Functional Software Requirement NFR-02

Requirement number	NFR-02
Requirement name	Ubuntu Compatibility
Requirement priority	High
Requirement status	Invariable
Requirement description	The project must be able to run on an Ubuntu system running version 20.04+.

The proposed use cases are the project executions. Here is an example of a use case:

Tabla 32 A-4: User case UC-01

Identifier	UC-01
Use case name	Execution of the autonomous mission.
Personal	User.
Objective	Execute the autonomous mission and obtain the necessary output data.
Use case environment	The user must start the Unreal project with the map and enter the data of the drone to be used for simulation.
Output	Once the drone finishes its mission, a folder with all the images captured by the drone and the JSON files with their information will be obtained.
Guide	<ul style="list-style-type: none">- User initiates Unreal project- User determines the characteristics of the drone- User starts the programme- The user waits for the mission to finish- The user can access the images and JSON obtained during the mission.

A.5.3 Budget

This is the table with the characteristics of the project and its final cost:

Tabla 33: Offer breakdown

Title	Detección y geolocalización de objetos automática desde drones.
Author	Pablo de Alba Martínez
Start Date	31 January 2022
End Date	7 September 2022
Elapsed time	32 weeks
Final budget	15491,42 €

A.6 Implementation / Results

This section will explain the different steps to execute the project correctly and obtain the expected results.

A.6.1 Implementation

System requirements:

- Operating System: Windows 10 or 11 with access to the Microsoft Store and the Ubuntu LTS 20.04 application.
- Applications: Visual Studio Code, Windows Subsystem for Linux, Unreal 4.27+, AirSim, MavSDK, PX4 Autopilot and Python/Anaconda.
- Required libraries: Channels, numpy, opencv and pymap3d.
- Internet: A stable internet connection will be required to run the Google Colab part.

- Cesium map: It will be necessary to have a Cesium map compatible with the AirSim version.

The implementation has been separated into two parts, firstly the part carried out in the AirSim simulation and secondly the part carried out in Google Colab.

For the first case, which coincides with Tabla 21: Caso de uso CU-01, the following steps have to be followed:

1. Start Unreal with the project in which the map is located.
2. Start the program with the mainNoGUI.py class.
3. Wait for the connection of MavSDK and PX4 with AirSiM.

This is the message that appears when a successful connection is established:



Ilustración 34 A-1: Successful connection message

4. Watch from the Unreal screen how the drone performs the mission.

In the following images you can see how the autonomous mission is carried out:



Ilustración 35 A-2: Flight from the drone



Ilustración 36A-3: Image captured from the drone

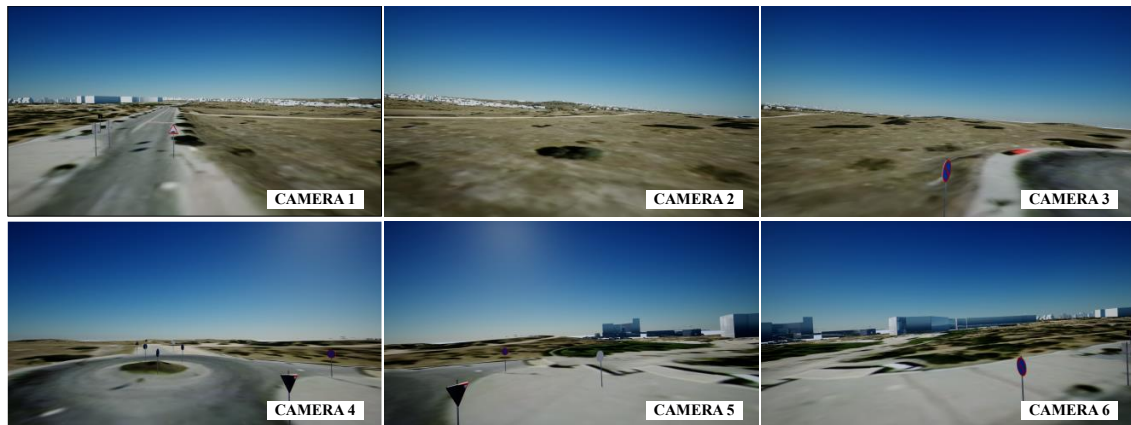


Ilustración 37 A-4: Collage of images captured by the drone at the checkpoint

5. Optionally you can consult the `mission_handler.log` which details the steps through the checkpoints.
6. When the drone completes the mission, it will return to the starting point.
7. The output of the images captured by the drone and the JSONS with their data are located in the `droneSim/logs/instance1/sensors` folder.

For the second case, which coincides with case 2, involving the use of Google Colab, the following steps must be followed:

1. Open the Google Colab project.
2. Execute the code snippets that install the dependencies not installed in the environment. The dependencies not included are Pyyaml and Detectron, this one installs some more dependencies during installation.

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:09:46_PDT_2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.TC455_06.29190527_0
torch: 1.12 ; cuda: cu113
detectron2: 0.6
```

Ilustración 38A-5: Successful installation message

3. Execute the code snippets to mount the drive and decompress the dataset. The dataset chosen was the Mapillary dataset, which contains more than 10000 images of traffic signs, each one of them has an associated JSON in the annotations folder, with the information of each one of the signs in that image. In the splits folder there are three .txt files containing the name of the images for each purpose. test.txt will contain the test images, train.txt for training and val.txt for validation.

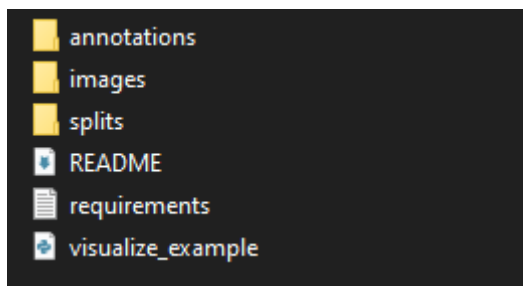


Ilustración 39 A-6: Dataset distribution

4. Execute the code snippets to prepare the training. These fragments prepare the dataset for training by listing the number of signs, quantifying the samples and making a sampling example.



Ilustración 40 A-7: Training Sample Image

5. Execute the code fragment that performs the training. Explained in section 4.3 Sección de Google Colaboratory , it is a lengthy process that can take a long period of time (up to three hours).

```
[08/27 11:28:01 d2.engine.train_loop]: Starting training from iteration 0
/usr/local/lib/python3.7/dist-packages/torch/functional.py:478: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered int
return _VF.meshgrid(tensors, **kwargs) # type: ignore[attr-defined]
[08/27 11:28:21 d2.utils.events]: eta: 3:07:01 iter: 19 total_loss: 2.808 loss_cls: 2.5 loss_box_reg: 0.1519 time: 0.5070 data_time: 0.1821 lr: 4.9953e-06 max_mem: 5324M
[08/27 11:28:30 d2.utils.events]: eta: 3:03:23 iter: 39 total_loss: 3.611 loss_cls: 3.25 loss_box_reg: 0.3011 time: 0.4878 data_time: 0.2339 lr: 9.9902e-06 max_mem: 5333M
[08/27 11:28:39 d2.utils.events]: eta: 3:07:11 iter: 59 total_loss: 3.845 loss_cls: 3.444 loss_box_reg: 0.3964 time: 0.4839 data_time: 0.2595 lr: 1.4085e-05 max_mem: 5333M
[08/27 11:28:47 d2.utils.events]: eta: 2:59:14 iter: 79 total_loss: 3.669 loss_cls: 3.157 loss_box_reg: 0.3698 time: 0.4524 data_time: 0.1469 lr: 1.998e-05 max_mem: 5333M
[08/27 11:28:55 d2.utils.events]: eta: 2:59:06 iter: 99 total_loss: 3.22 loss_cls: 2.826 loss_box_reg: 0.3045 time: 0.4427 data_time: 0.1910 lr: 2.4975e-05 max_mem: 5333M
[08/27 11:21:04 d2.utils.events]: eta: 3:04:39 iter: 119 total_loss: 3.398 loss_cls: 3.077 loss_box_reg: 0.1405 time: 0.4430 data_time: 0.2170 lr: 2.997e-05 max_mem: 5333M
[08/27 11:21:13 d2.utils.events]: eta: 3:05:23 iter: 139 total_loss: 3.076 loss_cls: 2.653 loss_box_reg: 0.392 time: 0.4433 data_time: 0.2283 lr: 3.4965e-05 max_mem: 5333M
[08/27 11:21:20 d2.utils.events]: eta: 2:59:20 iter: 159 total_loss: 3.348 loss_cls: 3.103 loss_box_reg: 0.2279 time: 0.4325 data_time: 0.1269 lr: 3.996e-05 max_mem: 5333M
[08/27 11:21:28 d2.utils.events]: eta: 2:58:32 iter: 179 total_loss: 3.246 loss_cls: 2.809 loss_box_reg: 0.4175 time: 0.4318 data_time: 0.2211 lr: 4.4955e-05 max_mem: 5333M
[08/27 11:21:37 d2.utils.events]: eta: 2:59:56 iter: 199 total_loss: 3.156 loss_cls: 2.978 loss_box_reg: 0.1488 time: 0.4331 data_time: 0.2262 lr: 4.995e-05 max_mem: 5333M
[08/27 11:21:46 d2.utils.events]: eta: 2:58:55 iter: 219 total_loss: 3.367 loss_cls: 3.077 loss_box_reg: 0.2285 time: 0.4313 data_time: 0.2063 lr: 5.4945e-05 max_mem: 5333M
[08/27 11:21:56 d2.utils.events]: eta: 3:02:50 iter: 239 total_loss: 4.042 loss_cls: 3.413 loss_box_reg: 0.3222 time: 0.4385 data_time: 0.2894 lr: 5.994e-05 max_mem: 5333M
[08/27 11:22:04 d2.utils.events]: eta: 3:01:26 iter: 259 total_loss: 3.247 loss_cls: 2.757 loss_box_reg: 0.3311 time: 0.4360 data_time: 0.1814 lr: 6.4935e-05 max_mem: 5333M
[08/27 11:22:12 d2.utils.events]: eta: 2:57:46 iter: 279 total_loss: 3.523 loss_cls: 3.105 loss_box_reg: 0.3371 time: 0.4319 data_time: 0.1701 lr: 6.993e-05 max_mem: 5333M
[08/27 11:22:19 d2.utils.events]: eta: 2:56:26 iter: 299 total_loss: 2.915 loss_cls: 2.793 loss_box_reg: 0.2139 time: 0.4290 data_time: 0.1681 lr: 7.4925e-05 max_mem: 5333M
```

Ilustración 41 A-8: Start of training

```
[08/27 14:19:05 d2.utils.events]: eta: 0:00:30 iter: 25859 total_loss: 0.8155 loss_cls: 0.582 loss_box_reg: 0.1879 time: 0.4139 data_time: 0.1121 lr: 0.00025 max_mem: 5531M
[08/27 14:19:12 d2.utils.events]: eta: 0:00:30 iter: 25879 total_loss: 0.5167 loss_cls: 0.3663 loss_box_reg: 0.1226 time: 0.4139 data_time: 0.1582 lr: 0.00025 max_mem: 5531M
[08/27 14:19:20 d2.utils.events]: eta: 0:00:22 iter: 25899 total_loss: 0.4614 loss_cls: 0.3324 loss_box_reg: 0.1333 time: 0.4139 data_time: 0.1406 lr: 0.00025 max_mem: 5531M
[08/27 14:19:28 d2.utils.events]: eta: 0:00:14 iter: 25919 total_loss: 1.306 loss_cls: 0.7935 loss_box_reg: 0.2872 time: 0.4139 data_time: 0.1922 lr: 0.00025 max_mem: 5531M
[08/27 14:19:36 d2.utils.events]: eta: 0:00:06 iter: 25939 total_loss: 0.7883 loss_cls: 0.5866 loss_box_reg: 0.1751 time: 0.4139 data_time: 0.2044 lr: 0.00025 max_mem: 5531M
[08/27 14:19:44 d2.utils.events]: eta: 0:00:00 iter: 25955 total_loss: 0.5498 loss_cls: 0.5728 loss_box_reg: 0.1791 time: 0.4139 data_time: 0.1786 lr: 0.00025 max_mem: 5531M
[08/27 14:19:44 d2.engine.hooks]: Overall training speed: 25954 iterations in 2:59:01 (0.4139 s / it)
[08/27 14:19:44 d2.engine.hooks]: Total training time: 2:59:32 (0:00:31 on hooks)
```

Ilustración 42 A-9: End of training

Once finished, we will obtain the model, called model_final.pth.

6. We run the training validation snippets.

These fragments check by means of val.txt images (images not used during training) whether the model has been correctly trained. Although the model will recognise most signals correctly, 100% efficiency cannot be guaranteed.

7. We run the code fragment that detects the signals and identifies them.

As explained in 4.3 Sección de Google Colaboratory, each image is passed through the model, and for each detected signal it is established whether it is an existing signal or not, in which case a JSON is created with the detection information.

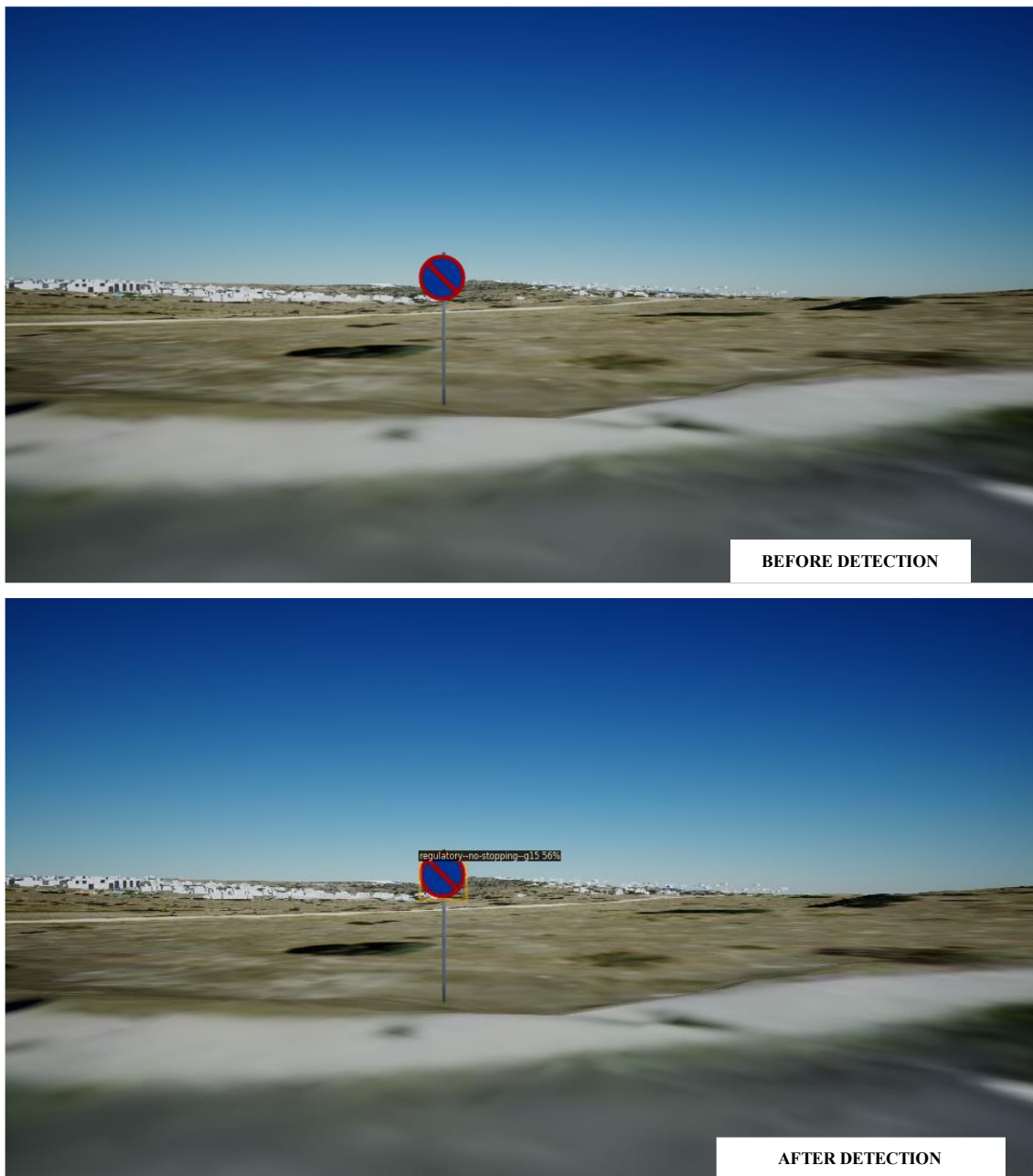


Ilustración 43 A-10: Comparison before/after detection

In this image we can see what the before and after images look like when they are passed through the detection model. This image will be saved in the output folder.

```
[
  {
    "camera": 2,
    "tiempo": "20220827-121932",
    "latitudDron": 40.5448613,
    "longitudDron": -4.0124445
  },
  {
    "camera": 3,
    "tiempo": "20220827-121940",
    "latitudDron": 40.5448926,
    "longitudDron": -4.0124446
  },
  {
    "camera": 6,
    "tiempo": "20220827-121940",
    "latitudDron": 40.5448926,
    "longitudDron": -4.0124446
  }
]
```

Ilustración 44 A-11: Signal detected multiple times example

8. We run the snippet that copies the output to google drive.

This snippet simply copies the output folder to the personal drive, so that it is not lost when the Google Colab development environment is closed.

A.6.2 Results

We have two types of results, firstly the images and JSONS obtained during the simulation, and secondly the results of the Google Colab part (images of the detected signals and a JSON for each different signal detected).

The results of the first part do not give much room for interpretation, as they are intermediate data intended to serve as input data for the second part of the project.

However, the results of the Google Colab part must be interpreted:

We have one output image for each input image but in addition we have a total of 20 JSON files with each detected signal, taking into account that there were 13 signals in the simulation, we can deduce the following:

- All signs have been detected except for the disabled parking sign and the tow truck sign has been confused with the bus stop sign.
- There are some signs in the simulation that have been identified as several different signs in the detection. The main reason for this problem is that the signs in the simulation are still simulated signs, they are not 100% perfect, either in shape, colour or fit (fit being understood as framing the content of the sign in the sign itself). This causes the model to detect unusual patterns in the signal in question and to be able to detect multiple signals in the same signal. In a real environment this is likely to happen very rarely.
- It detected 7 of the 8 different signals in the simulation, giving it 87.5% accuracy, where accuracy is defined as the proportion of correctly identified signals.
- It has detected 20 signals in total out of the 13 total signals that existed in the simulation, which gives it 65% effectiveness, effectiveness being understood as

the proportion of detected signals with respect to the total number of existing signals.

A.7 Conclusions

In this section we will evaluate the results obtained in the project, the possible future work and the different problems that have arisen during the development of the project.

It could be said that all the objectives proposed for the project have been met, but realistically, it should be noted that the title of the project is Automatic Object Detection and Geolocation from drones and the geolocation carried out is very primitive. This is mainly due to lack of time, as a project of this calibre is difficult to complete in a TFG period, and the result has forced to be less precise in this field. However, the result of this TFG is satisfactory and opens up very interesting lines of work for the future.

A.7.1 Lines of future work

Once the development of the project has been completed, a number of different future works can be proposed that would improve the implementation of this project and allow for a better detection system:

- Testing in real environments: to be able to move from using a simulated environment to testing this on real streets. This would involve getting all the necessary licences and taking extra precautions with the drone.
- Improving the detection system: The model is currently trained to detect traffic signs. The system could be improved by extending it to the rest of the elements of a road, such as traffic lights, lights or other vehicles. There are datasets in Mapillary that allow this line of future work to be considered.
- Pinpointing: Using positioning algorithms, the position of the signal could be approximated much more accurately. The project is designed to initiate this line of work, as the telemetry obtained from the images includes data necessary for these algorithms, such as the rotation of the camera, or the degree of stability at the time of the image.

A.7.2 Problems

The main problem has been to find a program or library that would allow a model to be trained on any computer. Most of them have many restrictions on their use (including Detectron itself). Because of Detectron's need for CUDA to train correctly, which means that any non-NVIDIA GPU cannot train correctly. This is why Google Colab was used. Another, albeit much smaller, problem has been false detections of signals in the simulated environment. This occurs because the model is trained to detect real signals, and the simulated signals have patterns that may not match the real ones. For example, there are signals that are not perfectly framed, or signals that lose a little bit of quality due to pixels. There is really no solution that is 100% effective against this problem, but I have tried to minimise the impact that this problem can have on a use case.