



# ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

TELÉCOMUNICACIÓN

Campus Sur  
POLITÉCNICA

## PROYECTO FIN DE GRADO

**TÍTULO:** Sistema de identificación y clasificación de objetos tridimensionales para la utilización de drones para la agricultura de precisión mediante Deep Learning.

**AUTOR:** Roberto Ríos Guzmán

**TITULACIÓN:** Telemática

**TUTOR:** Jesús Rodríguez Molina

**DEPARTAMENTO:** Ingeniería Electrónica y Telemática

VºBº

**Miembros del Tribunal Calificador:**

**PRESIDENTE:** Pablo Merodio Cámara

**TUTOR:** Jesús Rodríguez Molina

**SECRETARIO:** Hugo Alexer Parada Gélvez

**Fecha de lectura:**

**Calificación:**

El secretario,



# *Agradecimientos*

Este proyecto finaliza una etapa de mi vida muy especial, en el que he vivido muchas experiencias y me gustaría agradecer a todas las personas que me han acompañado a lo largo de esta etapa.

En primer lugar, agradecer a mi familia por acompañarme siempre y confiar en mí. En especial a mi hermana y mis padres por darme la oportunidad de obtener unos estudios, pero fundamentalmente por estar para ayudarme cuando lo he necesitado.

También agradecer a las personas de la universidad, como a *Jaime, Jose, Irene, Elia, López o Raquel*, entre muchos más, que me han acompañado en las asignaturas y en las anécdotas vividas. Igualmente, a mis amigos de toda la vida, en especial a *Juanjo*, que a pesar de los años pasados siguen conmigo.

Por último, agradecer a mi tutor *Jesús* por su ayuda en este Proyecto de Fin de Grado y a todos los profesores que me han ayudado a lo largo de toda la carrera.



## ÍNDICE

<b>1</b>	<b>INTRODUCCIÓN .....</b>	<b>11</b>
<b>2</b>	<b>MARCO TECNOLÓGICO.....</b>	<b>15</b>
2.1	Introducción.....	17
2.2	Drones .....	17
2.3	Simuladores.....	17
2.3.1	ROS.....	17
2.3.2	Gazebo .....	18
2.3.3	V-REP .....	18
2.3.4	AirSim .....	18
2.4	Redes neuronales.....	20
2.4.1	Redes neuronales convolucionales .....	22
2.4.2	Funciones de activación.....	24
2.5	Deep Learning.....	26
2.5.1	Aprendizaje por refuerzo .....	27
2.5.1.1	DQN .....	28
2.6	Librerías y herramientas Deep Learning .....	30
2.6.1	Anaconda.....	30
2.6.2	Tensorflow.....	31
2.6.3	Keras.....	31
2.7	Librerías Deep Reinforcement Learning .....	31
2.7.1	Keras-rl.....	32
2.8	Otras librerías de Python .....	32
2.8.1	Numpy .....	32
2.8.2	PyQt5.....	32
2.8.3	Gym .....	32
<b>3</b>	<b>ESPECIFICACIONES Y RESTRICCIONES DE DISEÑO ...</b>	<b>33</b>
3.1	Introducción.....	35
3.2	Especificaciones .....	35
3.3	Restricciones.....	35
<b>4</b>	<b>DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA.....</b>	<b>37</b>
4.1	Introducción.....	39

4.2 Instalación y configuración del entorno.....	39
4.2.1 AirSim .....	39
4.2.2 Entorno de trabajo Anaconda.....	41
4.2.3 Comunicación AirSim.....	42
4.2.4 Librerías de Deep Learning y Deep Reinforcement Learning	42
4.2.5 Otras librerías.....	43
4.3 Arquitectura aplicación .....	43
4.3.1 Paquete utilesAirSim .....	44
4.3.2 Paquete conduccionManual.....	44
4.3.3 Paquete DQN .....	47
4.3.4 Paquete interfazGrafica .....	53
4.3.5 Paquete principal.....	55
4.4 Solución en dron real .....	56
<b>5     RESULTADOS.....</b>	<b>59</b>
5.1 Introducción.....	61
5.2 Acceso y menú principal .....	61
5.3 Modo entrenamiento .....	62
5.3.1 Entrenamiento de red neuronal predeterminada.....	62
5.3.1.1 Pruebas fallidas .....	69
5.3.2 Interfaz gráfica modo entrenamiento.....	70
5.4 Modo validación.....	72
5.4.1 Validación de red neuronal predeterminada .....	72
5.4.2 Interfaz gráfica modo validación .....	76
5.5 Modo conducción autónoma.....	78
5.6 Modo pilotaje manual.....	82
<b>6     PRESUPUESTO .....</b>	<b>85</b>
6.1 Introducción.....	87
6.2 Presupuesto de la aplicación .....	87
6.3 Presupuesto implementación real.....	88
<b>7     CONCLUSIONES Y TRABAJOS FUTUROS.....</b>	<b>89</b>
<b>8     REFERENCIAS BIBLIOGRÁFICAS .....</b>	<b>93</b>
8.1 Referencias.....	95

## ÍNDICE DE FIGURAS

Figura 1: Imagen de profundidad .....	20
Figura 2: Esquema de una neurona artificial.....	21
Figura 3: Estructura de una red neuronal .....	22
Figura 4: Proceso de convolución de kernel en imagen de entrada .....	23
Figura 5: Proceso de Max-Pooling.....	24
Figura 6: Función de activación sigmoide .....	25
Figura 7: Función de activación tangente hiperbólica.....	25
Figura 8: Función de activación tangente ReLU .....	26
Figura 9: Esquema de Deep Reinforcement Learning .....	28
Figura 10: Interfaz gráfica de usuario de Anaconda .....	31
Figura 11: Lanzador Epic Games Launcher con Unreal Engine .....	40
Figura 12: Ventana Developer Command Prompt for VS 2019.....	40
Figura 13: Ambiente Blocks en Unreal Engine.....	41
Figura 14: Activación y desactivación de entorno Anaconda .....	42
Figura 15: Arquitectura de módulos de la aplicación .....	43
Figura 16: Esquema dirección dron .....	45
Figura 17: Esquema dirección perpendicular dron .....	46
Figura 18: Variación de $\epsilon$ respecto a pasos realizados .....	51
Figura 19: Pantalla principal Qt Designer.....	54
Figura 20:Arquitectura de módulos de la aplicación para implementación en entorno real.....	57
Figura 21: Pantalla inicial de la aplicación .....	61
Figura 22: Menú principal de la aplicación.....	62
Figura 23: Distribución del campo de entrenamiento.....	63
Figura 24: Primera parte de la gráfica recompensa por episodio.....	64
Figura 25: Segunda parte de la gráfica recompensa por episodio.....	65
Figura 26: Gráfica de episodios realizados respecto a los pasos .....	65
Figura 27: Media por acción .....	66
Figura 28: Evolución del tiempo de episodio respecto la recompensa por episodio .....	67
Figura 29: Número de pasos por episodio .....	67
Figura 30: Valor mínimo, máximo y promedio de observaciones respecto a los pasos .....	68
Figura 31: Evolución de las acciones respecto a los episodios .....	68
Figura 32: Comparación de entrenamientos con distintas redes neuronales ....	69
Figura 33: Comparación entre modelos con distintos obstáculos en el entorno	70
Figura 34: Interfaz gráfica para entrenar nuevo agente DQN .....	71
Figura 35: Buscar directorio donde guardar pesos.....	71
Figura 36: Escenario de validación de dificultad baja .....	72
Figura 37: Escenario de validación de dificultad media .....	73
Figura 38: Escenario de validación de dificultad alta.....	75

Figura 39: Validación de recompensa en distintos escenarios .....	76
Figura 40: Interfaz gráfica para validar agente DQN.....	77
Figura 41: Selección de carpeta para guardar resultados validación.....	77
Figura 42: Búsqueda de fichero con los pesos de la red en modo validación....	77
Figura 43: Interfaz gráfica inicial del modo conducción autónoma.....	78
Figura 44: Campo de simulación para la conducción autónoma .....	79
Figura 45: Comienzo de conducción autónoma.....	80
Figura 46: Imagen cargada en modo de conducción autónoma.....	81
Figura 47: Finalización del modo conducción autónoma.....	81
Figura 48: Directorio de salida en modo conducción autónoma .....	82
Figura 49: Interfaz gráfica para conducción manual.....	83
Figura 50: Visualización de la interfaz conducción manual alcanzando un objetivo .....	83
Figura 51: Resultado de la captura de imágenes en conducción manual.....	84

## ÍNDICE DE TABLAS

Tabla 1: Q-Table de valores iniciales.....	29
Tabla 3: Resultados de recompensa en escenario de dificultad baja.....	73
Tabla 4: Resultados de recompensa en escenario de dificultad media.....	74
Tabla 5: Resultados de recompensa en escenario de dificultad alta .....	75
Tabla 6: Posiciones del dron y puntos de referencia. ....	79
Tabla 7: Distancias en las coordenadas X e Y .....	80
Tabla 8: Costes de hardware en aplicación simulada.....	87
Tabla 9: Costes por mano de obra .....	87
Tabla 10: Coste total de aplicación simulada.....	87
Tabla 11: Costes de hardware en aplicación real .....	88
Tabla 12: Coste total de aplicación real .....	88



## ÍNDICE DE ECUACIONES

Ecuación 1: Ecuación de una neurona artificial.....	21
Ecuación 2: Sigmoide .....	24
Ecuación 3: Tangente hiperbólica.....	25
Ecuación 4: ReLU .....	26
Ecuación 5: Softmax.....	26
Ecuación 6: Ecuación de Bellman .....	29
Ecuación 7: Función de recompensa .....	48
Ecuación 8: Calculo distancia actual .....	49
Ecuación 9: Calculo de la distancia anterior .....	49

## ACRÓNIMOS

IA	Inteligencia Artificial
PIB	Producto Interior Bruto
NDVI	Normalized Difference Vegetation Index – Índice de Vegetación de Diferencia Normalizada
RGB	Red, Green, Blue – Rojo, Verde, Azul
GPS	Global Positioning System – Sistema de Posición Global
ROS	Robot Operating System – Sistema Operativo Robótico
CPU	Central Processing Unit – Unidad Central de Procesamiento
GPU	Graphics Processing Unit – Unidad de Procesamiento Gráfico
CNN	Convolutional Neural Network – Red Neuronal Convolucional
NED	North, East, Down – Norte, Este, Bajo
ReLU	Rectified Linear Unit – Unidad Lineal Rectificada
DDPG	Deep Deterministic Policy Gradient – Gradiente de Política Determinista Profundo
SARSA	State, Action, Reward, State, Action – Estado, Acción, Recompensa, Estado, Acción
DQN	Deep Q Network – Profunda Q Red
CUDA	Compute Unified Device Architecture - Arquitectura Unificada de Dispositivos de Cómputo

## Resumen

El propósito de este proyecto se centra en el diseño de una aplicación enfocada a la agricultura con la utilización de drones e Inteligencia Artificial (IA). Esta aplicación tiene como objetivo diseñar un caso de uso de un dron que sea capaz de, mediante conducción autónoma, muestrear y analizar terrenos usando toma de imágenes, gracias a un diseño económico a la hora de la implementación real.

Para alcanzar este objetivo se ha utilizado el motor gráfico de videojuegos Unreal Engine como simulador de ambientes y del dron, interactuando con él mediante AirSim. En consecuencia, se han diseñado mediante Deep Reinforcement Learning los elementos necesarios para que el dron aprenda su finalidad: llegar a las coordenadas indicadas por el usuario y mientras va tomando capturas del terreno. A la hora de implementar dichos algoritmos, se han utilizado librerías específicas para Deep Learning como Keras o Tensorflow o Keras-rl para Deep Reinforcement Learning.

Con ello, se han definido las acciones que el dron va a realizar y los estados que implican un error, como la colisión con obstáculos o si el dron no ha alcanzado la coordenada objetivo.

Tras alcanzar este objetivo, se ha procedido a analizar los resultados de las validaciones realizadas mediante la observación de la precisión y el acierto del dron simulado, evaluando así la viabilidad de la aplicación.

A su vez, se ha diseñado una interfaz gráfica que se emplea en la interacción, de una manera sencilla, del usuario con la aplicación desarrollada. Se ha añadido también la posibilidad de la conducción manual del propio dron y la opción de realizar los entrenamientos que el usuario desee. De esta forma, se podrá mejorar la efectividad y la validación de dichos resultados.

Para finalizar, se ha intentado dar una posible solución a la implementación de la aplicación en un dron real. De esta manera, se puede evaluar la viabilidad del proyecto, utilizando cámaras y sensores de un precio económico en el desarrollo virtual, pero sin perder efectividad en el objetivo.

## Abstract

The main purpose of this project focuses on the design of an application made for agriculture operations, using drones and Artificial Intelligence (AI). This application is aimed to design a drone use case that, through autonomous driving, would be able to sample and analyze any kind of land, thanks to the pictures that are being taken along its way. Due to its economic design, it may work not only in a virtual environment, but also in real life.

To achieve this goal, Unreal Engine has been used as the environment and drone simulator, interacting with the drone through AirSim. Consequently, the elements that were considered to be necessary for this task, have been developed using Deep Reinforcement Learning. Thus, the drone will learn its purpose: reaching the coordinates that were indicated by the user and taking pictures of the land while going all over it. These algorithms have been implemented through specific libraries for Deep Learning, such as Keras or TensorFlow or Keras-rl, for Deep Reinforcement Learning.

Therefore, we have defined the actions that the drone will perform and also the states that may imply a mistake, such as a collision with any object or whether the drone has not reached the defined.

After achieving this objective, the results of the validations previously performed were analyzed by observing the accuracy and precision of the simulated drone. With this, the viability of the application is evaluated too.

At the same time, a graphical interface has been designed, so the user could interact, in a simple way, with the application. It has also been included the possibility of manual driving of the drone itself and the option to perform whatever training the user desires. In this way, it will be possible to improve the effectiveness and validation of these results.

Finally, it has been tried to give a possible solution to the implementation of the application in a actual drone. Thereby the viability of the project can be evaluated, using economic cameras and sensors in the virtual development, but without losing effectiveness in the objective.

# Capítulo 1

---

## 1 INTRODUCCIÓN



En los últimos años la inteligencia artificial ha ido evolucionando de forma significativa. Gracias a la capacidad computacional actual, cada vez son más los proyectos en los que se utiliza para el aumento del rendimiento y automatización de los resultados. Junto a este campo, en la sociedad se ha ido aumentando el uso de los drones para conseguir distintas finalidades que antes no se podían realizar por los propios humanos o eran de extrema dificultad.

Por estos motivos, los campos de la inteligencia artificial junto al manejo de drones nos abren grandes posibilidades para avanzar y evolucionar en determinados sectores. Uno de los sectores con más importancia en una gran mayoría de países es la agricultura. Por ejemplo, en España [1] el sector agrícola representa el 4,2% de la distribución porcentual de los ocupados por sector económico y provincia en el primer trimestre de 2021, mientras que en 2019 aportó al PIB español un promedio del 2,65%.

Esta automatización se ha comenzado a realizar en dicho sector con el uso de distintos proyectos [2], como la cosechadora automática de Harvest Crop Robotics, la cual tarda aproximadamente ocho segundos por planta y cubre al día 3,2 hectáreas, realizando el trabajo de 30 cosecheros. Tenemos otro ejemplo en el uso de drones en agricultura para la fumigación [3] , como se ha empezado a realizar en el sur de la India, o el realizado por Aerocamaras llamado AeroHyb Hexacopter [4], siendo un dron híbrido capaz de trasladar cargas de hasta 5 kg durante dos horas. Esto nos indica que la integración de estos elementos realmente sí que tienen un avance significativo en este sector.

Actualmente, utilizando nuevas tecnologías para el análisis de imágenes como NDVI o imágenes RGB se pueden detectar cosechas en mal estado o realizar un correcto calendario de cultivo para cada parcela agrícola que se disponga. Para la realización de estas capturas de los terrenos, se suelen utilizar drones ya que se producen ciertas ventajas como el ahorro del tiempo en el muestreo y análisis del terreno.

Por estas razones, la motivación de este Proyecto de Fin de Grado es poder realizar un sistema con elementos económicos como un dron, junto a la automatización que permite la inteligencia artificial en dicho sector. Dentro de todas las posibilidades, en este proyecto se explica la manera de conseguir que un dron llegue a los distintos puntos dentro de un campo de agricultura de manera autónoma, para así lograr la captura de imágenes del cultivo.

Para estos desarrollos en el campo de la conducción autónoma, hay múltiples herramientas como los motores de juegos Unreal Engine o Unity, que actúan como simuladores de estos vehículos y de entornos controlados para realizar pruebas y ajustes para la posterior implementación en sistemas reales. Estos utilizan complementos dedicados a la inteligencia artificial como puede ser

AirSim y que permiten la conectividad entre los vehículos y la inteligencia artificial entrenada por el usuario.

En este proyecto, se contribuirá a hacer viable la conducción autónoma, utilizando un lenguaje de programación de alto nivel como Python y la comunicación entre los algoritmos de inteligencia artificial que se desarrollan con el entorno simulado. Estos algoritmos se basarán en la utilización de Deep Learning, más concretamente en el aprendizaje Deep Reinforcement Learning, con la finalidad de obtener un porcentaje de acierto lo suficientemente aceptable que permita llegar a los puntos objetivos de una manera eficaz y segura. Una vez resuelto el problema, se buscará desarrollar una interfaz gráfica para el uso del usuario en la que pueda indicar las coordenadas donde el dron deberá llegar sin sufrir daños y deba realizar las imágenes del terreno. Como parte final, se intentará buscar una posible solución para la implementación en un dron real sin perder la funcionalidad que se haya desarrollado.

Para ello, se dividirá el documento en varios capítulos. Una introducción al Proyecto Fin de Grado y sus temas se ofrece en esta primera sección. En segundo lugar, en el apartado llamado Marco tecnológico, se explicará el momento actual tanto de las tecnologías utilizadas como de las diferentes alternativas que se han utilizado en el desarrollo. En tercer lugar, en el apartado Especificaciones y Restricciones De Diseño, se definen las especificaciones necesarias junto con las limitaciones que se han encontrado durante el desarrollo del trabajo. Se continua con el desarrollo del proyecto junto con los resultados finales en los apartados nombrados como Descripción De La Solución Propuesta y Resultados, respectivamente. Antes del último apartado, se desglosará el presupuesto necesario para realizar dicha aplicación en el capítulo Presupuesto. Por último, se razonarán las conclusiones finales y futuros trabajos y modificaciones a realizar.

# Capítulo 2

---

## 2 MARCO TECNOLÓGICO



## 2.1 Introducción

En este capítulo se presenta una descripción básica acerca de la historia de los drones (apartado **¡Error! No se encuentra el origen de la referencia.**), los simuladores para la simulación de objetos de inteligencia artificial y en especial los drones (apartado 2.3), redes neuronales y sus tipos (apartado 2.4), sobre el Deep Learning en el apartado 2.5 y sobre las librerías utilizadas en este proyecto en los apartados 2.6, 2.7 y 2.8.

## 2.2 Drones

Aunque dispositivos casi similares a un dron fueron utilizados en el siglo XIX con fines militares, los primeros drones o aeronaves no tripuladas [5], se comenzaron a utilizar entre los años 1914 y 1918 durante el transcurso de la Primera Guerra Mundial, con de nuevo un uso militar. Los más destacados fueron el Aerial Target, construido en Reino Unido en 1916, con la función de servir como defensa contra los dirigibles alemanes y el Torpedo Aéreo Kettering, construido en 1917 [6]. Estos fueron evolucionando realizándose en 1943 la primera aeronave controlada por control remoto y saltando a 2006 donde se emitieron los primeros permisos para drones comerciales.

Más tarde, en 2010, salió al mercado el primer dron comercial y recreativo que se podía controlar remotamente vía wifi, el Parrot AR Dron. Hasta la actualidad se ha ido incrementando su uso, intentando el manejo de estos drones de manera autónoma con ayuda de distintos elementos como sensores y GPS.

## 2.3 Simuladores

En este apartado se exponen los simuladores que se investigaron para realizar este proyecto y finalmente se explica el simulador elegido, AirSim.

### 2.3.1 ROS

Robot Operating System [7] , conocido como ROS, es un framework flexible utilizado a nivel mundial para escribir software de robots. Este simulador contiene herramientas, bibliotecas y convenciones con la finalidad de crear un comportamiento robusto y complejo para distintas plataformas robóticas.

Este framework se creó para el desarrollo cooperativo de software robótico, intentando la combinación de distintas áreas de especialización de robótica. Por este motivo, ROS es un gran proyecto con muchos contribuyentes por la necesidad que se sintió de instaurar un marco de colaboración abierta.

Por consiguiente, es una plataforma que permite la sencilla integración de paquetes construidos por distintas entidades, ya que es una plataforma modular basada en nodos y consta con decenas de miles de usuarios en todo el mundo.

Al ser una capa que permite desarrollar aplicaciones para robots y tratarse de alto nivel, tiene la ventaja de no ser necesario tener conocimiento del hardware del robot sobre el cual se va a desarrollar la aplicación. Esta ventaja es un punto muy importante ya que cada elemento vendrá definido por su fabricante, pero no afectaría a la aplicación. Otra gran ventaja sería la posibilidad de desarrollar estas aplicaciones en lenguajes de programación como Python o C++.

### **2.3.2 Gazebo**

Gazebo [8] es un simulador en 3D con gráficos de alta capacidad y gratuito que permite la capacidad de simular entornos de manera precisa y eficiente. Además, proporciona simulación de las físicas con mayor grado de fidelidad, una cantidad de sensores e interfaces para los usuarios y programas.

Por estas razones, es un simulador con una gran comunidad y dispone de variedad de modelos pudiendo utilizar los creados por otra gente en tus proyectos. Están disponibles las opciones de utilizar motores de alto rendimiento como ODE o Bullet, o la modelación de sensores como telémetros o cámaras.

Por ello, los principales usos de este software son el testeo de algoritmos, diseño de robots y testeo de funcionamientos en escenarios realistas.

### **2.3.3 V-REP**

V-REP [9] es un programa para simular robots que está disponible para Linux, Windows y Mac OS, siendo gratuito y de código abierto. Este software realiza simulaciones de cada una de las piezas que forman un robot pudiendo controlar el movimiento del mismo o configurar sus distintos sensores. Admite distintos lenguajes de programación como Java, Python, C++ o Matlab.

### **2.3.4 AirSim**

AirSim [10] es el simulador elegido para este proyecto ya que es un software específico para la simulación de drones y coches; así pues, se describe con más detalle. Este simulador desarrollado por Microsoft está construido sobre Unreal Engine siendo de código abierto, multiplataforma y admitiendo conocidos controladores de vuelo como PX4 o ArduPilot. También se puede utilizar en Windows y en Linux, además de utilizarse otros simuladores como Unity, aunque no esté tan desarrollado para este motor y sea de momento experimental.

Está desarrollado como un complemento de Unreal Engine que se puede añadir a cualquier proyecto de esta plataforma y con el principal objetivo de la

investigación de IA para poder realizar experimentos, especialmente con algoritmos de Deep Learning, en proyectos de vehículos autónomos.

Al estar enfocado especialmente en la simulación de drones y coches contiene una API para interactuar con el tipo de vehículo que elijamos, estando diseñada para realizar los proyectos tanto en Python como en C++. Esta API nos permite realizar múltiples opciones, dividiéndose en diferentes APIs con distintas funciones como la modificación de las condiciones climáticas, el horario del día, la obtención de imágenes con múltiples cámaras o la utilización de varios tipos de sensores y procesamiento de los respectivos datos. En cuanto a las APIs utilizadas en este proyecto, destacan las necesarias para obtención de imágenes y para la obtención de datos de sensores.

En la API disponible para la obtención de imágenes destacan las posibilidades de elección entre distintas cámaras como imágenes RGB, de profundidad o infrarrojos, entre otras. Permite la obtención de la imagen en cualquier momento donde se encuentre el dron y de múltiples tipos al mismo tiempo, pudiendo guardarlas en la ruta que se indique y en distintos formatos.

Hay que resaltar que AirSim utiliza el sistema de coordenadas NED en el que moverse al norte sería +X, +Y al este y +Z hacia abajo mientras que en Unreal Engine el sistema de coordenadas varía en el eje Z, ya que una variación positiva en el eje Z implica moverse hacia arriba. Cabría destacar una característica importante como es que AirSim trabaja en metros mientras que Unreal Engine trabaja en centímetros.

AirSim permite una gran personalización de cada entorno con la modificación del fichero predeterminado que incorpora llamado *settings.json*. En este fichero, indicamos la utilización de un coche o un dron, el punto inicial donde comenzará nuestro dron o la cámara activada en cada momento, entre otras múltiples posibilidades de personalización que admite este software.

En cuanto a los elementos y sensores disponibles por AirSim en el manejo de drones, destacan para nuestro proyecto los siguientes:

- GPS: Este sensor permite saber al dron en qué punto se encuentra en cada momento. Gracias a este sensor se han implementado distintas utilidades como el modo de vuelta a casa, en el que el dron guarda su punto de inicio y sabe regresar en caso de pérdida del mismo mientras se está conduciendo. Otra función es la posición de bloqueo, en la que se le ordena al dron una posición fija de altura y es capaz de mantenerse sabiendo en todo momento su altura actual. AirSim permite utilizarlo a través de sus APIs siendo fundamental para la realización de actividades de conducción autónoma

- Cámara de profundidad: este simulador permite la utilización de este tipo de cámaras, entre otras más, que permitirá la realización de imágenes en blanco y negro, mostrando cada pixel la distancia al objeto en metros. El rango de distancia está definido entre 0 y 100 metros siendo un valor de 0 correspondiente a 0 metros y 255 correspondiente a 100 o más metros; de otra manera, cuanto más negro más cerca está el objeto y cuanto más lejos más blanco. Un ejemplo se ve la Figura 1



**Figura 1: Imagen de profundidad**

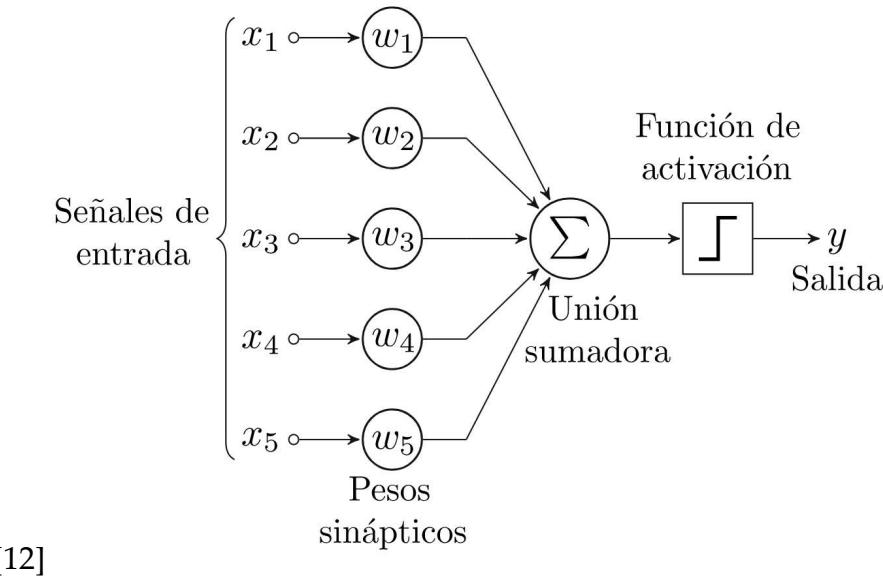
Por último, destacar la posibilidad de manejar las componentes más importantes de los drones en especial el ángulo yaw. Este componente se definirá en AirSim como el ángulo entre el eje X y la dirección del dron, midiendo el giro del dron sobre sí mismo.

## 2.4 Redes neuronales

Antes de describir el concepto de Deep Learning o las técnicas utilizadas para realizar el proyecto, hay que definir qué son las redes neuronales. En primer lugar, se define el concepto de neurona en el campo de la inteligencia artificial siendo estas un sistema matemático que pretende imitar el comportamiento de una neurona biológica [11]. Como se observa en la Figura 2, una neurona artificial tiene las siguientes partes y el siguiente funcionamiento:

- Una neurona tiene un número N de señales de entradas, en el caso de la Figura 2, n es igual a 5.
- Estas señales con su respectivo valor se multiplicarán cada una por un valor determinado a esa entrada, llamados pesos sinápticos y representados como w. Por ende, habrá tantos pesos en una neurona como entradas x disponga. En el ejemplo de la Figura 2 observamos que son cinco los pesos.

- Se realiza el sumatorio de los valores anteriores resultantes y se aplica la función de activación correspondiente en dicha neurona, dando como resultado el valor y de salida.



[12]

**Figura 2: Esquema de una neurona artificial**

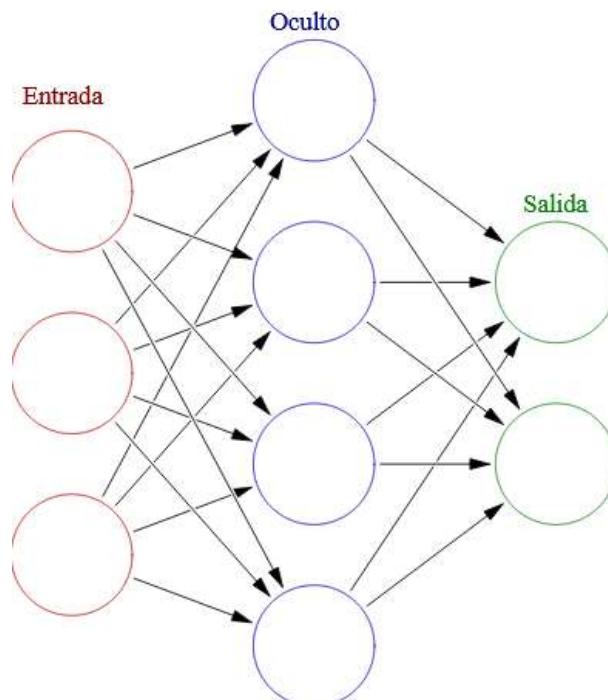
Según los parámetros y comportamientos explicados y definiendo como  $N$  el número de entradas,  $x_n$  como el valor de la señal de cada entrada  $n$ ,  $y$  como el valor de salida,  $w_n$  como el valor del peso para cada entrada  $n$ ,  $g(x)$  como la función de activación, podremos desarrollar matemáticamente la ecuación

$$y = g\left(\sum_{n=1}^N x_n * w_n\right)$$

**Ecuación 1: Ecuación de una neurona artificial**

Una vez explicado qué es una neurona se define a continuación el concepto de red neuronal como el conjunto de neuronas artificiales, conectadas entre sí y transmitiéndose señales. Cuando se dispone de múltiples neuronas se reagrupan en capas, clasificándose según su lugar en la red en tres tipos como se puede observar en la Figura 3

- Capa de entrada, corresponden a las entradas de la red.
- Capas ocultas, corresponden a las capas intermedias entre la capa de entrada y de salida.
- Capa de salida, correspondiente a las salidas del sistema.



[13]

**Figura 3: Estructura de una red neuronal**

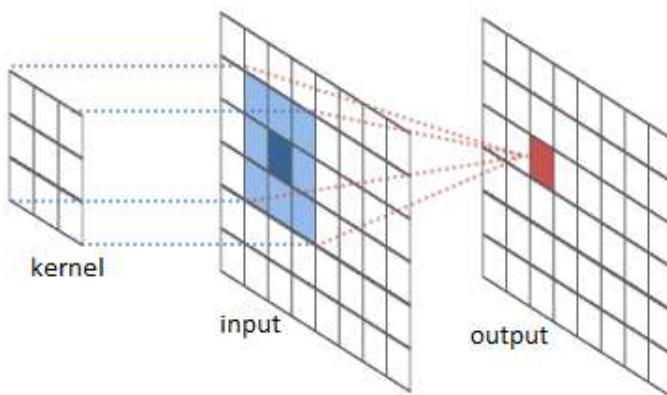
Con estas capas se forma una red neuronal y siguiendo la fórmula de una neurona artificial, las entradas de cada neurona serán las salidas de las neuronas de la capa anterior. Por esa razón, la finalidad de una red neuronal es que a través de unos pesos y unos valores encuentre una determinada solución. Con esta solución, se actualizarán los valores de los pesos progresivamente para que según que entradas tenga el sistema se generen unas determinadas salidas.

#### 2.4.1 Redes neuronales convolucionales

Una red neuronal convolucional, CNN, es un tipo de red neuronal artificial que intenta imitar el córtex visual del ojo humano para identificar distintas características en las entradas y poder reconocer objetos.

Esto implica que la entrada a este sistema son los píxeles de una imagen multiplicándose por tres en caso de ser una imagen a color, ya que se necesitarían tres canales para el rojo, otro para el verde y uno para el azul mientras que sólo se multiplicaría por uno en caso de que estuviese en una escala de grises. Estos píxeles tendrán un valor entre 0 y 255.

Una vez obtenida la imagen de entrada se realizaría la convolución; en otros términos, se va recorriendo grupos de píxeles de la imagen de entrada de izquierda a derecha y de arriba abajo, realizando el producto escalar con una matriz menor denominada kernel. Este proceso se representa en la Figura 4.



**Figura 4: Proceso de convolución de kernel en imagen de entrada**

El proceso mostrado en la Figura 4: Proceso de convolución de kernel en imagen de entrada es el correspondiente a la aplicación de un kernel, pero en la realidad se aplicaría varios en un conjunto llamado filtros. En caso de ser una imagen RGB, deberíamos aplicar el triple de filtros uno por cada canal, sumándose el resultado para obtener la misma solución que obtendríamos con una escala de grises. Con este proceso, obtendríamos una nueva imagen por cada kernel del filtro que irán mostrando ciertas características.

Una vez conseguidas el mismo número de imágenes como de filtros se definan, se aplica una función de activación como las definidas en el apartado 2.4.2 a cada una de estas matrices de píxeles. Por ello, una vez realizado este proceso, se obtendrán tantos mapas de características como filtros se hayan aplicado anteriormente. A su vez, lograremos las características básicas de la imagen, como líneas o curvas. Para imágenes más complejas, habrá que realizar más convoluciones repitiendo todo este proceso. De esta forma, se perfeccionará la red convolucional.

Previamente, se deberá llevar a cabo la acción de *Subsampling* después de cada convolución realizada. Este proceso consiste en la reducción del tamaño de los píxeles de los mapas de características obtenidos. De no realizar dicho proceso, se dispondría de un gran número de píxeles, lo cual implicaría mayores neuronas para la red neuronal y, en definitiva, mayor procesamiento. Este paso se suele ejecutar utilizando *Max-Pooling* siendo este el tipo de Subsampling más utilizado, el cual consiste en utilizar una matriz de dimensiones inferiores recorriendo de izquierda a derecha y arriba hacia abajo cada uno de los mapas de características, obteniendo el máximo valor entre los valores abarcados por la dimensión de esta matriz Max-Pooling. Este desarrollo se puede observar en la Figura 5 donde se ha conseguido reducir el número de píxeles de los mapas de características, pero se han mantenido a su vez las más importantes.



**Figura 5: Proceso de Max-Pooling**

Una vez aplicadas las capas de convolución que sean necesarias, se conectan a la red neuronal tradicional explicada en el apartado 2.4. Para ello, la última capa oculta donde se ha realizado el *Subsampling* se convierte en la entrada de la red neuronal, con el mismo funcionamiento explicado en el anterior apartado.

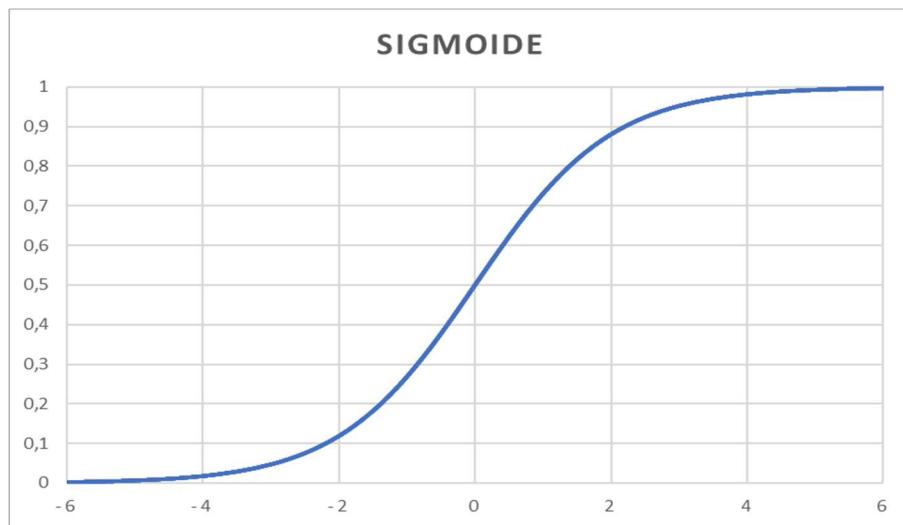
#### 2.4.2 Funciones de activación

En este apartado se explican las funciones de activación más importantes y utilizadas en las redes neuronales artificiales descritas. Estas funciones devuelven valores entre 0 y 1 o -1 y 1 para conseguir simplificar el coste computacional, consiguiendo funciones simples.

- Sigmoide: está función transforma los valores en un rango entre 0 y 1, teniendo los valores más alto valores aproximados a 1 y los muy bajos a 0. La función será la representada en la Figura 6 y es definida matemáticamente según la Ecuación 2.

$$f(x) = \frac{1}{1 + e^{-x}}$$

**Ecuación 2: Sigmoide**

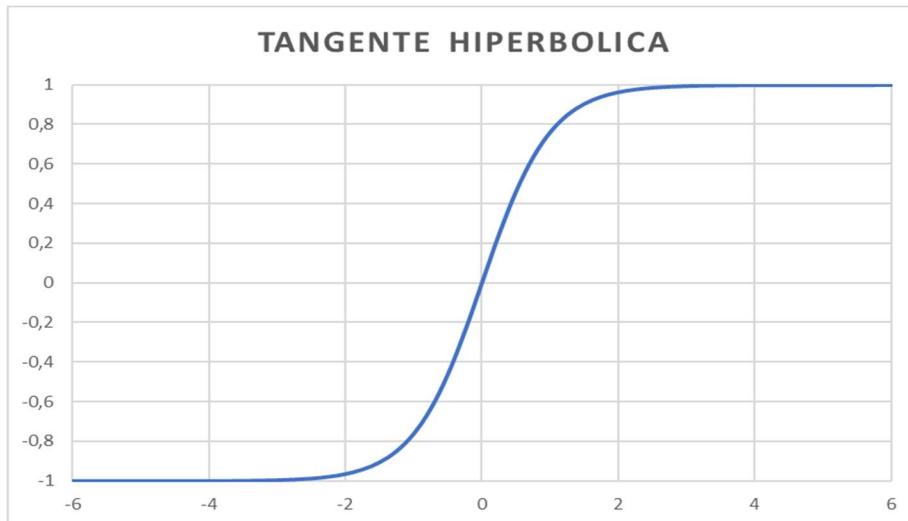


**Figura 6: Función de activación sigmoide**

- Tangente hiperbólica: transformará los valores a una escala entre -1 y 1. Función similar a la función Sigmoide como se observa en la Figura 7 y con una representación matemática como la Ecuación 3.

$$f(x) = \frac{1}{1 + e^{-x}}$$

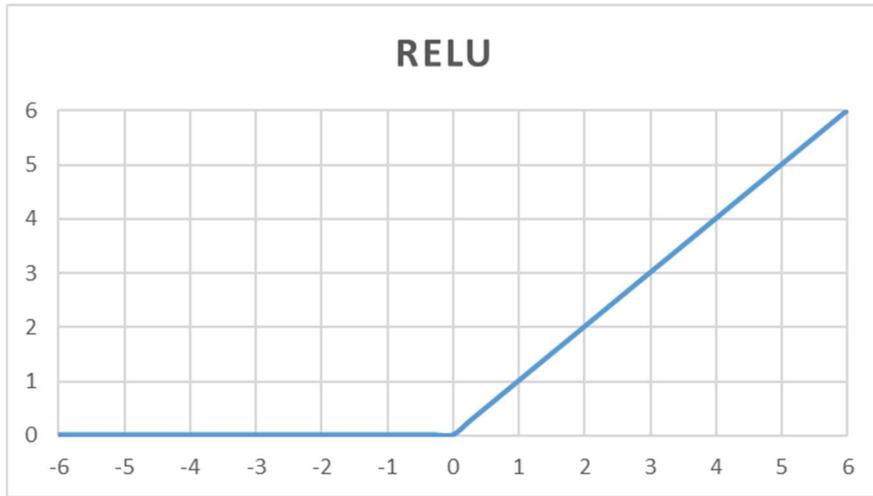
**Ecuación 3: Tangente hiperbólica**



**Figura 7: Función de activación tangente hiperbólica**

- ReLU: función Rectified Lineal Unit que transforma los valores introducidos anulando los negativos y dejando los positivos con sus propios valores. Es una función que no está acotada y suele ser utilizada sobre todo para redes neuronales convolucionales. Se representa según la Figura 8 y se expresa matemáticamente según la Ecuación 4.

$$f(x) = \max(0, x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

**Ecuación 4: ReLU****Figura 8: Función de activación tangente ReLU**

- Softmax: transforma las salidas a una representación en forma de probabilidades, siendo el sumatorio de todas las probabilidades de las salidas de uno. Está acotada en el rango entre 0 y 1, se puede expresar matemáticamente según la Ecuación 5.

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}$$

**Ecuación 5: Softmax**

## 2.5 Deep Learning

El Deep Learning [14] es un subcampo del Machine Learning que se ocupa de algoritmos inspirados en redes neuronales artificiales. Este subcampo se basa en el aprendizaje de un ordenador u otra pieza de hardware similar, configurando distintos datos y reconociendo patrones mediante el uso de las capas intermedias de procesamiento.

El Deep Learning destaca al no requerir reglas programadas previamente, si no que el propio sistema es capaz de aprender por sí mismo a través de un entrenamiento previo, destacando el uso de las redes neuronales artificiales definidas anteriormente entrelazadas entre sí para el procesamiento de la información.

Siguiendo este modelo se pueden definir dos tipos de entrenamiento para una red neuronal artificial:

- Aprendizaje supervisado: se basa en la generación de un modelo predictivo gracias a los datos de entrada y salida. Los datos de entrada estarán etiquetados y clasificados; esto es, se saben a qué grupo o categoría pertenecen los datos de entrada, llamándose este conjunto grupo de entrenamiento. El modelo de red neuronal artificial se encarga de comparar los resultados que ha precedido previamente de los datos de entrada del grupo de entrenamiento con los datos de salida correspondientes, ajustando sus pesos para reducir el error cometido y mejorando de este modo la precisión del acierto. Este proceso se repite hasta conseguir que la diferencia entre la salida computada y la salida correspondiente al grupo de entrenamiento sea considerablemente pequeña.
- Aprendizaje no supervisado: se basa en la generación de un modelo predictivo gracias a los datos únicamente de entrada. El grupo de entrenamiento sólo contendrá los datos de entrada y tendrá como objetivo organizarlos en salidas consistentes, en otras palabras, entradas muy parecidas. Este proceso se basa en la extracción de características de las entradas, agrupándolas en clases según la similitud de estas características.

### 2.5.1 Aprendizaje por refuerzo

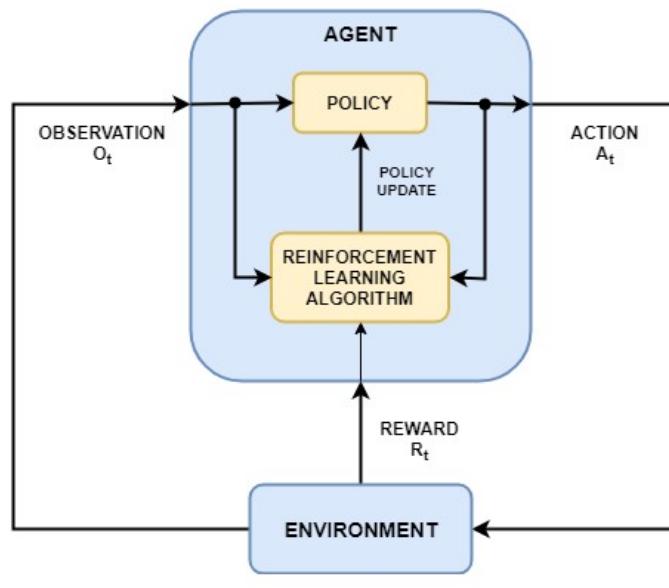
Dentro de aprendizaje supervisado y no supervisado hay varios tipos de técnicas, como por ejemplo Deep Reinforcement Learning o aprendizaje por refuerzo, la cual es una técnica utilizada dentro del aprendizaje supervisado y será la empleada para la realización de este proyecto. El Deep Reinforcement Learning consiste en la recopilación de estados y acciones realizadas previamente y la evaluación del resultado obtenido, llamado recompensa. Dicha recompensa se maximizará seleccionando como mayor calidad las acciones que han obtenido una recompensa de mayor valor. A diferencia con el aprendizaje supervisado convencional en que se determinaba las mejores acciones para un estado en el grupo de entrenamiento, en este aprendizaje se evaluará el resultado de la acción tomada en cada estado.

Para un estado determinado no se indican las acciones que se pueden ejecutar, sino que se le da la posibilidad al sistema de realizar todas las acciones posibles y observar la recompensa obtenida, explorando así todas las posibilidades del entorno.

En definitiva, los sistemas de aprendizaje por refuerzo, tendrán los siguientes elementos:

- Agente: es el elemento que interactúa con el entorno y realiza las acciones.
- Entorno: es el sistema que interactúa con el agente; es decir, el agente realiza una acción en el entorno en el que se encuentra y está tendrá una consecuencia en el entorno, observando el efecto.
- Políticas: elemento que hace referencia a la forma de comportarse del agente. Da el sentido al agente sobre la toma de decisiones y determina su comportamiento. Este elemento se irá actualizando según el tipo de algoritmo elegido para este aprendizaje y optimizando las elecciones de acciones a realizar.
- Función recompensa: elemento más característico de este tipo de aprendizaje. Determina la recompensa por cada acción en un estado y por tanto determina el objetivo del aprendizaje.

En la Figura 9 se observa una explicación gráfica de este procedimiento



**Figura 9: Esquema de Deep Reinforcement Learning**

El entorno tiene que ser explorado con el fin de aprender a seleccionar las mejores acciones y a su vez, descubrir cuales son las acciones con mayores recompensas para el futuro.

### 2.5.1.1 DQN

Existen varios algoritmos de aprendizaje por refuerzo, pero en este apartado se explicará el utilizado para este proyecto, el algoritmo *Deep Q-Network* o DQN.

Como primer paso para entender este cálculo se explicará el algoritmo Q-Learning, ya que es la base de DQN. Se basa en la utilización de una tabla llamada

Q-table que tendrá una fila por cada estado posible y una columna por cada acción que pueda realizar el agente. Cada celda de esta tabla comenzará con los valores inicializados a 0 y a medida que el agente interactue con el entorno mejorará estos valores Q hasta converger en valores óptimos. Se puede ver un ejemplo en la Tabla 1, definiendo S como los estados y su subíndice el número del estado mientras que A será la acción y su subíndice el número de acción que puede realizar.

**Tabla 1: Q-Table de valores iniciales**

Estado/Acción	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>
S <sub>1</sub>	0	0	0	0
S <sub>2</sub>	0	0	0	0
S <sub>3</sub>	0	0	0	0
S <sub>4</sub>	0	0	0	0

Estos valores se irán ajustando por la ecuación de Bellman que se puede observar en la Ecuación 6 indicando  $s$  como el estado,  $a$  como acción,  $Q(s,a)$  como valor actual de la celda a actualizar de la tabla Q-Table,  $Q^*(s,a)$  como el nuevo valor de la celda,  $R$  como la recompensa recibida,  $\alpha$  la tasa de aprendizaje llamada learning rate,  $\lambda$  como la tasa de descuento que tendrá en cuenta la recompensa a corto y largo plazo y  $\max Q(s')$  como el mayor valor óptimo esperado.

$$Q^*(s,a) = Q(s,a) + \alpha[R + (\lambda \max Q(s')) - Q(s,a)]$$

#### Ecuación 6: Ecuación de Bellman

Por esa razón, según el valor  $\epsilon$  se elige una acción,  $A_t$  para el estado actual,  $S_t$ , ejecutándose y obteniendo una recompensa,  $R_t$ , junto al siguiente estado,  $S_{t+1}$ . Entonces, elige el valor Q más alto para  $S_{t+1}$ , esto es, la acción con valor Q igual a  $\max Q(s')$ . Con estos parámetros ya se puede aplicar la Ecuación 6 y así va actualizando todos los valores según va avanzando con el paso del tiempo.

Como se puede observar, Q-Table está construida con tantas filas como estados haya en un entorno y por eso se desarrolla DQN. En un entorno real o más complejo, sería imposible o tendría un alto coste computacional tener en memoria todos los posibles estados, en consecuencia, se sustituye la Q-Table por una red neuronal artificial Q Neuronal Network. Esta red tendrá de entrada una imagen

y de salida las mismas neuronas que las posibles acciones que puede realizar el agente en un entorno actualizando los pesos de la misma manera que se actualizaban los valores Q de la Q-Table. Estás neuronas de salida mostrarán el valor Q para una acción según el estado de entrada de la red.

En este tipo de arquitectura se añaden dos elementos a parte de la mencionada red neuronal llamada Q Neuronal Network, el componente Experience Replay y otra red neuronal idéntica llamada Target Neuronal Network.

El elemento Experience Replay selecciona una acción para el estado actual según el valor de  $\epsilon$ , obteniendo una recompensa y el siguiente estado, guardando la acción, el estado actual, el estado futuro y la recompensa y llamándolo observación. Una vez obtenido suficientes observaciones para entrenar, tomamos un lote de estas aleatoriamente y se ingresan en ambas redes. De la red Q Neuronal Network se obtiene el valor Q para esa acción y ese estado, llamándose Valor Q previsto. Para la red Target Neuronal Network, se introduce del lote el siguiente estado y predice el mejor Q posible, llamándose Valor Q objetivo.

Con el Valor Q previsto, la recompensa y el Valor Q objetivo, calculamos la perdida llamada Loss y se entrena la red Q Neuronal Network. Después, tras un número de pasos copiaremos los pesos en la red Target Neuronal Network para ir así actualizando los valores.

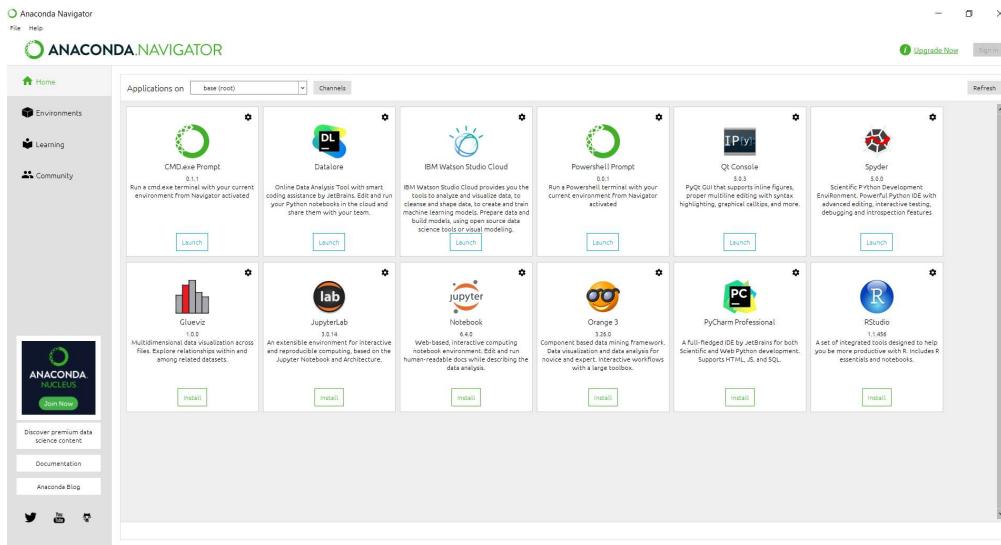
## 2.6 Librerías y herramientas Deep Learning

Este proyecto se realiza en casi su totalidad en Python ya que es un lenguaje con multitud de herramientas y librerías para la inteligencia artificial además de ser uno de los dos lenguajes soportados para AirSim, el simulador elegido. En los siguientes apartados se explican estas herramientas y librerías más importantes del proyecto.

### 2.6.1 Anaconda

Anaconda [16] es una distribución libre y abierta desarrollada para los lenguajes de R y Python, utilizada principalmente para la ciencia de datos y aprendizaje automático. Tiene multitud de ventajas en las que resaltan principalmente su capacidad de crear entornos independientes y personalizarlos instalando en cada uno de ellos las librerías que se necesiten. Además, contiene diversos entornos de trabajo como Jupyter, JupyterLab o Spyder. Dispone de una interfaz gráfica

de usuario sencilla de utilizar en la que se puede realizar administrar los entornos y sus paquetes de manera intuitiva y rápida.



**Figura 10: Interfaz gráfica de usuario de Anaconda**

### 2.6.2 Tensorflow

Tensorflow [17] es una biblioteca open source utilizada para aprendizaje automático, desarrollado por Google para poder construir y entrenar redes neurales artificiales. Esta librería deriva las operaciones de las redes neuronales en arrays multidimensionales de datos llamados tensores proviniendo de estos elementos su nombre. Proporciona APIs para múltiples lenguajes como Java, C++ o como la utilizada en este proyecto Python.

### 2.6.3 Keras

La librería Keras [18] es una biblioteca de código abierto escrita en Python para trabajar con redes neuronales. Se ejecuta sobre Tensorflow y tiene la ventaja de tener una sintaxis amigable para el usuario, modular y extensible. Para este proyecto, al ser necesarias las redes neuronales convolucionales, es importante contar con esta librería ya que ofrece soporte sobre este tipo.

## 2.7 Librerías Deep Reinforcement Learning

A parte de las librerías utilizadas para Deep Learning, Python dispone de librerías específicas para realizar Deep Reinforcement Learning. En este apartado, se explicará la librería Keras-rl que se eligió en este trabajo.

### 2.7.1 Keras-rl

Keras-rl [19] es una librería diseñada en 2016 que implementa algunos algoritmos de Deep Reinforcement Learning integrándose con la librería Keras. Algunos de esos algoritmos son DDPG, SARSA o DQN, utilizando el último para este trabajo. Esta librería es sencilla de instalar y tiene libertad para modificar o añadir código si fuese necesario, siendo una librería muy útil para el trabajo que se ha realizado.

Trabaja a su vez con Keras y Tensorflow; por tanto, contiene mucha información y multitud de soluciones a problemas en sus páginas oficiales, así como tutoriales. En Keras-rl las partes más importantes serán definir el modelo de las redes neuronales, la política a utilizar y la memoria, además de otros parámetros importantes que se explican con más detalle en el apartado 4.

## 2.8 Otras librerías de Python

Como último apartado se explican las librerías adicionales que se han necesitado para el resto de funciones como la interfaz gráfica, manejo de imágenes o arrays.

### 2.8.1 Numpy

Numpy [20] es una biblioteca en Python para el manejo de vectores y matrices, siendo de código abierto además de tener múltiples colaboradores. Es capaz de manejar objetos de grandes dimensiones y posee una gran cantidad de funciones matemáticas de alto nivel.

### 2.8.2 PyQt5

PyQt5 [21] es uno de las librerías más importantes para desarrollar interfaces gráficas tanto en C++ como en Python. Contiene más de 620 clases que cubren interfaces gráficas de usuario con comunicación con red, bases de datos SQL o navegación web. Está disponible para Windows, Linux y Mac OS además de funcionar para iOS y Android.

### 2.8.3 Gym

Gym [22] es una herramienta desarrollada por OpenAI, siendo está una compañía de investigación de inteligencia artificial lanzada en 2015 por Elon Musk y Sam Altman entre otros. Esta herramienta contiene varios entornos y se busca estandarizar la manera de definir estos mismos, dicho de otra manera, es una librería utilizada a nivel global para la creación de los entornos.

# Capítulo 3

---

## 3 ESPECIFICACIONES Y RESTRICCIONES DE DISEÑO



### 3.1 Introducción

En este capítulo se definen las especificaciones del material utilizado para desarrollar este proyecto junto con las restricciones fundamentales que se han encontrado para realizar el mismo.

### 3.2 Especificaciones

La aplicación se codifica en el lenguaje de programación Python, versión 3.6. Se utiliza el motor gráfico Unreal Engine versión 4.25, junto con el simulador AirSim versión 1.4. La aplicación debe permitir la simulación de vuelo de drones a coordenadas específicas con una altura fija, realizando esta función tanto por conducción autónoma como con conducción manual, realizando fotos con la cámara horizontal del terreno inferior del dron para su posterior análisis.

Otra especificación es la escalabilidad del proyecto para futuras mejoras y poseer una interfaz gráfica agradable y de sencilla utilización para el usuario. Ya que los elementos utilizados por el dron en la simulación por AirSim son gratuitos y se busca que sea una aplicación accesible, se debe encontrar la manera más efectiva de poder cumplir las anteriores especificaciones siendo a su vez económicamente viable para la implementación en un sistema real.

### 3.3 Restricciones

Las restricciones del proyecto vienen producidas por el material hardware, es decir, por el dron real junto con los elementos y sensores que necesita y por el procesador necesario para completar los tiempos de entrenamiento de la red en simulador.

Por la primera restricción, el sistema desarrollado no se ha podido implementar en un dron físico por la falta de acceso a drones reales junto con elementos como la cámara de profundidad, la cámara RGB y el sensor GPS.

En relación al procesamiento, una restricción es la cantidad de porcentaje de uso tanto de GPU como de CPU por la realización de los entrenamientos de la red y la utilización al mismo tiempo de Unreal Engine. Esto implica a su vez la necesidad de una gran inversión de tiempo para realizar entrenamientos,

aumentando el tiempo del mismo según se deseé perfeccionar los resultados producidos.

# Capítulo 4

---

## 4 DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA



## 4.1 Introducción

En este capítulo se explica la solución propuesta para la creación de una aplicación enfocada a la agricultura. Primero, se explica la instalación necesaria para llevar a cabo este desarrollo. Se continua con la solución consistente en el desarrollo de un dron funcionando en el simulador AirSim, mediante inteligencia artificial, que sea capaz de llegar a distintos objetivos y realizar capturas de imágenes para la investigación y análisis de estas. También se habilitarán opciones que permitan al usuario entrenar su propia red neuronal y agente DQN para la optimización del algoritmo, junto con la validación de la misma. Como última opción, se permitirá la conducción manual del dron para dar la posibilidad de ajustes más finos que los que se realizan con esta conducción autónoma. Por último, se desarrolla una posible solución para un entorno real con un dron, en otras palabras, no hay necesidad de la utilización de un simulador.

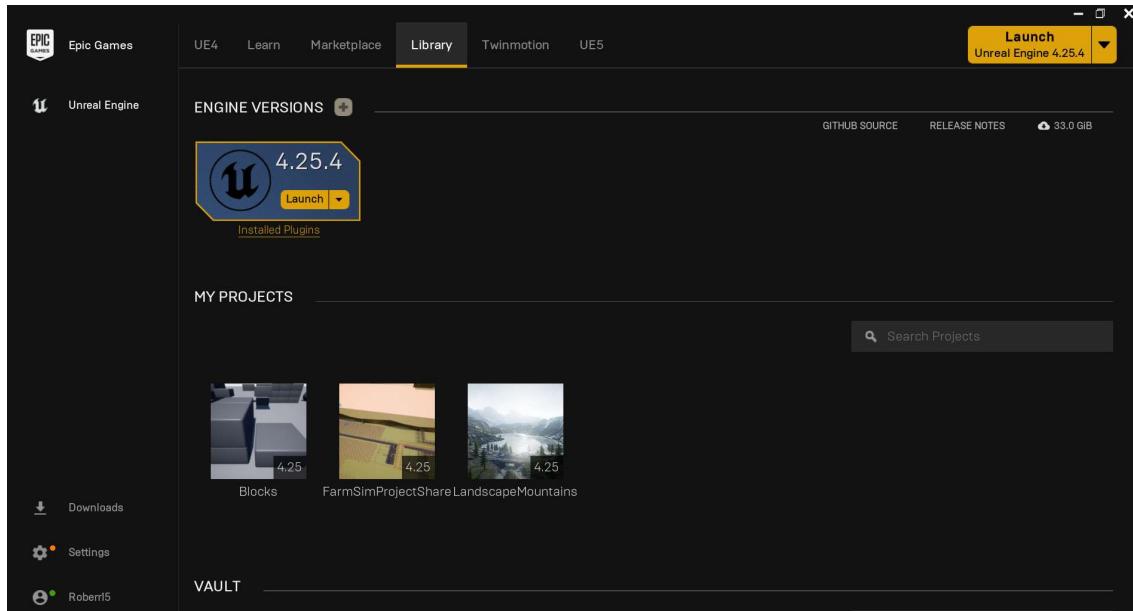
## 4.2 Instalación y configuración del entorno

En este apartado se explica la instalación y configuración del entorno necesario para realizar la solución propuesta. Se divide en subapartados donde se agrupa la instalación en distintos campos del proyecto.

### 4.2.1 AirSim

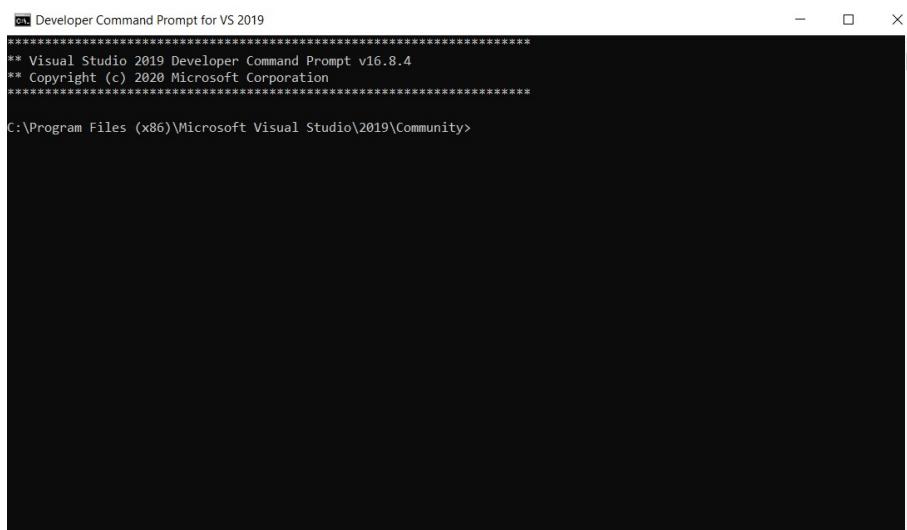
En este primer apartado de la instalación y configuración del entorno, se explica el elemento correspondiente al simulador y complemento utilizado, Unreal Engine y, en especial, AirSim para el sistema operativo Windows [23].

Primero, se debe descargar Epic Games Launcher siendo este un lanzador de programas de videojuegos y en especial para nuestro caso de Unreal Engine. Una vez descargado, instalado y registrado se procede a descargar la versión de Unreal Engine 4.25.4 debiendo ser esta mayor que la 4.25 para el funcionamiento de AirSim y recomendada la 4.25. En la Figura 11 se observa la instalación de Unreal Engine desde Epic Games Launcher.



**Figura 11: Lanzador Epic Games Launcher con Unreal Engine**

Como segundo paso se procede a descargar el Visual Studio Code 2019 [24] con los complementos necesarios Desktop Development with C++ y Windows 10 SDK 10.0.18362. Una vez instalado se tiene acceso a la herramienta Developer Command Prompt for VS 2019 teniendo una interfaz similar a la ventana de comandos de Windows, como se puede ver en la Figura 12.



**Figura 12: Ventana Developer Command Prompt for VS 2019**

Desde esta aplicación se clona el directorio GitHub con el repositorio de AirSim en la ubicación seleccionada, recomendando no instalar en rutas con caracteres especiales como ñ o tildes y de no instalarlo en el disco C o D directamente. Para realizar este paso, se introduce desde el directorio elegido el comando `git clone https://github.com/Microsoft/AirSim.git`. Una vez clonado este repositorio, se

avanza al directorio AirSim con `cd AirSim` y se ejecuta el comando `build.cmd`. Esto crea la carpeta Plugins dentro de la carpeta Unreal de AirSim, la cuál será necesaria para crear los entornos propios, pero en este caso se utilizará el entorno por defecto Blocks.

Este entorno viene incluido en AirSim para realizar las pruebas necesarias. Para iniciararlo, se navega hasta la ruta `AirSim\Unreal\Environments\Blocks` y se accede al archivo `.sln`, que será el fichero donde se ha cargado el proyecto en *Visual Studio*. Una vez iniciado este archivo se ejecuta el *Visual Studio*, indicando modo de ejecución *Develop Editor* y *Win64*. Por último, se inicia el simulador, abriéndose el entorno que se puede observar en la Figura 13.

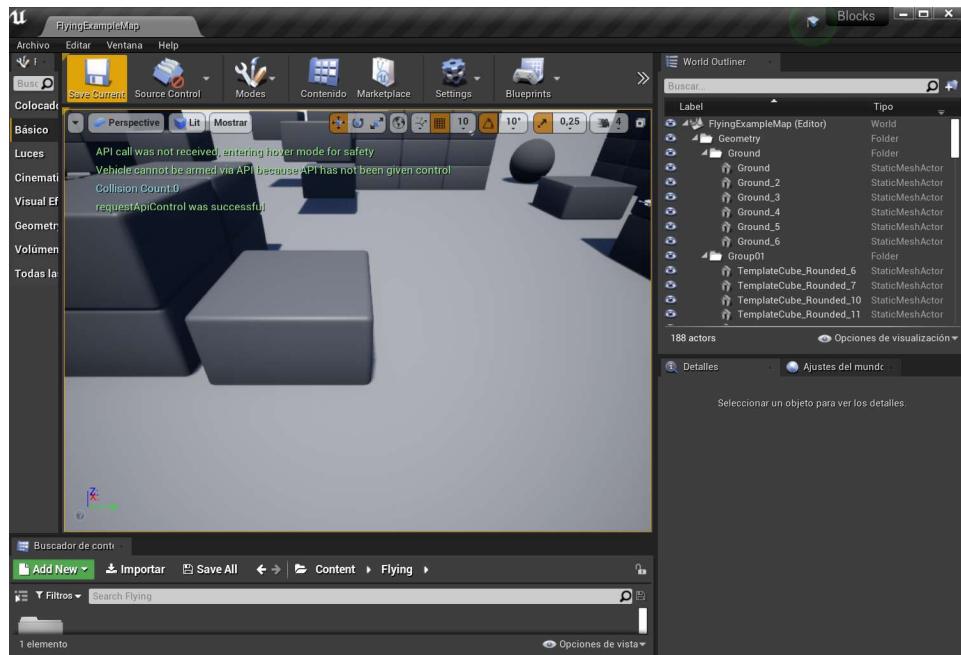


Figura 13: Ambiente Blocks en Unreal Engine

#### 4.2.2 Entorno de trabajo Anaconda

Para desarrollar este proyecto se ha utilizado Python y se ha elegido Anaconda [16] para crear un entorno de programación específico en el que descargar todas las librerías indicadas en los siguientes subapartados.

Una vez instalado este software, se genera el entorno mediante el comando `conda create -n entrenamientoKeras Python=3.7` desde la ventana de comandos de Windows, indicando la versión de Python la cual se ha podido comprobar que para este proyecto por motivos de aceptación de versiones es el más indicado, además de indicar para este caso `entrenamientoKeras` como nombre del entorno. Para activar este entorno se introduce el comando `conda activate entrenamientoKeras`, al igual que para desactivarlo `conda deactivate` como se puede ver en la Figura 14.

```
C:\Users\robertorg>conda activate entrenamientoKeras  
(entrenamientoKeras) C:\Users\robertorg>conda deactivate  
C:\Users\robertorg>
```

**Figura 14: Activación y desactivación de entorno Anaconda**

Este entorno se usa en todo el proyecto para instalar en él las librerías y desarrollar la solución; por ese motivo, de aquí en adelante todos los comandos utilizados se han realizado en este entorno previamente activado.

#### 4.2.3 Comunicación AirSim

Se instalan los paquetes necesarios para utilizar las APIs de AirSim [10] para Python y los paquetes necesarios para la comunicación con el simulador. Estos son el paquete *AirSim*, que contiene las APIs de AirSim con versión 1.3.0 y los paquetes *msgpack-python* y *msgpack-rpc-python*, que servirán para la comunicación con el simulador con versiones 0.5.4 y 0.4.1, respectivamente. Para ello, se utiliza en el entorno de Anaconda realizando el comando *pip install AirSim==1.3.0*, *pip install msgpack-python==0.5.4* y *pip install msgpack-rpc-python==0.4.1*.

#### 4.2.4 Librerías de Deep Learning y Deep Reinforcement Learning

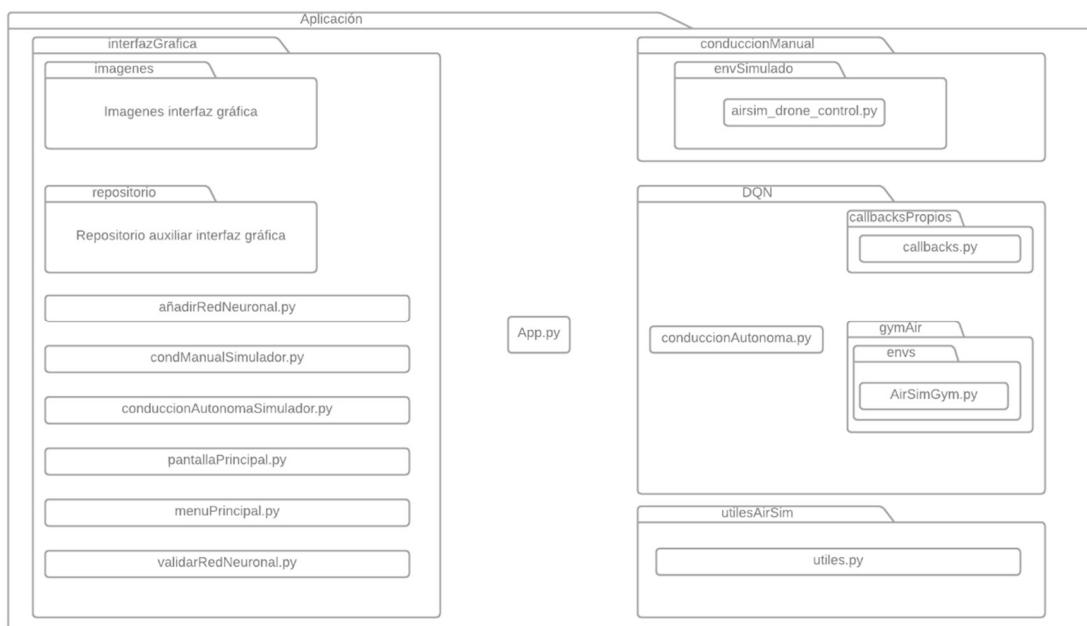
Los paquetes necesarios para realizar Deep Learning serán Tensorflow mediante el comando *pip install tensorflow==2.1.0* y Keras mediante el comando *pip install Keras==2.3.1*. Además, este proyecto ha sido realizado con GPU por lo que habrá que habilitar su uso para Tensorflow. Para ello, se descarga el instalador de CUDA versión 10.1, siguiendo el proceso del instalador y completándose, teniendo como requisito tener instalado Visual Studio. Una vez completado este proceso, se descargan las librerías cudnn desde la página oficial previo registro y descargando la versión 7.6. Del directorio descargado, se obtiene el fichero .dll de la carpeta bin y se copia en la carpeta de CUDA dentro del directorio bin, se repite el proceso con el fichero cudnn.h del directorio include copiándose en el directorio include de CUDA. Como último fichero, se vuelve a repetir el proceso descrito con cudnn.lib copiándose en el directorio x64 de CUDA. Manualmente, se deben editar las variables de entorno del sistema añadiendo al PATH la ruta que apunte al directorio bin y libvvp de CUDA. En cuanto a la librería de Deep Reinforcement Learning, instalaremos keras-rl utilizando *pip install keras-rl2==1.0.4*.

#### 4.2.5 Otras librerías

Otras librerías que requieren instalación son numpy con `pip install numpy==1.19.5` y la librería Gym que se utiliza para generar los ambientes de los agentes de Deep Reinforcement Learning con el comando `pip install gym==0.18.0`. Otra librería necesaria para el uso de la interfaz gráfica es PyQt5 mediante el comando `pip install PyQt5`.

### 4.3 Arquitectura aplicación

En este apartado y subapartados se explica la arquitectura de la aplicación y el desarrollo de cada uno de sus elementos. La arquitectura de módulos de la aplicación se puede ver en la Figura 15.



**Figura 15: Arquitectura de módulos de la aplicación**

La aplicación consta de cuatro grandes paquetes, el paquete *interfazGrafica*, el paquete *DQN*, *utilesAirSim* y *conduccionManual*, además del módulo principal *App.py* que maneja todo el funcionamiento de la aplicación. En los siguientes subapartados se explican estos cinco componentes.

#### 4.3.1 Paquete `utilesAirSim`

Este paquete contiene un único script llamado `utiles.py` para realizar operaciones que se utilizan en varios paquetes. Para ello, utiliza las librerías `airsim`, `numpy`, `time` y `PyQt5`. Los métodos que contiene para realizar estas operaciones son los de `grabar` en el que paralelamente se conecta, a pesar de las conexiones que pueda ya haber, con el simulador y según la opción elegida va capturando imágenes verticales o horizontales del entorno guardándolas en un elemento de una interfaz gráfica que se le indique, repitiendo la operación cada 0.1s dando la sensación al usuario de la grabación de un video. También contiene los métodos `tomarImagen` que devuelve una única imagen en RGB de la cámara indicada y que además se utiliza en el método `grabar`. Por último, contiene el método `guardarImagen` para guardar una imagen en la ruta indicada con formato png.

#### 4.3.2 Paquete `conduccionManual`

AirSim no tiene incorporado para drones la funcionalidad de conducción manual; sin embargo, para esta aplicación desarrollada para la agricultura con toma de imágenes se trata de una funcionalidad necesaria para sacar una toma con mayor exactitud o ajustarse después de un trayecto con conducción autónoma. Por ello este paquete contiene a su vez el módulo `envSimulado` con el script `airsim_drone_control.py` desarrollado para obtener dicha funcionalidad.

El objetivo de este script es la conducción del dron mediante controles del teclado, añadiendo además la posibilidad de realizar capturas de pantalla tanto con la elección de la cámara frontal o la cámara inferior para obtener imágenes de cualquier terreno. Para ello se utiliza la clase de Python `keyboard` que responde a los eventos realizados al pulsar cualquier tecla del teclado.

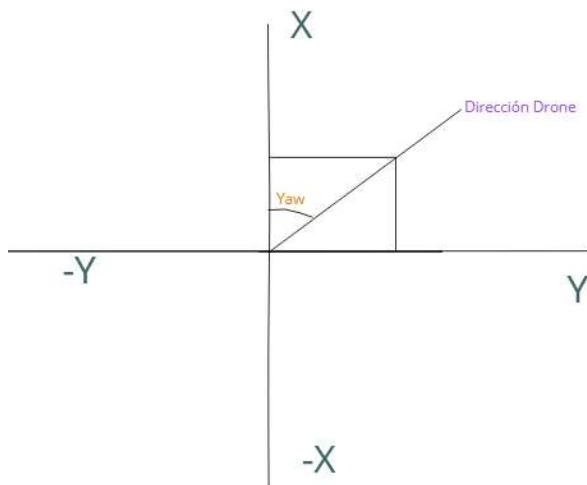
Este script desarrolla la clase `DroneController` en el que al inicializarla se define una velocidad máxima de seis m/s, una aceleración de 3 m/s<sup>2</sup>, una velocidad angular de 90 grados/s, una duración de acción de 0.4 s, un decremento de velocidad de 0.5 m/s y un vector de velocidad en las tres direcciones. Estas tres direcciones corresponden a una primera componente de avance hacia delante y detrás, la segunda de derecha e izquierda y la tercera arriba o abajo, tomando como referencia el dron. Adicionalmente, se definen los siguientes controles:

- Flecha hacia arriba para avanzar recto
- Flecha hacia abajo para avanzar hacia atrás
- Flecha hacia la izquierda para girar sobre sí mismo hacia la izquierda

- Flecha hacia la derecha para girar sobre sí mismo hacia la derecha
- Letra w para ascender
- Letra s para descender
- Letra a para moverse a la izquierda
- Letra d para moverse a la derecha
- Barra espaciadora para tomar imagen

Además, utilizando la API de AirSim se establece la comunicación con el simulador Unreal Engine, asegurándose que la comunicación establecida es correcta y realizando también un despegue. Esta clase contiene el método principal *fly\_by\_keyboard* utilizado para la funcionalidad de la clase. Esté método espera cada 0.05s a recibir un evento generado por la pulsación o liberación de pulsación de una de las teclas definidas en los controles para guardar cuál de ellas sigue pulsada o ha sido liberada. Por lo tanto, cuando una tecla es pulsada obtiene la orientación actual del dron con su ángulo yaw, explicado en el apartado 2.3.4, y la dirección de dron, para según la tecla que esté pulsada realizar una acción u otra. El cálculo de esta dirección se realiza calculando dos componentes de la siguiente manera:

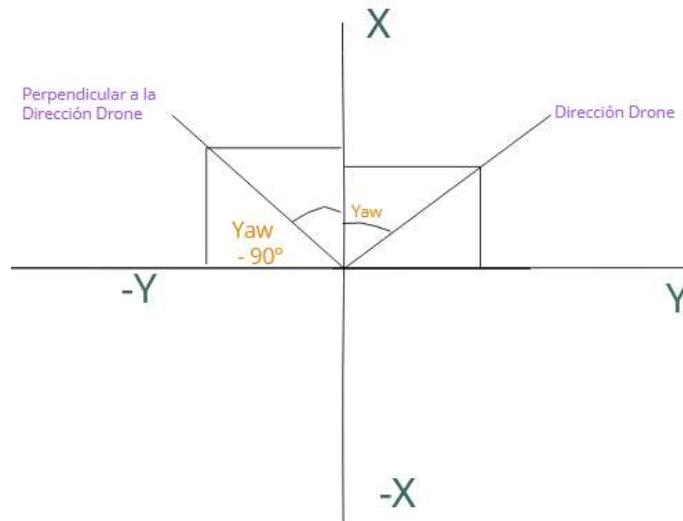
- Componente paralela a la dirección que está apuntando el dron, almacenada en la variable *forward\_direction*. La dirección definida para el sentido del dron se podrá calcular como el coseno de yaw para el eje X y el seno de yaw para el eje Y. Esta representación se puede ver en la figura Figura 16



**Figura 16: Esquema dirección dron**

- Componente perpendicular a la dirección para los movimientos de izquierda y derecha serán almacenados en la variable *left\_direction*. Está

dirección se podrá calcular como el coseno de yaw – 90° en el eje X y el seno de yaw - 90° para el eje Y. Esta representación se puede observar en la Figura 17



**Figura 17: Esquema dirección perpendicular dron**

Si las teclas pulsadas corresponden a avanzar o retroceder (la flecha superior o inferior del teclado), se utiliza la variable *forward\_direction*, la cual estará multiplicada por la duración de la acción y por la aceleración, que se sumará a la velocidad actual del dron. En el caso de no estar activadas dichas teclas, se multiplicará esta variable por la fricción definida y se sumará a la velocidad actual del dron si la flecha pulsada es la flecha de arriba, (esto es la flecha de avanzar). Si se trata de la flecha inferior empleada para retroceder, se restará. Se repite el mismo proceso para las teclas de moverse a la izquierda o derecha, pero en este caso utilizando la variable de dirección *left\_direction*, sumándose si se quiere ir a la izquierda del dron o restándose si es a la derecha del dron. Si la acción es de subir, se aplicará una resta en el eje Z ya que como se comentaba en el apartado 2.3.4, AirSim trabaja con coordenadas NED (North, East, Down). La resta empleada corresponderá a la duración por la aceleración definida; en cambio, si se quiere descender se realizará una suma. Estás acciones de suma y resta de velocidades se controlarán para que como máximo puedan llegar en módulo a la velocidad máxima definida para el dron. Para las últimas acciones de girar sobre sí mismo, si se quiere realizar un giro hacia la izquierda se aplica una resta del yaw y una suma para la acción opuesta, con la velocidad definida de giro. Por último, si se activa la barra espaciadora se realizará una captura en la ruta indicada por el usuario con el nombre concatenado del día, mes, año junto a la hora, minuto y segundo en formato png haciendo uso del script explicado en el apartado 4.3.1.

Por último, este script también recibe elementos de la interfaz gráfica definida para este modo, el cual se explica en el apartado 4.3.4, para saber el tipo de imagen a capturar según la elección del usuario.

#### 4.3.3 Paquete DQN

Este paquete es el principal de esta aplicación, debido a que contiene la funcionalidad para realizar una conducción autónoma. La funcionalidad de este paquete la controla el script *conduccionAutonoma.py* utilizando los scripts alojados en los módulos *gymAir* y *callbacksPropios*. Por ello, para entender esta funcionalidad primero es necesario explicar en profundidad el funcionamiento de un algoritmo DQN con la biblioteca utilizada *keras-rl*, basándonos en la explicación teórica de este algoritmo dada en el apartado 2.5.1.1. Este algoritmo se inicializa con el valor de la memoria, **M**, que será el tamaño de Experience Replay que contendrá la experiencia u observaciones del agente. Cada observación o experiencia contendrá el estado **S**, la acción **A**, la recompensa **R** y el siguiente estado obtenido **S'**. Se inicializará a su vez la red neuronal convolucional que se defina, realizando una copia de ella con el fin de tener las dos redes neuronales necesarias, y junto con el número de acciones que se llevará a cabo para terminar el entrenamiento. En primer lugar, se dedicarán unas acciones a realizar observaciones e ir completando el elemento Experience Replay. Después de este periodo comenzará el entrenamiento de las redes, definiendo dos fases en este bucle: la fase exploración y fase explotación. Aunque se haya dedicado un periodo previo a ultimar parte de Experience Replay, no tiene por qué haberse completado, o de haberlo hecho, se deberán ir cambiando estas observaciones obsoletas por otras. A la vez se ha de entrenar esta red con el objetivo de buscar el mayor valor posible de recompensa en cada acción. La primera fase se llamará exploración y la segunda explotación. Así pues, también se definirá  $\epsilon$ , el valor que determinará si debemos explorar o explotar y que irá decreciendo según avancemos, ya que será menos necesario explorar las acciones y estados posibles del dron. Una de las políticas disponible y utilizada para este proyecto es la que sigue una forma lineal, comenzando en el valor más alto y disminuyendo linealmente hasta el valor mínimo.

En definitiva, en el modo explotación se utiliza la red neuronal teniendo como entrada el estado **S** y como salida contiene tres posibilidades definidas por las acciones de nuestro proyecto: continuar en la dirección del dron, girar 15° a la derecha o girar 15° a la izquierda, eligiéndose la acción que posea mayor valor **Q** para el estado **S**.

Este proceso dará como resultado la acción futura  $S'$  y una recompensa  $R$ , guardándose como una memoria  $M$  en el Experience Replay. Después, se seleccionará aleatoriamente una observación, ejecutando el mismo proceso y ajustando valores con la ecuación de Bellman definida en la Ecuación 6. Esto se realiza hasta que se llega al objetivo o bien obtenemos una recompensa muy pequeña, en nuestro caso -100, por una colisión o estar alejándonos durante demasiados pasos.

Una vez terminado, se debe reiniciar al estado inicial y repetir todo el proceso desde la elección de modo exploración o explotación, habiendo completado así un episodio y terminando definitivamente cuando se haya llegado al número de pasos límite a realizar.

Con esta explicación del modelo de DQN utilizado por keras-rl se crearon los elementos claves para llevarlos a cabo. Primero, el entorno que obtendrá los estados según se definan estos, las acciones que se haya definido que puede realizar el agente y la función recompensa que dictamine el objetivo. Se realizó en el script *AirSimGym.py* contenido en el módulo *envs* dentro a su vez del paquete *gymAir*. Para ello se definió dentro de dicho script la clase *AirSimEnv* que se inicializará con la comunicación hacia el simulador, la duración de movimientos de un segundo, la velocidad de 15° por segundo de giro y velocidad de dirección de 4 m/s, además de obtener la actual dirección del dron y posición. El primer gran método de esta clase es el método que te calcula la recompensa obtenida por acción, la función de recompensa, que se denomina **computeReward**. La recompensa se calculará según la Ecuación 7, siendo  $D$  la distancia actual hacia el objetivo y  $D'$  la distancia anterior que estábamos al objetivo y  $R$  la recompensa por acción.

$$R = D - D' - 1$$

### Ecuación 7: Función de recompensa

Esta recompensa está pensada para que la red neuronal aprenda que realizar una acción debe acercarle al objetivo, mientras que la resta de la constante 1 se realiza ya que si el dron girará sobre sí mismo obtendría recompensa 0 y está sería un valor mayor en comparación con un valor negativo. Para calcular estas distancias, debido a que no se tiene en cuenta para el dron el eje Z para la simplificación en el número de acciones, se utilizaría el cálculo de módulos de vectores, siendo  $G_x$  la coordenada X del objetivo,  $G_y$  la coordenada Y del objetivo,  $P_x$  la coordenada X actual del dron,  $P_y$  la coordenada Y actual del dron,  $P'_x$  la

coordenada X antes de realizar la actual acción del dron y  $P'$ , la coordenada Y antes de realizar la acción actual del dron, se calcula  $D$  y  $D'$  según las ecuaciones Ecuación 8 y Ecuación 9 respectivamente.

$$D = \sqrt{(G_x - P_x)^2 + (G_y - P_y)^2}$$

#### Ecuación 8: Calculo distancia actual

$$D' = \sqrt{(G_x - P'_x)^2 + (G_y - P'_y)^2}$$

#### Ecuación 9: Calculo de la distancia anterior

Como segundo método principal se declara el método **getScreenSceneVis**, que devolverá lo que se ha definido por estado. Para este proyecto se ha tomado como estado lo que observa el dron por su cámara frontal con imágenes de profundidad de un tamaño estático de 30x100, además de estar compuesto de una línea negra como información extra que le indicará en qué sentido se encuentra el objetivo. Como dos últimos métodos principales, contendrá el método **reset** que se encargará de reiniciar el dron al estado inicial cuando termine un episodio y el método **step**. Este último método recibe como parámetro la acción elegida por la red neuronal, conteniendo un valor de 0,1 o 2. Si recibe un 0, deberá continuar recto en la dirección del dron actual con la velocidad de 4 m/s durante 1 segundo de duración, mientras que si se recibe un 1 tendrá que girar a la derecha y con un 2 deberá girar a la izquierda. Una vez realizada la acción, obtendrá información del simulador para comprobar si ha habido una colisión o no, en caso de que sí la recompensa de ese paso será de -100 y habrá finalizado el episodio. En caso de que no, utilizando la función de recompensa definida, guardará el valor de recompensa de esa acción y se sumará a la recompensa en ese episodio, terminando este si la suma total llega a un valor menor o igual a -100. Si no sucede, guardará el nuevo estado y el episodio continuará a no ser que el dron se encuentre a una distancia de 2 metros, que se considerará que habrá llegado al objetivo recibiendo una recompensa de +100 parando el episodio cuando la recompensa sea igual o mayor a 100.

Una vez explicado este script, se puede desarrollar el script principal *conduccionAutonomia*. Este script contiene la clase principal llamada *conduccionAutonomia*, que a su vez será la clase que suministre a la aplicación principal todas las funcionalidades. Se inicia definiendo como constantes las

dimensiones de 30x100 de las imágenes de profundidad que serán nuestros estados y el número de acciones que se pueden realizar, en este caso tres. También, inicializa la primera conexión con el servidor creado una instancia de la clase definida anteriormente *AirSimGym.py*. Como métodos principales, el primero que se debe utilizar para realizar cualquier operación es la inicialización del modelo de red neuronal utilizado y definido para esta aplicación, mediante el método **inicializarModelo**. Este modelo contiene las siguientes capas en este orden:

- Capa convolucional de dos dimensiones con 16 filtros de kernels con dimensiones de 4x4 y una función de activación de tipo ReLU. Tiene como entrada imágenes de 30x100.
- Capa de normalización de resultados para normalizar los valores de salida.
- Capa convolucional de dos dimensiones con 32 filtros de kernels con dimensiones de 3x3 y una función de activación de tipo ReLU.
- Capa de normalización de resultados para normalizar los valores de salida.
- Capa convolucional de dos dimensiones con 32 filtros de kernels con dimensiones de 1x1 y una función de activación de tipo ReLU.
- Capa de normalización de resultados para normalizar los valores de salida.
- Una capa de Max-Pooling de 2x2 dimensiones
- Una capa Flatten que permite la conexión de las capas convolucionales con las capas de la red neuronal tradicional.
- Una capa Dense de 128 neuronas con función de activación ReLU.
- La capa de salida Dense con 3 neuronas correspondientes a las 3 acciones posibles, con función de activación Softmax.

El siguiente método que se debe instaurar para utilizar este proyecto, es el método **inicializarAgenteDQN**. Utiliza algunos valores por defecto si no se ha indicado ninguno por el usuario, según con el prototipo que se realizó en este proyecto con valores aceptables. Estos parámetros si los dividiésemos en categorías serían los siguientes:

- Determinar valor  $\epsilon$ : como se explicó anteriormente, se ha utilizado un algoritmo para el valor de  $\epsilon$  lineal. Este valor determinará si se realiza exploración o explotación disminuyendo la probabilidad de exploración con el aumento de los pasos o acciones en el entrenamiento. Por este motivo, se definirá el valor mínimo que llega tener con un valor por defecto de 0.1 como un parámetro, el valor máximo con un valor por

defecto de 1 como otro parámetro y el rango de esta función en otro parámetro, en otras palabras, se definirán cuantos pasos se tardará en llegar al valor mínimo con un valor por defecto de 50000. La evolución que tendrá este valor  $\epsilon$  según estos valores por defecto se puede ver en la Figura 18



**Figura 18: Variación de  $\epsilon$  respecto a pasos realizados**

- Experience Replay: se debe indicar el tamaño de memoria, el cual debe ser un número considerable, ya que hay muchas combinaciones posibles de estados; sin embargo, no conviene excederse ya que esto reduciría el rendimiento del entrenamiento. Por defecto se toma un valor de 7500. También se indicará el periodo previo al entrenamiento en sí para recopilar observaciones, utilizando por defecto un valor de 1000.
- Learning rate: este valor es de los más importantes ya que tiene bastante peso en la ecuación de Bellman. Por defecto, para este modelo de red neuronal, se tomó un valor de 0.001.

Una vez obtenido estos parámetros, bien porque los introduzca el usuario o los utilizados por defecto, se crea una instancia de la librería DQNAgent con estos datos y el modelo de red neuronal inicializado previamente.

Después de utilizar estos dos métodos, aparecerán los métodos que utilice el usuario. El método **entrenarNuevoModelo**, que se encargará de entrenar al agente para realizar la conducción autónoma. Esté método recibirá del usuario el punto objetivo donde se realizará el entrenamiento respecto al cual se calcularán

las recompensas, la ruta donde se guardarán los pesos de la red una vez se termine de entrenar, los pasos que durará el entrenamiento y los booleanos check, wandb y History. Los booleanos check y wandb, indicarán si el usuario quiere utilizar los callbacks ModelIntervalCheckpoint implementado por Keras y WandbLogger implementado como una clase del script contenido en el módulo *callbacks.py* del paquete *callbacksPropios*. Antes de continuar con este método se define que son los objetos Callbacks, siendo estos objetos utilizados durante un entrenamiento de una red neuronal para realizar acciones en medio de estos procesos como guardados de pesos, creamiento de gráficas, entre otras múltiples opciones, proporcionando Keras una gran variedad. En este proyecto se pensó en utilizar la plataforma wandb para el monitoreo del entrenamiento con múltiples gráficas desde un navegador web ya que son entrenamientos con una gran cantidad de tiempo de procesamiento. Con esta plataforma se pueden seguir los datos registrados desde cualquier lugar si se tiene acceso a la red con un usuario y contraseña obteniendo una cuenta gratuita en esta página web. Para ello, se necesita ir actualizando valores en este proceso de entrenamiento y Keras permite la posibilidad de crear callbacks; por ello, en este script se realiza este proceso, definiendo la clase *WandbLogger*.

Está clase sigue los métodos obligados de los callbacks de Keras, métodos que serán invocados en su respectivo momento. Se utiliza el método **on\_train\_begin**, cuando comienza el entrenamiento y en él se inicia el contador de tiempo que durará el entrenamiento. En el método **on\_episode\_begin** se llevará a cabo en el comienzo de cada episodio, inicializándose la recompensa total del mismo, los valores de las observaciones, las acciones realizadas, entre otras estadísticas, mientras que **on\_episode\_end** se activará al finalizar los episodios, enviando la información recaudada durante dicho periodo. Esto implica que después de cada mencionado intervalo se refrescarán los datos registrados en la página web Wandb. En **on\_step\_end** se obtendrán datos como la recompensa por paso o la acción realizada. Por otro lado, el callback ModelIntervalCheckpoint tendrá la función implementada por Keras de guardar cada un número de pasos indicados los pesos actuales de la red como una copia de seguridad.

Por ende, los booleanos check y wandb, permitirán elegir al usuario si quiere utilizar estos callbacks. Utilizando al agente instanciado como DQNAgent, con el método **fit** se realizará el entrenamiento mediante el procedimiento previamente explicado. Una vez terminado, este método devolverá un historial que contendrá el valor de recompensa por cada episodio realizado, llevando a cabo una copia de este historial en un archivo .csv según el valor del booleano History. Por último, guardará los pesos del entrenamiento en la ruta previamente indicada como un archivo h5f.

El siguiente método que da funcionalidad a la aplicación con otra opción es el método **validarEntrenamiento**. Esté método realiza la validación de resultados de una red neuronal entrenada. Para ello, el usuario debe indicar el punto objetivo respecto al cual se debe iniciar la validación, la ruta donde se encuentran los pesos entrenados de la red, la ruta donde se guardará el fichero csv con los resultados de la validación y el número de episodios que se desean realizar para hacer una validación. Los resultados que contendrá el fichero solución mostrarán la recompensa en cada uno de los episodios, por ese motivo, si en el episodio se ha conseguido más de 100 puntos significará que se ha conseguido llegar al punto objetivo.

El último método es el método que da la funcionalidad principal a la aplicación, el método que te permite realizar vuelos autónomos realizando imágenes RGB de la cámara inferior del dron simulando captura de imágenes de cultivo, se llama **conduccion**. Este método recibe una ruta con los pesos de la red neuronal que se utiliza para la conducción autónoma y son cargados en el modelo, una lista de coordenadas donde que se deben alcanzar y realizar las capturas, una ruta donde se guardarán estas capturas y, por último, un elemento de la interfaz gráfica que irá mostrando estas capturas. En primer lugar, se cargarán los pesos, en segundo lugar, se cargará el primer punto objetivo en el ambiente y se realizará un bucle que no terminará hasta que se llegue a dicho punto objetivo. Dentro de este bucle, se reciben un array de tres valores con los valores Q de cada acción para cada estado, siendo la posición 0 el valor Q para la acción de avanzar en la dirección del dron para dicho estado, la posición 1 el valor Q para la acción de girar a la derecha y la posición 2 el valor Q para la acción de girar a la izquierda. La posición con mayor valor Q será la acción predicha por la red neuronal, invocando después el método **step** con esta posición en el entorno. Una vez localizado el destino, el dron esperará tres segundos para estabilizarse y poder tomar una buena captura de su cámara inferior en RGB, guardándola en la ruta indicada. Este proceso se repetirá tantas veces como puntos tenga el array de coordenadas.

#### 4.3.4 Paquete interfazGrafica

Este paquete contiene los scripts que sirven la interfaz gráfica al script principal de la aplicación para mostrarla al usuario. Estas interfaces se han diseñado con el editor Qt Designer [25] utilizando su software para crear los elementos de una manera más sencilla, como se puede ver en la Figura 19.

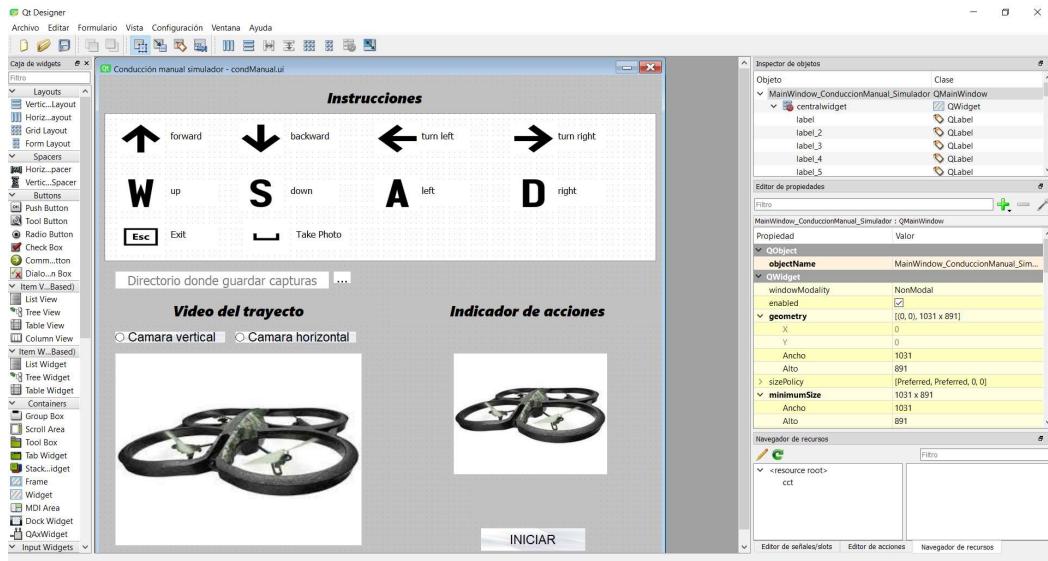


Figura 19: Pantalla principal Qt Designer

Este software crea archivos .ui que después se pueden convertir a scripts de Python mediante el comando `pyuic5 -x nombreArchivo.ui -o nombreArchivo.py`, siendo nombreArchivo.ui el fichero creado por Qt Designer y nombreArchivo.py el nuevo script que se generará de ese archivo. En estos archivos se han utilizado imágenes e iconos para realizar la aplicación, los cuales son guardados como recursos y están alojados en el directorio imágenes de este mismo módulo. La ventaja de esta manera de realización es la posibilidad de modificar cualquier ventana abriendo su archivo .ui desde el software Qt Designer, modificar el elemento correspondiente, ejecutar el comando indicado, sobrescribiendo el script antiguo por el nuevo modificado y no influyendo en el resto de los scripts que utilizan la respectiva ventana.

Los scripts creados con este proceso son: *pantallaPrincipal.py*, siendo este la pantalla inicial de la aplicación. Esta ventana dará la posibilidad de acceder al menú principal, en definitiva, a la interfaz generada por el script *menuPrincipal.py*. En esta interfaz se ofrecerán varias opciones al usuario correspondiente a los modos permitidos en la aplicación, la posibilidad de entrenar un agente DQN correspondiente al script *añadirRedNeuronal.py*, la posibilidad de validar el entrenamiento del agente correspondiente al script *validarRedNeuronal.py*, el modo de conducción manual dando la funcionalidad la interfaz gráfica *condManualSimulador.py* y el modo para la conducción autónoma con el script *conduccionAutonomaSimulador.py*.

#### 4.3.5 Paquete principal

El paquete principal es el contenedor de todos los anteriores paquetes más el script encargado de unir todas las funcionalidades, el script de Python *app.py*. Este script se encarga en primera instancia de inicializar todas las ventanas y asociar a cada botón de cada una de ellas las acciones que deben realizar, dicho de otra forma, asignar los eventos correspondientes a cada una de las interfaces gráficas del apartado 4.3.4. Esto implica también la inicialización de los objetos correspondientes de los paquetes *DQN* y *conduccionManual*, explicados en los anteriores apartados. Como parte fundamental de este script entra la definición de hilos para poder ejecutar las acciones en segundo plano, utilizando los objetos *QThread* correspondiente a los hilos para las interfaces gráficas definidas con PyQt5. Para ello, se han definido cinco clases que realizan las operaciones que corresponden a los modos permitidos de esta aplicación. Con este proceso el usuario podrá seguir utilizando la interfaz gráfica sin que se bloquee mientras que se realiza las operaciones elegidas en el simulador Unreal Engine.

La primera clase creada es *hilo\_entrenamiento*, siendo ésta la clase que se inicializa cuando el usuario inicie el entrenamiento en la interfaz gráfica asociada al script *añadirRedNeuronal.py*. Esta clase se crea con los parámetros necesarios para crear una red neuronal y el agente DQN para después entrenarlo, ejecutando en su método principal *run* heredado de la clase *QThread* el siguiente proceso. Instancia un objeto *conduccionAutonoma*, realiza la inicialización de un modelo con el método *inicializarModelo*, la inicialización del agente DQN con el método *inicializarAgenteDQN* para así utilizar el método *entrenarNuevoModelo* como se explica en el apartado 4.3.3. Cuando este hilo termina es destruido para no ocupar memoria.

La segunda clase creada será la correspondiente a la operación de la validación de la red *hilo\_valRed*, inicializada cuando comience la operación de validación del entrenamiento en la interfaz proveída por el script *validarRedNeuronal.py*. Realizará el mismo proceso que el anterior hilo utilizando los datos necesarios dados por el usuario, como la ruta de pesos y la ruta donde escribir los resultados de la validación, ejecutando la operación *validarEntrenamiento* de la clase *conduccionAutonoma*.

El tercer hilo creado se asocia a la operación de la conducción manual que está proporcionada por la interfaz gráfica creada en *condManualSimulador.py*. Esta clase se inicia con el parámetro correspondiente al objeto de la interfaz gráfica que maneja el usuario para elegir qué cámara ver en cada momento, el objeto donde se ven las imágenes y la ruta donde guardar las capturas manuales. En el

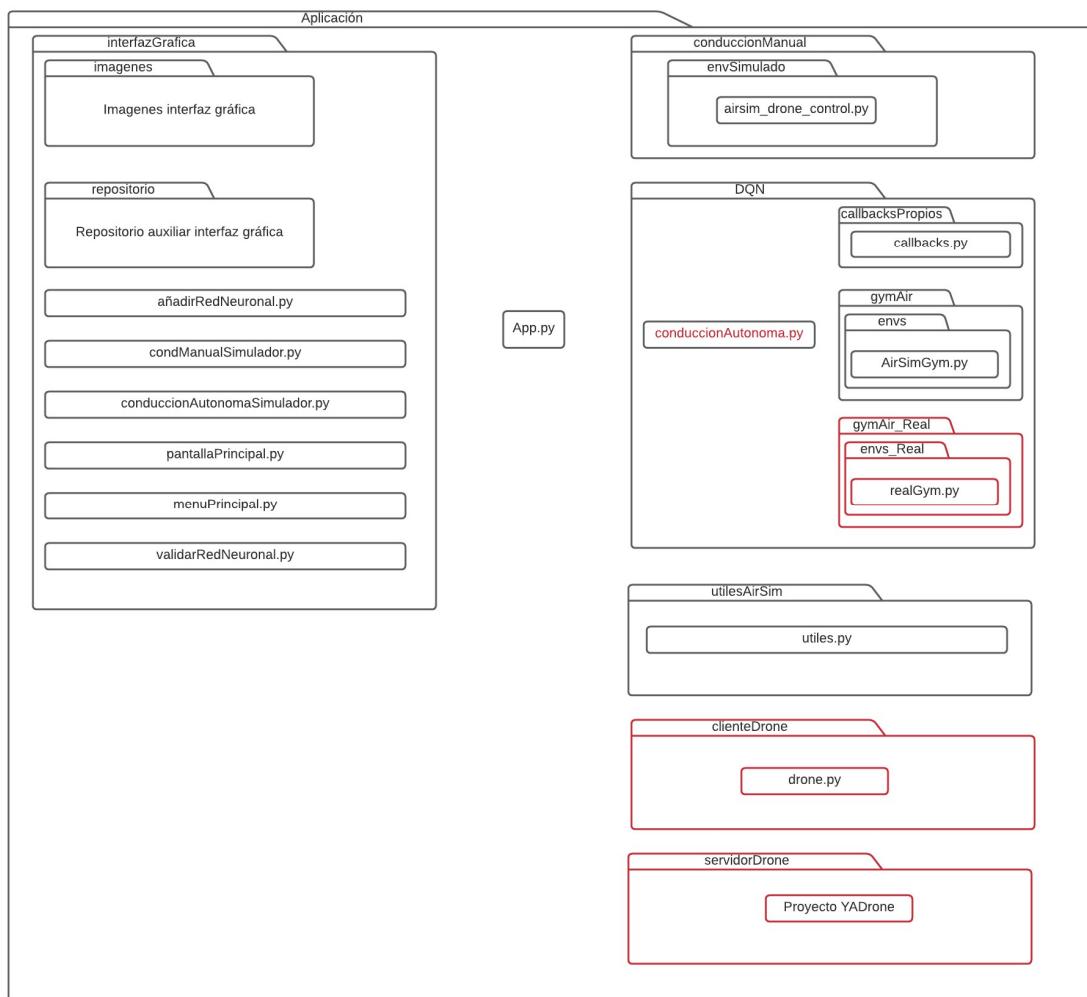
método *run* se creará una instancia para este hilo de la clase *airsim\_drone\_control.py*, que dará las funcionalidades de conducción manual.

El cuarto hilo correspondiente a la conducción autónoma se iniciará a la vez que se cree el quinto hilo creado, correspondiente a la acción de grabar, siendo estos las clases *hilo\_condAutoSim* y *hilo\_grabar* respectivamente. En el hilo creado para la conducción autónoma se utilizarán los parámetros suministrados por el usuario necesarios para utilizar el método *conduccion*. A la vez el hilo grabar utilizará como parámetros los objetos de la interfaz gráfica correspondientes a la muestra de la vista en tiempo real de lo visto por el dron y el objeto que permita cambiar de cámara.

#### 4.4 Solución en dron real

Por las limitaciones indicadas en el apartado 0, no se ha podido realizar una aplicación que se implemente en un dron real, utilizando en todo momento el simulador Unreal Engine con AirSim. En este apartado se propone una posible solución para un dron real explicando el posible nuevo modo para la aplicación conducción autónoma en un entorno real.

Para ello se propone utilizar la librería de Java YADrone [26] que es capaz de interactuar por una API propia con el dron y ejecutando de una manera sencilla las tres acciones que están definidas para nuestro agente DQN y en nuestro entorno, giro sobre sí mismo en ambos sentidos de 15° y seguir recto en la dirección del dron. Para ello se definiría la siguiente nueva arquitectura en la Figura 20.



**Figura 20:Arquitectura de módulos de la aplicación para implementación en entorno real**

Para explicar estas modificaciones primero es necesario explicar los módulos añadidos *clienteDrone* y *servidorDrone*. La idea desarrollada consiste en crear un programa escrito en Java que utilice la API YADrone que se implementa en el módulo *servidorDrone* y un módulo de Python que sea capaz de enviarle las acciones que debe realizar el dron y de recibir las imágenes RGB y de profundidad necesaria para los estados, codificada en el módulo *clienteDrone*. Para realizar este proceso se ha elegido la comunicación por sockets, de tal forma que en el proyecto *servidorDrone* se reciban las peticiones enviadas por el script codificado en el módulo *clienteDrone*, este proyecto mediante YADrone se comunique a su vez con el dron para satisfacer dichas peticiones. Para ello habría que definir las posibles peticiones a realizar por el *clienteDrone*.

- La inicialización para verificar la comunicación entre cliente y servidor.
- Realizar la acción *TakeOff*, para despegar el dron
- La acción avanzar en la dirección del dron, *seguirRecto*
- La acción de girar 15° a la derecha, *girarDerecha*
- La acción de girar 15° a la derecha, *girarIzquierda*
- La petición para pedir coordenadas actuales, *pedirPosicion*
- La petición para pedir el ángulo Yaw actual, *pedirOrientacion*
- La petición de obtener imagen en RGB o de profundidad, *pedirImagen*
- La petición de finalizar la conexión para terminar la comunicación con el servidor, *finalizar*.

Estas acciones se envían como cadenas de texto por la clase creada en el script del módulo *clienteDrone* y son recibidas por sockets por el proyecto en Java codificada con la API YADrone, realizando la acción correspondiente.

El nuevo modo podría utilizar la misma interfaz gráfica que provee *conduccionAutonomaSimulador.py*, ejecutando la aplicación principal el mismo proceso, pero habría que añadir un nuevo módulo y una modificación para el paquete *DQN*. La clase *conduccionAutonoma* se inicializa con el entorno que suministra la clase *AirSimEnv*, la cual se comunica con el simulador y realiza las acciones con la API de *AirSim*. Si se quiere añadir esta funcionalidad será necesario añadir un nuevo ambiente desarrollado en el script *realGym.py* que repita el proceso, pero cambiando la inicialización de comunicación y acciones desarrolladas con *AirSim* por un objeto del módulo *clienteDrone*. De esta forma se realizaría el mismo proceso que la API de *AirSim*, pero utilizando el nuevo objeto desarrollado que enviaría las peticiones al servidor dron y este ejecutaría las acciones correspondientes.

Por último, para no perder la funcionalidad del simulador y tener la posibilidad de la utilización de ambas, se realiza una modificación en la clase *conduccionAutonoma* para que al instanciarse se elija un ambiente simulado o un ambiente real, dicho de otra forma, el agente *DQN* interactue con el ambiente descrito en el script *AirSimGym.py* o *realGym.py*.

# Capítulo 5

---

## 5 RESULTADOS



## 5.1 Introducción

En este apartado se indicará el resultado del desarrollo de la aplicación, viendo la interfaz gráfica y el resultado del modelo entrenado que se ha realizado para poder utilizar la aplicación. Para ello, se va describiendo cada modo que utiliza la red neuronal y agente DQN; es decir, entrenamiento, validación y conducción autónoma con los resultados obtenidos, además de la descripción de la interfaz. En el último modo se desarrolla el resultado de la conducción manual obtenida.

## 5.2 Acceso y menú principal

Al ejecutar la aplicación resultante, se inicia la interfaz gráfica correspondiente a la Figura 21. Como se puede observar, se trata de una interfaz bastante sencilla, con la posibilidad de avanzar al menú principal mediante la tecla *iniciar*, o salir de la aplicación, mediante la tecla *salir*.



**Figura 21:** Pantalla inicial de la aplicación

En el menú principal, el usuario elegirá entre las opciones permitidas en la aplicación como son: el entrenamiento de una nueva red neuronal, validación de red neuronal, conducción manual y conducción autónoma en el simulador. La apariencia se observa en la Figura 22



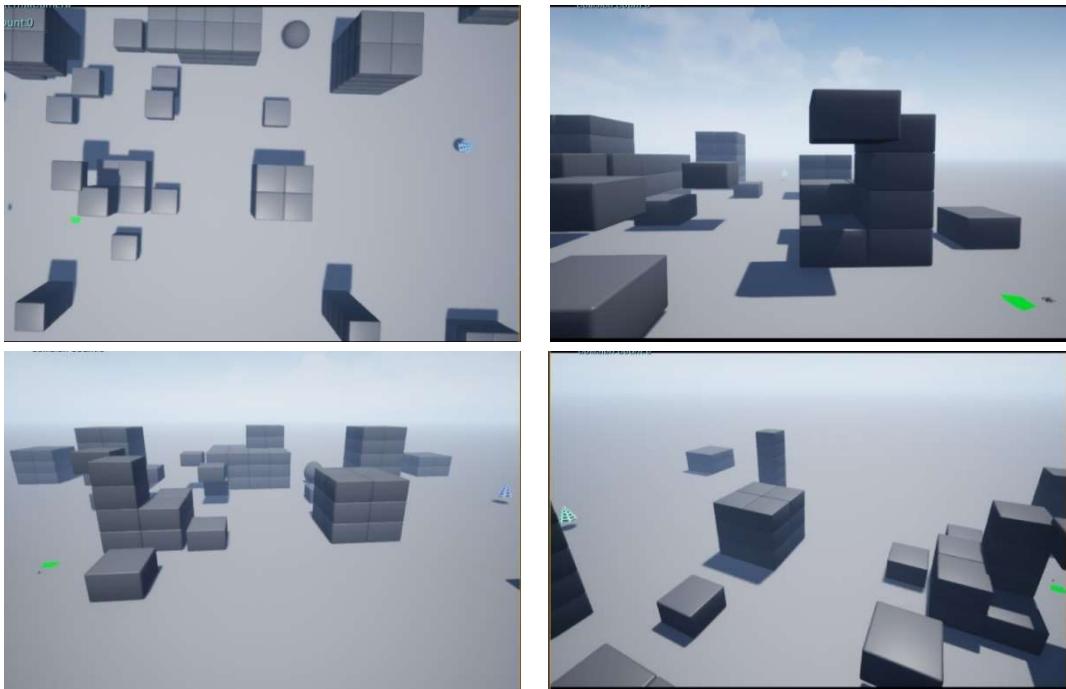
Figura 22: Menú principal de la aplicación

### 5.3 Modo entrenamiento

En este apartado se analizan los resultados del entrenamiento de la red neuronal predeterminada generada, estudiando los parámetros y gráficas. También se explica la interfaz gráfica de este modo, analizando las posibilidades que dispone el usuario para personalizar su modelo de red neuronal.

#### 5.3.1 Entrenamiento de red neuronal predeterminada

Como se explicó en el apartado 4.3.3, se entrenó una red neuronal y un agente DQN, guardando sus pesos para la utilización inmediata en la aplicación, aunque esta permita entrenar otras redes neuronales personalizando sus parámetros. Para realizar este proceso de entrenamiento se utilizó el entorno *Blocks* de Unreal Engine con AirSim, llevando a cabo ciertas modificaciones para poder producir ciertos obstáculos y así no realizar un entrenamiento simple para la red. Para la visualización de este entorno de entrenamiento de una manera más intuitiva, se le activó un sensor de color verde al dron y en el punto objetivo se insertó un objeto de forma de cono como referencia. Esto se puede ver definido en la Figura 23.



**Figura 23: Distribución del campo de entrenamiento**

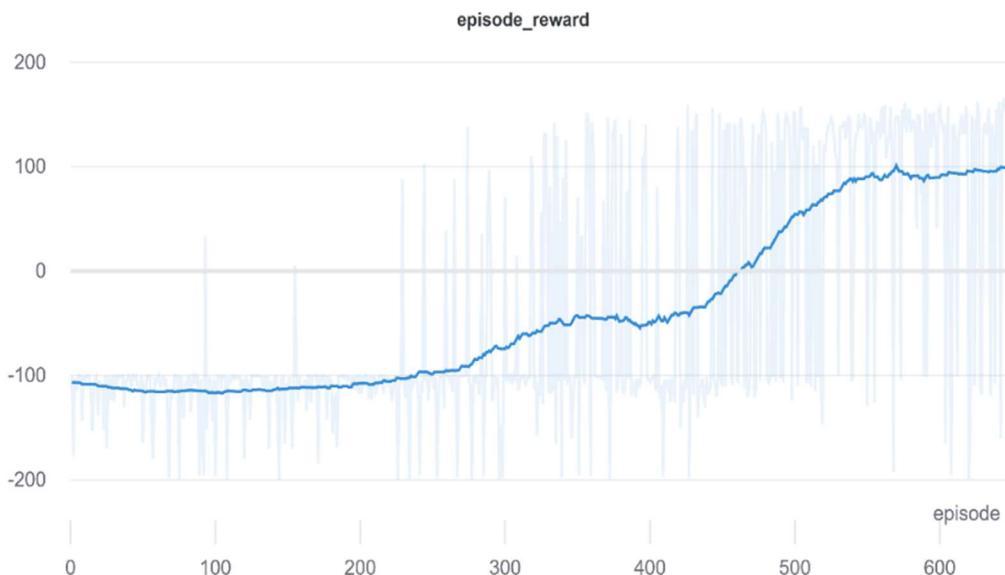
Por otra parte, esta red neuronal se entrenó con los parámetros que coinciden con los parámetros por defecto. En este entrenamiento se utilizó Wandb para registrar los resultados, buscando como objetivo una recompensa por episodio que convergiese durante un tiempo considerable en la máxima posible y una mayor recompensa por acción, para que la red buscase no sólo cumplir un objetivo, si no la efectividad en sus acciones. Para ello, se debe analizar cuáles son estas recompensas mínimas y máximas por episodio y por acción.

Para esto hay que analizar lo explicado en el apartado 4.3.3, viendo que la recompensa por acción viene definida por la Ecuación 7 y que se puede calcular  $D$  y  $D'$  con la Ecuación 8 y Ecuación 9 respectivamente, hay que encontrar el valor máximo que puede variar entre  $P$  y  $P'$ . Si el dron realiza desplazamientos 4 m/s durante 1 segundo, lo máximo que se puede ir acercando o alejando respecto a cada paso realizado es de  $\pm 4$  m. Esto implica que  $D - D'$  puede ser de 4 metros en el mejor de los casos y -4 metros en el peor de los casos, menos la resta de la constante 1, encontramos que la Ecuación 7 tiene como valor máximo 3 y valor mínimo -5.

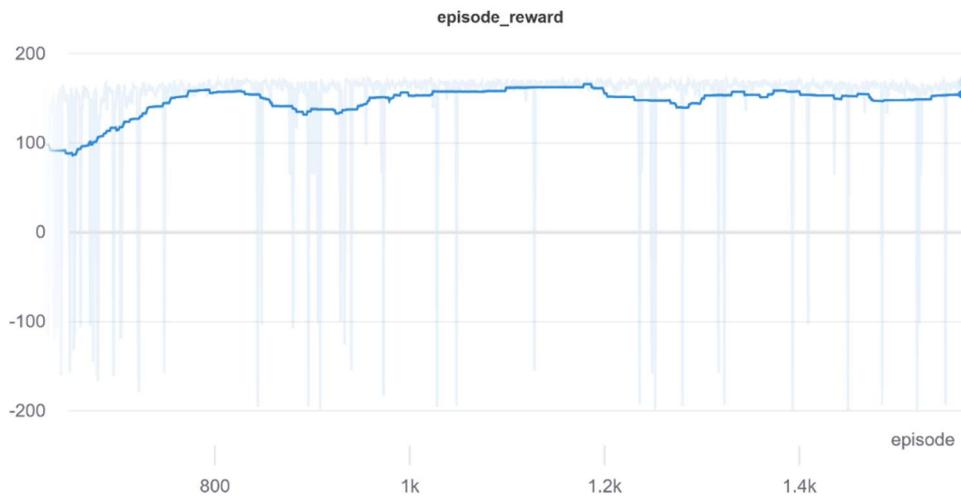
Ahora para calcular el máximo y mínimo valor posible de recompensa por episodio se deben tener en cuenta varios criterios. Para el valor máximo se puede calcular un valor aproximado, teniendo en consideración que si se llega al objetivo se obtiene una recompensa de +100 y se deja de contar recompensa para este episodio, por consiguiente, si por acción el dron se puede acercar 4 metros en el mejor de los casos, equivalente a +3 de recompensa como máximo, el número de pasos obteniendo las máximas recompensas que puede obtener el

dron es de la distancia inicial en metros entre el avance máximo por paso. Si este número de pasos se multiplica por el valor obtenido de recompensa en cada paso, se obtendrá el número de recompensa por episodio sumándole 100 de llegar al objetivo. En el caso del valor mínimo, se tendrá que tener en cuenta que a partir de un valor de -100 se deja de contar ya sea por acumulación de recompensas negativas tanto como por obtener -100 directamente por una colisión, implicando que en el peor de los casos el valor mínimo sería la acumulación de un valor muy cercano a -100 por alejarse demasiado y una colisión resultando aproximadamente -200.

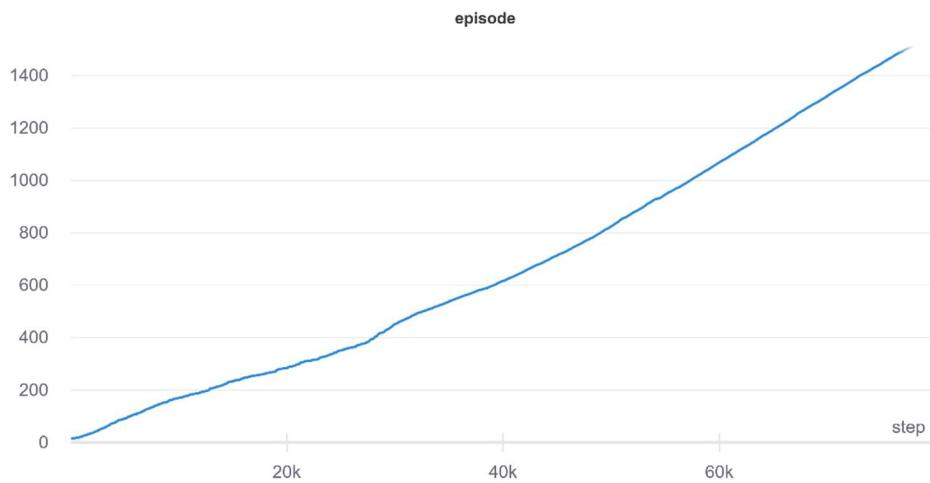
Teniendo en cuenta estas explicaciones y habiendo entrenado la red con la coordenada objetivo (50,100), concretamente, un objetivo a 111,80 metros aproximadamente, el valor máximo de episodio obtenible será aproximado de 83,85 metros más los 100 de alcanzar dicho objetivo, 183,85 mientras que el valor mínimo es siempre constante y es de 200. Se puede corroborar estos cálculos viendo la gráfica la cual mide la recompensa por episodio a lo largo de todos los episodios del entrenamiento. Para analizar dicha gráfica de un total aproximado de 1570 episodios se ha dividido en dos partes, del episodio 0 al 650 en la Figura 24 y del 650 al 1570 en la Figura 25 siendo también necesaria la gráfica que nos indique la relación entre pasos realizados y episodios dados, ya que los parámetros predeterminados están en su mayoría en la unidad de pasos, viéndose en la Figura 26



**Figura 24: Primera parte de la gráfica recompensa por episodio**



**Figura 25: Segunda parte de la gráfica recompensa por episodio**

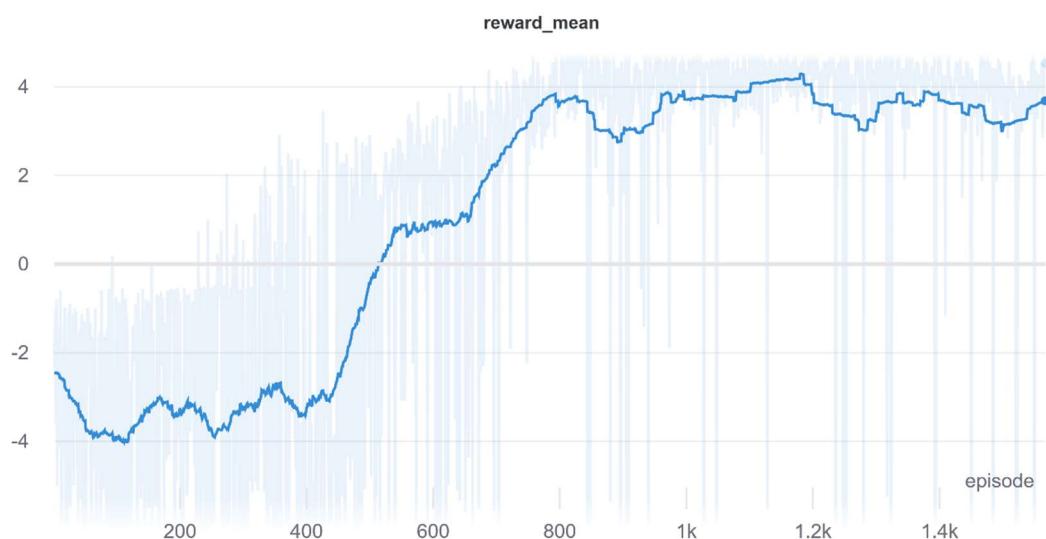


**Figura 26: Gráfica de episodios realizados respecto a los pasos**

Como se explicó en el apartado 4.3.3, los primeros 1000 pasos se utilizaban para recapitular memoria en concreto, no se está entrenando realmente y no se está buscando maximizar la recompensa por episodio. Viendo la gráfica Figura 26, este periodo corresponde aproximadamente a los primeros 50 episodios, si a esto se suma que como se vio en la Figura 18 los primeros 22 000 pasos se tiene un valor de  $\epsilon$  muy elevado y por ese motivo se está en la fase de exploración en la mayoría del tiempo, se observa que hasta el episodio 300 no habrá prácticamente episodios en fase de explotación y, por consiguiente, la recompensa por episodio es mínima en la mayoría de dichos episodios. Se corrobora en la Figura 24 que en su mayoría tiene un valor menor que -100 por la acumulación de acciones con recompensa negativa y por colisiones. A partir de este episodio aproximadamente, empieza a aumentar las recompensas y la red comienza aprender, viendo un aumento de dichas recompensas y llegando al aproximadamente al objetivo en uno de cada dos episodios entre el episodio número 350 y 450. A partir del episodio 500 se empieza a observar una convergencia alrededor del valor de 100, continua después aumentando hasta el

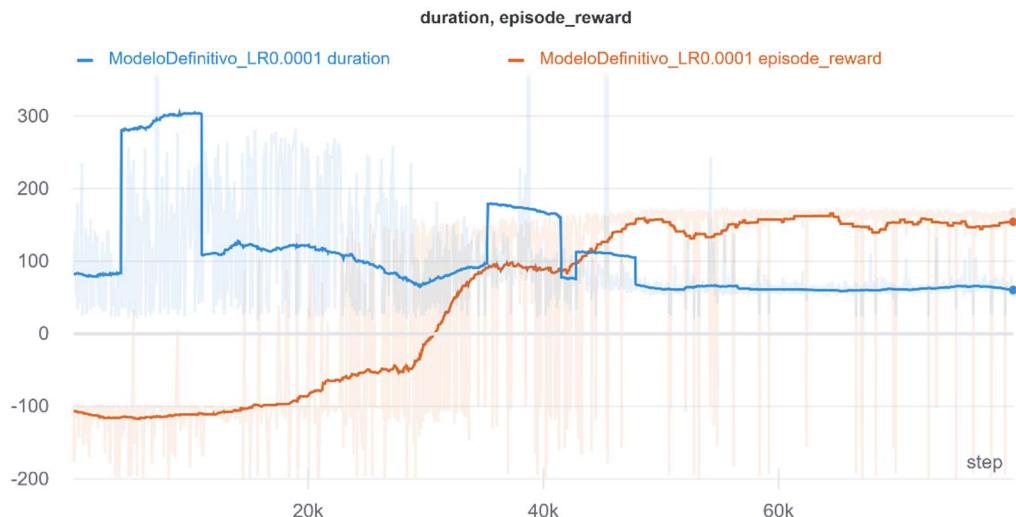
valor máximo de episodio, de aproximadamente 180 indicado anteriormente y que se puede ver a lo largo de la Figura 25. Esto se corresponde también a la llegada del valor mínimo de  $\epsilon$  a partir de los 50000 pasos concretamente, a que la red está en la gran mayoría de episodios entrenando y no recapitulando observaciones. Como se ve en los últimos 800 episodios o, dicho de otra forma, los últimos 25 000 pasos, se obtiene exceptuando algún episodio la mayoría de recompensa, dando por finalizado el entrenamiento.

Esto también se puede ver en la Figura 27 como aumenta la media de acciones, pasando de ser aproximadamente el valor mínimo de media, entre -4 y -5, al valor máximo mayor que +3. Esta gráfica obtiene valores menores de -5 y mayores que +3 ya que tiene en cuenta la recompensa de colisión y acierto, de -100 y 100, disminuyendo y aumentando el teórico valor por acción.

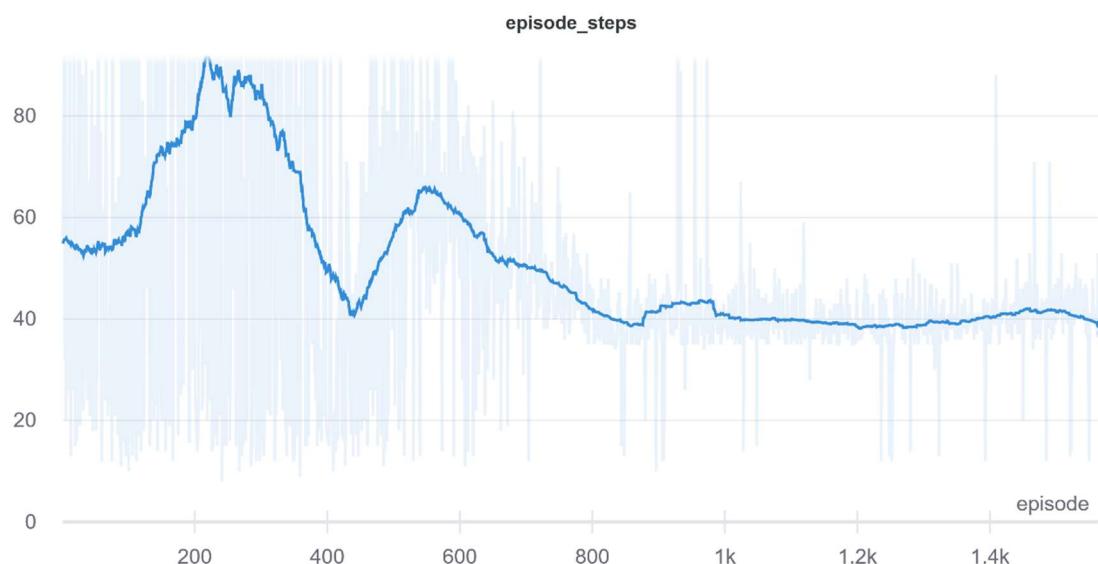


**Figura 27: Media por acción**

Otro dato interesante es analizar como a la vez que aumenta la recompensa máxima también disminuye el tiempo de entrenamiento, analizando que cuando ya converge en los últimos episodios con valor máximo el tiempo en alcanzar el objetivo es de aproximadamente de 50 segundos como indica la gráfica Figura 28

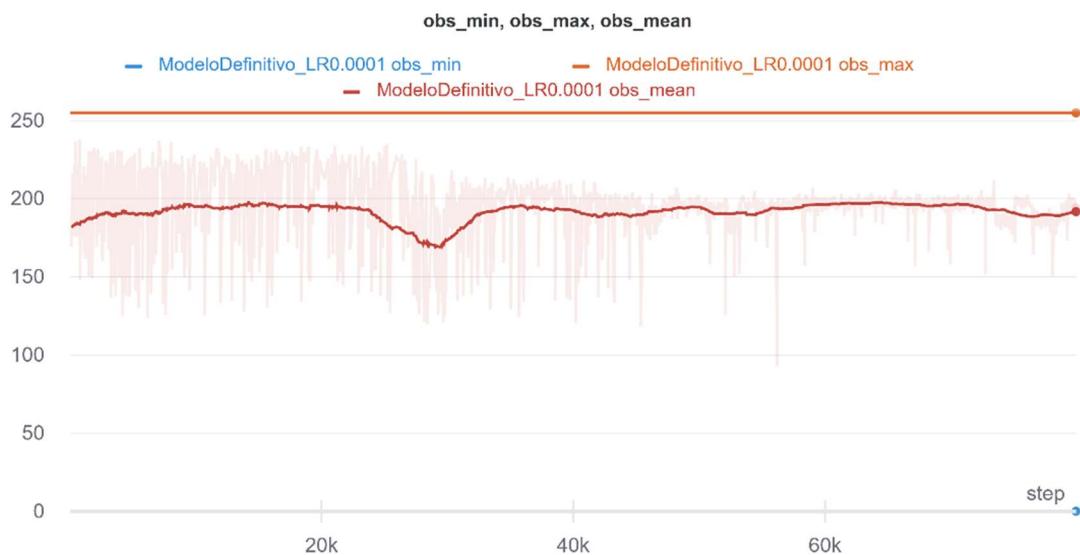


Este último dato también se verifica con la Figura 29 que muestra el número de pasos dados respecto a los episodios, mostrando que son aproximadamente de 40 pasos que suponiendo la máxima distancia que se puede ir acercando serían 160 metros recorridos por los 110 que se recorrerían idealmente. Esto no sucede por los obstáculos que tiene el entorno y que tiene que esquivar el dron y evita que gane siempre la máxima recompensa, pero se observa que es efectivo en la mayoría de los episodios y acciones.



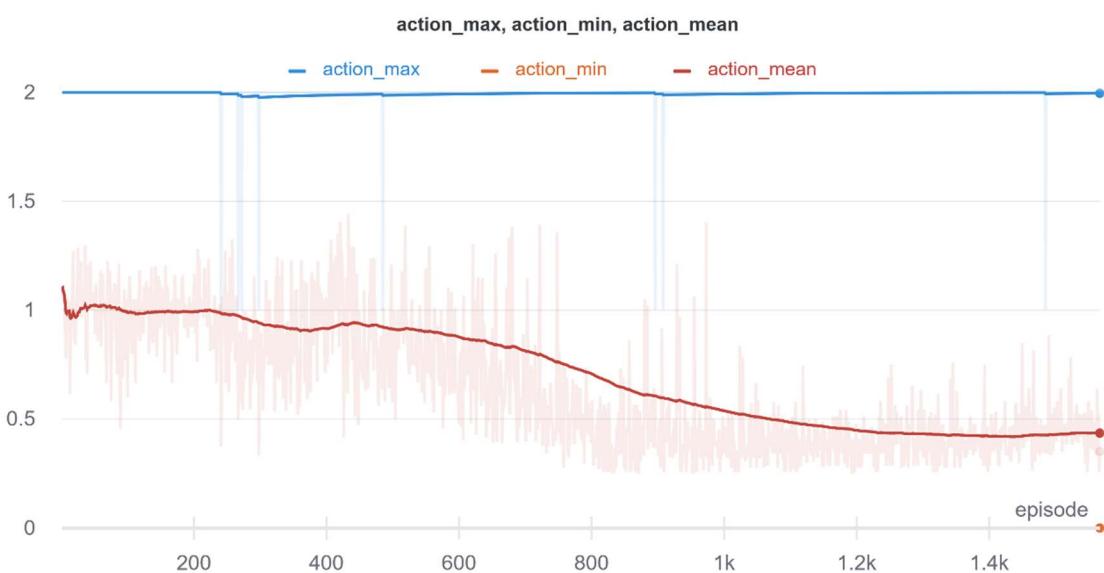
También observamos datos interesantes en las observaciones obtenidas, concretamente en los estados obtenidos por el dron, ya que se corrobora la indicación de los valores de las matrices obtenidas en el rango de 0 a 255 proporcional a los 0 a 100 metros del rango de la cámara de profundidad con un

valor constante de la observación mínima de 0, un valor constante máximo de y el promedio de estos valores en la Figura 30.



**Figura 30: Valor mínimo, máximo y promedio de observaciones respecto a los pasos**

En cuanto a los valores de las acciones predichas por la red, con valor de 0 para seguir recto en la dirección del dron, 1 girar hacia la derecha sobre sí mismo y 2 girar hacia la izquierda sobre sí mismo, se ve en la Figura 31 que en todos los episodios se han realizado las tres opciones, pero que la media de estas claramente pasa a estar en 0 y 1 según la red ha aprendido a alcanzar el objetivo. Esto tiene sentido ya que el objetivo estaba en el punto (50,100), o sea, a la derecha y delante del dron significando que, aunque tenga que esquivar los obstáculos, debe girar a la derecha y avanzar en la mayoría de las acciones.

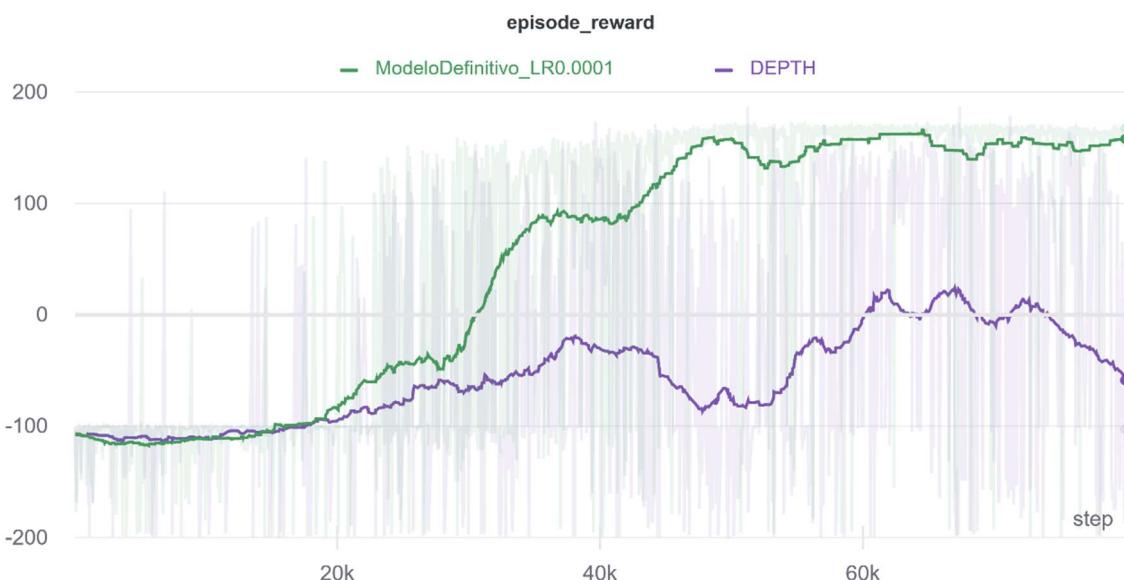


**Figura 31: Evolución de las acciones respecto a los episodios**

Por todos estos criterios, se considera que este fue finalmente un buen entrenamiento para este proyecto y podía ser utilizado como red predeterminada para la aplicación a falta de la validación explicada en el 5.4.1 y el ejemplo real realizado en el apartado 5.5.

### 5.3.1.1 Pruebas fallidas

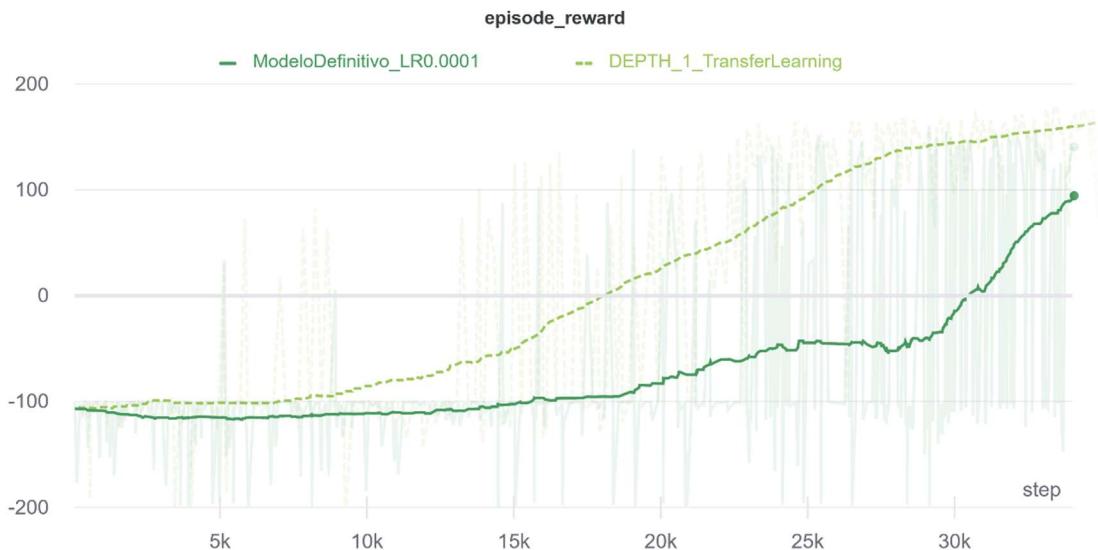
Antes de realizar este intento se ejecutaron varios entrenamientos con distintos parámetros. Cabe destacar que hay dos entrenamientos más realizados con el suficiente número de acciones para realizar la comparación. El primer entrenamiento fallido se realizó con una red neuronal con dos capas de profundidad y sin realizar Max Pooling, eligiendo como punto de referencia el mismo objetivo. También se aumentó el número de neuronas en las capas intermedias y no se utilizó la normalización de parámetros, teniendo como efecto un entrenamiento mucho más lento como se puede ver en la Figura 32, al mismo nivel de acciones realizadas convergiendo el entrenamiento fallido alrededor de un valor de 0 de recompensa, mientras que el entrenamiento correcto ya estaba convergiendo al nivel de la máxima recompensa.



**Figura 32: Comparación de entrenamientos con distintas redes neuronales**

En el segundo intento, se realizó con la red neuronal finalmente elegida, pero con un objetivo de referencia en el que no había prácticamente obstáculos en la ruta al destino. Esto se puede observar en el proceso para llegar a la recompensa máxima siendo éste excesivamente rápido y lo más importante, no se obtuvo prácticamente ningún valor mínimo. Esto es un defecto ya que la red es importante que aprenda que colisionar con obstáculos debido a que se trata de un error muy grave, y si no se enfrenta a estas situaciones no puede aprender de ellas. Sería como enseñar a la red a alcanzar objetivos, no a esquivar los

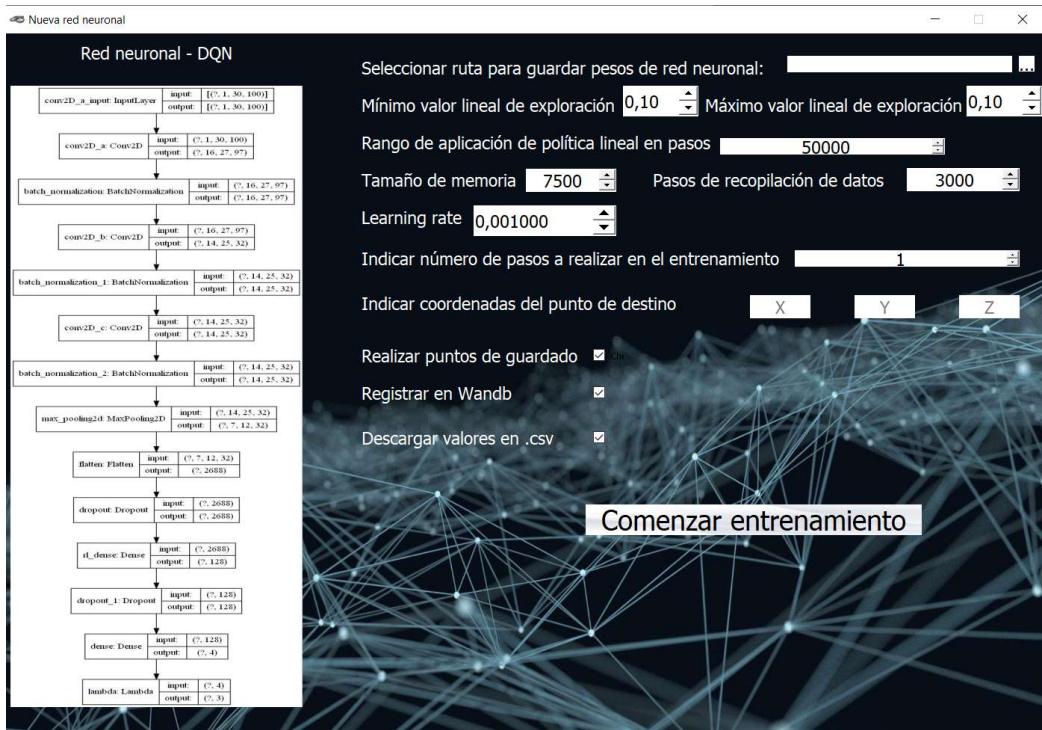
obstáculos en el trayecto. Se observa en la Figura 33 el excesivo cambio respecto al modelo finalmente elegido.



**Figura 33: Comparación entre modelos con distintos obstáculos en el entorno**

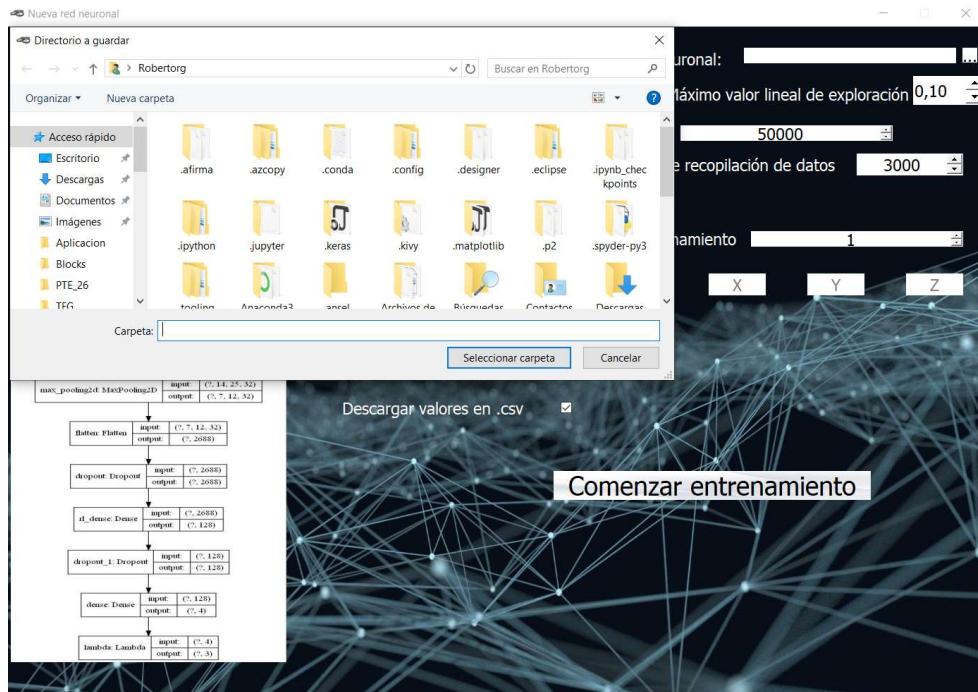
### 5.3.2 Interfaz gráfica modo entrenamiento

En este apartado se enseña finalmente la ventana que el usuario puede utilizar para crear su propio agente DQN. El modelo de red neuronal será el predeterminado, pero permitiendo la personalización del valor mínimo lineal de exploración, del valor máximo y del rango de aplicación, el tamaño de la memoria, de los pasos previos realizando la recapitulación de memoria, del learning rate, del número de acciones para completar el entrenamiento y de las coordenadas del punto destino. Además, se puede indicar si se desea guardar puntos de salvado intermedio, registrar en Wandb o descargar el csv final. Se indica un esquema de la red neuronal predeterminada, viéndose esta ventana en la Figura 34.



**Figura 34: Interfaz gráfica para entrenar nuevo agente DQN**

Resaltar el único parámetro que no se rellena por defecto, siendo este la ruta a la carpeta donde se guardará los pesos, mostrando un explorador para elegir dicha carpeta como se muestra en la Figura 35



**Figura 35: Buscar directorio donde guardar pesos**

Una vez llenados estos datos, basta con dar al botón de comenzar entrenamiento para iniciar dicha acción.

## 5.4 Modo validación

En este apartado se explica la validación de la red neuronal por defecto entrenada en el apartado 5.3.1 y se muestra la interfaz gráfica utilizada por el usuario para este modo.

### 5.4.1 Validación de red neuronal predeterminada

A continuación, se explican las validaciones realizadas, ejecutando 20 veces el mismo estado en tres puntos distintos de menor a mayor dificultad. En todos ellos se intentará que el dron consiga llegar a un objetivo que se marcará con un cilindro rojo.

En el escenario más sencillo, están separados por distancia en la dirección del dron, esto es en el eje X, de aproximadamente 45 metros y tiene entre medias una columna como obstáculo, mostrando este escenario en la Figura 36.



**Figura 36: Escenario de validación de dificultad baja**

Los resultados obtenidos de esta prueba se pueden ver en la Tabla 2, que nos muestra una media de recompensa de prácticamente 100, habiendo obtenido una única prueba con valor negativo. Esto significa que el resultado de acierto es del 95%.

**Tabla 2: Resultados de recompensa en escenario de dificultad baja**

Episodio	Recompensa
1	114,97
2	115,00
3	115,03
4	115,03
5	114,30
6	115,24
7	114,99
8	115,16
9	115,01
10	115,04
11	115,06
12	114,86
13	114,89
14	115,16
15	115,05
16	115,01
17	114,96
18	-198,40
19	112,63
20	115,02
Media de recompensa	
99,20	

El segundo escenario, se aleja en la misma dirección hasta estar a 50 metros y se añaden dos columnas más rodeando el objetivo como se muestra en la Figura 37.

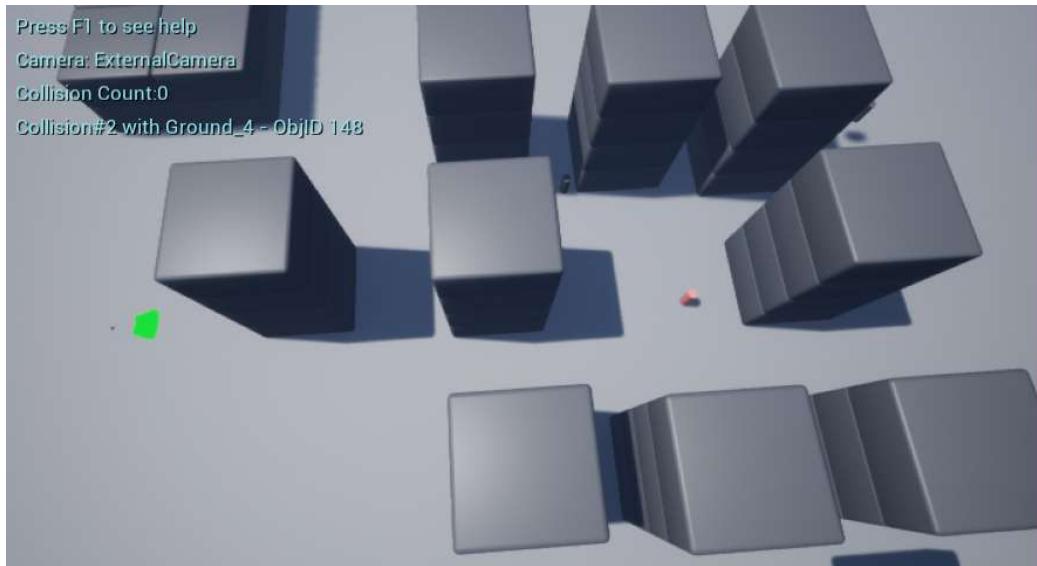
**Figura 37: Escenario de validación de dificultad media**

Los resultados obtenidos de este escenario se encuentran en la Tabla 3 y muestra una media de recompensa de 57,06, habiendo obtenido cuatro pruebas con valor negativo. Esto implica que se ha conseguido llegar con éxito al objetivo en un 80% de los casos.

**Tabla 3: Resultados de recompensa en escenario de dificultad media**

Episodio	Recompensa
1	113,89
2	114,72
3	-100,85
4	110,67
5	109,84
6	94,68
7	-100,94
8	116,58
9	57,96
10	84,48
11	-100,04
12	101,21
13	19,78
14	118,38
15	83,74
16	90,93
17	118,16
18	103,89
19	-100,97
20	105,18
<b>Media de recompensa</b>	
<b>57,06</b>	

En el último escenario, se rodea el objetivo de múltiples columnas manteniendo la distancia teniendo que sortear más obstáculos para llegar al objetivo, mostrándose en la Figura 38: Escenario de validación de dificultad alta



**Figura 38: Escenario de validación de dificultad alta**

En este último escenario se indican los resultados en la Tabla 4, mostrando que se han producido cinco recompensas negativas, esto es un acierto del 75% con una media de recompensa de 50,21.

**Tabla 4: Resultados de recompensa en escenario de dificultad alta**

Episodio	Recompensa
1	-100,12
2	89,15
3	94,22
4	95,91
5	112,71
6	117,22
7	-100,76
8	100,37
9	-100,61
10	88,31
11	-100,36
12	119,03
13	118,26
14	118,73
15	102,73
16	111,34
17	119,75
18	28,50
19	103,85
20	-113,94
Media de recompensa	
50,21	

Analizando los tres resultados en su conjunto cabe resaltar que errores por colisión sólo se han producido dos veces; es decir, en aproximadamente en el 3,33% de los casos mientras que, en total, no se ha llegado al destino en el 16,67% de los casos. Esta observación se puede corroborar con los resultados dados en las tablas, ya que los valores ligeramente inferiores a -100, siempre serán muy probables que sean acumulos de acciones negativas hasta llegar al número inferior a este umbral. Se puede decir que la red ha aprendido bastante bien a evitar las colisiones, efecto que era muy importante aprender. Sin embargo, cuando hay muchos obstáculos a pesar de no producirse colisiones, no sabe muy bien qué acción tomar para acercarse al objetivo en un porcentaje bajo de veces, en estas pruebas en total un 13,13% de las veces, aproximadamente. Estos resultados se pueden observar gráficamente en su conjunto en la Figura 39, indicando que se ha tenido entre todas las pruebas un 83,33 % de éxito.

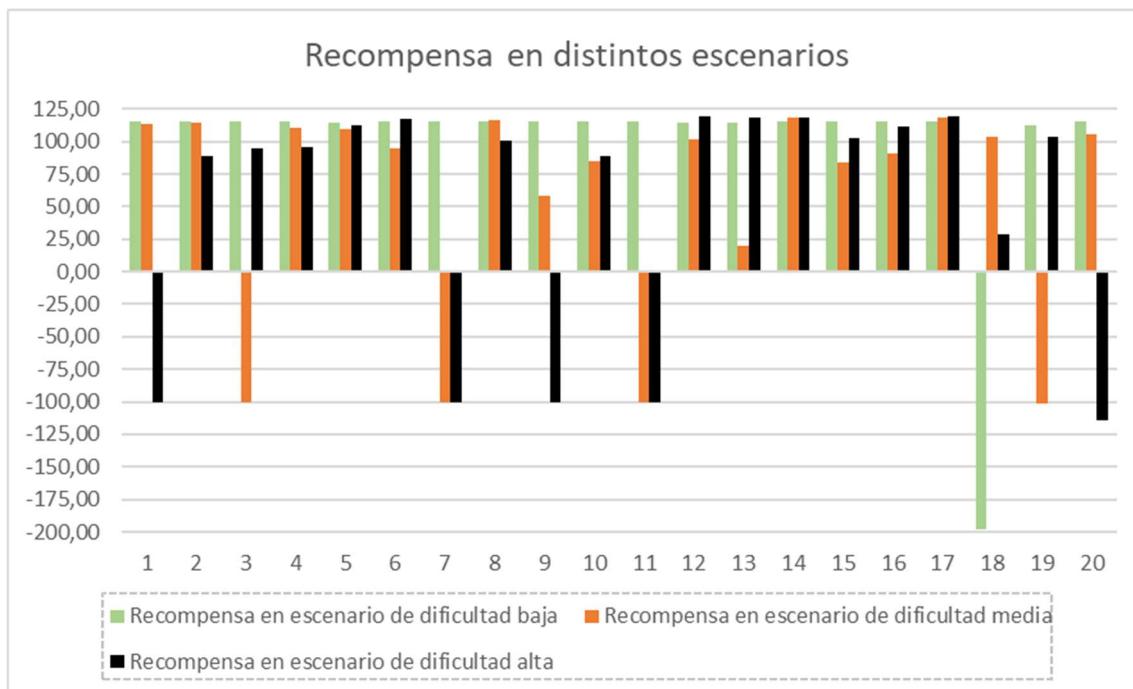
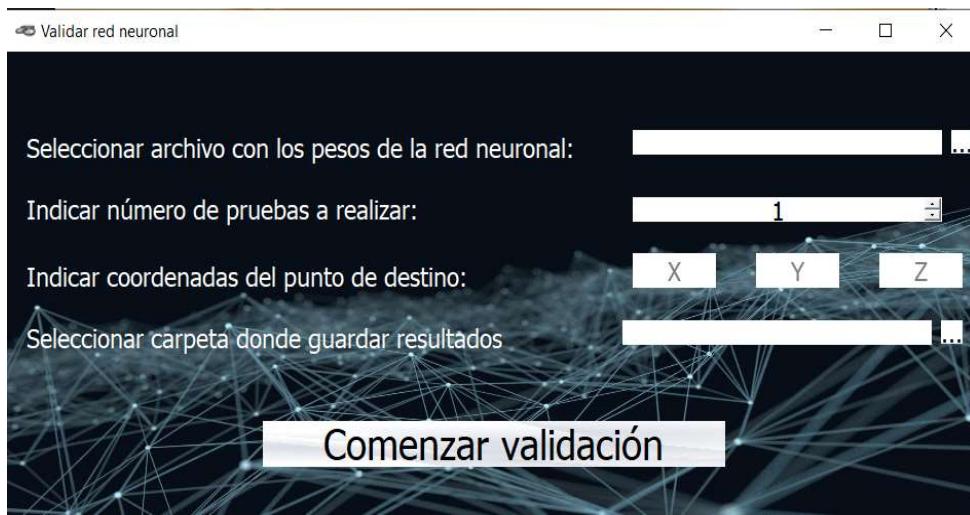


Figura 39: Validación de recompensa en distintos escenarios

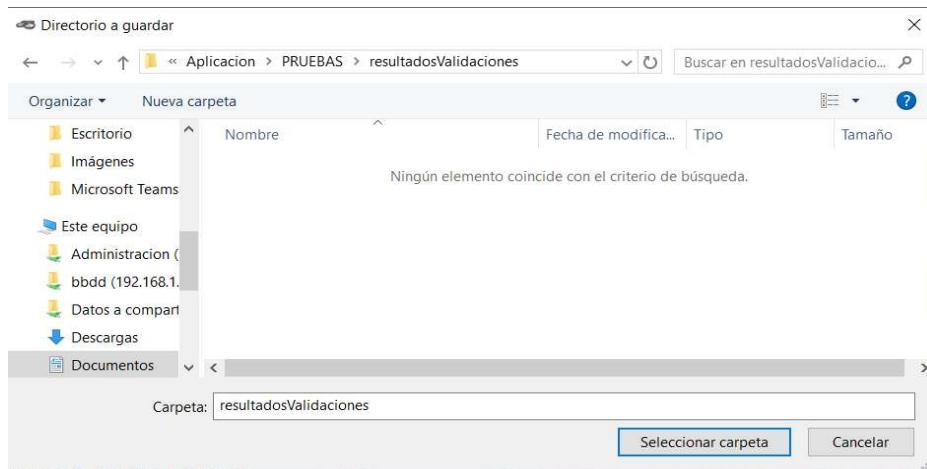
#### 5.4.2 Interfaz gráfica modo validación

En este apartado se enseña la interfaz gráfica final que utiliza el usuario para validar sus agentes DQN. En esta interfaz el usuario indicará el punto sobre el cuál realizar la validación y también el número de validaciones, dicho de otro modo, de episodios que desea, viéndose está ventana en la Figura 40

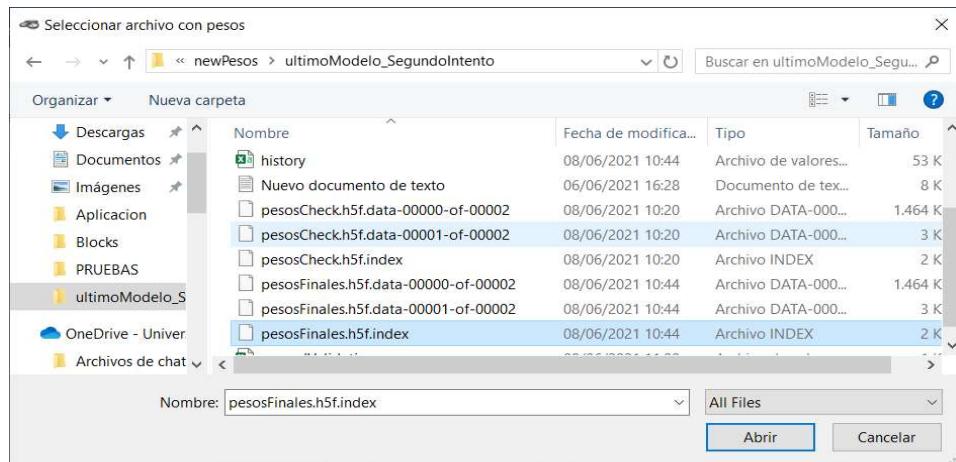


**Figura 40: Interfaz gráfica para validar agente DQN**

Además, deberá seleccionar el archivo *h5f.index* que contiene los pesos de la red neuronal y la carpeta donde se guarda el fichero *csv* con los resultados como muestra Figura 41 y Figura 42



**Figura 41: Selección de carpeta para guardar resultados validación**



**Figura 42: Búsqueda de fichero con los pesos de la red en modo validación**

## 5.5 Modo conducción autónoma

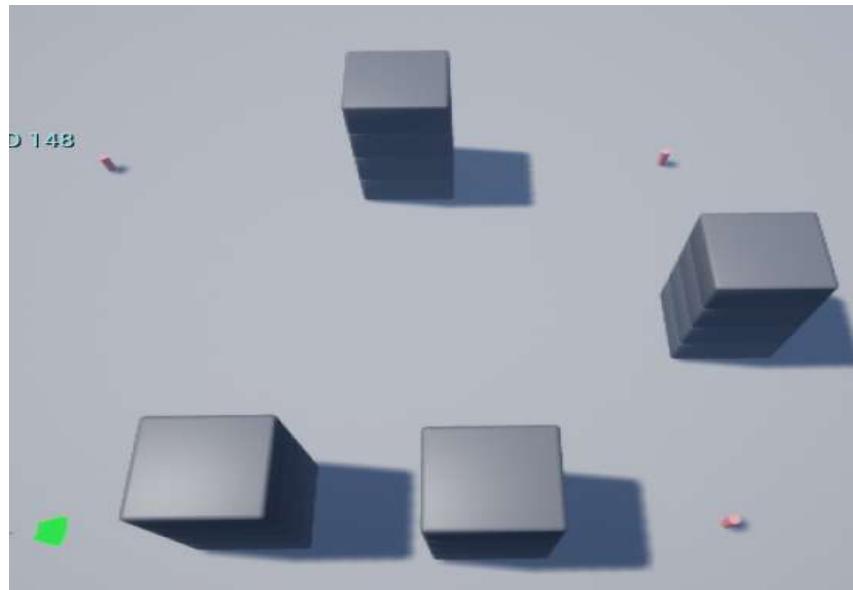
En este apartado se explica con un ejemplo el funcionamiento de la conducción autónoma con el agente DQN entrenado por defecto para este proyecto. La iniciación de este modo ejecuta la interfaz que se ve en la Figura 43.



Figura 43: Interfaz gráfica inicial del modo conducción autónoma

Esta interfaz se compone de un campo donde se selecciona los pesos de la red neuronal del mismo modo que se veía en la Figura 42. Dispone de tres campos donde se introducen las coordenadas de un punto y donde se añaden a la lista dando en el botón con el símbolo del más. Si se quisiese eliminar algún punto bastaría con seleccionar dicho punto en la lista y activar el botón del símbolo menos. A su vez, se dispone de otro campo donde se indica la ruta a la carpeta donde se guardarán las capturas que realiza el dron una vez llega a su destino con la cámara inferior, siendo esta búsqueda de carpeta idéntica a la representada en la Figura 41. El cuadrado con el icono del dron a la izquierda de la ventana, realizará la función de ir mostrando en directo lo que va visualizando el dron, pudiendo elegir tanto la cámara vertical como horizontal. En cuanto al icono de dron de la parte derecha de la interfaz gráfica, se muestra las imágenes que se van realizando según se va llegando a los puntos, pudiendo elegir estas desde la lista situada a la izquierda de esta zona.

Para iniciar el ejemplo se han introducido tres puntos, colocando tres cilindros rojos en cada posición como se muestra en la Figura 44, siendo el objetivo obtener las tres imágenes de los cilindros en el directorio sin recibir ninguna colisión de manera autónoma.



**Figura 44: Campo de simulación para la conducción autónoma**

Las coordenadas del dron y de estos puntos se ve en la tabla Tabla 5. Estás coordenadas, representadas en metros, están sacadas desde Unreal Engine y serán utilizadas para calcular las distancias que se deben rellenar en la interfaz gráfica.

**Tabla 5: Posiciones del dron y puntos de referencia.**

	Posición (m)	
	X	Y
Punto inicial del dron	-36,3	-77,5
1	26,3	-77,5
2	26,3	-131,2
3	-36,3	-131,2

Por tanto, si se restan las coordenadas de los puntos objetivos las coordenadas correspondientes del punto inicial del dron se obtiene la distancia en metros de cada coordenada, observándose en la tabla Tabla 6. Estos valores se introducen en la interfaz gráfica, añadiendo para la coordenada Z la altura a la que se quiere realizar la fotografía y se comienza la acción pulsando el botón de comenzar.

**Tabla 6: Distancias en las coordenadas X e Y**

Distancias (m)	
X	Y
62,6	0
62,6	-53,7
0	-53,7

Una vez comience la conducción autónoma se obtiene la figura Figura 45 con esos valores y la imagen que se va viendo en directo del dron, siendo ésta intercambiable entre la cámara vertical y horizontal.

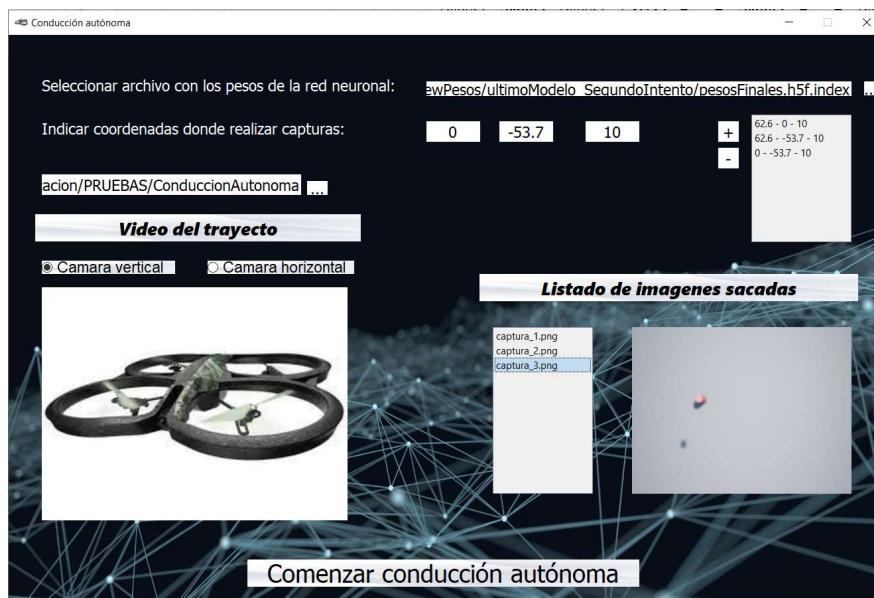
**Figura 45: Comienzo de conducción autónoma**

Una vez llegado a cada punto se realizará la fotografía y se irán añadiendo a la lista en la cual podemos seleccionar que se cargue la imagen, como por ejemplo se ve en la Figura 46 la imagen del primer punto



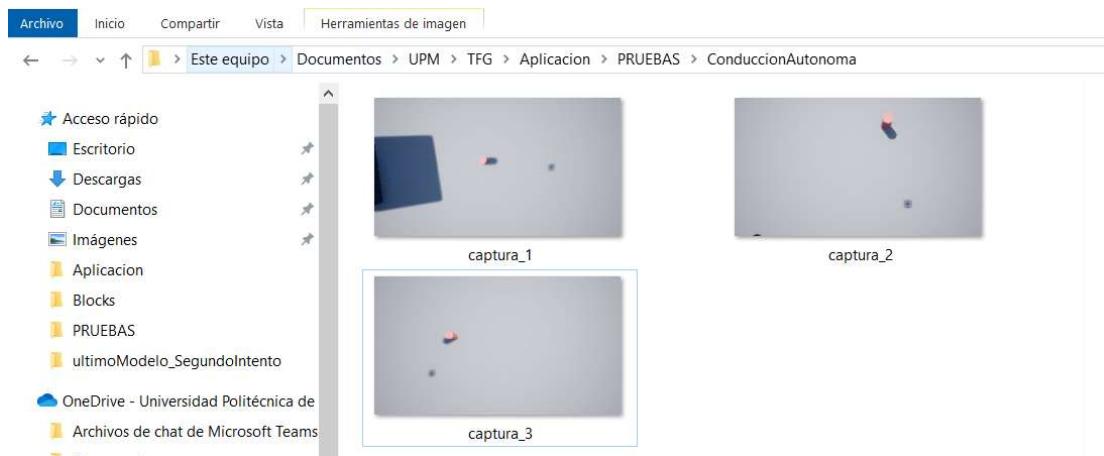
**Figura 46: Imagen cargada en modo de conducción autónoma**

Una vez finaliza el recorrido, obtenemos las tres imágenes pudiendo navegar por ellas seleccionando la que el usuario desee como se ve en la Figura 47



**Figura 47: Finalización del modo conducción autónoma**

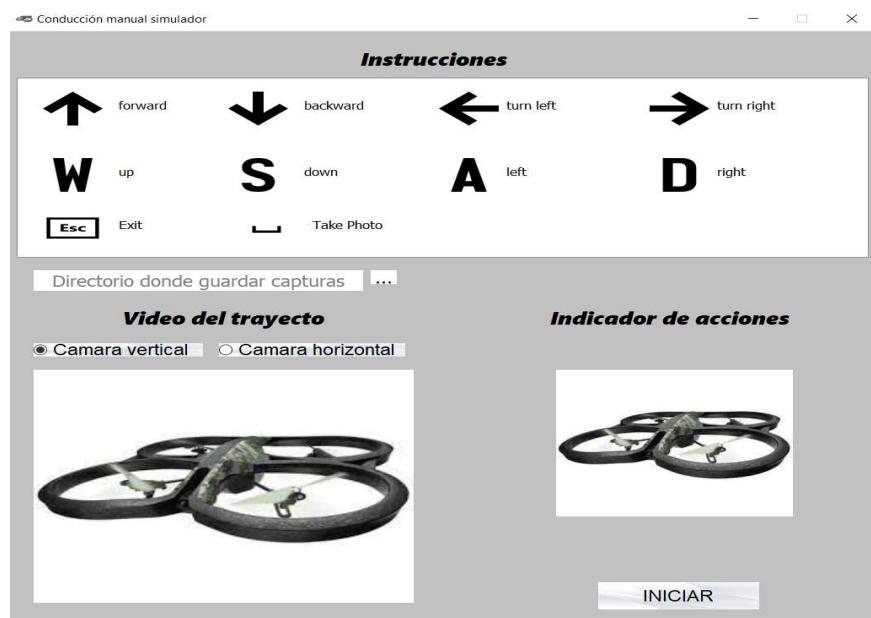
Por último, en el directorio previamente seleccionado observamos las tres capturas realizadas como se indica en la Figura 48.



**Figura 48: Directorio de salida en modo conducción autónoma**

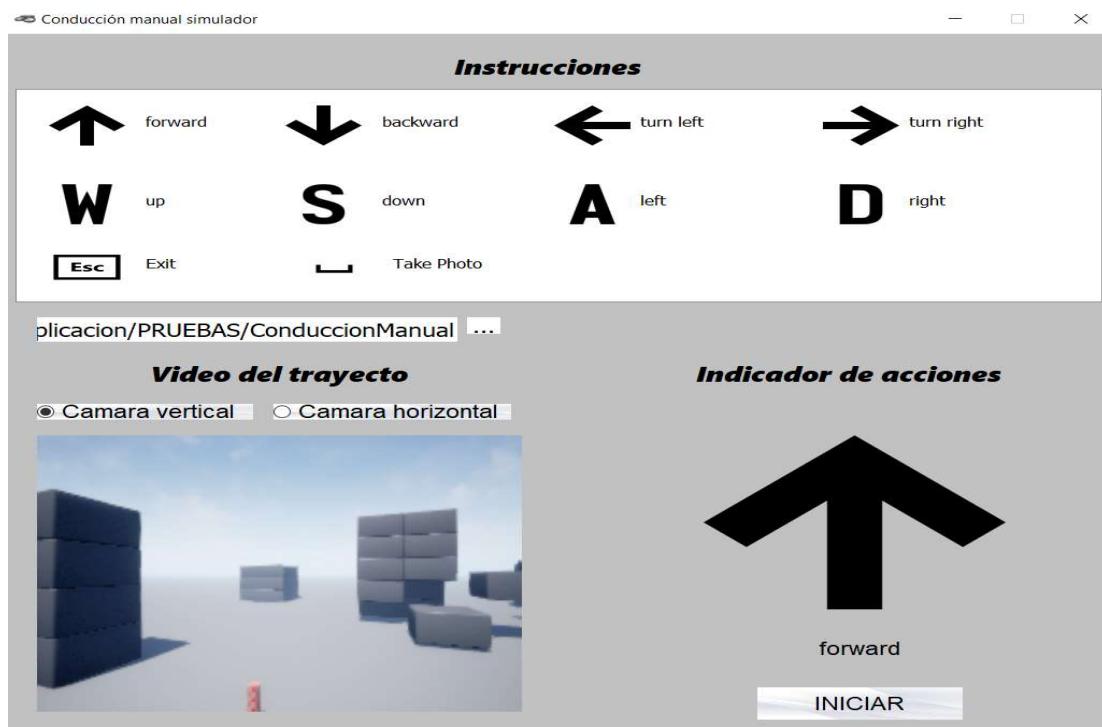
## 5.6 Modo pilotaje manual

El modo pilotaje manual dispone de una interfaz gráfica compuesta por un panel con las correspondientes equivalencias de las teclas del ordenador a las acciones que se pueden realizar en el dron. A su vez, se compone de un buscador de directorios como el de la Figura 41, para seleccionar la carpeta donde se guardarán las imágenes que se realicen. Estas imágenes se guardarán con un nombre específico, siguiendo el formato día, mes, año, concatenado con una barra baja y seguido de la hora, minuto y segundo. En el contenedor izquierdo se podrá observar el trayecto en directo, pudiendo cambiar entre cámara vertical y horizontal, realizando la captura de la misma. Por otro lado, el panel derecho irá mostrando la acción que se vaya activando con una imagen y una descripción de la misma en la zona inferior. La apariencia de esta interfaz según sea iniciada desde el menú tendrá la misma que en la Figura 49.



**Figura 49: Interfaz gráfica para conducción manual**

Como resultado se realiza la misma prueba hecha en el apartado 5.5 sacando capturas de los mismos cilindros rojos definidos. Este modo permite al usuario ser más preciso buscando objetivos ya que ofrece la oportunidad de acercarse y ajustar la posición de una manera sencilla. En la Figura 50 se observa la visualización en vivo junto con el indicador de acciones.



**Figura 50: Visualización de la interfaz conducción manual alcanzando un objetivo**

Para terminar, se pulsa la tecla escape y se ve como resultado en la Figura 51 las tres capturas realizadas desde el directorio elegido para guardar dichas imágenes.



**Figura 51: Resultado de la captura de imágenes en conducción manual**

# Capítulo 6

---

## 6 PRESUPUESTO



## 6.1 Introducción

En este capítulo se desglosa el presupuesto necesario para el desarrollo del proyecto creado y el presupuesto para la implementación del mismo en un dron real.

## 6.2 Presupuesto de la aplicación

Todos los materiales utilizados en cuanto software son gratuitos y no han supuesto ningún coste; por esta razón, sólo se desglosarán los costes de hardware en la Tabla 7 y por mano de obra en la Tabla 8. En definitiva, los costes totales se apuntan en la Tabla 9.

**Tabla 7: Costes de hardware en aplicación simulada**

Cantidad	Descripción	Coste
1	Ordenador MSI Prestige 15 A11SCS	1.399,00 €
SUBTOTAL		1.399,00 €

**Tabla 8: Costes por mano de obra**

Horas	Cargo	Coste	Coste
340	Ingeniero Telemático Desarrollador	18,00 €/h	6.120,00 €
40	Ingeniero Telemático Jefe	45,00 €/h	1.800,00 €
SUBTOTAL		7.920,00 €	

**Tabla 9: Coste total de aplicación simulada**

Concepto	Coste
Costes de hardware en aplicación simulada	1.399,00 €
Costes por mano de obra	7.920,00 €
SUBTOTAL	9.319,00 €

### 6.3 Presupuesto implementación real

Para la implementación en un dron real, se tendrá en cuenta el presupuesto resultante de la Tabla 9 y el presupuesto necesario para la implementación en un sistema real que se indica en la tabla Tabla 10, viendo el coste total en la Tabla 11

**Tabla 10: Costes de hardware en aplicación real**

Cantidad	Descripción	Coste
1	Parrot AR Drone 2.0 Elite Edition Snow	262,00 €
1	Intel RealSense Depth Camera D415	260,00 €
SUBTOTAL		522,00 €

**Tabla 11: Coste total de aplicación real**

Concepto	Coste
Costes de hardware en aplicación real	522,00 €
Coste total de aplicación simulada	9.319,00 €
SUBTOTAL	9.841,00 €

# Capítulo 7

---

## 7 CONCLUSIONES Y TRABAJOS FUTURO



Una vez realizado todo el proyecto, en este capítulo se desarrollarán las conclusiones y posibles trabajos futuros para mejorar el mismo. Recapitulando entre los objetivos marcados, se puede decir que se ha logrado alcanzar el objetivo principal con el desarrollo final, consiguiendo desarrollar una aplicación utilizable por el usuario en un entorno simulado para la obtención de imágenes. Para ello, también se ha logrado el objetivo de la utilización de drones e inteligencia artificial, conjuntándose ambos campos. Exactamente, se ha conseguido la correcta comunicación de los algoritmos de inteligencia artificial utilizando Deep Reinforcement Learning, específicamente DQN, con los drones simulados en un simulador como Unreal Engine mediante AirSim.

Como se ha explicado en los resultados, la aplicación ha conseguido alcanzar un porcentaje de acierto considerable haciendo la misma fiable a la hora de una posible implementación en un sistema real. Por otra parte, se habilitado en misma aplicación la posibilidad de mejorar más dando la opción de posibles nuevos entrenamientos, pudiendo realizarse entrenamientos aún más eficientes según la capacidad de computación posea el usuario. Como última opción, se permite la conducción manual del dron para un posible ajuste de posición, pudiendo complementar la inteligencia artificial con la interacción humana.

Gracias a las funciones habilitadas, este proyecto se puede enfocar a su vez a múltiples tareas no solo enfocados a la agricultura, si no a tareas como vigilancia, transporte de mercancía, inspección de líneas eléctricas, entre otras.

Por último, como posibles mejoras y trabajos futuros, se podrían aumentar el número de acciones como el movimiento en el eje Z ascendente y descendente, movimientos perpendiculares al eje de dirección del dron o, en definitiva, el aumento de acciones hasta configurar las mismas que se han implementado en conducción manual y se han evitado en la conducción autónoma por el tiempo que implicaría de entrenamiento. Precisamente, respecto a dicho tiempo, se podría reducir creando una implementación del entrenamiento en un sistema distribuido siendo una posible mejora del proyecto, junto al uso de otras librerías de Deep Reinforcement Learning que permitan utilizar entornos vectorizados; dicho de otra forma, el uso de multiprocesos para aumentar el tiempo de entrenamiento considerablemente y, en conclusión, realizarlo más robusto. Otra mejora a realizar, podría ser la posibilidad de darle al usuario la opción de crear su propio modelo de red neuronal, ya que, aunque pueda configurar los parámetros del agente DQN, se pueda indicar el número de capas convolucionales, tipos de funciones de activación, entre otros valores de la red. En cuanto a la personalización de red, otra posible mejora sería el añadir más redes predeterminadas actuales y más utilizadas como los tipos AlexNet, VGG, ResNet, entre otras.

En referencia a los estados utilizados (actualmente imágenes de profundidad) se podría implementar el uso de distintos tipos de imágenes de cámaras más económicas, pero con mayor tiempo de entrenamiento necesario por la dificultad de sacar información de ellas, como las imágenes RGB o sacadas por infrarrojos. Otro cambio sería la utilización de sensores, como el sensor LIDAR, utilizando para definir los estados arrays con los valores proporcionados por el mismo sensor.

Por último, el trabajo futuro a realizar más inmediato sería la implementación de la posible solución definida en el apartado 4.4 en un sistema real para su utilización y verificación de resultados, con su respectiva comparación con los valores obtenidos por el simulador analizando su viabilidad

# Capítulo 8

---

## 8 REFERENCIAS BIBLIOGRÁFICAS



## 8.1 Referencias

- [1] INE, Instituto Nacional de Estadística, «INE,» [En línea]. Available: <https://www.ine.es/jaxiT3/Datos.htm?t=3995>. [Último acceso: 21 Junio 2021].
- [2] Redagrícola, «El avance de la automatización en la agricultura,» Noviembre 2017.
- [3] The Hindu, «Drones used to spray micronutrient on crops,» 8 Enero 2021.
- [4] Aerocamaras, «Dron hibrido,» [En línea]. Available: <https://dronehibrido.com/es/>. [Último acceso: 21 Junio 2021].
- [5] Real Academia Española, Drone, Diccionario de la lengua española, 23.<sup>a</sup> ed., [versión 23.4 en línea], 2021.
- [6] L. E. Trejo Medina, J. M. Cabrera Peña, R. J. Aguasca Colomo y B. Galvan Gonzalez, «REVISIÓN TECNOLÓGICA, NORMATIVA Y APLICACIONES DE LOS VEHICULOS AEREOS NO TRIPULADOS EN LA INGENIERÍA (Parte 1),» *DYNA*, vol. 91, pp. 517-521, Septiembre 2016.
- [7] W. Garage, «ROS,» [En línea]. Available: <https://www.ros.org/>. [Último acceso: 30 Mayo 2021].
- [8] Proyecto Player, «Gazebo,» [En línea]. Available: <http://gazebosim.org/>. [Último acceso: 30 Mayo 2021].
- [9] CoppeliaSim, «Coppelia Robotics,» [En línea]. Available: <https://www.coppeliarobotics.com/>. [Último acceso: 2021 Mayo 30].
- [10] Microsoft, «AirSim,» [En línea]. Available: <https://microsoft.github.io/AirSim/>. [Último acceso: 2021 Mayo 31].
- [11] R. Flórez López, J. M. Fernández y J. M. Fernández Fernández, Las Redes Neuronales Artificiales Metodología y Análisis de Datos en Ciencias Sociales, Netbiblo, 2008.
- [12] A. Cartas, *Trabajo propio*, CC BY-SA 4.0.
- [13] Cburnett, *Colored neural network*, CC BY-SA 4.0.
- [14] W. Zhang, G. Yang, Y. Lin, C. Ji y M. M. Gupta, «On Definition of Deep Learning,» *2018 World Automation Congress (WAC)*, pp. 1-5, 2018.

- [15] Mathworks, «Mathworks,» [En línea]. Available: <https://es.mathworks.com/help/reinforcement-learning/ug/what-is-reinforcement-learning.html>. [Último acceso: 3 Junio 2021].
- [16] Anaconda, Inc, «Anaconda,» [En línea]. Available: <https://www.anaconda.com/>. [Último acceso: 04 Junio 2021].
- [17] Google Brain Team, [En línea]. Available: <https://www.tensorflow.org/>. [Último acceso: 4 Junio 2021].
- [18] François Chollet, «Keras,» [En línea]. Available: <https://keras.io/>. [Último acceso: 4 Junio 2021].
- [19] M. Plappert, «keras-rl,» GitHub, 2016. [En línea]. Available: <https://github.com/keras-rl/keras-rl>.
- [20] T. Oliphant, «Numpy,» [En línea]. Available: <https://numpy.org/>. [Último acceso: 6 Junio 2021].
- [21] Riverbank Computing, «Riverbank Computing,» [En línea]. Available: <https://riverbankcomputing.com/news>. [Último acceso: 6 Junio 2021].
- [22] Open AI, «Gym - Open AI,» [En línea]. Available: <https://gym.openai.com/>. [Último acceso: 6 Junio 2021].
- [23] Microsoft, «Microsoft github,» [En línea]. Available: [https://microsoft.github.io/AirSim/build\\_windows/](https://microsoft.github.io/AirSim/build_windows/). [Último acceso: 6 Junio 2021].
- [24] Microsoft, «Visual Studio Code,» [En línea]. Available: <https://visualstudio.microsoft.com/es/downloads/>. [Último acceso: 6 Junio 2021].
- [25] Qt Designer, «Qt Designer Manual,» [En línea]. Available: <https://doc.qt.io/qt-5/qtdesigner-manual.html>. [Último acceso: 11 Junio 2021].
- [26] YADrone, [En línea]. Available: <https://vsis-www.informatik.uni-hamburg.de.oldServer/teaching/projects/yadrone/index.html>. [Último acceso: 11 Junio 2021].

