GRADO EN INGENIERÍA EN TECNOLOGÍA DE TELECOMUNICACIÓN

# TRABAJO FIN DE GRADO

# *MODELOS DE APRENDIZAJE PROFUNDO POR REFUERZO (DEEP REINFORCEMENT LEARNING) PARA EL MOVIMIENTO AUTÓNOMO DE VEHÍCULOS AEREOS NO TRIPULADOS*

**Alumno:** Chuga Perugachi, José Daniel

**Director:** Del Ser, Lorente, Javier

**Curso:** 2019 - 2020

**Fecha:** Martes, 10 de Febrero de 2020

PÁGINA EN BLANCO

# Resumen trilingüe

## Resumen

El objetivo de este proyecto consiste en el desarrollo de un sistema basado en algoritmos de Reinforcement Learning, que sea capaz de otorgar autonomía a un vehículo aéreo no tripulado, el cual deberá aprender a realizar un desplazamiento autónomo desde un punto origen, hasta un punto destino, sin llegar a colisionar con los obstáculos que se pueda encontrar a lo largo de su trayectoria. Para ello, se harán uso de las herramientas software necesarias, que permitan la simulación de escenarios tanto para el entrenamiento, como para la evaluación del algoritmo.

**Palabras clave:** agente, entorno, aprendizaje por refuerzo, Q-learning, dron.

## Laburpena

Proiektu honen helburua Reinforcement Learning-eko algoritmoetan oinarritutako sistema garatzea izango da, non gida gabeko aire ibilgailu bati jatorrizko puntu batetik helmuga punto baterainoko ibilbidea burutzeko autonomía emango zaio. Horrenbestez, ibilgailuak zehaztutako ibilbidea egiten ikasi beharko du aurki ditzakeen oztopoak saihestuz. Horretarako, entrenamendu zein ebaluazio inguruneak simulatzeko beharrezkoak diren software tresnak erabiliko dira.

**Hitz gakoak:** agente, ingurunea, errefortzu bidezko ikaskuntza, Q-learning, drone.

## Abstract

The aim of this Project is the development of a system based on Reinforcement Learning algorithms, which is capable of granting autonomy to an unmanned aerial vehicle, which must learn to make an autonomous displacement from an origin point, to a destination point, without colliding with the obstacles that can be found along its path. For this, the necessary software tools will be used, which allow the simulation of scenarios both for training and for the evaluation of the algorithm.

**Keywords**: agent, environment, reinforcement learning, Q-learning, drone.

# Índice

# Lista de ilustraciones y figuras



# Lista de Graficas.



# Lista de Tablas

# Lista de acrónimos

**IA** Inteligencia artificial.

**PX4** PIXHAWK Project.

**SITL** Software in the loop.

**HITL** Hardware in the loop.

**ROS** Robot Operating System

**API** Application Programming Interface.

**UAV** unmanned aerial vehicle.

**MOCAP** motion capture.

**RGB** red, green, blue.

**DQN** Deep Q-Network.

**SDG** Stochastic Gradient Descent.

**DDQN** Double Deep Q-Network.

# 1 Introducción

Actualmente la Inteligencia artificial (IA) está muy presente dentro del entorno de las personas, aunque en la gran mayoría de casos estas no son conscientes de ello. El interés que existe por dar autonomía a ciertos procesos o actividades, aumentando así su productividad, es uno de los grandes motivos por los que este concepto ha adquirido tanta importancia durante la última década.

La inteligencia artificial se define como la combinación de algoritmos capaces de otorgar cierto conocimiento a las máquinas, para que estas puedan desarrollar acciones como lo haría un ser humano. En este marco, el campo de Machine Learning es una de las ramas que más ha crecido, donde sus algoritmos se basan en aprender a partir del procesamiento de una gran cantidad de datos. Este proceso consiste en la definición previa de un problema, y a partir de múltiples ejemplos, crear un modelo que generalice el comportamiento observado, de tal manera que sea capaz de evaluar y crear predicciones para nuevos casos que se le presente. Dentro de Machine learning los algoritmos se clasifican en 3 subcampos de aprendizaje diferentes: Aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

- **Aprendizaje supervisado:** el entrenamiento del modelo se realiza en base a una gran cantidad de datos previamente clasificados.
- **Aprendizaje no supervisado**: a diferencia del aprendizaje supervisado, en este caso los datos de entrada con los que se entrena el modelo no se encuentran etiquetados. El aprendizaje reside en la comprensión y abstracción de patrones de información.
- **Aprendizaje por refuerzo:** en este caso, el modelo aprende en base a la información intercambiada con el entorno que le rodea. Tiene una similitud con el aprendizaje no supervisado al no recibir información etiquetada, sin embargo, este se encarga de aprender en función de las recompensas recibidas por las acciones llevadas a cabo.

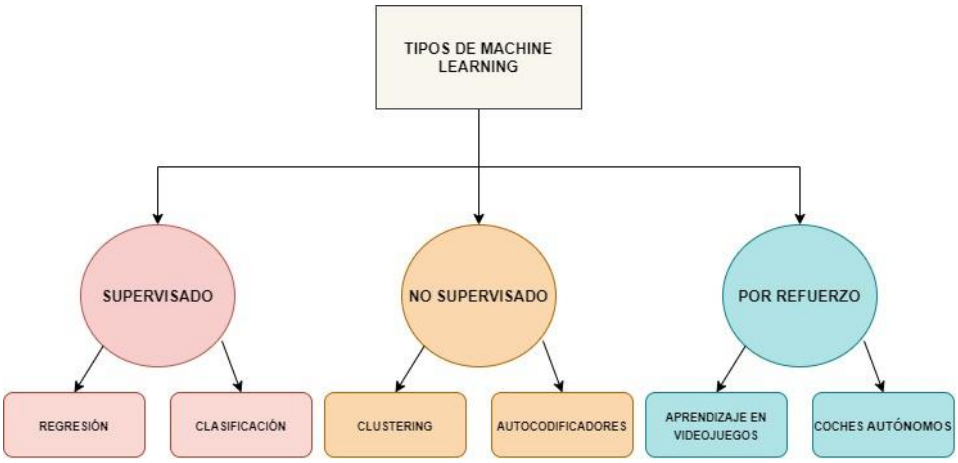



**Figura 1:** Tipos de aprendizaje en Machine Learning.

Estos tipos de aprendizaje los podemos encontrar en diversos sectores de la sociedad, donde un claro ejemplo son los teléfonos móviles. La mayoría de los Smartphones cuenta con asistentes personales, los cuales a través de un reconocimiento de voz, son capaces de ejecutar órdenes dentro del dispositivo. Por otra parte, existen diversos estudios basados en estos algoritmos, que han sido capaces de dar soluciones en otros ámbitos como la medicina, permitiendo predecir algunas enfermedades con mayor probabilidad que los médicos, en el ámbito de las finanzas para la detección de fraudes y blanqueo de dinero o incluso en la industria de vehículos, donde se pretende dotarlos de autonomía para facilitar el desplazamiento. Este último está siendo cada vez más explotado, con el fin de conseguir sistemas que puedan circular automáticamente cumpliendo los objetivos marcados. Uno de los vehículos más de moda en los últimos años ha sido el dron, debido a la simplicidad que requiere su entrenamiento en comparación con otros vehículos y a los múltiples beneficios técnicos y económicos que supone hacer uso de ellos, sobre todo en tareas que requieren de un gran despliegue.

# 2 Contexto

Un dron o drone consiste en un vehículo aéreo no tripulado, al cual se le puede incorporar accesorios como cámaras, GPS y sensores de todo tipo, para que puedan ser usados en diferentes ámbitos y para diversas finalidades. Hace unos años este tipo de vehículos solo era imaginable para uso militar, sin embargo, gracias al abaratamiento de los costes de fabricación, se hizo posible que cada vez más se puedan introducir dentro del uso cotidiano, convirtiéndose en un producto comercial. Tal es la importancia que ha adquirido su uso, que hoy día se pueden ver varios ejemplos donde se emplean drones para dar solución a tareas que antes requerían de procesos más engorrosos.

Un ejemplo claro se encuentra en la provincia de Cáceres [1], donde se está haciendo uso de drones para llevar a cabo la vigilancia frente a la extracción ilegal de agua en los cauces del río Tajo. Además, se pretende facilitar y agilizar la detección de otro tipo de irregularidades que vayan en contra del medio ambiente. También en España, se ha empezado a establecer la monitorización y control de tráfico [2] para identificar infracciones de una manera más efectiva y económica, el cual requiere de un operativo técnico y humano más reducido.

Finalmente, cabe destacar el propósito de la empresa UPS en California [3], donde se ha puesto en marcha un nuevo servicio para el transporte de suministros médicos entre hospitales, de esta manera se busca reducir los retrasos y costes económicos que supone el traslado de medicamentos a determinadas zonas de difícil acceso, donde la entrega se debe realizar mediante helicópteros médicos.

Estos ejemplos consolidan que el uso de drones para facilitar labores en algunos sectores, es un mercado que se encuentra en auge, gracias a su facilidad para amoldarse a cualquier situación que conlleve un desplazamiento aéreo. Además, como se ha visto en el ejemplo de la empresa UPF, si se consigue dotar de cierto conocimiento al dron, se puede llegar a conseguir una mayor reducción en los costes. Se puede decir que el crecimiento en cuanto al uso de estos vehículos, ira ligado a los avances tecnológicos dentro del campo de la inteligencia artificial.

Es por esto, que hoy en día se pueden encontrar una gran cantidad estudios sobre la actividad de drones dotados de conocimiento, capaces de completar tareas de forma autónoma eliminando cualquier intervención humana. Llevar a cabo este tipo de procesos requiere de un sistema capaz de entender el entorno en el que se encuentra, seleccionar la mejor acción a realizar en cada momento y medir las consecuencias que conlleva ejecutar dicha acción. Precisamente este tipo de sistemas, es lo que permite resolver una de las áreas dentro del Machine Learning denominada Reinforcement Learning o aprendizaje por refuerzo.

El objetivo de Reinforcement Learning consiste en proporcionar a un agente la capacidad de aprender a comportarse en un entorno, ejecutando ciertas acciones en base a un aprendizaje asentado en recompensas y penalizaciones. En este marco, es donde se ha destacado la plataforma de simulación Airsim, construida bajo Unreal Engine, uno de los mejores entornos

de desarrollo para la creación de videojuegos o simulaciones. Airsim está pensado para llevar a cabo simulaciones que permitan entrenar algoritmos de aprendizaje por refuerzo, tanto en drones como en coches.

# 3 Objetivos y alcance del trabajo

Partiendo de la plataforma mencionado en el apartado anterior, se pretende elaborar un nuevo modelo que sea capaz de realizar el desplazamiento autónomo de un agente desde un punto de partida a un punto destino, evitando los obstáculos presentes en su trayecto. Para ello será necesario elaborar diversos escenarios, haciendo uso del entorno de desarrollo Unreal Engine, en los cuales estarán establecidos múltiples obstáculos que el agente deberá sortear para llegar a su destino. El agente será entrenado mediante un algoritmo de Reinforcement learning, el cual le permitirá aprender a driblar los objetos que le produzcan una penalización a lo largo de su trayectoria.

Con este estudio, se pretende llevar a cabo una demostración de la capacidad que puede adquirir un agente para realizar acciones autónomas en función del entrenamiento que se le otorgue. En primer lugar, se dotará al agente de la capacidad de alcanzar un destino dado dentro de un escenario inicial, sin que este llegue a colisionar. Una vez completada esta tarea, se pasará a comprobar la actuación de dicho agente en un nuevo escenario, con el fin de que pueda llevar a cabo las mismas acciones que en el primer escenario donde ha sido entrenado. De esta manera se pretende demostrar que el agente ha adquirido el conocimiento necesario para resolver el problema planteado, independientemente del escenario en el que se encuentre.

Para conseguir completar estos objetivos, será necesario la instalación del software encargado del entorno de simulación (Unreal Engine) y codificación (Visual Studio 2017), además del aprendizaje sobre el uso de las librerías del simulador AirSim. Por otra parte, se deben adquirir ciertos conocimientos sobre Reinforcement learning, ya que el código a realizar estará asentado en las funciones matemáticas que describen el algoritmo Double Deep Q-Learning.

# 4 Beneficios

## 4.1 Beneficios Técnicos

La implementación de este tipo de soluciones, como ya se ha visto en los ejemplos introducidos en el apartado de Contexto , permite obtener grandes beneficios técnicos gracias a su fácil adaptación en diversos sectores, sobre todo en los que se requiere de un desplazamiento. El simplemente uso de drones, sin la introducción de IA, ya facilita la elaboración de ciertas actividades, al poder llegar a zonas de difícil acceso o al proporcionar diferentes ángulos de visión. Si además se les proporciona conocimiento para que puedan desarrollar estas actividades de forma autónoma, el beneficio obtenido es mayor, gracias a la reducción de errores y a las acciones preventivas que pueda llevar a cabo.

## 4.2 Beneficios Económicos

Los beneficios técnicos que proporcionan estas soluciones, pueden ser implementadas por las empresas, haciendo que estas reduzcan sus costes en cuanto a personal técnico. El hecho de dotar a una máquina de cierto conocimiento para que pueda llevar a cabo una acción, al igual que la haría un ser humano, puede otorgar una mayor rentabilidad en cuanto a productividad y costes.

## 4.3 Beneficios Sociales

En contraposición, como se puede deducir, el sector laboral se puede ver perjudicado en algunos ámbitos, dando origen a una problemática moral. La incorporación de la automatización origina que detrás de cada máquina, pueda existir una pérdida de trabajo. Sin embargo, desde otro punto de vista, se gana en cuanto a fiabilidad y seguridad en la elaboración de las tareas, ofreciendo así un mejor servicio de cara al usuario final. Por otra parte, el uso de este tipo de vehículos en entornos hostiles para un humano, como pueden ser las situaciones de rescate en áreas de difícil acceso, garantiza la disminución de riesgos a tener en cuenta.

# 5 Análisis del estado del arte

El desarrollo y análisis de algoritmos de aprendizaje en vehículos autónomos, es un proceso complejo que requiere de mucho tiempo para la realización de pruebas, las cuales no siempre proporcionan resultados satisfactorios. Por otra parte, en este tipo de estudios es necesario recopilar una gran cantidad de datos de entrenamiento que garanticen unos buenos resultados. Esta recopilación de datos en el mundo real es inviable, ya que sería necesario pasar por todos los escenarios posibles incluyendo los menos satisfactorios, que en la gran mayoría de casos representan colisiones. Es por ello que hoy en día se ha llevado a cabo el desarrollo de diferentes entornos de simulación, que permiten imitar de la mejor manera posible el comportamiento de un vehículo, dentro de un entorno real. En este apartado se hará mención de los entornos virtuales más potentes en la actualidad, para poder desarrollar estudios de simulación en cuanto a vehículos autónomos.

## 5.1 Entornos de simulación

### 5.1.1 Gazebo

Gazebo [4] es un potente simulador 3D para robots autónomos, permitiendo analizar el comportamiento de estos en cuanto a la acción con objetos y la visión computacional. Es un programa de código abierto distribuido bajo la licencia Apache 2.0, que soporta uno de los controladores de vuelo más populares en la actualidad, PX4, tanto con SITL (Software in the loop) como con HITL (Hardware in the loop). Este simulador permite simular escenarios complejos, realizar pruebas de regresión y entrenar un sistema de inteligencia artificial. Por otra parte, cuenta con una gran comunidad de usuarios, permitiendo una gran expansión en cuanto a su desarrollo. Entre todas sus características podemos destacar la simulación dinámica, múltiples sensores (de contacto, fuerza de choque, cámaras 2D/3D...), comunicación con servidores mediante TCP/IP y diversos modelos de vehículos ya creados para realizar pruebas, donde podemos encontrar: *Quad* (iris y Solo*), Tail-Sitter*, aviones, *VTLO* y en un futuro será capaz de simular submarinos.

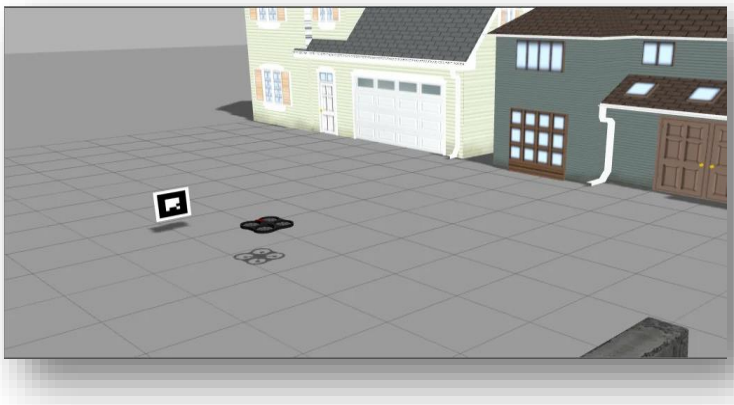


**Ilustración 1:** Entorno de simulación Gazebo, Imagen extraída de [15].

### 5.1.2 jMAVSim

JMAVSim [5] es un simulador simple y fácil de usar, creado para la simulación de vehículos aéreos no tripulados en un entorno virtual. Al igual que Gazebo, hace uso del controlador de vuelo PX4 con la posibilidad de usar tanto SITL o HITL. En cuanto a su aspecto gráfico, está creado con Java3d, y hace uso del protocolo UDP para las comunicaciones, además es compatible con el framework ROS (Robot Operating System). En este caso solo soporta los vehículos multirotor para la simulación.


**Ilustración 2:** Entorno de simulación jMAVSim, Imagen extraída de [16].

### 5.1.3 Sim4CV

Sim4CV [6] es un simulador de entrenamiento y evaluación fotorrealista, capaz de simular automóviles, vehículos aéreos no tripulados y actores humanos animados en diversos entornos 3D. Este simulador está creado sobre el motor de juego Unreal Engine, el cual le otorga un gran realismo en los objetos simulados. A su vez, permite el uso de diversas arquitecturas de deep neural networks para entrenar y dotar de un aprendizaje autónomo a los vehículos simulados.

**Ilustración 3:** Entorno de simulación Sim4CV, Imagen extraída de[17].

### 5.1.4 AirSim

Airsim [7] es un simulador de código abierto y multiplataforma, el cual también hace uso de Unreal Engine para la renderización de la simulación. Sus características más destacadas se encuentran en la posibilidad de usar la interfaz MAVlink y el uso de HITL o SITL junto con el controlador de vuelo PX4, para obtener una simulación física y visualmente mas realista. Por otra parte, implementa cámaras monoculares y profundas, así como diversos sensores de movimiento y detección de colisión. El objetivo de Airsim es proporcionar una plataforma de investigación para poder experimentar con la visión computacional, el cual consiste en enseñar a los ordenadores a "ver". De esta manera se pretende que los vehículos simulados obtengan información a través de las imágenes captadas, y junto a la combinación de algoritmos de aprendizaje, sean capaces de adquirir cierta autonomía. Para este propósito, Airsim, pone a disposición el acceso a sus APIs para la recepción de datos y control de vehículos, independientemente de la plataforma.


**Ilustración 4:** Entorno de simulación Airsim, Imagen extraída de [18].

## 5.2 Algoritmos de aprendizaje por refuerzo.

Por otra parte, además de tener en cuenta el simulador a emplear, es importante estudiar qué tipo de algoritmo se va a utilizar para el aprendizaje del dron. Como se ha mencionado en la introducción, para la resolución de problemas donde se requiere que el sistema desempeñe acciones de una forma autónoma dentro de un entorno, se hace uso de algoritmos de Reinforcement Learning. Estos algoritmos permiten que las máquinas sean capaces de aprender a través de su entorno, recibiendo una recompensa o penalización en función de la acción que haya tomado, de esta manera aprende a decidir qué acciones tomar en cada momento, con el fin de cumplir el objetivo propuesto maximizando la recompensa acumulada. Por lo tanto, se definen dos sujetos partícipes en este aprendizaje: agente y entorno. El entorno es el escenario donde se encuentra el agente, y del cual se obtiene la información necesaria para el entrenamiento de este. En cuanto al agente, es el sistema encargado de ejecutar las acciones dentro del entorno.

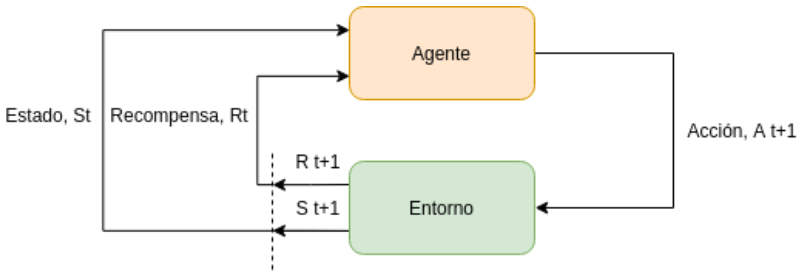



**Figura 2:** Representación del aprendizaje por refuerzo.

Uno de los puntos que diferencia a este tipo de aprendizaje respecto otros, como el aprendizaje supervisado o no supervisado, reside en la inexistencia de una colección de datos previos que permitan al agente encontrar una solución al problema planteado. En este aprendizaje, el agente parte de cero, es decir, no es informado sobre qué acciones debe realizar en los diferentes estados que se encuentre, sino que debe descubrir qué acciones realizar a medida que este va avanzando dentro del entorno.

Es aquí donde surge el concepto de exploración y explotación, el cual permite que el agente pase por una fase previa, donde ejecuta acciones aleatorias observando el resultado de estas y a medida que el entrenamiento avanza, ejecuta las acciones en función de la experiencia adquirida. Dicho de otra forma, para que el agente pueda adquirir una mayor recompensa debe explotar las acciones que le han funcionado en el pasado, a su vez, debe existir un proceso de exploración que le permita descubrir nuevos estados que le proporcionen una mayor recompensa. Durante el entrenamiento debe existir un equilibrio entre estos dos procesos, para ello se cuenta con varias técnicas que pueden ser útiles. Estas políticas básicas son: *greedy, ε-greedy*, y *soft-max*.

Antes de enunciar algunos de los algoritmos más importantes de Reinforcement Learning para la resolución de este tipo de problemas, se definen tres de los elementos principales dentro de un sistema de aprendizaje por refuerzo, que ayudarán a la compresión del funcionamiento de los algoritmos. Estos elementos son: la política, la función recompensa y la función valor.

- La **política** es la que determina el comportamiento del agente seleccionando la opción más óptima o, dicho de otra forma, es el mapeo de los posibles estados del entorno y las acciones que deben tomarse en esos estados.

- La **función recompensa** define el objetivo a completar dentro de un problema de aprendizaje por refuerzo. En cada paso, el agente recibe una recompensa del entorno, y debido a su objetivo de maximizar la recompensa acumulada, esta recompensa indicará si el estado es perjudicial o beneficioso para el agente.

- La **función valor** estima la recompensa total que puede adquirir un agente a futuro desde el estado en el que se encuentra.

Una vez introducidas estas características, se da paso a describir los algoritmos más significativos dentro del Reinforcement Learning.

### 5.2.1 Q-Learning.

El objetivo de este algoritmo es aprender una serie de normas que le permitan al agente decidir qué acción realizar en función del estado en el que se encuentre. El agente debe aprender, mediante las recompensas que le proporciona el entorno, a elegir la secuencia de acciones que

maximice la recompensa acumulada, donde la recompensa acumulada es la suma de todas las recompensas obtenidas a lo largo del episodio de entrenamiento. Se define $Q(s_t, a_t)$ como el valor máximo de recompensa que se puede obtener en un estado $s_t$ tomando la acción $a_t$.

En los casos en los que el conjunto de estados que definen el entorno y las acciones a tomar son valores finitos, se puede representar estos valores $Q(s_t, a_t)$ en forma de tabla, donde las acciones son las filas y los estados las columnas. El valor almacenado en cada celda representa la probabilidad de realizar dicha acción en ese estado.

Los valores de las celdas $Q(s_t, a_t)$ se van actualizando en función de la ecuación Bellman [8]:

$$Q(s_t, a_t) = (1 - \alpha) * Q(s_t, a_t) + \alpha * [R + \delta * \max[Q'(s_{t+1}, a_{t+1})] \quad (1)$$

Esta ecuación se puede definir como, la suma del valor ya almacenado en la tabla Q más el valor del nuevo aprendizaje. El nuevo valor de aprendizaje corresponde a la suma entre la recompensa inmediata R y el valor máximo de $Q'(s_{t+1}, a_{t+1})$, de entre todas las acciones del siguiente estado t+1, multiplicado por el factor de descuento (gamma).

Gamma y Alpha son dos parámetros de la ecuación de Bellman, comprendidos entre 0 y 1, que tienen una gran influencia sobre el aprendizaje del agente.

- **Alpha o ratio de aprendizaje**: afecta a la segunda parte de la ecuación, el cual representa el nuevo aprendizaje adquirido. Por lo tanto, este parámetro nos permite valorar la importancia de la recompensa adquirida, donde un valor cercano a 0 dará más importancia a lo que ya sabemos, actualizando lentamente la tabla, y un valor cercano a 1 dará más importancia al nuevo aprendizaje adquirido, acelerando la actualización de la tabla.

- **Gamma o factor de descuento**: dentro de la ecuación se encuentra multiplicando a las posibles recompensas futuras, determinando la importancia de estas. Por lo tanto, si el valor es cercano a 0, se valorará más las recompensas inmediatas que las recompensas que se puedan adquirir a futuro. En consecuencia, un valor cercano a 1 dará más importancia a las recompensas futuras que a las inmediatas.

El funcionamiento del algoritmo sería el siguiente:

Inicializar la tabla de estados $Q(s,a)$
Repetir para cada episodio hasta que no se alcance un estado final:
Inicializar el estado S.
Repetir para cada step hasta que no se alcance un estado final:
- Observar el estado S.
- Selecciona una acción a siguiendo la política que esté utilizando el agente. (Ejemplo: ε-greedy)
- Tomar la acción a, la recompensa R y el estado futuro $s_{t+1}$
- Aplicar la fórmula de Bellman para actualizar el valor de Q (s,a).

### 5.2.2 SARSA

Un algoritmo similar a Q-Learning que también deriva del método basado en diferencia temporal. Es un método on-policy, es decir, tiene una política inicial y la actualiza al final de cada episodio. En este caso, la actualización de los valores Q se obtiene a través de una aproximación sobre el valor de Q en el siguiente estado.

$$Q(s_t, a_t) = (1-\alpha) * Q(s_t, a_t) + \alpha * [R + \delta * \max[Q'(s_{t+1}, a_{t+1})] \quad (2)$$

Como podemos ver, a diferencia de Q-Learning donde se escoge la mejor acción de un estado $s_{t+1}$ para la actualización, SARSA elige una acción a siguiendo la política que esté utilizando el agente. Para entender mejor este cambio, veamos el funcionamiento de este algoritmo:

Inicializa todos los valores $Q(s,a)$
Repetir para cada episodio hasta que no se alcance un estado final:
Inicializar el estado S0.
Selecciona una acción a siguiendo la política que está utilizando el agente.
(Ejemplo: ε- greedy)
Repetir para cada step hasta que no se alcance un estado final:
- Con la acción $a_t$ observo el reward obtenido y el siguiente estado $s_{t+1}$
- Interpreta la acción $a_t$, pasa al estado siguiente $S_{t+1}$ obteniendo una recompensa $R_t$.
- Selecciona una nueva acción $a_{t+1}$del estado $s_{t+1}$siguiendo la política que esté utilizando el agente. (Ejemplo: ε-greedy)
- Actualizo el valor de $Q(s_t, a_t)$ con los valores obtenidos hasta ahora y haciendo uso de la expresión de SARSA.
- Actualizo los valores $a_t = a_{t+1}$; $s_t = s_{t+1}$.

### 5.2.3 Deep Q-Learning

Otra alternativa del aprendizaje por refuerzo es Deep Q-Learning, DQN, donde se incorporan redes neuronales. Como ya se ha mencionado antes, cuando el número de estados y acciones es finito, es posible representar los valores $Q(s_t, a_t)$ en una tabla, sin embargo, en un entorno virtual como puede ser un videojuego o un simulador, los estados a los que puede acceder un agente tienden a ser infinitos. En este tipo de entornos, crear y actualizar una tabla Q no es muy eficiente. No obstante, el uso de redes neuronales permite obtener a partir de un estado $s_t$, un array de valores Q, el cual representa el valor obtenido para cada acción en ese estado.

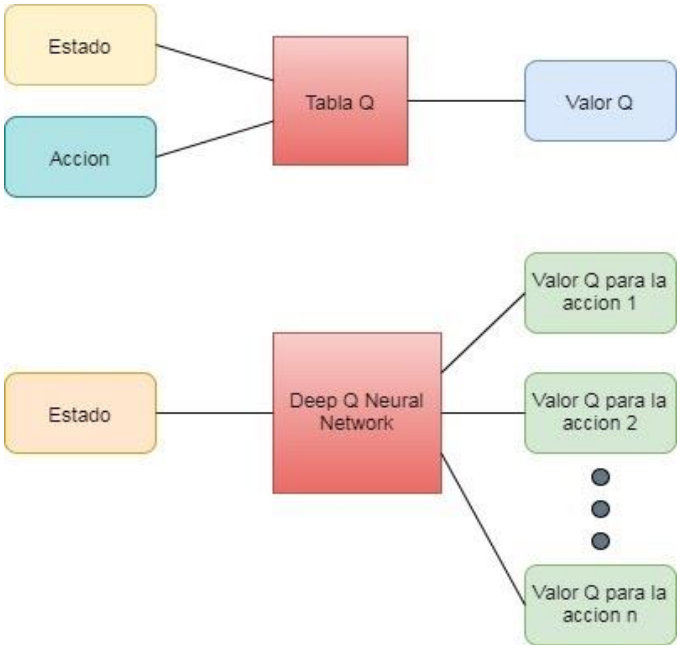



**Figura 3:** Diferencias entre Q-Learning y Deep Q-Learning.

Para que el proceso de aprendizaje sea más eficaz se hace uso del concepto Replay Memory [14], con el objetivo de reducir la correlación que puede ocasionar el hecho de usar muestras consecutivas durante el entrenamiento. Replay memory es un buffer en el que se van almacenando muestras que representan la experiencia del agente, del cual se extraen de forma aleatoria, cada cierto tiempo, un conjunto de muestras para el entrenamiento de la red.

Al introducir redes neuronales dentro de este algoritmo, el término de aprendizaje dentro de este marco consiste en minimizar la función error o pérdida asociada. La función error la podemos expresar de diversas formas dependiendo del método de cálculo que se utilice. En este caso la podemos representar como:

$$Loss = Q_{target} - Q_{actual} \quad (3)$$

Esta expresión refleja la diferencia entre el valor máximo posible de Q para el siguiente estado, y la predicción actual del valor Q. Como el objetivo es minimizar este parámetro, existen diversos tipos de optimizadores que nos permiten ajustar su valor, donde uno de los más conocidos es SDG (Stochastic Gradient Descent).

Nuevamente, para entender mejor este algoritmo se procede a explicar su funcionamiento:

Repetir para cada episodio hasta que no se alcance un estado final:
Inicializar el buffer Reply Memory.
Obtener el estado S0.
Repetir para cada step hasta que no se alcance un estado final:
Selecciona una acción $a_t$ siguiendo la política que está utilizando el agente (Ejemplo: ε-greedy).
Interpreta la acción $a_t$, pasa al estado siguiente $S_{t+1}$ obteniendo una recompensa $R_t$.
Almacena la tupla ($S_t$ , $S_{t+1}$, $a_t$, $R_t$) en el Replay Memory.
Actualizo la función de error:

$$Q_{target} = R + \delta * \max[Q'(s_{t+1})]$$
$$Loss = Q_{target} - Q_{actual}$$

### 5.2.4 Double Deep Q-Learning

Este algoritmo consta de dos redes neuronales, Q-Network y Target-Network, una para el cálculo del valor de Q-actual y otra para el cálculo del valor estimado Q-target. Esto permite resolver el problema de sobreestimación que puede darse en algunos casos.

El funcionamiento es similar al algoritmo DQN, salvo que en este caso se utiliza una red neuronal adicional para calcular el valor Q-target. Este valor se obtiene de la siguiente forma:

1- Se hace uso de la red Q-network para elegir la mejor acción a para el estado $S_{t+1}$. Para ello seleccionamos el valor máximo proporcionado por la red:

$$a = \max[\, Q_{Q-net}(S_{t+1}, a)] \quad (4)$$

2- Una vez hallada esta acción se utiliza la nueva red, Target-Network, para calcular el valor de Q estimada con la acción obtenida en el paso anterior.

$$Q_{estimada} = \max[\, Q_{T-net}(S_{t+1}, a)] \quad (5)$$

3- Con esta nueva Q estimada se da paso a calcular el valor de la $Q_{target}$:

$$Q_{target} = R + \delta * Q_{estimada} \tag{6}$$

4- Calculo la función de pérdidas utilizando la ecuación (3).

Cada cierto intervalo de repeticiones, se actualizan los parámetros de la red Target-Network con los de la red Q-network.

$$Q_{T-net} = Q_{Q-net}$$

En este caso, el parámetro de aprendizaje Alpha no se tiene en cuenta a la hora de realizar la actualización, ya que se utiliza en la etapa de optimización.

# 6 Análisis de alternativas

En este apartado se procede a evaluar los diferentes casos expuestos en el apartado anterior, tanto para la selección de un simulador adecuado, como para la elección de un algoritmo de Reinforcement Learning, que permita entrenar el agente de una forma óptima.

Por un lado, la elección del simulador adecuado dependerá tanto del objetivo del proyecto, como de las diferentes características que ofrezca el simulador. En algunas aplicaciones es necesario incluir aspectos físicos como la densidad del aire, cizalladura del viento, nubes, precipitación y otros aspectos climáticos. En cuanto al aprendizaje, se han introducido diversos algoritmos que permiten entrenar a un agente para que sea capaz de llevar a cabo una tarea.

## 6.1 Entornos virtuales

### 6.1.1 Ventajas y desventajas

#### 6.1.1.1 Gazebo

Como se ha comentado en el apartado anterior, Gazebo es un potente simulador que se centra en la simulación de robots, es por esto que el realismo obtenido por este simulador es inferior al de los otros simuladores expuestos. Al centrarse en la simulación de robots genéricos, no aporta gran ayuda a la hora de crear entornos realistas. Por otra parte, su principal ventaja reside en la gran comunidad de usuarios que lo rodea, permitiendo así obtener una gran documentación de ayuda para la elaboración de otros proyectos.

#### 6.1.1.2 JMavSim

Al contrario que Gazebo, este simulador se centra únicamente en la simulación de vehículos aéreos no tripulados (UAV). Su principal ventaja está en la sencillez a la hora de utilizarlo, sin

embargo, esta sencillez y el hecho de contar con un motor de renderizado muy simple, hace imposible la creación de objetos y obstáculos en el entorno. Además, tampoco cuenta con sensores que ayuden al agente en el descubrimiento del entorno.

#### 6.1.1.3 Sim4CV

Este simulador está desarrollado sobre el motor de juegos Unreal Engine, permitiendo de esta manera obtener un alto nivel de realismo. Se presenta como un simulador fotorrealista muy útil para estudios relacionados con la visión artificial, para ello hace uso de técnicas como MOCAP (motion capture), el cual permite recrear mejor los movimientos complejos haciendo que estos sean más realistas. La principal desventaja a destacar, es la difícil integración de otras tecnologías con su propio framework.

#### 6.1.1.4 Airsim

Finalmente, se da paso a una de las herramientas más potentes en la actualidad en cuanto a simulación de vehículos. Airsim, al igual que Sim4CV, usa el motor gráfico de Unreal Engine y hace uso de MOCAP para obtener un gran nivel de realismo. A su vez, tiene la posibilidad de importar diversos escenarios ya creados y disponibles en la plataforma de Unreal. Este simulador tiene como finalidad convertirse en una de las mayores plataformas para contribuir en la investigación y desarrollo de vehículos autónomos, es por esto, que cada vez se encuentran más trabajos y documentos sobre diferentes tipos de proyectos. Esto garantiza poder obtener más información de ayuda a la hora de llevar a cabo el proyecto.

### 6.1.2 Selección de la alternativa

Para seleccionar la mejor alternativa, se resumen las características de los simuladores presentados en la siguiente tabla:

| **Simulador** | **Gazebo** | **JMavSim** | **Sim4CV** | **Airsim** |
|---|---|---|---|---|
| **Vehículos** | UAVs, robots | UAVs | UAVs, coches | UAVs, coches |
| **Sensores** | fácil modificación de sensores | no incorpora | sensores RGB-D | cámaras monoculares y de profundidad, sensor de choque |
| **Motion Capture** | No | No | Si | Si |
| **Obstáculos** | Si | No | Si | Si |

| | | | | |
|---|---|---|---|---|
| **Realismo** | bajo | medio | alto | alto |
| **Documentación** | alto | bajo | medio | alto |

**Tabla 1:** Comparación de simuladores.

En este tipo de estudios es necesario contar con un espacio que se encuentre caracterizado por una gran cantidad de detalles, los cuales permitan disponer de un entorno lo más realista posible, ya que este será el que proporcione la información necesaria para el entrenamiento de del agente. Por lo tanto, Gazebo y JMavSim, no son las mejores alternativas para llevar a cabo este trabajo. En cuanto a los entornos Sim4CV y Airsim, el hecho de implementar el motor de juegos de Unreal Engine e incorporar Motion capture, hace que el entorno adquiera un alto nivel de realismo con una gran cantidad de detalles. Aunque ambas herramientas serian útiles para este estudio, ya que las diferencias entre ellas son mínimas, se opta por elegir Airsim. Este entorno dispone de una mayor documentación respecto a Sim4CV y la integración de otras tecnologías con su propio framework es menos aparatosa.

## 6.2 Algoritmos de aprendizaje por refuerzo

### 6.2.1 Ventajas y desventajas

Una vez seleccionado el entorno de desarrollo que se va a utilizar, se procede a analizar los diferentes algoritmos expuestos en el apartado anterior. Los dos primeros algoritmos, SARSA y Q-Learning, son muy similares entre sí, donde SARSA puede considerarse una versión más general de Q-Learning. La única diferencia reside en la actualización de los valores Q, sin embargo, dependiendo del caso de estudio, el rendimiento obtenido por cada uno de los algoritmos puede ser una gran diferencia. Esta diferencia la se puede ver reflejada en el ejemplo de caminar por acantilados de Sutton y Barto [9], donde el algoritmo Q-learning se decanta por el camino más óptimo y SARSA por el camino más seguro.

Por otra parte, Deep Q-Learning y Double Deep Q-Learning, son algoritmos a los que se incorporan redes neuronales. Estos algoritmos son muy útiles para añadir el término aprendizaje por refuerzo dentro de entornos virtuales como videojuegos o simulación de vehículos autónomos, donde los estados por los cuales tiene que pasar el agente son innumerables. Al igual que en el caso anterior, estos algoritmos son similares y la diferencia principal reside en el cálculo de la función de pérdidas, la cual se debe minimizar para garantizar que el aprendizaje sea óptimo. El método para calcular los valores de la función de pérdidas del algoritmo Deep Q-Learning en entornos ruidosos, tiende a dar problemas de sobreestimación. Esto se debe al uso de la misma red, con los mismos parámetros (pesos), para el cálculo del valor estimado de Q-target y el valor actual de Q. Por otro lado, el algoritmo Double Deep Q-Learning, reduce este problema gracias a la incorporación de una segunda red neuronal. A diferencia de DQN, donde el cálculo del valor de $Q_{target}$ se realiza a través del valor máximo de Q para un estado $S_{t+1}$,

DDQN hace uso de la red primaria para seleccionar una acción $a_t$, y usa la nueva red, Target network, para el cálculo del valor de $Q_{target}$ con la acción $a_t$, de esta manera se evita la convergencia de los resultados de este parámetro.

### 6.2.2 Selección de la alternativa

Por una parte, los dos primeros algoritmos, SARSA y Q-Learning, se ven limitados a un número de estados, con lo cual para estudios como el que se quiere llevar a cabo, donde el número de estados por los que tiene que pasar el agente tiende a ser infinito, la utilidad de estos algoritmos es propensa a ser baja. Es por esto, que se implementan redes neuronales para corregir esta limitación. Los dos siguientes algoritmos, DQN y DDQN, implementan esta solución, los cuales serían factibles para dar un resultado al problema planteado. Sin embargo, teniendo en cuenta que nuestro agente trabaja en un entorno donde el tiempo de entrenamiento tenderá a ser alto, y dando así una mayor posibilidad de correlación entre nuestros resultados, y tras el breve análisis llevado a cabo en el apartado anterior, vemos que la opción más viable y óptima para dar solución a nuestro problema reside en el algoritmo Double Deep Q-Learning, de esta manera prevenimos el problema de sobreestimación y nos aseguramos un mayor rendimiento respecto al resto de algoritmos.

# 7 Descripción de la solución propuesta.

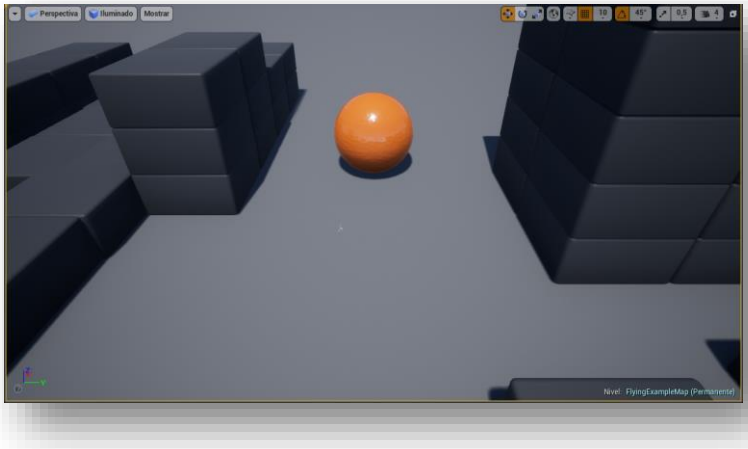
Una vez seleccionadas las alternativas para poder elaborar este estudio, se procede a explicar cuál ha sido el procedimiento experimental haciendo uso de estas. Como ya se ha mencionado en apartados anteriores, el objetivo de este trabajo es conseguir el desplazamiento autónomo de un vehículo no tripulado dentro de un entorno simulado, evitando los obstáculos establecidos a lo largo de su trayectoria, la cual variará entre los 20 y 30 metros. Para ello se han realizado pruebas en dos escenarios distintos, cada uno compuesto por diversos obstáculos. Los puntos que se han seguido en cada uno de estas pruebas han sido los siguientes:

- Construcción del escenario.
- Desarrollo del algoritmo.

El primer punto se ha realizado haciendo uso del entorno virtual Airsim, el cual proporciona un escenario principal de prueba, donde se pueden añadir diversos elementos como cubos, cilindros, esferas, etc. Este entorno ha sido instalado siguiendo los pasos indicados en la documentación oficial de Airsim [10]. En uno de los puntos iniciales se indica la necesidad instalar la herramienta Visual Studio, la cual será utilizada para la codificación del algoritmo en Python.

## 7.1 Primer escenario: Pista de cilindros.

Una vez instalado Airsim, se procede a editar el escenario de prueba, para ello se debe ejecutar el archivo *blocks.uproject* ubicado en la ruta *Unreal\Environments\Blocks.* Esto abrirá directamente la aplicación Unreal con el escenario inicial.



### 7.1.1 Construcción del escenario

En este primer estudio, el agente deberá ser capaz de alcanzar su destino situado a 30 metros, driblando los cilindros que se encuentre a lo largo de su trayecto. Para ello se diseña un

escenario que consta de múltiples cilindros de diversos colores, repartidos de forma regular a lo largo de la trayectoria a seguir por parte del Dron. Es importante que estos cilindros sean de diferentes colores, de esta manera se le proporciona el mismo obstáculo, pero con diversas características. Esto ayuda a que el agente aprenda a distinguir mejor la silueta, dando menos importancia al color y más a su forma. Los cilindros estarán situados de tal manera que, la distancia entre ellos sea aproximadamente de un 1.5 metros, dejando suficiente espacio para que el vehículo pueda maniobrar correctamente.

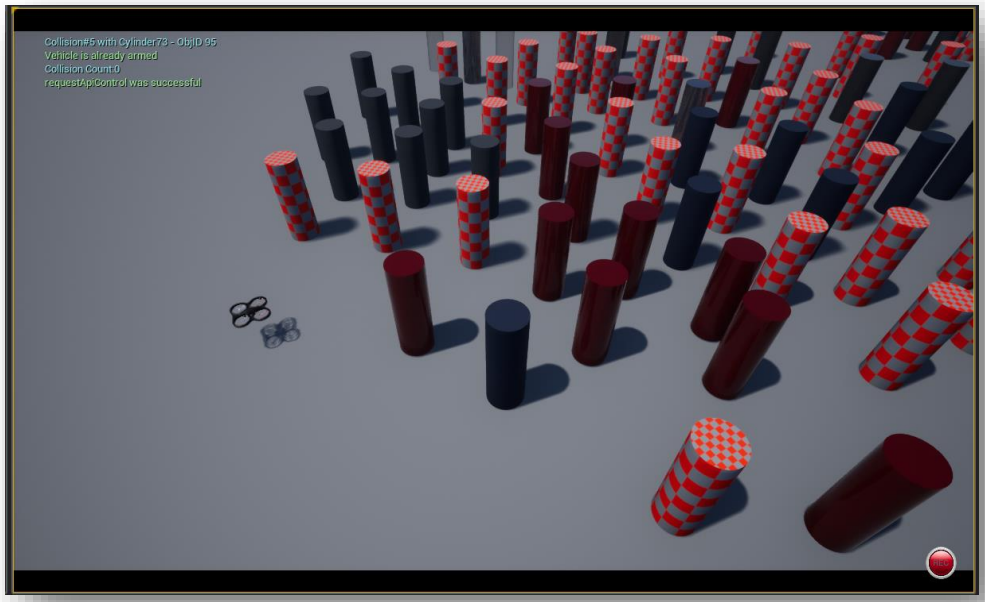



**Ilustración 5:** Pista de cilindros.

Además de este escenario, se construye otro similar, pero con una distribución diferente de los obstáculos. Este segundo escenario será utilizado para validar el entrenamiento del agente. Es útil realizar este tipo de validación, para garantizar que el agente ha aprendido a esquivar cilindros en cualquier escenario, y no a resolver únicamente el circuito donde ha sido entrenado.

### 7.1.2 Desarrollo del algoritmo

Una vez construido el entorno, se da paso a la codificación del algoritmo que determinara los movimientos del dron dentro del escenario. Para ello, se parte de un tutorial sobre Reinforcement Learning disponible en la página oficial de Airsim [10]. Este ejemplo consiste en el movimiento autónomo de un dron para el seguimiento de líneas de tensión eléctrica, a su vez dispone de otros ejemplos donde se explica cómo utilizar las librerías de para el control de los vehículos [11].

En primer lugar, es necesario establecer una conexión con el simulador Airsim, para obtener las imágenes capturadas por el dron durante el entrenamiento. En cada paso dado por el agente, se capturan 4 imágenes desde la cámara frontal, las cuales serán procesadas a través de redes convolucionales y redes neuronales para la obtención de los valores Q para cada acción posible.

Retomando los conceptos sobre los que se centran los algoritmos de aprendizaje por refuerzo, y en concreto DDQN, se establecen los principales puntos a tener en cuenta:

- **Política para la selección de acciones**: define el método que seguirá el agente para la selección de acciones, y las acciones que pueda llevar a cabo.
- **Función de recompensa:** define el método con el que procederá a calcular la recompensa obtenida en función del estado en el que se encuentre.
- **Algoritmo de aprendizaje**: Se explica el algoritmo utilizado, así como las diversas pruebas realizadas para la elección de alguno de sus parámetros.

A continuación, se detalla cómo se han llevado a cabo estos puntos principales.

#### 7.1.2.1 Política para la selección de acciones:

Una vez procesadas las imágenes y obtenido un estado el cual pueda ser interpretado por la red neuronal, será necesario establecer una política que establezca el procedimiento a llevar a cabo por parte del agente, para la elección de acciones a tomar en cada estado por el que pase. Tal y como se ha explicado en apartados anteriores, este proceso requiere de un periodo de exploración, que a su vez debe ser combinado con otro periodo de explotación. En este estudio, se ha decidido alternar estos dos procesos mediante el método linear e-greedy. Este método hace que el agente seleccione una acción en función de un parámetro épsilon, el cual tomará valores entre 0 y 1. Esta variable es inicializada en 1 y se irá decrementando su valor en cada paso que el agente tome. En el periodo de entrenamiento se elegirá un número aleatorio, de tal forma que si este es inferior a épsilon se procederá a explorar, y en caso contrario se realizará el proceso de explotación. Esto permite que durante el inicio del entrenamiento el agente esté explorando y conociendo el entorno, y a medida que este avance, el valor de épsilon irá disminuyendo, permitiendo así que el agente obtenga una mayor posibilidad de seleccionar una acción en base al conocimiento ya adquirido hasta ese momento.

Una de las cuestiones a resolver reside en la velocidad con la que épsilon irá disminuyendo, puesto que, si este se reduce rápidamente, dificultaría el aprendizaje del agente al no obtener suficiente información acerca de su entorno, debido al escaso tiempo que ha tenido para ello. Por otra parte, si épsilon decrece lentamente puede ocasionar un retardo en el aprendizaje o incluso hacer que el agente adquiera un conocimiento erróneo. En este caso se ha decidido actualizar épsilon de la siguiente manera:

```
e_start = 1
e_end = 0.1
total_steps = 10000
step_size = (e_start - e_end)/total_steps
épsilon = 1 - step_size*step
```

Se estima que el total de pasos que dará el agente hasta alcanzar un conocimiento estable sobre el escenario en el que se encuentra, serán aproximadamente 10000. Una vez alcanzado este número de pasos, el valor de épsilon se mantendrá constante en 0.1 para el resto del entrenamiento. A partir de este instante el modelo hará un gran uso del conocimiento obtenido hasta ese momento, dejando un 10% de probabilidad para la toma de una acción aleatoria. Es importante dejar este margen de probabilidad, para que el agente tenga la opción de seguir descubriendo nuevos estados que le puedan ofrecer una mayor recompensa.

A continuación, se procede a explicar cómo se ha implementado esta teoría dentro del código. Para ello, en vez de adjuntar imágenes del código, se intenta reflejar el funcionamiento de cada método definido a través de esquemas que resulten más compresivos.
El primer método definido es “acción”, el cual retornara un numero entero que represente la acción a ejecutar por parte del agente en base a la política e-greedy. El funcionamiento que seguirá este método “acción” se ve reflejado en la siguiente imagen:

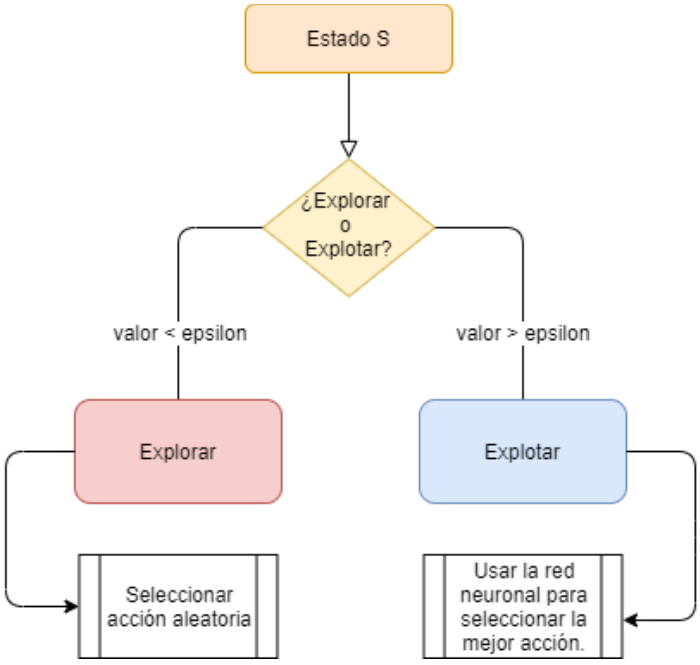



**Figura 4:** método "Acción" con política e-greedy.

Como se puede observar, para cada estado, el agente estará en un proceso u otro en función de épsilon. En el caso de que este se encuentre en el proceso de explorar, retornara un valor aleatorio perteneciente al rango del número de acciones totales. Por otra parte, en el caso de que este se encuentre en el proceso de exploración, se hará uso de la red neuronal para obtener, dentro de un array, los valores Q de cada acción en ese estado, retornando el máximo de estos.

Una vez establecida la política que se va a seguir para la selección de acciones, se procede a enumerar y explicar las acciones que el dron podrá llevar a cabo.

- 0: desplazamiento recto.
- 1: desplazamiento izquierdo.
- 2: desplazamiento derecho.

En este caso, la función "acción" devolverá valores entre 0 y 2 para representar los movimientos. A la hora de decidir qué acciones puede tomar el dron, se han descartado algunos movimientos tales como los ascendentes o descendentes, así como la marcha atrás, con el fin de facilitar el entrenamiento. De esta manera se disminuye el número de acciones a tomar por parte del agente, ya que estos movimientos no aportan ningún beneficio en cuanto al aprendizaje, e incluso pueden llevar a tomar caminos no deseados. Sin embargo, tras realizar unas primeras pruebas para analizar el comportamiento del dron, se ha observado que este tiende a realizar algunos de estos movimientos involuntariamente, siendo así necesario un control sobre ellos.

Para que el dron pueda ejecutar estas acciones, es necesario crear un método capaz de interpretarlas y retornar la información necesaria para que puedan ser ejecutadas. Este método será: "InterpretaAccion".
Por una parte, gracias a la documentación de Airsim, se sabe que el dron ejecuta los movimientos de desplazamiento a través del método *moveByvelocityAsync()* de la clase *MultirotorCliente*. Este método tiene varios parámetros de entrada, de los cuales tres se usan para representar el movimiento a través de un vector de velocidades (vx, vy, vz), y un cuarto para indicar el tiempo de ejecución del movimiento. Por lo tanto, nuestra función "InterpretaAccion" tiene como objetivo adaptar los valores 0, 1 y 2 en un vector de velocidades que simule el movimiento correspondiente a dicho valor. El resultado es el siguiente:

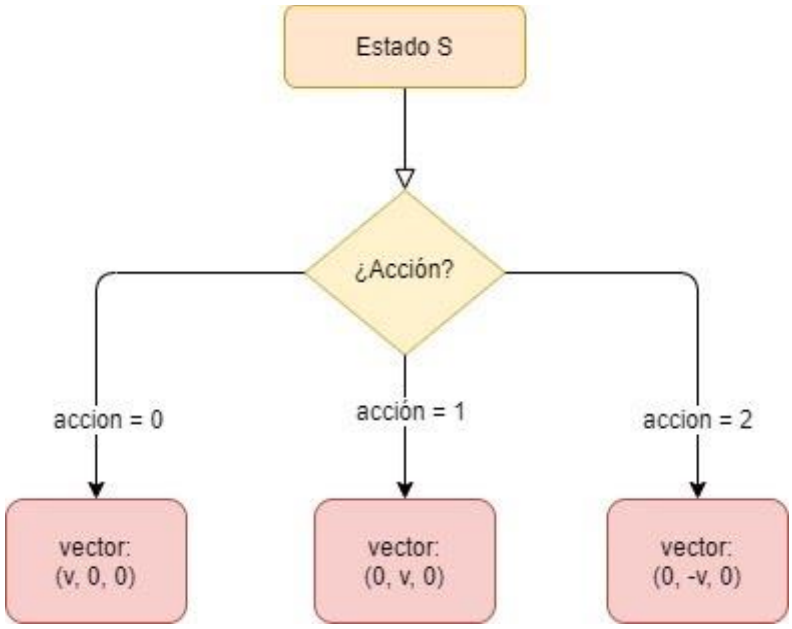



**Figura 5:** método "Interpreta Acción".

El parámetro "v "representa la velocidad que tomará el dron en el eje que se le indique, en este caso tendrá un valor de 0.4 metros/segundo.

#### 7.1.2.2 Función recompensa:

Tras definir la política que se usara para la selección de acciones, se procede a explicar la función de recompensa que marca el objetivo a completar. Esta función define los valores de recompensa que obtendrá el agente en base a las acciones que tome y los estados por los que pase. Es intuitivo pensar que los estados favorables darán recompensas positivas y los estados indeseados recompensas negativas, sin embargo, el impacto de estas dentro del aprendizaje no reside solo en el signo, sino que también es necesario establecer el peso adecuado para que el agente pueda identificar correctamente los estados erróneos. En este primer estudio se han identificado los estados por los que puede pasar el agente, y en función de su repercusión, se le otorgara un peso más o menos significativo a la recompensa. Los estados han sido clasificados en no terminales y terminales, siendo estos últimos los estados donde se finaliza el episodio actual dando comienzo a uno nuevo. La condición establecida para identificar mediante código, si un estado es terminal o no, reside en el signo de su recompensa, siendo terminales los estados que obtienen una recompensa negativa, y los no terminales una recompensa positiva. Los estados y recompensas se definen a continuación:

#### *7.1.2.2.1 Estado 1: En movimiento.*

Este es un estado no-terminal que representa todos los estados en los cuales el dron se desplaza en cualquiera de los 3 ejes (x, y, z) sin llegar a colisionar. Como uno de los objetivos marcados consiste en hacer que el agente se desplace desde un punto inicial hasta un punto destino, la recompensa otorgada por estos estados deberá representar el camino hacia este objetivo. Es por esto, que se ha decidido expresar este valor en función de la distancia restante hasta el punto destino, es decir, cuanto mayor sea esta distancia menor será la recompensa y viceversa. La función adquiere la siguiente forma:

$$recompensa = \frac{Distancia_{destino-}\ Distancia_{actual}}{Distancia_{total}} \quad (7)$$

Como podemos ver, los valores que obtendrá la recompensa estarán en el rango de 0 a 1.

#### *7.1.2.2.2 Estado 2: Colisión.*

El agente entra en este estado cuando detecta una colisión con un obstáculo, con lo cual este estado ha sido clasificado como terminal y dará inicio a un nuevo episodio. Este es uno de los estados no deseados, por lo tanto, la recompensa lo debe reflejar con un valor inferior al mínimo obtenido en el caso anterior. Como la recompensa en los estados en movimiento toma valores de 0 a 1, y el aprendizaje del modelo depende en gran parte de la recompensa acumulada, se debe asignar un valor lo suficientemente alto que pueda contrarrestar el valor de la recompensa acumulada hasta el momento de la colisión. Se estima que a distancias próximas del punto destino, el agente puede llegar a acumular una recompensa de 20-30, es por esto que se ha optado por dar un valor de -100, que identifique claramente que se trata de un estado perjudicial.

#### *7.1.2.2.3 Estado 3: Llegada al destino.*

Este representa el lado opuesto del anterior, con lo cual es similar en cuanto a concepto, pero diferente en cuanto al valor de la recompensa obtenido. Este estado debe reflejar el cumplimiento del objetivo con una recompensa amplia, con lo cual se ha decidido que la recompensa sea de 100.

Por otra parte, tal y como se ha comentado previamente, a pesar de que el agente está programado para tomar 3 acciones lo que deriva en 3 movimientos, el dron tiende a realizar ciertos desplazamientos involuntariamente. Uno de los casos más relevantes surge cuando el dron se desplaza continuamente de izquierda a derecha. Estos movimientos hacen que el dron se balance y poco a poco vaya adquiriendo altura, de tal manera que simula la ejecución de un movimiento ascendente. Este movimiento no es de interés para este primer análisis, puesto que, si el dron logra alcanzar una altura superior a la de los cilindros, este no encontraría obstáculos en su camino, y aprendería que la mejor opción a tomar es el desplazamiento en línea recta.

Para dar solución a estos inconvenientes, se decide definir 2 estados terminales que finalicen el episodio en el caso de entrar en estos estados. Aunque la probabilidad de entrar en ellos es mínima, el periodo de entrenamiento es largo y las pruebas iniciales realizadas han demostrado que es conveniente anular estos casos, ya que dañan considerablemente los resultados. Los estados terminales son los siguientes:

#### *7.1.2.2.4 Estado 4: altura superada.*

Este estado se alcanza cuando se supera el límite máximo de altura establecido. Para dar valor a este límite se ha tenido en cuenta la altura de los cilindros, la cual es de aproximadamente 2 metros. El límite establecido ha sido de 1.9 metros y si este es superado se obtiene una recompensa de -1. Este valor se debe a que el movimiento es involuntario y lo único que se desea conseguir, es cumplir con la condición para que el estado sea considerado como terminal e inicie un nuevo episodio, sin influir en el valor de recompensa acumulada.

#### *7.1.2.2.5 Estado 5: límite inferior superado.*

Este estado ha sido añadido como método de precaución, ya que se ha observado que el dron tiende a disminuir su altura muy lentamente debido a los movimientos laterales ejecutados durante un gran periodo. El límite establecido ha sido de 0.1 metros, ya que el dron empieza el entrenamiento a una altura de 1 metro aproximadamente. Al igual que en el caso anterior, el valor de recompensa obtenido es -1, para obtener el mismo resultado que se ha explicado antes.

### 7.1.2.3 Algoritmo de aprendizaje

Para el desarrollo del algoritmo, se parte de un ejemplo proporcionado en el tutorial de Airsim. Este algoritmo se basa en el estudio *Human-level control through deep reinforcement learning*" [12], donde se explica el uso de un algoritmo DQN para resolver juegos de Atari. Sin embargo, el algoritmo que proporciona Airsim agrega una segunda red neuronal para el cálculo de los valores Q target, convirtiéndolo así en un algoritmo DDQN. Como ya se ha explicado en el estado del arte, los parámetros más importantes que rigen el comportamiento de este algoritmo son Alpha y gamma. El objetivo de este apartado consiste en averiguar los valores de estas variables, que mejor se adapten a nuestro estudio para conseguir un aprendizaje adecuado. A continuación, se procede a explicar los valores que se ha empleado para estas variables y el motivo.

**Alpha o ratio de aprendizaje:**

En el caso de DDQN, la variable Alpha solo se tiene en cuenta para la optimización de la función de pérdidas, en la cual se hace uso del optimizador Adam [13]. Este parámetro tiene como objetivo decidir si dar más importancia al nuevo aprendizaje adquirido o al que ya se conoce hasta ese momento. En este primer escenario, todos los puntos por los que puede pasar el

agente son similares, con lo cual el conocimiento que adquiera en los puntos iniciales de la prueba, será útil cuando logre avanzar hasta los puntos finales. Por lo tanto, se decide dar un valor cercano a 0 que dé prioridad al conocimiento ya adquirido.

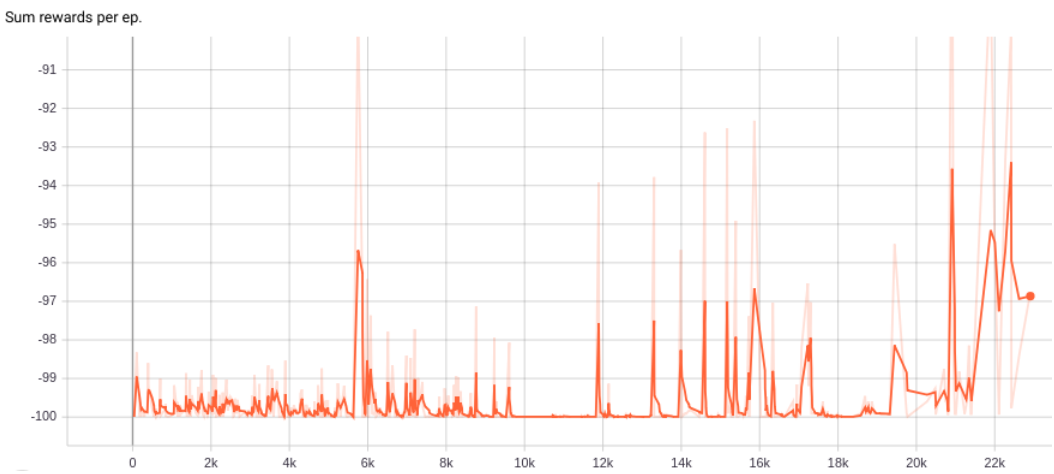
**Gamma o factor de descuento:**

La variable gamma, dentro del algoritmo DDQN, es usado para el cálculo del parámetro Q target y tiene como objetivo dar mayor o menor importancia a las recompensas futuras respecto a las inmediatas. Un valor cercano a 0 dará más importancia a las inmediatas y un valor cercano a 1 a las futuras. Teóricamente se puede deducir que un mayor peso sobre las recompensas futuras puede resultar más útil, sin embargo, las pruebas realizadas han demostrado lo contrario. Para averiguar el valor óptimo de este parámetro, se han realizado varias pruebas dando diferentes valores a gamma. Estos valores han sido 0.2, 0.5, 0.7 y 0.9.

Las siguientes gráficas representan el valor de la recompensa acumulada, para los diferentes valores de gamma, que obtiene el agente a medida que avanza el entrenamiento. Los gráficos se encuentran suavizados, de tal manera que se pueda apreciar mejor la variación de la recompensa acumulada. Es por esto que, pese a obtener episodios donde se consigue completar el objetivo y la recompensa obtenida debería ser 100, los valores de recompensa máxima que se ven reflejados están comprendidos en un rango entre -30 y -100.

- **Gamma: 0.2**

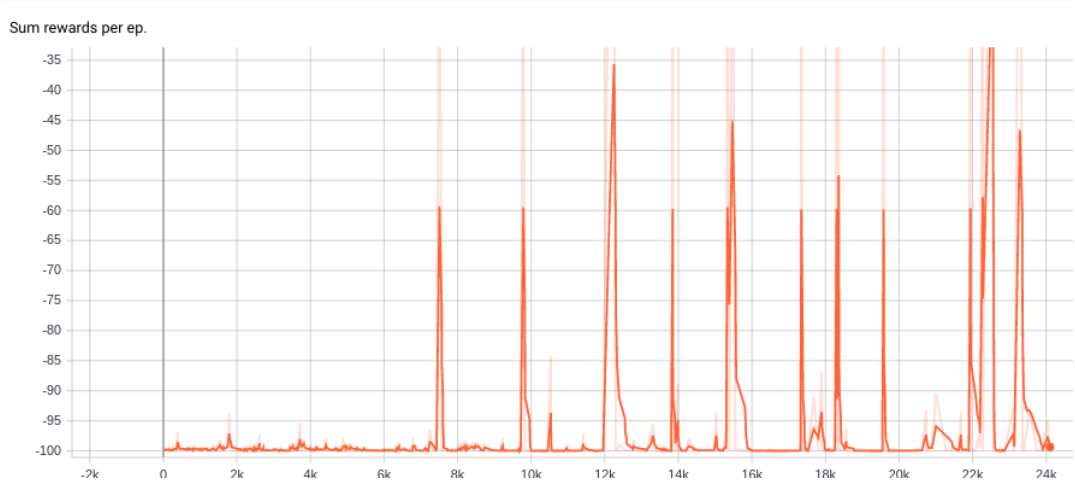
  En la gráfica se puede observar que a pesar que existe un pico en el paso 6000, el agente empieza a adquirir conocimiento a partir de los 12000 pasos, donde la puntuación adquirida se acerca a un valor próximo de -97. Sin embargo, es a partir del paso 22000, donde se obtienen las mejores puntuaciones, llegando a un valor aproximado de -93.




**Gráfica 1:** Cilindros, resultado para gamma = 0.2

- **Gamma: 0.5**

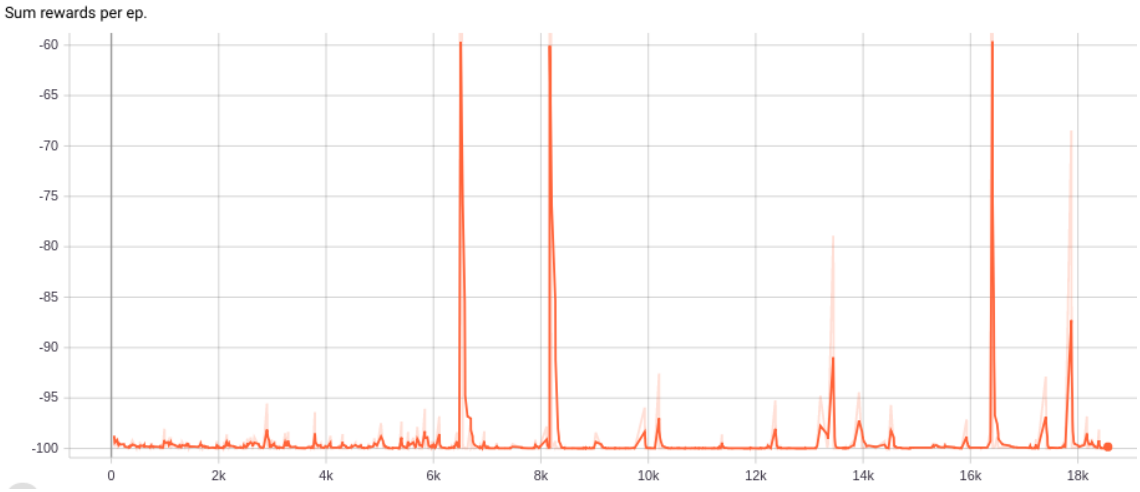
  En este caso, se puede asegurar que el conocimiento se adquiere aproximadamente a partir del paso 7000, ya que en este caso si se mantiene constante la regularidad con la que el agente avanza para el resto de entrenamiento. Además, los valores obtenidos para la recompensa acumulada llegan a superar el valor de -35.




**Gráfica 2:** Cilindros, resultados para gamma = 0.5.

- **Gamma: 0.9**

  Para esta última gráfica, se puede observar que el aprendizaje ha resultado casi nulo, teniendo unos leves picos, pero sin llegar a mantenerse constantes.




**Gráfica 3:** Cilindros, resultados para gamma = 0.9.

## 7.2 Segundo escenario: Laberinto.

Tras ver el éxito del algoritmo en el primer escenario, se decidió probar dicho modelo en un segundo escenario, en el que los obstáculos son diferentes pero el objetivo sigue siendo el mismo. Este segundo escenario representa un laberinto, donde el agente debe ser capaz de evitar colisionar con las paredes y llegar a su destino.

### 7.2.1 Construcción del escenario

Al igual que en el caso anterior, se han diseñado dos laberintos dentro del entorno Unreal, uno para realizar el entrenamiento y otro para evaluar el aprendizaje. Para la creación de estos, se han utilizado cubos de una altura aproximada de 4 metros. El resultado es el siguiente:

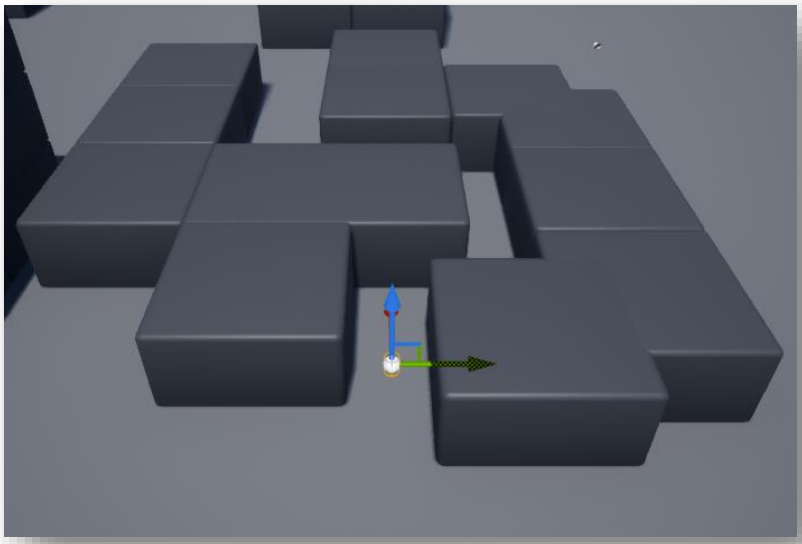


**Ilustración 6:** Laberinto de entrenamiento.

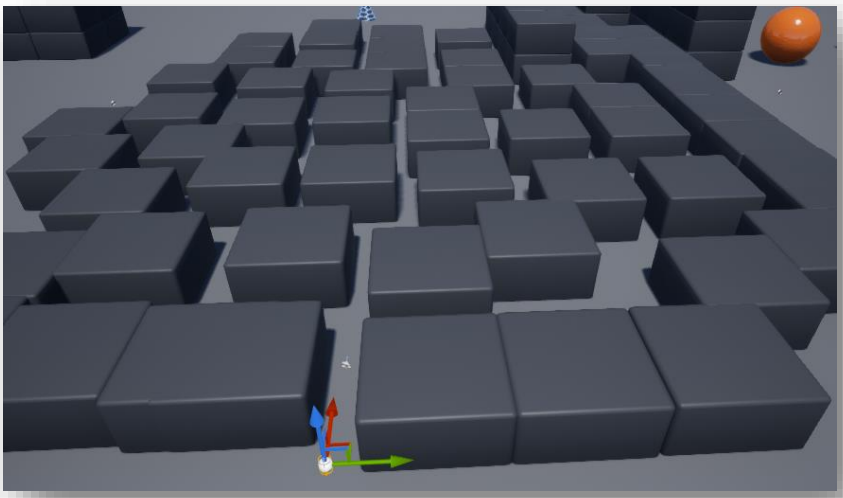


**Ilustración 7:** Laberinto de evaluación.

El laberinto de entrenamiento es más simple que el de evaluación, debido que se pretende entrenar al agente en un entorno sencillo, para así luego poder evaluar como de bueno es el conocimiento adquirido en escenario mucho más complejo.

### 7.2.2 Desarrollo del algoritmo

En primer lugar, se han realizado pruebas con el modelo obtenido en el primer escenario, con el fin de averiguar cómo de útil podría llegar a ser el conocimiento ya adquirido. Los resultados

de estas pruebas no son del todo satisfactorios, debido a que en algunos casos el agente colisiona contra las paredes, y solo en este momento es cuando realiza un desplazamiento lateral que le permita evitar el obstáculo, en otros muchos casos, el agente decide tomar como única alternativa el desplazamiento rectilíneo. Se llegó a la conclusión de que esto se debía a la interpretación que daba el agente sobre la pared, al no distinguir ninguna silueta, el agente interpreta este obstáculo como si de un fondo se tratase, es decir, no observaba obstáculo alguno y por ello elegía un movimiento rectilíneo. Por lo tanto, es necesario realizar alguna modificación en el algoritmo para que el agente pueda resolver este nuevo problema. Nuevamente se explicará la política de selección de acciones, función recompensa y el algoritmo de aprendizaje que se han llevado a cabo en este nuevo escenario.

#### 7.2.2.1 Política para la selección de acciones:

Para este nuevo caso, se ha decido mantener la política de selección de acciones que se ha utilizado en el primer escenario. Sin embargo, como se ha comentado en el apartado anterior existe un nuevo problema a abordar. Para dar solución a este nuevo problema se decide cambiar las acciones que pueda tomar el agente dentro de este escenario. En el caso anterior, el agente se desplazaba horizontalmente y hacia delante, manteniendo la cámara que se encarga de captar las imágenes apuntando en todo momento a la parte frontal. Se deduce que el nuevo problema reside en estos movimientos, debido a que en ningún momento se realiza un cambio de plano para que el agente pueda observar nuevas alternativas. Es por esto que se opta por unas acciones, las cuales no determinen únicamente el desplazamiento, sino que también tengan en cuenta la dirección a donde apunte la cámara. Para ello será necesario que el dron realice giros sobre su propio eje, consiguiendo rotar 90 grados hacia la izquierda o hacia la derecha, en función del desplazamiento que se quiera realizar. Las nuevas acciones son las siguientes:

- 0: Giro de 0 grados y desplazamiento rectilíneo.
- 1: Giro de 90 grados y desplazamiento hacia la izquierda.
- 2: Giro de -90 grados y desplazamiento hacia la derecha.

En este caso, como además de realizar el desplazamiento se requiere de un cambio de plano, es necesario calcular el vector de velocidades en función del plano en el que nos encontremos. Por lo tanto, la única diferencia en cuanto a la función “InterpretaAccion” del caso anterior, reside en el cálculo del vector de velocidades, el cual se resuelve haciendo uso de una matriz de rotación.

$$(v_x', v_y') = (v_x, v_y) * \begin{pmatrix} \cos(\alpha) & sen(\alpha) \\ -sen(\alpha) & \cos(\alpha) \end{pmatrix} \quad (8)$$

Al igual que en el caso anterior, se hace uso de moveByvelocityAsync() para efectuar los movimientos. En este caso también es necesario utilizar el método moveByAngleZAsync() para efectuar el movimiento de rotación.

#### 7.2.2.2 Función recompensa

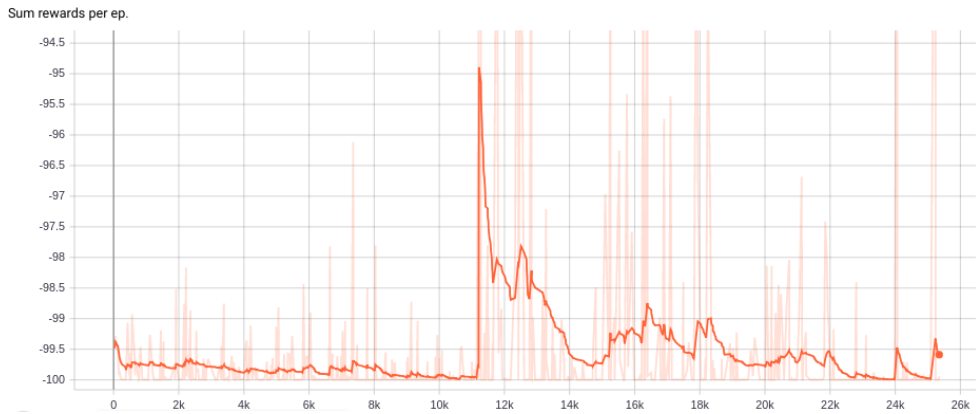
Como el objetivo sigue siendo llegar a un punto destino partiendo desde un punto origen, se decide mantener el procedimiento para el cálculo de las recompensas, teniendo en cuenta la distancia restante hasta el punto destino. Los estados por los cuales puede pasar el dron siguen siendo los mismos, con lo cual se mantiene los respectivos valores para los estados terminales.

#### 7.2.2.3 Algoritmo de aprendizaje

Al igual que en el caso anterior se utiliza el mismo algoritmo DDQN para entrenar el modelo. Nuevamente se considera que Alpha debe adquirir un valor cercano a 0, y se realizan diferentes pruebas para encontrar el valor adecuado para la variable gamma. En este caso los valores de recompensa máxima que se ven reflejados están comprendidos en un rango entre -100 y -70. A diferencia de las gráficas anteriores, estas tienden a ser más continuas. Esto se debe a las características del nuevo escenario, ya que el agente cuenta con un espacio más abierto para realizar el recorrido, haciendo menos frecuentes las colisiones y manteniendo el valor de las recompensas adquiridas durante más tiempo.

- **Gamma: 0.2**

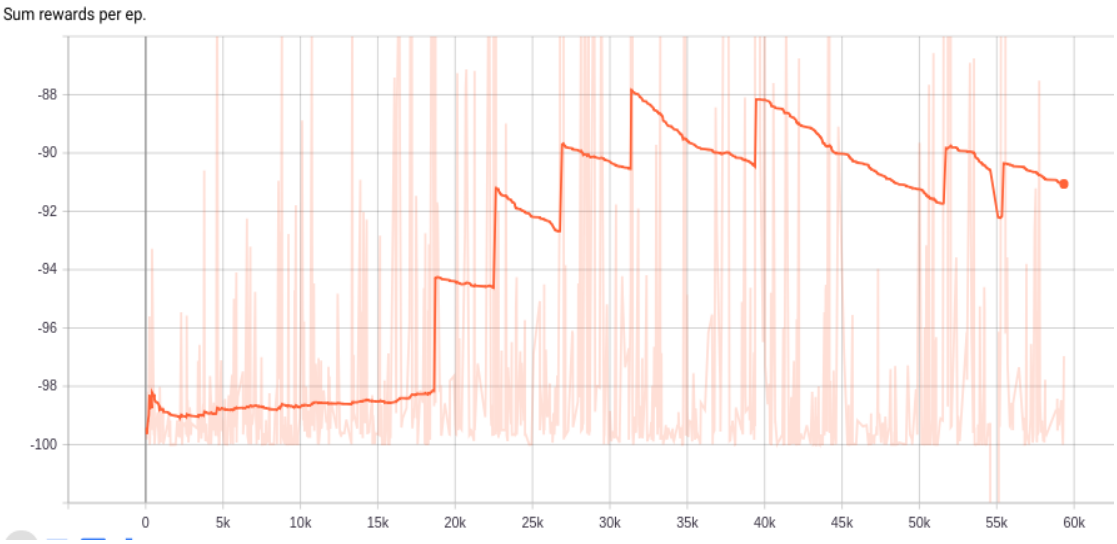
Para un valor de gamma igual a 0.2, llega a adquirir algo de conocimiento, pero no logra retenerlo a lo largo del entrenamiento.




**Gráfica 4:** Resultado para gamma = 0.2

- **Gamma: 0.5**

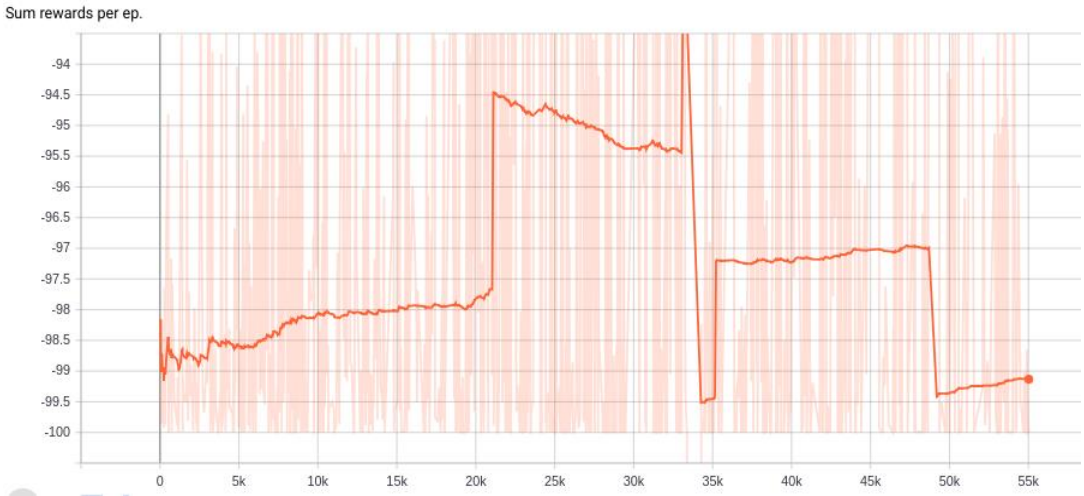
Para un valor de gamma igual a 0.5, el agente empieza a adquirir conocimiento una vez a completado una cantidad aproximada de 18000 pasos. A partir de este momento la recompensa acumulada va aumentando progresivamente hasta mantenerse constante.




**Gráfica 5:** Resultado para gamma = 0.5.

- **Gamma: 0.9**

Finalmente, vemos que para un valor de 0.9 el agente adquiere conocimiento al de 21000 pasos. Sin embargo, los valores no son muy altos y tiende a desaprender a medida que avanza.




**Gráfica 6:** Resultado para gamma = 0.9.

# 8 Análisis de resultados

Las gráficas obtenidas en los anteriores análisis, describen perfectamente el comportamiento realizado por parte del agente durante su entrenamiento. En ellas se puede ver el momento exacto en el que empieza a adquirir conocimiento, aumentando así la puntuación obtenida. Este incrementa representa el momento en el que el agente supera los primeros estados,

Todas las gráficas tienen unos valores iniciales de -100, representando así la fase inicial del entrenamiento, donde el agente se encuentra en el proceso de exploración, tomando acciones aleatorias. Al encontrarse rodeado de obstáculos y debido a la toma de decisiones inicial, las colisiones son más frecuentes adquiriendo así una puntuación de -100, sin embargo, a medida que el entrenamiento avanza, se da paso al proceso de explotación, y es en este momento donde se ve reflejado como de bueno ha sido el conocimiento adquirido en la fase inicial. Si el conocimiento adquirido es favorable, el agente obtiene una mayor recompensa, y como esta es proporcional a la distancia restante hasta el punto destino, se puede deducir que el agente es capaz de esquivar los obstáculos avanzando correctamente hacia su objetivo.

## 8.1 Análisis del primer escenario: pista de cilindros.

Las graficas obtenidas para este escenario, reflejan que para un valor de gamma igual a 0.9, el conocimiento adquirido por el agente es desfavorable, obteniendo unos leves picos, pero sin llegar a mantenerse constantes. Estos resultados pueden ser originados debido a una adquisición de conocimiento erróneo en la fase inicial, donde el agente aprende a no arriesgar en sus movimientos, desplazándose en el mismo punto de izquierda a derecha, obteniendo una recompensa aproximada de 0.01 por cada paso que da, y finalmente, debido al 10% de probabilidad otorgado a la continuidad del proceso de exploración, termine ejecutando acciones que ocasionen una colisión. Este conocimiento se propaga en los siguientes pasos gracias al valor de gamma de 0.9, dando así menos importancia a las recompensas inmediatas que pueda obtener al encontrar un camino más óptimo.

Por otra parte, para un valor de gamma de 0.2, se observa que el agente es capaz de llegar a aprender, pero sin obtener una gran puntuación. Para ello puede que requiera de más tiempo de entrenamiento, ya que en los pasos finales de la gráfica 1 se aprecia un mayor valor de recompensa.

Finalmente, en la gráfica 2, se observan los mejores resultados obtenidos para un valor de gamma igual a 0.5. En esta grafica se puede ver como el agente empieza a adquirir los primeros conocimientos a partir del paso 7000 aproximadamente, y a partir de ese instante, los episodios con una alta recompensa acumulada empiezan a ser más frecuentes. Además, la recompensa total obtenida, alcanza unos valores máximos de -35 (teniendo en cuenta el suavizado),

haciendo ver que para este valor de gamma el agente ha sido capaz de llegar a su destino, sorteando los obstáculos. El resultado obtenido se puede observar a través del link [19].

## 8.2 Análisis del segundo escenario: laberinto.

Analizando las gráficas 4, 5 y 6 se observa que, en este nuevo escenario, el aprendizaje requiere de un mayor tiempo de entrenamiento. En este caso, el agente empieza a adquirir conocimiento a partir de unos 20.000 pasos aproximadamente. Esto puede deberse al nuevo reparto de los obstáculos que, a diferencia del caso anterior, en este nuevo escenario el agente se mueve con mayor libertad y ocasionando menos colisiones. Por otra parte, el hecho de añadir rotación en los movimientos, añade un segundo más de retardo que en el primer escenario.

En este nuevo escenario, se mantiene el objetivo inicial de alcanzar un punto destino, intuitivamente se puede deducir que el valor de gamma para 0.5 iba a ser el más útil, sin embargo, al encontrarnos con nuevos obstáculos, era necesario volver a realizar las pruebas para comprobar que valor de gamma garantizaba un mayor aprendizaje.

Los resultados obtenidos en las graficas 4, 5 y 6, vuelven a confirmar que para un valor de gamma igual a 0.5 se obtienen los mejores resultados. Sin embargo, a la hora de evaluar este modelo en el escenario de evaluación, se comprueba que es capaz de alcanzar su objetivo, pero tiende a realizarlo con cierta dificultad. Como ya se ha dicho, en este nuevo escenario el agente tiene mayor espacio para maniobrar, con lo cual se ha comprobado que este tiende a acumular recompensa positiva desplazándose de izquierda a derecha, y finalmente optando por avanzar hacia delante. Es decir, como su objetivo es acumular mayor recompensa, primero realiza movimientos horizontales, sin llegar a colisionar, y finalmente tiende a avanzar. Por otra parte, se observa que en algunas situaciones el dron tiende a colisionar levemente, para luego optar la opción correcta para evitar seguir colisionando. Se intuye que esta situación se debe al nuevo movimiento de rotación introducido, ya que este introduce un retardo en la ejecución del código, de tal manera que, durante el entrenamiento, algunas colisiones no son detectadas por parte del agente. El comportamiento del agente se puede ver a través del link [20].

# 9 Planificación

En este apartado se procede a explicar la planificación que ha sido necesaria para la realización de este proyecto. En primer lugar, se define el ciclo de vida y posteriormente se procede a reflejar este proceso a través de un diagrama de Gantt.

## 9.1 Ciclo de vida

El ciclo de vida de un proyecto nos ayuda a definir el proceso y la metodología que se ha llevado a cabo para el desarrollo de este. Para ello, se han identificado 4 fases:
Gestión del proyecto, preparación del proyecto, desarrollo del proyecto, documentación y preparación del proyecto.

### 9.1.1 Gestión del proyecto.

Este apartado refleja la fase inicial donde se deben establecer, junto al director del proyecto, las condiciones y los objetivos a cumplir por parte del alumno. En esta primera fase se llevan a cabo varias reuniones, con el fin de asegurar que los retos marcados son viables y los recursos a los que se es accesible, son suficientes para conseguir los resultados esperados.

### 9.1.2 Preparación del proyecto

Una vez definidos los objetivos a completar, se procede a realizar un primer estudio sobre las diferentes alternativas que puedan dar solución a nuestro problema. Para ello se realiza una búsqueda de información sobre estudios, proyectos y publicaciones relacionadas con el proyecto. Una vez seleccionado el método con el que se pretende dar una solución, se procede a la adquisición del conocimiento previo necesario para el desarrollo de esta.

### 9.1.3 Desarrollo del proyecto.

Esta fase representa la parte práctica del proyecto, donde una vez adquiridos los conocimientos teóricos, se procede a la codificación de la solución. Al inicio de esta fase, será necesario dedicar una gran parte del tiempo a la familiarización con el entorno y el lenguaje de programación. Finalmente, se irán realizando diversas pruebas con el fin de obtener los mejores resultados.

### 9.1.4 Documentación del proyecto.

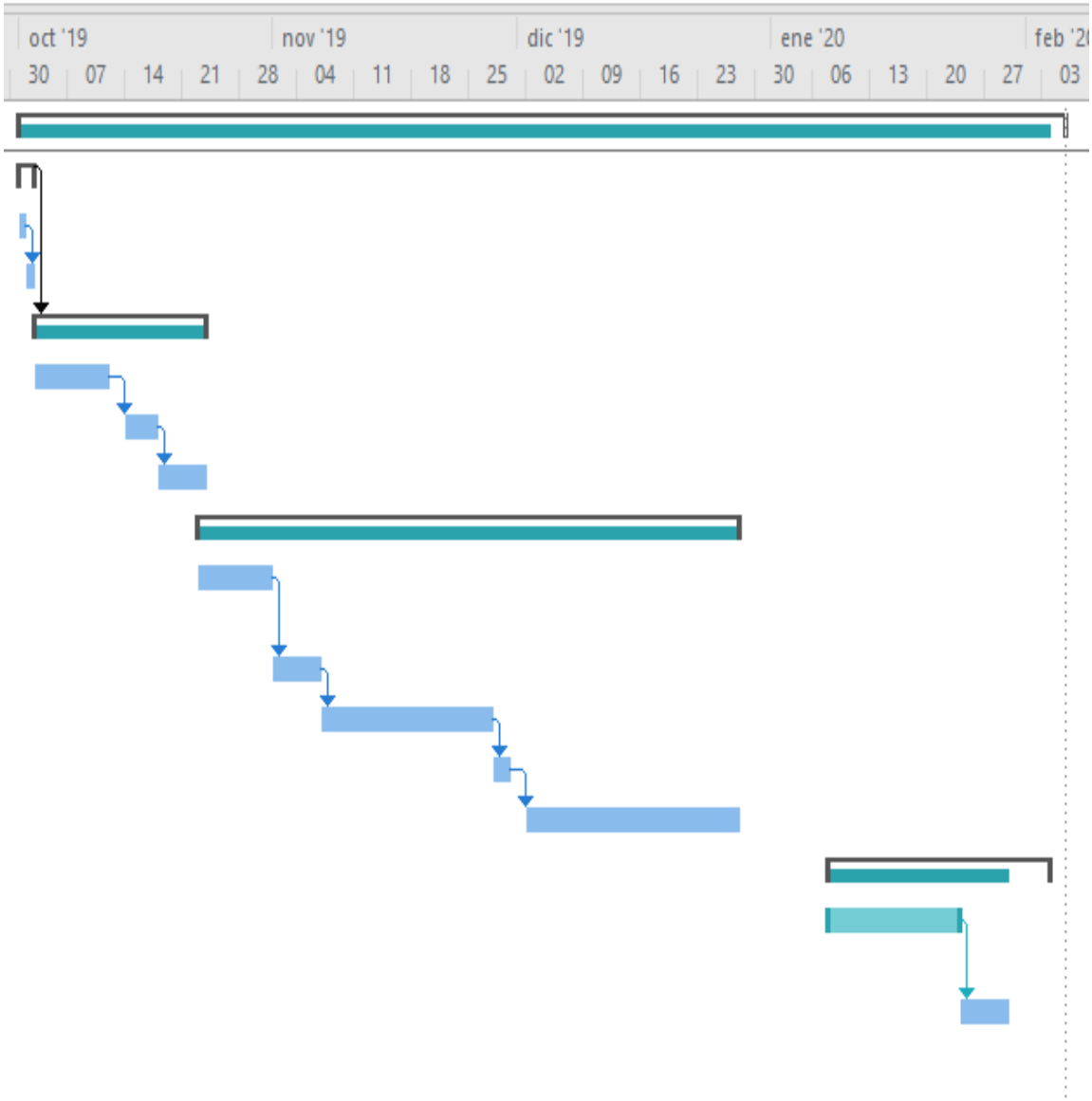
Una vez finalizada la parte práctica, se da paso al desarrollo de la documentación, donde se deben ver reflejado todo el trabajo realizado y los resultados obtenidos.

## 9.2 Diagrama de Gantt

Con el objetivo de obtener una representación más detallada de las fases expuestas en el apartado anterior, se procede a la elaboración de una tabla donde se ilustren las fases, las actividades que estas contienen y el tiempo de duración. Finalmente se elabora un diagrama de Gantt que representa el flujo de estas fases en el periodo de tiempo.

| Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|
| **Trabajo Fin de Grado** | **92 días** | **mar 01/10/19** | **mié 05/02/20** |
| **Gestión del proyecto** | **2 días** | **mar 01/10/19** | **mié 02/10/19** |
| Definición de objetivos. | 1 día | mar 01/10/19 | mar 01/10/19 |
| Definición de recursos HW y SW. | 1 día | mié 02/10/19 | mié 02/10/19 |
| **Preparación del proyecto** | **15 días** | **jue 03/10/19** | **mié 23/10/19** |
| Adquisición de conocimientos previos | 7 días | jue 03/10/19 | vie 11/10/19 |
| Análisis de alternativas | 4 días | lun 14/10/19 | jue 17/10/19 |
| Selección de alternativa y profundización en su marco teórico. | 4 días | vie 18/10/19 | mié 23/10/19 |
| **Desarrollo del proyecto** | **48 días** | **mié 23/10/19** | **vie 27/12/19** |
| Familiarización con el entorno de simulación y el lenguaje de programación | 7 días | mié 23/10/19 | jue 31/10/19 |
| Codificación del primer algoritmo | 4 días | vie 01/11/19 | mié 06/11/19 |
| Pruebas del algoritmo en el primer escenario | 15 días | jue 07/11/19 | mié 27/11/19 |
| Codificación del segundo algoritmo | 2 días | jue 28/11/19 | vie 29/11/19 |
| Pruebas en el segundo escenario | 20 días | lun 02/12/19 | vie 27/12/19 |
| **Fase Documentación** | **19 días** | **mié 08/01/20** | **lun 03/02/20** |
| Elaboración del documento que define el contexto del proyecto, objetivos, | 12 días | mié 08/01/20 | jue 23/01/20 |
| Elaboración de la presentación | 4 días | vie 24/01/20 | mié 29/01/20 |

**Tabla 2:** Tabla de tareas.




**Gráfica 6:** Diagrama de Gantt

# 10 Presupuesto y costes

En esta sección se procede a presentar los costes y recursos empleados durante el proyecto. En los cálculos se han tenido en cuenta el coste de recursos humanos y el coste de los recursos materiales, teniendo en cuenta tanto amortizaciones como consumibles.

## 10.1 Recursos humanos

Para la resolución de este proyecto han sido necesarios los servicios de un ingeniero técnico, para el desarrollo de la solución y documentación, y de su correspondiente director encargado de la coordinación. Los costes de cada uno vienen recogidos en la siguiente tabla:

| ID | Personal | Coste unitario (€/h) |
|---|---|---|
| P1 | Director del proyecto | 80 |
| P2 | Ingeniero Junior | 30 |

**Tabla 3:** Coste unitario del grupo de trabajo.

Siguiendo la duración de los paquetes de trabajo, mostrados en el Diagrama de Gantt, se calculan las horas correspondientes a cada grupo.

| ID | Horas | Coste unitario (€/h) | Coste total (€) |
|---|---|---|---|
| P1 | 50 | 80 | 1500 |
| P2 | 300 | 30 | 9000 |
| | | **Total** | **10500** |

**Tabla 4:** Coste total de recursos humanos.

## 10.2 Recursos materiales

### 10.2.1 Materiales amortizables

La siguiente tabla el coste de los materiales amortizables.

| Concepto | Coste de adquisición (€) | Vida útil (meses) | Tiempo de uso (meses) | Coste (€) |
|---|---|---|---|---|
| PC HP Z400 Workstation | 900 | 36 | 4 | 100 |
| | | | **Subtotal** | 100 |

**Tabla 5:** Coste total de los materiales amortizables.

### 10.2.2 Materiales fungibles

La siguiente tabla muestra el coste de los materiales fungibles utilizados:

| Concepto | Coste (€) |
|---|---|
| Material de oficina | 30 |
| Documentación | 30 |
| **Subtotal** | 60 |

**Tabla 6:** Coste total de materiales fungibles.

## 10.3 Coste total del proyecto

Finalmente se resume el coste total del proyecto, incluyendo los costes asociados a los recursos humanos y los costes por materiales.

| Concepto | Coste (€) |
|---|---|
| Recursos humanos | 10500 |
| Materiales amortizables | 100 |
| Documentación | 60 |
| **Total** | 10660 |

**Tabla 7:** Costes totales del proyecto.

# 11 Conclusiones

Este proyecto se inicio con el fin de obtener un sistema capaz de realizar un movimiento autónomo y que pueda ser utilizado para la elaboración de diferentes tareas, en diferentes escenarios. El principal objetivo se marcó en conseguir llegar a un destino, partiendo desde un origen, y a lo largo de este documento se ha ido explicando cual ha sido su evolución. Tras su ejecución, se han llegado a las siguientes conclusiones.

En primer lugar, después de analizar los resultados obtenidos, tanto para el escenario de cilindros como para el laberinto, se deduce que para un escenario donde el agente tiene mayor probabilidad de acceder a estados erróneos, como el de una colisión, el aprendizaje obtenido tiende a ser mayor. El hecho de proporcionar mayores ejemplos de estados indeseados, ayuda a que el agente puede aprender a distinguirlos con mayor facilidad. Por otra parte, como ya se ha comentado antes, los resultados evaluados en el segundo escenario no son del todo satisfactorios. Se tuvo en cuenta una variedad de posibilidades por las que esto podía suceder y como solucionarlo, desde cambiar la función de recompensa, teniendo en cuenta no solo la distancia hasta el destino, sino también incorporando una penalización por el número de pasos dados, incluso cambiado las acciones que el dron pueda ejecutar. El breve análisis elaborado en estas situaciones no otorgó mejores resultados, llegando a la conclusión de que proponer dos objetivos al agente (llegar a un destino y llegar con el menor número de pasos) puede ocasionar una confusión en su verdadero objetivo final. Para optimizar este resultado, sería necesario realizar varias pruebas teniendo en cuenta otros posibles motivos, como el tiempo de rotación del agente, ya que esto impide detectar ciertos casos de colisión.

Como conclusión final, tras la elaboración de este proyecto, se ha comprendido que pese a la complejidad y tiempo, sobre todo en el procesado de datos, que requiere la elaboración de este tipo de estudios, si las soluciones técnicas elegidas tanto para la definición de la función recompensa, como para la política de selección de acciones son las adecuadas, el conocimiento adquirido por parte del agente tendera a ser óptimo. Es por esto, que es fundamental hacer un previo análisis donde se estructuren bien los objetivos a cumplir, para así poder definir correctamente las bases del aprendizaje.

# 12 Bibliografía

[1] Hoy, Drones para vigilar infracciones durante la campaña de riego en la cuenca del Tajo, Julio 2018.

https://www.hoy.es/prov-caceres/drones-vigilar-campana-20180725105827-nt.html

[2] Revista DGT, Así es la vigilancia del tráfico con drones, Abril 2019.

http://revista.dgt.es/es/multimedia/infografia/2019/04ABRIL/0417-Drones.shtml#.XhNdvlUzbIU

[3] Stephen Burns, Drone meets delivery truck ,Febrero 22, 2017

https://www.ups.com/us/es/services/knowledge-center/article.page?kid=cd18bdc2

[3] Bianca-Cerasela-Zelia Blaga, Sergiu Nedevschi, Semantic Segmentation Learning for Autonomous UAVs using Simulators and Real Data, Septiembre 2019.

[4] Página oficial Gazebo.

http://gazebosim.org/

[5] Página oficial Jmavsim.

https://dev.px4.io/v1.9.0/en/simulation/jmavsim.html

[6] Página oficial SIM4CV.

https://sim4cv.org/

[7] Documentación sobre Airsim.

https://microsoft.github.io/AirSim/

[8] Teoría sobre la ecuación de Bellman.

https://es.wikipedia.org/wiki/Ecuaci%C3%B3n_de_Bellman

[9] Packtpub, *SARSA versus Q-learning – on-policy or off?*

https://subscription.packtpub.com/book/data/9781789345803/1/ch01lvl1sec13/sarsa-versus-q-learning-on-policy-or-off

[10] Reinforcement Learning in AirSim.

https://microsoft.github.io/AirSim/docs/reinforcement_learning/

[11] Repositorio PythonClient de Aisim.

https://github.com/microsoft/AirSim/tree/master/PythonClient/multirotor

[12] Nature 518. *"Human-level control through deep reinforcement learning"* (Mnih & al. 2015)

https://tugofweb.files.wordpress.com/2019/09/blaga_nedevschi_semantic_segmentation_drones.pdf

[13] Jason Brownlee, *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,* Julio 2017.

https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[14] Ruishan Liu & James Zou , *The Effects of Memory Replay in Reinforcement Learning.*

[15] Artur Banach, *"Visual control of the Parrot drone with OpenCV, Ros and Gazebo Simulator.",* Junio 2016.

http://repositorio.upct.es/bitstream/handle/10317/5442/pfc6362.pdf?sequence=1&isAllowed=y

[16] Imagen de JMAVSIM.

https://hangar.windhoverlabs.com/code/projects/PUB/repos/jmavsim/browse?at=aca28a4bf21

[17] Imagen de Sim4CV.

https://www.researchgate.net/figure/UAV-flying-in-Sim4CV-while-being-controlled-by-our-controller-fusion-network_fig1_332522619

[18] Imagen de Airsim.

https://www.elespanol.com/omicrono/software/20170216/simulador-drones-microsoft-demuestra-potencial-ensenar-ia/194231521_0.html

[20] Video evaluación del primer escenario: pista de cilindros.

https://studio.youtube.com/video/0efCUEaBapQ/edit

[19] Video evaluación del segundo escenario: laberinto.

https://www.youtube.com/watch?v=0g5YEMovzZs&feature=youtu.be