MÁSTER EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

PLATAFORMA PARA SIMULACIÓN DEL COMPORTAMIENTO DE UN DRON MEDIANTE ALGORITMOS DE APRENDIZAJE POR REFUERZO

Autor
Rodrigo de Lazcano Pérez Vicente

Director
Miguel Ángel Sanz Bobi

Madrid
Agosto 2020

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título Plataforma para simulación del comportamiento de un dron mediante algoritmos de aprendizaje por refuerzo en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2019/20 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada

de otros documentos está debidamente referenciada.

Fdo.: Rodrigo de Lazcano Pérez Vicente Fecha: 27/ 08/ 2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Miguel Angel Sanz Bobi Fecha: 27/ 08/ 2020

MÁSTER EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

PLATAFORMA PARA SIMULACIÓN DEL COMPORTAMIENTO DE UN DRON MEDIANTE ALGORITMOS DE APRENDIZAJE POR REFUERZO

Autor
Rodrigo de Lazcano Pérez Vicente

Director
Miguel Ángel Sanz Bobi

Madrid
Agosto 2020

PLATAFORMA PARA SIMULACIÓN DEL COMPORTAMIENTO DE UN DRON MEDIANTE ALGORITMOS DE APRENDIZAJE POR REFUERZO

Autor: **Pérez Vicente, Rodrigo de Lazcano**.
Director: Sanz Bobi, Miguel Ángel.
Entidad Colaboradora: ICAI –Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

El objetivo de este proyecto consiste en el desarrollo de una plataforma gráfica en la que se podrán implementar algoritmos de aprendizaje por refuerzo para facilitar la navegación autónoma de un dron simulado. Dicha plataforma es una primera aproximación para un sistema que permitirá el acceso cómodo a usuarios que busquen investigar sobre la navegación autónoma de drones aplicando técnicas de aprendizaje por refuerzo. En el contexto de este proyecto se desarrollaron y compararon los resultados de cuatro algoritmos de aprendizaje por refuerzo: Q-learning, Double Q-learning, Deep Q-learning y Double Deep Q-learning.

Palabras clave: UAV, aprendizaje por refuerzo, navegación autónoma, OpenAI Gym, ROS

1. Introducción

Los vehículos aéreos no tripulados, o drones,llevan siendo estudiados en el sector aeronáutico desde hace varios años, sin embargo, no ha sido hasta hace poco que se ha incrementado el auge de los drones en una gran diversidad de sectores de la industria. Tal es el caso, que Morgan Stanley estima que el valor de mercado de los drones ascienda a 1,5 trillones de dolares Americanos para el año 2040[**1**]. A medida que incrementa la demanda de los drones en diversos sectores industriales, también se está produciendo un gran avance en las diversas tecnologías que permiten a estos drones adaptarse a nuevas aplicaciones. Una de estas tendencias tecnológicas que está siendo investigada con mucha actividad es la navegación autónoma de los drones. La navegación autónoma de drones permitirá en un futuro que estas maquinas no requieran de ninguna supervisión humana. Gracias a ello se conseguirá que los drones sean capaces de navegar en entornos con alta densidad de obstáculos evitando el error humano de colisión. Además, al dotar a los drones de una autonomía completa se eliminará la dependencia del control de estos mediante señales externas que se pueden ver debilitadas en espacios cerrados o con muchos obstáculos de por medio que produzcan interferencias.

Por otro lado, para dotar a los drones de autonomía, actualmente se están aplicando e investigando técnicas de aprendizaje automático, en particular un área dentro de ellas llamada aprendizaje por refuerzo. El aprendizaje por refuerzo define dos componentes el agente, en este caso el dron, y por otro lado el entorno donde interactúa el agente. Este método de aprendizaje se asemeja a la manera en que los seres humanos aprenden, es decir mediante prueba y error, interactuando con el entorno hasta alcanzar cierto objetivo. Una de las grandes ventajas que permite este método de aprendizaje es la incorporación de simulaciones, evitando gastos económicos de material para pruebas reales. Además, comparado con los aprendizajes supervisado y no supervisado, el aprendizaje por refuerzo tiene la ventaja de requerir menos datos clasificados inicialmente y de ser un aprendizaje en tiempo real, es decir, el dron aprenderá a la vez que intenta o llega al objetivo.

De esta manera, en este proyecto se busca proporcionar contenido a las investigaciones de la navegación autónoma de drones creando una interfaz de usuario gráfica que permita entrenar a una simulación de un dron mediante aprendizaje por refuerzo. La tarea a aprender consistirá en crear una trayectoria con el dron hasta un objetivo, evadiendo los obstáculos en su camino. Este proyecto será una primera aproximación y se sugerirán nuevas lineas de trabajo para poder acercar esta plataforma a mas investigaciones relacionadas con drones y aprendizaje por refuerzo.

2. Objetivos

Una vez situado el contexto del proyecto, a continuación, se expondrán los objetivos que se persiguen en el proyecto realizado:

- Implementar en ROS y Gazebo la simulación del control de un dron y el entorno con el que interactuará en los entrenamientos de aprendizaje por refuerzo.
- Compatibilizar la simulación con la librería de Python, OpenAI Gym. Con esta librería se desarrollaran los algoritmos para entrenar al dron a ir de un punto inicial a otro objetivo en el mapa evitado los obstáculos. Los algoritmos de aprendizaje por refuerzo que se implementaran son: Q-learing, Double Q-learning, Deep Q-learning y Double Deep Q-learning.
- Diseñar una interfaz gráfica de usuario que permitirá interactuar con los distintos algoritmos, así como analizar los resultados de los entrenamientos. La aplicación se desarrollara tanto en Windows 10 como en Ubuntu para permitir un mayor acceso a mas investigadores o usuarios.

- Desarrollar en la interfaz gráfica de usuario una opción para comparar los distintos entrenamientos e inferencias que se han realizado con los distintos algoritmos de aprendizaje por refuerzo.

3. Descripción del sistema

Para el desarrollo de la plataforma se emplearon la herramientas software del Cuadro 1.

Componentes	**Versión**	**Funcionalidad**
Sistema Operativo	Ubuntu 18.04 (x64)	Sistema operativo en el que se desarrolla la aplicación
Python	2.7	Lenguaje de programación con el que se desarrolla la aplicación
ROS	Melodic	Midalware encargado de las comunicaciones de la plataforma. Simulación, Algoritmos e interfaz gráfica de usuario
Gazebo	9	Simulador de entornos virtuales en 3D. Mapas y dron
OpenAi Gym	0.16	Simplifica la aplicación de algoritmos de aprendizaje por refuerzo en entornos ya definidos
Keras	2.0.8	Desarrollo de redes neuronales
TensorFlow	1.4	Librería backend usada en Keras para los cálculos
Keras-RL	0.4.2	Integra las redes neuronales desarrolladas en Keras con OpenAI Gym
PyQt	5	Desarrollo de la interfaz gráfica de usuario

Cuadro 1: Componentes del sistema.

En primer lugar, mediante el sistema de nodos y lineas de mensajes de ROS se desarrollo la estructura de la plataforma. En la Figura 1 se observa esta estructura en la que las elipses representan los nodos y los rectángulos las lineas de mensajes o *"topics"*. Como se puede apreciar también, en la Figura 1 la estructura de ROS se puede seccionar en tres partes o nodos principales: Simulación, Algoritmos y *GUI* o interfaz gráfica de usuario. Cada una de estas partes realizarán una función de la plataforma y se se enviarán entre sí la información necesaria. A continuación, se explicarán brevemente la funcionalidad de cada una de las partes de la estructura

ROS.

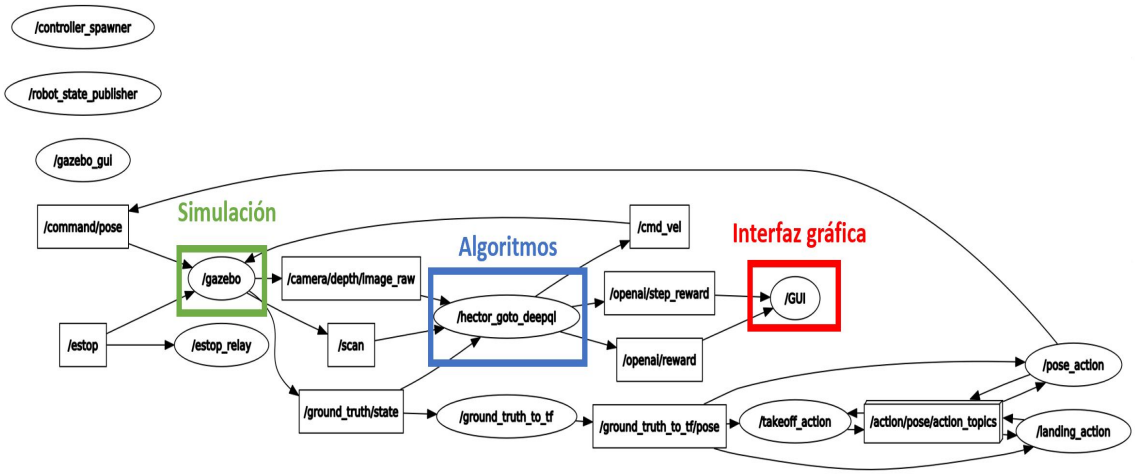


Figura 1: Gráfico de nodos del entorno en ROS

3.1. Simulación

La simulación del entorno se realizó en Gazebo. Por un lado, se utilizó la librería *Hector*, la cual proporciona la simulación de un dron con un control y física de vuelo muy realista. Además esta librería proporcionó los sensores utilizados para recoger la información del entorno por el dron[**2**]. Un sensor láser *Hokuyo UTM-30LX* y una cámara de profundidad *ASUS Xtion Pro*.

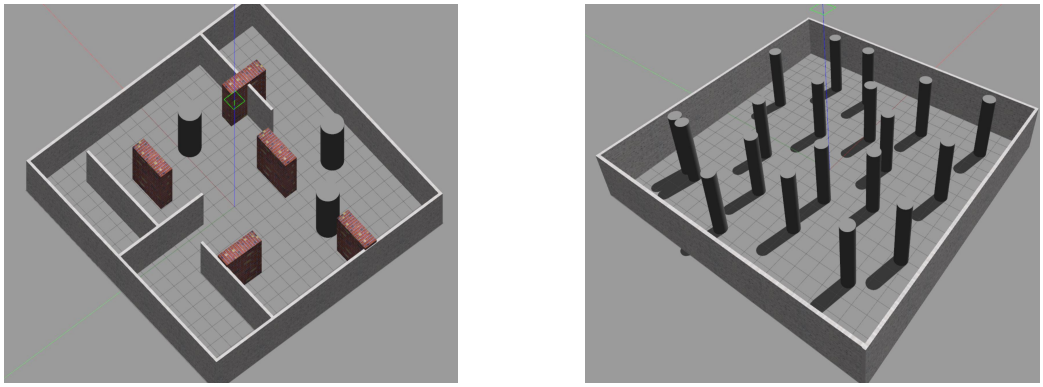
Por otro lado, se diseñaron dos mapas, 20x20 metros, para que el dron realizase los entrenamientos e inferencias de los algoritmos de aprendizaje por refuerzo. *Maze* y *Forest*. Para eliminar la parcialidad del entrenamiento en la inferencia, la tarea

(a) Cuadricóptero Hector

(b) Hokuyo UTM-30LX

(c) ASUS Xtion Pro

Figura 2: Elementos simulados del dron Hector



(a) Maze *(b) Forest*

Figura 3: Mapas diseñados en Gazebo

del dron se entreno en el mapa *Maze*, mientras que las inferencias en el mapa *Forest*. De esta manera, el dron se exponía a un entorno con nuevos obstáculos para comprobar si había aprendido eficientemente la tarea.

3.2. Algoritmos

Para entrenar al dron a ir de un punto de partida a otro objetivo evitando obstáculos se implementaron cuatro tipos de algoritmos de aprendizaje por refuerzo. Antes de implementar los algoritmos fue necesario integrar ROS con OpenAI Gym, lo cual se hizo con la estructura establecida por la librería *openai_ros* [**3**]. Estos algoritmos se dividen en dos grupos, algoritmos tabulares (*Q-learning* y *Double Q-learning*) y algoritmos de aprendizaje profundo (*Deep Q-learning* y *Deep Double Q-learning*). Ambos se diferencian en que en los algoritmos tabulares al función de valores de acciones *Q(s,a)* se define en una tabla con todas las combinaciones de estado/acciones posibles. Mientras que los algoritmos de aprendizaje profundo permiten expandir las dimensiones del problema realizando una aproximación funcional con redes neuronales de la función de valores de acción. Para este proyecto todos los algoritmos tienen en común la definición del espacio de acciones del dron, así como la función de asignación de recompensas. Sin embargo, el espacio de observaciones varía para los algoritmos tabulares y los algoritmos de aprendizaje profundo. El diseño del entorno de aprendizaje por refuerzo profundo fue inspirada por [**4**].

En primer lugar, el espacio de acciones del dron en el entorno consta de tres acciones discretas en el plano horizontal mostradas en el Cuadro 2. No se doto

al dron de acciones para el ascenso o descenso en el eje z ya que el espacio de acciones hubiese sido demasiado grande para la potencia computacional disponible. De esta manera, al inicio de cada episodio el dron ascendería a una distancia de aproximadamente 2 metros y se mantendría constante a lo largo del resto del episodio.

Cuadro 2: Espacio de acciones del dron

Acción	Valor
Hacia delante	1 m/s
Derecha	30°
Izquierda	30°

Seguidamente, la función que asigna las recompensas se define de la siguiente manera:

$$\text{recompensa} = \begin{cases} +100 & \text{dron llega al objetivo y termina episodio} \\ -100 & \text{dron colisiona y termina episodio} \\ -1 & \text{dron toma acción de girar izquierda o derecha} \\ \pm\Delta d & \text{dron se acerca o aleja del objetivo} \end{cases} \tag{1}$$

En esta función se dara una recompensa negativa de -1 si el dron se mantiene girando en el sitio ya que se intentara que llegue en el menor tiempo posible al objetivo. Por otro lado, Δd representa la distancia que el dron se ha alejado o aproximado al objetivo. Además, el dron tiene un margen de error de 1 metro al rededor del objetivo para considerar el episodio como satisfactorio. El final de un episodio se determinará si una de las mediciones láser toma un valor por debajo de 0,6 metros considerándose que el dron esta demasiado cerca del obstáculo.

Finalmente,el espacio de estados se define de dos formas distintas dependiendo de si el algoritmo es tabular o de aprendizaje profundo. En los aprendizajes por refuerzo tabular el espacio de estados se definirá como $[láser1, láser2, láser3, \phi, d]$. En la Figura 4 se muestra los elementos que definen este espacio. Por un lado, se miden tres direcciones láser, **láser1**, **láser2** y **láser3**. Estas mediciones láser son una en dirección paralela al eje x del dron y las otras dos formando un ángulo de 16° y $-16°$ respectivamente. Además estas mediciones serán discretizadas en 6 valores con un rango máximo de 6 metros, es decir, con un resolución de 1 metro. El siguiente valor de las observaciones es ϕ el cual mide el ángulo relativo que forman el je x del dron con respecto a la distancia euclidia del dron al objetivo.

La resolución de este ángulo es de 30°, es decir, puede tomar 12 valores diferentes. Por ultimo, el valor d representa la distancia euclídea del dron al objetivo con una resolución de decímetros y un valor máximo de 282.

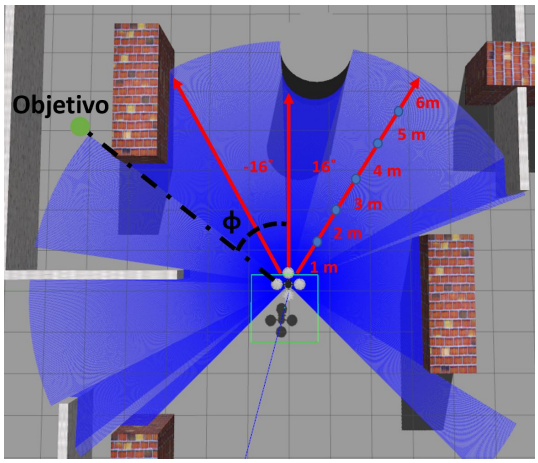


Figura 4: Espacio de estados de algoritmos tabulares

En cuanto al espacio de estados creado para el aprendizaje profundo por refuerzo se tomo como referencia [**4**]. Para ello se tomo una imagen de profundidad 30x100 de la cámara estéreo. La parte superior de esta imagen muestra una sección negra en un fondo blanco cuya posición cambiara con respecto al ángulo relativo del dron y el objetivo. Estas imágenes serán introducidas en la red neuronal para ajustar los pesos de la función parametrizada desarrollada con Keras-RL [**5**].

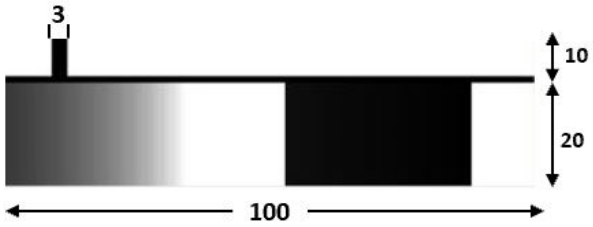


Figura 5: Espacio de estados de algoritmos de aprendizaje profundo

3.3. Interfaz gráfica de usuario

La interfaz gráfica permite seleccionar los parámetros del entorno de aprendizaje y ejecutarlo, al mismo tiempo que muestra los resultados y comparaciones de los aprendizajes. La interfaz se divide en dos ventanas. La primera ventana llamada "*Test/Train*"permite al usuario seleccionar el algoritmo con el que se quiere

entrenar el dron, así como de los parámetros que lo definen, a la vez que muestra gráficas de los resultados en tiempo real. Por otro lado, la ventana "*Compare*"dará la opción al usuario a comparar gráficamente los resultados de los aprendizajes realizados previamente.

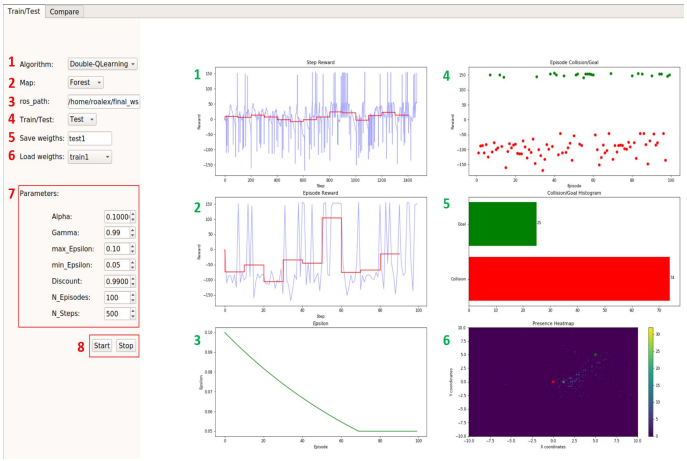


Figura 6: Ventana "Train/Test"de la interfaz gráfica de usuario

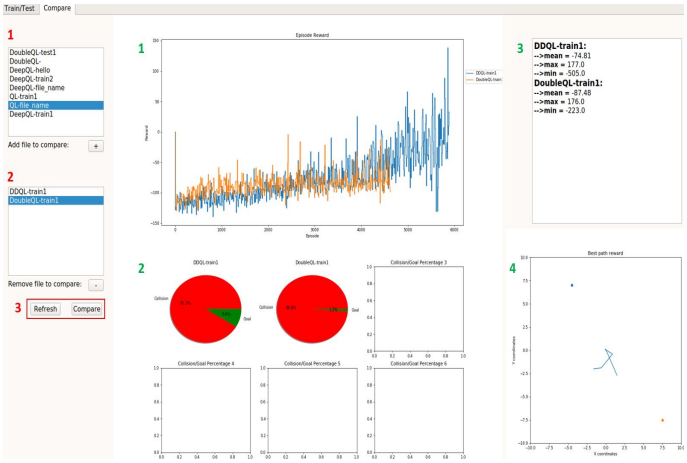


Figura 7: Ventana "Train/Test"de la interfaz gráfica de usuario

4. Resultados

Para determinar los resultados del desarrollo del proyecto, se entrenaron y probaron los algoritmos que se implementaron en la aplicación. En vista a estos

resultados, se confirmo que el dron es capaz de definir trayectorias mas eficientes y satisfactorias aplicando los algoritmos de aprendizaje por refuerzo profundo. En el Cuadro 3, se muestra que para una inferencia de 100 episodios, *Deep Q-learning* proporciona los resultados más satisfactorios seguido por *Double Deep Q-learning*. En la diferencia de los resultados de los algoritmos, influye en gran medida la discretización del espacio de observaciones en un menor número de estados en los algoritmos tabulares. A diferencia de los algoritmos de aprendizaje profundo que permiten una mejor aproximación de la tabla Q mediante funciones parametrizadas.

Algoritmo	**Media**	**Máx-Mín**	**Porcentaje acierto**
Q-learning	-45,31	[174/-209]	11,8 %
Double Q-learning	-48,85	[177/-192]	11,1 %
Deep Q-learning	130,39	[175/-98]	89 %
Double Deep Q-learning	91,52	[177/-95]	74 %

Cuadro 3: Comparación de inferencias de algoritmos

En la Figura 8 también se puede observar como los algoritmos de aprendizaje profundo definen trayectorias más óptimas hacia el objetivo, a diferencia de los algoritmos tabulares que necesitan de mas movimientos para terminar el episodio satisfactoriamente.

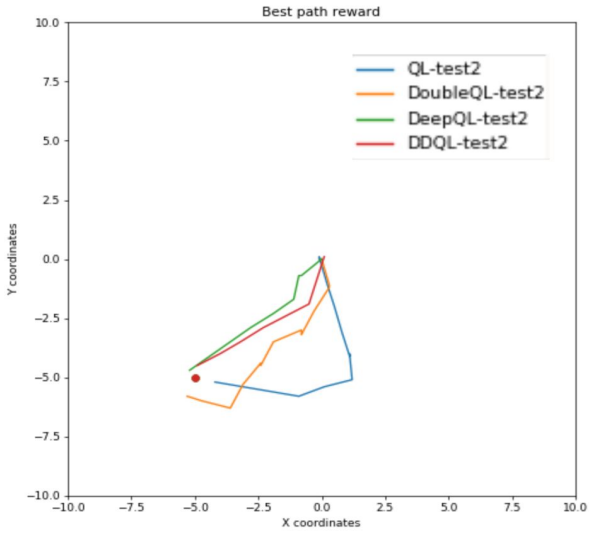
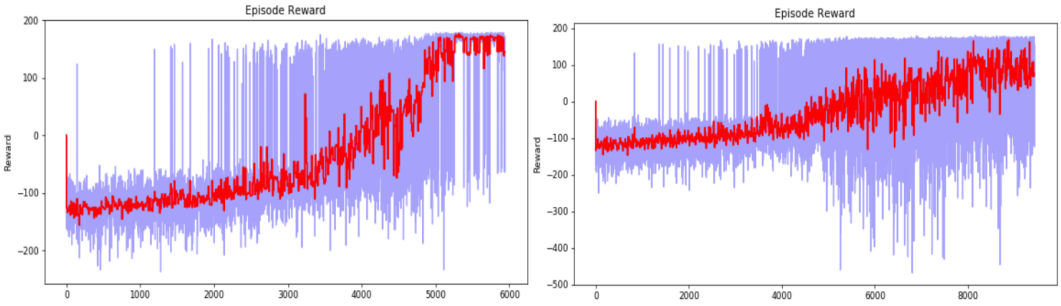


Figura 8: Trayectorias óptimas de algoritmos de aprendizaje por refuerzo

Finalmente, durante los entrenamientos de aprendizaje por refuerzo profundo se



(a) Entrenamiento con objetivo único (b) Entrenamiento con objetivo aleatorio

Figura 9: Gráficas de recompensa/episodio entrenamiento Deep Q-learnin

comprobó un sobre-entrenamiento por parte del agente, es decir, el dron memorizaba como llegar a un único objetivo sin recoger suficientes datos como para llegar a otro distinto. Para solucionar este problema se asigno durante el entrenamiento objetivos aleatorios al inicio de cada episodio. Como se puede comprobar en la Figura 9, los entrenamientos con objetivos aleatorios generan una gráfica de recompensa por episodio mas lenta y con mas ruido que con un solo objetivo. Esto se debe a que el espacio de observa iones es mas grande y se requiere de as tiempo para aprenderlo.

5. Conclusiones

En vista a los resultados de este proyecto se puede decir que se ha llegado a cumplimentar el objetivo principal de crear una plataforma de fácil acceso que habilita el estudio de algoritmos de aprendizaje por refuerzo aplicados a la navegación de drones. Además, a parte de implementar esos algoritmos en una simulación de un dron se consiguió probar todos ellos y compararlos entre sí. Por otro lado, al tratarse de una primera aproximación la linea de este trabajo será continuada en el futuro. Comenzando por desarrollar la aplicación para Windows 10, único subobjetivo que no se cumplimento de los acordados. Otro inconveniente que se encontró durante el desarrollo fue la baja potencia computacional de la que se disponía, lo que provoco simulaciones lentas y entrenamientos de entre 4 a 5 días. Para solucionar este problema se propone desarrollar la aplicación en un sistema en la "*nube*"que proporcione una mayor potencia computacional y permita acelerar las simulaciones de los entornos. Para acceder al código fuente que se ha desarrollado para este proyecto se adjunta el enlace del repositorio de GitHub[1].

[1]Repositorio HectorRL:`https://github.com/Rodridelazcano/HectorRL`

6. Referencias

[1] Haye Kesteloo. *Drone industry is growing fast to $100 billion by 2020 and $1.5 trillion by 2040*. 2019. url: https://dronedj.com/2019/11/25/drone-industry-100-billion-2020/ (visitado 08-08-2020).

[2] S.Kohlbrecher J.Meyer A.Sendobry. *"Comprehensive Simulation of Quadrotor UAVsUsing ROS and Gazebo"*. SIMPAR(2012).

[3] Miguel Angel Rodriguez Alberto Ezquerro y Ricardo Tellez (of The Construct). *openai_ros*.URL:url:http://wiki.ros.org/openai_ros(visitado:29.05.2020).

[4] Kersandt, Kjell and Muñoz, Guillem and Barrado, Cristina. *"Self-training by Reinforcement Learning for Full-autonomous Drones of the Future"*. 2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC). IEEE. 2018, págs. 1-10.

[5] Matthias Plappert. *keras-rl*. https://github.com/keras-rl/keras-rl. 2016.

PLATFORM FOR SIMULATING A DRONE'S BEHAVIOR USING REINFORCEMENT LEARNING ALGORITHMS

Author: **Pérez Vicente, Rodrigo de Lazcano**.
Director: Sanz Bobi, Miguel Ángel.
Collaborating Entity: ICAI –Universidad Pontificia Comillas.

ABSTRACT

The objective of this project is the development of a graphical platform on which reinforcement learning algorithms can be implemented to facilitate the autonomous navigation of a simulated drone. This platform is a first approximation for a system that will allow convenient access to users looking to investigate autonomous drone navigation by applying reinforcement learning techniques. In the context of this project, the results of four reinforcement learning algorithms were developed and compared: Q-learning, Double Q-learning, Deep Q-learning and Double Deep Q-learning.

Palabras clave: UAV, reinforcement learning, autonomous navigation, OpenAI Gym, ROS

1. Introduction

Unmanned aerial vehicles, or drones, have been studied in the aeronautical sector for several years, however, not until recently drones have experienced an increasing adaptation in a wide variety of industry sectors. Such is the case, that Morgan Stanley estimates that the market value of drones will rise to 1.5 trillion US dollars by 2040[**1**]. As the demand for drones in various industrial sectors increases, there is also a breakthrough in the various technologies that allow these drones to adapt to new applications. One of these technological trends that is being researched with a lot of activity is the autonomous navigation of drones. Autonomous drone navigation will in the future allow these machines to require no human oversight. This will allow drones to be able to navigate in environments with high density of obstacles avoiding human collision error. In addition, providing drones with full autonomy will eliminate dependence on the control of these machines by external signals that can be weakened in enclosed spaces or with many obstacles in the way, causing interferences.

On the other hand, to give drones autonomy, they are currently being applied and researching machine learning techniques, in particular, an area within them

called reinforcement learning. Reinforcement learning defines two components: the agent, in this case the drone, and on the other hand the environment where the agent interacts. This method of learning resembles the way humans learn, i.e. by trial and error, interacting with the environment to a certain goal. One of the great advantages that this learning method allows is the incorporation of simulations, avoiding economic expenses of material for real tests. In addition, compared to supervised and unsupervised learning, reinforcement learning has the advantage of requiring less initially classified data and supporting a real-time learning, i.e. the drone will learn while trying or reaching the goal at the same time.

In this way, this project seeks to provide content to the investigations of autonomous navigation of drones by creating a graphical user interface that allows training a simulation of a drone through reinforcement learning. The task to learn will be to create a trajectory with the drone to a goal, avoiding obstacles in its path. This project will be a first approximation and new lines of work will be suggested to be able to bring this platform closer to more research related to drones and reinforcement learning.

2. Objectives

Once the context of the project is located, the objectives pursued in the project will then be explained:

- Implement in ROS and Gazebo the simulation of the control of a drone and the environment with which the drone will interact in reinforcement learning trainings.

- Integrate the simulation with the Python package, OpenAI Gym. With this package the algorithms will be developed to train the drone to go from one starting point to another target on the map avoiding obstacles. The reinforcement learning algorithms to be implemented are: Q-learing, Double Q-learning, Deep Q-learning and Double Deep Q-learning.

- Design a graphical user interface that will allow to interact with the different algorithms, as well as analyze the results of the tests. The application will be developed in both Windows 10 and Ubuntu to allow greater access to more researchers or users.

- Develop in the graphical user interface an option to compare the different trainings and tests that have been made with the different learning algorithms by reinforcement learning.

3. System description

For the development of the platform, the software tools in Table 1 were used.

Componentes	**Versión**	**Funcionalidad**
Operating System	Ubuntu 18.04 (x64)	Operating system in which the application is developed
Python	2.7	Programming language with which the application is developed
ROS	Melodic	Midalware in charge of the communications of the platform. Simulation, Algorithms and graphical user interface
Gazebo	9	Simulator of virtual environments in 3D. Maps and drones
OpenAi Gym	0.16	Simplifies the application of reinforcement learning algorithms in already defined environmentss
Keras	2.0.8	Neural network development
TensorFlow	1.4	Backend package used in Keras for calculations
Keras-RL	0.4.2	Integrates neural networks developed in Keras with OpenAI Gym
PyQt	5	Development of the graphical user interface

Table 1: System components.

First of all, the ROS message line and node system develops the platform structure. Figure 1 shows this structure in which ellipses represent nodes and rectangles message lines or *"topics"*. As can also be seen, in Figure 1 the ROS structure can be sectioned into three main parts or nodes: Simulation, Algorithms and *GUI* or graphical user interface. Each of these parties will perform a function of the platform and the necessary information will be sent to each other. The functionality of each part of the ROS structure will then be briefly explained.

3.1. Simulation

The simulation of the environment was performed in Gazebo. On the one hand, the package used was *Hector*, lwhich provides the simulation of a drone with a very

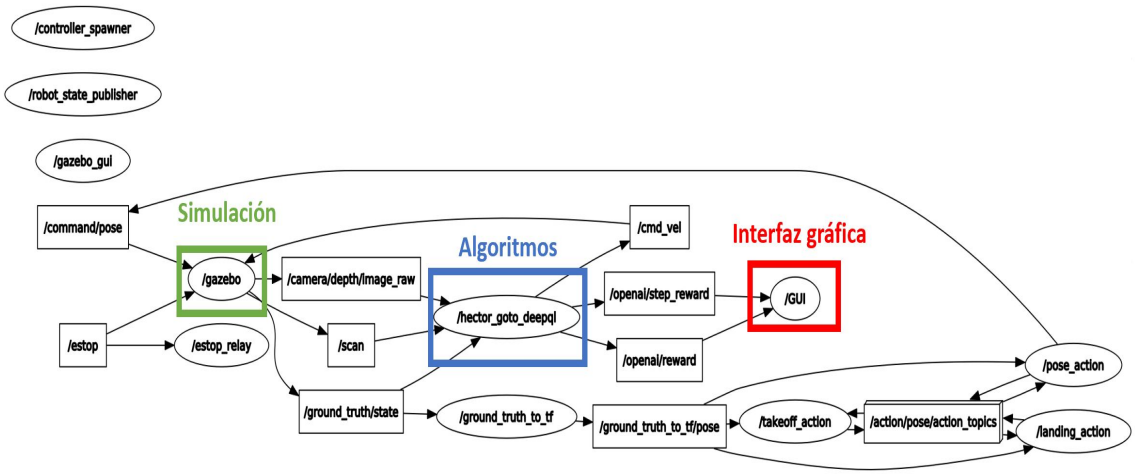


Figure 1: Graph of environment nodes in ROS

(a) Hector quadrotor

(b) Hokuyo UTM-30LX

(c) ASUS Xtion Pro

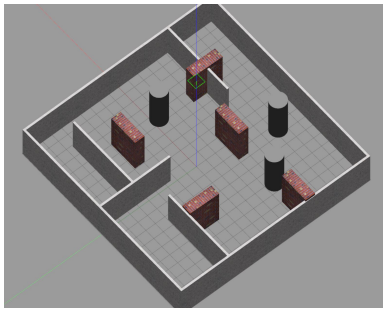
Figure 2: Simulated elements of the Hector drone

realistic flight control and physics. In addition this library provided the sensors used to collect the environment information by the drone[**2**].A range laser sensor, *Hokuyo UTM-30LX* and a depth camera, *ASUS Xtion Pro*.

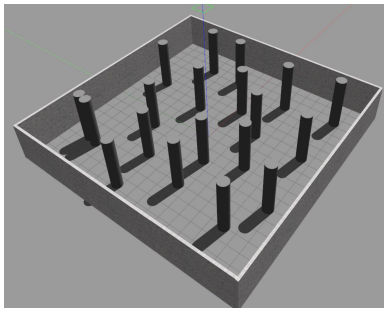
On the other hand, two maps were designed, with dimensions 20x20 meters, for the drone to perform training and inferences of the learning algorithms by reinforcement. *Maze* and *Forest*. To eliminate the bias of training in inference, the drone's task is trained on the map *Maze*, while the tests on the map *Forest*. In this way, the drone was exposed to an environment with new obstacles to check if it had efficiently learned the task.

3.2. Algorithms

Four types of reinforcement learning algorithms were implemented to train the



(a) Maze



(b) Forest

Figure 3: Maps designed in Gazebo

drone to go from one starting point to another goal avoiding obstacles. Before implementing the algorithms, it was necessary to integrate ROS with OpenAI Gym, which was done with the structure established by the package *openai_ros* [**3**]. These algorithms are divided into two groups, tabular algorithms (*Q-learning* y *Double Q-learning*)and deep reinforcement learning algorithms (*Deep Q-learning* y *Deep Double Q-learning*). The two differ in that in tabular algorithms, the action value function *Q(s,a)* is defined in a table with all possible state/action combinations. While deep reinforcement learning algorithms allow to expand the dimensions of the problem by performing a functional approximation with neural networks of the action value function. For this project, all algorithms have in common the definition of the drone's action space, as well as the reward allocation function. However, the observation space varies for tabular algorithms and deep learning algorithms. The design of the deep reinforcement learning environment was inspired by [**4**].

First of all, the drone's action space in the environment consists of three discrete actions in the horizontal plane shown in Table 2. The drone was not allowed to ascent or descent on the axis z as the action space would have been too large for the available computational power. Thus, at the beginning of each episode the drone would rise at a distance of approximately 2 meters and remain constant throughout the rest of the episode.

Table 2: Drone action space

Acción	Valor
Hacia delante	1 m/s
Derecha	30°
Izquierda	30°

The function that assigns rewards is then defined as follows:

$$\text{reward} = \begin{cases} +100 & \text{drone reaches objective and episode ends} \\ -100 & \text{drone crashes and episode ends} \\ -1 & \text{drone takes action of turning left or right} \\ \pm\Delta d & \text{drone gets closer or further from destination point} \end{cases} \tag{2}$$

This function will give a negative reward of -1 if the drone is kept turning in the same position as it will attempt to reach the target in the shortest possible time. On the other hand Δd represents the distance the drone has moved away or approximated from the target. In addition, the drone has a margin of error of 1 meter around the target to consider the episode satisfactory. The end of an episode will determine whether one of the laser measurements takes a value below 0.6 meters considering that the drone is too close to the obstacle.

Finally, the state space is defined in two different ways depending on whether the algorithm is tabular or deep learning. In tabular reinforcement learning, the state space will be defined as $[laser1, laser2, laser3, \phi, d]$. The Figure 4 shows the elements that define this space. On the one hand, three laser directions are measured, **laser1**, **laser2** y **laser3**. These laser measurements are one in the direction parallel to the shaft of the drone and the other two forming an angle of 16° and $-16°$ respectively. In addition, these measurements will be discretized in 6 values with a maximum range of 6 meters, that is, with a resolution of 1 meter. The next value of the observations is ϕ which measures the relative angle that form the drone's x axis relative to the euclide distance from the drone to the target. The resolution of this angle is 30°, that is, it can take 12 different values. Finally, the value d represents the euclidean distance from the drone to the target with a decymeter resolution and a maximum value of 282.
As for the state space created for deep learning by reinforcement, it was taken as a reference [**4**]. To do this, a 30x100 depth image of the stereo camera is taken. The top of this image shows a black section on a white background whose position will change from the relative angle of the drone and the target. These images will be entered into the neural network to adjust the weights of the parameterized function developed with Keras-RL [**5**].

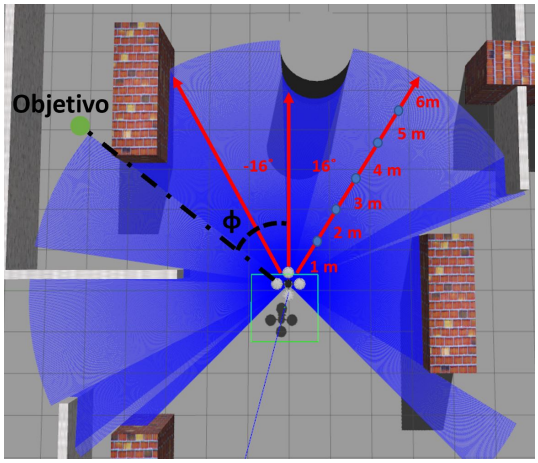


Figure 4: Tabular algorithm state space

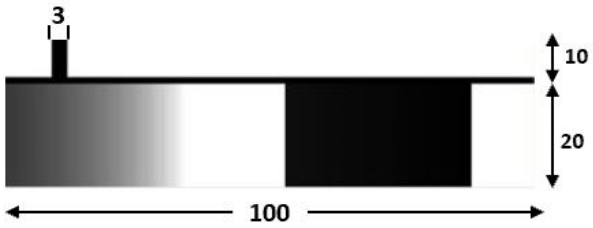


Figure 5: Deep learning algorithm state space

3.3. Graphical user interface

The graphical interface allows to select the parameters of the learning environment and run it, while showing the results and comparisons of the trainings. The interface is divided into two windows. The first window called "*Test/Train*.ªllows the user to select the algorithm with which the drone is to be trained, as well as the parameters that define it, while displaying graphs of the results in real time. On the other hand, the window "*Compare*"will give the user the option to graphically compare the results of the previously performed tutorials.

4. Resultados

To determine the results of project development, the algorithms that were implemented in the application were trained and tested. In view of these results, it was confirmed that the drone can define more efficient and satisfactory paths by

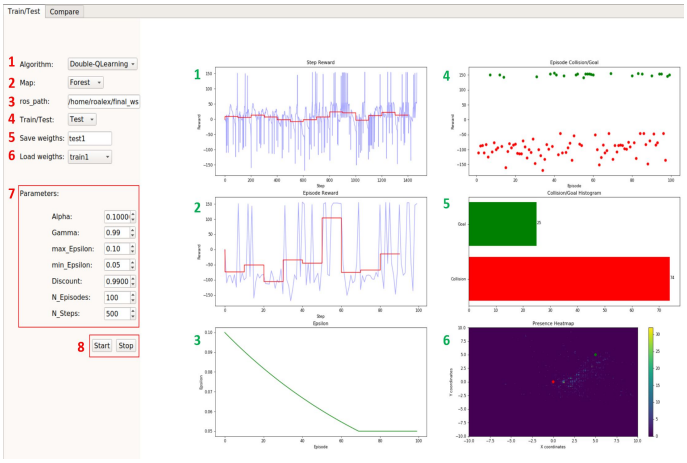


Figure 6: "Train/Test"window of the graphical user interface

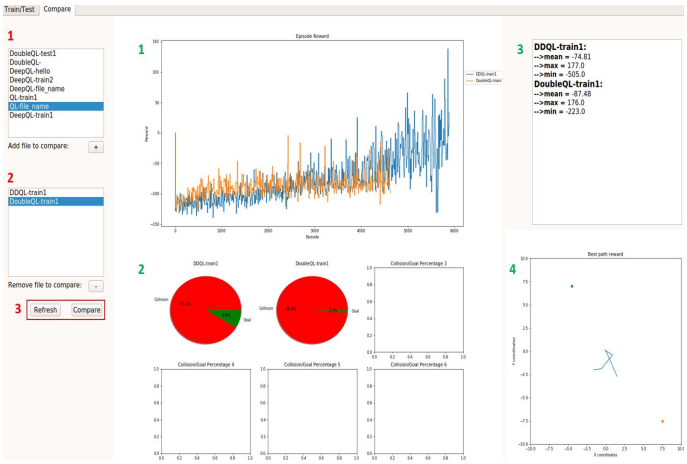


Figure 7: "Train/Test"window of the graphical user interface

applying the deep reinforcement learning algorithms. Table 3 shows that for a 100-episode inference, the most satisfactory results are provided by *Deep Q-learning*, followed by the *Double Deep Q-learning*. In the difference in the results of the algorithms, it greatly influences the discretization of the observation space in a smaller number of states in the tabular algorithms. Unlike deep learning algorithms that allow a better approximation of the Q table using parameterized functions.
Figure 8 can also be seen how deep reinforcement learning algorithms define more optimal paths towards the target, unlike tabular algorithms that need more movements to finish the episode successfully.
Finally, during deep reinforcement learning training, an over-training was tested by the agent. The drone memorized how to reach a single goal without collecting

Algoritm	Mean	Max-Min	Percentage hit
Q-learning	-45,31	[174/-209]	11,8 %
Double Q-learning	-48,85	[177/-192]	11,1 %
Deep Q-learning	130,39	[175/-98]	89 %
Double Deep Q-learning	91,52	[177/-95]	74 %

Table 3: Comparison of Algorithm Inferences

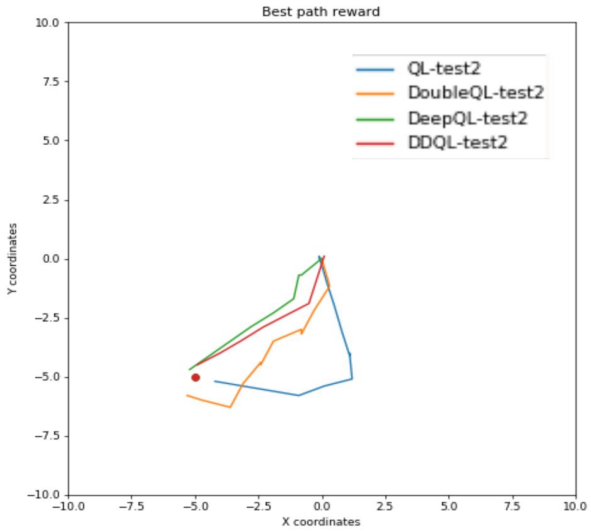
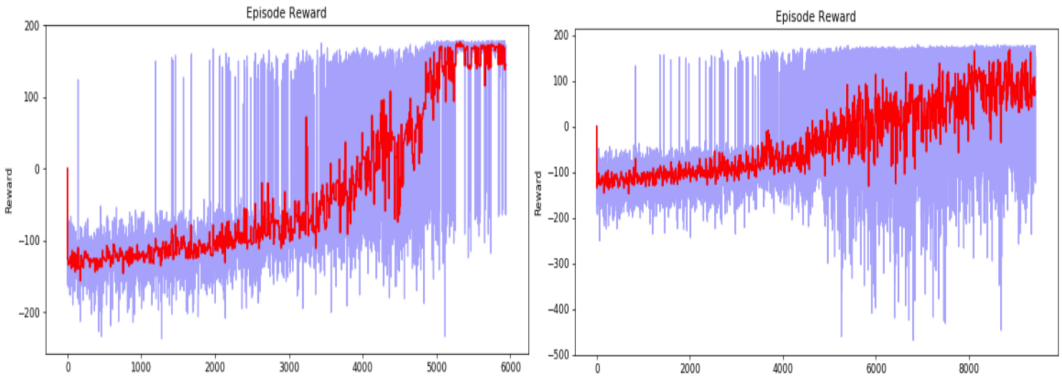


Figure 8: Optimal paths of reinforcement learning algorithms

enough data to reach a different one. To solve this problem it was assigned during training random targets at the beginning of each episode. As you seen in Figure 9, trainings with random targets generate a slower, more noisy per-episode reward graph than with a single target. This is because the observation space is larger and it takes time to learn it.

5. Conclusiones

In view of the results of this project it can be said that the main objective of creating an easily accessible platform has been fulfilled, that enables the study of reinforcement learning algorithms applied to drone navigation. In addition, apart from implementing those algorithms in a drone simulation, it was managed to test all of them and compare them to each other. On the other hand, as it is a first approximation the line of this work will be continued in the future. Starting by developing the application for Windows 10, the only sub-goal that was not met by those agreed. Another drawback encountered during development was the low



(a) Single-goal training (b) Random target training

Figure 9: Deep Q-learnin Training/Reward Charts

computational power available, which caused slow simulations and trainings with a duration of 4 to 5 days. To solve this problem, it is proposed to develop the application on a system in the emph-cloud"that provides greater computational power and allows to accelerate the simulations of the environments. To access the source code that has been developed for this project, the GitHub[2] repository link is attached.

6. References

[1] Haye Kesteloo. *Drone industry is growing fast to $100 billion by 2020 and $1.5 trillion by 2040*. 2019. url: `https://dronedj.com/2019/11/25/drone-industry-100-billion-2020/` (accessed: 08-08-2020).

[2] S.Kohlbrecher J.Meyer A.Sendobry. *"Comprehensive Simulation of Quadrotor UAVsUsing ROS and Gazebo"*. SIMPAR(2012).

[3] Miguel Angel Rodriguez Alberto Ezquerro y Ricardo Tellez (of The Construct). *openai_ros*.URL:`url:http://wiki.ros.org/openai_ros`(visitado:29.05.2020).

[4] Kersandt, Kjell and Muñoz, Guillem and Barrado, Cristina. *"Self-training by Reinforcement Learning for Full-autonomous Drones of the Future"*. 2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC). IEEE. 2018, págs. 1-10.

[2]Repositorio HectorRL:`https://github.com/Rodridelazcano/HectorRL`

[5] Matthias Plappert. *keras-rl*. `https://github.com/keras-rl/keras-rl`. 2016.

Índice general

Índice de figuras

Índice de cuadros

Listados

Parte I

Memoria

Capítulo 1

Contexto

En este capítulo se expondrá la situación actual del contenido y tecnología empleada en el proyecto. En primer lugar se describirá las tendencias en el mercado de drones autónomos, tanto en el presente como en su perspectiva hacia el futuro. A continuación, se hará referencia al campo de aprendizaje por refuerzo. Incluyendo que aplicaciones se están beneficiando de esta tecnología así como sus limitaciones. Finalmente, se procederá a hacer una breve investigación sobre el estado del arte correspondiente con drones autónomos y aprendizaje por refuerzo.

1.1. Mercado de drones autónomos

El dron es un vehículo aéreo no tripulado que, a pesar de haber estado presente a lo largo de muchos años en el sector aeronáutico, actualmente ha cobrado un gran interés en el campo de la investigación y en un amplio abanico de sectores de la industria. En sus comienzos, el uso de drones predominaba en aplicaciones militares. Sin embargo, recientemente, se han desarrollado numerosas aplicaciones comerciales que emplean vehículos aéreos no tripulados para diversas tareas. Por ejemplo, en la industria petrolera, los drones son empleados para la inspección de oleoductos que puedan presentar defectos o fugas; en la agricultura los drones pueden facilitar la monitorización de los campos de cultivo con amplias vistas aéreas; mientras que en el sector de la construcción es posible detectar fallos antes de tiempo y corregirlos gracias a las imágenes aéreas tomadas por drones. Otros sectores que se están beneficiando de la funcionalidad de los drones son la fotografía, la topografía, la minería, la seguridad, logística, así como muchos otros. En la Figura 1.1 publicada por [1], se prevé un crecimiento exponencial en los sectores que están integrando drones en sus actividades.

Como respuesta a la rápida expansión de los drones en la industria, Goldman Sachs

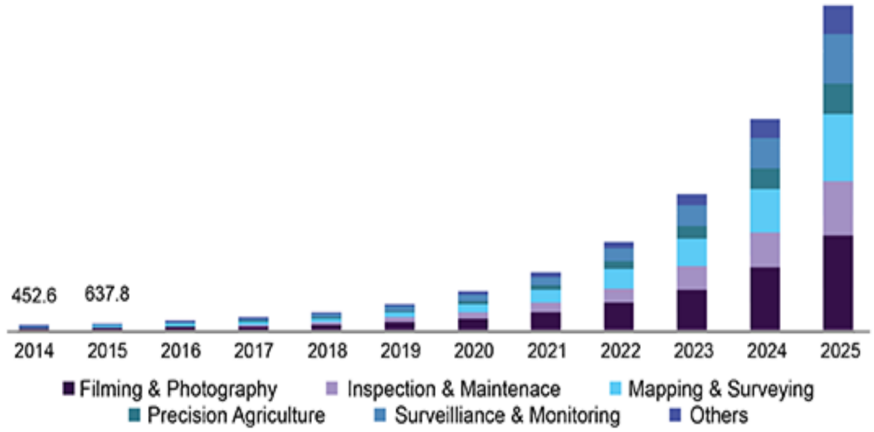


Figura 1.1: Tamaño del mercado de drones en función del sector

publicó un informe [2] en el que se estima que la inversión de mercado en drones, entre 2016 y finales del año 2020, alcance un valor de 100 billones de dolares Americanos. De este presupuesto, 70 billones se distribuirían en aplicaciones militares, mientras que 17 billones estarían destinados a drones dirigidos al consumidor, y los 13 billones restantes a desarrollo civil y comercial. Por otro lado, otro informe publicado por Morgan Stanley [3] estima que la cifra ascienda a 1,5 trillones de dolares Americanos en el año 2040, como se puede observar en la Figura 1.5, siendo el país con el valor de mercado mas alto China.

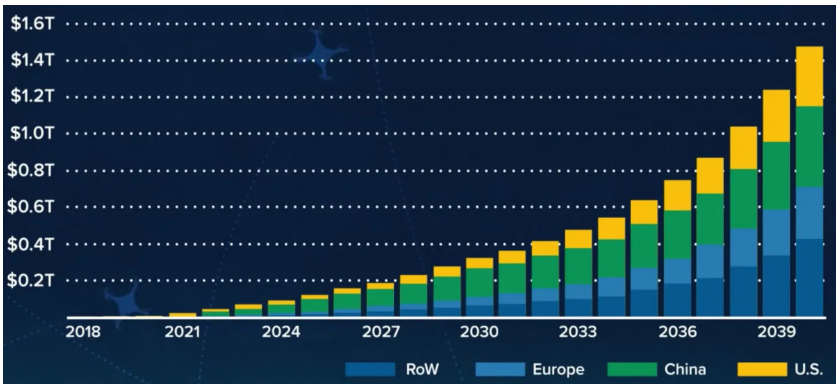


Figura 1.2: Inversión en el mercado de drones 2018 - 2040

A medida que los avances tecnológicos crecen en el sector de los drones industriales, también la demanda de pilotos experimentados capaces de manejar estos aparatos a aumentado. Actualmente, para obtener una licencia de piloto de drones se requiere de una gran inversión de tiempo. Además, en muchos casos el piloto

deberá tener una gran destreza a la hora de maniobrar para evitar obstáculo y no colisionar. Este puede ser el caso de drones que operan en zonas metropolitanas, en interiores de edificios o en áreas boscosas. En todas estas situaciones que se puede ver involucrado el piloto, la planificación de la trayectoria debe tener en cuenta la presencia de obstáculos desconocidos y en ubicaciones variables, como edificios o arboles. Definir una trayectoria segura con este tipo de condiciones puede resultar complicado. De esta manera, según [4] y muchos otros inversores que apuestan por el campo de la investigación, el futuro de los drones industriales se encuentra en dotarles de una completa autonomía. Esto quiere decir que la tendencia en la tecnología de drones no requiera de supervisión humana en todos los sentidos, tanto en el despegue y aterrizaje, como en la carga de baterías y la evasión de obstáculos.

Por otro lado, además de la evasión de obstáculos, los drones autónomos pueden resolver otros problemas relacionados con las actividades en espacios cerrados. En muchas situaciones donde el dron opera en exteriores es común el uso del sistema global de navegación por satélite (Global Navigation Satellite System, GNSS), el cual consiste en una navegación soportada por satélites desde la posición de partida hasta el destino geolocalizado. Sin embargo, la señal de los satélites se debilita en zonas de interiores y con muchos obstáculos, sobre los que la señal puede rebotar. Como consecuencia, el dron no es capaz de recibir las ordenes de navegación y operar con aptitud. Así, al implementar una navegación autónoma al dron, este podrá actuar de forma individual sin requerir de ordenes de agentes externos como puede ser un satélite.

Para poder desarrollar drones que posean de la destreza necesaria para evadir obstáculos de forma autónoma, se requiere de una combinación entre funcionalidades de percepción así como de planificación mediante el uso de sensores ligeros y de altas prestaciones. Entre estos sensores destacan los Lidar, las cámaras de profundidad, sonar, acelerómetros y muchos otros. Además de los componentes hardware, también se requiere del desarrollo de complejos algoritmos de navegación. Un campo de investigación que ha incrementando su actividad en el desarrollo de estos algoritmos es el de la inteligencia artificial, y en particular el aprendizaje por refuerzo como se verá en la próxima sección. La inteligencia artificial tuvo su principal enfoque en la robótica, pero no ha sido hasta ahora que se esta volcando a otros sectores como el de los vehículos terrestres y el entrenamiento de drones, eventualmente, convirtiéndolos en maquinas completamente autónomas.

Como especifica Goldman Sachs en su informe [2], una de las limitaciones del desarrollo de la industria de los drones son las regulaciones gubernamentales. El rápido crecimiento de este mercado ha superado la velocidad de desarrollo de los

(a) EHange 184

(b) DHL Express

(c) Sunflower

Figura 1.3: Drones autónomos que están siendo introducidos en el mercado

sistemas y normas para regular su uso. A pesar de ello, grandes instituciones están tomando iniciativa en instaurar nuevas regulaciones, tales como la NASA con una inversión multimillonaria de dolares Americanos para desarrollar un espacio aéreo en los Estados Unidos capaz de coordinar de forma segura tanto vehículos aéreos tripulados como no tripulados. Al mismo tiempo, la UE tiene planeado renovar su legislación internacional con respecto a los drones para 2021 [5].

Como se puede comprobar los drones autónomos están expandiéndose en el mercado y se está incitando su desarrollo. Así es, que algunas empresas ya están apostando por esta tecnología y han anunciado productos en diferentes ámbitos y sectores. En la Figura 1.3 se hace referencia a algunos de estos drones. Entre estas empresas se encuentra EHang, la cual esta desarrollando un dron autónomo que desempeñaría la función de un taxi, el EHang 184 [6]. Este dron entrará en servicio en Dubái y tendrá capacidad de un pasajero, además de una autonomía de 30 minutos a una velocidad máxima de 160 km/h. Por otro lado, esta empresa China, se ha asociado con la empresa de logística DHL para implementar drones de reparto de mercancías a domicilios en China [7]. También cabe destacar algunas empresas mas pequeñas como Sunflower Labs, la cual está introduciendo drones completamente autónomos en el sector de la seguridad y videovigilancia [8].

1.2. Aprendizaje por refuerzo en sistemas autónomos

En la sección anterior se hizo referencia a la gran importancia que están cobrando los drones autónomos en la actualidad, y se especificó brevemente las posibilidades que puede ofrecer la inteligencia artificial para desarrollar la navegación autónoma de estos drones. La inteligencia artificial ha permitido abordar problemas que se encontraban presentes en otras tecnologías como la localización simultanea y mapeo ("simultaneous localization and mapping, SLAM), la cual resuelve

problemas de localización, pero presenta limitaciones en entornos cambiantes.
El enfoque de la inteligencia artificial normalmente se divide en dos campos del aprendizaje automático. Por un lado se encuentra el aprendizaje supervisado y no supervisado, y por otro lado, el aprendizaje por refuerzo. En el aprendizaje supervisado, se requiere de recoger una gran cantidad de datos que describan el entorno donde el dron realizará sus actividades de navegación. Este método, puede ser efectivo cuando el tamaño del conjunto de datos recogidos es lo suficientemente grande y la clasificación de estos datos muy precisa. Sin embargo, recolectar tan gran tamaño de conjunto de datos puede resultar poco eficiente teniendo en cuenta que el entorno puede variar. Además, la gran cantidad de datos no solo resulta tedioso de recoger, sino que también no se puede asegurar con total certeza que los grupos de datos establecidos en la clasificación sean óptimos para la ejecución de la tarea. Debido a esta inconveniencia, técnicas de aprendizaje automático no supervisado son aplicadas para la clasificación automática de los grandes conjuntos de datos en diversas investigaciones.

Por otro lado, los métodos de aprendizaje por refuerzo tratan de solventar los problemas del aprendizaje supervisado y no supervisado. El aprendizaje por refuerzo permite al dron aprender en un entorno simulado mediante prueba y error. Generalmente el proceso consiste en enseñar a un agente, en este caso un dron, en un entorno virtual similar al mundo real, para finalmente implementar el modelo entrenado a un dron real para realizar la inferencia del modelo.

El aprendizaje por refuerzo, se asemeja a la manera en que los seres humanos aprenden. Este método consiste en aprender, interactuando con el entorno, a como responder para alcanzar cierto objetivo. Se le llama *agente* al aprendiz o tomador de decisiones, mientras que el *entorno* sera representado por todo lo ajeno al agente, es decir, con todo aquello con lo que interactúa. Las interacciones con el entorno tienen lugar en cada uno de los tiempos discretos en una secuencia, o también conocido como pasos de tiempo discreto. En cada escalón de tiempo, el agente observa un *estado* S_t del *espacio de estados* S y en función del estado selecciona una acción $A(S_t)$ a tomar dentro del *espacio de acciones* A_t. Posteriormente, en el siguiente paso de tiempo, el agente obtiene una *recompensa* $R_{t+1} \subset \mathbb{R}$ en función de su buena o mala toma de acción. En la Figura se muestra la interacción de agente-entorno mediante un esquema. Las acciones son las decisión tomadas por el agente, los estados u observaciones determinarán que acción se toma, y por último, la recompensa evalúa el tipo de decisión tomada. De esta manera, el objetivo final del agente es conseguir la máxima recompensa dentro del entorno establecido.

En cada escalón de tiempo, se designan unas probabilidades para decidir entre

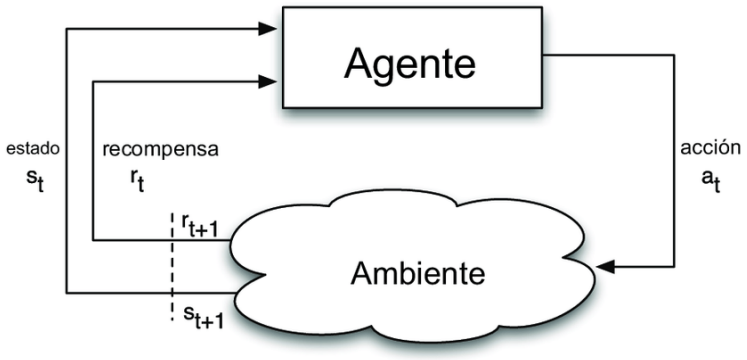


Figura 1.4: Interacción agente-entorno en aprendizaje por refuerzo

las posibles acciones en función del estado en el que se encuentra el agente. Este proceso se conoce como la *política* del agente π_t, siendo $\pi_t(a|s)$ la probabilidad de que $A_t = a$ solo si $S_t = s$. La ecuación correspondiente seria la siguiente:

$$\pi_t(a|s) \doteq P(A_t = a|S_t = s) \tag{1.1}$$

Existen diversos métodos de aprendizaje por refuerzo los cuales se diferencian en como especifican como su política cambia en función de la experiencia del agente. De esta manera, en una aplicación de aprendizaje por refuerzo se persigue la política que maximice la recompensa total. Se buscará, no solo maximizar la recompensa instantánea R_{t+1}, sino que también la recompensa acumulada a lo largo del tiempo G_t. Esta recompensa se la denomina de retorno y se define con la siguiente ecuación:

$$G_t \doteq \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{1.2}$$

donde $\gamma \in [0, 1]$ se conoce como *factor de descuento*, y determinara la importancia de las recompensas futuras. Así, un valor de 0 hará al gente centrarse mas en maximizar la recompensa en el presente, mientras que con un valor cerca de 1 el agente se enfocara mas en maximizar la recompensa que recibirá en el futuro. En la sección **2.3. Algoritmos de aprendizaje por refuerzo** se describirá con mas detalle como los sistemas de aprendizaje por refuerzo siguen un proceso de decisión de Markov, los algoritmos de diferencias temporales como Q-learning y Double Q-learning, así como las variantes que incorporan aprendizaje profundo con redes neuronales como Deep Q-learning y Double Deep Q-learning. Este conjunto de algoritmos serán los que se implementen para desarrollar la aplicación de aprendizaje para la navegación del dron.
Esta sección sirve de breve introducción a lo que es el aprendizaje por refuerzo

y como se desarrolla la base de sus algoritmos. Estos algoritmos, al igual que el campo de investigación de los drones autónomos, están comenzando a ser adoptados por diversos sectores. Sin embargo, la mayoría de estas aplicaciones aun se encuentran en su fase de investigación. Por ejemplo en el área de la salud el aprendizaje por refuerzo esta siendo investigado para poder realizar diagnósticos eficientes, mientras que, en el área de las finanzas se publico en el *Financial Times* que JPMorgan Chase esta usando un sistema basado en aprendizaje por refuerzo para optimizarlas gestiones de sus inversiones [9]. Pero uno de los campos que mas cambio esta notando gracias al aprendizaje por refuerzo es el de la robótica y la automatización. Una de las compañías con mas renombre es OpenAI, cuyo desarrollo mas reciente es el sistema Dactyl, el cual realiza entrenamientos de aprendizaje por refuerzo de forma virtual para después comprobar la eficiencia de estos métodos en maquinas reales.Entre estos entrenamientos se encuentra enseñar a una mano robótica Shadow Dexterous, a resolver un Cubo de Rubik [10]. OpenAI también desarrolló una aplicación para comparar algoritmos de aprendizaje por refuerzo llamada OpenAI Gym, la cual ha sido utilizada para desarrollar este proyecto final de máster.

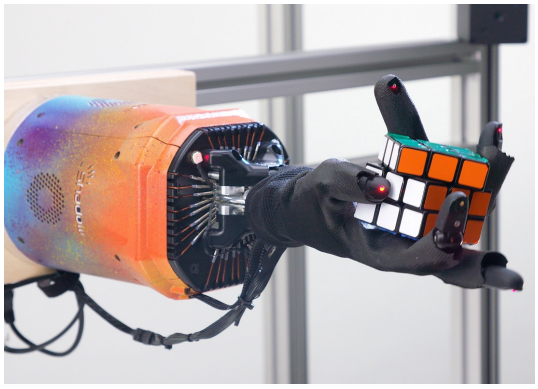


Figura 1.5: OpenAI Dactyl

Como se puede comprobar, la investigación e implementación de aplicaciones de aprendizaje por refuerzo todavía se encuentran en crecimiento pero no se dejan de lado debido a las ventajas que presentan frente a otros métodos de aprendizaje automático:

- A diferencia que el aprendizaje supervisado, el aprendizaje por refuerzo no requiere de grandes bases de datos clasificados. Esto en la práctica es muy importante ya que la cantidad de datos en el mundo entero esta creciendo y cada vez es mas costoso realizar su clasificación. En el caso de un dron

que está aprendiendo a evitar obstáculos, el dron utilizará en tiempo real los datos que recoge del entorno a través de los sensores mientras realiza la navegación de forma continua. Para implementar una navegación con evasión de obstáculos mediante aprendizaje supervisado o no supervisado se hubiese requerido de información y datos del mapa previamente a realizar la navegación y definir la trayectoria.

- Con el aprendizaje por refuerzo se pueden descubrir nuevas formas de afrontar o resolver un problema. A diferencia del aprendizaje supervisado, el cual imita el algoritmo o información que el usuario le proporciona inicialmente.

- En los algoritmos de aprendizaje por refuerzo se elimina la parcialidad que puede presentar la clasificación de los datos en los aprendizajes supervisados. El dron aprendería de forma progresiva con las recompensas establecidas en el aprendizaje por refuerzo. Sin embargo, en el aprendizaje supervisado el dron seria dependiente inicialmente de la forma en que se hayan clasificado los datos de los sensores los cuales pueden presentar de error humano.

- El aprendizaje por refuerzo se considera un aprendizaje *online*, es decir, el agente es capaz de buscar nuevas posibles soluciones durante la *exploración*, a la vez que usa los resultados que ha aprendido en la *explotación*. Como se especifica en el primer punto, en el aprendizaje por refuerzo, el dron recogerá los datos para el aprendizaje en tiempo real a la vez que busca la trayectoria con mayor recompensa. Por otro lado, en los aprendizajes supervisado y no supervisado se necesitarían los datos previamente a realizar la trayectoria óptima.

- Por último, los algoritmos de aprendizaje por refuerzo son compatibles con simulaciones virtuales del agente y del entorno real. Esto permite ahorrar una gran cantidad de costes económicos ya que si el aprendizaje se realizase en entornos reales muchos equipos podrían resultar dañados. Como es el caso de este proyecto, en el que, haber usado un dron real, el aprendizaje podría haber resultado en muchas colisiones perjudicando la funcionalidad del dron.

1.2.1. Limitaciones

El aprendizaje por refuerzo esta muy relacionado con la teoría de control óptima clásica [11]. Ambos métodos se basan en resolver el problema buscando la política óptima que maximiza una función objetivo. Además, ambos dependen de un sistema integrado por una serie de estados, acciones y un modelo que recoge las transiciones de un estado a otro. En la teoría de control óptimo se trabaja con una noción perfecta del sistema con un modelo como base, es decir, el algoritmo

determinara cual sera el siguiente estado del dron, dado un estado y una acción inicial. Por otro lado, el aprendizaje por refuerzo infiere directamente en observaciones y recompensas como resultado de las interacciones con el ambiente, por lo que no requiere de un modelo. Estos métodos de aprendizaje por refuerzo se clasifican como *"model-free"* o libre de modelo.

Sin embargo, los métodos de aprendizaje por refuerzo experimentan los mismos problemas que los algoritmos de control clásico. Tanto complicaciones de memoria como de computación.Esto resulta en problemas de escalabilidad de datos y limita las aplicaciones a problemas de reducidas dimensiones. Para solucionar estos problemas, recientemente se ha introducido el concepto de *aprendizaje profundo*. El aprendizaje profundo emplea las cualidades de aproximación de funciones de las *redes neuronales*, las cuales pueden reducir datos de altas dimensiones, como imágenes, en representaciones mas compactas con menos dimensiones. Un ejemplo que combina las redes neuronales con el aprendizaje por refuerzo es el juego de videojuegos automático a partir del procesamiento de píxeles [12]. En este proyecto se analizarán tanto algoritmos de aprendizaje por refuerzo básicos, como aquellos que implementan redes neuronales, definidos con el termino de *aprendizaje profundo por refuerzo*.

A pesar de la incorporación de aprendizaje profundo, el aprendizaje por refuerzo todavía presenta algunas limitaciones a considerar. En primer lugar, la cantidad de experiencia requerida para que un agente aprenda comportamientos eficientes es muy elevada, lo que resulta en muchas pruebas para muestrear el comportamiento el entorno. Como se especificó anteriormente, una solución para ahorrar costes económicos es utilizar simulaciones de entornos virtuales en vez de realizar el aprendizaje en el mundo real. Esto hace al aprendizaje por refuerzo muy dependiente de la potencia computacional con el que se implemente y aun teniendo grandes prestaciones de computación, los problemas de aprendizaje complejos pueden requerir de mucho tiempo, incluso días.

En segundo lugar, a raíz de la implementación en simulaciones virtuales surge un inconveniente al realizar pruebas en el mundo real. En una simulación, a pesar de tener la ventaja de poder colisionar el agente tantas veces como se desee, simular un modelo que sea idéntico al entorno real es muy complicado. En general, en las simulaciones que difieren mucho de la realidad ocurre un submodelado del aprendizaje. Debido a este submodelado, pequeños errores se van acumulando y el agente simulado puede divergir rápidamente del sistema real. Por ejemplo, en el caso del dron, en una simulación es complejo tener en cuenta las condiciones meteorológicas o de iluminación del mundo real, lo que producirá una incertidumbre

de su funcionalidad en la realidad al finalizar el aprendizaje simulado. Una posible solución es realizar un aprendizaje inicial simulado para posteriormente ajustar el modelo con un dron real [13].

1.3. Motivación y objetivos

Como se ha mencionado anteriormente, tanto la investigación de drones autónomos como en el campo del aprendizaje por refuerzo esta incrementando en la actualidad. En este proyecto final de máster se propone combinar ambos campos de investigación para solucionar un problema no-trivial en la navegación de drones autónomos y comprobar su viabilidad. Este problema consiste en dirigir un dron hacia una posición objetivo evitando obstáculos en el camino sin previamente conocer el entorno. Como se expone en la siguiente sección, existen investigaciones relacionadas con la navegación de drones autónomos aplicando aprendizaje por refuerzo. Sin embargo, estas investigaciones tienen en falta la facilitación de una interfaz que permita a los usuarios implementar, de forma sencilla y sin conocimientos de programación, los distintos algoritmos de aprendizaje por refuerzo que se emplean y poder comparar sus resultados. De esta manera, este proyecto fin de máster se enfocara en desarrollar una aplicación con los siguientes objetivos:

- Desarrollar un entorno virtual con un dron y sus sensores, este dron se introducirá en un mapa con obstáculos. La simulación sera implementada con ROS y Gazebo.

- Desarrollar cuatro algoritmos de aprendizaje por refuerzo: Q-learning, Double Q-learning, Deep Q-learning y Double Deep Q-learning. Estos algoritmos tendrán como objetivo enseñar al dron a ir de un punto a otro del mapa evadiendo los obstáculos de la trayectoria. Como requisito, también se usará la librería de Python, OpenAI Gym, para facilitar la implementación de conceptos de aprendizaje por refuerzo en entornos virtuales.

- Diseñar una interfaz gráfica de usuario en la que se pueda seleccionar entre los diversos algoritmos de aprendizaje por refuerzo y modificar los parámetros de cada algoritmo. La interfaz será diseñada en Ubuntu 18.04 con la posibilidad de ser implementada en Windows 10 para el fácil acceso a otros desarrolladores.

- Implementar en la interfaz una opción de comparación de entrenamientos para poder diferenciar entre los resultados de cada algoritmo y comprobar cual de ellos es mas eficiente en la aplicación de evasión de obstáculos del dron.

1.4. Estado del arte

Actualmente existen varias investigaciones relacionadas con la navegación autónoma de drones implementando técnicas de aprendizaje por refuerzo y técnicas para evasión de obstáculos en aplicaciones robóticas. La mayor diferencia entre estas aplicaciones son los datos de entrada que utilizan los agentes para realizar el aprendizaje, es decir, los sensores que se implementan en el agente y la información que estos recogen del entorno. En [14], se especifican los tipos de sensores que los drones pueden emplear para efectuar la detección de obstáculos. Entre estos sensores destacan las cámaras estéreo y monoculares mediante las cuales se pueden derivar imágenes de profundidad del entorno, sensores ultrasónicos que a través de altas frecuencias determinan la distancia a los obstáculos que rodean al agente, y sensores LIDAR los cuales funcionan de la misma manera que los sensores de ultrasonido salvo que emiten luz en vez de sonido y permiten una mayor precisión a la hora de detectar obstáculos.

En el campo del aprendizaje por refuerzo aplicado a la navegación de agentes autónomos, el uso de un tipo de sensor u otro, dependerá del algoritmo y del tipo de agente o dron para el cual se efectuara dicho algoritmo. En [15], se justifica el uso de sensores que los drones de uso comerciales suelen llevar incorporados y no suponen un coste o carga adicional para el dron, como por ejemplo, acelerómetro, cámara monocular o IMU. Con la entrada de estos sensores se desarrolla un sistema de navegación autónomo basado en *"Deep Q-learning"(DQL)*. Sin embargo, existen otras investigaciones en las que se emplea sensores con mejores prestaciones, permitiendo la aplicación de algoritmos más complejos como *"Deep Q-learning"(DQL)*, *"Double Deep Q-learning"(DDQL)* y *"Dueling Deep Q-learning"(Dueling DQL)*. En [16] se integran dos cámaras, una que toma medidas de profundidad mientras que la otra es una cámara RGB enfocada a clasificación de objetos. En esta investigación, las imágenes capturadas por las dos cámaras son procesados por una red neuronal convolucional (CNN) para posteriormente implementar un aprendizaje por refuerzo basado en *"Double Deep Q-learning"(DDQL)* y *"Dueling Deep Q-learning"(Dueling DQL)*. Esta aplicación da resultados muy eficientes en los experimentos realizados en diversos entornos virtuales desconocidos, pero la computación requerida es muy alta y el sistema empleado tiene un alto coste económico.

Por otro lado, otras investigaciones simplifican la exploración de entornos desconocidos empleando técnicas similares y reduciendo el número de sensores. Por ejemplo, en [17], se comparan dos técnicas de RL, *"Deep Q-learning"(DQL)* y *"Double Deep Q-learning"(DDQL)*, aplicadas a un dron con una única cámara

estéreo que detecta la profundidad a la que se encuentran los obstáculos en su foco de visión y sobre la que se aplicara la red neuronal que procesa las imágenes. Además, en otra investigación, [18], se implementa un sensor láser adicional para realizar la función de recompensa del algoritmo de aprendizaje, volviendo a inicializar el aprendizaje si el dron se aproxima a un obstáculo dentro de unos límites.

En cuanto a los algoritmos más básicos de RL, como *"Q-learning"*, estos son implementados normalmente usando sensores láser o ultrasónicos ya que son más efectivos en entornos donde los estados y acciones del dron son lineales, lo cual permiten que los algoritmos de *"Q-learning"* y *"Double Q-learning"* se puedan desarrollar como un proceso de decisión de Markov finito con estados discretos simples. En [19], se utiliza un robot humanoide equipado con un sensor láser cuya finalidad es navegar un entorno desconocido aplicando *"Q-learning"* en la evasión de obstáculos a la vez que realiza un mapeo del entorno con la técnica de SLAM. Por otra parte, en [20], se utilizan dos sensores ultrasónicos económicos para guiar la navegación de un vehículo terrestre mediante *"Q-learning"* y conseguir que llegue a un destino de forma autónoma y evitando obstáculos del entorno simulado.

Finalmente, para crear la plataforma de aprendizaje se integrarán OpenAIGym Gym, ROS y Gazebo. En las investigaciones mencionadas anteriormente que aplican RL, todas ellas utilizan OpenAI Gym para crear los entornos de aprendizaje y comparar los distinto algoritmos de RL. Para simular los entornos, existen diversas maneras de afrontarlo. Un simulador de entornos comúnmente usado en aplicaciones para drones es Microsoft AirSim [21] [16]. Este simulador y controlador esta basado en Unreal Engine y proporciona un alto nivel de realismo con el coste de mayor computación. Por otro lado, en otros proyectos aplican la interfaz de simulación de Gazebo junto con ROS (Robot Operating System) [15] [18]. Gazebo proporciona un simulador para todo tipo de robots capaz de simular entornos de exteriores e interiores con gran precisión, además de una gran variedad de sensores y librerías complementarias que permiten la implementación de diversas tecnologías. La interfaz de Gazebo esta perfectamente diseñada para interactuar con ROS, un *"midelware"* que provee librerías y herramientas para ayudar a los desarrolladores de software a crear aplicaciones para robots [22]. Para integrar las tres interfaces de OpenAIGym, ROS y Gazebo, se han desarrollado diversas librerías que permiten trabajar con ellas a la vez y se basan en el mismo concepto de comunicación entre interfaces. *Gym-Gazebo* [23], es una de las primeras librerías que se publico con este propósito y a resultado ser efectiva en varios proyectos de RL [18][24]. También la pagina web oficial de ROS hace referencia a otra librería publica llamada *openai_ros* para este tipo de proyectos y el cual contiene diversos ejemplos con diferentes agentes [25].

En el Cuadro 1.1 se realiza una comparación de las características de los proyectos mencionados anteriormente, con las caracteristicas que se han implementado en este proyecto final de máster.

		Sensores					**Algoritmos**					**Simulación**	
Referencia	Dron Autónomo	Sonar	LIDAR	Cámara RGB	Cámara Estéreo	IMU	Q-learning	Double QL	Deep QL	Double DQL	Dueling DQL	ROS/Gazebo	AirSim
[15]	✓	x	x	✓	x	✓	x	x	✓	x	x	x	✓
[16]	✓	x	x	✓	✓	x	x	x	x	x	✓	x	✓
[17]	✓	x	x	x	✓	x	x	x	✓	✓	x	x	✓
[18]	✓	x	✓	✓	x	x	x	x	✓	x	x	✓	x
[19]	x	x	✓	x	x	x	✓	x	x	x	x	x	x
[20]	x	✓	x	x	x	x	✓	x	x	x	x	x	x
[21]	✓	x	x	x	✓	x	x	x	✓	✓	✓	x	✓
[23]	x	x	x	x	x	x	✓	x	x	x	x	✓	x
[24]	x	x	x	x	x	x	✓	x	x	x	x	✓	x
[25]	x	x	x	x	x	x	✓	x	x	x	x	✓	x
Proyecto Final	✓	x	✓	x	✓	x	✓	✓	✓	✓	x	✓	x

Cuadro 1.1: Comparación del estado del arte con contenido del proyecto.

Capítulo 2

Metodología

A lo largo de este capítulo se expondrán los procedimientos seguidos para desarrollar la aplicación de aprendizaje por refuerzo que compone este proyecto de acorde con la planificación de trabajo mostrada en la sección **2.1. Cronograma de trabajo**. Se comenzará por analizar las herramientas que se van a emplear para el desarrollo. Incluyendo las herramientas de programación para desarrollar los algoritmos de aprendizaje por refuerzo y la simulación del dron, así como el hardware que va a realizar la computación de los entrenamientos. Seguidamente, se describirá como se han integrado todas esas herramientas de programación para que sean compatibles entre ellas, y así poder desarrollar la aplicación. A continuación, se explicará la funcionalidad y como se han integrado los distintos algoritmos de aprendizaje por refuerzo. En esa misma sección también se describirá las características del agente, sensores, acciones y recompensas; así como, las características del entorno. Por último, se darán pautas para interactuar con el diseño de la aplicación e interpretar los resultados representados en las gráficas.

2.1. Cronograma de trabajo

Para el desarrollo del proyecto se han definido una serie de tareas que se asemejan a la estructuración de este capitulo y el que continua de la memoria:

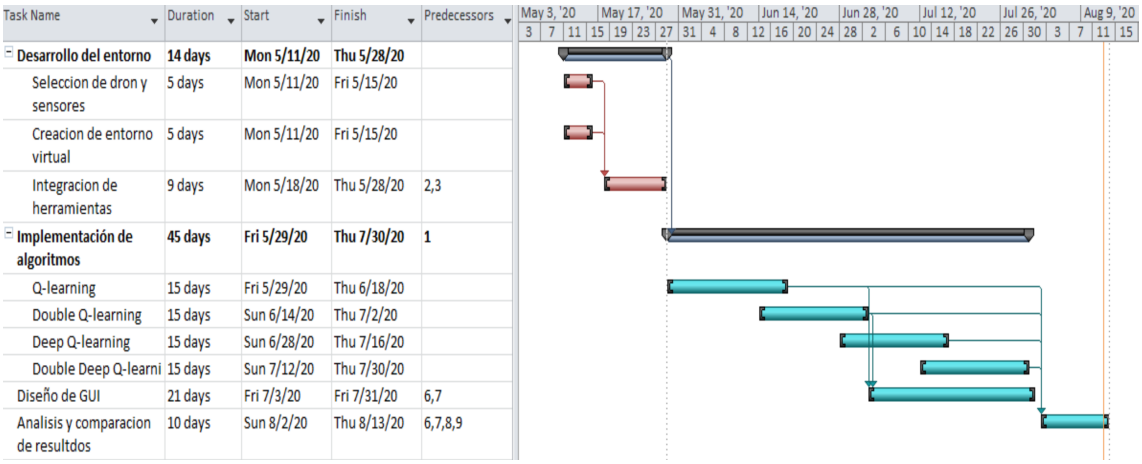
1. **2.2. Desarrollo del entorno simulado**: análisis y justificación de las herramientas software empleadas, así como su integración para que sean compatibles entre ellas y desarrollar la aplicación de aprendizaje por refuerzo.

2. **2.3. Algoritmos de aprendizaje por refuerzo**: estudio e implementación de los algoritmos de aprendizaje por refuerzo en la aplicación: Q-learning, Double Q-learning, Deep Q-learning y Double Deep Q-learning.

3. **2.4. Diseño y uso de la aplicación**: descripción del diseño de la interfaz gráfica de usuario y como interpretar los resultados de los entrenamientos.

4. **Capítulo 3 - Resultados**: análisis y comparación de los resultados obtenidos al entrenar el dron con cada algoritmo de aprendizaje por refuerzo.

En la Figura 2.1 se representa el diagrama de Gantt del proyecto donde se exponen las tareas ordenadas temporalmente.



Task Name	Duration	Start	Finish	Predecessors
Desarrollo del entorno	**14 days**	**Mon 5/11/20**	**Thu 5/28/20**	
Seleccion de dron y sensores	5 days	Mon 5/11/20	Fri 5/15/20	
Creacion de entorno virtual	5 days	Mon 5/11/20	Fri 5/15/20	
Integracion de herramientas	9 days	Mon 5/18/20	Thu 5/28/20	2,3
Implementación de algoritmos	**45 days**	**Fri 5/29/20**	**Thu 7/30/20**	**1**
Q-learning	15 days	Fri 5/29/20	Thu 6/18/20	
Double Q-learning	15 days	Sun 6/14/20	Thu 7/2/20	
Deep Q-learning	15 days	Sun 6/28/20	Thu 7/16/20	
Double Deep Q-learni	15 days	Sun 7/12/20	Thu 7/30/20	
Diseño de GUI	21 days	Fri 7/3/20	Fri 7/31/20	6,7
Analisis y comparacion de resultdos	10 days	Sun 8/2/20	Thu 8/13/20	6,7,8,9

Figura 2.1: Cronograma de trabajo, diagrama de Gantt

2.2. Desarrollo del entorno simulado

2.2.1. Herramientas

En esta sección aparte de describir los componentes de programación empleados en el proyecto, también se hace referencia a las características del ordenador que ejecutara la aplicación. Las herramientas de programación se pueden dividir en tres áreas, la primera corresponde a la simulación virtual del dron para la cual se usará ROS y Gazebo. La segunda área corresponde con el desarrollo de los algoritmos de aprendizaje por refuerzo, implementados con la ayuda de OpenAI Gym junto con Keras-RL. Y por último, para desarrollar la interfaz gráfica de usuario se empleará PyQt.

Las especificaciones de los componentes de los componentes software del sistema se desglosa en el Cuadro 2.1.

Componentes	Versión
Sistema Operativo	Ubuntu 18.04 (x64)
Python	2.7
ROS	Melodic
Gazebo	9
OpenAi Gym	0.16
Keras	2.0.8
TensorFlow	1.4
Keras-RL	0.4.2
PyQt	5

Cuadro 2.1: Componentes del sistema.

ROS - Gazebo

ROS y Gazebo son dos herramientas software independientes pero que suelen combinarse en proyectos de investigación robótica ya que tienen una gran compatibilidad y muestran resultados muy eficientes. Por un lado, con Gazebo se pueden desarrollar simulaciones gráficas 3D incluyendo física de objetos realista. Mientras que ROS, actúa como un sistema operativo para el robot real o a simular. En el caso de este proyecto, en ROS se desarrollarán las diversas funcionalidades del dron (control de simulación, acciones, aprendizaje e interfaz gráfica) creando *nodos*, los cuales se comunicarán entre ellos mediante canales de mensajes denominados *topics*. Se dice que un *nodos* esta *subscrito* a un canal de mensaje cuando recibe información de este, y *publicador* cuando envía mensajes a ese canal.

La razón por la que se seleccionó este método para implementar la simulación del dron y del entorno por encima de otras opciones como AirSim, está relacionado con la versatilidad que aporta la combinación de ROS y Gazebo. AirSim, a pesar de ofrecer entornos mas realistas, no permite tener tanto control sobre el entorno simulado, ya que todo el sistema, tanto simulación como las funcionalidades del dron se integran en la interfaz de AirSim. Por otro lado, ROS permite crear tantas funcionalidades como se necesite en el dron, lo que ha permitido implementar en el proyecto la aplicación de aprendizaje por refuerzo junto con la interfaz gráfica de usuario como nodos independientes con sus canales de mensajes en un mismo espacio de trabajo. Además, a pesar de que Gazebo simule entornos menos realistas que AirSim, la computación gráfica requerida es menor y los resultados son lo suficientemente representativos como para poder implementar el sistema en la realidad con cierta confianza.

ROS y Gazebo presentan numerosas librerías con licencias de código abierto, de las cuales hay una gran variedad de simulaciones de drones disponibles. De entre todas las librerías disponibles se seleccionó *Hector*. La librería *Hector* dispone de un modelo simulado en Gazebo de un cuadricóptero,mostrado en la Figra 2.2. Así como, de diversos controles que permiten estabilizar el vuelo del dron mediante comandos de velocidad y posición. Hector cuadricóptero fue seleccionado para llevar acabo este proyecto ya que, a pesar de que el dron no exista físicamente, proporciona una simulación realista de la física de vuelo así como del control de un dron en el mundo real [26], permitiendo centrar el desarrollo del proyecto en los objetivos de navegación por aprendizaje por refuerzo. Además, la librería proporciona de diversos sensores reales que se pueden integrar con el cuadricóptero. Los sensores empleados y sus características se muestran en el Cuadro 2.2 y Figura 2.3.

Figura 2.2: Simulación en Gazebo de Hector cuadricóptero

OpenAI Gym

Una vez integradas las simulaciones del agente y del entorno en ROS y Gazebo, hay que configurar al agente para que realice el aprendizaje por refuerzo. Ni ROS ni Gazebo proporcionan funcionalidades de aprendizaje por refuerzo. Por esta razón, se integra OpenAI Gym en la simulación, una herramienta que permite desarrollar y comparar algoritmos de aprendizaje por refuerzo. La utilidad base de OpenAI Gym consiste en estandarizar con métodos de Python la interacción de los componentes del aprendizaje por refuerzo, agente y entorno, lo que permite probar y comparar distintos algoritmos sobre una misma combinación de agente-entorno. Esta combinacion en OpenAI Gym se simplifica con la clase `Env` o *environment*.

La librería OpenAI Gym incluye diversos *environments* en los que se pueden probar distintos algoritmos de aprendizaje. Sin embargo, en el caso de este pro-

(a) Hokuyo UTM-30LX

(b) ASUS Xtion Pro

Figura 2.3: Sensores empleados de la librería Hector

Sensor	Modelo	Características
Telémetro láser de escaneo	Hokuyo UTM-30LX	▪ Rango: 100-30.000 [mm] ▪ 25 msec/escaneo ▪ Rango del área de escaneo 270° ▪ Resolución angular de 0,25°
Cámara estéreo de profundidad	ASUS Xtion Pro	▪ Rango: 0.8-3.5 [m] ▪ Resolución: 1280x1024 ▪ Profundidad del tamaño de la imagen: VGA(640x480) : 30 fps

Cuadro 2.2: Sensores utilizados de la librería Hector.

yecto, hay que desarrollar un nuevo *environment* creando nuevos métodos que simplifiquen el proceso de agente-entorno del cuadricóptero *Hector*.

Las clases estandarizadas `Env` están formadas por los siguientes métodos:

- `reset(self)`: reinicializa el entorno al estado inicial, al comienzo de un entrenamiento o cuando se termina un episodio para empezar otro nuevo.
- `step(self, acción)`: procesa un paso temporal dentro de un episodio en función de la acción especificada por el algoritmo de aprendizaje por refuerzo a partir de la observación del entorno. Este método retorna como respuesta el estado u observación del entorno tras el paso (`observation`), la recompensa al tomar esa acción y encontrase en el nuevo estado (`reward`) y un valor booleano de si el episodio se encuentra en un estado de terminacion o no (`done`).

Keras-RL

Keras-RL es otra librería implementada en Python [27], la cual está diseñada para el desarrollo de aprendizajes profundos por refuerzo. Keras-RL tiene integrada la librería de aprendizaje profundo *Keras*[28], y con base de computación de cálculos *TensorFlow*. La gran ventaja que aporta para este proyecto, es la implementación directa, tanto para probar como evaluar, distintos algoritmos de aprendizaje profundo por refuerzo desarrollados a partir de estado del arte. Los algoritmos que se implementaron usando esta librería son *Deep Q-learning* y *Double Deep Q-learning*.

PyQt

Qt se trata de un framework, con orientación a objetos, utilizado para el diseño de interfaces gráficas de usuario. Aunque el lenguaje nativo de esta aplicación es C++, también puede ser compatible con Python a través de la librería PyQt. De esta manera, se escogió esta aplicación para desarrollar la interfaz interactiva de aprendizaje por refuerzo ya que proporciona compatibilidad con Python y otras librerías para representación de datos gráficamente. Además, tiene una gran versatilidad para migrar las aplicaciones a distintos dispositivos con distintos sistemas operativos, pudiéndose transportar la aplicación de un sistema a otro.

Hardware

El ordenador empleado para realizar los entrenamientos de aprendizaje por refuerzo no estaba diseñado para este tipo de aplicaciones, pero tenía las prestaciones suficientes como para recoger resultados y entrenar los diversos algoritmos de aprendizaje por refuerzo. Debido a la baja memoria VRAM de la tarjeta gráfica, fue necesario usar únicamente la CPU para la computación de cálculos de las redes neuronales implementadas en el aprendizaje profundo, dejando a la simulación virtual que se beneficiase de toda la potencia de la GPU. El mayor inconveniente,

fue la duración de tiempo que tardaban los entrenamientos en converger, de aproximadamente 4 días. A continuacion, se desglosa las prestaciones del ordenador empleado para el entrenamiento del dron:

- **RAM**: 8 GB.
- **CPU**: Intel Core i7-2600k 3.4 GHz y 4 núcleo
- **GPU**: Nvidia GTX560 1 GB

2.2.2. Integración de la aplicación de aprendizaje por refuerzo

Para desarrollar la aplicación de aprendizaje por refuerzo se requiere combinar e integrar las diferentes herramientas expuesta en la sección anterior. De esta manera, el proyecto se dividirá en tres partes relacionadas entre sí: la simulación del entorno y el dron, la funcionalidad de aprendizaje por refuerzo, y la interfaz gráfica de usuario. Además, cada una de estas partes se desarrollará con una de las librerías especificadas anteriormente. Para desarrollar la simulación se usará Gazebo, para la implementación de los algoritmos se empleará un entorno de OpenAI Gym junto con la librería de Keras-RL, y por último, para crear la interfaz gráfica se dará uso de PyQt.

La comunicación entre todas estas aplicaciones se realizará mediante el sistema de *"nodos"* y *"topics"* de ROS. Como se hizo referencia anteriormente, ROS es un *framework* que aporta la funcionalidad de sistema operativo o *"midelware"* del sistema robótico, en este caso el dron. De esta manera, ROS será el sistema más alto y abstracto en la jerarquía de programación dentro de la aplicación, permitiendo desarrollar la simulación, los algoritmos y la interfaz gráfica independientemente, para posteriormente combinarlos en una misma estructura bajo el sistema de comunicación de *"nodos"* de ROS. En la Figura 2.4 se muestra el diagrama de nodos de ROS donde las distintas funcionalidades de la aplicación se representan como nodos contenidos en las elipses, mientras que los canales de comunicación entre los nodos, denominados *"topics"*, se representan como rectángulos.

A continuación, se describirá como el diagrama de la Figura 2.4 se divide en las partes que constituyen el proyecto y como se comunican entre si para formar el sistema final.

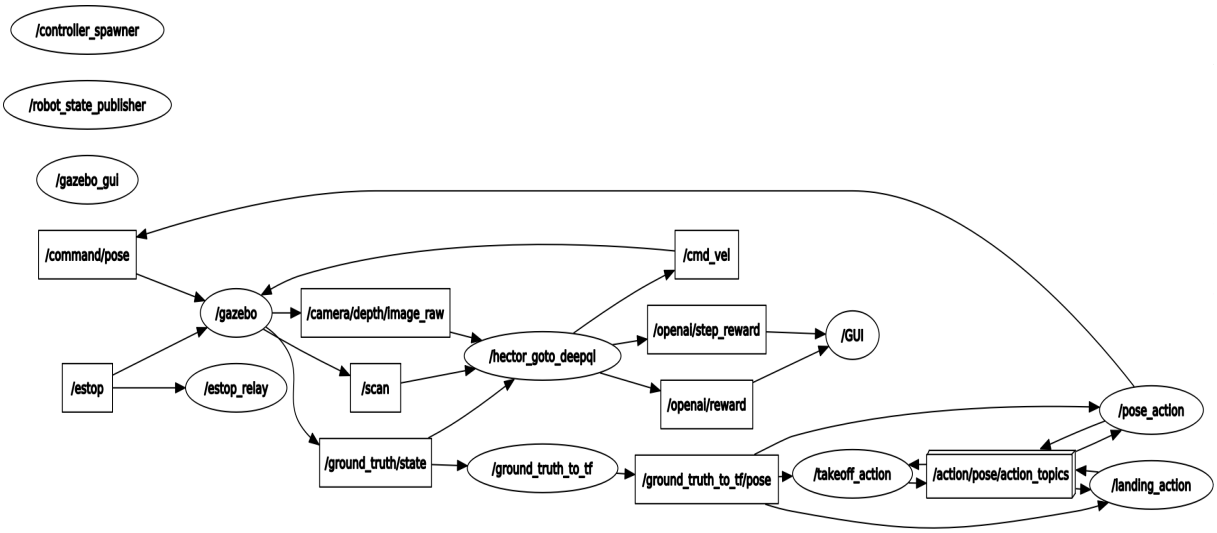


Figura 2.4: Diagrama de nodos de ROS

Simulación

La simulación incluye la representación visual 3D en Gazebo, así como los módulos que proporciona la librería *Hector* para controlar el vuelo del dron. En la Figura 2.5 se muestra el conjunto de nodos y canales de mensajes del diagrama de la Figura 2.4 que definen la *simulación* en el sistema de ROS. Los elementos mas relevantes son el nodo */gazebo* y el canal de mensajes */ground_truth/state*.
En el nodo */gazebo*, se representa el modelo del dron además del entorno o mapa en el que interactúa. Este nodo esta subscrito a canales de mensajes que permiten dar comandos de posición y velocidad (*command/pose* y */cmd_vel*) al dron para que navegue por la simulación virtual, al mismo tiempo que puede publicar información en otros canales de mensajes sobre los datos recogidos por los sensores del dron.

Al inicializar el nodo */gazebo*, además de introducir la simulación del modelo de *Hector* en Gazebo, también se inicializa un entorno o *mundo*. El *mundo* es el entorno virtual 3D de Gazebo donde el dron realizará el aprendizaje. Para este proyecto se crearon dos *mundos* distintos . Se crearon dos *mundos* para proporcionar parcialidad a la hora del entrenamiento y comprobar que independientemente del entorno en el que se realice el entrenamiento, el dron es capaz de aplicar lo aprendido en ambos y conseguir el objetivo de evitar obstáculos previamente desconocidos en cualquier entorno. Los *mundos* se muestran en la Figura 2.6 y Figura 2.7. Ambos *mundos* tienen una dimensión total de 20x20 [m] y la posición inicial en la que aparece el dron en cada episodio de aprendizaje es en el centro del mapa, (0,0).

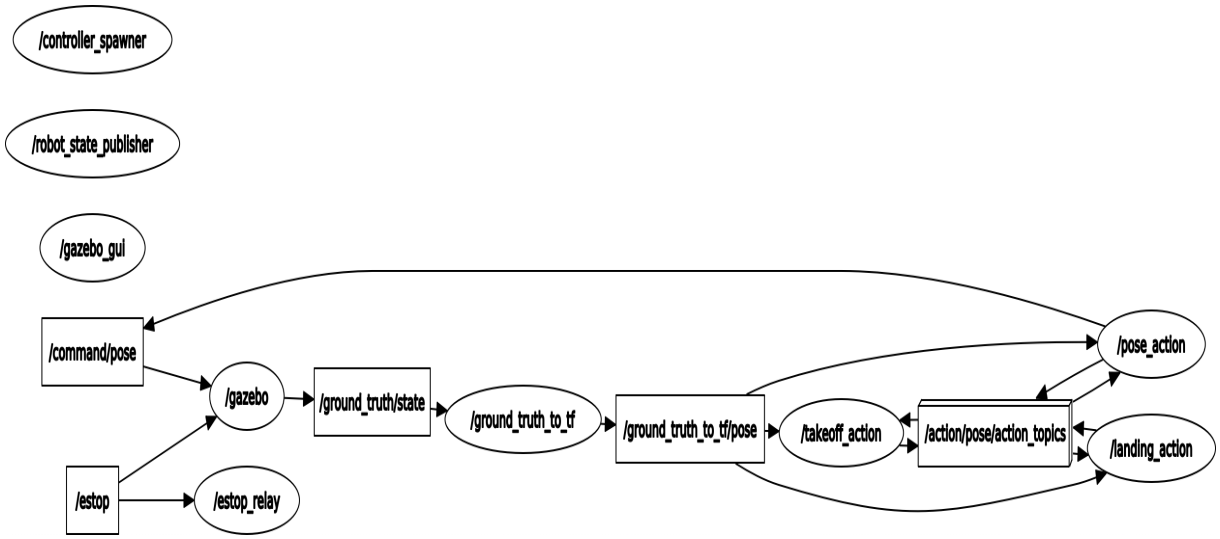


Figura 2.5: Diagrama de nodos de ROS de la simulación Hector

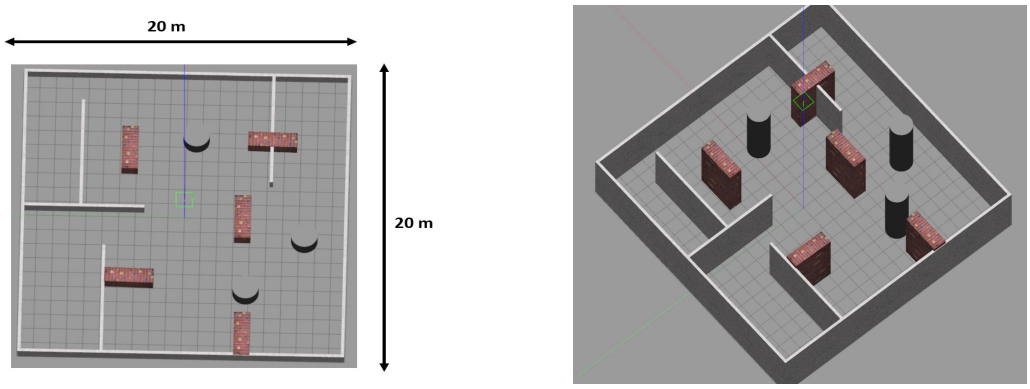
Por otro lado, */gazebo* publica sobre el canal de mensajes */ground_truth/state* mensajes con la estructura *nav_msgs/Odometry.msg*. Este mensaje puede ser utilizado como los datos recogidos por un sensor IMU ya que proporciona información sobre la posición cartesiana en tres dimensiones, así como de la orientación del dron en cuaterniones y las velocidades lineales y angulares sobre todos los ejes.

Entorno de aprendizaje por refuerzo

El entorno de aprendizaje por refuerzo del sistema en ROS corresponde a un único nodo de la Figura 2.4, etiquetado con el nombre */hector_goto_deepql*. El nombre del nodo dependerá del algoritmo de aprendizaje por refuerzo que se haya decidido ejecutar en la interfaz gráfica de usuario. De esta manera, la interfaz es la encargada de inicializar el nodo de aprendizaje, el cual puede ser */hector_goto_qlearn* para el algoritmo de *Q-learning*, */hector_goto_doubleql* para *Double Q-learning*, y */hector_goto_deepql* tanto para *Deep Q-learning* como *Double Deep Q-learning*.

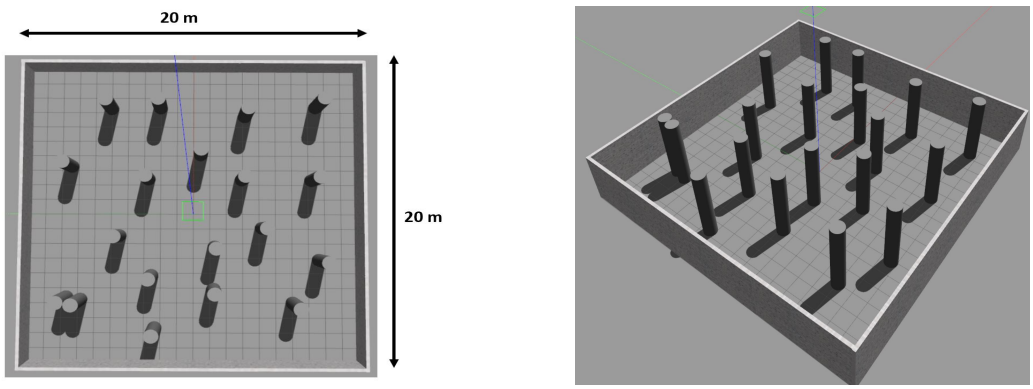
El entorno de aprendizaje inicializado por la interfaz esta formado por una serie de clases con funcionalidades independientes que heredan los métodos de unas a otras jerárquicamente. La estructura del código del entorno está inspirada en la librería *openai_ros*[1] [25] y en la Figura 2.8 se muestra como se relacionan entre si los distintos ficheros que constituyen el entorno con sus métodos mas característicos.

[1]openai_ros:http://wiki.ros.org/openai_ros



(a) Planta (b) Perspectiva

Figura 2.6: Mundo *"Maze"*



(a) Planta (b) Perspectiva

Figura 2.7: Mundo *"Forest"*

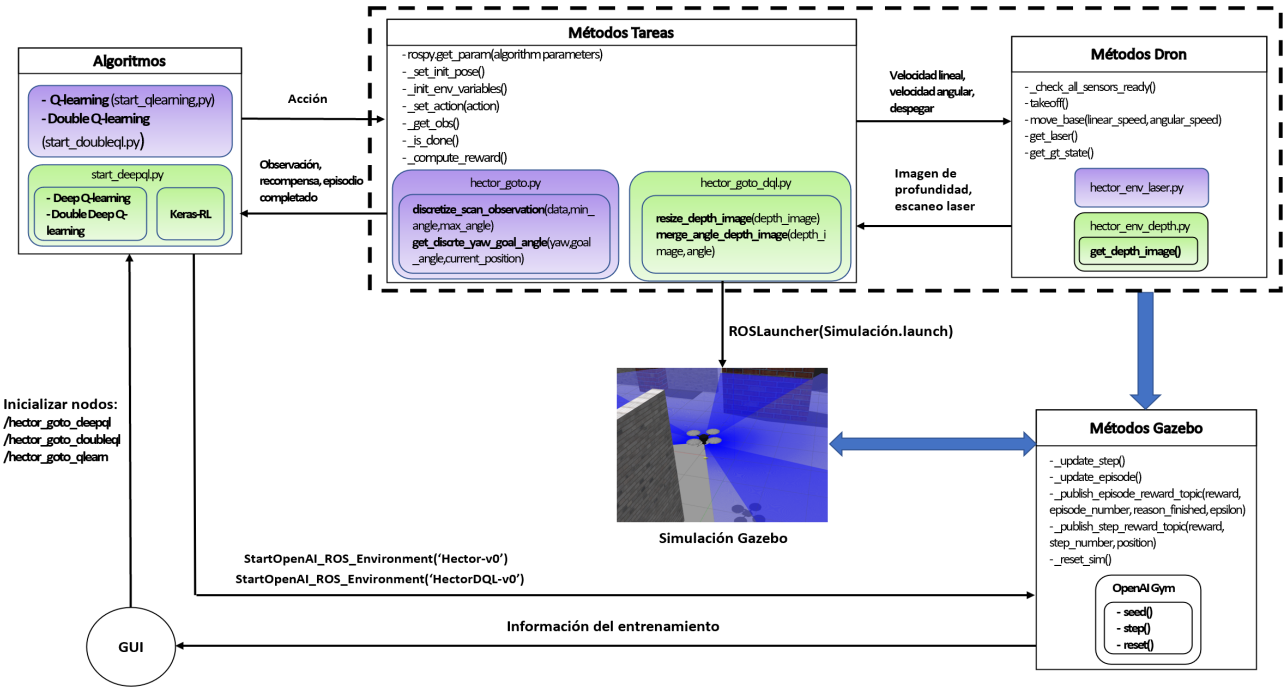


Figura 2.8: Estructuración del nodo de aprendizaje por refuerzo

Cuando se inicializa un entrenamiento de un algoritmo de aprendizaje por refuerzo en la interfaz, el primer fichero que se ejecuta es **Algoritmos**. Este fichero, por un lado construirá un entorno de OpenAI Gym compatible con ROS y Gazebo. Por otro lado, usará los métodos de ese entorno para definir matemáticamente el algoritmo de aprendizaje por refuerzo con el formato de `reset(self)` y `step(self,action)` de un entorno típico de OpenAI Gym. Una ventaja de esta estructura es que el entorno de OpenAI Gym con ROS y Gazebo es completamente independiente del fichero con el algoritmo de aprendizaje, por lo que se pueden desarrollar varios algoritmos distintos para un mismo entorno sin realizar cambios en este último. La inicialización del entorno de entrenamiento se especifica con la función `StartOpenAI_ROS_Environment(Env)`. La variable `Env` puede tener dos variables distinta, `Hector-v0` o HectorDQL-v0. Se pueden inicializar dos entornos distintos ya que las características del dron para los algoritmos de aprendizaje por refuerzo tabular, en morado, tienen prestaciones distintas al dron con aprendizaje por refuerzo profundo, en verde. Además, como se puede observar en la Figura 2.8, variables de entrada y salida del fichero **Algoritmos** son las usuales de un entorno de OpenAI Gym, salidas: *acción*, y entradas: *observación*, *recompensa* y *episodio completado*.

El entorno de entrenamiento por lo tanto estará constituido por los **Métodos** de las clases **Tareas**, **Dron**, **Gazebo** y **OpenAI Gym**. Estas clases heredan los métodos unas de otras formando una jerarquía de métodos. En esta jerarquía la clase más alta es **Tareas**, la cual hereda los métodos de **Dron**, seguidamente de **Gazebo**, y por último los métodos del entorno de **OpenAI Gym**.

Primeramente, los métodos de la clase **Gazebo** permiten aplicar las interacciones entre las simulaciones en Gazebo y el entorno de entrenamiento. Esta clase incluye el método para reinicializar la simulación después de cada paso y de los controles del dron (`_reset_sim()`). Por otro lado, esta clase es la encargada de implementar las funciones de OpenAI Gym requeridas para desarrollar un entrenamiento de aprendizaje por refuerzo, `reset(self)`, `step(self,action)`, `seed(self)` y `close(self)`. Además, en estas funciones de OpenAI Gym se llamaran a funciones de las clases hijo **Tareas** y **Dron**, para implementar las acciones y recoger los datos de las observaciones. Finalmente, la clase **Gazebo** también publicará en los canales de mensajes de ROS, */openai/reward* y */openai/step_reward*, información sobre cada paso del entrenamiento y episodio para que la interfaz gráfica de usuario pueda recoger los datos y procesarlos en gráficas de forma visual.

La siguiente clase que hereda los métodos de **Gazebo** es **Dron**. Esta clase contiene las funciones necesarias para interactuar directamente con el dron, es decir, esta clase contendrán todas las funcionalidades de ROS que l dron necesita para ser controlado durante el entrenamiento. En primer lugar, al principio de cada episodio, comprueba que todas las comunicaciones con los sensores y servicios de ROS están activos con la función `_check_all_sensors_ready()` y despega el dron con la función takeoff(). Por otro lado, para enviar comandos de velocidad, tanto angular como lineal, al dron por el canal de mensajes */cmd_vel*, se emplea la funcion move_base(linear_speed,angular_speed). Además la clase **Dron**, contiene las funciones get_laser y get_gt_state, las cuales obtienen los datos del sensor *Hokuyo UTM-30LX* y de la odometría del dron en la simulación. Finalmente existen dos ficheros para definir la clase **Dron** en los dos entornos de aprendizaje por refuerzo, tabular y profundo. Ambos utilizan las funciones especificadas previamente, sin embargo el fichero *hector_env_depth.py*, representado como un rectángulo morado en la Figura 2.8, añade la función get_depth_image() ya que en el entrenamiento profundo se incorpora al dron una cámara de profundidad y esta función recoge la información de dicho sensor.

La última clase que define el entorno de entrenamiento, **Tareas**, también se puede instanciar con dos ficheros independientes dependiendo del tipo de algoritmo de aprendizaje por refuerzo seleccionado para el entrenamiento, *hector_goto.py* y

hector_goto_dql.py. Las funciones de esta clase proporcionan todo el contexto sobre la tarea que debe cumplir el dron, ir de un punto a otro del mapa evitando los obstáculos. Más específicamente, las funciones de la clase **Tareas** definen como se desarrolla la acción en cada paso del entrenamiento dentro de la función de la clase **Gazebo** `step(self,action)`. A continuación se describen brevemente algunos de estos métodos:

- **initial_environment()**: inicializa los parámetros iniciales del entorno al principio de cada episodio. Por ejemplo, en esta función está incluida la selección de destino aleatorio durante el entrenamiento.
- **initialize_pose()**: inicializa el dron en la posición inicial al principio de cada episodio.
- **take_action()**: especifica como se va a procesar la acción seleccionada por el algoritmo de aprendizaje.
- **get_observation()**: retorna y procesa la observación resultante de la acción procesada en la función take_action().
- **return_reward()**: procesa el resultado de la acción tomada y la observación para obtener la recompensa respectiva.
- **episode_done()**: comprueba en cada paso del episodio si el episodio es terminal o no, es decir si llega al objetivo el dron o si colisiona.

A parte de estas funciones, se definen otras especificas de cada algoritmo de aprendizaje. Así, para el fichero *hector_goto.py* los métodos adicionales son:

- **discretize_scan_observation(data,min_angle,max_angle)**: discretiza los valores tomados por el sensor láser para definir el estado observado del aprendizaje por refuerzo tabular.
- **get_discrete_yaw_goal_angle(yaw,goal_angle,current_position)**: discretiza el ángulo relativo del objetivo con respecto al eje x del dron para definir el estado observado del aprendizaje por refuerzo tabular.

Mientras que en el fichero *hector_goto_dql.py* contiene funciones especificas para las tareas de aprendizaje por refuerzo profundo, las cuales se definen a continuación:

- **resize_depth_image(depth_image)**: esta función hace un filtrado y ajuste de la imagen obtenida por la cámara de profundidad para que pueda ser procesada por la red neuronal.

- **get_discrete_yaw_goal_angle(depth_image,angle)**: añade a la imagen de la función resize_depth_image(depth_image) uno píxeles que permiten a la red neuronal identificar el ángulo relativo del eje x del dron con respecto a la posición objetivo.

Finalmente, la simulación del dron en ROS y Gazebo se instancia desde la clase **Tareas** con la función ROSLaunch(simulación.launch). Esta función ejecuta un comando roslaunch, en la linea de comandos de Ubuntu, de un fichero *simulación.launch*. Existen dos ficheros de este tipo para lanzar dos simulaciones distintas dependiendo del entorno que se vaya a ejecutar, *test_flight_gazebo_depth.launch* y *test_flight_gazebo_laser.launch*, las cuales corresponde con los entornos `Hector-v0` o HectorDQL-v0 respectivamente.

Interfaz gráfica de usuario

La interfaz gráfica de usuario será el primer nodo que se inicialice en ROS mediante un icono ejecutable, Figura 2.9. En la Figura 2.4 este nodo esta representado por la elipse */GUI*. Este nodo representa la interfaz gráfica de PyQt y desde el se seleccionará el tipo de entrenamiento y los parámetros de este para ejecutar el resto de nodos del sistema. Aparte de inicializar el aprendizaje mediante ficheros *.launch*, el nodo */GUI* es realimentado por el entorno de aprendizaje con información sobre los episodios de los entrenamientos mediante los canales de mensajes */openai/step_reward* y */openai/reward*. El primer canal proporciona información sobre cada paso dentro del episodio con mensajes del tipo *Step_message.msg*, mientras que el segundo canal proporciona información al finalizar un episodio con mensajes del tipo *Episode_message.msg*.

Figura 2.9: Icono para ejecutar interfaz gráfica de usuario

Los mensajes del tipo *Episode_message.msg* tienen la estructura del Listado 2.1. Las variables que forman este mensaje son: *episode_number*, el cual especifica el número de episodio en el que se encuentra el entrenamiento, *episode_reward*, guarda el valor de la recompensa total de ese número de episodio, *épsilon*, especifica el

valor de la probabilidad de exploración en ese episodio (ϵ-*greedy*), y *reason*, envía la razón por la que se ha acabado el episodio, por colisión o por objetivo alcanzado.

Listado 2.1: Episode_message.msg

```
int32   episode_number
float32 episode_reward
float32 epsilon
string  reason
```

Finalmente, el tipo de mensajes *Step_message.msg* contiene las variables mostradas en el Listado 2.2. Estas variables son: *step_number*, el cual especifica el número de paso en el que se encuentra el episodio, *step_reward*, guarda el valor de la recompensa total de ese número de paso dentro del episodio, *x_pos*, especifica las coordenadas del dron en el eje x, y *y_pos*, haciendo referencia a la posición del dron en el eje y en ese paso del episodio.

Listado 2.2: Step_message.msg.msg

```
int32   step_number
float32 step_reward
float32 x_pos
float32 y_pos
```

2.3. Algoritmos de aprendizaje por refuerzo

En esta sección se describirán los distintos algoritmos de aprendizaje por refuerzo que se han implementado en la aplicación y como se han relacionado los distintos parámetros que los componen con las funcionalidades de las simulaciones del dron *Hector*. En la sección **1.2. Aprendizaje por refuerzo en sistemas autónomos**, se exponen las ideas clave dentro de los algoritmos de aprendizaje por refuerzo. Se explica la interfaz de interacción entre agente-entorno y como se trata de buscar una política óptima que maximice la recompensa adquirida. Como se puede observar, en la interacción de agente-entorno, el agente toma las decisiones en función del estado observado del entorno. Para poder aplicar algoritmos de aprendizaje por refuerzo, esta interacción, debe cumplir la *propiedad de Markov*, en la cual los posibles estados sucesivos son dependientes únicamente del estado actual. Es decir, con la propiedad de Markov, la historia de estados pasados hasta el momento actual equivale a conocer el estado actual. En la simulación del dron se traduce en que en un estado actual del dron en el entorno, los sucesivos estados o posiciones en las que se encontrará el dron son independientes de los estados pasados. Otra propiedad de los entornos con características de *Markov*, es la posibilidad de determinar el próximo estado y su recompensa esperada, dada

un estado y acción actual.

Un entorno que satisfaga las propiedades de *Markov* se le denomina **proceso de decisión de Markov**(*"Markov decission process"* o *MDP*). De esta manera, la decisión del agente en $t+1$, dependerá únicamente del estado y acción tomada en t. Las **probabilidades de las transiciones de estado**, dado cualquier estado s y acción a, que derivan en un estado s' y recompensa r, se representa con la siguiente formula

$$p(s'|s,a) \doteq Pr(S_{t+1}=s', R_{t+1}=r|S_t=s, A_t=a) = \sum_{r\epsilon\mathbb{R}} p(s',r|s,a) \tag{2.1}$$

Para que la política del agente sepa que acción tomar en t, el agente deberá estimar como de favorable es estar en un estado u otro. Para medir esta estimación se emplea lo que se conoce como **función de valores**. Se pueden definir dos tipos de funciones de valores con respecto a una política π determinada, **función de valores de estados** $v_\pi(s)$ y la función de valores de acciones $q_\pi(s,a)$. En la función de valores de estados representa el retorno de recompensa esperado estando en un estado s y siguiendo una política π. Mientras que la función de valores de acciones devuelve el retorno esperado tomando una acción a en un estado s con una política π. A continuación, se muestran las ecuaciones que definen a estas dos funciones

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t=s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t=s] \tag{2.2}$$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t=s, A_t=a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t=s, A_t=a] \tag{2.3}$$

Una propiedad muy importante de las funciones de valores es que cumplen con la relación recursiva el valor de un estado y el valor de los estados que le siguen influenciados por la propiedad de *Markov*. La *ecuación de Bellman* que se muestra a continuación satisface esta condición

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r+\gamma v_\pi(s')], \forall s \in S \tag{2.4}$$

En definitiva, en el entorno de aprendizaje por refuerzo, el objetivo es encontrar la política que de como resultado la mayor recompensa a largo plazo. A la política óptima se la define con el símbolo π_*. De esta manera la función de valores de estados y la función de valores de acciones óptimas se muestran en las siguientes ecuaciones respectivamente y su derivación en la forma de *Bellman*

$$v_*(s) \doteq \max_{\pi} v_\pi(s) = \max_{a} \mathbb{E}_\pi[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \tag{2.5}$$

$$q_*(s,a) \doteq \max_{\pi} q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \tag{2.6}$$

En los procesos de *Markov* se suelen distinguir entre entornos **basados en modelo** o **sin modelo**. Los entornos basados en modelo se caracterizan por que tienen un conocimiento completo de todos los posibles estados S, acciones A, las recompensas inmediatas $r(s, a, s')$, así como de las probabilidades de transición entre estados $p(s'|s, a)$. Por otro lado, los entornos sin modelo se distinguen por no tener ninguno conocimiento previo del entorno y como consecuencia el agente obtiene información de este a través de muestras. El caso de este proyecto el dron simulado representa un entorno sin modelo ya que aprenderá sobre el entorno con ayuda de la información que obtiene con los distintos sensores.

En este proyecto, los algoritmos de aprendizaje por refuerzo están basados en la resolución de problemas sin modelo con **métodos de diferencias temporales**, los cuales se basan en los problemas con modelo de **programación dinámica** junto con los problemas sin modelo de **Monte Carlo**. En la referencia [29] se explican en detalle los dos tipos de problemas, tanto la programación dinámica como Monte Carlo.

2.3.1. Métodos de diferencias temporales

El aprendizaje por métodos de diferencias temporales combina cualidades ventajosas tanto de la programación dinámica como de Monte Carlo. Por un lado, al igual que en Monte Carlo, los métodos de diferencias temporales pueden aprender a partir de los datos recogidos en las interacciones con el entorno y, por lo tanto, no necesitan un conocimiento completo del entorno. Por otro lado, en los métodos de diferencias temporales se actualizan los estimadores de recompensa en base a los estimadores ya aprendidos, como sucede en la programación dinámica.

La función de valores $V(S_t)$ de los métodos de diferencias temporales se actualiza con la recompensa inmediata R_{t+1} y la estimación $V(S_{t+1})$ de la siguiente manera

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.7}$$

El valor entre corchetes representa la diferencia entre la estimación $V(S_t)$ y la estimación mas precisa del próximo estado $R_{t+1} + \gamma V(S_{t+1})$, también designado este valor como *error*.

Aprendizaje de la función de valores de acciones

En el caso de los problemas de diferencias temporales en los que no existe un modelo del entorno, se pueden obtener resultados más eficientes si la función de valores es de acciones, $Q(s, a)$ en vez de estados $V(s)$. Cuando se tiene una función de valores de acciones óptima q_*, no es necesario que el agente haga un estudio de las posibilidades del siguiente estado, sino que es suficiente con tomar una acción a que maximiza la función $q_*(s, a)$. En otras palabras, con una función de valores de acción se tiene en cuenta la recompensa a largo plazo sin necesidad de saber cuales van a ser los estados siguientes o recompensas después de tomar una acción óptima para la tarea de entrenamiento. Para los casos de diferencias temporales esto significa que el agente no necesita conocer nada del entorno previamente al entrenamiento y que puede converger en una política lo suficientemente óptima.

Explotación y exploración

En las aplicaciones de aprendizaje por refuerzo uno de los mayores retos se presenta en compensar la **exploración** con la **explotación** de la interacción agente-entorno. Para poder definir una política que maximice la recompensa final, el agente de aprendizaje por refuerzo debe de basarse en su experiencia y acciones conocidas en cada estado, las cuales son efectivas en generar recompensas altas. Mientras tanto, primeramente, restas decisiones o acciones efectivas deben ser descubiertas tomando acciones en estados que todavía no habían sido exploradas. Esto significa que el agente debe *explotar* lo que ya se conoce del entorno para conseguir buenas recompensas, sin embargo, también tiene que *explorar* para encontrar mejores posibles acciones.

Con y libre de política

El dilema entre explotación y exploración intenta buscar la política óptima, sin embargo buscar una política óptima es complicado ya que no se puede saber con antelación cual es esta política si no se explora y tan solo se explotan las acciones con buena recompensa en el primer episodio. Para solucionar este problema se proponen dos enfoques: con política y **política libre**. En los métodos con política, usa una única política que se compromete a que el agente siempre tenga momentos de exploración y por lo tanto trata de buscar la mejor política pero que siga explorando. Este método no puede llegar a alcanzar políticas completamente óptimas debido al aspecto de exploración. Por otro lado, los métodos de política libre evitan estos problemas usando dos tipos de políticas. Una política encargada de aprender y convertirse en la óptima final, mientras que la otra que se enfoque en explorar y escoger las acciones. La política que se encarga del aprendizaje se denomina **política objetivo** π, y la política encargada de las tomas de decisiones

o del comportamiento del agente **política de comportamiento** μ. Se llama de política libre ya que el agente está aprendiendo a partir de la experiencia, que está *"libre"* de la política objetivo. Los algoritmos implementados en este proyecto siguen métodos de política libre.

ϵ-Greedy: política de comportamiento

Entre las políticas de comportamiento la más característica y la que se aplicará a lo largo de este proyecto es la política **ϵ-greedy**. La exploración del agente es controlada por el parámetro ϵ, el cual determina la probabilidad de escoger una acción aleatoria o empleando explotación en un paso t del episodio. La gran ventaja de esta política de comportamiento con respecto a otras es que no requiere de memorizar información sobre la exploración, por lo que se puede emplear para espacios de estados muy grandes o incluso continuos.

$$\mu(S_t) = \begin{cases} argmax_a Q(S_t, A) & \text{probabilidad 1-}\epsilon \\ aleatorio(A) & \text{probabilidad } \epsilon \end{cases} \tag{2.8}$$

2.3.2. Descripción de los agentes

Los algoritmos que se han implementado en este proyecto todos tienen en común que se han basado en métodos de diferencias temporales y por lo tanto su interacción de agente-entorno debe cumplir la estructura del proceso de decisión de *Markov*. De esta manera, todos los entornos de los algoritmos de aprendizaje por refuerzo cumplen con los componentes de *Markov* y sus **espacios de acciones** y la **asignación de recompensas**, sin embargo el **espacio de observaciones** será distinto para los algoritmos de aprendizaje tabular que para los de aprendizaje profundo, ya que en los primeros se lidiará con un espacio de observaciones mucho más pequeño. En los aprendizajes tabulares se describe en formato de tabla todas las posibles combinaciones estado/acciones posible, y se asigna un valor real a la intersección de fila(estado) con columnas (acción). La tabla se denomina matriz Q. Cundo no es posible desarrollar esta tabla por la dimensionalidad del problema, se sustituye por una función que calcula el valor de Q por cada pareja estado/acción. Esta función consiste en una función paremetrizada que aprende ajustando sus pesos como lo haría una red neuronal.

En las próximas secciones se describen las acciones, observaciones y recompensas que el dron de la aplicación recoge para definir el proceso de decisión de *Markov*.

Espacio de acciones

El espacio de acciones que el dron puede tomar será discreto en todos los algoritmos implementados y estas acciones serán tomadas únicamente sobre el plano horizontal del dron. Es decir, el dron ascenderá a una cierta altura en el despegue de 2 metros aproximadamente para mantener esta altura constante durante el episodio. No se dotó de libertad de movimiento al dron en el eje z ya que se hubiese generado un espacio de observaciones excesivamente grande para los medios de computación que se disponían en el proyecto.

Las acciones que el dron puede tomar en el espacio se muestran en el Cuadro 2.3 y Figura 2.10. Las acciones que puede tomar son un desplazamiento lineal sobre el eje x a una velocidad de 1 metro por segundo durante un segundo, es decir, el dron recorre 1 metro en total hacia delante, y dos rotaciones sobre el eje z, también conocido en ingles como "*yaw*", de 30° tanto a izquierda como a derecha. En total, el dron se puede encontrar en 12 orientaciones distintas con respecto al origen.

Cuadro 2.3: Espacio de acciones del dron

Acción	Valor
Hacia delante	1 m/s
Derecha	30°
Izquierda	30°

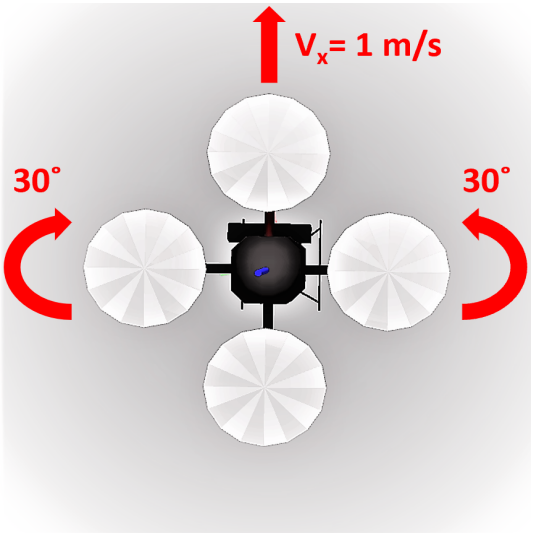


Figura 2.10: Acciones dron Hector

Espacio de observaciones

Las observaciones que el dron puede recoger dependerán de los sensores que se usen para obtener los datos que alimentan a los algoritmos de aprendizaje por refuerzo. Estas observaciones definen el espacio de estados en el que el dron se puede encontrar siguiendo el modelo de Markov. De esta manera, el espacio de observaciones varia entre los algoritmos de aprendizaje tabular y los de aprendizaje profundo. En el primero se usará información sobre los sensores IMU y láser, mientras que en el aprendizaje profundo se empleará la información de estos mismo sensores y se incluirá una imagen de profundidad para que la procese la red neuronal. En los siguientes puntos se describe en mas profundidad como se definen las observaciones tanto para el aprendizaje tabular como para el profundo.

- En los algoritmos de aprendizaje por refuerzo tabular (*Q-learning* y *Double Q-learning*), el espacio de observaciones tiene que ser finito, ya que, la tabla de funciones de valores de acciones, como se verá en la sección **2.3.3. Aprendizaje por refuerzo tabular**, tiene un rango de estados designado con saltos discretos. Por lo tanto, cada estado se definirá de la siguiente manera: $[láser1, láser2, láser3, \phi, d]$. Como se puede comprobar cada estado esta definido por un grupo de observaciones de los sensores del dron *Hector*. El sensor láser recoge las observaciones **láser1**, **láser2** y **láser3**, las cuales corresponden a la distancia que miden tres direcciones del sensor láser, uno de ellos paralelo al eje x del dron, y dos direcciones que forman ángulos de $16°$ y $-16°$ con respecto al eje x. El resto de las direcciones láser se emplearán para determinar si el dron colisiona con un obstáculo. Además, es necesario discretizar las medidas que se obtienen de las tres direcciones láser para que el rango de valores que representen el espacio de estados no sea excesivamente grande. Para la discretización del láser se le asigno una distancia máxima de medición de 6 metros y el rango de medidas se dividió en 6 posibles valores o secciones, es decir, con una resolución de 1 metro, como se muestra en la Figura 2.11.

  Seguidamente, en la estructura de los estados se encuentran las observaciones adquiridas por el sensor IMU, ϕ y $\boldsymbol{d}$. ϕ, representa el ángulo relativo entre el eje x del dron y la posición objetivo en el mapa. Esta observación también es discretizada con una resolución de $30°$ y, por lo tanto, pudiendo tomar 12 valores distintos, ya que, el ángulo máximo que puede tomar es de $360°$. La siguiente ecuación muestra como se calcula el ángulo ϕ, con los parámetros $p_{x,y}$ que representa la posición del dron en los ejes x e y, $obj_{x,y}$ que da a conocer la localización del objetivo, y ψ que representa el ángulo "yaw"del dron.

$$\phi = [atan2(obj_y - p_y, obj_x - p_x) - \psi]/30^\circ \tag{2.9}$$

Finalmente, el valor de d es la distancia euclídea entre la posición actual del dron y la posición objetivo. Al igual que el resto de observaciones, el valor de la distancia también es discretizado tomando un numero entero que representa la distancia con unidades en decímetros y cuyo valor máximo es la diagonal del mapa cuadrado, 282 metros. Des esta manera, el espacio de observaciones del dron en los algoritmos de aprendizaje por refuerzo tabular el numero de estados posibles es de $6^3x12x282 = 730944$.

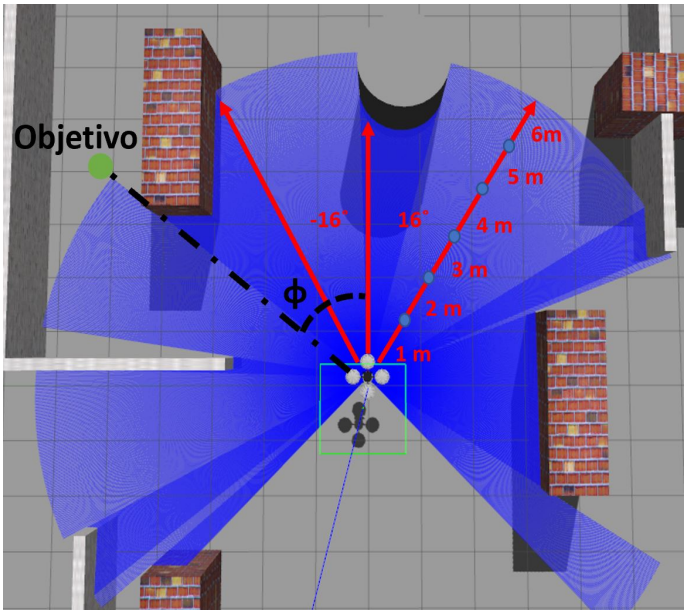


Figura 2.11: Espacio de observaciones en algoritmos de aprendizaje por refuerzo tabular

- Por otro lado, en los algoritmos de aprendizaje por refuerzo profundo el espacio de estados se sustituirá por observaciones continuas. Como se describirá en la sección **2.3.4. Aprendizaje por refuerzo profundo**, al proceso de decisión de *Markov* se le puede introducir un espacio de estados continuo al combinarlo con redes neuronales. Este espacio de estado continuo se definirá mediante imágenes de profundidad tomadas por la cámara *ASUS Xtion Pro*, mejorando la ambigüedad generada por los espacios discretos. Las imágenes que recoge esta cámara tienen un tamaño de 640x480 y los valores de los

píxeles están comprendidos entre 0 y 255 en la escala de grises. Dependiendo de la profundidad de los objetos que se detectan en la imagen, el color blanco será más intenso para los objetos lejanos que para los más cercanos, los cuales se verán de tonalidad negra. Para reducir el tamaño de los datos introducidos en la red neuronal, la imagen de profundidad se procesa de tal forma que su tamaño se comprime a una imagen de 20x100. La compresión de la imagen se realiza tomando la sección central de la imagen de profundidad obtenida por la cámara y cambiando la escala al tamaño deseado con la librería *OpenCV*[2]. Adicionalmente, los píxeles se normalizaron a valores entre 0 y 1, y para generar más contraste en los colores, se sustituyeron los píxeles con valor 0 (*negro*) por 1 (*blanco*). Finalmente, para suministrar mas información a la red neuronal sobre los estados, a la imagen comprimida se le anexiono otra imagen de 10x100 en la parte superior que define el ángulo relativo del dron con respecto al objetivo como se muestra en la Figura 2.12 y Figura 2.13. El ángulo se representa sobre la imagen blanca como una sección de esa misma imagen de tamaño 3x10 con color negro. Para desarrollar esta forma de definir los estados a partir de una imagen de profundidad se tomó como referencia [21].

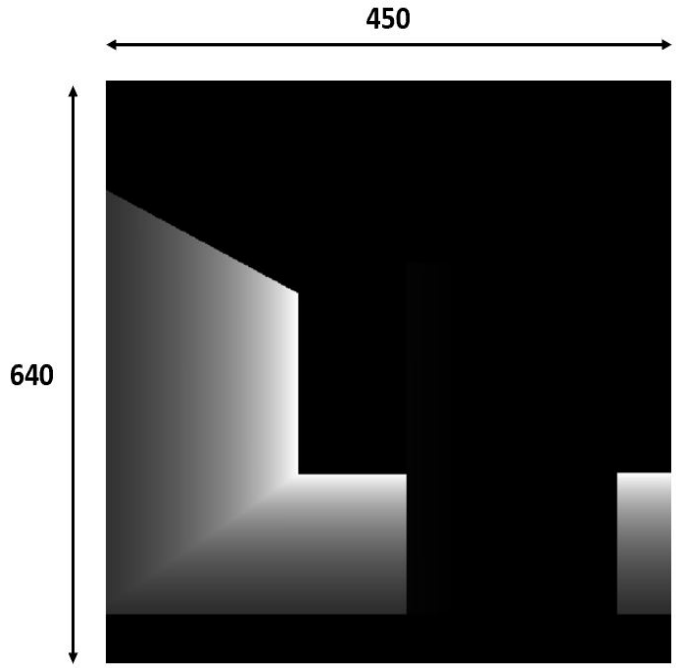


Figura 2.12: Imagen de profundidad con cámara *ASUS Xtion Pro*

[2]OpenCV:https://opencv.org/

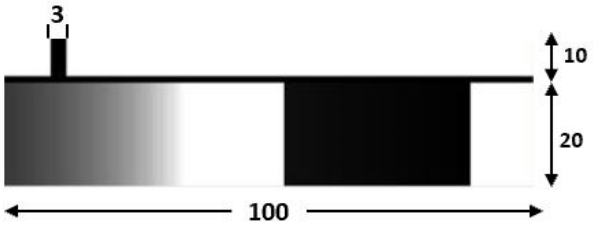


Figura 2.13: Representación del estado con observación de la imagen de profundidad

Asignación de recompensas

La función que asigna las recompensas es idéntica para todos los algoritmos de aprendizaje por refuerzo, al igual que la determinación de la finalización de un episodio. Es importante que la función de recompensas modele de forma correcta el objetivo del aprendizaje. De esta manera, la recompensa por llegar al objetivo sera de 100, mientras que si colisiona o se aproxima demasiado a un obstáculo el entorno otorgara una recompensa adicional de -100 en la finalización. Además, al tratar de buscarse un camino óptimo que se recorra en el menor tiempo posible, hay que penalizar las acciones que no recorran espacio. Por lo tanto, las acciones de desplazamiento angular sobre el eje z sumarán una recompensa de -1 a la total, y en cada paso del episodio se asignará otra recompensa cuyo valor corresponde con la distancia euclídea con la que el dron se acerca o aleja del objetivo medida en metros, Δd.

$$\text{recompensa} = \begin{cases} +100 & \text{dron llega al objetivo y termina episodio} \\ -100 & \text{dron colisiona y termina episodio} \\ -1 & \text{dron toma acción de girar izquierda o derecha} \\ \pm\Delta d & \text{dron se acerca o aleja del objetivo} \end{cases} \tag{2.10}$$

Finalmente, se deben definir las condiciones para que un episodio concluya. En el entorno descrito pueden ocurrir dos resultados, una colisión con un obstáculo o que el dron llegue al destino marcado. En primer lugar, las colisiones se identificarán con las mediciones láser del sensor *Hokuyo UTM-30LX*. Estas medidas

son tomadas desde el ángulo $-135°$ hasta $135°$ con respecto al eje x del dron y con una resolución de $1°$, es decir, se tomarán un total de 270 mediciones. En la Figura 2.11 se muestra en azul el abanico de direcciones láser que el sensor mide, mientras que en la Figura 2.14 se aprecia la nube de puntos que generan las mediciones láser. Si una de estas mediciones baja del valor de 0,6 metros, se considera que el dron ha colisionado y se inicializa un episodio nuevo. En segundo lugar, se le otorga un margen de error al dron para alcanzar el objetivo. Este margen de error se define mediante un radio de 1 metro entorno a la posición objetivo, si el dron entra dentro de la circunferencia delineada se considera que el episodio termina satisfactoriamente y se continua el entrenamiento con el próximo episodio.

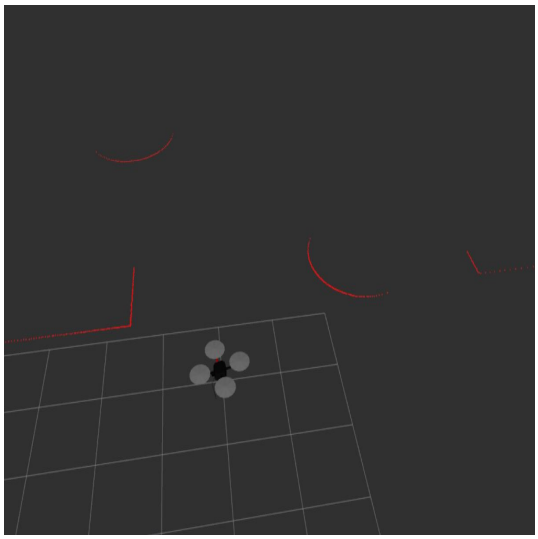


Figura 2.14: Nube de puntos generada por mediciones láser

2.3.3. Aprendizaje por refuerzo tabular

El aprendizaje por refuerzo tabular es el más básico dentro de los métodos de diferencias temporales. Entre estos algoritmos de aprendizaje se encuentran *Q-learning* y *Double Q-learning*. Se denomina tabular ya que la función de valores $Q(S, A)$ se describe en un formato de tabla como en la Figura 2.17. En esta tabla las columnas representan las posibles acciones del dron (recto, izquierda y derecha), mientras que las filas representan todos los posibles estados del dron ($[láser1, láser2, láser3, \phi, d]$) descritos en la sección **2.3.2. Descripción de los agentes**. Los vaores de los estados tienen un rango de $[0, 0, 0, 0, 0]$ a $[6, 6, 6, 12, 282]$. En la intersección de las columnas y filas se registra el **valor Q**, el cual define la probabilidad de cuan bueno es tomar dicha acción en dicho estado, permitiendo

definir una política de aprendizaje. A continuación se describirán las dos variantes de este tipo de algoritmos y como la tabla de Q realiza el aprendizaje en el entorno de este proyecto.

Q-Table		Acciones (A)		
				
ESTADOS (S)	00000	0	0	0
	.	.	.	.
	.	.	.	.
	.	.	.	.
	6661228.2	0	0	0

Figura 2.15: Tabla Q

Q-learning

Q-learning es un método de aprendizaje por refuerzo tabular sin política. Esto quiere decir que la acción tomada en cada momento A_t es escogida en base a la política de comportamiento μ (ϵ-greedy), esta acción es escogida aleatoriamente en cada estado y no tiene porque ser la óptima. Por otro lado, *Q-learning* también considera otra posible acción A' en cada estado siguiendo la política objetivo π, esta acción sera la que mas recompensa se estima que puede otorgar en cada estado, es decir, la que mas valor $Q(S_t, A_t)$ tenga en la tabla Q. El algoritmo de control que actualiza la tabla Q tras tomar una acción A_t, se define con la derivación de la ecuación de *Bellman* siguiente

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.11}$$

$Q(S_t, A_t)$ representa el valor de la intersección estado actual y acción tomada en la tabla Q, α (tamaño de paso) es un valor comprendido entre 0 y 1 que especifica la importancia que se da a la nueva información con respecto a la ya almacenada, R_{t+1} representa la recompensa adquirida tras tomar la acción seleccionada en el estado actual, γ (factor de descuento) es un parámetro en 0 y 1 que especifica

la importancia que se da a las recompensas en el futuro con respecto a las instantáneas, y finalmente, $\max_a Q(S_{t+1}, a)$ escoge el valor Q óptimo de la tala Q del estado en el que se encuentra el dron S_{t+1} tras tomar la acción A_t.
A continuación se define el pseudocódigo del algoritmo empleado.

Algorithm 1 Q-learning

Inicializar $Q(s, a)$ aleatoriamente
Inicializar factor de descuento γ
Inicializar tamaño de paso α
while (episodio):
    Observar estado inicial S_t
    while (paso):
        Escoger acción A_t derivada de la política de Q (e.g. ϵ-greedy)
        Tomar acción A_t
        Recoger recompesa R_{t+1} y observar nuevo estado S_t
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
        $S_t \leftarrow S_{t+1}$

Double Q-learning

En [30] se justifica por que la implementación de algoritmos *Q-learning* en algunos entornos estocásticos no resultan efectivos. Como solución a los problemas que pueden surgir se plantea una variante de este, llamada *Double Q-learning*. El mayor de estos problemas es la sobreestimación de los valores de las acciones al usar $\max_a Q(S_{t+1}, a)$.

Como solución, el *Double Q-learning*, propone el aprendizaje de dos funciones de valores o tablas Q, Q^A y Q^B. Ambos se actualizan compartiendo sus valores entre ellos. De esta manera, buscar la acción A_{t+1} que maximice Q^A en el próximo estado, es decir, $Q(S_{t+1}, A_{t+1}) = \max Q(S_t, A_t)$. Para posteriormente usar A_{t+1} para obtener el valor de $Q^B(S_{t+1}, A_{t+1})$ con el objetivo de actualizar $Q^A(S_t, A_t)$.
A continuación se muestra el pseudocódigo del aprendizaje *Double Q-learning* implementado en el proyecto.

Algorithm 2 Double Q-learning

Inicializar Q^A y Q^B aleatoriamente
while (episodio):
    Elegir accion A_t en función de $Q^A(S_t, .)$ y $Q^B(S_t, .)$
    Elegir (e.g. aleatoriamente) entre **ACTUALIZAR(A)** o **ACTUALIZAR(B)**
    si ACTUALIZAR(A) **entonces**
        $a^* = \text{máx}_a\, Q^A(S_t, a)$
        $Q^A(S_t, A_t) \leftarrow Q^A(S_t, A_t) + \alpha[R_{t+1} + \gamma Q^B(S_{t+1}, a^*) - Q^A(S_t, A_t)]$
    si ACTUALIZAR(B) **entonces**
        $b^* = \text{máx}_a\, Q^B(s', a)$
        $Q^B(S_t, A_t) \leftarrow Q^B(S_t, A_t) + \alpha[R_{t+1} + \gamma Q^A(S_{t+1}, b^*) - Q^B(S_t, A_t)]$
    $S_t \leftarrow S_{t+1}$

2.3.4. Aprendizaje por refuerzo profundo

Cuando se combinan las redes neuronales con el aprendizaje por refuerzo aparecen otros algoritmos denominados de aprendizaje por refuerzo profundo. En los algoritmos tabulares uno de sus inconvenientes es la discretización del entorno, tanto de las acciones como de los estados. Esto genera cierta inestabilidad y divergencias en el aprendizaje. Además, para los espacios de estado excesivamente grandes debido a la alta carga de datos de los sensores, las redes neuronales permiten definir un espacio de estados continuo. Los datos que representan el espacio de estado suelen ser imágenes, por lo que las redes neuronales convolucionales (CNN) son de gran importancia en el aprendizaje profundo. Por lo tanto, para implementar Q-learning con un espacio continuo se aplican redes neuronales al entorno de aprendizaje dando lugar a los algoritmos que se describirán a continuación, *Deep Q-learning* y *Double Q-learning*.

Los algoritmos de aprendizaje por refuerzo profundo se implementaron usando la librería *Keras-RL*[3], la cual presenta unos metodos que aplican directamente el proceso de aprendizaje del entorno de OpenAI Gym generado. Para ello, inicialmente se debe definir un modelo de la red neuronal implementado con *Keras*. El modelo de la red neuronal implementada en este proyecto para ambos algoritmos, *Deep Q-learning* y *Double Q-learning*, se muestra en la Figura 2.16 [31]. Esta red neuronal consta de dos entradas paralelas, una capa de 30x100 por la cual se procesara la imagen de profundidad, y otra entrada de 1x1 por la cual se introducirá la distancia que hay del dron al objetivo. Las capas de la red neuronal que procesan

[3]Keras-RL: https://github.com/keras-rl/keras-rl

la imagen inicialmente son una capa *Convolucional* con 32 filtros 4x4 de paso 4, y otra capa *Convolucional* con 64 filtros 3x3 con paso 2. Ambas entradas convergen para ser procesadas por 3 capas *Densas* con 256 unidades. Finalmente, la última capa corresponde con una *Densa* de 3 salidas, una por cada acción. En total en la red neuronal implementada se definen un total de 885155 parámetros que deben ser determinados durante el aprendizaje.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1, 30, 100)	0	
conv2d_1 (Conv2D)	(None, 32, 7, 25)	544	input_1[0][0]
conv2d_2 (Conv2D)	(None, 15, 3, 64)	14464	conv2d_1[0][0]
input_2 (InputLayer)	(None, 1, 1)	0	
flatten_1 (Flatten)	(None, 2880)	0	conv2d_2[0][0]
reshape_1 (Reshape)	(None, 1)	0	input_2[0][0]
concatenate_1 (Concatenate)	(None, 2881)	0	flatten_1[0][0] reshape_1[0][0]
dense_1 (Dense)	(None, 256)	737792	concatenate_1[0][0]
dense_2 (Dense)	(None, 256)	65792	dense_1[0][0]
dense_3 (Dense)	(None, 256)	65792	dense_2[0][0]
dense_4 (Dense)	(None, 3)	771	dense_3[0][0]

Total params: 885,155
Trainable params: 885,155
Non-trainable params: 0

Figura 2.16: Estructura de la red neuronal implementada en *Deep Q-learning* y *Double Q-learning*

Deep Q-learning

En las técnicas de aprendizaje por refuerzo profundo las redes neuronales que se desarrollan constan de una entrada que representara el espacio de estados, así como de tantas salidas como acciones se puedan tomar. De esta manera, la última capa de salida de la red neuronal corresponde con los valores Q de cada acción que la red neuronal predice para cada estado introducido en la capa de entrada. Seguidamente la política seleccionará la acción con el valor Q más alto. Como se especificó anteriormente, todo ello se puede computar gracias a la abstracción de las imágenes y a dos redes convoluciones de la red neuronal aplicadas en este proyecto. La red neuronal actualizará después de una serie de episodios o de cada uno de los parámetros que definen sus funciones internas parametrizadas. Estos parámetros se designan como θ y se actualizan tras tomar la acción A_t, en el estado S_t y recogiendo las observaciones de la recompensa R_{t+1} y el estado S_{t+1} con la ecuación

$$\theta = \theta_t + \alpha[\mathcal{L}_t^Q - Q(S_t, A_t; \theta_t)]\nabla_{\theta_t} Q(S_t, A_t; \theta_t) \tag{2.12}$$

donde $\mathcal{L}_t^Q$ representa la recompensa estimada de la siguiente manera

$$\mathcal{L}_t^Q = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta) \tag{2.13}$$

Esta forma de actualización se asemeja al descenso por gradiente, actualizándose el valor $Q(S_t, A_t; \theta_t)$ sobre el error de diferencia temporal enfocado hacia un valor objetivo $\mathcal{L}_t^Q$.

En ocasiones, el entrenamiento de las redes neuronales no puede ser efectivo al generarse demasiada correlación entre las muestras tomadas de los estados. Para solucionar esta correlación se introduce el concepto de **repetición de experiencia**. Con este concepto se crea una **memoria de repetición** donde se guarda el estado que observa el dron en cada paso del episodio. Las observaciones se identifican como experiencia $e_t(S_t, A_t.R_{t+1}, S_{t+1})$ y se guardan en la memoria de repetición $D\{e_1, e_2, ..., e_N\}$, siendo N el limite de la memoria. Cuando el limite de la memoria es alcanzado se sustituirán las nuevas experiencias por las antiguas. Esta memoria de repetición permite entrenar la red neuronal con paquetes formados por imágenes adquiridas directamente del entorno mientras que otras serán extraídas de la memoria, proporcionando cierta aleatoriedad en las observaciones

de entrenamiento y evitando su correlación.

Por otro lado, el error de diferencia temporal en Q-learning no es estacionario, es decir, a medida que la función objetivo va aprendiendo los valores de esta cambian a la par. Cuando se implementan redes neuronales esto puede causar divergencia al aproximar la función Q, ya que cada paso de gradiente puede cambiar los valores de dos pares acción-estado que no se encuentre relacionados entre sí. Esta inestabilidad se puede mitigar introduciendo una segunda instancia de la red neuronal denominada **red objetivo $\hat{\mathbf{Q}}$**. Los parámetros θ^- de la red neuronal objetivo se actualizan con los parámetros θ de la red neuronal Q cada X pasos. De esta manera, el objetivo de *Deep Q-learning* se aprendera con

$$\mathcal{L}_t^Q = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta^-) \tag{2.14}$$

Finalmente, en vez de actualizarse la red objetivo cada X pasos, se define el concepto de **actualización de objetivo leve**. En este concepto los pesos o parámetros de la red neuronal se actualizan haciéndoles seguir moderadamente la red neuronal ya aprendida en cada paso dependiendo del **factor objetivo**, t << 1.

Double Deep Q-learning

Al igual que en el algoritmo de *Double Q-learning* en el que dos funciones de valores de acciones Q^A y Q^B realizan el aprendizaje, en los algoritmos de *Double Deep Q-learning* se instancian y actualizan dos grupos de pesos θ y θ'. Como en *Double Q-learning*, existen dos funciones objetivo que se actualizarán de la siguiente manera

$$\mathcal{L}_t^{Q_1} = R_{t+1} + \gamma\theta_2(S_{t+1}, \max_a Q_1(S_{t+1}, a; \theta_t); \theta'_t) \tag{2.15}$$

$$\mathcal{L}_t^{Q_2} = R_{t+1} + \gamma\theta_1(S_{t+1}, \max_a Q_1(S_{t+1}, a; \theta'_t); \theta_t) \tag{2.16}$$

2.4. Diseño y uso de la aplicación

La interfaz gráfica de usuario tiene como función juntar en un mismo sistema los distintos entornos de simulación y algoritmos, para que puedan ser entrenados y comparados entre si de forma intuitiva. Como se especifico en la sección **2.2.2. Integración de la aplicación de aprendizaje por refuerzo**, la interfaz se desarrolló con *PyQt*[4] y se pone en marcha ejecutando un icono en el escritorio igual

[4]PyQt: https://www.riverbankcomputing.com/software/pyqt/

que el que aparece en la Figura 2.9. Una vez, ejecutada la aplicación la interfaz visual principal constará de dos ventanas. La primera ventana que aparece se llama **"Train/Test"** y tiene como función ejecutar y visualizar, individualmente, los resultados de los entrenamientos e inferencias de los distintos algoritmos de aprendizaje por refuerzo que se han desarrollado. Por otro lado, la segunda ventana, denominada "**Compare**", permite comparar gráficamente los resultados de los distintos entrenamientos e inferencias que se han realizado. A continuación, se describirá con mas detalle el uso y funcionalidad de ambas ventanas.

2.4.1. Ventana 'Train/Test'

La ventana **"Train/Test"** se muestra en la Figura 2.17. Esta ventana permite ejecutar los algoritmos de aprendizaje especificando sus parámetros mas característicos, enumerados de color rojo, así como un seguimiento de los parámetros de aprendizaje durante la simulación mediante gráficas, enumeradas en verde.

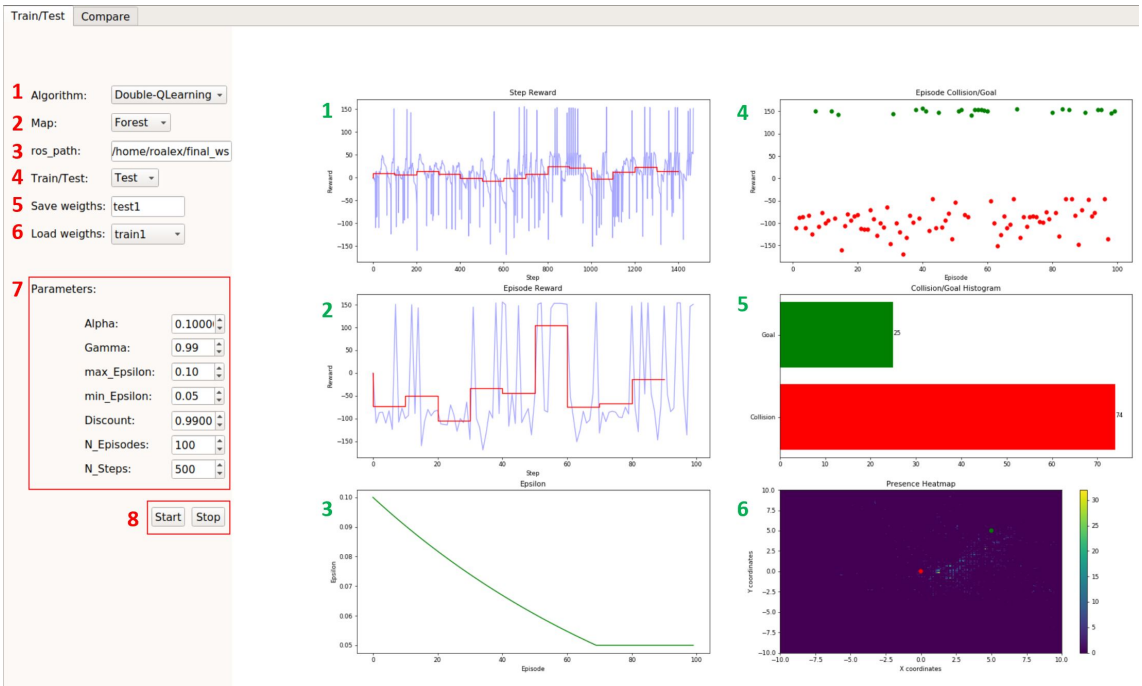


Figura 2.17: Ventana 'Train/Test' de la interfaz gráfica de usuario

A continuación, se describen los parámetros y objetos enumerados en rojo que dan la funcionalidad a la ventana **"Train/Test"**.

1. **Algorithm**: pestaña desplegable que permite seleccionar el algoritmo de

aprendizaje por refuerzo que se va a emplear en el entrenamiento o inferencia. Se puede elegir entre *Q-learning*, *Double Q-learning*, *Deep Q-learning* y *Double Deep Q-learning*.

2. **Map**: en esta pestaña se puede seleccionar entre dos opciones el mapa desarrollado en *Gazebo* donde el dron realizará el aprendizaje. Estos dos mapas se describen en la sección **2.2.2. Integración de la aplicación de aprendizaje por refuerzo**, llamados *Maze* y *Forest*.

3. **ros_path**: en este objeto se especifica el directorio donde se encuentra alojado el espacio de trabajo desarrollado en *ROS*.

4. **Train/Test**: pestaña desplegable para seleccionar entre entrenamiento, *Train*, o inferencia, *Test*, en la ejecución del algoritmo seleccionado.

5. **Save weights**: salva en un fichero con el nombre especificado los valores y parámetros aprendidos durante el entrenamiento.

6. **Load weights**: inicializa el entrenamiento o inferencia con los parámetros del fichero seleccionado en la pestaña desplegable. Estos ficheros corresponden con los guardados en el parámetro *Save weights*.

7. **Parameters**: a continuación se especifican y describen los hiperparámetros de los algoritmos de aprendizaje modificables en la interfaz.

    - **Alpha**: parámetro α que especifica la velocidad de aprendizaje.
    - **Gammma**: parámetro γ que especifica el factor de descuento.
    - **max_Epsilon**: máximo valor de ϵ durante el entrenamiento.
    - **min_Epsilon**: mínimo valor de ϵ durante el entrenamiento.
    - **Discount**: descuento aplicado a ϵ al final de cada episodio.
    - **N_Episodes**: máximo numero de episodios en el entrenamiento/inferencia.
    - **N_Steps**: máximo número de pasos en cada episodio.
    - **Target Factor**: factor objetivo para actualizar la red neuronal.
    - **Memory Size**: especifica el tamaño de la memoria de observaciones de la red neuronal.
    - **Batch Size**: especifica el tamaño de los paquetes de la red neuronal.

8. **Start/Stop**: el pulsador *Start* da comienzo a al entrenamiento/inferencia del algoritmo y parámetros seleccionados y su simulación, mientras que el pulsador *Stop* se finaliza el aprendizaje.

En esta misma ventana, también se muestran gráficas con información sobre la simulación de aprendizaje que se este llevando acabo. Estas gráficas se enumeran en verde en la Figura 2.17 y se describen a continuación.

1. **Step Reward**: publica la recompensa que se va acumulando por cada acción que toma el dron. En azul claro se representa cada acción individualmente y en rojo se representa la media de 100 acciones consecutivas para visualizar mejor la tendencia de la gráfica.

2. **Episode Rewar**: publica la recompensa de cada episodio. En azul claro se representa la recompensa en cada episodio individualmente, mientras que la gráfica roja representa la media de 10 episodios consecutivos.

3. **Epsilon**: esta gráfica representa como decae el parámetro ϵ de exploración/explotación.

4. **Episode Collision/Goal**: cada punto representa el valor de la recompensa de un episodio y si su terminación fue debida a una colisión (rojo) o a que alcanzo el destino (verde).

5. **Collision/Goal Histogram**: histograma que refleja el numero de episodios que terminan debido a colisión a destino alcanzado.

6. **Presence Heatmap**: mapa de densidad en el que se representa en una escala de color las veces que el dron ha estado en una posición determinada.

2.4.2. Ventana 'Compare'

La ventana "**Compare**"se muestra en la Figura 2.18. Esta ventana permite seleccionar los entrenamientos e inferencias que se han realizado previamente mediante los objetos gráficas enumerados en rojo. Posteriormente, de estos aprendizajes seleccionados se realiza una comparación de sus resultados en las gráficas enumeradas de color verde.

Los objetos gráficos enumerados en rojo se describen a continuación.

1. **Add file to compare**: se selecciona de una lista uno de los aprendizajes realizados y se añaden a la lista de aprendizajes para comparar pulsando el botón '+'.

2. **Remove file to compare**: se selecciona uno de los aprendizajes de la lista de aprendizaje elegidos para comparar y se elimina de esta pulsando el botón '-'.

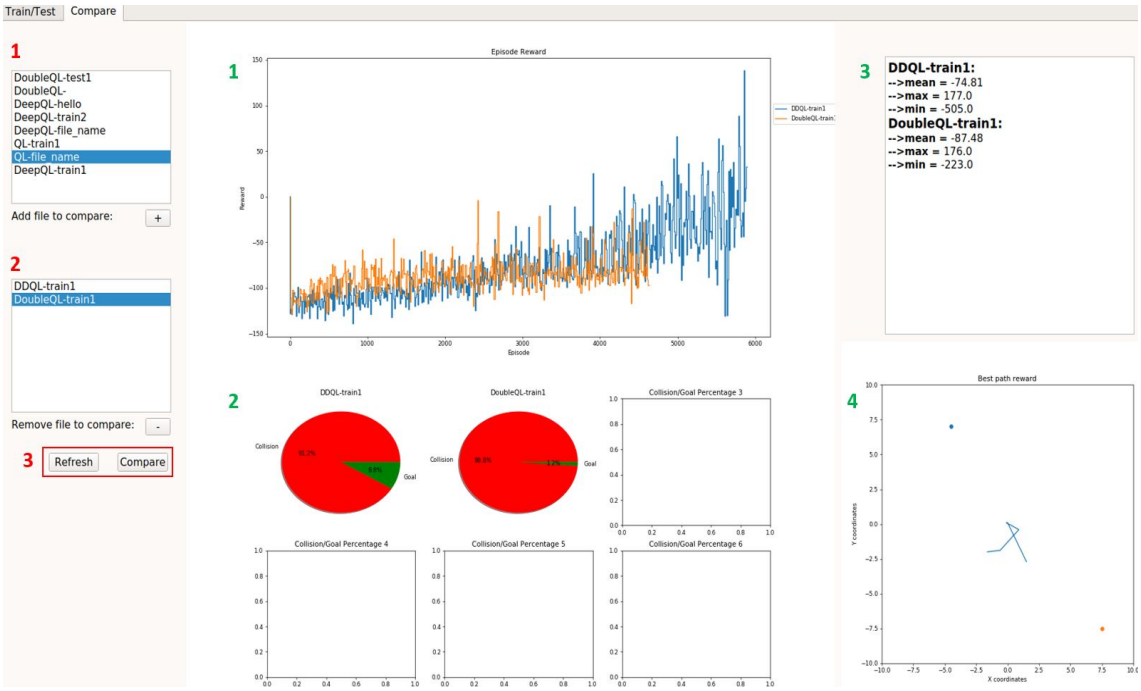


Figura 2.18: Ventana 'Compare' de la interfaz gráfica de usuario

3. **Refresh/Compare**: el pulsador *Refresh* actualizará la lista de aprendizajes en caso de que se haya iniciado uno nuevo, mientras que el pulsador *Compare* iniciará la comparación de los aprendizajes seleccionados mostrando la información en las gráficas.

Finalmente, en los siguientes puntos se describe la funcionalidad de las gráficas enumeradas en verde.

1. **Episode Reward**: se representan la recompensa por episodio de cada uno de los aprendizajes seleccionados para la comparación.

2. **Collision/Goal Percentage**: muestra en diagramas fracciónales el porcentaje de colisiones con respecto a objetivo alcanzado en cada uno de los aprendizajes seleccionados para la comparación. No se pueden seleccionar mas de 6 aprendizajes en cada comparación.

3. En esta lista de texto se muestra la media, así como el valor máximo y mínimo de la recompensa de cada aprendizaje que se está comparando.

4. **Best path reward**: esta gráfica dibuja el camino hacia el objetivo con el que mas recompensa haya obtenido el aprendizaje a comparar.

Capítulo 3

Resultados

En este capítulo se realizara una prueba del funcionamiento de la aplicación. Para ello se dará uso de la interfaz gráfica de usuario diseñado, en la que se ejecutarán los distintos algoritmos de aprendizaje por refuerzo desarrollados en el entorno de simulación del dron *Hector*. En primer lugar, se analizaran los resultados de las gráficas de la interfaz gráfica del entrenamiento e inferencias de cada algoritmo individualmente. Finalmente, se llevara acabo una comparación de los algoritmos entre sí. La comparación de los algoritmos se dividirá en algoritmos tabulares y profundos, ya que utilizan un espacio de estados distintos, ya que el aprendizaje tabular presenta una clara desventaja frente al profundo.

3.1. Entrenamiento e Inferencia

Para que el mapa donde se realice el entrenamiento no influencie la parcialidad de las inferencias, el entrenamiento y las inferencias se realizarán en dos mapas distintos. Para los entrenamientos se empleará el mapa *Maze*, mientras que para comprobar la eficiencia del aprendizaje las inferencias se realizarán en el mapa *Forest*.

3.1.1. Q-learning

Para realizar el entrenamiento del algoritmo *Q-learning* se especificaron los hiperparámetros del Cuadro 3.1, designando un único objetivo en el mapa *Maze* con coordenadas *x=-6.0*, *y=-7.5*.

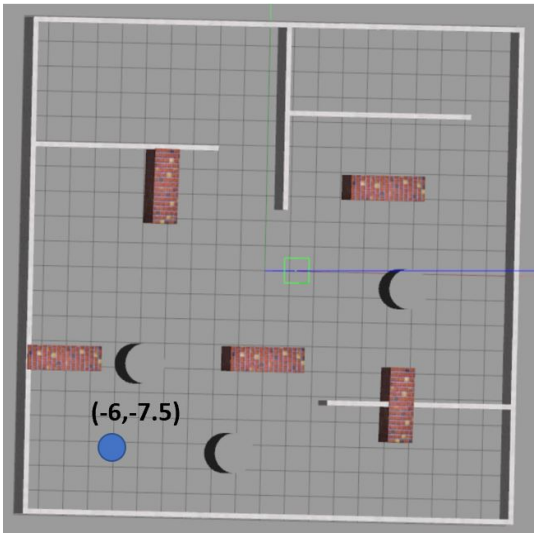


Figura 3.1: Objetivo de entrenamiento Q-learning en mapa *Maze*

Hiperparámetro	**Valor**
Alpha	0.1
Gamma	0.99
max_Epsilon	1
min_Epsilon	0.05
Discount	0.999
N_Episodes	4000
N_Steps	500

Cuadro 3.1: Parámetros de entrenamiento Q-learning

En la Figura 3.2 se muestra en rojo la media de 10 recompensas por episodio. En esta gráfica se puede apreciar un aprendizaje muy leve, con un valor de recompensa final que se estabiliza en aproximadamente -80. Como se puede comprobar este tipo de aprendizaje no es muy óptimo, aunque se puede apreciar que el dron si es capaz de llegar en ciertas ocasiones al objetivo durante la parte final del aprendizaje. En la Figura 3.3 se muestra el número de veces que alcanza el objetivo con respecto al número de veces que el episodio termina en colisión. Por otro lado, en el mapa de calor de la Figura 3.4 se puede observar cierta tendencia del dron a navegar hacia el objetivo. Esto se puede deber a que el dron es capaz de direccionarse hacia el objetivo con ayuda de la observación del ángulo relativo. Sin embargo debido a la excesiva discretización del espacio de estado el dron no es capaz de distinguir ciertos estados críticos entre si y como consecuencia identifica que ambos estados son el mismo. Debido a este inconveniente, el dron desapren-

de lo aprendido y es mas propenso a colisionar. Las acciones discretas también influyen a la hora de aprender en las transiciones entre estados, ya que la acción tomada dentro del espacio de estados puede ser muy pequeña y no modificar el estado actual manteniéndose igual en la siguiente observación realizada por el dron. Además, esta incertidumbre al identificar los estados en los que el dron se encuentra se reflejan en los valores de las recompensas individuales que aparecen en la Figura 3.2 en azul de casi -500. En estos casos el dron se quedaría dando vueltas sobre si mismo sin encontrar una acción óptima debido a la incertidumbre.

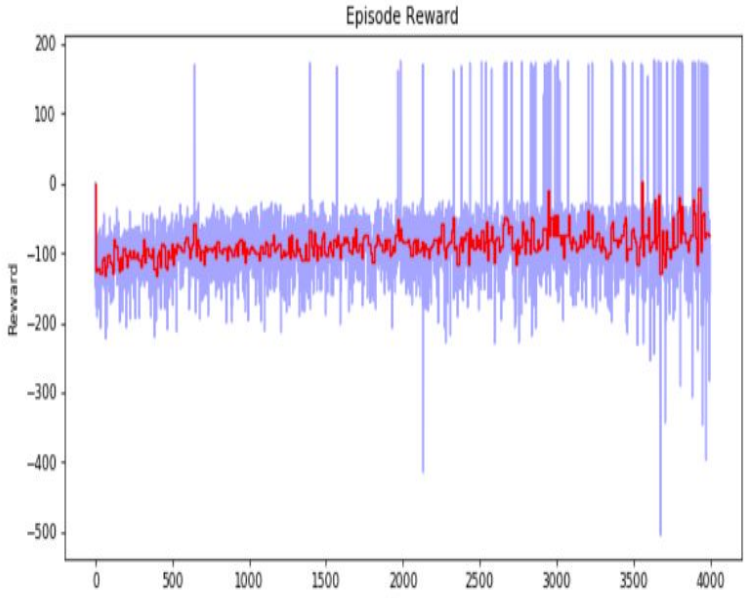


Figura 3.2: Gráfica *Episode/Reward* del entrenamiento Q-learning

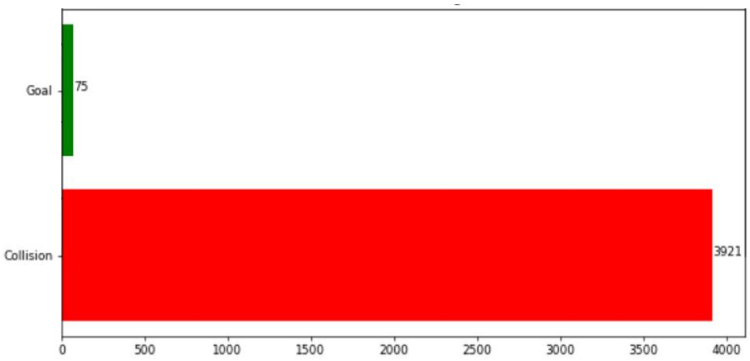


Figura 3.3: Gráfica *Goal/Collision* del entrenamiento Q-learning

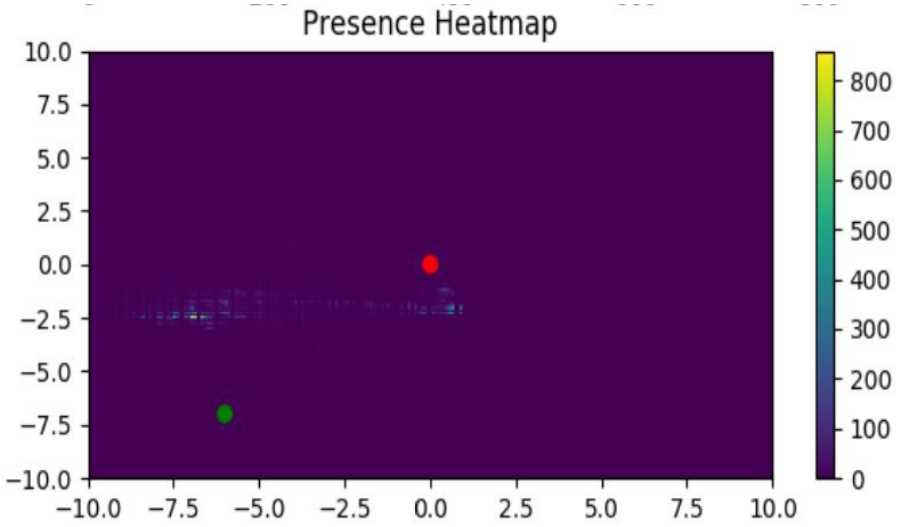


Figura 3.4: Gráfica *Presence Heatmap* del entrenamiento Q-learning

Por otro lado, para comprobar la efectividad del aprendizaje se realizan dos inferencias en el mapa *Forest* con los destinos objetivo, **A** y **B**, marcados en la Figura 3.5. En cada inferencia se realiza un total de 100 episodios. A continuación se muestran los resultados obtenidos en las inferencias.

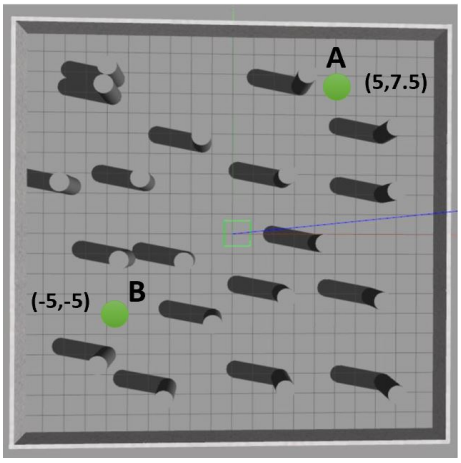


Figura 3.5: Objetivos de inferencia Q-learning en mapa *Forest*

Inferencia A

Las colisiones y episodios satisfactorios se representan en rojo y en verde respectivamente en la Figura 3.6, además, cada punto representa un episodio y su recompensa. Como era de esperar al analizar el entrenamiento previamente, el dron no ha aprendido de forma eficiente la tarea propuesta en el proyecto, en un total de 100 episodios tan solo llega en 12 episodios al objetivo, es decir, aproximadamente un 11,8 %. La gráfica de la Figura 3.6 se puede resumir en el Cuadro 3.2, donde se comprueba que la media de la recompensa de los episodios esta por debajo de 0 (-45,31), pero que, a pesar de errar en la mayoría de los episodios, es capaz de conseguir una recompensa máxima satisfactoria de 174.

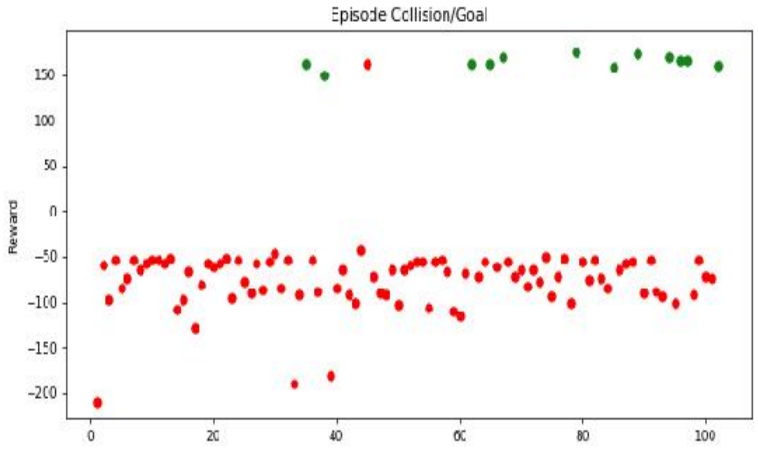


Figura 3.6: Gráfica *Episode Collision/Goal* inferencia A de Q-learning

Parámetro	Valor
Media	-45,31
Máximo	174
Mínimo	-209

Cuadro 3.2: Resultados inferencia A, Q-learning

Inferencia B

El objetivo establecido para la inferencia **B** se encuentra en una zona con un acceso mas complicado que el objetivo **A**, debido al estrechamiento que hay entre los obstáculos de la trayectoria. Como era de esperar, los resultados de esta inferencia son incluso menos eficientes que en la inferencia **A**. Como se muestra en la

Figura 3.23, el dron llega al objetivo nada más que 4 veces de los 100 intentos, dando lugar a una efectividad de la inferencia del 2,8 %.

En el Cuadro 3.10 se puede apreciar que la media de recompensa es mucho mas baja que la de la inferencia **A**, así como un valor máximo y mínimo menor.

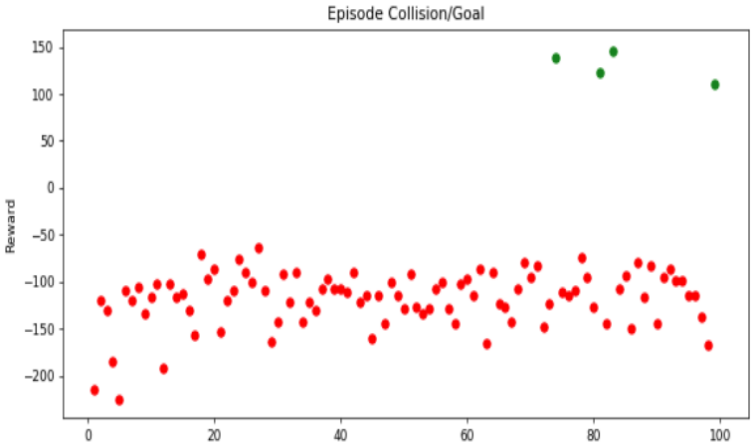


Figura 3.7: Gráfica *Episode Collision/Goal* inferencia B de Q-learning

Parámetro	**Valor**
Media	-105.56
Máximo	144.0
Mínimo	-217.0

Cuadro 3.3: Resultados inferencia B, Q-learning

3.1.2. Double Q-learning

Para el entrenamiento del dron empleando el algoritmo *Double Q-learning* se designaron los mismos hiperparámetros que para *Q-learning* mostrados en la Figura 3.2, así como el mismo objetivo final de la Figura 3.5.

Como se puede observar en la Figura 3.8 el desarrollo del aprendizaje se asemeja al aprendizaje del algoritmo *Q-learning*. En esta gráfica la recompensa se estabiliza muy rápido y con una recompensa de aproximadamente -90. La duración de este entrenamiento fue de 5000 episodios. Hay que destacar también que

en esta gráfica las recompensas individuales en azul no bajan de -200, a diferencia del algoritmo *Q-learning* el cual llega a tener episodios por debajo de los -300 de recompensa. Esto ocurre debido a la actualización de dos tablas de valores Q que presenta *Double Q-learning*. El dron entrenado por *Q-learning* se ve envuelto en episodios sin fin con recompensas exageradamente bajas, ya que solo tiene una función de valores Q para tomar decisiones. Por otro lado, *Double Q-learning* es capaz de aportar mayor aleatoriedad y reducir la parcialidad, ya que alternan el aprendizaje aleatoriamente entre dos tablas de valores Q aportando mayor arbitrariedad en las tomas de decisiones de acciones.

Sin embargo, en la Figura 3.9 se muestra que el dron tan solo llega 56 veces al objetivo de los casi 5000 episodios que dura el entrenamiento.

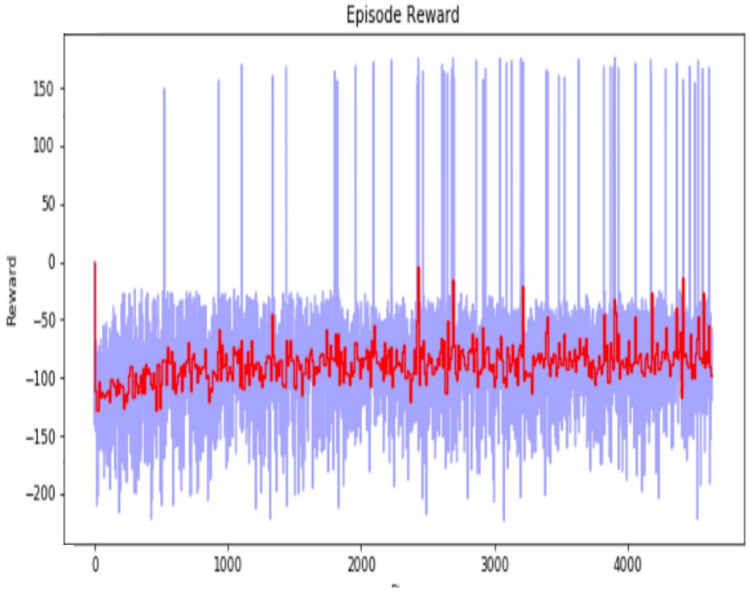


Figura 3.8: Gráfica *Episode/Reward* del entrenamiento Double Q-learning

Al igual que en el entrenamiento de *Q-learning*, en *Double Q-learning* el dron presenta una tendencia a navegar hacia el objetivo pero sin terminar satisfactoriamente el episodio. En el mapa de calor de la Figura 3.10 se puede apreciar esta tendencia.
Finalmente, una vez el dron haya aprendido la tarea de navegación con el algoritmo *Double Q-learning* se comprueba su efectividad mediante las inferencias **A** y **B** con los objetivos mostrados en la Figura 3.5.

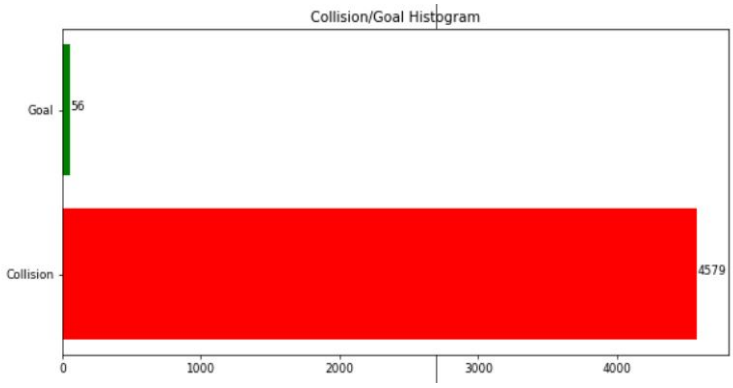


Figura 3.9: Gráfica *Goal/Collision* del entrenamiento Double Q-learning

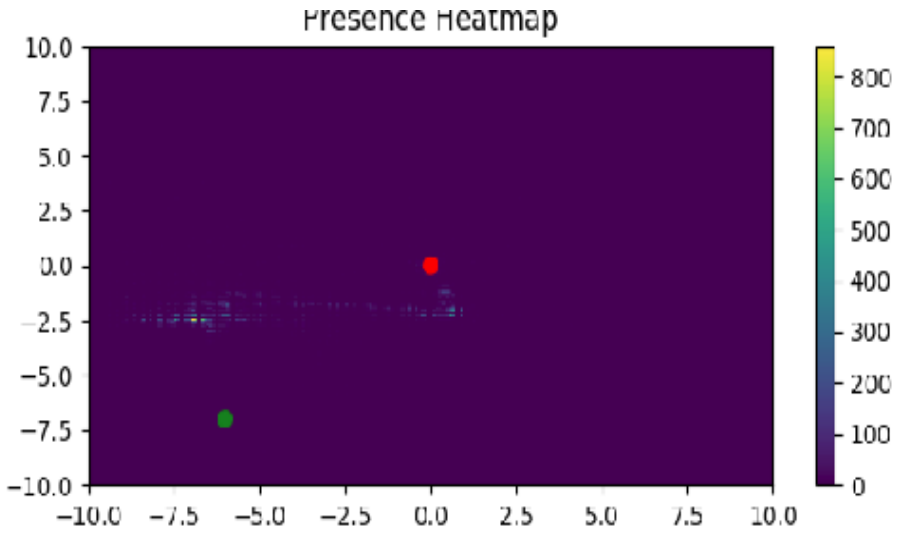


Figura 3.10: Gráfica *Presence Heatmap* del entrenamiento Double Q-learning

Inferencia A

La inferencia **A** del aprendizaje por refuerzo *Double Q-learning* presenta resultados muy similares a la misma inferencia con *Q-learning*. En la Figura 3.11 se interpreta que el dron llega un total de 11 veces al objetivo de 100 episodios de inferencia, es decir, un 11,1 %. La media de las recompensas de los episodios sigue teniendo un valor bajo de -48,85. Sin embargo, como ocurría en los entrenamientos de *Q-learning*, en la Figura 3.12 se puede observar como las posiciones del dron a lo largo de la inferencia toman una trayectoria en dirección al objetivo final.

El problema principal no se trata de la orientación del dron hacia el objetivo, sino de la evasión de obstáculos. Como se especificó en los resultados de *Q-learning*

la utilización de mediciones láser discretas para la observación de los estados puede resultar en una confusión durante el aprendizaje ya que dos estados críticos diferentes pueden ser confundidos con el mismo.

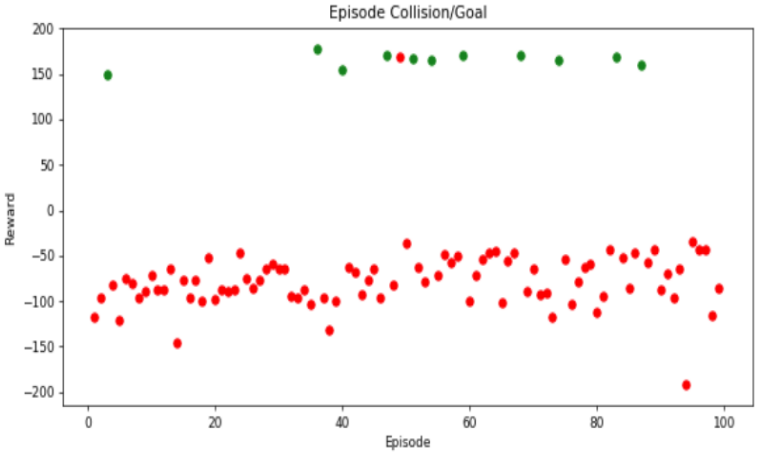


Figura 3.11: Gráfica *Episode Collision/Goal* inferencia A de Double Q-learning

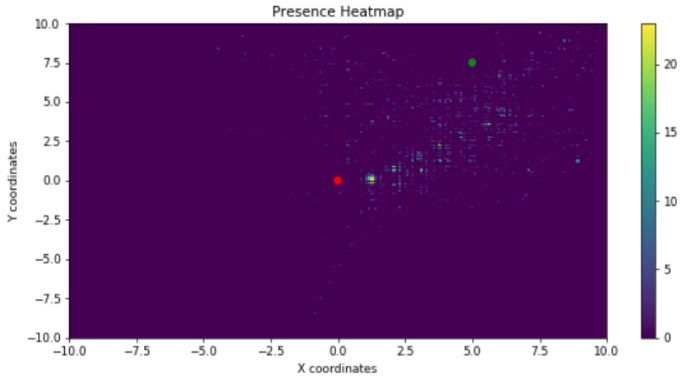


Figura 3.12: Gráfica *Presence Heatmap* inferencia A de Double Q-learning

Parámetro	**Valor**
Media	-48,85
Máximo	177
Mínimo	-192

Cuadro 3.4: Resultados inferencia A Double Q-learning

Inferencia B

Los resultados de la inferencia **B** se muestran en la Figura 3.13. Este objetivo, como explicado anteriormente, presenta una mayor dificultad para ser alcanzado debido como consecuencia del posicionamiento cercano de los obstáculos. En el Cuadro 3.7 se observa como la media de la recompensa baja drásticamente a un valor de -97,4 y se alcanza un valor minimo de recompensa de -243.

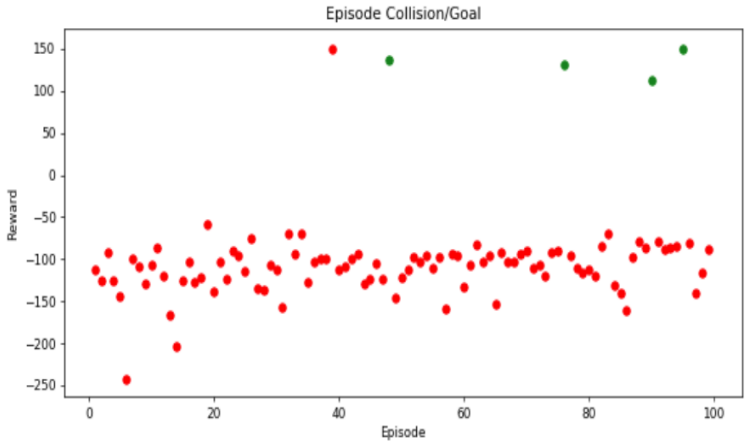


Figura 3.13: Gráfica *Episode Collision/Goal* inferencia B de Double Q-learning

Parámetro	Valor
Media	-97,4
Máximo	149
Mínimo	-243

Cuadro 3.5: Resultados inferencia A Double Q-learning

3.1.3. Deep Q-learning

El aprendizaje para el algoritmo *Deep Q-learning* se inicio con los parámetros definidos en el Cuadro 3.6. Estos parámetros se describieron anteriormente en los apartados **2.3.4. Aprendizaje por refuerzo profundo** y **2.4. Diseño y uso de la aplicación**. Como se puede comprobar el parámetro que define el número de episodios es mayor que para los algoritmos tabulares descritos previamente. Se decidió así, ya que el aprendizaje profundo requiere de una recolección de datos

mucho mayor para poder ajustar eficientemente los pesos de la red neuronal. Como se verá más adelante el aprendizaje profundo no aprende tan rápido como el tabular pero a la larga es mucho más eficiente.

Para realizar los entrenamientos del dron con el algoritmo *Deep Q-learning*, se estableció como objetivo único el mismo que para *Q-learning* y *Double Q-learning* de la Figura 3.1. Los resultados que se obtuvieron en la gráfica de recompensa por episodio se muestran en la Figura 3.14. Estos resultados muestran como el dron aprendido a navegar perfectamente hacia un único objetivo evitando los obstáculos. Sin embargo, se considera que para estos episodios el dron está **sobre-entrenado**, ya que ha memorizado la sucesión de observaciones y acciones mas óptimo pero que le llevan únicamente al objetivo establecido en el entrenamiento. Esto quiere decir, que en el caso de que el dron se viese en un entorno distinto y aplicase lo aprendido en el entrenamiento, no sería capaz de desempeñar la tarea correctamente y la mayoría de las observaciones serían consideradas como nunca entrenadas. Para solucionar este inconveniente es necesario recoger una mayor diversidad de observaciones del entorno para guardarlas en la memoria de observaciones de la red neuronal. Como solución al **sobre-entrenamiento** se propuso una selección aleatoria del objetivo en cada episodio. De esta manera, cada episodio comenzará con una nueva posición objetivo aleatoria de entre las cuatro mostradas en la Figura 3.15.

Hiperparámetro	**Valor**
Alpha	0.1
Gamma	0.99
max_Epsilon	1
min_Epsilon	0.05
Discount	0.999
N_Episodes	6000
N_Steps	500
Target Factor	0.01
Memory Size	100000
Batch Size	32

Cuadro 3.6: Parámetros de entrenamiento Deep Q-learning

Una vez implementada la selección de objetivo aleatorio se ejecuto el entrenamiento del dron con los hiperparámetros especificados en el Cuadro 3.6. La gráfica con la recompensa por episodio se muestra en la Figura 3.21. En esta gráfica se

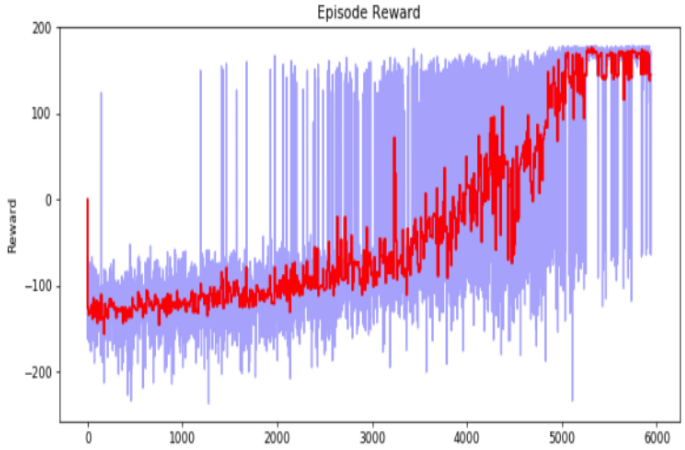


Figura 3.14: Gráfica *Episode/Reward* del entrenamiento Deep Q-learning con objetivo único

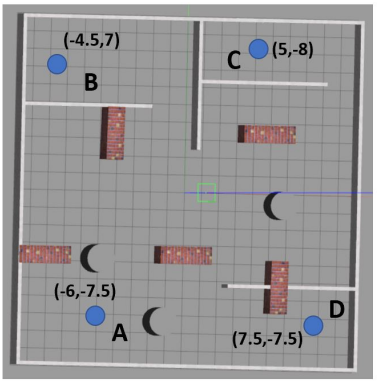


Figura 3.15: Mapa *Maze* con 4 objetivos aleatorios para entrenamiento profundo

aprecia como el entorno va aprendiendo progresivamente hasta alcanzar cierta recompensa estable alrededor de 50. Ahora bien, los resultados no son tan satisfactorios como los de la Figura 3.14 con un único objetivo. En el final del entrenamiento se puede observar cierto ruido generado por la selección aleatoria del objetivo en cada episodio. Esto se debe a que el proceso y tiempo de aprendizaje es distinto para cada objetivo dependiendo de la complejidad de la trayectoria que hay que generar para llegar hasta él.
Por ejemplo, para el punto **D** del mapa de la Figura 3.15 resulto muy difícil que el dron alcanzase el objetivo ya que los obstáculos de la trayectoria estaban muy juntos entre sí, impidiendo el desarrollo de las acciones discretas.

Por otro lado, el numero de episodios realizados fue entorno a 6000, y de entre

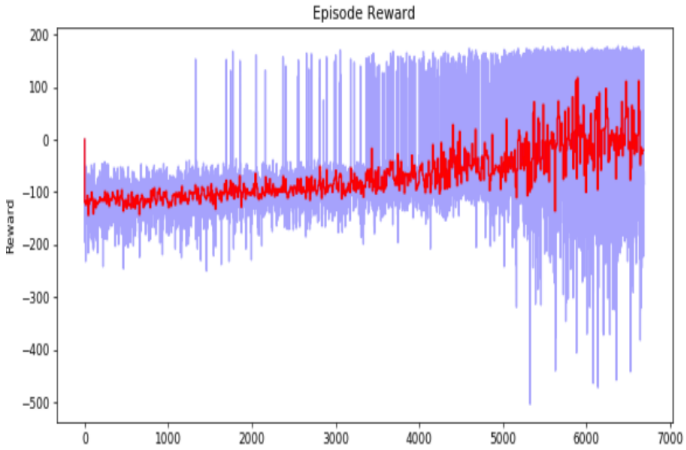


Figura 3.16: Gráfica *Episode/Reward* del entrenamiento Deep Q-learning con objetivos aleatorios

esos 6000 el dron llego un total de 833 a los diferentes objetivos durante el entrenamiento, como se muestra en la Figura 3.17.

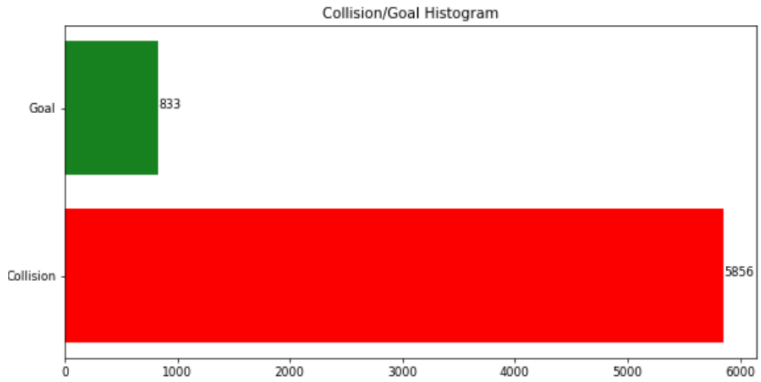


Figura 3.17: Gráfica *Goal/Collision* del entrenamiento Deep Q-learning

A continuación, se expondrán las dos inferencias realizadas con el entrenamiento de *Deep Q-learning* sobre los objetivos **A** y **B** de la Figura 3.5.

Inferencia A

Los resultados de la inferencia **A** con *Deep Q-learning* muestran una tendencia muy satisfactoria como se puede ver en la Figura 3.18. Llegando un total de 89 episodios al objetivo de los 100 intentos, es decir, un 89 % de aciertos en total. Además, comparado con los entrenamientos tabulares, la media de recompensa en la inferencia, mostrada en el Cuadro 3.7, ha aumentado drásticamente a 130,39,

un valor excelente ya que supera la barrera de las recompensas negativas.

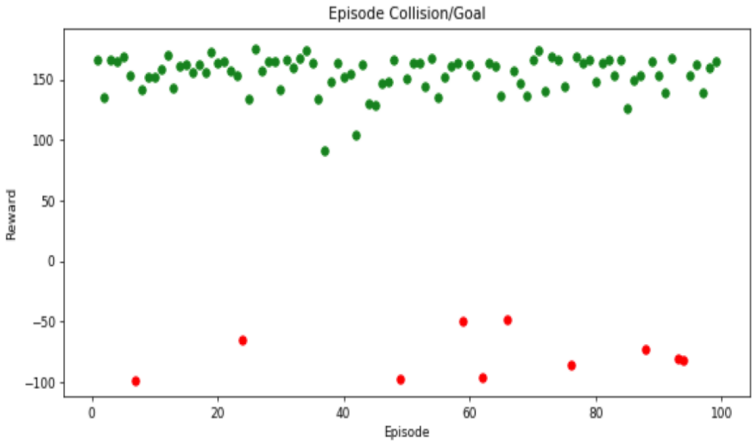


Figura 3.18: Gráfica *Episode Collision/Goal* inferencia A de Deep Q-learning

Parámetro	**Valor**
Media	130,39
Máximo	175
Mínimo	-98

Cuadro 3.7: Resultados inferencia A, Deep Q-learning

Inferencia B

En la inferencia realizada con el punto **B** como objetivo, como en las anteriores pruebas, se aprecia un decaimiento de la recompensa media a 27,42. A pesar de reducirse la recompensa media debido a la dificultad de evadir los nuevos obstáculos del camino, la recompensa media es positiva. Esto va ligado también a una reducción en el número de episodios satisfactorios. El dron llega un total de 54 veces al objetivo de los 100 intentos, un 54 %.

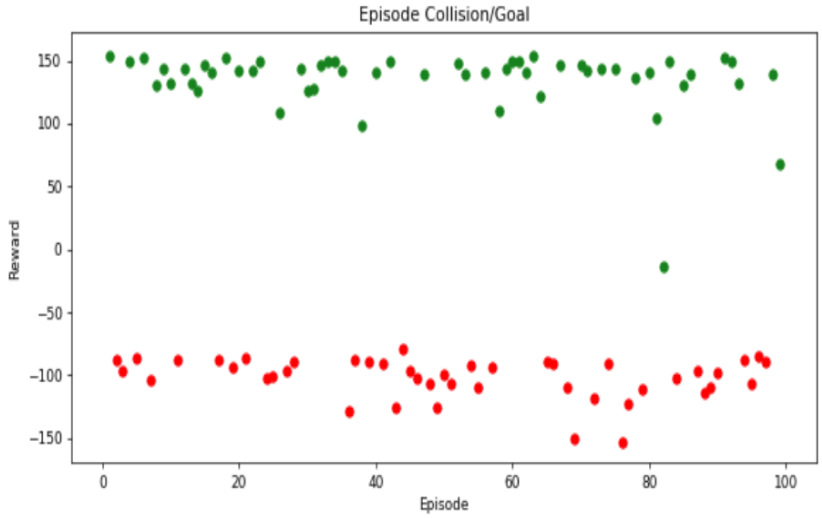


Figura 3.19: Gráfica *Episode Collision/Goal* inferencia B de Deep Q-learning

Parámetro	**Valor**
Media	27,42
Máximo	154
Mínimo	-153

Cuadro 3.8: Resultados inferencia B Deep Q-learning

3.1.4. Double Deep Q-learning

Finalmente, se obtienen los resultados del último algoritmo desarrollado, *Double Deep Q-learning*, implementando los mismos hiperparámetros que en el Cuadro 3.6. Los resultados de la Figura 3.20, muestran el aprendizaje de este algoritmo. Los resultados son similares al entrenamiento por *Deep Q-learning*, sin embargo para el mismo numero de episodio se puede observar que el aprendizaje por *Double Deep Q-learning* es un poco más lento. En total el dron alcanzo 719 veces el objetivo de los 6500 episodios.

En las próximas secciones se muestran los resultados de las inferencias **A** y **B**.

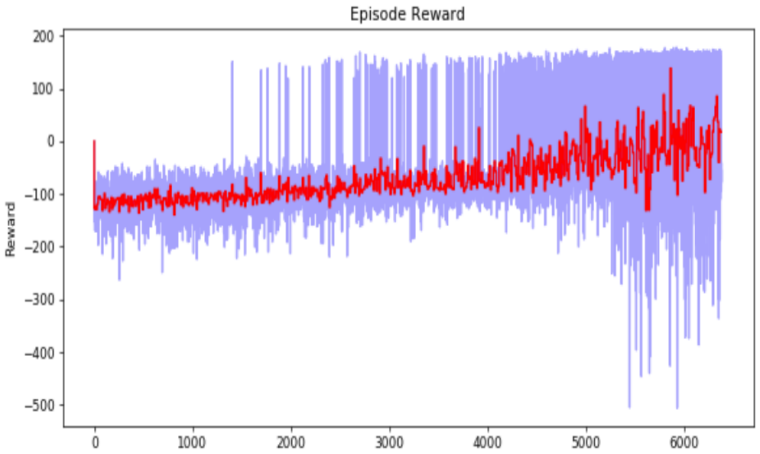


Figura 3.20: Gráfica *Episode/Reward* del entrenamiento Double Deep Q-learning con objetivos aleatorios

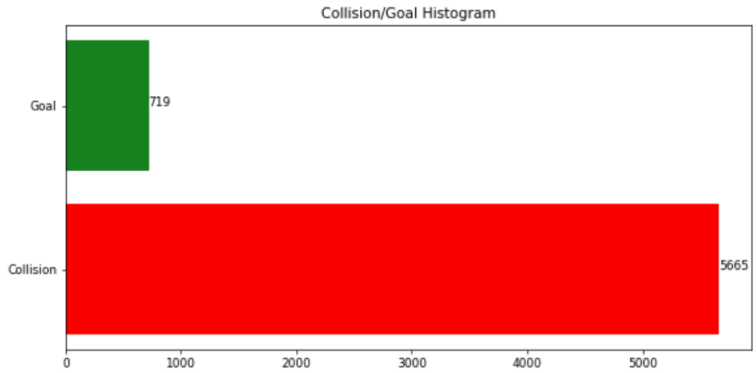


Figura 3.21: Gráfica *Goal/Collision* del entrenamiento Double Deep Q-learning

Inferencia A

La inferencia hacia el objetivo **A** de la Figura 3.5 muestra resultados prometedores. Se puede comprobar que el dron ha aprendido con el algoritmo de *Double Deep Q-learning* a llegar al objetivo evadiendo obstaculos. Los resultados se asemejan a los de *Deep Q-learning* con un total de 74 aciertos de 100 episodios, 74 % de aciertos. La media sigue estando por encima de 0, 91,52. Además el episodio con mínima recompensa también es notablemente alto con un valor de -95.

Figura 3.22: Gráfica *Gráfica* Episode Collision/Goal *inferencia A de Double Deep Q-learning*

Parámetro	**Valor**
Media	91,52
Máximo	177
Mínimo	-95

Cuadro 3.9: Resultados inferencia A Double Deep Q-learning

Inferencia B

La inferencia **B**, como es de esperar, proporciona resultados peores que **A**, pero se puede comprobar que ha aprendido adecuadamente a llegar al destino objetivo. En esta inferencia se consigue una media positiva de 7,63 y llega 41 veces al objetivo de los 100 episodios. Los resultados recogidos son peores que para el entrenamiento con *Deep Q-learning*, probablemente debido al aprendizaje más lento ya que se tienen que entrenar dos conjuntos de pesos de la red neuronal en vez de uno.

Parámetro	**Valor**
Media	7,63
Máximo	154.0
Mínimo	-152.0

Cuadro 3.10: Resultados inferencia B Double Deep Q-learning

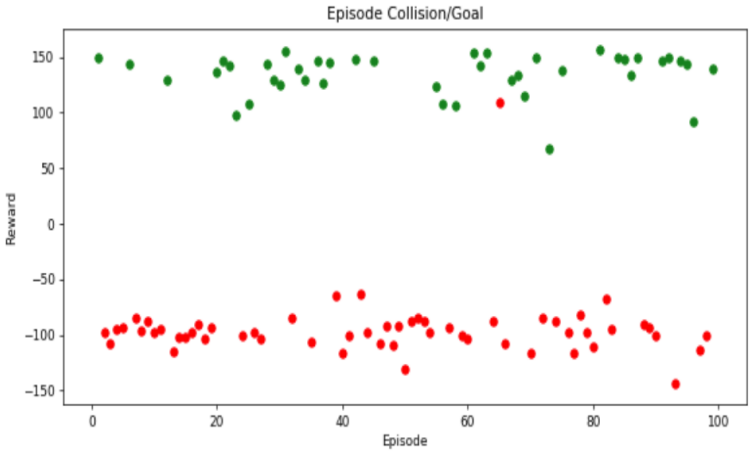


Figura 3.23: Gráfica *Episode Collision/Goal* inferencia B de Double Deep Q-learning

3.2. Comparación de algoritmos

La comparación de los algoritmos se realizara en la ventana "*Compare*"de la interfaz gráfica de usuario. La comparación de los entrenamientos se realizará únicamente para distinguir las diferencias entre los algoritmos de aprendizaje por refuerzo tabular y profundo, ya que los algoritmos de un mismo grupo se asemejan mucho entre ellos y las gráficas de entrenamiento se superponen.

En la Figura 3.24 se comparan las gráficas de recompensa por episodio de los algoritmos *Double Q-learning* en naranja y *Double Deep Q-learning* en azul. Como se puede comprobar el entrenamiento de *Double Q-learning* se realiza en menos episodios. Esto es debido a que el algoritmo aprende de forma más rápida al tener un espacio de estados mucho menor. Por lo tanto, en la gráfica se observa como en los primeros episodios, el aprendizaje por *Double Q-learning*, se acerca a un valor de aproximadamente -80 y mas o menos estable hasta el fin del entrenamiento. La pendiente de la recompensa , del aprendizaje por refuerzo *Double Deep Q-learning* es menor al comienzo y se observa como el proceso de aprendizaje es más lento. A pesar de ello, tras entrenar durante un periodo de episodios mas lago, este tipo de aprendizaje último es capaz de estabilizarse en un valor de recompensa mucho mas elevada, de aproximadamente 50.

La comparación de todos los tipos de entrenamiento no se muestra ya que las gráficas se superponen unas sobre otras. Los entrenamientos tabulares, como los de aprendizaje profundo siguen una misma tendencia entre ellos sin mostrar grandes

diferencias.

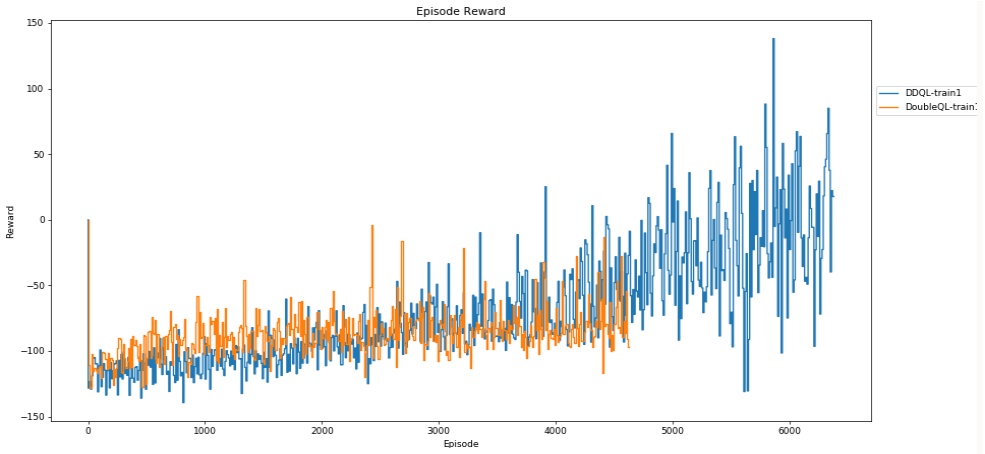


Figura 3.24: Comparación del entrenamiento de algoritmos Double Q-learning y Double Deep Q-learning

En cuanto a las inferencias, se comparan los algoritmos tabulares y profundos entre sí para ambas inferencias **A** y **B** de la Figura 3.5.

Para la inferencia **A**, en la Figura 3.25 se observa que el porcentaje de episodio satisfactorio para ambos algoritmos tabulares, es muy similar. A pesar de ello, las cifras son notablemente bajas. Por lo tanto, el entrenamiento por medio de estos algoritmos, para realizar la tarea de evitar obstáculos, no se considera satisfactoria y no produce resultados eficientes como para poder ser implementados en un dron real.

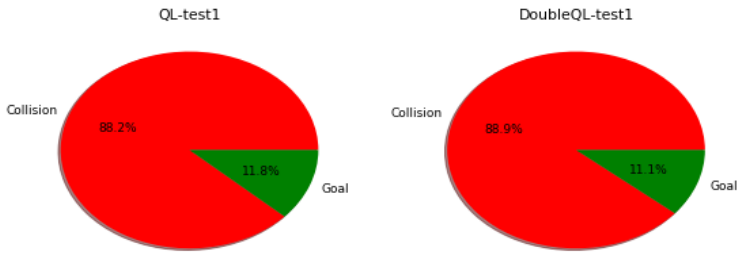


Figura 3.25: Porcentaje colisión/objetivo inferencia A, Q-learning y Double Q-learning

Por otro lado, en las inferencias hacia el punto **A** realizadas con los algoritmos

de aprendizaje profundo se aprecia un aprendizaje más eficiente con resultados que dan confianza para ser implementados en la realidad. En la Figura 3.26 se observa que ambos algoritmos proporcionan mas del 50 % de episodios satisfactorios. Sin embargo, *Deep Q-learning* presenta resultados mejores que *Double Deep Q-learning*, 89,9 % a 70,7 % respectivamente. Esto se puede deber a que ambos algoritmos han sido entrenados con el mismo numero de episodios, y como se expuso anteriormente, el algoritmo *Double Deep Q-learning* requiere de un mayor tiempo de computación al tener que entrenar el doble de pesos de la red neuronal.

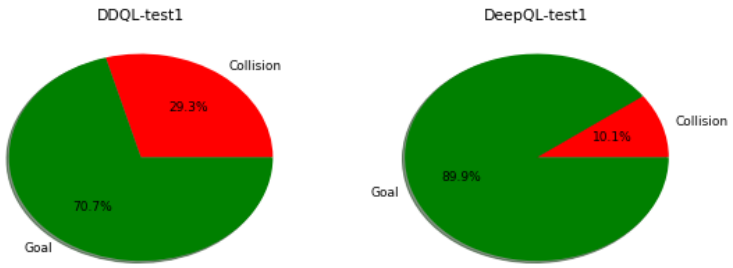


Figura 3.26: Porcentaje colisión/objetivo inferencia A, Deep Q-learning y Double Deep Q-learning

Para concluir con los resultados de la inferencia en **A**, en la Figura 3.27 se exponen los caminos mas óptimos que se han desarrollado con cada algoritmo de aprendizaje por refuerzo. De esta gráfica se puede distinguir que el camino mas largo y menos eficiente es el desarrollado por *Q-learning*, como era de esperar.

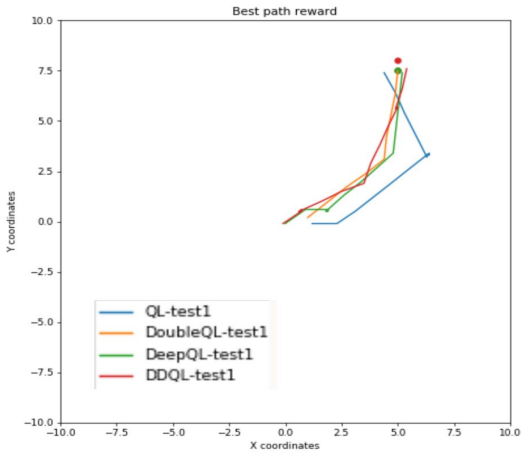


Figura 3.27: Mapa de las trayectorias del dron para inferencia A

Finalmente, se analizan los resultados de las inferencias de todos los algoritmos

con objetivo el punto **B** de la Figura 3.5. Este objetivo presenta una mayor complicación para definir una trayectoria debido a la proximidad de los obstáculos. En la Figura 3.28 se observa que los algoritmos tabulares no son capaces de llegar al objetivo deseado frecuentemente con unos porcentajes de éxito por debajo de 5 %.

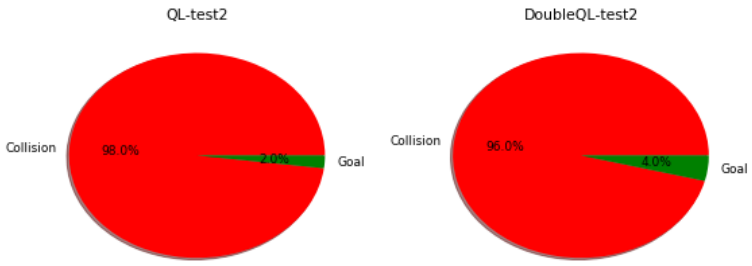


Figura 3.28: Porcentaje colisión/objetivo inferencia B, Q-learning y Double Q-learning

Por otro lado, en los algoritmos de aprendizaje profundo se aprecia un descenso de los episodios satisfactorios con respecto a las inferencias en el objetivo **A**. A pesar de ello, los porcentajes son estables y sugieren que el dron llega con cierta certeza al objetivo y se pueden considerar entrenamientos efectivos. En estos algoritmos el *Deep Q-learning* sigue mostrando mejores resultados que el *Double Deep Q-learning*.

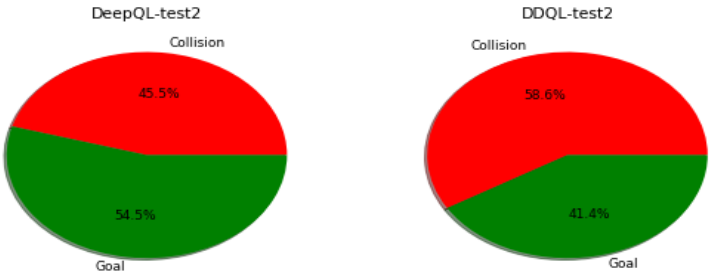


Figura 3.29: Porcentaje colisión/objetivo inferencia B, Deep Q-learning y Deep Q-learning

Finalmente, en la Figura 3.30 se dibujan los caminos mas óptimos que ha seguido el dron hasta el objetivo. En esta gráfica se puede observar una clara diferencia entre los algoritmos de aprendizaje tabular y los de aprendizaje profundo. Los algoritmos de aprendizaje profundo definen trayectorias mas simples y con el menor numero de pasos posibles, yendo el dron directamente al objetivo por un camino

óptimo. Mientras que los algoritmos de aprendizaje tabular iteran más veces hasta llegar al objetivo creando trayectorias excesivamente largas.

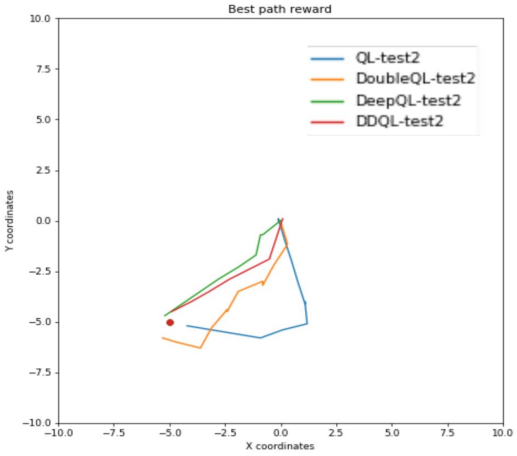


Figura 3.30: Mapa de las trayectorias del dron para inferencia B

Capítulo 4

Conclusiones y trabajos futuros

En este capítulo se expondrán las conclusiones del trabajo desarrollado. Se comentarán aquellos objetivos que han sido alcanzados y los que no, así como una breve introducción a posibles mejoras que se podrían hacer en trabajos futuros sobre este proyecto.

4.1. Conclusión

En vista a los objetivos establecidos inicialmente para el desarrollo de este proyecto y los resultados obtenidos, se puede afirmar con confianza que se ha conseguido satisfactoriamente la primera aproximación al desarrollo de una plataforma que permita el entrenamiento de un dron mediante algoritmos de aprendizaje por refuerzo. Para acceder al código fuente que se ha desarrollado para este proyecto se adjunta el enlace del repositorio de GitHub[1]. A pesar de que se hayan encontrado algunos impedimentos, este proyecto sirve de introducción para la implementación de otros desarrollos futuros que mejoren la aplicación.

En primer lugar, se ha desarrollado una simulación semi-realista empleando las herramientas de programación ROS y Gazebo. Para la simulación se emplea la librería de un dron llamado *Hector*, el cual no existe en el mundo real pero que simula el control y física de vuelo de una forma muy realista. Además, los sensores que incluye esta librería para el dron han sido de gran ayuda para poder desarrollar los algoritmos de aprendizaje por refuerzo. El inconveniente que apareció para el desarrollo de este sistema fue la compatibilidad de ROS Melodic con el sistema operativo Windows 10,para el cual no se pudo desarrollar la aplicación.

[1]Repositorio HectorRL:`https://github.com/Rodridelazcano/HectorRL`

En segundo lugar, se ha integrado OpenAI Gym con la simulación desarrollada en ROS y Gazebo, llegándose a implementar los algoritmos de aprendizaje por refuerzo *Q-learning*, *Double Q-learning*, *Deep Q-learning* y *Double Deep Q-learning*. La implementación de estos algoritmos resulto ser satisfactoria para los algoritmos de aprendizaje por refuerzo profundo, con trayectorias óptimas definidas por el dron tras aplicarle los pesos de la red neuronal aprendida. A diferencia de los algoritmos tabulares cuyos resultados no son del todo eficientes, debido principalmente a la discretizacion del espacio de observaciones y acciones. Uno de los inconvenientes encontrados al implementar los algoritmos de aprendizaje por refuerzo fue el tiempo de computación. Debido a la bajas prestaciones computacionales del ordenador empleado para desarrollar el proyecto, los entrenamientos de cada algoritmo duraban una media de entre 4 a 5 días.

Finalmente, se diseño una interfaz gráfica de usuario que permite ejecutar los algoritmos de aprendizaje por refuerzo en la simulación así como ver los resultados de los entrenamientos e inferencias de estos en tiempo real. Además, se añadió una opción para poder comparar los resultados de los aprendizajes entre sí. La interfaz ha sido de gran utilidad para poder visualizar el correcto funcionamiento de los entrenamientos y analizar los datos posteriormente. Una de las mayores ventajas de esta interfaz es la posibilidad de ser implementada por cualquier usuario de forma intuitiva y sin conocimientos previos de aprendizaje por refuerzo. Sin embargo, la aplicación solo fue desarrollada en Ubuntu, limitando a los usuarios de Windows 10 de su uso.

4.2. Trabajo futuro

En este proyecto se ha realizado una primera aproximación al desarrollo de una plataforma que permita entrenar a un dron por algoritmos de aprendizaje por refuerzo. De esta manera, a continuación se propondrán algunas lineas de trabajo futuras que podrían resultar ventajosas para una continuación del proyecto.

- Uno de los objetivos no cumplimentados en este proyecto es integrar la plataforma desarrollada en Windows 10. Como linea futura se podría desarrollar la compatibilidad del proyecto con Windows 10 para que puedan beneficiar a un mayor numero de usuarios.

- Como se expuso anteriormente, uno de los impedimentos al de implementar entrenamientos mas largos y a mayor velocidad fue la potencia computacional del ordenador que se disponía. Por lo tanto, una linea de trabajo futura consistiría en implementar la aplicación en un sistema mas potente o un

servidor en la "*nube*"que aporte a los entrenamientos la potencia necesaria para aumentar la velocidad de las simulaciones.

- Uno de los mayores impedimentos del entrenamiento del dron fue el espacio de acciones discreto el cual incitaba a colisiones. Para solucionar este problema se podría investigar sobre implementar un espacio de acciones continuo con velocidades variables. Por otro lado, con respecto a la movilidad del dron se podría añadir libertad en la altura del dron añadiendo más posibles acciones al espacio.
- Por último, sería interesante implementar los algoritmos en un dron en el mundo real y ver si los resultados difieren de las simulaciones.

Appendices

Apéndice A

ODS

En 2015 la Asamblea General de las Naciones Unidas acordó que para 2030 se perseguirían un conjunto de 17 objetivos denominados Objetivos de Desarrollo Sostenible. Estos objetivos tienen como finalidad desarrollar un plan para lograr un futuro más sostenible y mejor para todos por igual.

El contenido de este proyecto se basa en dotar a un dron de una funcionalidad esencial, la autonomía, la cual se puede aplicar en la mayoría de aplicaciones que actualmente existen y las que se prevén que aparezcan en un futuro para los drones. Para este proyecto en concreto, al poder ser aplicable en casi todos los sectores de la industria en los que participan los drones, se hará referencia a algunos de estos sectores más relevantes y como están influenciando al cumplimiento de los ODS. A continuación, se definen y analizan aquellos ODS [32] que más beneficiados se hayan visto por el desarrollo de este proyecto:

1. Salud y Bienestar (ODS3): Garantizar una vida sana y promover el bienestar para todos en todas las edades.

   Objetivo 3.8: Lograr la cobertura sanitaria universal, en particular la protección contra los riesgos financieros, el acceso a servicios de salud esenciales de calidad y el acceso a medicamentos y vacunas seguros, eficaces, asequibles y de calidad para todos.[32]

   Gracias al desarrollo de nuevas técnicas de navegación de drones, como las que se han implementado en este proyecto, se está facilitando el acceso a los servicios de salud para aquellos grupos de población que viven en áreas remotas y con pocos recursos sanitarios. Numerosos proyectos relacionados con drones autónomos están en marcha para realizar entregas de material médico y vacunas, así como el retorno de muestras de sangre. Esto es de

gran ayuda en países en vías de desarrollo donde los viajes con drones se pueden hacer de forma rápida y segura en áreas poco accesibles por otros medios de transporte.

Según la Organización Mundial de la Salud, son mas de 400 millones de personas en todo el mundo que carecen de servicios sanitarios básicos [33]. Actualmente existe una iniciativa en África llamada *Zipline* que ya ha realizado mas de 1 millón de kilómetros de vuelo en Rwanda que aproxima a 13000 entregas. Estos drones han llegado a enviar el 35 % de la sangre para transfusiones. Por otro lado, en Ghana están comenzando a enviarse pruebas de COVID-19 con estos aparatos [34].

2. Trabajo Decente y Crecimiento Económico (ODS8): Promover el crecimiento económico inclusivo y sostenible, el empleo y el trabajo decente para todos.

   Objetivo 8.2: Lograr niveles más elevados de productividad económica mediante la diversificación, la modernización tecnológica y la innovación, entre otras cosas centrándose en los sectores con gran valor añadido y un uso intensivo de la mano de obra. [32]

   Como se expuso en la sección **1.1. Mercado de drones autónomos**, la tecnología de drones autónomos tiene como expectativas expandirse por una gran variedad de sectores de la industria comercial. Además, traerá con ello un gran valor añadido esperándose un crecimiento económico en las industrias que incorporen esta tecnología, así como nuevos puestos de trabajo que incluyan tanto pilotos como técnicos especializados en mantenimiento y diseño. El proyecto que se ha desarrollado en esta memoria tiene como objetivo aportar en el desarrollo de drones autónomos y su posible introducción en sectores como la agricultura y transporte. Además, según un informe de la Asociación Internacional de Sistemas de Vehículos No Tripulados (AUVSI), se estima la apertura de 100,000 nuevos puestos de trabajo en los sectores que incorporaran drones para 2025 [35].

3. Industria, Innovación e Infraestructura (ODS9): Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación

   Objetivo 9.5: Aumentar la investigación científica y mejorar la capacidad tecnológica de los sectores industriales de todos los países, en particular los países en desarrollo, entre otras cosas fomentando la innovación y aumentando considerablemente, de aquí a 2030, el número de personas que trabajan

en investigación y desarrollo por millón de habitantes y los gastos de los sectores público y privado en investigación y desarrollo. [32]

Este proyecto implementa una técnica de aprendizaje artificial recientemente introducida en el sector de la industria. El aprendizaje por refuerzo todavía se considera en fase de investigación y son numerosas las instituciones que están invirtiendo en desarrollar nuevas aplicaciones debido a su gran potencial y resolución de problemas que van de acuerdo con los ODS. Este proyecto será publicado sin licencias para que el sector de la investigación, tanto de drones como de aprendizaje reforzado, se pueda beneficiar de el y desarrollar nuevas aplicaciones basadas en su contenido.

Bibliografía

[1] *Commercial Drone Market Size, Share Trends Analysis Report By Application (Filming Photography, Inspection Maintenance), By Product (Fixed-wing, Rotary Blade Hybrid), By End Use, And Segment Forecasts, 2019 - 2025*. Grand View Research, 2019.

[2] Goldman Sachs. *Drones: Reporting for Work*. 2019. URL: https://www.goldmansachs.com/insights/technology-driving-innovation/drones/ (visitado 08-08-2020).

[3] Haye Kesteloo. *Drone industry is growing fast to $100 billion by 2020 and $1.5 trillion by 2040*. 2019. URL: https://dronedj.com/2019/11/25/drone-industry-100-billion-2020/ (visitado 08-08-2020).

[4] *The State of the Drone Market 2020: Scale and Growth*. Inf. téc. 2020.

[5] EASA. *Drones - regulatory framework timeline*. 2019. URL: https://www.easa.europa.eu/drones-regulatory-framework-timeline (visitado 08-08-2020).

[6] Juan Antonio Pascual. *EHang 184, el dron taxi volador autónomo y eléctrico que ya transporta pasajeros*. 2019. URL: https://computerhoy.com/noticias/motor/ehang-184-dron-taxi-autonomo-electrico-ya-transporta-pasajeros-383533 (visitado 08-08-2020).

[7] Redacción Computing. *DHL Express lanza su primer servicio de entrega con drones urbanos*. 2019. URL: https://www.computing.es/movilidad/noticias/1111991046501/dhl-express-lanza-primer-servicio-de-entrega-drones-urbanos.1.html (visitado 08-08-2020).

[8] Juan Antonio Pascual. *Este es el primer dron de seguridad autónomo para proteger la casa*. 2020. URL: https://computerhoy.com/noticias/tecnologia/primer-dron-seguridad-autonomo-proteger-casa-600511 (visitado 08-08-2020).

[9] Ben Lorica. *Practical applications of reinforcement learning in industry*. 2017. URL: https://www.oreilly.com/radar/practical-applications-of-reinforcement-learning-in-industry/ (visitado 08-08-2020).

[10] OpenAI. *Learning Dexterity*. 2018. URL: `https://openai.com/blog/learning-dexterity/` (visitado 08-08-2020).

[11] Andres G Barto Ronald J Williams Richard S Sutton. "Reinforcement learning is direct adaptive optimal control". En: *IEEE Control Systems* 12.2 (1992), págs. 19-22.

[12] et al Marc G Bellemare Alex Graves. "Human-level control through deep reinforcement learning". En: *Nature* 518.7540 (2015), págs. 529-53.

[13] T. Darrell K. Saenko. E.T. J. Hoffman. "Simultaneous deep transfer across domains and tasks". En: *IEEE Interna-tional Conference on Computer Vision* (2015), págs. 4068-4076.

[14] Zubair Ahmed Khan. "Obstacle Avoidance Methods in UAVs". Tesis doct. 2019.

[15] Yungcheol Byun Mudassar Liaq. "Autonomous UAV Navigation Using Reinforcement Learning". En: *International Journal of Machine Learning and Computing* 9.6 (2019), págs. 756-761.

[16] Sang-Yun Shin, Yong-Won Kang y Yong-Guk Kim. "Obstacle Avoidance Drone by Deep Reinforcement Learning and Its Racing with Human Pilot". En: *Applied Sciences* 9.24 (2019), pág. 5571.

[17] Guillem Muñoz y col. "Deep Reinforcement Learning for Drone Delivery". En: *Drones* 3.3 (2019), pág. 72.

[18] Dhruv Mauria Saxena y col. "Autonomous Quadrotor Flight in Simulation using Reinforcement Learning". En: ().

[19] Shuhuan Wen y col. "The Q-learning obstacle avoidance algorithm based on EKF-SLAM for NAO autonomous walking under unknown environments". En: *Robotics and Autonomous Systems* 72 (2015), págs. 29-36.

[20] Tiago Ribeiro y col. "Q-Learning for Autonomous Mobile Robot Obstacle Avoidance". En: *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE. 2019, págs. 1-7.

[21] Kjell Kersandt, Guillem Muñoz y Cristina Barrado. "Self-training by Reinforcement Learning for Full-autonomous Drones of the Future". En: *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*. IEEE. 2018, págs. 1-10.

[22] Morgan Quigley y col. "ROS: an open-source Robot Operating System". En: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, pág. 5.

[23] Iker Zamora y col. "Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo". En: *arXiv preprint arXiv:1608.05742* (2016).

[24] Artur Sagitov y col. "Transfer of learned exploration strategies of a mobile robot from a simulated to real environments". En: (2019).

[25] Miguel Angel Rodriguez Alberto Ezquerro y Ricardo Tellez (of The Construct). *openai_ros*. URL: `http://wiki.ros.org/openai_ros`. (accessed: 29.05.2020).

[26] S.Kohlbrecher J.Meyer A.Sendobry. "Comprehensive Simulation of Quadrotor UAVsUsing ROS and Gazebo". En: *SIMPAR* (2012).

[27] Matthias Plappert. *keras-rl*. `https://github.com/keras-rl/keras-rl`. 2016.

[28] F. Chollet et al. *Keras*. `https://github.com/fchollet/keras`. 2015.

[29] Richard S Sutton Andrew G Barto. *Reinforcement learning: An introduction, volume 1*. MIT press Cambridge, 1998.

[30] Hado Van Hasselt, Arthur Guez y David Silver. "Deep reinforcement learning with double q-learning". En: *arXiv preprint arXiv:1509.06461* (2015).

[31] Cristina Barrado Enric Pastor Ender Çetin. "Counter a Drone in a Complex Neighborhood Area by Deep Reinforcement Learning". En: *Sensors 20(8)* (2020).

[32] Naciones Unidas. *Objetivos de desarrollo sostenible*. 2015. URL: `https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/` (visitado 08-08-2020).

[33] World Health Organization. *New report shows that 400 million do not have access to essential health services*. 2015. URL: `https://www.who.int/mediacentre/news/releases/2015/uhc-report/en/` (visitado 08-08-2020).

[34] Vignesh Santhanam. *How drones could change the future of healthcare delivery*. 2020. URL: `https://www.weforum.org/agenda/2020/05/medical-drone-delivery-india-africa-modernize-last-mile/` (visitado 08-08-2020).

[35] Darryl Jenkins y Bijan Vasigh. *The economic impact of unmanned aircraft systems integration in the United States*. Association for Unmanned Vehicle Systems International (AUVSI), 2013.

Parte II

Presupuestos

Capítulo 1

Mediciones

En este capítulo se listarán en tablas las unidades de cada elemento tanto software como mano de obra que se han aplicado en el desarrollo del proyecto

1.1. Componentes Principales

Componentes	Cantidad
Ordenador de mesa	1
Ordenador portátil	1

Cuadro 1.1: Unidades componentes principales

1.2. Software

Componentes	Cantidad	Horas
Ubuntu 18.04	1	400
Python 2.7	1	400
ROS Melodic	1	200
Gazebo	1	100
Qt5	1	200
LaTeX	1	50

Cuadro 1.2: Mediciones software

1.3. Mano de Obra

Actividad	Horas
Desarrollo de la simulación	100
Integración de la aplicación	200
Desarrollo de algoritmos	250
Desarrollo de la interfaz gráfica	150
Obtención y análisis de resultados	300
Redacción de documentos	50

Cuadro 1.3: Mediciones mano de obra

Capítulo 2

Precios unitarios

En este capítulo se definirá el coste por unidad de cada elemento empleado para desarrollar el proyecto.

2.1. Componentes Principales

Componentes	Precio(€/Ud.)
Ordenador de mesa	1,000
Ordenador portátil	1,200

Cuadro 2.1: Precios unitarios componentes principales

2.2. Software

Componentes	Precio(€/Ud.)
Ubuntu 18.04	0
Python 2.7	0
ROS Melodic	0
Gazebo	0
Qt5	0
LaTeX	0

Cuadro 2.2: Precios unitarios software

2.3. Mano de Obra

Actividad	Precio(€/Ud.)
Desarrollo de la simulación	10
Integración de la aplicación	15
Desarrollo de algoritmos	20
Desarrollo de la interfaz gráfica	10
Obtención y análisis de resultados	15
Redacción de documentos	10

Cuadro 2.3: Precios unitarios mano de obra

Capítulo 3

Sumas parciales

En este capitulo se obtiene el coste total de cada uno de los recursos empleados en el proyecto a partir de las unidades de las mediciones y los precios unitarios.

3.1. Componentes Principales

Componentes	Cantidad	Precio(€/Ud.)	Coste(€)
Ordenador de mesa	1	1,000	1,000
Ordenador portátil	1	1,200	1,200
		Total	2,200

Cuadro 3.1: Sumas parciales componentes principales

3.2. Software

Componentes	Cantidad	Precio(€/Ud.)	Coste(€)
Ubuntu 18.04	1	0	0
Python 2.7	1	0	0
ROS Melodic	1	0	0
Gazebo	1	0	0
Qt5	1	0	0
LaTeX	1	0	0
		Total	0

Cuadro 3.2: Sumas parciales software

3.3. Mano de Obra

Actividad	Horas	Precio(€/Ud.)	Coste(€)
Desarrollo de la simulación	100	10	1,000
Integración de la aplicación	200	15	3,000
Desarrollo de algoritmos	250	20	5000
Desarrollo de la interfaz gráfica	150	10	1,500
Obtención y análisis de resultados	300	15	4,500
Redacción de documentos	50	10	500
		Total	15,500

Cuadro 3.3: Sumas parciales mano de obra

Capítulo 4

Presupuesto general

En este capítulo se calcula las suma de todos los costes del proyecto.

Concepto	Coste(€)
Componentes principales	2,200
Software	0
Mano de obra	15,500
Total	17,700

Cuadro 4.1: Presupuesto general total