UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID
Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Universitat Politècnica de València
Departamento de Sistemas Informáticos y Computación

# Desarrollo de un simulador para la coordinación de drones usando la plataforma SPADE

**TRABAJO FIN DE MÁSTER**

Máster en inteligencia artificial, reconocimiento de formas e imagen digital

*Autor:* Masanet López, Joan

*Tutor:* Julian Inglada, Vicente Javier

Curso 2019-2020

# Resumen

En la actualidad el uso de drones para la ejecución de distintas tareas se encuentra en aumento. Este hecho crea la necesidad de utilizar mecanismos que permitan gestionar el control de estas aeronaves de forma automática.

Es por ello que en este proyecto se ha creado una herramienta de simulación que permite el control de drones de forma automática. En esta herramienta se han unido las posibilidades que ofrece un sistema de agentes inteligente junto con un entorno de simulación.

La herramienta desarrollada se ha implementado usando una arquitectura que permite utilizar una unión del sistema de agentes SPADE y el simulador de vehículos AirSim. Gracias a esta estructura es posible llevar a cabo el desarrollo de aplicaciones con modelos de coordinación con agentes inteligentes y modelos de aprendizaje por refuerzo adaptado a drones.

El modelo de coordinación entre agentes, en este caso adaptado a drones, permite realizar tareas conjuntamente gracias a la mensajería instantánea entre los mismos. Durante el transcurso del proyecto se ha desarrollado un ejemplo de coordinación entre drones consiguiendo realizar una serie de maniobras por el entorno de simulación a modo de seguimiento.

En cuanto al modelo de aprendizaje por refuerzo desarrollado, consta de realizar acciones que implican movimientos y obtener una recompensa. Esta recompensa evalúa el grado de beneficio aportado para conseguir el objetivo final. A medida que se realizan las acciones, el algoritmo aprende qué acciones son más beneficiosas bajo determinados estados.

**Palabras clave:** Drone, Agentes inteligentes, SPADE, AirSim, Coordinación, Algoritmo de aprendizaje por refuerzo

---

# Resum

En l'actualitat l'ús de drons per a l'execució de diferents tasques es troba en augment. Aquest fet crea la necessitat d'utilitzar mecanismes que permeten gestionar el control d'aquestes aeronaus de manera automàtica.

És per això que en aquest projecte s'ha creat una eina de simulació que permet el control de drons de manera automàtica. En aquesta eina s'han unit les possibilitats que ofereix un sistema d'agents intel·ligent juntament amb un entorn de simulació.

L'eina desenvolupada s'ha implementat usant una arquitectura que permet utilitzar una unió del sistema d'agents SPADE i el simulador de vehicles AirSim. Gràcies a aquesta estructura és possible dur a terme el desenvolupament d'aplicacions amb models de coordinació amb agents intel·ligents i models d'aprenentatge per reforç adaptat a drons.

El model de coordinació entre agents, en aquest cas adaptat a drons, permet fer tasques conjuntament gràcies a la missatgeria instantània entre aquests. Durant el transcurs del projecte s'ha desenvolupat un exemple de coordinació entre

drons aconseguint realitzar una sèrie de maniobres per l'entorn de simulació a manera de seguiment.

En quant al model d'aprenentatge per reforç desenvolupat, consta de realitzar accions que impliquen moviments i obtindre una recompensa. Aquesta recompensa avalua el grau de benefici aportat per aconseguir l'objectiu final. A mesura que es realitzen les accions, l'algorisme aprén quines accions són més beneficioses baix determinats estats.

**Paraules clau:** Dron, Agents intel·ligents, SPADE, AirSim, Coordinació, Algorisme d'aprenentatge per reforç

# Abstract

Currently the use of drones for the execution of different tasks is increasing. This fact generates a need for using mechanisms to manage the control of these aircraft automatically.

For that reason this project has developed a simulation tool that allows the control of drones automatically. Furthermore, this implementation merges the capabilities offered by an intelligent agent system along with a simulation environment.

The developed tool allows the user to use a union of the SPADE agent system and the AirSim vehicle simulator. An architecture has been implemented to support these two technologies, for the development of applications with models of coordination with intelligent agents and models of learning by reinforcement adapted to drones.

The coordination model between agents, in this case adapted to drones, allows the user to perform joined tasks through instant messaging between agents. In the current project an example of coordination between drones has been developed, managing to carry out a series of manoeuvres through the simulation environment using a "following" mode.

Regarding the reinforcement learning model developed, it consists of performing actions that involve movements and obtaining a reward from them. This reward assesses the degree of benefit brought into the system to achieve the ultimate goal. As actions are performed, the algorithm learns which actions are most beneficial under determinate states.

**Key words:** Drone, Smart Agents, SPADE, AirSim, Coordination, Reinforcement Learning Algorithm

# Índice general

# Índice de figuras



# Índice de tablas

# Listings

# CAPÍTULO 1
# Introducción

## 1.1 Motivación

El proyecto actual está impulsado por la motivación de crear una herramienta de simulación que disponga de la integración de un sistema de agentes inteligente junto con la simulación de entornos de vuelo para drones. La unión de estas dos tecnologías ofrece un amplio campo de investigación y desarrollo de aplicaciones de distintos ámbitos. En este proyecto se decidió abordar modelos de coordinación entre agentes y modelos de aprendizaje por refuerzo.

La coordinación entre agentes es una parte muy importante a la hora de ejecutar tareas entre distintos agentes. De este modo, facilita una organización correcta entre elementos distribuidos como son los agentes. Por otra parte, los modelos de aprendizaje por refuerzo recopilan información beneficiosa para la ejecución de tareas. En este tipo de aprendizaje no es necesario la supervisión del aprendizaje, basta con definir el refuerzo de modo apropiado.

De este modo, surgió la necesidad y motivación de desarrollar una herramienta de simulación que fuera capaz de albergar modelos de coordinación y modelos de aprendizaje automático entre otros.

## 1.2 Objetivos

Los objetivos de este proyecto se basan en una serie de puntos adquiridos a partir de la inquietud de desarrollar una herramienta que permita la coordinación entre agentes inteligentes y modelos de aprendizaje por refuerzo aplicados a la conducción de drones de forma autónoma.. A continuación se detallan los objetivos del proyecto.

- Estudiar las posibles opciones para interconectar un sistema multiagente con un simulador de drones.
    - Seleccionar un sistema multiagente para el desarrollo del proyecto.
    - Seleccionar un simulador de drones para el desarrollo del proyecto.
    - Implementar una conexión entre el sistema multiagente y el simulador de drones.

- Diseñar e implementar una plataforma de simulación de drones para poder probar diferentes estrategias y desarrollar nuevos casos.
- Validar la plataforma mediante el desarrollo de casos base donde los agentes interactúan con los drones, a modo de herramienta para ejecutar las acciones sobre un entorno de simulación.
    - Desarrollar un caso de coordinación de agentes donde existe un agente/drone que toma el rol *Maestro* e indica al resto de agentes/drones que toman el rol *Esclavo*, indicándose donde tiene que desplazarse a modo de seguimiento "FollowMe".
    - Implementar un modelo de aprendizaje con un algoritmo de *Reinforcement Learning* con DQN para desarrollar un sistema de conducción autónoma en drones basada en aprendizaje por refuerzo.

## 1.3 Estructura del documento

El presente documento se encuentra dividido en cuatro partes. El primer bloque describe el estado del arte, incluyendo los inicios de los drones y como el nombre del bloque indica, el estado del arte actual. En la actualidad se detallan algunos ejemplos interesantes sobre drones autónomos. Pasando a los simuladores, se detalla de forma más extensa el simulador seleccionado para el proyecto y se hace una pequeña revisión de unos simuladores de drones para el desarrollo de aplicaciones que han parecido más interesantes para este proyecto.

Seguidamente se realiza del mismo modo una revisión de los sistemas de agentes inteligentes más apropiados para el proyecto. De forma idéntica a la revisión anterior, se detalla de forma más extensa el sistema de agentes más apropiado para el proyecto.

Para finalizar el bloque se detalla de forma breve en qué consiste el aprendizaje por refuerzo en especial el algoritmo que se ha implementado para la solución del proyecto.

El segundo bloque describe el proceso de desarrollo que se ha seguido para obtener la plataforma donde poder desarrollar las soluciones. Este bloque consta de una parte donde se explica la configuración que se ha realizado en el simulador de drones AirSim. Seguidamente se detalla los pasos que se han realizado para adaptar un entorno de Unreal al uso del simulador AirSim.

Posteriormente, se expone la instalación de las librerías necesarias para el desarrollo en el lenguaje de programación Python y la conexión entre el proyecto y el simulador Airsim. En referencia a la conexión entre el proyecto y el simulador AirSim. En el siguiente punto se detalla como se ha adaptado el sistema de agentes inteligentes al proyecto y de forma conjunta al simulador AirSim.

Para finalizar el bloque, la última sección trata sobre la estructura de la solución general, cómo está organizado el código. Esto es esencial para conocer la estructura y poder realizar futuros desarrollos.

El tercer bloque describe las soluciones que se han implementado y cómo se han abordado los objetivos del proyecto. En primer lugar, la descripción de la so-

lución que permite que distintos agentes se comuniquen entre sí, y que todos los agentes o dicho de otro modo, drones, obedezcan las órdenes de movimiento establecidas por un agente *Maestro*. En segundo lugar, la descripción de la solución que implementa un algoritmo de aprendizaje por refuerzo adaptado a un drone para que tome decisiones con autonomía en relación a la conducción autónoma.

Para finalizar el documento, el último apartado reúne unas conclusiones globales acerca de todo el proyecto junto con una serie de trabajos futuros. Finalizando el documento con la bibliografía consultada en todos los bloques del documento y utilizados para el desarrollo del proyecto.

# CAPÍTULO 2
# Estado del arte

En la primera sección se hace una introducción sobre la aparición de las aeronaves y posteriormente un pequeño inciso sobre el estado del arte de las aeronaves no tripuladas en especial de los drones. A continuación se recopila los diferentes simuladores propuestos para el desarrollo del proyecto y las plataformas de agentes. Por último una breve explicación del concepto de Reinforcement Learning.

## 2.1 Inicios

En los inicios de las aeronaves no tripuladas, los drones como quirópteros aparecieron en 1907 creados por los inventores y hermanos Jacques y Louis Bréguet. Este drone tenía unas limitaciones importantes y tan solo se elevó dos pies del suelo[2].

En este tipo de industria los primeros fueron los militares, en 1917 se elevó el primer avión sin piloto Ruston Proctor Aerial Target", controlado por radiocontrol. Como objetivo tenían la misión de convertirse en bombas voladoras. Pero nunca se utilizó en combate.[3]

En 1943 se creó la primera aeronave no tripulada controlada por control remoto operativa. El FX-1400 fue la primera arma aérea que se utilizó para hundir barcos. Fue el antecesor de los actuales misiles antibuque y otro tipo de armas.[4]

Dando un salto a 2006, se emitieron los primeros permisos a los drones comerciales, eliminando restricciones aéreas.

De este modo se dieron grandes posibilidades a las empresas para desarrollar drones de uso profesional o recreativo.[5]

En 2010 salió al mercado el primer drone comercial y recreativo que permitía ser controlado por wifi. Se trata del Parrot AR Drone que recibió el premio CES Innovations 2010.[6]

## 2.2 Actualidad

El la actualidad, el uso de drones se ha ido incrementando. Aportando a gran cantidad de tareas los beneficios que permiten que un drone ayude a realizar tareas ya existentes o realizar tareas nuevas que no existían anteriormente.

En diciembre de 2016, Amazon lanzó el primer viaje de reparto que consistía en enviar un paquete desde la central de reparto hasta el cliente. Este viaje consiste en enviar un drone con el paquete donde las características del vuelos son que el drone es autónomo y no necesita un piloto.

El drone despega, vuela hasta el objetivo y una vez allí desciende sobre una plataforma proporcionada por la compañía y deja el paquete. De nuevo se eleva y vuela hasta la central de los almacenes y vuelve a aterrizar sobre la plataforma de aterrizaje. Todo esto se lleva a cabo de forma autónoma con la ayuda de distintos sensores y GPS.[7]

En febrero de 2019, se ha aprobado el primer permiso para volar al primer drone autónomo en Europa. Este avance ha ocurrido en Francia, la Dirección General de Aviación Civil de Francia (DGAC) ha aprobado el uso de el drone autónomo de la empresa Azur Drones que se dedica al sector de la seguridad.

Este drone totalmente automatizado se trata de un hito histórico en los avances del uso de los drones civiles para uso profesional. Da paso al desarrollo de nuevas aplicaciones e investigación en el campo de los vehículos aéreos no controlados por pilotos. Las características de este drone son las siguientes[8]:

- Capacidad operacional 24/7.
- Listo para actuar en menos de 30 seg.
- Gran precisión en el aterrizaje.
- 100 % automatizado.
- No requiere un piloto.

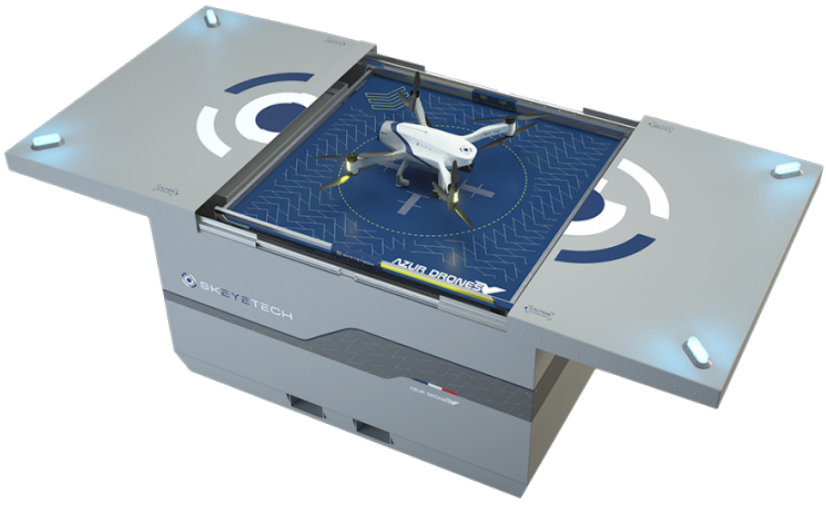



**Figura 2.1:** Drone autónomo [8]

En lo que respecta al estudio de los drones autónomos, actualmente se ha publicado un artículo el cual destaca el estudio sobre la navegación autónoma de vehículos aéreos no tripulados a gran escala con entornos complejos con enfoque en el aprendizaje profundo por refuerzo.

Este artículo cabe destacar la utilización de entornos de simulación de diferentes complejidades y la utilización de diferentes algoritmos de aprendizaje. El problema que proponen se trata de un proceso de decisión de Markov parcialmente observable. En el artículo se resuelve mediante un algoritmo de Deep Reinforcement Learning online, diseñado en base a dos teoremas probados. [9]

## 2.3 Simuladores de Drones

En el apartado de simuladores se explica algunos de los simuladores que se propusieron para realizar el desarrollo del proyecto. En especial el simulador AirSim el cual fue seleccionado.

### 2.3.1. AirSim

El simulador AirSim[10], se trata de un software desarrollado por Microsoft, capaz de simular drones y coches. AirSim se apoya de Unreal Engine para renderizar el entorno y proporcionar un escenario para la simulación. Este software es de código abierto y multiplataforma. Además es compatible con la controladora PX4, esto proporciona la posibilidad de poder migrar al mundo físico los desarrollos testeados en las simulaciones.

Al estar especialmente enfocado en la simulación para coches y drones, contiene diferentes APIs adaptadas para ello:

- Inclemencias meteorológicas, viento.
- Horario del día, día y noche.
- Integración con Unity.
- Detección de terrenos para bosques o campos.
- Tres tipos de cámaras.
- Sensor Lidar
- Capacidad de soportar múltiples vehículos en una simulación.

El software tiene una API para desarrollos implementados con Python y otra API para desarrollos implementados con C++. Estas APIs permite cualquier interacción con el entorno y con los agentes del entorno. AirSim permite la simulación de uno o varios vehículos ya sea coches o drones, por el momento no permite múltiples vehículos y distintos tan solo vehículos del mismo tipo.

El simulador AirSim tiene en proyecto ser multiplataforma, actualmente se ejecuta en Windows perfectamente, tanto el motor gráfico Unreal desarrollado

para el uso en Windows, como la base del simulador AirSim. La versión para Linux está un poco más limitada ya que AirSim funciona perfectamente pero el motor gráfico Unreal no contempla estas características a la perfección.

De este modo, el software AirSim permite la conexión a un motor gráfico mediante red, asociado a una IP. Actualmente contempla la ejecución del entorno gráfico para visualizar el simulador mediante Unity de forma experimental.

La ejecución de forma distribuida permite un mayor uso computacional a los recursos que se necesiten dependiendo del desarrollo que se implemente.

Como se ha mencionado anteriormente, AirSim permite la ejecución distribuida, perfecta para desarrollos de aplicaciones que utilizan *Machine Learning* o *Deep Learning*. AirSim también aprovecha el uso del Toolkit propio de Microsoft [11]

#### Microsoft Cognitive Toolkit

Este toolkit de Microsoft(CNTK) es de código abierto, específicamente para *deep learning*. Entre sus características, permite realizar redes neuronales con una serie de pasos que computacionalmente forman un grafo dirigido. Permite realizar redes neuronales de distintos modelos, feed-forward DNNs, redes neuronales convolucionales DNNs y redes neuronales recurrentes RNNs/LSTMs. CNTK implementa el descenso por gradiente SGD y backpropagation. Este toolkit implementa la paralelización en múltiples GPU y servidores.

#### Image API

La API de imagen se utiliza para interactuar con el entorno, se puede visualizar el entorno y obtener una imagen con esta API. Configurando las cámaras se puede obtener una imagen de la escena donde se encuentran los vehículos. Como se ha comentado con anterioridad, la API permite su uso desde Python y C++.

#### Sensores

Los sensores en AirSim por defecto para un drone son los siguientes:

- IMU: Sensor de velocidad, orientación y fuerzas gravitacionales.
- Magnetómetro: Dispositivo que mide la señal magnética que tiene como origen los polos magnéticos de la tierra.
- GPS: Sensor que permite determinar la ubicación en cualquier punto del planeta.
- Barómetro: Sensor que mide la presión atmosférica, para determinar la altura a la que se encuentra el drone.

Los sensores en un vehículo de tipo coche en AirSim son los siguientes:

- GPS: Sensor que permite determinar la ubicación en cualquier punto del planeta.

Además a los dos tipos de vehículo se puede añadir el sensor Lidar.

### Lidar

El Lidar (Light Detection and Ranging) se trata de un sensor que permite determinar la distancia que existe desde el origen o emisor del láser hasta el objeto o superficie en el que impacta el haz de luz. La distancia se determina midiendo el tiempo que tarda en ser reflejada la señal de luz. [18]

Como se puede observar en el ejemplo de la Figura 2.2 el dispositivo origen emite en todas direcciones un un haz de luz que permite detectar los objetos más cercanos indicados con una línea naranja. Todo aquello que rebasa la distancia a la que el lidar permite reconocer objetos, queda descartado.

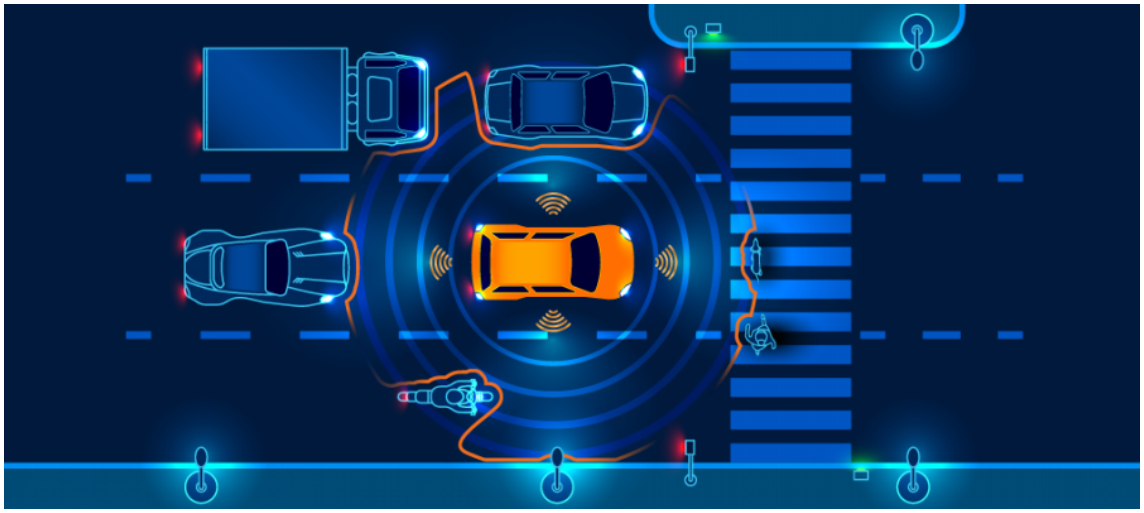


**Figura 2.2:** Lidar

En AirSim los sensores de tipo lidar tienen una configuración adicional:

- Número de canales: Hace referencia a la cantidad de líneas de puntos de forma horizontal que contiene el Lidar.
- Rango: Se trata de la distancia máxima a la que percibe la colisión del haz de luz contra un objeto.
- Puntos por segundo: Es la cantidad de puntos de luz que emite el sensor por segundo.
- X,Y,Z: Posición a la que se coloca el sensor desde el centro del vehículo.
- Visualización de los puntos de luz.

### Configuración de AirSim

El simulador AirSim tiene gran cantidad de parámetros, los cuales se pueden configurar para cada tipo de simulación. Para configurar estos parámetros que se detallan a continuación, existe un archivo ubicado en *Documents/AirSim* denominado *settings.json* 2.1.

El parámetro de configuración más importante es la selección entre *Coche* o *Multirotor*, también existe la opción *ComputerVision* que solo utiliza cámara, sin vehículo, ni físicas. está opción lleva la etiqueta *SimMode*. Seguidamente se encuentra distintos parámetros sobre la configuración del reloj del sistema para la simulación. La velocidad del reloj, que permite incrementar o disminuir las velocidades de simulación.

Para la configuración de la cámara existe la opción con etiqueta *ViewMode*. Por defecto, está opción realiza la acción seguir al vehículo con la cámara. En el caso del multirrotor el modo por defecto es *FlyWhitMe*.

Existen los siguientes modos :

- FlyWithMe: Cámara que sigue al vehículo con 6 grados de libertad.
- GroundObserver: Sigue al vehículo desde el suelo.
- Fpv: Vista desde la parte frontal del vehículo
- Manual: La cámara no se mueve automáticamente. Se utiliza los controles ASDW para moverla.
- SpringArmChase: Sigue al vehículo desde un brazo invisible.
- NoDisplay: Este modo se utiliza para optimizar recursos.

La utilización de la API del Tiempo va ligada a la configuración en la etiqueta *TimeOfDay*. Se utiliza para realizar una simulación del movimiento del sol. Indicando la hora de la ejecución el sol se sitúa en la posición adecuada como si del mundo real se tratase.

*OriginGeopoint*, se la etiqueta de configuración que permite seleccionar las coordenadas donde van a iniciar cada uno de los vehículos de la simulación.

De forma adicional a la cámara, existe la posibilidad de visualizar 3 tipos de cámara en la parte inferior del display. Para activar las cámaras se pulsan las teclas del 0 al 3 para activar o desactivar dichas cámaras.

- 0: Activar o desactivar todas las cámaras.
- 1: Activar o desactivar la cámara de profundidad.
- 2: Activar o desactivar la cámara de segmentación.
- 3: Activar o desactivar la cámara frontal FPV.

Otro apartado importante se trata de las configuración de los vehículos. Cada vehículo permite la configuración de todos los sensores, cámaras, etc.

Los vehículos pueden ser de 4 tipos:

- PhysXCar: Para seleccionar un coche.
- SimpleFlight: Para seleccionar un drone.
- PX4Multirotor: Para seleccionar un drone que tiene incorporada la controladora PX4, de este modo se puede migrar a una configuración en un drone real.
- ComputerVision: Si se ha elegido el modo de *ComputerVision*.

Al iniciar, los drones pueden iniciar con los motores armados o desarmados. está configuración se determina desde la etiqueta *DefaultVehicleState*. Asociada a esta, hay una etiqueta denominada *AutoCreate* que permite la activación para que el vehículo se genere si se trata de un vehículo compatible con el modo de simulación.

Los controles a distancia con Radio Control (RC) está soportado por AirSim, por ello existe una etiqueta para dicha configuración *RC*.

En cuanto a la posición inicial en la que se encuentra el vehículo al iniciar en el entorno, vienen dadas por las siguientes etiquetas para la configuración: X, Y, Z, Yaw, Roll, Pitch.

Los vehículos permiten varias configuraciones de cámara, una de ellas puede ser FPV como se determina en *ViewMode* que va asociada al vehículo que lleva la cámara indicándose en la configuración *IsFpvVehicle*. Las cámaras pueden configurar la posición, ángulo de visión, tipo de cámara, etc.

A continuación en la siguiente figura 2.1 se muestra un ejemplo de las configuraciones posibles del simulador con el fichero *settings.json*

```
{
  "SimMode": "",
  "ClockType": "",
  "ClockSpeed": 1,
  "LocalHostIp": "127.0.0.1",
  "RecordUIVisible": true,
  "LogMessagesVisible": true,
  "ViewMode": "",
  "RpcEnabled": true,
  "EngineSound": true,
  "PhysicsEngineName": "",
  "SpeedUnitFactor": 1.0,
  "SpeedUnitLabel": "m/s",
  "Recording": {
    "RecordOnMove": false,
    "RecordInterval": 0.05,
    "Cameras": [
        { "CameraName": "0", "ImageType": 0, "PixelsAsFloat": false, "
            Compress": true }
    ]
  },
  "CameraDefaults": {
    "CaptureSettings": [
      {
        "ImageType": 0,
        "Width": 256,
        "Height": 144,
```

```
        "FOV_Degrees": 90,
        "AutoExposureSpeed": 100,
        "AutoExposureBias": 0,
        "AutoExposureMaxBrightness": 0.64,
        "AutoExposureMinBrightness": 0.03,
        "MotionBlurAmount": 0,
        "TargetGamma": 1.0,
        "ProjectionMode": "",
        "OrthoWidth": 5.12
      }
    ],
    "NoiseSettings": [
      {
        "Enabled": false,
        "ImageType": 0,

        "RandContrib": 0.2,
        "RandSpeed": 100000.0,
        "RandSize": 500.0,
        "RandDensity": 2,

        "HorzWaveContrib":0.03,
        "HorzWaveStrength": 0.08,
        "HorzWaveVertSize": 1.0,
        "HorzWaveScreenSize": 1.0,

        "HorzNoiseLinesContrib": 1.0,
        "HorzNoiseLinesDensityY": 0.01,
        "HorzNoiseLinesDensityXY": 0.5,

        "HorzDistortionContrib": 1.0,
        "HorzDistortionStrength": 0.002
      }
    ],
    "Gimbal": {
      "Stabilization": 0,
      "Pitch": NaN, "Roll": NaN, "Yaw": NaN
    }
    "X": NaN, "Y": NaN, "Z": NaN,
    "Pitch": NaN, "Roll": NaN, "Yaw": NaN
  },
  "OriginGeopoint": {
    "Latitude": 47.641468,
    "Longitude": -122.140165,
    "Altitude": 122
  },
  "TimeOfDay": {
    "Enabled": false,
    "StartDateTime": "",
    "CelestialClockSpeed": 1,
    "StartDateTimeDst": false,
    "UpdateIntervalSecs": 60
  },
  "SubWindows": [
    {"WindowID": 0, "CameraName": "0", "ImageType": 3, "Visible": false
        },
    {"WindowID": 1, "CameraName": "0", "ImageType": 5, "Visible": false
        },
```

```
    {"WindowID": 2, "CameraName": "0", "ImageType": 0, "Visible": false
        }
  ],
  "SegmentationSettings": {
    "InitMethod": "",
    "MeshNamingMethod": "",
    "OverrideExisting": false
  },
  "PawnPaths": {
    "BareboneCar": {"PawnBP": "Class'/AirSim/VehicleAdv/Vehicle/
      VehicleAdvPawn.VehicleAdvPawn_C'"},
    "DefaultCar": {"PawnBP": "Class'/AirSim/VehicleAdv/SUV/SuvCarPawn.
      SuvCarPawn_C'"},
    "DefaultQuadrotor": {"PawnBP": "Class'/AirSim/Blueprints/
      BP_FlyingPawn.BP_FlyingPawn_C'"},
    "DefaultComputerVision": {"PawnBP": "Class'/AirSim/Blueprints/
      BP_ComputerVisionPawn.BP_ComputerVisionPawn_C'"}
  },
  "Vehicles": {
    "SimpleFlight": {
      "VehicleType": "SimpleFlight",
      "DefaultVehicleState": "Armed",
      "AutoCreate": true,
      "PawnPath": "",
      "EnableCollisionPassthrogh": false,
      "EnableCollisions": true,
      "AllowAPIAlways": true,
      "RC": {
        "RemoteControlID": 0,
        "AllowAPIWhenDisconnected": false
      },
      "Cameras": {
        //same elements as CameraDefaults above, key as name
      },
      "X": NaN, "Y": NaN, "Z": NaN,
      "Pitch": NaN, "Roll": NaN, "Yaw": NaN
    },
    "PhysXCar": {
      "VehicleType": "PhysXCar",
      "DefaultVehicleState": "",
      "AutoCreate": true,
      "PawnPath": "",
      "EnableCollisionPassthrogh": false,
      "EnableCollisions": true,
      "RC": {
        "RemoteControlID": -1
      },
      "Cameras": {
        "MyCamera1": {
          //same elements as elements inside CameraDefaults above
        },
        "MyCamera2": {
          //same elements as elements inside CameraDefaults above
        },
      },
      "X": NaN, "Y": NaN, "Z": NaN,
      "Pitch": NaN, "Roll": NaN, "Yaw": NaN
    }
  }
```

```
137 }
```

**Listing 2.1:** Settings.json

### 2.3.2. Otros simuladores

Existen diferentes simuladores para realizar tareas similares a la que se desarrolla en este proyecto. A continuación se mencionan algunos de los simuladores que se asemejan en cuanto a características al simulador seleccionado AirSim.

#### ROS

Se trata de un sistema operativo orientado a robots. ROS (Robot Operating Sistem), contiene herramientas y librerías para realizar desarrollos de aplicaciones para robots. [12]

ROS es, en sí, una capa que permite desarrollar aplicaciones para robots. Se trata de una capa de alto nivel de este modo no es necesario conocer el hardware sobre el cual se están desarrollando aplicaciones. Esto en robots es algo muy importante ya que cada motor, cámara, sensores son de un fabricante y con un hardware distinto.

A grandes rasgos, ROS funciona con nodos de suscripción y publicación con tópicos. Cada uno de los nodos formará parte del sistema interactuando cada nodo con cada uno de los elementos del robot: una cámara, un motor, un sensor, etc. Para gestionar la información que se transmite entre estos nodos existe un nodo máster que se encarga de identificar cada uno de los nodos y comunicar al resto si están disponibles. Con la cooperación de todos los nodos, se realizan las acciones de un robot.

Este sistema es muy útil para casos de programación de robots y por ello también de drones. Además tiene la posibilidad de realizar proyectos desarrollados en Python.

#### Gazebo

Gazebo se trata de un entorno de visualización del sistema ROS. Es una de las herramientas esenciales para un buen diseño que permite probar rápidamente desarrollos como diseño de robots, test de algoritmos, entrenamientos de inteligencia artificial, etc. Se trata de un software de código libre y tiene una comunidad importante.[22]

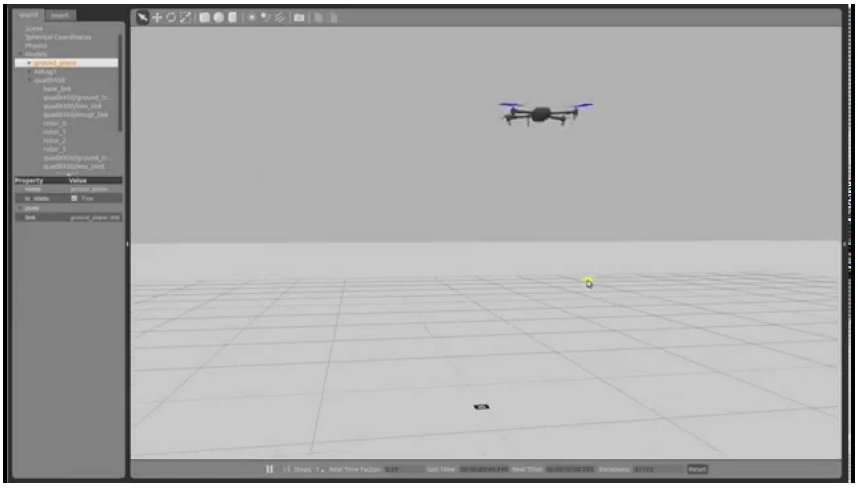


**Figura 2.3:** Entorno de simulación Gazebo

### V-REP

Una de las interfaces de visualización del entorno que pertenece al sistema ROS es V-REP. [23] Este framework contiene cinco APIs:

- The regular API
- The remote API
- The ROS interfaces
- The BlueZero interface
- The auxiliary API

*The regular API* es la API normal que contiene varias funciones que se invocan de de C/C++. Las funciones de esta API tienen un prefijo común, todas empiezan denominándose por 'sim' o '_sim'.

*The remote API* es la API que permite la comunicación entre V-REP y una aplicación externa, es multiplataforma. admite llamadas de servicios y tiene transmisión de datos bidireccional.

*The ROS interfaces* es la API que se encarga de la comunicación con el sistema ROS.

*The BlueZero interface* es un middleware multiplataforma que permite interactuar entre hilos de ROS, múltiples procesos y múltiples máquinas. está basado en ZeroMQ y orientado a cliente servidor y publicados suscriptor.

*The auxiliary API* es a API que permite algunas funciones auxiliares. Por ejemplo *External kinematics* es un conjunto de funciones que contiene esta API.

En definitiva, se ha seleccionado el simulador que destaca respecto a los demás. En este proyecto se necesita el uso de las tecnologías de *Machine Learning* y *Deep Learning*, como se explica en los siguientes secciones. Las plataformas de más adaptadas a estas tecnologías están en lenguaje Python, por ello se necesita

un simulador compatible con el lenguaje de programación Python. Para este requisito, el simulador más apropiado es AirSim. Además, AirSim permite visualizar los resultados de forma más agradable que el resto de simuladores. Junto con todas las posibilidades de compatibilidad con otros entornos. Dicho de otra forma, AirSim es el simulador más adecuado para un proyecto de estas características.

## 2.4 Plataformas de agentes

En esta sección se ofrecen las características de la plataforma de agentes SPADE que se ha seleccionado para el proyecto y seguido se detallan algunas plataformas de agentes más utilizadas en el lenguaje de programación Java.

### 2.4.1. Spade

Spade se trata de un entorno de agentes inteligentes desarrollado en Python. [14] Es una plataforma de sistemas multiagente desarrollada en Python y basada en mensajería instantánea con el protocolo XMPP.

XMPP es un protocolo de comunicación de mensajes instantáneos, abierto y extensible, está basado en XML. XMPP hereda las características de XML, su sencillez y adaptabilidad. Para su uso se utiliza servidor, existen algunos gratuitos, los puertos de conexión son 5222 de Cliente a Servidor y 5269 de Servidor a Cliente.

Volviendo a las características de Spade, la plataforma que se ha utilizado para implementar los Agentes en este proyecto. A continuación se detallan algunas de sus características.

- Plataforma multiagente basada en el protocolo XMPP.
- Notificaciones de presencia permite al sistema conocer el estado de los agentes en tiempo real.
- Utiliza Python 3.6 o mayor.
- Basado en Asyncio.
- Modelo de agentes basado en *behaviours*.
- Soporta FIPA en los mensajes en XMPP.
- Interfaz web.
- Puede utilizar cualquier servidor XMPP.

Los agentes en SPADE están compuestos por un emisor de mensajes y un conjunto de comportamientos los cuales reciben los mensajes. Cada uno de los agentes está identificado por un ID denominado JabberID en la plataforma SPADE se conoce como JID, junto con una contraseña para realizar la conexión al protocolo XMPP.

Los identificadores JID son únicos para cada agente y están compuestos por

*nombre_de_usuario@servidor_del_dominio*

Las comunicaciones en SPADE se realizan de forma interna utilizando el protocolo de comunicaciones instantáneas XMPP. Este protocolo es el mecanismo para registrar y autenticar los usuarios en un servidor XMPP.

Cuando la conexión junto con el registro son aceptados, cada agente mantiene un canal de comunicación XMPP abierto con la plataforma SPADE. Todo este proceso se realiza automáticamente y forma parte del registro de los agentes en la plataforma.

En cuanto a la forma de enviar y recibir mensajes, cada agente tiene un componente que se encarga de la administración de los mensajes. A modo de cartero, al recibir un mensaje lo coloca en el buzón y al enviar mensajes se encarga de colocar el mensaje en la cola de envíos. El envío de mensajes se realiza desde la biblioteca de SPADE automáticamente cada vez que se envía o recibe un mensaje.

Los agentes pueden ejecutar varios comportamientos simultáneamente. Los comportamientos son tareas que puede realizar un agente de forma repetitiva. En SPADE hay algunos comportamientos predefinidos:

- Cyclic
- One-Shot
- Periodic
- Time-out
- Finite
- State Machine

Estos tipos de comportamiento ayudan a la implementación de nuevas tareas que puede ejecutar un agente. Los comportamientos repetitivos se realizan utilizando *Cyclic*, de forma repetitiva y *Periodic*, de forma repetitiva también pero con un intervalo desde una tare a la siguiente. Los comportamientos que se realizan de forma causal son One-shot, para realizar la tarea una sola vez y Time-Out, de la misma forma que la anterior pero si llega a un tiempo máximo de espera ya no se realiza. Los comportamientos Finite y State Machine, permiten construir comportamientos más complejos.

Los agente pueden tener todos los comportamientos que necesiten. Cuando un agente recibe un mensaje el administrador de mensajes se encarga de colocar este mensaje en el buzón de mensajes recibidos del comportamiento adecuado. Los comportamientos tiene asociada una plantilla de mensajes, por lo tanto el administrador utiliza esta plantilla de mensajes para distribuir los mensajes de forma correcta.

### 2.4.2. Otras plataformas de agentes

A continuación se realiza una breve explicación de otras plataformas de agente más utilizadas.

### Jason

Jason se trata de un framework multiagente. Ellos se auto denominan el primer intérprete completo a partir de una versión mejorada de AgentSpeak[26].

AgentSpeak es un lenguaje de programación orientado a agentes, basado en el paradigma de programación de creencias, deseos e intenciones (BDI).

Existen varias implementaciones de sistemas BDI pero una de las características de AgentSpeak es la base teórica, es una implementación de la semántica operacional. Junto con las características de AgentSpeak, Jason está implementado con Java con lo cual es multiplataforma. Además contiene gran cantidad de características:

- Manejo de fallos en el plan.
- Comunicación entre agentes, basada en actos de habla, anotaciones y creencias de fuentes de información.
- Anotaciones en las creencias y en los planes
- Soporte para entornos de desarrollo.
- Soporte para organizaciones más y agentes que razonan utilizando un modelo Moise+.
- Posibilidad de ejecutar un sistema de múltiples agentes distribuido a través de una red utilizando Saci o Jade.
- Funciones de selección personalizables en Java.
- Una librería de acciones internas.
- Extensibilidad mediante acciones internas definidas por el usuario programadas en Java.
- Un IDE como complemento jEdit o en Eclipse, que contiene una ayuda para la depuración de código.

### Jade

Jade es un middleware de código abierto y libre que permite el desarrollo y mantenimiento de sistemas multi-agente. Proporciona un entorno de desarrollo y un entorno de ejecución, el entorno de desarrollo lo forma un conjunto de librerías desarrolladas en Java que permite la implementación de agente de manera fácil e independiente de la plataforma sobre la que se ejecuta la solución.

El entorno de ejecución permite que los agentes se ejecuten y se comunique entre ellos. El middleware está desarrollado completamente en Java y permite un conjunto de herramientas gráficas útiles y de uso fácil que permiten al desarrollador interpretar los resultados y depurar en tiempo de ejecución.[24][27]

Jade proporciona una plataforma FIPA(Foundation for Intelligent Physical Agents) para la ejecución de los agentes. FIPA por una parte es un organismo

para el desarrollo y establecimiento de estándares para agentes. Y los estándares denominados FIPA que han establecido son estándares que proporcionan una heterogeneidad en los agentes para desarrollos de software que interactúan con agentes y sistemas basados en agentes.

Jade es una plataforma distribuida que por cada host contempla un contenedor en el que se ejecutan los agentes. Además contiene una serie de herramientas que permiten la depuración y la posibilidad de ejecuciones paralelas de los agentes. Los contenedores principales contiene dos agentes.

- DF: *Directory Facilitator*, proporciona en un directorio lo agentes que hay disponibles.
- AWM: *Agent Management System*, es el encargado de controlar la plataforma. Administra la creación y eliminación de agentes y contenedores.

Los agentes en Jade tienen un ciclo de vida propuesto por el estándar FIPA:

- 1. Iniciado. El agente está creado pero no ha sido registrado por AWM.
- 2. Activo. El agente ya ha sido registrado y puede comunicarse con otros agentes.
- 3. Suspendido. El agente está parado, por su hilo de ejecución está suspendido.
- 4. Esperando. El agente se encuentra a la espera de algún suceso.
- 5. Eliminado. El agente ha sido eliminado y no está registrado por AWM.
- 6. Tránsito. El agente está migrando a una nueva ubicación.

Para finalizar esta serie de plataformas de agentes inteligentes. se determinan las características por las cuales se ha seleccionado la plataforma de agentes SPADE.

Las causas por las cuales se ha seleccionado SPADE, de nuevo es por que esta implementado en Python como ha ocurrido con anterioridad en la selección de simulador. En la próxima sección se realiza un inciso en la técnica de Reinforcement Learning y los algoritmos utilizados.

## 2.5 Reinforcement Learning

El aprendizaje por refuerzo se trata de una técnica que recopila estado y acciones que se han realizado y se evalúe el resultado obtenido. Este resultado denominado recompensa se maximiza seleccionando como mayor calidad las acciones que en un determinado estado obtienen una mejor recompensa. De forma distinta al aprendizaje supervisado en el cual existe alguien que determina las mejores acciones ante un estado. En el aprendizaje por refuerzo se evalúa el resultado de la acción tomada ante un estado.[19] [20]

En este método de aprendizaje no se indican las acciones que se pueden ejecutar en los estados, sino que se da la posibilidad al sistema de realizar todas las acciones posibles y observar el resultado obtenido, de este modo el sistema explora todas las posibilidades en el entorno.

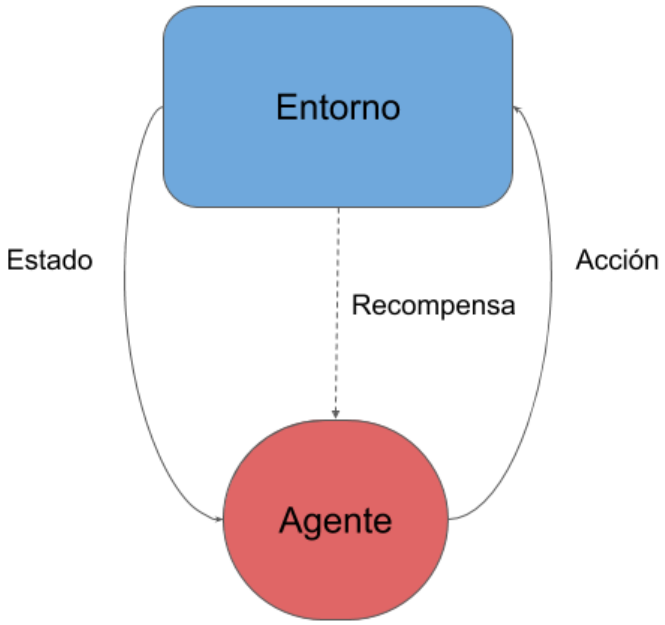



**Figura 2.4:** Reinforcement Learning

Los sistemas de aprendizaje por refuerzo tienen los siguientes elementos:

- Agente
- Entorno
- Políticas
- Función valor

El agente es el que interactúa con el entorno y realiza acciones de las cuales aprende.

El entorno es el sistema que interactúa con el agente. El agente realiza una acción, esta acción tiene una consecuencia en el entorno en el cual se encuentra el agente y se observa su repercusión.

Las políticas hacen referencia a la forma de comportarse el agente. Una política es la consecuencia que tiene una acción ante un estado del agente en un entorno. Al fin y al cabo la política es lo que da sentido al agente sobre la toma de decisiones y determinan su comportamiento.

La función de recompensa determina el objetivo del aprendizaje por refuerzo. está función se encarga de obtener un valor por cada acción en un estado. El objetivo del agente es maximizar el valor total de la recompensa recibida. La función

valor va ligada a la función recompensa de modo que la función recompensa obtiene lo bueno o malo que es una acción en un estado en sentido inmediato y la función valor obtiene lo bueno o malo que es a largo plazo una acción. El valor en un estado es la cantidad total de recompensa que puede esperar acumular en el futuro desde el estado actual.

El entorno tiene que ser explorado por el agente con el objetivo de aprender el conocimiento al seleccionar las mejores acciones para el futuro. El agente tiene que seleccionar las acciones con recompensa más alta pero a su vez tiene que descubrir la recompensa de las acciones no ejecutadas hasta el momento.

En el aprendizaje por refuerzo existen dos problemas principales, el problema de la asignación temporal y el problema estructural. En cuanto al problema temporal es a causa de la obtención de la recompensa puede no ser inmediata y obtenida con retraso. El valor de recompensa o refuerzo viene dado después de realizar una serie de acciones. El problema se encuentra en determinar cuál es la recompensa de una acción. Para solucionar este problema existen algunos algoritmos y uno de ellos es Q-learning [21] y su extensión el DQN[25].

### 2.5.1. DQN

Existen varios algoritmos de aprendizaje por refuerzo basados en la función valor o recompensa, pero uno de los más extendidos es el siguiente, se trata del algoritmo DQN[15][21][25].

En la inicialización del algoritmo DQN se definen valores denominados $Q$ *values*, compuestos por el estado $s$ y la acción $a$, tal que:

$$Q^{\pi}(s,a) = E[R|a,s,\pi]$$

El valor óptimo de la función $Q$ se denomina la ecuación de Bellman, que encuentra el valor óptimo de $Q, Q^*(s_t, a_t)$ teniendo en cuenta que la estrategia óptima es seleccionar la acción siguiente $a_{t+1]}$ que maximiza el valor esperado de $r + \gamma Q^*(s_{t+1}, a_{t+1})$:

$$Q^*(s_t,a_t) = E[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})|s_t, a_t]$$

Las técnicas que se utilizan para tratar de calcular o estimar los valores $Q$ se denominan Aprendizaje-Q (*Q-learning*). Estas técnicas se basan en el paso anterior para estimar el valor $Q$ de forma iterativa, en el paso $i$, tal que:

$$Q_{i+1}(s_t, q_t) \leftarrow Q_i(s_t, a_t) + \alpha * (r_t + \gamma * max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Siendo $\alpha$ un valor positivo y menor o igual a 1, se denomina *Learning Rate*, decide la cantidad de cambio de se realiza en al algoritmo por cada iteración.

Las diferencias que existen dentro de los algoritmos de aprendizaje por refuerzo dentro de los *Q-learn* es la técnica que utilizan para calcular o estimar el valor de la función $Q$. En este caso el algoritmo DQN( *Deep Q-Network*) utiliza una red

neuronal para obtener estos valores $Q$, esta red neuronal se entrena mediante la expresión anterior. Una vez estimados los valores $Q$ de los estados, se pueden utilizar para aproximar a una política óptima. La forma de hacerlo es seleccionando la acción con el valor máximo de $Q(s_t, a_t)$ .

La exploración en el algoritmo DQN se realiza respecto a una técnica denominada *Epsilon-greedy*. En está técnica se define un $\epsilon$ que es un término utilizado en el entrenamiento. Cuando se decide que acción tomar, se toma la acción indicada por la política del agente con una probabilidad $1 - \epsilon$ o una acción aleatoria seleccionadas uniformemente con probabilidad $\epsilon$ .

### 2.5.2. Conclusiones

En este capítulo del estado del arte se ha planteado la selección de plataformas y tecnologías adecuadas para el desarrollo del proyecto. Tras una breve introducción a los inicios de los drones. Se han mostrado las diferentes etapas de evolución de los drones y, en la actualidad, de los drones autónomos que forman parte de uno de los objetivos del proyecto. El catálogo de simuladores de drones para el desarrollo de aplicaciones es amplio y de plataformas de agentes inteligentes también. Pero las facilidades para la interconexión entre estos dos tipos de tecnologías no son tan amplias.

Para finalizar se concluye con la determinación de la adecuación de todas las opciones seleccionadas, tanto por la parte del simulador como por parte de la plataforma de agentes inteligentes. Las dos opciones seleccionadas (AirSim y Spade) tienen en común la utilización del lenguaje de programación Python. Además de una API muy útil en cada caso que simplifica el proceso de integración en una única solución. En el próximo capítulo se detalla la solución que se ha implementado a nivel técnico.

# CAPÍTULO 3
# Desarrollo

En el presente capítulo se detalla a nivel técnico toda la implantación y configuración que se ha realizado para obtener las soluciones propuestas en el proyecto. De esta forma se plantea el desarrollo a nivel de configuración del simulador, del entorno, de las conexiones e instalación de librerías, de las adaptaciones entre los distintos softwares y estructuras del desarrollo del código en el lenguaje de programación Python.

## 3.1 Simulador

En el proyecto, se ha utilizado el simulador AirSim mencionado en el Capítulo 2. A continuación, se detalla como se ha utilizado el simulador y las configuraciones que se han empleado.

En primer lugar, se ha utilizado el mapa de bloques. Este mapa consta de una serie de bloques apilados que se asemejan a unos edificios. Entre estos edificios se quedan separaciones a modo de calles. Siendo de este modo un mapa simple que no requiere gran cantidad de recursos de hardware para renderizar. Por estas características se escogió el mencionado mapa.

La configuración del simulador ha sido la proporcionada en Listing 3.1 La configuración de tipo de vehículo para el simulador está definida como "Multirrotor". Existen dos vehículos de tipo drone denominados "Drone_1z "Drone_2". Estos vehículos tiene añadidos 2 sensores, un lidar y un GPS. Los sensores hay que denominarlos para hacer referencia a ellos. La nomenclatura que se ha utilizado es la siguiente, nombre del sensor seguido del número de identificador del drone al que pertenece.

Entrando en detalle, los sensores de tipo lidar tienen una configuración adicional. Como se indica en la Subsección 2.3.1. En este caso se han configurado los canales a 4, rango a 20, puntos por segundo 10000, con una orientación hacia abajo de forma que el lidar queda observando la superficie del suelo.

Por el contrario,el sensor de tipo GPS no contiene parámetros de configuración adicionales al resto de sensores. Para finalizar con las propiedades del vehículo, se indica la posición y orientación del vehículo en el espacio del simulador. En este caso el Drone 1 se encuentra en las coordenadas $X = 4$, $Y = 0$, $Z = -2$, esta última coordenada se encuentra en la parte superior del plano,

se desconoce el significado de la representación con valores negativos a la parte superior del plano dentro del simulador.

En cuanto al segundo drone, se ofrece con las mismas características y configuraciones pero desplazado en las coordenadas a $X = 4, Y = 4, Z = -2$, de este modo quedando posicionado a la derecha del drone 1.

Por último, la cámara se configura como se indica en la Subsección 2.3.1. La configuración de los parámetros de la cámara se puede observar en Listing 3.1.

```
{
  "ClockSpeed": 1,
  "SeeDocsAt": "https://github.com/Microsoft/AirSim/blob/master/docs/
     settings.md",
  "SettingsVersion": 1.2,
  "SimMode": "Multirotor",
  "Vehicles": {
    "Drone_1": {
      "VehicleType": "SimpleFlight",
      "Sensors": {
        "MyLidar1_1": {
          "SensorType": 6,
          "Enabled": true,
          "NumberOfChannels": 4,
          "Range":20,
          "PointsPerSecond": 10000,
          "X": 0,
          "Y": 0,
          "Z": -1,
          "DrawDebugPoints": true
        },
        "GPS_1": {
          "SensorType": 3,
          "Enabled" : true
        }
      },
      "X": 4.0,
      "Y": 0.0,
      "Z": -2,
      "Pitch": 0,
      "Roll": 0,
      "Yaw": 0
    },
    "Drone_2": {
      "VehicleType": "SimpleFlight",
      "Sensors": {
        "MyLidar2_1": {
          "SensorType": 6,
          "Enabled": true,
          "NumberOfChannels": 4,
          "Range":20,
          "PointsPerSecond": 10000,
          "X": 0,
          "Y": 0,
          "Z": -1,
          "DrawDebugPoints": true
        },
        "GPS_2": {
          "SensorType": 3,
```

```
            "Enabled" : true
          }
        },
        "X": 4.0,
        "Y": 4.0,
        "Z": -2,
        "Pitch": 0,
        "Roll": 0,
        "Yaw": 0
      },
      "Cameras": {
        "camera_1": {
          "CaptureSettings": [
            {
              "ImageType": 0,
              "Width": 672,
              "Height": 376,
              "FOV_Degrees": 90,
              "TargetGamma": 1.5
            },
            {
              "ImageType": 1,
              "Width": 672,
              "Height": 376,
              "FOV_Degrees": 90,
              "TargetGamma": 1.5
            }
          ],
          "X": 0.5,
          "Y": -0.06,
          "Z": 0.10,
          "Pitch": 0.0,
          "Roll": 0.0,
          "Yaw": 0.0
        }
      }
    }
}
```

**Listing 3.1:** Settings.json

## 3.2 Entorno

El simulador AirSim hace uso de los entorno de Unreal para la simulación. Para utilizar un entorno en una simulación de AirSim es necesario configurarlo correctamente. A continuación se detallan los pasos que se han seguido para realizar correctamente la instalación y la configuración del entorno basándose en la documentación oficial de simulador AirSim.[10]

En primer lugar hay que mencionar una característica de este modelo de simulación, para los usuarios que utilizan Linux, de momento no se encuentra disponible la *customización* (personalización) de un entorno.

De lo contrario, para usuarios que utilizan el sistema operativo Windows, tienen disponibles todas las herramientas para la personalización de cualquier entorno de Unreal.

### 3.2.1. Instalación de AirSim

Como se indica en la documentación, el primer requisito es instalar Unreal con la versión **Unreal 4.18**. Para obtener Unreal, hay que descargar *Epic Games Lanuncher* [16].

Al acceder al software, entramos en el apartado de *Library* y se añade una versión de Unreal, en está ocasión la versión 4.18 y se instala automáticamente.

Para crear un entorno de Unreal para utilizarlo desde AirSim, en la documentación se indica que hay que instalar *Visual Studio 2017*, junto con *VC++* y *Windows SDK 8.1*.

Desde la terminal de comandos *x64 Native Tools Command Prompt for VS 2017* se ejecutan los siguientes comandos para la descarga del proyecto desde GitHub.

```
git clone https://github.com/Microsoft/AirSim.git
cd AirSim
build.cmd
```

Ejecutados correctamente los comandos anteriores, se accede a la carpeta

```
Unreal\Environments\Blocks
```

En está carpeta se ha generado un archivo *".sln"* el cual se encarga de cargar el proyecto del entorno en *Visual Studio*. Cargado el proyecto en *Visual Studio* hay que cambiar el modo de ejecución. El modo de ejecución que hay que indicar es *"Develop Editor"* y *"Win64"*. Tras ejecutar la solución del proyecto, se genera un archivo ejecutable. Este archivo se utiliza para iniciar el simulador como se observa en la siguiente Figura 3.1.

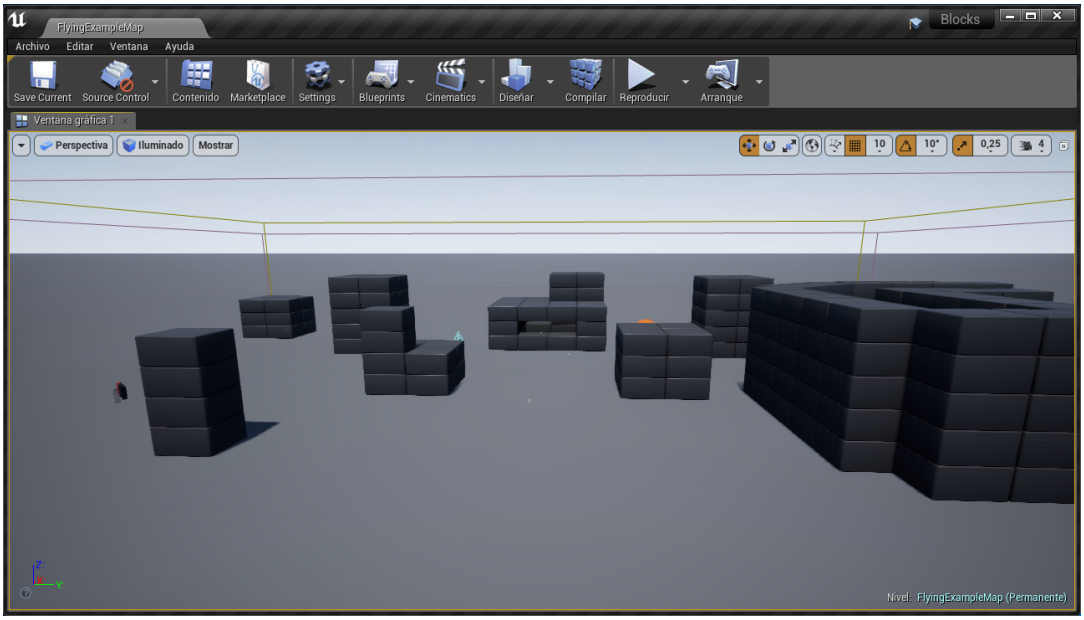



**Figura 3.1:** Entorno Bloques

El entorno que se visualiza en la Figura 3.1 se trata del mundo de bloques mencionado anteriormente. Este entorno se ha modificado para la realización de experimentos durante el desarrollo que se detallan en el Capítulo 4.

### 3.2.2. Personalización del Entorno

En este apartado se trata las posibilidades de personalización de los entornos, que permite Unreal.

Por una parte Unreal permite todo tipo de entornos que sean compatibles con Unreal del mismo modo ArSim es compatible con los entornos que estén diseñados para Unreal. Algunos de los canales oficiales donde se pueden descargar entornos gratuitamente es la tienda de Unreal "*Unreal Marketplace*"[17]. Además, en la documentación de AirSim se citan varias web donde descargar entornos compatibles con Unreal y AirSim.

Una vez descargado el entorno seleccionado, en el software *Epic Games Launcher* se accede al menú *Learn* y buscamos el entorno descargado. En nuestro caso se trata del entorno *Landscape Mountains* el que contiene una serie de montañas, ríos, árboles, valles, etc. Al encontrar nuestro entorno seleccionado, se crea un proyecto desde sus opciones.

Este proyecto genera unos archivos con una nomenclatura de este modo:

$$\{nombre\_entorno\}.uproject$$

Abriendo este archivo se inicia Unreal, desde el cual se permite eliminar, cambiar y añadir nuevos elementos al diseño del entorno. Para el uso del entorno Unreal en el simulador AirSim, es necesario añadir una clase en C++ y guardar la solución del entorno.

A continuación, desde la carpeta donde se ha generado el archivo *.uproject* con el botón derecho del ratón desplegamos el menú y seleccionamos la opción *Generate Visual Studio project files*. Esto genera los archivos necesarios para abrir Visual Studio con el proyecto del entorno seleccionado. Ejecutamos el modo de *"Develop Editor"* y *"Win64"* como se indica en la Subsección 3.2.1. De este modo se generan los archivos necesarios para ejecutar el entorno de simulación con el nuevo escenario.

Para finalizar el apartado, se menciona que el tipo de vehículo con el cual se generan las simulaciones se puede modificar con distintos parámetros, esto se describe en la siguiente Sección 3.3 .

## 3.3 Conexión del simulador e instalaciones

En esta Sección se realizará una explicación detallada de los pasos que se han seguido para realizar las conexiones desde el código de la solución propuesta desarrollada en Python hasta el entorno de simulación ejecutado desde Unreal junto con AirSim y las librerías que se han utilizado para el desarrollo de las dos soluciones propuestas.

En primer lugar, se detallarán las instalaciones que se han realizado. Como principal librería del proyecto y desde la cual se gestiona los agentes inteligentes y las funciones y comportamientos que estos adoptan, SPADE. Anteriormente se ha detallado en la Subsección 2.4.1 las características más importantes y las utilidades de esta plataforma. En el actual proyecto se ha explotado el uso de los agentes para la comunicación entre ellos y para la ejecución de acciones como consecuencia de comportamientos.

Comando instalación SPADE:

```
pip install spade
```

La API del simulador AirSim es necesaria para desarrollar proyectos que tengan interacción con el simulador AirSim. Como se ha explicado anteriormente en la Subsección 2.3.1 todas las características de este simulador, a continuación detallaremos la configuración que se ha utilizado para el desarrollo del proyecto. La instalación de AirSim para Python es muy sencilla, tan solo hay que ejecutar el comando indicado y se instala automáticamente.

Comando instalación AirSim:

```
pip install AirSim
```

A partir de ahora ya tenemos el acceso a las librerías que permiten el control del simulador AirSim. Administración de las acciones de los vehículos, sensores, modificaciones en el entorno. Uso de las APIs asociadas al simulador AirSim como se detalla en la Subsección 2.3.1.

Por otra parte, la segunda solución que se propone en este proyecto, que mencionaremos más adelante en la Sección 4.2, utiliza el framework de *Machine Learning* PyTorch.

Comando instalación Pytorch:

```
pip3 install torch==1.3.1+cpu torchvision==0.4.2+cpu -f https://
    download.pytorch.org/whl/torch\_stable.html
```

El modo de selección de la versión que se quiere instalar según las características de la máquina es muy intuitiva. Como se puede observar el la Figura 3.2, se introducen las característica tales que:

- Versión de PyTorch = Stable.
- Sistema Operativo = Windows.
- Package = Pip.
- Language = Python 3.7
- Cuda = None


**Figura 3.2:** Instalación PyTorch

Por último instalamos la librería OpenCV para python, se trata de una librería orientada a la visión por computador y al tratamiento de imágenes.

Comando instalación OpenCV:

```
pip install opencv-python
```

Una vez realizada la instalación de todas las librerías necesarias para ejecutar el proyecto, se pasa a explicar como se ha realizado la conexión desde la solución del proyecto al simulador AirSim desde Unreal.

Para iniciar una simulación, es necesario que el entorno en Unreal esté ejecutándose y el drone listo para conectar desde la API de Python.

```
client = airsim.MultirotorClient()
client.confirmConnection()
```

**Listing 3.2:** Conexión al simulador AirSim

Ejecutando desde Python, se conecta al simulador y al drone. Desde la variable *client* se permite realizar las acciones pertinentes que repercuten en el control del dron. Estas funciones se explican más adelante en la Subsección 3.5.3.

En el proyecto, esta conexión se realiza de modo que cada agente realiza la conexión con un drone, quedando asignados un drone por agente. Esta implementación se detalla a continuación en la Sección 3.4.

## 3.4 Adaptación a Spade

En la plataforma SPADE los agentes tienen uno o más comportamientos. Estos comportamientos definen las acciones que realizan y la forma de actuar de los agentes y por extensión de los drones. La manera en que se ha decidido implementar esta adaptación entre la estructura de SPADE y las conexiones a AirSim, se detalla en la Figura 3.3.

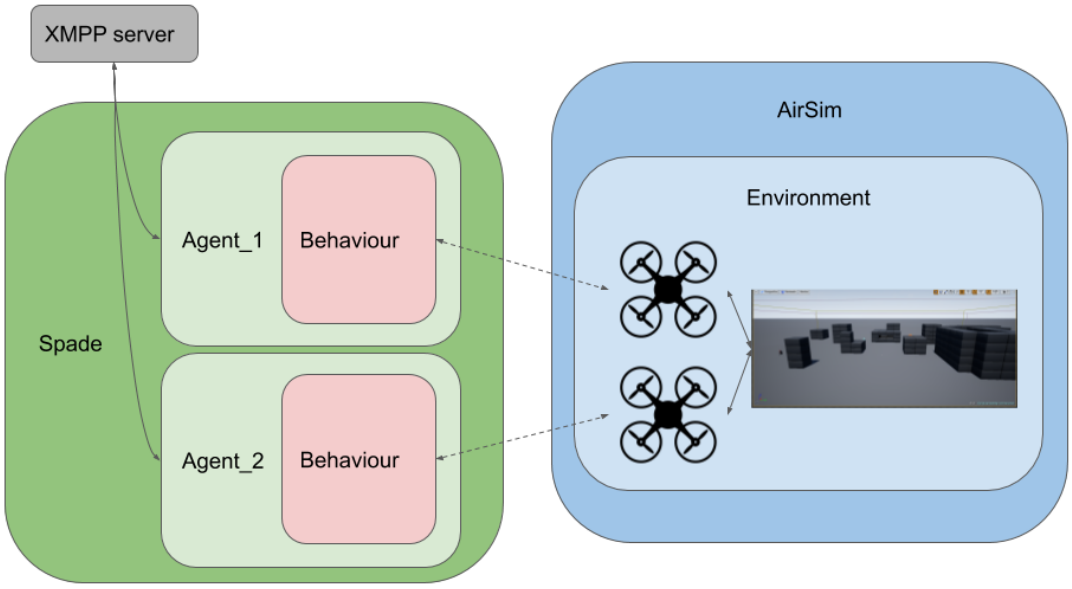



**Figura 3.3:** Conexión entre SPADE y AirSim

Como se puede observar, en la Figura 3.3 contemplamos dos estructuras distintas, SPADE y AirSim, además existe un servidor de mensajería instantánea XMPP.

Comenzamos detallando el funcionamiento de Spade. SPADE está implementado para albergar uno o más agentes, estos agentes utilizan un protocolo XMPP de mensajería instantánea que les permite comunicarse entre sí. Pueden enviar o recibir todo tipo de mensajes como se ha mencionado anteriormente. En este proyecto las comunicaciones entre agentes se utilizan para establecer un seguimiento mediante coordenadas que se detalla en el siguiente capítulo.

Cada agente puede contener uno o más *behaviours*(comportamientos). Estos comportamientos controlan las acciones que realiza el agente y cómo reacciona ante situaciones. Desde el *behaviur* se ha decidido implementar la conexión al drone de AirSim. Esto permite que el drone forme parte del comportamiento del agente. De este modo, el agente puede reaccionar ante estados del drone y el dron actuar con acciones definidas por el comportamiento del agente.

Trasladándose al simulador AirSim, en concreto a los drones. Un drone es la herramienta por la que actúa el Agente, como se detalla en Subsección 3.5.3, todas las funciones que permiten interactuar con el drone del simulador. Los drones interactúan con el entorno, en este caso en la Figura 3.3 se puede observar una imagen del entorno de bloques donde se han ejecutado ambas soluciones del proyecto que se detallan más adelante en el Capítulo 4.

Todo este conjunto de conexiones permite operar de forma que los dos sistemas, el simulador de drones y el sistema de agentes inteligentes interactúen entre sí de forma coordinada.

## 3.5 Estructura de la solución

En esta sección se tratará toda la estructura de la herramienta desarrollada, en primer lugar los archivos *main* desde donde se ejecutan las soluciones detalladas en la Sección 4.1 y la Sección 4.2. Seguidamente se detalla la estructura que se ha desarrollado para generar los agentes. Se ha realizado una unión entre los agentes de SPADE y la API de AirSim. Por último, se detallan todas las funcionalidades que puede realizar un drone.

### 3.5.1. Simulador

En primer lugar, la clase en Python *simulador* se utiliza para crear los agentes de cada solución e iniciarlos. Este es el momento donde todos los agentes obtienen la configuración que se les ha predefinido para el funcionamiento de la solución.

En el proyecto existen dos archivos de inicio denominados *SimuladorFollowMe.py* y *SimuladorDQN.py*, cada uno de ellos correspondiente a una solución que se detalla más adelante en la Sección 4.1 y la Sección 4.2 respectivamente.

```
1  import time
2  from Agentes.AgenteSpadeDrone import PeriodicSenderAgent, ReceiverAgent
       , Config
3
4  if __name__ == "__main__":
5      jidMaster = "senderMasterAgent15@gtirouter.dsic.upv.es"
6      jidSlave = "reciberSlaveAgent15@gtirouter.dsic.upv.es"
7
8      MasterConfig = Config(jidMaster,"123","Drone_1","MyLidar1_1","
           MyLidar1_2","GPS_1",jidSlave,vel=5)
9      SlaveConfig = Config(jidMaster,"123","Drone_2","MyLidar2_1","
           MyLidar2_2","GPS_2",jidMaster,vel=5)
10
11     master = PeriodicSenderAgent(MasterConfig)
12     slave = ReceiverAgent(SlaveConfig)
13     slave.start()
14     master.start()
15     while master.is_alive():
16         try:
17             time.sleep(1)
18         except KeyboardInterrupt:
19             slave.stop()
20             master.stop()
21             break
22
23     print("Agents finished")
```

**Listing 3.3:** SimuladorFollowMe.py

Como se puede observar en el Listing 3.3 se crean las configuraciones para cada drone y se les aplica en la creación de los agentes. Seguidamente se inician y mientras los agentes estén activos el proceso no finaliza. En el siguiente punto, Subsección 3.5.2 se detalla la configuración y los tipos de agentes.

### 3.5.2. Agentes SPADE con AirSim

Formando parte del músculo del proyecto, donde se encuentra la mayor parte del código desarrollado son los agentes inteligentes del proyecto. En esta sección se detalla la forma que se ha desarrollado la unión entre los agentes de SPADE y el sistema de simulación AirSim.

Esta sección hace referencia al código desarrollado en Python sobre la clase *AgenteSpadeDrone.py*, aquí se detalla en primer lugar, la forma con la cual se configuran todos los agentes en el proyecto. Mezclando configuraciones de SPADE y AirSim.

```
class Config():
    def __init__(self,jid,password,name,lidar1,lidar2='',gps='',
        jidSlave="",vel=1,mov=1,verbose=False,num_episodes=0):
        self.jid = jid
        self.password = password
        self.jidSlave = jidSlave
        self.name = name
        self.lidar1 = lidar1
        self.lidar2 = lidar2
        self.gps = gps
        self.vel = vel
        self.mov = mov
        self.verbose = verbose
        self.num_episodes = num_episodes
```

**Listing 3.4:** Configuracion.py

Como se observa en el Listing 3.4, las configuraciones de *jid*, *password* y *jidSlave* forman parte de las comunicaciones de SPADE. Por otra parte, *name*, *lidar1*, *lidar2*, *gps*, *vel* y *mov* forman parte de las configuraciones para la conexión de AirSim. Por último, *verbose* y *num_episodes* forman parte del actual proyecto.

Por otra parte y haciendo referencia al nombre de la actual sección, esta clase en python contiene a los agentes inteligentes del proyecto. Un agente inteligente tiene una estructura como se detalla en la siguiente Listing 3.5

```
class PeriodicSenderAgent(Agent):
    def __init__(self, config):
        Agent.__init__(self, jid=config.jid, password=config.password,
            verify_security=False)
        self.config = config

    async def setup(self):
        print(f"PeriodicSenderAgent started at {datetime.datetime.now()
            .time()}")
        start_at = datetime.datetime.now() + datetime.timedelta(seconds
            =5)
        b = self.InformBehav(period=2, start_at=start_at, config=self.
            config)
        template = Template()
        template.set_metadata("performative", "inform")
        self.add_behaviour(b)

    class InformBehav(PeriodicBehaviour):
        def __init__(self, period, start_at, config):
```

```
16            PeriodicBehaviour.__init__(self, period=period, start_at=
                  start_at)
17            self.config = config
18
19        async def on_start(self):
20            self.drone = Master(self.config)
21            self.drone.takeoff()
22            initial_possition = self.drone.getPosition()
23            print("Initial Possition: " + str(initial_possition) + "
                  GPS: ")  # + str(self.drone.getGps()))
24            self.counter = 0
25
26        async def run(self):
27            print(f"PeriodicSenderBehaviour running at {datetime.
                  datetime.now().time()}: {self.counter}")
28            msg = Message(to=self.config.jid)  # Instantiate the
                  message
29
30            drone_pos = self.drone.getPosition()
31            msg.body = str(int(drone_pos.x_val)) + ';' + str(int(
                  drone_pos.y_val)) + ';' + str(int(drone_pos.z_val))
32
33            await self.send(msg)
34            print("Message sent!")
35            ...
36            ...
37
38        async def on_end(self):
39            # stop agent from behaviour
40            await self.agent.stop()
```

**Listing 3.5:** Estructura de un agente inteligente

Como se observa en el Listing 3.5 este agente pertenece al dron *Master* de la solución descrita en la siguiente Sección 4.1. Pero vamos a quedarnos con la estructura para cualquier agente. Empezando por la herencia, esta clase hereda de un Agente SPADE por ello es necesario configurarlo con *jid* y *password*.

Seguidamente, la función *setup* configura el agente con el comportamiento que va a adoptar el agente. El comportamiento en este caso, se trata de un comportamiento periódico, pero existen varios comportamientos ya implementados como se ha comentado en la Subsección 2.4.1.

Para definir un comportamiento hay que crear una clase dentro del agente que herede del tipo de comportamiento seleccionado. Los comportamientos tienen tres estados básicos, *on_start*, *run* y *on_end*.

- *On_start*: Se trata de las acciones que se van a llevar a cabo antes de realizar las acciones que contiene *run* y solo se ejecuta una vez en el momento de la inicialización del agente.
- *Run*: Se trata de las acciones que se realizan de forma repetitiva, empiezan tras finalizar *On_start*.
- *On_end*: Se trata de las acciones que se realizan tras finalizar el bucle de repetición *run*.

En el proyecto actual, se realiza la conexión al simulador AirSim desde *On_start*. Esto permite que nada más inicializar el agente, se le asigne al dron seleccionado en la configuración inicial. En el Listing 3.5 se puede observar cómo se crea la clase *Master* que contiene los métodos para realizar las comunicaciones y acciones con el drone de AirSim.

Desde la función de *run* se realiza el grueso del comportamiento y las acciones que interactúan con el drone de AirSim, para ello se utiliza la variable drone que permite el acceso a todas las funciones que se detallan en la Subsección 3.5.3. La función *On_end* se encarga de finalizar el agente.

El el proyecto se han desarrollado tres tipos de agentes inteligentes distintos que interactúan con los drones de Airsim, pero está preparado para poder albergar todos los tipos de agentes que completen las funcionalidades requeridas para cada proyecto.

### 3.5.3. Drone

Respecto al apartado el cual se realizan todas las acciones que pueden ejecutarse sobre el drone en base a los movimientos, acciones, obtención de parámetros de los sensores, etc. Todas estas funciones se encuentran en la clase *Drone.py*. La cual es llamada desde la clase *AgenteSpadeDrone.py* mediante la variable *drone* como se detalla en la Subsección 3.5.2.

A continuación se detallan las funciones que contiene la clase Drone y para que se utilizan.

| Función | Input | Output | Descripción |
|---|---|---|---|
| iniciar_drone | Configuración | None | La función de iniciar drone se utiliza para crear la conexión al entorno de simulación por parte del drone con todas las configuraciones necesarias |
| reset_env | None | None | Se utiliza para reiniciar la simulación. De este modo el drone vuelve a la posición inicial de forma que el drone se encuentra en tierra. |
| takeoff | None | None | Si el drone se encuentra en tierra, este despega hasta alcanzar una altura considerable para proporcionar un estado de vuelo. En el caso de que se encuentre en modo vuelo, el drone no hace nada y se mantiene en el aire. |

**Tabla 3.2:** Funciones Clase Drone I

| Función | Input | Output | Descripción |
|---|---|---|---|
| landing | None | None | En el caso de que el drone se encuentre volando aterriza de inmediato, descendiendo lentamente sobre su vertical. |
| parse_lidarData | String | list | Realiza una transformación de la información obtenida por el Lidar en texto plano a un array de puntos con 3 dimensiones por punto. |
| getLidar | None | Image | Genera una imagen en la cual se representa los puntos obtenidos de la información del Lidar. La imagen generada es en escala de grises y tiene un tamaño de 500x500 px. |
| getCollision | None | Collision-Info | Devuelve la información sobre si el drone ha colisionado contra alguna superficie. En el caso de que colisione devuelve True, de lo contrario devuelve False. |
| getQuadState | None | Vector3r | Obtiene la posición del drone en el instante actual y lo devuelve como estructura de datos Vector3r. |
| getQuadVel | None | Vector3r | Obtiene la velocidad del drone en el instante actual y lo devuelve como estructura de datos Vector3r. |
| moveDerecha | Int, Int | None | Utiliza la función moveTo para realizar un movimiento hacia la derecha. Necesita los parámetros de entrada y: tamaño del desplazamiento y vel: velocidad del desplazamiento. |
| moveIzquierda | Int, Int | None | Utiliza la función moveTo para realizar un movimiento hacia la izquierda. Necesita los parámetros de entrada y: tamaño del desplazamiento y vel: velocidad del desplazamiento. |
| moveDelante | Int, Int | None | Utiliza la función moveTo para realizar un movimiento hacia delante. Necesita los parámetros de entrada x: tamaño del desplazamiento y vel: velocidad del desplazamiento. |
| moveAtras | Int, Int | None | Utiliza la función moveTo para realizar un movimiento hacia atrás. Necesita los parámetros de entrada x: tamaño del desplazamiento y vel: velocidad del desplazamiento. |

**Tabla 3.4:** Funciones Clase Drone II

| Función | Input | Output | Descripción |
|---|---|---|---|
| setInfo | String, String, String, String | None | La función se utiliza junto la función de iniciar drone para establecer los parámetros de nombre, Lidar y GPS del drone y además realiza acciones de habilitar la API y armar el drone sobre el entorno de simulación. |
| moveArriba | Int, Int | None | Se utiliza para elevar el dron. En caso de estar volando por debajo de Z=50 se eleva incrementando la altura del drone el valor de z en caso que supere Z=50 se eleva hasta esta posición. |
| moveAbajo | Int, Int | None | Se utiliza para descender el dron. En el caso que la altura sea menor que Z=0 se desciende hasta mantener una altura de Z=1. |
| getPosition | None | Vector3r | Devuelve la posición actual del drone en coordenadas X,Y,Z. |
| moveTo | None | None | La función realiza una acción que repercute en un movimiento del drone en línea recta desde donde se encuentra el drone hasta las coordenadas indicadas como entrada en forma X,Y,Z a la velocidad de entrada V. |
| takeoffPosition-Start | Int | None | El uso de esta función va asociado a resetear el entorno con la función reset_env, realiza acciones de despegue desde el suelo y se eleva en Z + 5. |

**Tabla 3.6:** Funciones Clase Drone III

### 3.5.4. Conclusiones

Para finalizar el bloque del desarrollo, se concluye que el desarrollo de las distintas partes del proyecto ha sido costoso en cuanto a diseño y desarrollo, en especial la estructura adoptada para implementar la unión del simulador Airsim junto con el sistema de agentes inteligentes SPADE.

Este bloque ha detallado el funcionamiento del código del proyecto junto con todas las clases utilizadas y las herencias, de forma adecuada. Esto permite la posibilidad de realizar nuevos desarrollos por parte de otros equipos de trabajo. Se detalla qué estructura tiene que tener un agente para la correcta sincronización con los drones de AirSim y las propiedades que se pueden acceder para el control de un drone en AirSim.

Por otra parte se ha detallado cómo crear un entorno para reproducir las soluciones propuestas, esto permite poder utilizar esta memoria como fuente de documentación para desarrollar otros proyectos en el simulador AirSim haciendo uso de sistemas multiagente.

El siguiente capítulo se detalla las soluciones que se han implementado sobre modelos de coordinación entre agentes y modelos de aprendizaje por refuerzo aplicados a la estructura de la herramienta desarrollada.

# CAPÍTULO 4
# Caso de estudio

En el cuarto bloque, se detallan las dos soluciones planteadas en este proyecto. Estas soluciones son el resultado de alcanzar los objetivos que tienen como meta realizar una comprobación de la herramienta que incluye el sistema de agentes con :

- Desarrollar un caso de coordinación de agentes donde existe un agente/-drone que toma el rol *Maestro* e indica al resto de agentes/drones que toman el rol *Esclavo*, indicándose donde tiene que desplazarse a modo de seguimiento "FollowMe".
- Implementar un modelo de aprendizaje con un algoritmo de *Reinforcement Learning* con DQN para desarrollar un sistema de conducción autónoma en drones basada en aprendizaje por refuerzo.

## 4.1 FollowMe

La solución propuesta es la siguiente, en el simulador se configuran para que inicie con dos vehículos de tipo drone. Esta configuración se realiza como se explica en el apartado Subsección 2.3.1. La clave de esta configuración se encuentra en el Listing 4.1, cabe destacar que se han denominado todos los componentes de los vehículos con con nombres identificativos. De este modo desde la herramienta desarrollada, cada agente se conecta a su drone correspondiente y pasan a ser una unión.

```
"SimMode": "Multirotor",
  "Vehicles": {
    "Drone_1": {
      "VehicleType": "SimpleFlight",
      "Sensors": {
        "MyLidar1_1": {
          ...
        },
        "GPS_1": {
          "SensorType": 3,
          "Enabled" : true
        }
      },
      "X": 4.0,
```

```
      "Y": 0.0,
      "Z": -2,
      "Pitch": 0,
      "Roll": 0,
      "Yaw": 0
    },
    "Drone_2": {
      "VehicleType": "SimpleFlight",
      "Sensors": {
        "MyLidar2_1": {
          ...
        },
        "GPS_2": {
          "SensorType": 3,
          "Enabled" : true
        }
      },
      "X": 4.0,
      "Y": 4.0,
      "Z": -2,
      "Pitch": 0,
      "Roll": 0,
      "Yaw": 0
    },
```

**Listing 4.1:** Configuración AirSim modo FollowMe

La unión de agente y drone se detalla a continuación en la Figura 4.1 y la Figura 4.2 para el *Master* y el *Slave* respectivamente.

Como se ha detallado anteriormente en la Subsección 3.5.3 la clase *Drone.py* conecta directamente con el drone de AirSim que se indica en la configuración de cada agente pasando como nombre el drone al que se quiere conectar el agente y se ha configurado en el archivo de configuración de AirSim. De este modo cada agente ya tiene el control de su drone correspondiente.

A continuación se detalla como esta configurado cada agente y la función que tiene en el entorno.

### 4.1.1. Master

En primer lugar, el agente **Master** es el que interactúa con el drone denominado "Drone_1.[en] la configuración de AirSim. Este agente, como su nombre indica es el *Maestro* y tiene la función de dirigir al resto de agentes.

Esta jerarquía viene dada por la necesidad de transmitir una orden al resto de agentes. La orden indica que el resto de agentes tienen que moverse a la posición actual del agente *Maestro*. Para realiza la transmisión de la orden al resto de agentes es necesario comprender la estructura de la Figura 4.1.

Como se observa en la figura, existen dos apartados AirSim y SPADE. Estos apartados son los que tienen interacciones con cada una de las partes del software.

Empezando por la clase *Master* que hereda de la clase *Drone*. Esta clase ya se han mencionado anteriormente todas sus propiedades. Por otra parte, la clase

*SenderMasterAgent* hereda de la clase Agent del sistema SPADE. De este modo se implementa un agente nuevo en SPADE. El agente nuevo, como todos los agentes, tiene que tener un comportamiento. Los comportamientos en SPADE se definen creando una clase anidada interiormente ya que pertenece al agente, que hereda de la clase del comportamiento seleccionado. En este caso se trata del comportamiento *Periodicbehaviour*, este comportamiento realiza sus acciones en bucle con una periodicidad.

La acción que se le ha asignado al agente *Maestro* es indicar sus coordenadas al resto y se envían con el formato:

$$(coordenadaX; coordenadaY; coordenadaZ)$$

.

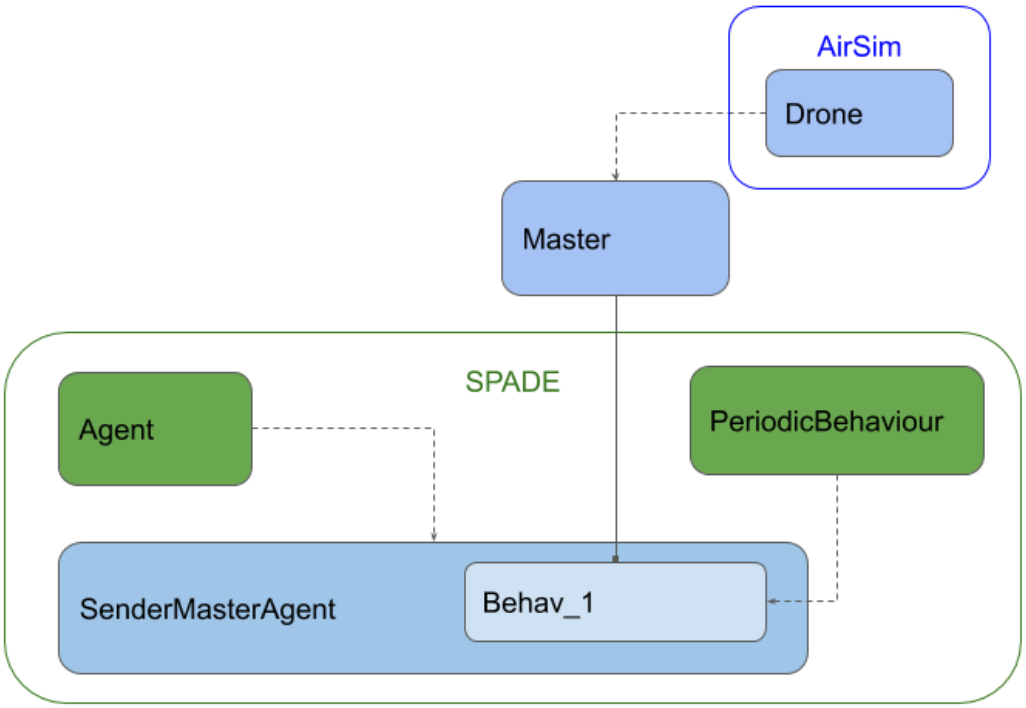



**Figura 4.1:** Esquema Master

Adicionalmente para realizar el experimento, se ha configurado un proceso para que el drone "Drone_1"se mueva por el escenario. Estas acciones se detallan en la Subsección 4.1.3

### 4.1.2. Slave

El agente **Slave**, como su nombre indica *Esclavo*, realiza todas las acciones que le indica el agente *Master*. Para recibir órdenes desde el agente *Master* es necesario implementar la estructura indicada en la Figura 4.2.

Del mismo modo que en el caso del agente *Master*, se distinguen las dos partes de la estructura de la herramienta, el simulador AirSim y el sistema SPADE. Al igual que en agente anterior, la clase *Slave* hereda de la clase Drone para tener todas las acciones sobre el drone en el simulador de AirSim.

Por otra parte, en el sistema multiagente SPADE. El agente creado, en este caso se trata de *ReciberSlaveAgent* que también hereda de Agente de SPADE. Del mismo modo se implementa los comportamientos, pero en este caso hereda de otro tipo de comportamiento. El comportamiento utilizado para el dron *Slave*, es un comportamiento cíclico *CyclicBehaviour*. La característica principal de este comportamiento es que nunca termina hasta que se mata el agente y mientras repite siempre el mismo ciclo(*run*).

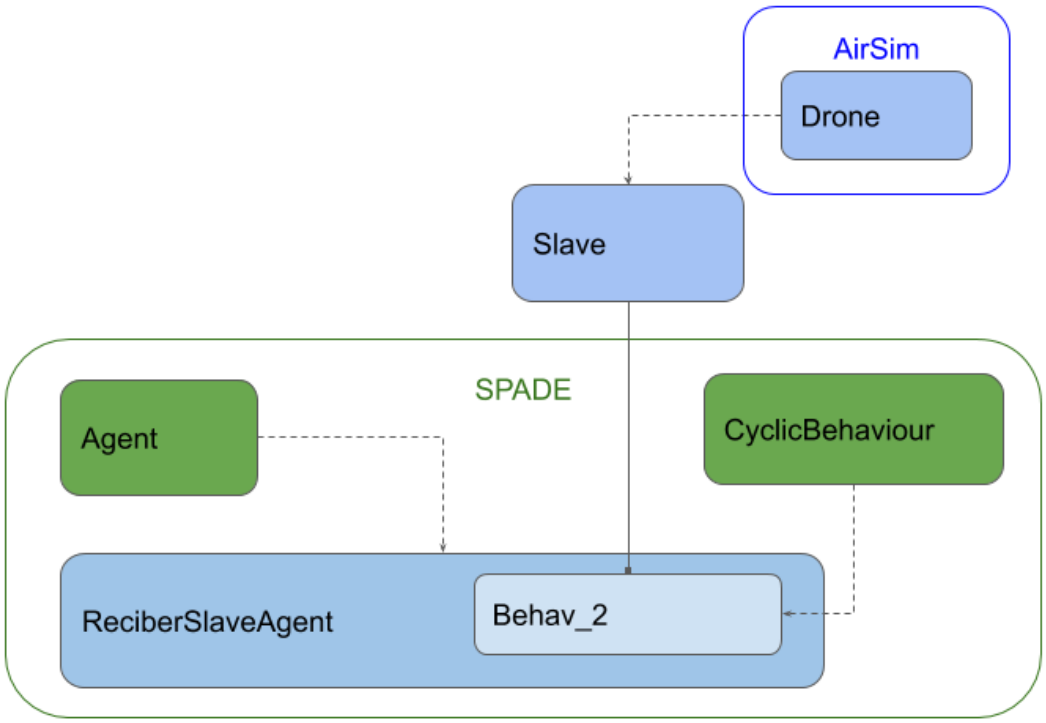



**Figura 4.2:** Esquema Slave

El modo en que recibe los mensajes y se realiza la acción, está implementado como se observa en el Listing 4.2. En primer lugar se espera a recibir el mensaje con un tiempo máximo de espera de 30 seg. En caso de no recibir mensaje se termina el agente. De lo contrario, obtiene las coordenadas separadas por ';' y las ejecuta en modo imperativo: "Muévete a la posición X,Y,Z".

```
msg = await self.receive(timeout=30)
if msg:
    coordenadas = [float(i) for i in str(msg.body).split(';')]
    self.drone.moveTo(coordenadas[0], coordenadas[1], coordenadas[2],
        self.config.vel)
else:
    print("Did not received any message after 30 seconds")
    self.kill()
```

**Listing 4.2:** Recepción de msg y realización de la acción.

### 4.1.3. Experimentos

El experimento que se ha realizado consta de realizar comunicaciones indicando las coordenadas a las que acudir el agente *Slave* informadas por el agente *Master*.

Para que los movimientos tenga alguna base desde donde tomar estas posiciones, se le proporciona al agente *Master* una lista con siete tripletas de coordenadas para ir alcanzando una tras otra en cada iteración(*run*) del comportamiento. Con anterioridad a la realización del movimiento para alcanzar la coordenada siguiente, se realiza la acción de enviar las coordenadas actuales al agente *Slave*.

El agente *Slave* recibe las coordenadas y de inmediato realiza la acción de moverse hasta estas coordenadas. Existen diferencias entre la orden de la acción hasta la ejecución de la orden.

En la Figura 4.3 de observa la diferencia de distancia entre drones, esto explica que cuanto mayor es la distancia mayor tiempo tarda en responder a la acción ordenada.

En el conjunto de Figuras 4.4 se observa como el dron secundario realiza el recorrido que le indica el dron principal o *Maestro*. En la Figura 4.4a se observa como el drone *Maestro* inicia el recorrido despegando del suelo, mientras tanto el dron secundario espera que que el drone *Maestro* le indique la orden a realizar. En la Figura 4.4b se muestra que el drone *Maestro* ya ha alcanzado uno de los puntos de la ruta, y indica al dron secundario que se mueva al punto indicado. Del mismo modo se puede observar en las Figuras 4.4c y 4.4d que se realizan movimientos de seguimiento por parte del drone secundario al drone *Maestro* indicado por el drone *Maestro*.

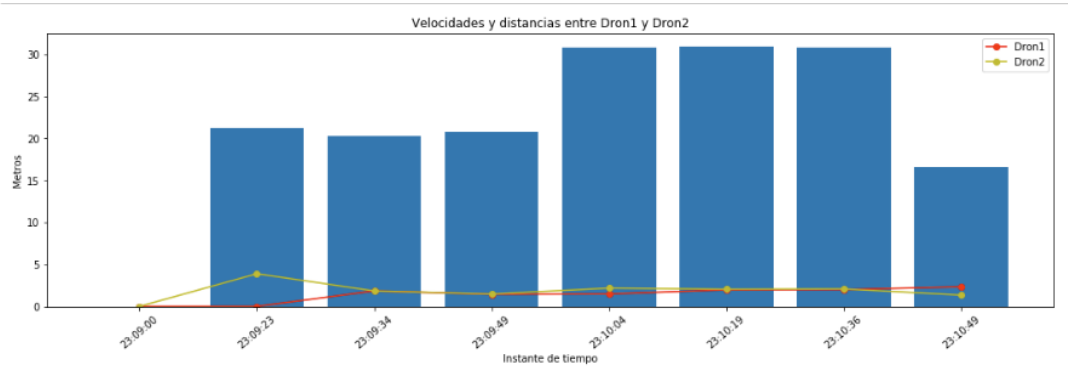
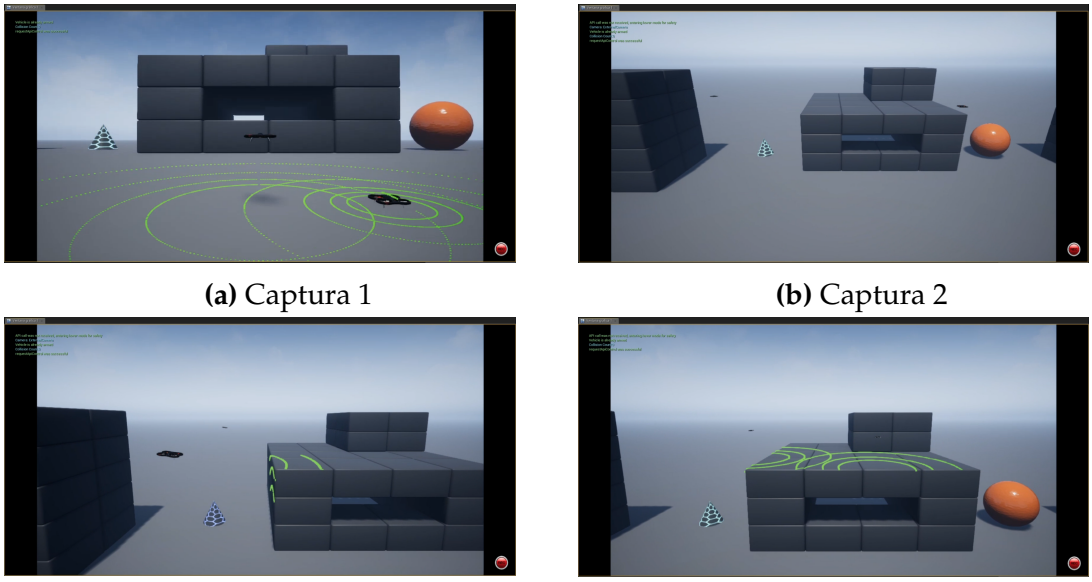



**Figura 4.3:** Velocidad y distancia entre Drone_1 y Drone_2



**(a)** Captura 1 **(b)** Captura 2

**(c)** Captura 3 **(d)** Captura 4

**Figura 4.4:** Ejecución del proceso "FollowMe"

## 4.2 Reinforcement Learning con DQN

En la Secciónactual, se detalla la estructura de la solución al objetivo que trata sobre implementar un algoritmo de *Reinforcement Learning* con *DQN*. Este objetivo se basa en la obtención de un algoritmo que permita pilotar un drone de forma autónoma.

En primer lugar se ha creado un tipo de agente que contiene el acceso a un drone con una serie de propiedades adicionales. El drone DroneDQN contiene además de todas las funcionalidades de un drone normal, todas las funciones necesarias para implementar el algoritmo de Deep Q-leaning con redes neuronales. A continuación en la Subsección 4.2.1 se detalla con exactitud los métodos utilizados.

Un algoritmo de aprendizaje por refuerzo, trata de aprender que mejor acción tomar en cada estado. Si el conjunto de estados es finito y no muy grande, se puede explorar todos los estados evaluar el resultado que se obtiene al realizar cada acción en cada estado y así obtener el camino más óptimo. Pero esto en un entorno de simulación no determinista y sin acotar o con una acotación muy amplia como es AirSim, no se puede realizar.

en referencia a este problema, se utiliza el siguiente Algoritmo 4.1. El algoritmo de Deep Q-learning, utiliza una memoria para almacenar las experiencias pasadas. Esta memoria de experiencias se utiliza para entrenar la red neuronal que tiene que predecir la siguiente acción.

El algoritmo funciona del siguiente modo:

Inicializa una memoria $M$ con el tamaño de estados, en este caso el número de estados es gigantesco, por ello se utiliza un número bastante grande. En nuestro caso se ha optado por 10000. Esta memoria contiene las experiencias que han ocurrido. Cada experiencia se almacena con una tupla de estado actual $S$, acción $A$,

recompensa $R$, estado siguiente $S'$. También se inicializan dos redes neuronales $Q$ Policy_net y $Q'$ Target_net. Estas dos redes son idénticas, tienen los mismos parámetros.

El algoritmo está compuesto por dos bucles. El primer bucle se relanza una nueva iteración cada vez que colisiona o alcanza el objetivo. Por ello cada vez que inicia una iteración en el primer bucle se Inicia el entorno y se obtiene el primer estado o estado inicial.

A continuación se inicia el segundo bucle seleccionando la acción a realizar. Existen dos tipos de seleccionar la acción a tomar a cabo. En primer lugar la *exploración*, esto consiste en realizar acciones al azar de forma que se va explorando los nuevos estados que se obtienen al realizar estas acciones al azar. La otra forma de seleccionar una acción es la *explotación* que consiste en utilizar la experiencia obtenida de casos anteriores realizar las acciones que mejor recompensa se obtenga a priori.

Este paradigma de exploración o explotación se gestiona mediante la probabilidad de $\varepsilon$. El valor de $\varepsilon$ va cambiando dependiendo del estado de la ejecución, cuánto tiempo lleva recopilando experiencias. Al principio cuando no existe ninguna experiencia siempre tiene que explorar, de este modo el valor de $\varepsilon$ tiene que ser favorable en todos los casos para la exploración. A medida que el algoritmo va avanzando el valor de la variable $\varepsilon$ va modificándose para hacer posible una explotación de los conocimientos adquiridos en experiencias anteriores.

Este modo de explotación es utiliza la red neuronal $Q$ policy_net, como entrada el estado actual $S$ y como salida la acción $A$ que maximice el valor $Q$ para el estado $S$.

Una vez aplicada la acción $A$ sobre el estado $S$ se obtiene una recompensa $R$ y un nuevo estado $S'$. Estos valores se almacenan en la memoria $M$ a modo de experiencias en forma de tupla $(S, A, R, S')$. De todas las experiencias anteriores, se selecciona aleatoriamente un bach de experiencias. Con estas experiencias se estima los targets $y$ desde la red neuronal $Q'$ target_net, en caso de colisión o estado terminal el target es $-1$.

Seguidamente se optimiza la red utilizando los parámetros extraídos, $(y - Q(S, A; \theta))^2$. De este modo se actualizan los pesos de la red neuronal policy_net para ir mejorando sus resultados.

En ocasiones, tras ciertos pasos la red neuronal target_net adquiere los pesos de la red neuronal policy_net, esto permite que que la optimización del algoritmo se realice correctamente. Si por el contrario no se realizará, y se utilizará la misma red. Los resultados de la red se anularían entre sí y no se optimizaría el modelo.

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max_{A'} Q(S', A') - Q(S, A))$$

Como se observa en la ecuación existen dos valores $Q$, el valor $Q(S, A)$ pertenece a la red neuronal *policy_net* y el valor $\gamma \max_{A'} Q(S', A')$ pertenece a la red neuronal *target_net*. En el hipotético caso de que todo fuera la misma $Q$ quedarían anuladas.

Por último el estado siguiente $S'$ pasa a ser el estado actual $S$.

**Algorithm 4.1** Deep Q-learning

0: **Inicializar:**
Memoria $M$ tamaño $S$
Red neuronal $Q$ con parámetros $\theta$ (Policy_net)
Red neuronal $Q'$ con parámetros $\theta'$ (Target_net)
$\theta' = \theta$
**for** episodios $1 : N$ **do**
Inicializar Entorno y preprocesar el estado $S$
**for** $1 : T$ **do**

$$A \begin{cases} accin\ random & con\ probabilidad\ \varepsilon \\ \underset{a}{argmax}\, Q(S, A; \theta) & otro\ caso \end{cases}$$

Tomar la acción $A$ y observar reward $R$, siguiente estado $S'$ y preprocesar $S'$
Almacenar la experiencia $(S, A, R, S')$ en la memoria $M$

Seleccionar minibach random $(S, A, R, S')$ desde la memoria $M$
Estimar targets $y$ desde las experiencias

$$y \begin{cases} -1 & if\ muere\ o\ termina \\ R + \underset{a}{max}\, Q'(S', A; \theta') & otro\ caso \end{cases}$$

Optimización del algoritmo $(y - Q(S, A; \theta))^2$
Tras $C$ steps $Q' = Q$
$S = S'$
**end for**
**end for**
=0

En la siguiente Subsección 4.2.1 se explica cómo este algoritmo se ha trasladado a la estructura del agente.

### 4.2.1. DroneDQN

En la sección actual se define como se ha representado las partes del algoritmo en el proyecto.

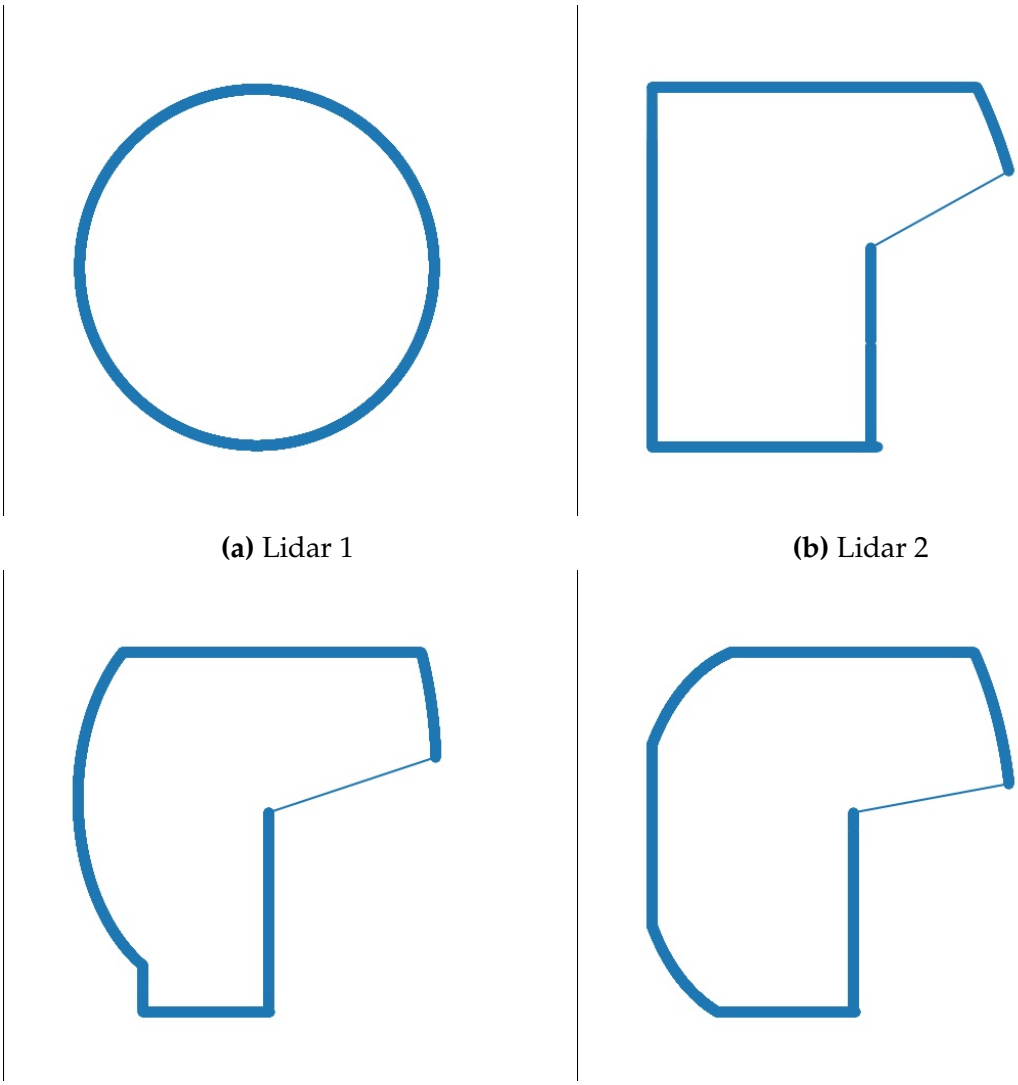
#### Estados

En primer lugar, la representación de los estado viene dada por el **Lidar**. El lidar es un sensor mencionado en la Subsección 2.3.1. Este sensor reconoce los objetos cercanos y obtiene un vector de puntos sobre los cuales el haz de luz ha colisionado con estos objetos cercanos.

Este vector de puntos se transforma a una imagen, el resultado se muestra en las figuras 4.5. En la primera figura, el sensor no detecta ningún objeto sobre el cual colisionar el haz de luz. Por ello devuelve una circunferencia, la cual representa el suelo. En el segundo ejemplo, el sensor detecta objetos a los laterales y delante y detrás, pero hay que destacar que el objeto que se encuentra a la derecha

es más próximo que el de la izquierda. En el tercer caso, es idéntico al anterior pero se observa una curvatura a la izquierda, esto puede indicar que no hay objetos en esta dirección y el sensor capta la superficie del suelo.

De este modo es por el cual se representan los estados en el algoritmo implementado.



**(a)** Lidar 1 **(b)** Lidar 2

**(c)** Lidar 3 **(d)** Lidar 4

**Figura 4.5:** Ejemplo de capturas de lidar que representan estados

### Acciones

Las acciones se representan en actos que realizan un cambio en el entorno. En este proyecto se ha optado por acotar el número de acciones a realizar. Las acciones son:

- Avanzar
- Retroceder
- Moverse a la izquierda

- Moverse a la derecha

En la herramienta desarrollada existen las opciones de elevarse y descender, pero en el algoritmo de aprendizaje por refuerzo se ha decidido acotar los grados de libertad para reducir la dificultad del problema.

En cuanto a la selección de las acciones, tal y como se ha mencionado anteriormente, existen dos formas de hacerlo por explotación y por exploración.

La función que se ha implementado para realizar este proceso, es la detallada en el Listing 4.3. Esta función utiliza se una variable $\varepsilon$, esta variable tiene la función de distribuir las propiedades dependiendo del paso en el que se encuentra. Existen muchas investigaciones sobre cómo realizar esta distribución de probabilidades de forma adecuada. En esta solución se ha adoptado la fórmula siguiente para determinar el valor de $\varepsilon$.

$$\varepsilon = EPS_{END} + (EPS_{START} - EPS_{END}) * e^{(-1 * \frac{STEP}{EPS_{DECAY}})}$$

En cuanto a la *Explotación* como se puede observar, devuelve el resultado de la red neuronal *policy_net* en este caso se trata de la acción a realizar por parte del agente. Por otra parte, la *Exploración*, revuelve una acción random. De este modo se obtiene la acción a ejecutar, ya sea de un modo u otro.

```
def select_action(self, state):
        global steps_done
        sample = random.random()
        eps_threshold = self.EPS_END + (self.EPS_START - self.EPS_END)
            * math.exp(-1. * steps_done / self.EPS_DECAY)
        steps_done += 1
        if steps_done > self.BATCH_SIZE and sample > eps_threshold:
            print("Explotaci n")
            with torch.no_grad():
                state1 = state.clone().detach().requires_grad_(True)
                return self.policy_net(state1).max(1)[1].view(1, 1)
        else:
            print("Exploraci n")
            return torch.tensor([[random.randrange(self.n_actions)]],
                device=self.device, dtype=torch.long)
```

**Listing 4.3:** Función de selección de la acción.

### Reward

El *reward* es la función que define el objetivo del aprendizaje por refuerzo. en este proyecto se ha indicado como objetivo sortear los objetos y paredes que ha en una calle y llegar hasta el final de la calle.

Por ello, cuanto más lejos se encuentre del final de la calle menor *reward* obtendrá. De lo contrario, cuanto más se acerque al final de la calle el *reward* será mayor. En caso de que el dron colisione, el agente obtendrá un valor de *reward* igual a $-1000$ siendo este el peor resultado que se puede obtener. De este modo la acción que consigue que en un estado determinado colisione contra un objeto, se valora de forma muy negativa.

En este caso el *reward* máximo, se obtiene posándose sobre las coordenadas del objetivo esto devuelve un *reword* de 1000, a medida que se aleja del objetivo el *reword* va decrementando.

```
if self.collision_info.has_collided:
    reward = -1000
else:
    x_objetivo = self.goal_x
    y_objetivo = self.goal_y
    maxreword = 1000
    distance = self.calculateDistance(self.quad_state.x_val, self.
        quad_state.y_val, x_objetivo, y_objetivo)

    if distance > maxreword:
        reward = 0
    else:
        reward = maxreword - distance
```

**Listing 4.4:** Función de reward de la acción en el estado actual.

### Memoria

La memoria $M$ en el proyecto actual se ha diseñado utilizando la implementación mostrada en el Listing 4.6. Se trata de una estructura formada por una lista que inserta experiencias registradas en tuplas. Se han implementado una serie de funciones que permite insertar las experiencias para ir alimentándose y otra función para obtener un bach de muestras con el tamaño indicado como parámetro.

Las tuplas están formadas por el estado, la acción, el siguiente estado y el reward como se indica en el Listing 4.5.

```
Transition = namedtuple('Transition', ('state', 'action', '
    next_state', 'reward'))
```

**Listing 4.5:** Experiencias en formato tupla.

```
class ReplayMemory(object):

    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []
        self.position = 0

    def push(self, *args):
        if len(self.memory) < self.capacity:
            self.memory.append(None)
        self.memory[self.position] = Transition(*args)
        self.position = (self.position + 1) % self.capacity

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

**Listing 4.6:** Replay memory.

**Red**

La red neuronal que se ha decidido implementar es simple, no contiene una gran cantidad de capas. En concreto las características son las siguientes:

- Convolución In:3 Out:16 Kernel:7 stride:2
- BatchNorm
- Convolución In:16 Out:32 Kernel:7 stride:2
- BatchNorm
- Convolución In:32 Out:32 Kernel:7 stride:2
- BatchNorm

Esta red tiene tres capas convoluciones con Batch Norm y función de activación Relu a la salida de cada capa. La salida de la res un función lineal.

La función de pérdida se ha implementado como se indica en el Listing 4.7. Tal que en formulación matemática sería:

$$loss(x, y) = \frac{1}{n}\sum_{i} z_i$$

tal que $z_i$ está representada por:

$$z_i = \begin{cases} 0{,}5(x_i - y_i)^2, & if|x_i - y_i| < 1 \\ |x_i - y_i - 0{,}5, & otros\ casos \end{cases}$$

```
1   loss = F.smooth_l1_loss(state_action_values,
        expected_state_action_values.unsqueeze(1))
```

**Listing 4.7:** Función de perdida.

### 4.2.2. Experimentos

Una vez implementado el algoritmo en la herramienta desarrollada, se han realizado varios experimentos. La finalidad de estos experimentos es obtener la certeza de que el algoritmo funciona correctamente, de forma que tras varias iteraciones del algoritmo de aprendizaje por refuerzo, reconoce los patrones que benefician los resultados para conseguir el objetivo que ofrece una mayor recompensa. Cabe destacar que estos experimentos no tienen como objetivo obtener el mejor resultado ni superar ninguna *baseline* propuesta.

En primer lugar se realizó un experimento configurando los parámetros de la red neuronal del siguiente modo. Se ha utilizado el optimizado SGD, con un *learning rate* de 0,01 y *momentum* de 0,2.

Tras varias ejecuciones los resultados que mejor representan al resto de ejecuciones, son, los mostrados en la Figura 4.6. La línea azul representa el número de

acciones que realiza hasta colisionar y en color anaranjado se representa la media de acciones que realiza hasta colisionar. En estos resultados se puede observar que el número de acciones que le permite estar en el aire, no es incremental. El algoritmo, en esta configuración y con el número de iteraciones mostrada en la figura, no son los adecuados para aprender.

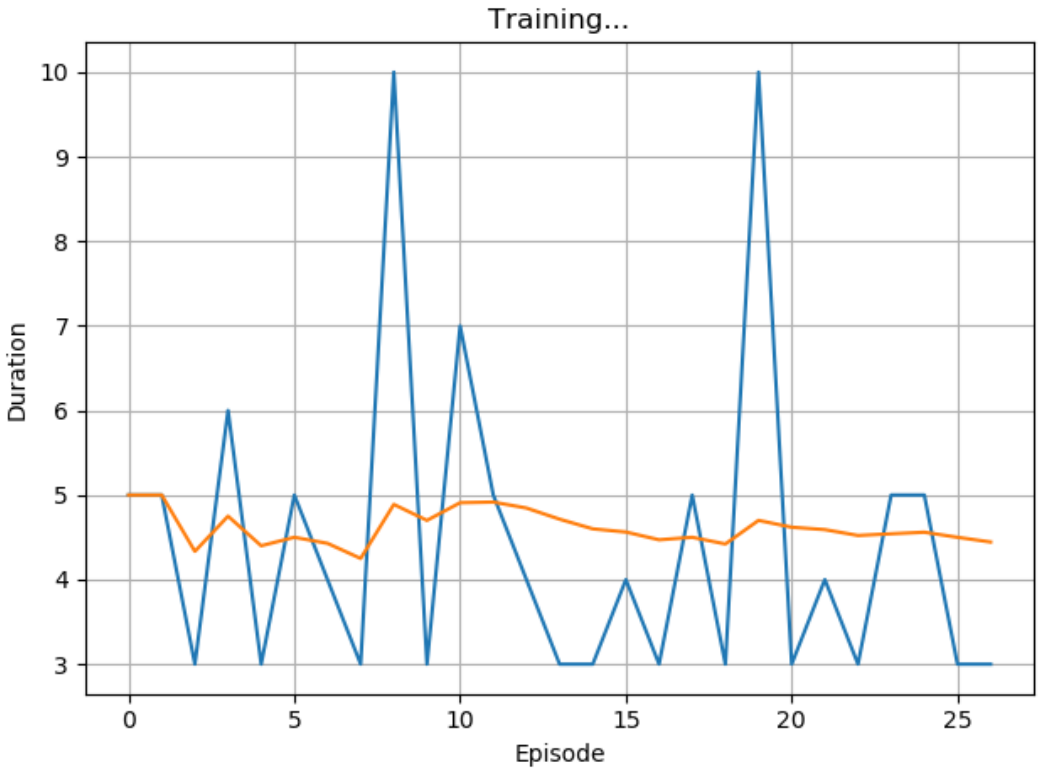



**Figura 4.6:** Experimento 1

El siguiente experimento se ajustaron los parámetros de *momentum* a 0,5 y el resto con la configuración idéntica a la anterior, optimizador SGD, *learning rate* 0,01.

El algoritmo con una esta configuración mostró mejores resultados. En la Figura 4.7 se puede observar que cada episodio que avanza, aumenta el número de acciones que le permite estar más tiempo en vuelo sin colisionar. En este caso, a partir del episodio 4, la media de las acciones realizadas en cada episodio, incrementa.

En el conjunto de Figuras 4.8 se han realizado una serie de capturas que indican el camino que ha tomado el agente para llevar al drone hasta el final de la calle. En la Figura 4.8a el drone ha realizado las acciones que le han llevada a un estado en el que se encuentra con una pared en la dirección que obtiene mayor recompensa. En la ejecución, ha colisionado varias ocasiones con esta pared. De lo contrario, en la Figura 4.8b tiene la posibilidad de tomar las acciones que van en la dirección que obtiene mayor recompensa. En la Figura 4.8c ocurre lo mismo que en la primera, tiene una pared frontal, pero en este caso la solución no es dirigirse hacia la izquierda, el algoritmo deberá aprender que debe dirigirse hacia donde no existan paredes donde poder colisionar. Por último, y tras varias

iteraciones del algoritmo, en la Figura 4.8d se observa que el drone ha aprendido cómo llegar hasta el final de la calle consiguiendo la mayor recompensa.

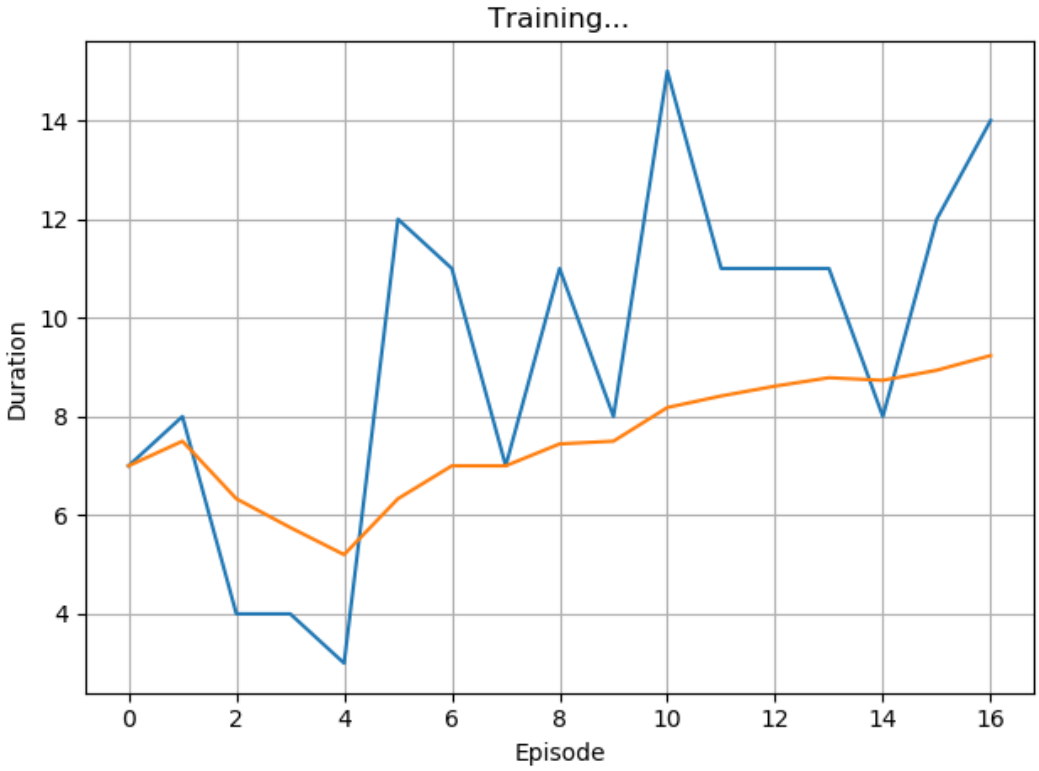



**Figura 4.7:** Experimento 2

**(a)** Captura 1

**(b)** Captura 2

**(c)** Captura 3

**(d)** Captura 4

**Figura 4.8:** Ejemplo de ejecución del algoritmo de aprendizaje por refuerzo

### 4.2.3. Conclusión

Dando por finalizado el último bloque, cabe destacar las dos propuestas que se han implementado utilizando las herramientas que proporciona el framework desarrollado. En este bloque se va más allá de la herramienta que se ha desarrollado y se detalla como se puede utilizar. Con las dos soluciones propuestas, se muestra la variedad de mecanismos que contiene esta herramienta, la mensajería instantánea entre agentes, los comportamientos que pueden adquirir los agentes, el control del drone en el simulador, el uso de los sensores para aplicar algoritmos, el uso de algoritmos de distintas categorías, en especial de reinforcement learning.

Con este bloque se finaliza la memoria y da paso a las conclusiones finales y trabajos futuros.

# CAPÍTULO 5
# Conclusiones y trabajos futuros

Como parte final del documento y para finalizar la memoria, en este capítulo se realizarán unas conclusiones globales del proyecto. Seguidamente los trabajos futuros que se pueden realizar a partir de la herramienta desarrollada.

## 5.1 Conclusiones

Un vez finalizado el proyecto y la memoria, es posible obtener unas conclusiones finales respecto al proyecto desarrollado. El proyecto consta de un serie de objetivos y subobjetivos que se han realizado correctamente y permiten obtener una conclusión final.

En primer lugar, se ha abordado exitosamente el objetivo que consta en estudiar las posibles opciones para interconectar un sistema inteligente con un simulador de drones. A su vez, este objetivo consta de subobjetivos, tal y como recopilar información de distintos sistemas multiagente y simuladores de drones.

Respecto a la información que se ha recopilado sobre los sistemas multiagente se encuentra los sistemas Jason y Jade de los cuales se detalla suficiente información. Pero cabe destacar el sistema de agentes inteligentes SPADE que fue el seleccionado el uso en la herramienta desarrollada.

En cuanto a la recopilación de información sobre simuladores de drones se encuentran el sistema operativo ROS, orientado a robots, Gazebo, un entorno de visualización del sistema ROS y V-REP, una interfaz de visualización del entorno ROS. Pero ante estos simuladores hay que destacar el simulador de vehículos AirSim, que permite la simulación de drones y coches y muchas más ventajas que se detallan en la memoria.

Por ello fue el simulador seleccionado para desarrollar, junto con el sistema de agentes SPADE, la herramienta que se ha realizado en este proyecto.

Una vez seleccionados los sistemas que se requerían para la implementación de la herramienta, se abordó exitosamente el objetivo de diseñar e implementar una plataforma de simulación de drones que permitiese la implementación y comparación de diferentes estrategias de coordinación en sistemas multiagente, así como desarrollar nuevos casos.

De esta forma, la plataforma desarrollada contiene las herramientas necesarias para poder realizar nuevas estrategias de coordinación y desarrollar nuevos casos como por ejemplo la integración de algoritmos de reinforcement learning. De este modo queda completado el segundo objetivo.

Respecto al último punto de los objetivos que hace foco en realizar desarrollos para validar el correcto funcionamiento de la herramienta desarrollada. Que a su vez contiene como subobjetivos la creación de un caso de coordinación entre agentes en el simulador de drones y la implementación de un algoritmo de aprendizaje por refuerzo para conducción autónoma en drones. Este objetivo se ha superado correctamente, tal y como se detalla a continuación.

En primer lugar, la creación de un caso de coordinación entre agentes se ha realizado correctamente ya que los agentes se coordinan para alcanzar el objetivo que trata en que un dron siga al otro en el modo "FollowMe". Además, este tipo de coordinación realizada por mensajes instantáneos se puede trasladar a cualquier tipo de estrategia de coordinación incrementando el número de agentes y haciendo más compleja la propia coordinación.

En segundo lugar la implementación de un modelo de aprendizaje reforzado se ha realizado de forma exitosa ya que el algoritmo implementado ofrece evidencias de que cuantas más iteraciones realiza mejores resultados obtiene.

Tras esta enumeración de los cumplimientos de todos los puntos y subpuntos de los objetivos. Se puede decir que se han completado correctamente todos los objetivos establecidos para este proyecto, dicho de otro modo el proyecto ha finalizado satisfactoriamente.

## 5.2 Trabajos futuros

En cuanto a trabajos futuros, el proyecto ofrece grandes posibilidades de ampliación por distintas partes. A continuación, se detallan algunas de ellas.

En primer lugar, y una de las más interesantes. Sería realizar una investigación más detallada sobre algoritmos de reinforcement learning para poder optimizar por completo el algoritmo DQN. De este modo, proporcionar un modelo que permita la conducción autónoma lo más perfecta posible. Además, ya que lo permite la plataforma desarrollada, migrar el modelo a un drone real para realizar pruebas en el mundo real.

En cuanto a la coordinación de agentes mediante el uso de drones, poder implementar y simular estrategias de coordinación que permitan realizar acciones de forma cooperativa, como el transporte de mercancías, la vigilancia de instalaciones, apagado de incendios de forma coordinada, búsqueda de personas perdidas...

En general, la herramienta desarrollada permite implementar cualquier tipo de aplicación en la que se utilicen drones y visualizarlo en el entorno de simulación.

# Bibliografía


[1] Simulador de Drones Consultado el 27 de junio de 2018 https://www.dronethusiast.com/simulador-de-drones/

[2] Louis Charles Breguet Consultado el 09 de Noviembre de 2019 https://es.wikipedia.org/wiki/Louis_Charles_Breguet

[3] The Early Days Of Drones – Unmanned Aircraft From World War One And World War Two Consultado el 09 de Noviembre de 2019 https://www.warhistoryonline.com/military-vehicle-news/short-history-drones-part-1.html

[4] Fritz X Consultado el 09 de Noviembre de 2019 https://es.wikipedia.org/wiki/Fritz_X

[5] La historia de los drones Consultado el 09 de Noviembre de 2019 https://es.digitaltrends.com/drones/la-historia-de-los-drones/

[6] Parrot AR.Drone Consultado el 09 de Noviembre de 2019 https://es.wikipedia.org/wiki/Parrot_AR.Drone

[7] Amazon Prime Air Consultado el 10 de Noviembre de 2019 https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=803772001

[8] Drone Autónomo Consultado el 10 de Noviembre de 2019 https://dronesguard.azurdrones.com/product/skeyetech/

[9] Autonomous Navigation of UAVs in Large-Scale Complex Environments: A Deep Reinforcement Learning Approach Consultado el 11 de Noviembre de 2019 http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8600371&isnumber=8667487

[10] AirSim is a simulator for drones, cars and more Consultado el 12 de Noviembre de 2019 https://microsoft.github.io/AirSim/

[11] The Microsoft Cognitive Toolkit Consultado el 12 de Noviembre de 2019 https://docs.microsoft.com/es-es/cognitive-toolkit/

[12] The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. Consultado el 12 de Noviembre de 2019 https://www.ros.org/news/robots/uavs/

[13] Parrot Ground SDK Consultado el 12 de Noviembre de 2019 https://developer.parrot.com/

[14] Smart Python Agent Development Environment Consultado el 12 de Noviembre de 2019 https://pypi.org/project/spade/

[15] Deep Reinforcement Learning with Double Q-learning Consultado el 12 de Noviembre de 2019 https://arxiv.org/abs/1509.06461

[16] Epic Games Launcher Consultado el 16 de Noviembre de 2019 https://www.unrealengine.com/en-US/download

[17] Unreal Marketplace Environments Consultado el 16 de Noviembre de 2019 https://www.unrealengine.com/marketplace/en-US/content-cat/assets/environments

[18] LIDAR Consultado el 19 de Noviembre de 2019 https://es.wikipedia.org/wiki/LIDAR

[19] ALGORITMO DE APRENDIZAJE POR REFUERZO CONTINUO PARA EL CONTROL DE UN SISTEMA DE SUSPENSIÓN SEMI-ACTIVA Mª Jesús López Boada, Beatriz López Boada, Vicente Díaz López Universidad Carlos III de Madrid Consultado el 23 de Noviembre de 2019 https://www2.uned.es/ribim/volumenes/Vol9N2Julio_2005/V9N2A08%20Lopez%20Boada.pdf

[20] KAELBLING, Leslie Pack; LITTMAN, Michael L.; MOORE, Andrew W. Reinforcement learning: A survey. Journal of artificial intelligence research, 1996, vol. 4, p. 237-285.

[21] Watkins, C.J.C.H. & Dayan, P. Mach Learn (1992) 8: 279. https://doi.org/10.1007/BF00992698

[22] Entorno de simulación Gazebo Consultado el 23 de Noviembre de 2019 http://gazebosim.org/

[23] Entorno de simulación V-REP Consultado el 23 de Noviembre de 2019 http://www.coppeliarobotics.com/

[24] JAVA Agent DEvelopment Framework consultado el 24 de Noviembre de 2019 https://jade.tilab.com/

[25] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015) doi:10.1038/nature14236 https://www.nature.com/articles/nature14236.pdf

[26] Rao, Anand S., Van de Velde, Walter and Perram, John W. AgentSpeak(L): BDI agents speak out in a logical computable language Agents Breaking Away 1996, Springer Berlin Heidelberg ISBN:978-3-540-49621-2

[27] Bellifemine, Fabio and Poggi, Agostino and Rimassa, Giovanni Developing Multi-agent Systems with JADE JADE (Java Agent Development Framework) is a software framework to make easy the development of multi-agent applications in compliance with the FIPA specifications ISBN:978-3-540-44631-6