Revisión memoria TFG v1.0

Entorno de simulación para el entrenamiento mediante Reinforcement Learning del vuelo autónomo de un cuadricóptero

Índice

1. *Introducción OK*
2. *Objetivos OK*
3. *Resultados*
4. *Conclusiones*
5. *Bibliografía*

Anexos:

- *Estado del Arte OK*
- *Tecnologías Empleadas OK*

INTRODUCCION

Una de las tecnologías que más destacan y llaman la atención en estos últimos años son los llamados drones, definidos formalmente como *aeronaves no tripuladas* por la RAE [http://dle.rae.es/?id=ED2QqnQ]. Estos aparatos se han convertido en una atracción para casi cualquier ámbito, comenzando en usos militares y llegando a ser juguetes de ocio, no sin pasar por usos comerciales como el transporte o relacionados con seguridad y defensa.

Su bajo coste, reducido tamaño y la posibilidad de manejarlos remotamente les da un potencial inmenso para una infinidad de usos, entre otras cosas, en tareas que debería hacer una persona pudiendo tener que asumir riesgos o peligros, como puede ser la vigilancia desde el aire en zonas de conflicto, la búsqueda de supervivientes tras una catástrofe natural o la actuación en incendios forestales. Sin embargo, para algunas tareas los recursos humanos son limitados y no es viable ocupar a un operario pilotando un dron, apareciendo así la necesidad de diseñar aeronaves que puedan llevar a cabo su cometido de forma autónoma.

Volar de forma autónoma es una tarea muy compleja en la que pueden suceder innumerables situaciones diferentes e imprevisibles, a las que se debe dar respuesta de la forma más rápida y acertada posible. Es necesario dotar a los drones de inteligencia para que puedan desenvolverse por sí mismos y aprender de su propia experiencia. Esta necesidad se puede satisfacer con técnicas de Reinforcement Learning (RL o Aprendizaje por Refuerzo), un área del Machine Learning que agrupa técnicas de aprendizaje automático basadas en la experimentación y la experiencia de la propia máquina. Dichas técnicas comparten una etapa de entrenamiento basada en sucesivos episodios de prueba y error en los que la máquina debe

procurar llegar a un objetivo. Durante estos episodios el aparato aprende de sus propias interacciones con el entorno que lo rodea.

Con la pretensión de experimentar con estas recientes tecnologías y la ilusión de llegar a proporcionar una funcionalidad útil a los posteriores desarrolladores de este campo, en este trabajo fin de grado se construirá un framework de simulación, desarrollo y entrenamiento de sistemas RL orientado a la experimentación en el vuelo autónomo de drones.

OBJETIVOS

El objetivo final de este trabajo es la obtención de un **framework de simulación, desarrollo y entrenamiento de sistemas *Reinforcement Learning* orientado a la experimentación en el vuelo autónomo de drones**. Es decir, un conjunto de herramientas que faciliten la experimentación, el desarrollo y el posterior entrenamiento en simulación de sistemas inteligentes (concretamente basados en técnicas de Reinforcement Learning) para drones.

Para crear esta arquitectura se partirá de una serie de tecnologías ya existentes como son: el entorno de simulación ***Microsoft AirSim***, el controlador de vuelo ***PX4 autopilot***, el freamwork robótico ***ROS***, y el toolkit para desarrollo de sistemas basados en Reinforcement Learning, ***GYM***. Todas estas tecnologías se recogen en el apartado "*TECNOLOGÍAS EMPLEADAS*" de los anexos, donde son introducidas y brevemente explicadas, así como también son señaladas las razones que han llevado a escogerlas para este proyecto.

Durante la implementación de este framework deberán integrarse las 4 tecnologías juntas, solucionando las diferentes incompatibilidades y problemáticas integraciones que guardan algunas de ellas.

RESULTADOS

. . . PRÓXIMAMENTE . . .

CONCLUSIONES

. . . PRÓXIMAMENTE . . .

BIBLIOGRAFÍA

. . . PRÓXIMAMENTE . . .

ANEXOS

ESTADO DEL ARTE

Desarrollar y experimentar con algoritmos para vehículos autónomos es un proceso muy complejo que requiere tiempo y muchas pruebas que no siempre terminan bien. Además, para desarrollar sistemas inteligentes a menudo es necesario recopilar una gran cantidad de datos de entrenamiento en diferentes condiciones y entornos. Realizar este proceso en el mundo real no es viable, por ello para llevarlo a cabo se utilizan entornos de simulación, sistemas informáticos que procuran imitar lo más fielmente posible el comportamiento que tendría el

vehículo en un entorno real. Este tipo de software nos permite experimentar, analizar y entrenar a nuestros aparatos evitando los importantes inconvenientes que implican hacer lo propio en la realidad: accidentes, golpes, costosas roturas, desajustes, etc.

Cabe destacar que no solo el entorno de simulación es importante, sino también su compatibilidad e integración con otras tecnologías tales como los **controladores de vuelo**, que son dispositivos encargados de controlar los diferentes componentes del vehículo; los **protocolos de comunicación** para interconectar el simulador y el controlador, así como también otros componentes; o los **frameworks de programación** para desarrollar el software que definirá el comportamiento del dron.

Debido a que el interés de este trabajo reside en encontrar un sistema que agrupe todas estas tecnologías para facilitar la implementación de sistemas de vuelo autónomo, es esencial hacer un análisis del estado del arte al respecto:

***** Datos comparativos entre simuladores sacados del paper de AirSim **[*** https://www.microsoft.com/en-us/research/wp-content/uploads/2017/07/1705.05065.pdf ***]***

A día de hoy existen variedad de entornos de simulación para drones, y entre ellos, uno de los más utilizados es **Gazebo**. [http://gazebosim.org/tutorials?tut=guided_b1&cat=#WhatisGazebo?] Un simulador de código abierto para todo tipo de robots, capaz de simular ambientes interiores y exteriores complejos con alto grado de fidelidad. Provee una amplia biblioteca de robots y entornos, gran cantidad de sensores, y plugins para integrar muchas otras tecnologías, tales como controladores de vuelo, frameworks de programación, sistemas de entrenamiento, etc. Algunas de las grandes ventajas que tiene utilizar Gazebo son la amplia comunidad de usuarios que participan activamente generando conocimiento y la gran cantidad de documentos académicos basados en la experimentación con este simulador.

La principal desventaja de Gazebo es su falta de realismo en la simulación, debido a su orientación a robots genéricos y a la complejidad que implica crear entornos realistas a gran escala y con gran cantidad de detalles que simulen mejor el mundo real.

Una alternativa que pretende solventar la falta de realismo es **Hector** [https://pdfs.semanticscholar.org/3ed3/948827e0949770e8583b51bd0fedf4fd73fe.pdf], un trabajo orientado específicamente a cuadricópteros, y que integra el simulador Gazebo junto al framework robótico ROS [Ref al anexo de ROS] . Lo que ofrece Hector [http://wiki.ros.org/hector_quadrotor] es una simulación más realista de la física del vuelo y el comportamiento del aparato, pero por el contrario carece de soporte para la integración de controladores de vuelo como PX4 y protocolos de comunicación estándares como MavLink [https://mavlink.io/en/].

De forma similar, **RotorS** [https://link.springer.com/chapter/10.1007/978-3-319-26054-9_23] es un framework modular de simulación que permite diseñar drones y desarrollar algoritmos para controlar su comportamiento y estimar su estado de forma más precisa. Al igual que en el caso anterior, RotorS [http://wiki.ros.org/rotors_simulator] se basa en ROS y Gazebo, pero en este caso sí se soporta la integración de controladores de vuelo.

Garantizando la compatibilidad de controladores de vuelo y alejándose de la simulación de Gazebo, existe otro software llamado **jMAVSim** [https://pixhawk.org/dev/hil/jmavsim], un extremadamente sencillo entorno de simulación diseñado en el contexto del proyecto Pixhawk

[http://pixhawk.org/] con el objetivo de experimentar con el firmware de la plataforma del propio proyecto. Se trata de un entorno muy ligero y fácil de utilizar que consta de un motor de renderizado muy simple, imposibilitando la generación de objetos en la escena. Además no aporta ningún tipo de integración con otras tecnologías.

Otro grupo de entornos de simulación es aquel basado en el potente motor de videojuegos creado por Epic Games [https://www.epicgames.com/site/es-ES/home], el Unreal Engine [https://www.unrealengine.com/en-US/what-is-unreal-engine-4]. Este grupo de frameworks se caracteriza por el alto grado de realismo tanto en gráficos como en física de comportamiento de los vehículos, debido a las avanzadas tecnologías desarrolladas por su creador. Notables ejemplos de este grupo son Sim4CV, DroneSimLab y Microsoft AirSim.

El primero de ellos, **Sim4CV** [https://sim4cv.org] [https://arxiv.org/pdf/1708.05869.pdf] [https://www.youtube.com/watch?v=SqAxzsQ7qUU], se presenta como un simulador fotorrealista orientado claramente a la experimentación en campos muy avanzados de la visión artificial. Gracias al potente motor gráfico, consigue alcanzar un alto nivel de realismo en la física de los vehículos. Como desventaja está su limitación para integrar tecnologías externas al propio framework.

En contraste con el anterior, **DroneSimLab** [https://arxiv.org/abs/1708.01938][https://github.com/orig74/DroneSimLab] es un framework con énfasis en la modularidad y la flexibilidad. Tanto es así, que no solo integra varios controladores de vuelo y motores de simulación de sensores, además del motor gráfico aportado por Unreal, sino que facilita la integración de otros muchos componentes mediante el uso de contenedores. Este conjunto de herramientas tiene una orientación genérica, permitiendo experimentar en visión artificial, vehículos autónomos y otros muchos campos para gran cantidad de vehículos. Claramente, este framework es la alternativa más cercana e interesante al objetivo del presente trabajo.

Microsoft AirSim [https://www.microsoft.com/en-us/research/wp-content/uploads/2017/07/1705.05065.pdf] forma parte del proyecto *Aerial Informatics and Robotics Platform* [https://www.microsoft.com/en-us/research/project/aerial-informatics-robotics-platform/] desarrollado por Microsoft, y consiste en un entorno de simulación basado en el motor Unreal Engine y en un conjunto de herramientas para el desarrollo de sistemas autónomos. Se trata de una plataforma que cuenta con las tecnologías más avanzadas de simulación en cuanto a física de los vehículos y visualización del entorno, reduciendo al máximo la diferencia entre realidad y simulación, y ayudando así a crear sistemas autónomos más avanzados, seguros y eficaces.

AirSim es un framework de código abierto en continuo desarrollo que pretende convertirse en una plataforma modular, altamente compatible y tecnológicamente avanzada para contribuír al desarrollo e investigación en vehículos autónomos, con la intención de crear “una comunidad de desarrolladores que impulse el estado del arte en este campo”. [Cita al paper de airsim, ultimas 2 lineas antes de Related Work]

La reciente novedad de AirSim y su apuesta por la muy alta fidelidad de la simulación, además de la creciente presencia de trabajos realizados con esta plataforma y el prestigio de la propia compañía, han sido los factores impulsores de la elección de este entorno de simulación para realizar este trabajo de fin de grado.

Existen muchos más entornos de simulación, pero la mayoría están orientados a robots de tipo genérico (como **ARGoS** [http://www.argos-sim.info/index.php] o **V-REP** [

http://www.coppeliarobotics.com/index.html]), por lo que no incorporan una simulación realista del comportamiento de un dron ni soportan compatibilidad con la mayoría de tecnologías necesarias, o bien no son de código abierto (como **XPLANE** [http://www.x-plane.com/] o **RealFlight** [https://www.realflight.com/]).

TECNOLOGÍAS EMPLEADAS

El framework de simulación, desarrollo y entrenamiento de sistemas Reinforcement Learning que se crea, explica y documenta en este trabajo fin de grado, se construye a partir de diferentes tecnologías específicas del campo de la robótica, la aeronáutica y la inteligencia artificial. Puesto que estos campos no se consideran en absoluto comúnmente conocidos, en los siguientes apartados se explicarán algunos conceptos para el lector que no esté familiarizado con la materia.

CONTROLADOR DE VUELO: PX4

El controlador de vuelo o "autopilot" es uno de los componentes base más esenciales de cualquier vehículo multi-rotor. Se trata de una placa electrónica a la que se conectan los sensores (GPS, barómetros, giroscopios, acelerómetros, etc) y los actuadores (motores) del vehículo. Esta placa lleva instalado un firmware que se encarga de controlar el comportamiento de los actuadores en función de las lecturas de los sensores y del estado o posición del vehículo al que se desee llegar, con la finalidad de mantener la aeronave estable así como de cambiar la velocidad o la dirección de movimiento de la misma, de forma controlada. [https://www.tomshardware.com/reviews/multi-rotor-quadcopter-fpv,3828-2.html].

Siendo más concretos, la labor principal del autopilot es tomar el estado o posición deseada del vehículo como entrada, estimar el estado real a partir de los datos que generan los sensores, y controlar el comportamiento de los actuadores de tal forma que el estado real sea lo más parecido posible al estado deseado. [https://github.com/Microsoft/AirSim/blob/master/docs/flight_controller.md]

{{ esquema_funcion_autopilot.png – Aquí irá una imagen que ilustra un poco lo de anterior}}

Cuando se utilizan entornos de simulación, los sensores y los actuadores son emulados por la física del vehículo, de forma que el controlador de vuelo consume los datos simulados de los sensores, y genera unas señales que recibirán los actuadores del vehículo simulado como entrada. Para integrar el autopilot en un entorno de simulación, existen dos métodos diferentes: SITL y HITL.

[https://github.com/Microsoft/AirSim/blob/master/docs/flight_controller.md]

[https://wiki.paparazziuav.org/wiki/HITL]

[https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1109&context=mae_facpub]

SITL (Software In The Loop) es una técnica que se basa en emular el hardware del controlador ejecutando directamente el firmware del autopilot sobre el equipo de desarrollo, de forma que interactúe con los sensores y actuadores del entorno de simulación.

HITL (Hardware In The Loop), en cambio, se basa en conectar el hardware real del autopilot al equipo de desarrollo, de forma que el controlador de vuelo interactúe directamente con los sensores y actuadores del entorno de simulación.

El método HITL es el más cercano al funcionamiento real, pero la ventaja más significativa del método SITL es que elimina por completo la necesidad de adquirir una plataforma hardware. Además, con el método HITL pueden darse problemas de desincronización si el reloj del simulador y del controlador tienen velocidades y/o precisiones distintas, o si la conexión entre el autopilot y el equipo de desarrollo no es capaz de transferir los datos suficientemente rápido. Sin embargo, cuando se trabaja con equipos de desarrollo con recursos limitados, el método HITL evita la sobrecarga de trabajo del equipo, permitiendo emplear toda la potencia disponible en el entorno de simulación, que normalmente requiere gran cantidad de recursos.

El proyecto *Dronecode* [https://www.dronecode.org/] es una plataforma de código abierto que trabaja en el desarrollo de diversas herramientas y tecnologías para drones, como protocolos de comunicación, controladores de vuelo (hardware y firmware), simuladores, etc. Su objetivo es crear una comunidad de desarrolladores que contribuyan al proyecto para ofrecer una plataforma tecnológicamente avanzada de herramientas para vehículos no tripulados.

PX4 autopilot [http://px4.io/] es un firmware de controlador de vuelo [https://github.com/PX4/Firmware] muy versátil y ampliamente utilizado, que forma parte de la *Dronecode Platform*. Sus características más destacables son la gran cantidad de vehículos que puede controlar (no solo todo tipo de aeronaves, sino también vehículos terrestres y submarinos), la variedad de modos de vuelo y características de seguridad que ofrece, y especialmente su compatibilidad con multitud de sensores, periféricos y plataformas hardware. [https://docs.px4.io/en/getting_started/px4_basic_concepts.html]

En este trabajo fin de grado, se utilizará el PX4 en modo SITL.

La elección de este autopilot se debe a diversos factores: en primer lugar, es un autopilot compatible con el entorno de simulación de Microsoft; además utiliza el protocolo MAVlink [https://mavlink.io/en/] (también soportado por AirSim), y es compatible con el framework ROS [Ref al anexo de ROS] para el desarrollo de algoritmos, el envío de comandos, etc; por otra parte, el hecho de que este autopilot se pueda montar sobre diferentes plataformas [https://docs.px4.io/en/flight_controller/] (Pixhawk [https://pixhawk.org/], Qualcomm Snapdragon Flight [https://docs.px4.io/en/flight_controller/snapdragon_flight.html], Intel Aero Compute Board [https://software.intel.com/en-us/aero/compute-board], Raspberry Pi 2/3 Navio2 [https://docs.px4.io/en/flight_controller/raspberry_pi_navio2.html], etc) aporta una gran fexibilidad y compatibilidad al sistema.

La integración de PX4 con AirSim se realizará por el método SITL, ya que se pretende obtener un framework de simulación que no dependa de ningún elemento hardware más allá del propio equipo de desarrollo.

FRAMEWORK DE PROGRAMACIÓN: ROS

ROS (Robot Operating System) [http://www.ros.org/] consiste en un conjunto de herramientas, librerías y convenciones de código abierto, desde drivers hasta avanzados algoritmos estado del arte, que componen un framework de programación orientado a robótica. [http://www.ros.org/about-ros/]

A pesar de su nombre, a menudo se define ROS como un meta sistema operativo más que como un sistema operativo convencional. Esto se debe a que, aunque tiene características

propias de los sistemas operativos, como la abstracción de hardware, el control de dispositivos a bajo nivel o la gestión de paquetes; también tiene rasgos más propios de middlewares o frameworks, como la implementación de funcionalidades básicas del sistema, el paso de mensajes entre procesos, y la existencia de herramientas y librerías de algoritmos a alto nivel. [http://wiki.ros.org/ROS/Introduction#What_is_ROS.3F]

ROS se caracteriza por basarse en un sistema de computación distribuida, donde cada proceso se denomina nodo, y cada nodo es responsable de una tarea. Los diferentes nodos pueden comunicarse mediante un modelo de publicación/subscripción, para comunicaciones asíncronas y visibles para todos los nodos, o bien mediante un modelo de petición/respuesta, si lo que se pretende es establecer una comunicación síncrona y privada entre dos nodos. [http://wiki.ros.org/ROS/Concepts]

*****Paper de ROS [* http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf *]*

{{ esquema_arquitectura_ros.jpg – Aquí irá un esquema que describe lo del parrafo anterior }}

En el contexto de este trabajo, ROS resulta un framework de gran utilidad ya que permite acceder a las lecturas de los sensores y las cámaras, enviar comandos a través de mavlink, e implementar hasta los algoritmos más complejos. Además, esta plataforma ofrece integración con otros proyectos de código abierto como OpenCV [https://opencv.org/], una importante librería para visión artificial, o PointCloudLibrary [http://pointclouds.org/], una librería de percepción 3D, entre otros [http://www.ros.org/integration/]. Estas librerías son de gran utilidad a la hora de implementar algoritmos de visión artificial para guiar el dron.

Por otra parte, existen implementaciones de ROS en Python y en C++ (entre otros lenguajes) que añaden a la versatilidad implícita de ROS toda la potencia de dichos lenguajes.

Por todas estas razones, en este trabajo fin de grado se integrará ROS como framework de desarrollo general.

ENTORNO DE SIMULACIÓN: AIRSIM

Nota al lector: en este apartado se mencionan los conceptos de **controlador de vuelo**, **autopilot**, **PX4**, **modo SITL**, **modo HITL** y **ROS**, que se explican en los apartados de "CONTROLADOR DE VUELO: PX4" y "FRAMEWORK DE PROGRAMACIÓN: ROS". Se recomienda al lector ajeno a la materia, que revise dichos apartados antes de continuar con el actual.

Microsoft AirSim [https://github.com/Microsoft/AirSim] es un entorno de simulación desarrollado sobre el motor Unreal Engine (UE) que ofrece simulaciones física y visualmente realistas. Está implementado como un plugin multiplataforma para UE que se puede integrar en cualquier proyecto de Unreal.

{{ drone_depth_materials.png – Aquí irá una imagen chula del simulador, solo para que se aprecie como es }}

Se trata de una plataforma modular en continuo desarrollo, con gran énfasis en la escalabilidad, dando la posibilidad de añadir nuevos vehículos, plataformas hardware y protocolos software. Además se trata de un proyecto de código abierto, por lo que cualquier desarrollador puede contribuir y utilizarlo.

Incluye un conjunto de APIs [https://github.com/Microsoft/AirSim/blob/master/docs/apis.md] que permiten leer el estado del vehículo, de los sensores y de las cámaras, así como también enviar comandos al vehículo.

Este entorno de simulación provee 3 interfaces diferentes de uso [https://github.com/Microsoft/AirSim/blob/master/README.md]:

- **Control manual:** se trata de una interfaz en la que el usuario controla el comportamiento del vehículo con ayuda de un joystick control remoto conectado al simulador.
- **Control automático:** con esta interfaz, el usuario no tiene control directo sobre el vehículo. Con la ayuda de las diferentes APIs y frameworks disponibles, es posible crear algoritmos para experimentar con vehículos autónomos.
- **Modo Visión Artificial:** esta interfaz está contemplada para usuarios que solamente están interesados en la parte gráfica del entorno. En este modo el simulador elimina el vehículo y su física de comportamiento, y ofrece el uso de APIs de visión artificial para explorar el entorno y experimentar con algoritmos en este campo.

{{ overview.png – Aquí tendremos una imagen de la arquitectura general del simulador }}

Actualmente AirSim ofrece la posibilidad de utilizar dos controladores de vuelo: **Simple Flight** en modo SITL o **PX4** en modo SITL o HITL. En el futuro se pretende compatibilizar otros autopilots como ROSflight [http://docs.rosflight.org/en/latest/] y Hackflight [https://github.com/simondlevy/Hackflight]

Simple Flight es un controlador de vuelo desarrollado por Microsoft y entendido como un conjunto de algoritmos empaquetados en una librería de cabeceras C++ sin dependencias. La principal ventaja que ofrece es la posibilidad de utilizar este autopilot en simulación o sobre un hardware real indistintamente, al contrario que la mayoría de controladores, que requieren una complicada configuración para hacerlos funcionar en modo SITL. Actualmente Simple Flight está en desarrollo, y la intención es adaptarlo a diferentes controladores hardware ampliamente utilizados como son las placas Pixhawk V2 [https://docs.px4.io/en/flight_controller/pixhawk-2.html] y Naze32 [https://quadquestions.com/Naze32_rev6_manual_v1.2.pdf].

PX4 [https://github.com/Microsoft/AirSim/blob/master/docs/px4_setup.md] es un controlador de vuelo de código abierto ampliamente utilizado y con soporte para multitud de dispositivos hardware y sensores. AirSim proporciona soporte para integrar este autopilot en modo SITL o HITL, sin embargo, la integración de este controlador sigue resultando considerablemente complicada en ambos casos.

En cuanto a compatibilidad con otras tecnologías, actualmente AirSim utiliza una librería propia [https://github.com/Microsoft/AirSim/tree/master/MavLinkCom] basada en MAVlink [https://mavlink.io/en/], un protocolo de comunicación para drones comúnmente utilizado, que permite la integración de PX4. Además también ofrece soporte para integrar ROS, permitiendo interactuar con los datos de la simulación directamente desde dicho framework.

REINFORCEMENT LEARNING: GYM

El Reinforcement Learning (RL) o Aprendizaje por Refuerzo es un área del Machine Learning que permite a un agente (entidad que va a aprender) descubrir un comportamiento óptimo de forma autónoma a través de interacciones de prueba y error con el entorno que lo rodea. En lugar de proporcionarle al agente la solución a un problema, en RL el entorno le devolverá una recompensa positiva (refuerzo) o negativa (castigo) al agente según la acción que haya llevado

a cabo. Con esto, el agente aprenderá qué tipo de conductas son las que debe mantener para obtener recompensas cada vez más positivas.

La razón de utilizar este tipo de aprendizaje para desarrollar sistemas autónomos reside en la inmensa complejidad que supone programar las infinitas situaciones que se pueden dar en un entorno real y que un vehículo debe saber resolver. Ante esta complejidad, resulta muy atractiva la solución de crear un sistema que pueda generar ese conocimiento por sí mismo a través de la asociación de relaciones causa-efecto de su propia experiencia.

[Referencias:
Paper http://www.ias.tu-darmstadt.de/uploads/Publications/Kober_IJRR_2013.pdf
Paper https://arxiv.org/pdf/cs/9605103.pdf
Libro: Artificial Intelligence, A Modern Approach
Libro: Reinforcement Learning, An Introduction]

{{ Esquema RL agente – entorno – Aquí irá un esquema que ilustra lo anterior }}

El objetivo de este trabajo es la obtención de un framework destinado a desarrollar y entrenar sistemas RL, por lo que dicho framework deberá tener un sistema de entrenamiento basado en prueba y error, es decir, en la repetición sucesiva de un episodio de entrenamiento al final del cual, se volverá a la situación inicial y se empezará de nuevo el episodio. Durante cada episodio, el dron intentará alcanzar su objetivo a base de ejecutar las diferentes acciones que tenga definidas (por ejemplo, moverse en una dirección o girar) e irá acumulando experiencia en función de los *refuerzos* que el sistema le devuelva.

Para implementar esta metodología de entrenamiento, se integrará Gym en nuestro proyecto, un toolkit de RL creado por OpenAI.

OpenAI [https://openai.com/about/] es una compañía de investigación sin ánimo de lucro especializada en Inteligencia Artificial (IA), que tiene como objetivo crear una Inteligencia Artificial General (AGI, por sus siglas en inglés) segura y accesible para toda la humanidad.

Gym [http://gym.openai.com/] consiste en un conjunto de herramientas para desarrollar y comparar algoritmos de RL. Se trata de un paquete de Python que contiene una colección de entornos y que provee unas guías de referencia para desarrollar agentes que interactúen con dichos entornos, así como para crear entornos nuevos. Tanto los entornos como los agentes siguen unas estructuras e interfaces determinadas, permitiendo así intercambiar los agentes y los algoritmos de aprendizaje que estos utilizan. De este modo, se pueden comparar diversos tipos de agentes entrenados en el mismo entorno.