UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Aplicación de Técnicas de Aprendizaje por Refuerzo para la Navegación de Robots Móviles

Trabajo Fin de Máster

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

AUTOR/A: Pescador Barreto, Germán Andrés

Tutor/a: Palanca Cámara, Javier

Cotutor/a: Julian Inglada, Vicente Javier

CURSO ACADÉMICO: 2022/2023

Universitat Politècnica
de València

Departamento de Sistemas Informáticos y Computación

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

Trabajo Fin de Máster

Aplicación de Técnicas de Aprendizaje por Refuerzo para la Navegación de Robots Móviles

Autor(a): Germán Andrés Pescador Barreto
Director(a): Javier Palanca Cámara
Cotutor(a): Vicente Javier Julián Inglada

Valencia, Septiembre 2023

Este Trabajo Fin de Máster se ha depositado en el Departamento de Sistemas Informàticos y Computación de la Universitat Politècnica de València para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Título: Aplicación de Técnicas de Aprendizaje por Refuerzo para la Navegación de Robots Móviles

Septiembre 2023

Autor(a):	Germán Andrés Pescador Barreto
Director(a):	Javier Palanca Cámara Departamento de Sistemas Informáticos y Computación Universitat Politècnica de València
Co-director(a):	Vicente Javier Julián Inglada Departamento de Sistemas Informáticos y Computación Universitat Politècnica de València

Resum

Este projecte d'investigació es centra en la formació d'un agent robòtic de navegació diferencial, utilitzant tècniques d'aprenentatge per reforç. L'objectiu principal és capacitar l'agent per a realitzar tasques de navegació en entorns desconeguts i complir amb èxit una tasca de cerca específica, utilitzant únicament la informació proporcionada pels seus sensors en temps real. Atès l'alt cost i la complexitat associats amb la formació en entorns físics reals, s'opta per dur a terme este procés en un entorn de simulació mitjançant l'ús del motor de videojocs Unity3D i la seua caixa d'eines d'aprenentatge per reforç ML-Agents.

Al llarg d'esta memòria relatem el procés incremental mitjançant el qual aconseguim formar amb èxit un model amb capacitats de generalització, i al final es proposen una sèrie de formes en què este model podria ser millorat o ampliat en base als resultats observats durant el procés d'experimentació.

Resumen

Este proyecto de investigación se enfoca en el entrenamiento de un agente robótico de navegación diferencial, utilizando técnicas de aprendizaje por refuerzo. El objetivo principal es capacitar al agente para realizar tareas de navegación en entornos desconocidos y cumplir con éxito una tarea de búsqueda específica, utilizando únicamente la información proporcionada por sus sensores en tiempo real. Dado el alto costo y la complejidad asociados con el entrenamiento en entornos físicos reales, se opta por llevar a cabo este proceso en un entorno de simulación mediante el uso del motor de videojuegos Unity3D y su kit de herramientas de aprendizaje por refuerzo ML-Agents.

A lo largo de esta memoria relatamos el proceso incremental mediante el cual logramos entrenar con éxito un modelo con capacidades de generalización, y al final se proponen una serie de formas en que este modelo podría ser mejorado o extendido en base a los resultados observados durante el proceso de experimentación.

Abstract

This research project focuses on training a differential navigation robotic agent using reinforcement learning techniques. The main objective is to train an agent able of performing navigation tasks in unknown environments and successfully complete a specific search task, relying solely on the information provided by its sensors in real-time. Given the high cost and complexity associated with training in real physical environments, the decision is made to carry out this process in a simulation environment using the Unity3D game engine and its ML-Agents (Machine Learning Agents) toolkit.

Throughout this report, we narrate the incremental process through which we successfully trained a model with generalization capabilities. In the end, a series of ways in which this model could be improved or extended are proposed based on the results observed during the experimentation process.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación

Existen registros de autómatas construidos hace más de dos mil años, aunque el término 'robot' no es acuñado hasta 1921 en una obra de ficción. Lejos de quedarse en la idea y aprovechando los continuos avances en el campo de la computación, se han ido sucediendo sistemas de control cada vez más sofisticados para un gran abanico de tareas. No obstante, a medida que la complejidad de estos sistemas ha aumentado, también lo ha hecho la dificultad de programarlos y controlarlos de manera efectiva. En muchos casos, se requiere de grandes equipos de ingenieros altamente especializados para desarrollar el código necesario, algo que puede suponer miles de horas de trabajo. Esto hace que en cualquier proyecto de robótica se acumulen los costes, tanto económicos como temporales, dejando fuera de juego a actores como grupos pequeños de investigación sin una buena financiación o compañías con bajo presupuesto.

Por su parte, la inteligencia artificial se encuentra en pleno apogeo. Una de las áreas más beneficiadas es la de los gráficos por ordenador, que están viviendo una revolución con la mejora constante de las unidades de procesamiento gráfico disponibles en el mercado, además de la llegada de nuevos paradigmas como los modelos generadores de imágenes, que enfocados de la manera correcta pueden aportar soluciones a problemas como el de síntesis de vistas nuevas, donde se busca renderizar poses de cámara no capturadas con anterioridad. Esto se consigue gracias a investigaciones que se suceden a un ritmo exponencial desde la publicación de NeRF (*Neural Radiance Fields*) [39], hasta llegar al reciente *Gaussian Splatting* [44]. Sin embargo, la robótica no está obteniendo de la explosión de la inteligencia artificial un beneficio igual de notorio.

Aún así, aprovechando los últimos avances en inteligencia artificial y como manera de paliar la problemática señalada anteriormente en el desarrollo de los sistemas de control en robótica, surge una idea por parte de diversos grupos de investigación. Dicha idea consiste en darle al propio robot las herramientas para aprender por sí mismo a completar tareas de alta complejidad, como una manipulación diestra de objetos pequeños [45]. Programar algoritmos heurísticos que atajen estas tareas conlleva una complejidad impensable, pero es posible conseguir sistemas funcionales mediante la aplicación de técnicas de aprendizaje por refuerzo, una rama de la

inteligencia artificial que, a través de la experimentación sujeta a un sistema de recompensas predefinido, permite a un sistema crear políticas que determinen sus acciones atendiendo a la información que sus sensores recaben de su entorno en todo momento.

Este enfoque revolucionario tiene el potencial de cambiar fundamentalmente la forma en que interactuamos con los robots y cómo estos se integran en diversas aplicaciones, desde la asistencia en el hogar hasta la industria y la exploración espacial. Es por ello que surge el deseo de llevar estas técnicas a un proyecto personal, en el que comprobar qué tan lejos se puede llegar con recursos limitados a nivel tanto de hardware como de software.

1.2. Objetivos

Atendiendo a los recursos disponibles, el objetivo principal de este proyecto es entrenar un agente robótico utilizando el paradigma de aprendizaje por refuerzo, como mencionamos en la sección anterior, de manera que tras el entrenamiento dicho agente sea capaz de navegar a través de un entorno desconocido con el fin de cumplir una tarea de búsqueda específica. Este entrenamiento se realizará en un entorno simulado, dado que el tiempo necesario para entrenar en un espacio físico, así como los costes derivados de los daños a los que estaría expuesto el sistema robótico, serían prohibitivos.

Para lograr la consecución de nuestro objetivo, definimos una serie de subobjetivos intermedios que deberán cumplirse a lo largo del desarrollo del proyecto, idealmente en el siguiente orden:

- Seleccionar un robot adecuado, es decir que tenga los componentes necesarios para completar con éxito las tareas propuestas tanto en simulación como fuera de ella.
- Digitalizar el robot seleccionado para poder integrar correctamente su modelo digital en el entorno virtual.
- Construir el entorno virtual en el que se llevarán a cabo los entrenamientos de nuestros agentes. Debe ser lo suficientemente realista como para representar de manera precisa las condiciones y desafíos que podría encontrar un agente en el mundo real.
- Preparar toda la lógica tras los entrenamientos. Esto incluye la parametrización de la red neuronal que determinará las acciones del agente, así como la implementación del código que rige el comportamiento del entorno de simulación y la gestión de los entrenamientos.
- Evaluar los resultados obtenidos a lo largo de los distintos entrenamientos para, tras revisar de nuevo los dos subobjetivos anteriores, realizar una iteración hacia entornos cada vez más complejos. De esta manera las políticas que obtendremos al final serán robustas, y se reducirá el choque producido por una transferencia al mundo físico, que sería la aplicación ideal de este trabajo.

1.3. Estructura de la memoria

La presente memoria, tal y como la estructuramos, se compone de cinco capítulos.

- **Capítulo 1: Introducción.** En este primer capítulo, en el cual nos encontramos, se presenta una introducción general al trabajo, comenzando por la motivación que impulsa esta investigación, para después exponer los objetivos a alcanzar. El capítulo termina con esta visión general de la estructura de la memoria.
- **Capítulo 2: Estado del Arte.** En el segundo capítulo realizamos un análisis exhaustivo del estado del arte en el campo de la robótica autónoma y el aprendizaje por refuerzo. Exploramos algunas de las tecnologías y herramientas más relevantes en el área, destacando el papel esencial del $framework$ de robótica ROS 2 (Robot Operating System 2) y presentando una selección de simuladores y algoritmos de aprendizaje por refuerzo. De esta manera proporcionamos el contexto necesario para comprender la selección de tecnologías utilizadas en el desarrollo del proyecto.
- **Capítulo 3: Desarrollo del Entorno Virtual y Parametrización del Aprendizaje.** Ya en el tercer capítulo comenzamos a entrar en materia, detallando el proceso de desarrollo del entorno virtual y la parametrización de la red neuronal. En este punto presentamos en profundidad el robot seleccionado para ser entrenado, para luego mostrar las distintas habitaciones a las que se expone a los agentes en la fase de experimentación, cada una con sus propios desafíos específicos. Además, explicamos la lógica detrás de los entrenamientos, revelando cómo se ha configurado el sistema para conseguir un aprendizaje efectivo.
- **Capítulo 4: Experimentos y Resultados.** Comenzando a cerrar la memoria, en el cuarto capítulo exponemos los distintos experimentos llevados a cabo, así como sus resultados. Aquí describimos cómo los experimentos van afectando al desarrollo del proyecto a medida que mejoramos los modelos obtenidos. Este capítulo proporciona una visión completa de los desafíos encontrados, las soluciones implementadas y las mejoras logradas a lo largo del proyecto.
- **Capítulo 5: Conclusiones.** El quinto y último capítulo contiene las conclusiones finales del proyecto, en las que vemos los logros alcanzados en relación con los objetivos planteados. Además, proponemos posibles líneas de trabajo para futuras investigaciones en este campo en constante evolución.

Capítulo 2

Estado del Arte

Para la consecución de este proyecto y el uso efectivo de sus resultados son necesarios tres pilares: un sistema operativo que controle al robot, un entorno virtual donde entrenar al robot y un conjunto de herramientas y algoritmos con los que realizar ese entrenamiento.

2.1. ROS 2

Robot Operating System 2 (ROS 2), un software ampliamente utilizado a día de hoy en el campo de la robótica, es publicado en 2022 como la segunda generación del *Robot Operating System* (ROS 1) surgido en 2009 de mano de la corporación sin fines de lucro Open Robotics. Es una plataforma de software libre para desarrollar aplicaciones robóticas, distribuida bajo la licencia Apache 2.0, que permite a los usuarios modificar, aplicar y redistribuir el software sin limitaciones [1].

Los contenidos de ROS 2 se dividen en tres categorías:

- *Middleware*: Incluyendo desde APIs (*Application Programming Interfaces*) de red hasta analizadores de mensajes, este apartado proporciona una capa de abstracción entre los diferentes componentes de ROS 2 y permite que se comuniquen y compartan información de manera eficiente. ROS 2 utiliza un enfoque de *middleware* "enchufable", lo cual da a los desarrolladores flexibilidad para elegir el *middleware* más adecuado para cada caso de uso específico. Esto permite optimizar el rendimiento, la seguridad y otros aspectos críticos del sistema para satisfacer las necesidades específicas de un proyecto.
- Algoritmos: ROS 2 provee muchos de los algoritmos utilizados con frecuencia en el área de la robótica, ofreciendo distintas librerías que implementan funciones como percepción, planificación o *SLAM* (localización y mapeado simultáneos), entre otras. También se permite la integración de algoritmos y librerías externas aprovechando la arquitectura modular de ROS 2, facilitando el uso de componentes personalizados.
- Herramientas de desarrollo: ROS 2 incluye una plétora de herramientas, tanto gráficas como por línea de comandos, para configurar, lanzar, visualizar, depurar, simular y registrar proyectos de robótica.

En resumen, ROS 2 es una plataforma de robótica altamente flexible y escalable que coge el relevo de su predecesor y continúa su rápida evolución. Con cada nueva versión, ROS 2 sigue mejorando en términos de rendimiento, estabilidad y funcionalidad, y mantiene su posición como una de las plataformas más populares en la robótica moderna.

2.2. Simuladores

Si bien la experimentación en el mundo real nos ofrece datos fiables para el desarrollo de investigaciones robóticas y/o de aprendizaje por refuerzo, esta puede alcanzar costes elevados tanto en el plano temporal como en el monetario. Es por ello que la construcción de entornos de simulación para entrenar y evaluar agentes de aprendizaje automático juega un papel crucial en la investigación. Existen distintas opciones en la actualidad que ofrecen distintos grados de semejanza con la realidad, a la vez que cubren necesidades diversas.

2.2.1. MineRL

MineRL [14] es un entorno de simulación para el aprendizaje por refuerzo basado en el popular videojuego Minecraft. Desde su presentación en 2019 como un proyecto de investigación de la Carnegie Mellon University, MineRL ha demostrado ser una herramienta muy útil para entrenar agentes de aprendizaje por refuerzo en entornos de juego complejos y dinámicos. El objetivo principal de MineRL es utilizar Minecraft como una plataforma de simulación para entrenar agentes de aprendizaje por refuerzo que puedan completar tareas complejas en el juego.

Una de las principales características de MineRL es que utiliza datos de demostración humana para entrenar a los agentes de aprendizaje por refuerzo. Esto significa que se utilizan datos recopilados de jugadores humanos que han jugado Minecraft para enseñar a los agentes de aprendizaje por refuerzo cómo completar tareas en el juego. Esto hace que MineRL sea un entorno de simulación único y muy interesante para la investigación en aprendizaje por refuerzo.

Desde su presentación, MineRL ha sido utilizado en varias investigaciones para explorar diferentes enfoques para el entrenamiento de agentes de aprendizaje por refuerzo en Minecraft. Algunos de los logros más destacados de la investigación de MineRL incluyen:

- La mejora del rendimiento de los agentes de aprendizaje por refuerzo en tareas complejas de Minecraft, como la minería y la construcción.
- El desarrollo de algoritmos de aprendizaje por refuerzo que pueden adaptarse a diferentes entornos de Minecraft y a diferentes niveles de complejidad.
- La exploración de técnicas de aprendizaje por refuerzo que utilizan la transferencia de conocimiento entre diferentes tareas y entornos de Minecraft.

Con todo esto, podemos decir que MineRL ha demostrado ser una plataforma de experimentación útil para la investigación de aprendizaje por refuerzo en entornos de juego complejos y dinámicos. Con su enfoque en el uso de datos de demostración humana y su capacidad para entrenar agentes de aprendizaje por refuerzo en diferentes

Figura 2.1: Imágenes de distintas fases de seis tareas individuales en MineRL [13]

tareas y entornos de Minecraft, MineRL ofrece una oportunidad única para explorar cómo los agentes de aprendizaje por refuerzo pueden aprender a realizar tareas complejas en juegos en línea.

2.2.2. Gymnasium

Gymnasium [15] es un entorno de simulación para el aprendizaje por refuerzo desarrollado a partir del reconocido OpenAI Gym, siendo un *fork* del mismo. Proporciona un conjunto de juegos y entornos de prueba estandarizados para la investigación de aprendizaje por refuerzo, permitiendo comparar y evaluar algoritmos de aprendizaje automático de manera objetiva y precisa.

Gymnasium proporciona una API simple y unificada para la comunicación con los entornos de simulación, lo que facilita la implementación y el desarrollo de nuevos algoritmos de aprendizaje automático. Además, el entorno es altamente personalizable, lo cual permite adaptar los escenarios de prueba para cualquier necesidad.

Uno de los aspectos más destacados de Gymnasium es su colección de juegos y entornos de prueba. Estos entornos incluyen juegos clásicos como el Ajedrez, Go y Atari, así como entornos más complejos como el RoboSumo y el simulador de físicas MuJoCo. Estos juegos y entornos de prueba son altamente desafiantes y requieren que los agentes de aprendizaje automático desarrollen habilidades avanzadas para competir y ganar.

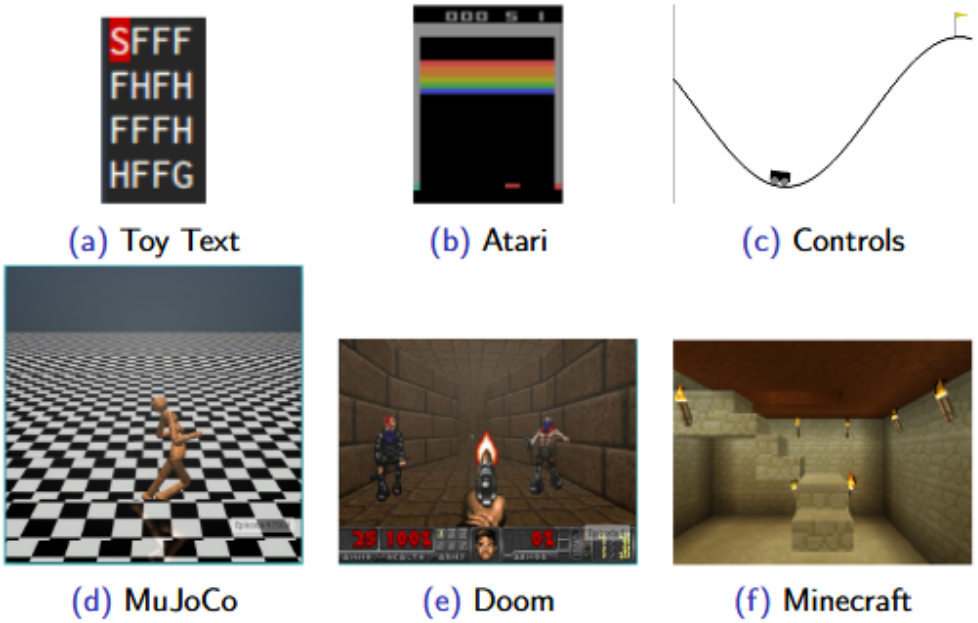


Figura 2.2: Seis dominios de ejemplo sobre los que se utiliza Gymnasium [16]

Gymnasium también tiene una gran comunidad de desarrolladores y usuarios que contribuyen al proyecto. Esto ha llevado a la creación de una amplia variedad de herramientas y bibliotecas de código abierto que se pueden utilizar para mejorar y ampliar el entorno.

En general, Gymnasium es un entorno de simulación de aprendizaje de máquina altamente sofisticado y personalizable que proporciona una plataforma estandarizada y desafiante para la evaluación y el desarrollo de algoritmos de aprendizaje automático. Con su amplia colección de juegos y entornos de prueba, Gymnasium es una herramienta valiosa para la investigación en el campo del aprendizaje por refuerzo y la inteligencia artificial en general.

2.2.3. CARLA

El simulador CARLA (*Car Learning to Act*) [17] es un entorno de simulación de conducción desarrollado por un equipo de investigación de la Universidad Autónoma de Barcelona. El objetivo principal de este simulador es proporcionar una plataforma realista y detallada para la investigación y desarrollo de sistemas de conducción autónoma. Desde su lanzamiento en 2017, se ha convertido en una herramienta muy popular en la investigación y desarrollo de vehículos autónomos.

Una de las características más destacadas de CARLA es su capacidad para simular una gran variedad de escenarios de conducción en un entorno altamente realista. La simulación incluye elementos como vehículos, peatones, semáforos, señales de tráfico y diferentes condiciones climáticas e iluminación, lo cual la convierte en una plataforma muy útil para simular diferentes situaciones de conducción y probar el comportamiento de los sistemas de conducción autónoma en diferentes condiciones.

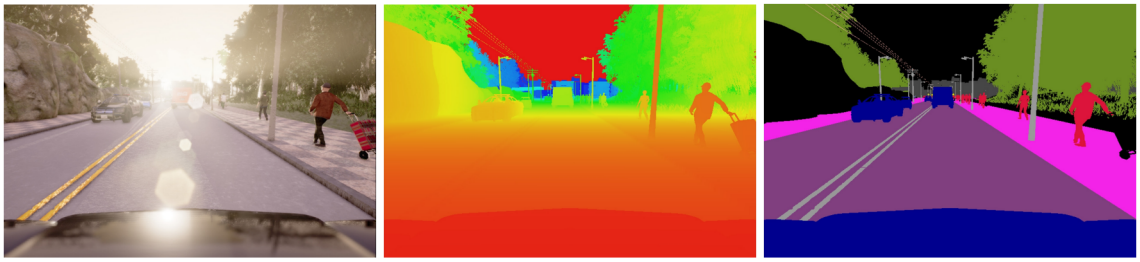


Figura 2.3: Tres de las modalidades sensoriales disponibles en CARLA [17]

Desde su lanzamiento, CARLA ha sido utilizado en diversas investigaciones para explorar diferentes enfoques para el desarrollo de sistemas de conducción autónoma. Entre sus logros más destacados se encuentran el desarrollo de sistemas de conducción autónoma que pueden navegar en situaciones complejas de tráfico urbano, como intersecciones y rotondas, la exploración de diferentes algoritmos de planificación de ruta y control de vehículo para la conducción autónoma, y la evaluación de la seguridad y eficacia de los sistemas de conducción autónoma en diferentes escenarios de conducción.

En resumen, CARLA es una plataforma de simulación de conducción autónoma muy popular y útil para la investigación y desarrollo de sistemas de conducción autónoma. Con su capacidad para simular una amplia variedad de situaciones de conducción y probar diferentes enfoques para el desarrollo de sistemas de conducción autónoma, ofrece una oportunidad única para explorar cómo los sistemas de conducción autónoma pueden ser mejorados y desarrollados en el futuro.

2.2.4. CoppeliaSim

CoppeliaSim [18] (anteriormente V-REP o *Virtual Robot Experimentation Platform*) es un entorno de simulación de robótica en 3D desarrollado por Coppelia Robotics, una empresa suiza especializada en robótica y simulación. Es utilizado en el diseño, control y programación de robots para diversas aplicaciones en la industria, la academia y la investigación. El objetivo del proyecto es proporcionar una herramienta de simulación de alta calidad para la investigación y el desarrollo de algoritmos de control y aprendizaje por refuerzo en el campo de la robótica.

Este entorno ofrece una interfaz gráfica de usuario que permite a los usuarios crear y diseñar entornos de simulación de robots personalizados. Con CoppeliaSim, los usuarios pueden simular tanto robots terrestres como aéreos, así como también pueden integrar sensores, actuadores y otros componentes que se utilizan en los robots reales. Además, CoppeliaSim también cuenta con un motor de física altamente preciso, lo que permite a los usuarios simular de manera realista el comportamiento de los robots en diferentes entornos.

CoppeliaSim también proporciona una API (Interfaz de Programación de Aplicaciones) que permite a los usuarios programar el comportamiento de los robots en la simulación utilizando varios lenguajes de programación como C++, Python y Matlab. Esto permite a los usuarios probar diferentes algoritmos de control y planificación de movimiento en un entorno simulado antes de implementarlos en robots reales.

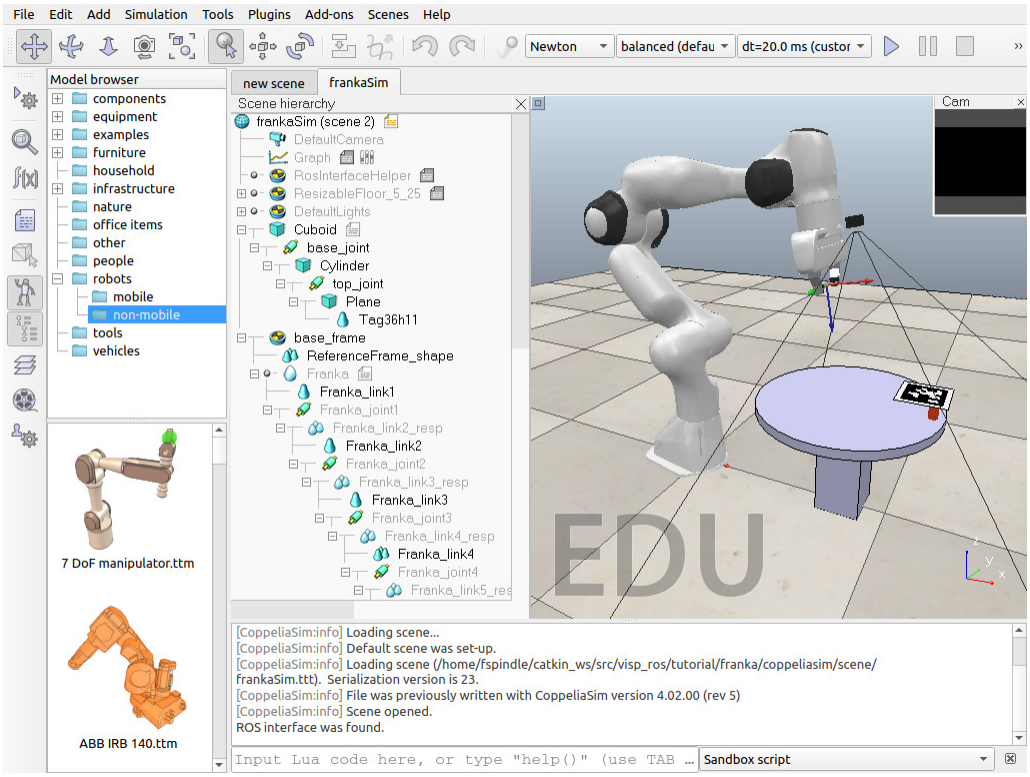


Figura 2.4: Interfaz gráfica de CoppeliaSim [19]

Desde su publicación en 2013 ha sido utilizado en una amplia variedad de aplicaciones en la industria, la academia y la investigación. Algunos de los usos más destacados incluyen el diseño y la programación de robots para la fabricación avanzada, la automatización de procesos industriales, la robótica móvil, la navegación autónoma y la robótica médica. Además, CoppeliaSim ha sido utilizado en competencias internacionales de robótica para simular robots y validar el rendimiento de los equipos de competición.

Resumiendo, CoppeliaSim es una plataforma de simulación de robots altamente personalizable y flexible que se utiliza en diversas aplicaciones en la industria, la academia y la investigación. Su interfaz gráfica de usuario fácil de usar, su motor de física preciso y su API programable permiten a los usuarios simular y probar diferentes enfoques de control y planificación de movimiento para robots en un entorno simulado antes de implementarlos en robots reales.

2.2.5. Gazebo

Gazebo es un simulador de robótica de código abierto publicado por el laboratorio de investigación robótica de la Universidad de California del Sur en 2004. Desde entonces ha sido ampliamente utilizado en la investigación de robótica, habiéndose realizado numerosos avances en este campo en los últimos años que han involucrado su uso.

Su desarrollo surge de la necesidad de reproducir con precisión las interacciones que pueda tener uno o varios robots tanto con un entorno de exterior como entre ellos mismos. [2] Esto lo consigue gracias a la integración de diversos motores de

físicas (siendo *Open Dynamics Engine* el utilizado por defecto), renderizado mediante OpenGL de entornos realistas, y código de soporte para la correcta simulación de los sensores y actuadores.

Uno de los avances más significativos en el uso de Gazebo ha sido la integración de la inteligencia artificial y el aprendizaje automático en la simulación de robótica. Esto ha permitido la creación de robots autónomos y sistemas de control robótico más sofisticados.

En términos de modelado, esta plataforma es capaz de simular objetos con diversos atributos físicos a la vez que puede representar a los robots correctamente como estructuras dinámicas compuestas de cuerpos rígidos conectados por articulaciones, y ha mejorado significativamente en la simulación de objetos deformables y en la integración de sensores de alta fidelidad como cámaras RGBD y LIDAR.

Otro avance importante en Gazebo ha sido su capacidad de simular sistemas robóticos complejos, como robots humanoides y robots enjambre. Esto ha permitido la investigación en áreas como la coordinación y la cooperación de robots en equipos.

En cuanto a la integración con otros software de robótica, Gazebo ha mejorado la integración con herramientas de diseño de robots como ROS (Robot Operating System) y MATLAB, lo que facilita el diseño, la simulación y la implementación de sistemas robóticos complejos.

En resumen, Gazebo ha avanzado significativamente en los últimos años en términos de simulación de robótica y se ha convertido en una herramienta esencial para la investigación de robótica. Su capacidad de simular sistemas robóticos complejos, integrarse con otras herramientas de robótica y permitir la integración de la inteligencia artificial y el aprendizaje automático la convierten en una herramienta poderosa para la investigación y el desarrollo de sistemas robóticos avanzados.

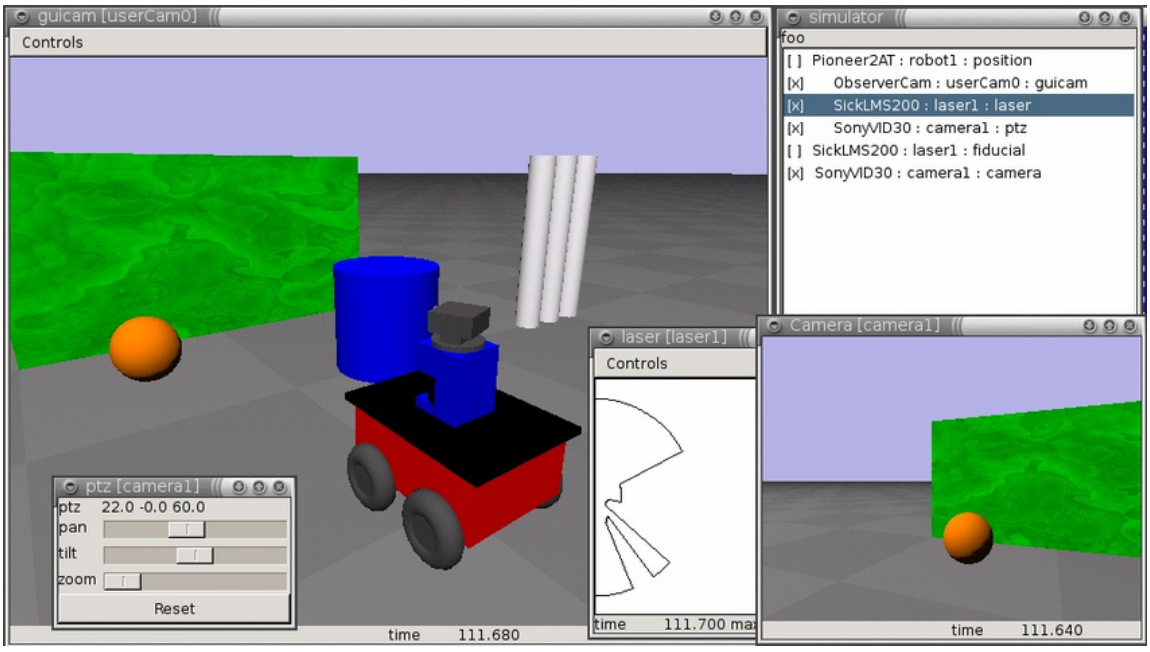


Figura 2.5: Interfaz gráfica de Gazebo [20]

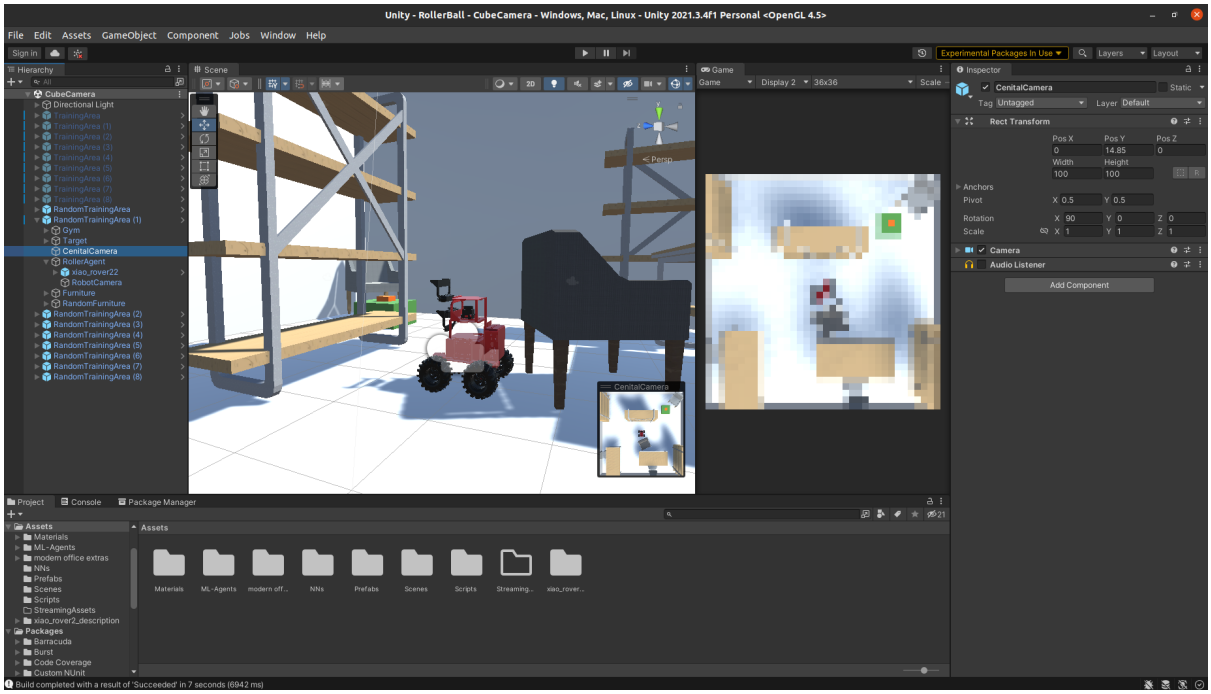


Figura 2.6: Interfaz gráfica de Unity (captura del proyecto)

2.2.6. Unity3D

Unity es un motor de videojuegos multiplataforma desarrollado por Unity Technologies. Desde su lanzamiento en 2005 ha recibido innumerables actualizaciones, las cuales siempre han asegurado la estabilidad del motor y su mantenimiento a la vanguardia del sector. Su facilidad de aprendizaje y uso lo hacen una herramienta muy popular entre la comunidad de los desarrolladores de videojuegos, aunque no son el único sector que la utiliza. Industrias como la cinematográfica, la automovilística, la arquitectónica, la ingeniera o incluso las Fuerzas Armadas de los Estados Unidos [3] han mostrado su interés por esta herramienta, y es que tiene un gran potencial para la creación de simulaciones interactivas, así como otras experiencias. Siendo conscientes de esto, Unity Technologies apoya distintos proyectos más allá de la línea de los videojuegos, como el *Unity Robotics Hub* [4] para simulación robótica, el paquete de herramientas de aprendizaje automático ML-Agents [5], o el paquete de herramientas *Unity Perception Package*, que permite generar bases de datos sintéticos a gran escala para entrenar y validar modelos de visión por computador [6].

En términos de simulación robótica, Unity cuenta con varias características que lo convierten en una opción interesante para la creación de ambientes simulados para robots. Para empezar posee un motor de físicas muy avanzado, Nvidia PhysX, que permite simular el comportamiento de objetos y materiales de manera muy realista, facilitando un modelado preciso y fiable de las interacciones entre un robot y su entorno. También dispone de un potente motor de renderizado tridimensional, con el que es posible crear ambientes fieles a la realidad, algo fundamental para la simulación robótica. Para aprovechar esa capacidad de renderizado, Unity también dispone de diversas bibliotecas de modelos 3D, facilitando la creación de simulaciones detalladas. Por su parte, el previamente mencionado paquete de herramientas ML-Agents abre la posibilidad de integrar algoritmos de aprendizaje automático en las simu-

laciones robóticas, aunque se hablará de ello con mas exhaustividad en apartados posteriores.

En conclusión, Unity3D es una herramienta muy útil para la simulación robótica gracias a sus potentes capacidades de físicas y renderizado 3D, su amplia variedad de bibliotecas de modelos 3D y su soporte para ML-Agents. Esto hace que sea una herramienta capaz de competir con otras opciones más especializadas, ganando popularidad entre los desarrolladores de robots que buscan crear simulaciones robóticas realistas y efectivas.

2.2.6.1. ML-Agents

ML-Agents es un kit de herramientas y recursos de código abierto desarollado por Unity Technologies que permite integrar y utilizar técnicas de aprendizaje automático para entrenar agentes inteligentes en entornos virtuales creados con su plataforma propietaria Unity3D. Cuenta con una amplia documentación, así como ejemplos y tutoriales que permiten a los desarrolladores aprender a manejar este kit de herramientas de manera eficaz en sus proyectos desde un nivel de entrada bajo [9].º

Si bien en su versión beta se utilizó TensorFlow como la librería de aprendizaje automático por defecto, actualmente en su versión 2.0 hace uso de PyTorch, aunque sigue aprovechando funcionalidades como TensorBoard para el visualizado de datos. ML-Agents incluye implementaciones estándar de varios algoritmos de entrenamiento por refuerzo presentes en el estado del arte, que cumplen distintas funciones: *Proximal Policy Optimization* (PPO), un algoritmo de propósito general, *Soft Actor Critic* (SAC), especializado en el entrenamiento de robótica, y *Generative Adversarial Imitation Learning* (GAIL), pensado para el aprendizaje por imitación.

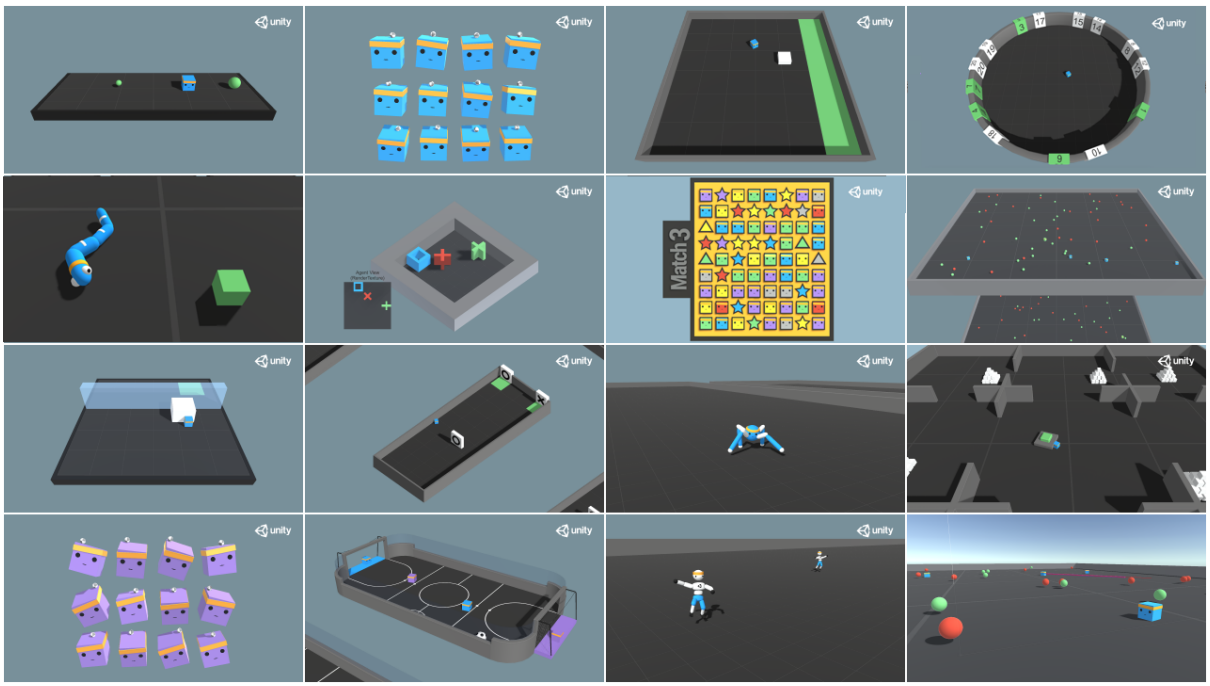


Figura 2.7: Diversos entornos de ejemplo incluidos en ML-Agents [24]

En resumen, ML-Agents es un kit de herramientas de aprendizaje por refuerzo de código abierto que permite entrenar agentes inteligentes en entornos virtuales creados con la plataforma Unity3D. La versión actual de ML-Agents hace uso de PyTorch, y permite visualizar los datos resultantes con TensorBoard. La herramienta cuenta con una amplia documentación, ejemplos y tutoriales que facilitan su uso, e incluye implementaciones estándar de varios algoritmos de entrenamiento por refuerzo, como PPO, SAC y GAIL.

2.2.7. Nvidia Omniverse

Nvidia Omniverse es una plataforma de colaboración en tiempo real que permite a los equipos de diseño y desarrollo de diferentes disciplinas trabajar juntos en un mismo entorno 3D con una amplia gama de herramientas y recursos que les permiten crear y modificar contenido en 3D de alta calidad aprovechando la tecnología de trazado de rayos Nvidia RTX [7].

Su componente más atractivo para este proyecto es *Isaac Gym*, una plataforma de simulación robótica de punto a punto de alto rendimiento centrada en el aprovechamiento de la computación por GPU para evitar los cuellos de botella que generan la mayoría de motores de físicas al depender de computación por CPU. Al realizar absolutamente todas las computaciones necesarias sin salir de la GPU, consigue aumentar el rendimiento respecto a otras plataformas al doble o incluso al triple, por no hablar del enorme potencial de paralelización [8]. Esta plataforma incluye una implementación básica del algoritmo de aprendizaje por refuerzo *Proximal Policy Optimization* (PPO) y utiliza PyTorch como librería de aprendizaje automático por defecto, pero se puede sustituir por TensorFlow, así como implementar otros algoritmos. Actualmente podemos afirmar que, desde su lanzamiento en 2021, se ha hecho rápidamente un hueco en la industria, habiendo sido utilizado con fines industriales por compañías como Pepsi o Amazon.

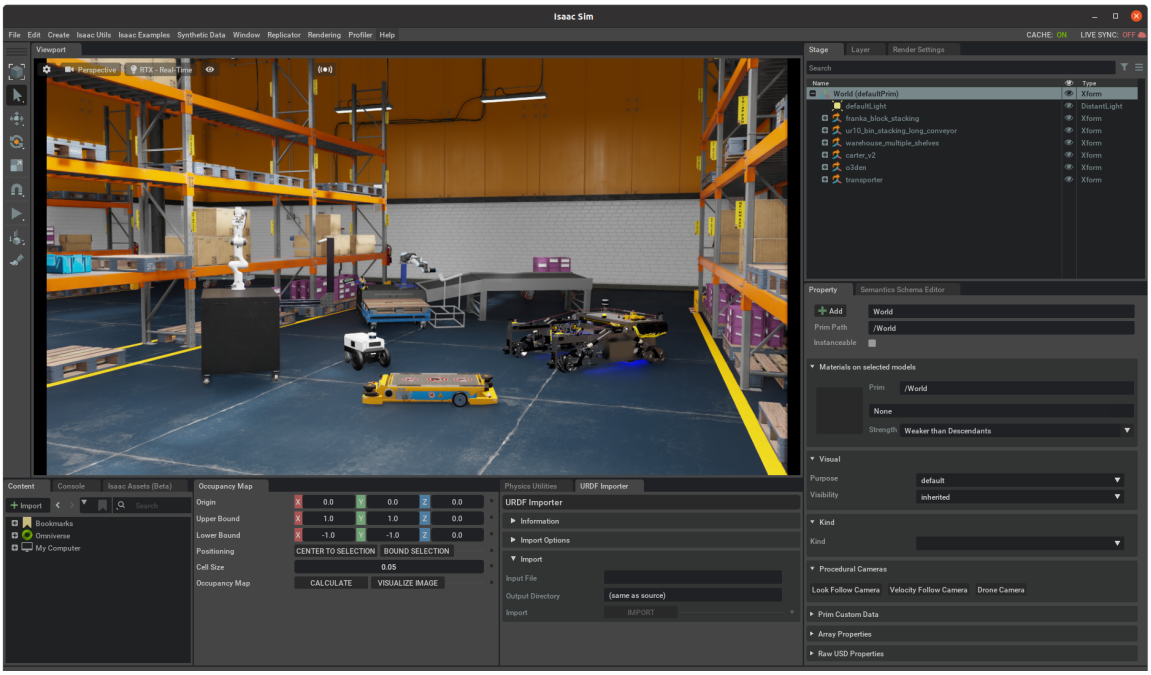


Figura 2.8: Interfaz gráfica del Isaac Gym [21]

En definitiva, Omniverse es una plataforma de colaboración en tiempo real que permite a los equipos de diseño y desarrollo trabajar juntos en un entorno 3D con una amplia gama de herramientas y recursos. *Isaac Gym*, su herramienta más atractiva para este proyecto, es una plataforma de simulación robótica de alto rendimiento que utiliza la computación por GPU para aumentar el rendimiento y evitar los cuellos de botella de los motores de física. A día de hoy puede considerarse una opción muy competitiva para desarrollo de robótica y diseño de simulaciones precisas y escalables.

2.3. Algoritmos de aprendizaje por refuerzo

Un agente robótico puede ser programado con métodos heurísticos que definan claramente qué acciones debe tomar en cada situación, pero en un contexto moderno y cambiante esta opción muchas veces no es realista. Lo ideal es que un agente pueda adaptarse a diversas situaciones, incluso algunas no previstas por sus programadores. Esto se puede conseguir mediante la implementación de algoritmos de aprendizaje por refuerzo, que a través de un entrenamiento del agente den lugar a una política estocástica que le permita comprender su entorno, así como entender qué acciones realizar para cumplir con sus objetivos según el estado en el que se encuentre su entorno.

En este trabajo revisamos los tres algoritmos mencionados previamente en el apartado de ML-Agents: Proximal Policy Optimization, Soft Actor Critic y Generative Adversarial Imitation Learning.

2.3.1. Proximal Policy Optimization

Al hablar de PPO podemos referirnos a distintos algoritmos, y es que existen diversas variantes de este método de optimización de políticas, pero normalmente, como es el caso de este trabajo, se habla de la variante de *clipping*. Cabe mencionar que puede oírse hablar de "PPO2", pero no es sino una adaptación del algoritmo para optimizar su computación en unidades de procesado gráfico.

En su núcleo, PPO es un algoritmo de aprendizaje por refuerzo *on-policy* que utiliza múltiples *epochs* de ascenso de gradiente estocástico para efectuar cada actualización de política, con el objetivo de mejorar la capacidad de un agente para tomar decisiones en entornos complejos. Nace como una mejora del algoritmo del mismo autor *Trust-Region Policy Optimization* (TRPO) [22], que a su vez elaboraba sobre los gradientes naturales de políticas [23]. Manteniendo la estabilidad y fiabilidad del TRPO, descarta algunas de las restricciones teóricas de su predecesor con el objetivo de hacerlo más práctico. De esta manera ofrece una implementación sencilla, pues solo se requieren de pequeños ajustes a un gradiente de políticas estándar, a la vez que mejora el rendimiento, permitiendo el entrenamiento con redes neuronales profundas de miles de parámetros. Otro gran punto a favor de la simplificación que realiza sobre el trabajo del TRPO es que, al abandonar el uso de gradientes de segundo orden, permite la utilización de optimizadores como Adam [27], que mejoran con creces los tiempos de entrenamiento,

Se basa en la idea de que una política óptima debe maximizar la recompensa esperada del agente en cada paso. Para llevar a cabo esto, PPO utiliza una función de pérdida que tiene en cuenta tanto la recompensa obtenida por el agente en cada paso como la diferencia entre la política actual del agente y la política anterior, o en otras palabras, la divergencia en la variedad estadística de ambas políticas. Además, la variante más utilizada de PPO utiliza una técnica de regularización llamada *clipping* de probabilidad que evita que la política del agente se aleje demasiado de la política anterior, a menos que sea necesario para recuperarse de una mala actualización, lo que mejora la estabilidad del entrenamiento, a la vez que permite la reutilización de muestras ya computadas, lo que mejora el tiempo de cómputo [10].

Resumiendo, PPO es un algoritmo de optimización de políticas que utiliza múltiples *epochs* de ascenso de gradiente estocástico para mejorar la capacidad de un agente para tomar decisiones en entornos complejos. PPO nace como una mejora del algoritmo TRPO, descartando algunas de las restricciones teóricas de su predecesor para hacerlo más práctico e implementar optimizadores como Adam. PPO utiliza una función de pérdida que tiene en cuenta tanto la recompensa obtenida por el agente como la diferencia entre la política actual y la anterior. Además, la variante más utilizada de PPO utiliza una técnica de regularización llamada *clipping* de probabilidad que mejora la estabilidad del entrenamiento y el tiempo de cómputo al permitir la reutilización de muestras ya computadas.

2.3.2. Soft Actor Critic

Si bien pueden alcanzarse buenos resultados con algoritmos como PPO, el entrenamiento de sistemas robóticos requiere de métodos que sean delicados con el equipo, sobre todo si se piensa realizar entrenamientos en entornos reales. De esta problemática nace el algoritmo Soft Actor Critic, publicado en conjunto por la Universidad de Berkeley y Google en 2018 [25].

El entrenamiento en entornos reales con equipamiento costoso conlleva la necesidad de conseguir eficiencia en el uso de muestras, no utilizar hiperparámetros altamente sensibles, emplear un aprendizaje *off-policy* y suavizar las acciones realizadas. Todos esto se traduce en una minimización del número de episodios necesarios para alcanzar buenos resultados, lo cual conlleva un ahorro temporal y ayuda a preservar en buen estado el equipamiento utilizado.

De esta manera, SAC se plantea como un algoritmo de aprendizaje profundo por refuerzo, sin modelo y *off-policy*, basado en el marco del aprendizaje por refuerzo de máxima entropía [26]. Si bien comparte con PPO la búsqueda de la maximización de su expectativa de recompensa, no trata de disminuir la divergencia entre políticas al actualizarse. En vez de eso, SAC busca maximizar la expectativa de su propia entropía, haciendo que pueda verse a la función objetivo como una maximización de la recompensa esperada limitada por la entropía. Así se logra crear un balance entre exploración y explotación que dota a los agentes de la capacidad de aprender más de una forma de hacer las cosas.

Tal y como puede verse en la figura 2.9, este planteamiento demuestra ser altamente efectivo al compararlo con otros métodos actuales, pues genera comportamientos que se defienden relativamente bien incluso en los peores casos.

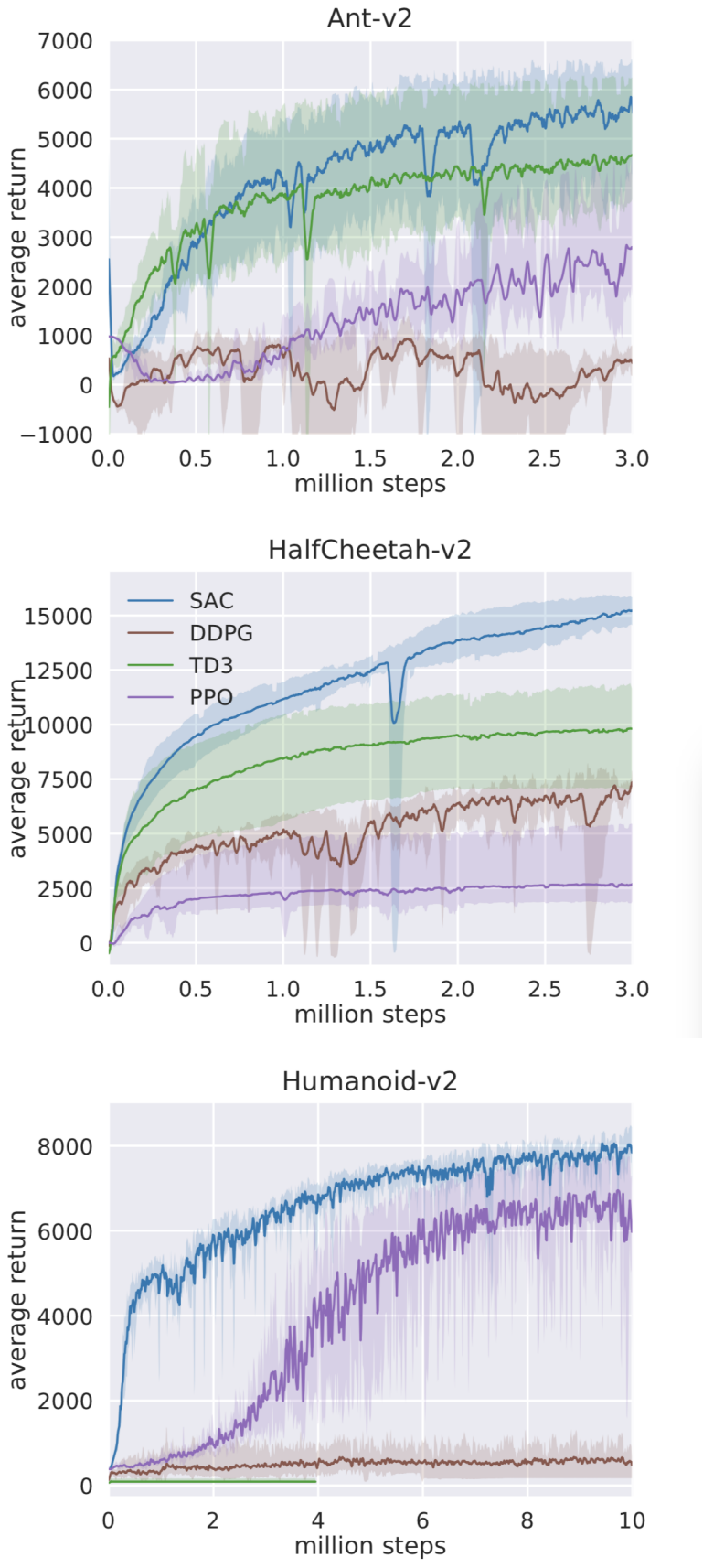


Figura 2.9: Comparación de distintas pruebas del algoritmo SAC contra otros algoritmos modernos realizadas en OpenAI Gym [25].

En síntesis, el entrenamiento de sistemas robóticos en entornos reales requiere de métodos cuidadosos con el equipo, lo cual ha llevado a la creación del algoritmo Soft Actor Critic (SAC) por la Universidad de Berkeley y Google. SAC es un algoritmo de aprendizaje profundo por refuerzo, sin modelo y off-policy, que maximiza la entropía para crear un balance entre exploración y explotación. Los resultados muestran que SAC es altamente efectivo y funciona bien incluso en los peores casos.

2.3.3. Generative Adversarial Imitation Learning

El Generative Adversarial Imitation Learning (GAIL) es un método de aprendizaje por imitación que utiliza redes generativas adversarias (GANs) para aprender políticas de comportamiento en entornos complejos a partir de la observación de expertos. Fue presentado en la conferencia NIPS (Neural Information Processing Systems) de 2016 en Barcelona por OpenAI y la Universidad de Stanford, y su objetivo es generar políticas de comportamiento similares a las de los expertos, utilizando técnicas de aprendizaje por refuerzo inverso para entrenar un discriminador que diferencie entre las políticas del experto y del aprendiz [28].

GAIL ha sido utilizado en diversas aplicaciones, incluyendo la robótica y la conducción autónoma. En robótica, se ha utilizado para aprender tareas como el movimiento de un brazo robótico, la navegación en un entorno desconocido y la manipulación de objetos. En conducción autónoma, se ha utilizado para aprender a navegar entornos urbanos.

Aunque GAIL ha demostrado ser una técnica efectiva en varias aplicaciones, aún existen desafíos que deben ser abordados. Uno de los desafíos principales es el sesgo de distribución, ya que la política generada por GAIL puede no ser suficientemente general para funcionar en entornos diferentes a los utilizados durante el entrenamiento. Además, GAIL puede ser computacionalmente costoso debido a la necesidad de entrenar tanto el generador como el discriminador.

A pesar de estos desafíos, GAIL es una técnica prometedora para el aprendizaje de políticas de comportamiento en entornos complejos y puede ser mejorado con técnicas como la regularización y la transferencia de conocimiento.

En suma, GAIL es una técnica efectiva de aprendizaje por imitación que tiene el potencial de ser aplicada en diversas áreas, pero aún requiere de más investigación para superar los desafíos presentes.

2.4. Tecnologías seleccionadas

Tras el estudio de las distintas posibilidades a nuestra disposición, seleccionamos las tecnologías sobre las que construir el proyecto.

En lo referente al control del robot si se llevara el modelo al mundo físico, la única decisión a tomar es si utilizar ROS o ROS 2. Nos decantamos por ROS 2 por las claras mejoras que presenta sobre su antecesor, que han llevado a la comunidad internacional a adoptarlo rápidamente. El resultado de este proyecto aprovecharía principalmente su apartado de *middleware*, utilizando ROS 2 para controlar el robot utilizando un modelo obtenido a través del entrenamiento en simulación, pero sus herramientas de desarrollo también serán de gran utilidad.

En cuanto a la plataforma de simulación, optamos por utilizar Unity3D junto al kit de herramientas ML-Agents dado el amplio rango de posibilidades que esta combinación ofrece, además de la experiencia previa del alumno con ambas tecnologías. La única duda en este apartado surge ante la posibilidad de utilizar el Isaac Gym de Nvidia Omniverse, plataforma claramente superior a la que se consiguió acceso, pero la carencia de una tarjeta gráfica Nvidia con tecnología RTX obligó a descartar esta posibilidad.

Finalmente, en lo que a algoritmos de aprendizaje automático respecta, en un principio se acordó realizar pruebas tanto con PPO como con SAC, pero tras profundizar un poco en la teoría tras la tecnología y ejecutar algunos simulacros, SAC demostró rápidamente ser una elección mejor para este proyecto, puesto que está diseñado específicamente para ser usado dentro del marco de la robótica.

Capítulo 3

Desarrollo del Entorno Virtual y Parametrización del Aprendizaje

En este capítulo exploraremos en detalle las diferentes etapas del proyecto, sin entrar en los resultados que se van obteniendo a lo largo del mismo, puesto que son abordados con profundidad en el siguiente capítulo. Comenzaremos examinando la comprobación y preparación del robot objetivo, realizando un análisis de dos robots: el Pioneer 2DX y el Xiao Rover 2. A continuación, nos adentraremos en la preparación del entorno simulado, describiendo cómo se creó un entorno básico para las primeras pruebas a partir del cual iteramos hacia configuraciones más complejas. Posteriormente, discutimos la definición de los parámetros de la red y los sensores del agente, para terminar abordando la función de recompensa con la que medimos el desempeño obtenido a lo largo de los entrenamientos.

3.1. Comprobación y preparación del robot objetivo

Dado que para poder realizar un entrenamiento efectivo del robot es necesario estar familiarizado con su estructura y funcionamiento, se priorizó conocer el hardware con el que se iba a trabajar. Este sería en un principio un robot Pioneer 2DX propiedad del departamento de robótica de la UPV, pero tras una serie de dificultades técnicas devenidas de la edad del equipo, se acabó haciendo uso de un robot personalizado apodado Xiao Rover 2, propiedad del profesor e investigador Jaime Rincón.

3.1.1. Pioneer 2DX

El Pioneer 2DX es un robot móvil terrestre de navegación diferencial con dos ruedas motorizadas y una rueda frontal de soporte. La presencia de un sensor de choque en la parte frontal así como un conjunto de sensores de ultrasonido lo convertían en una opción atractiva para el proyecto, pero al intentar hacer pruebas con el mismo no tardaron en aparecer problemas.

Figura 3.1: Pruebas de navegación en laboratorio con un Pioneer 2DX [30].

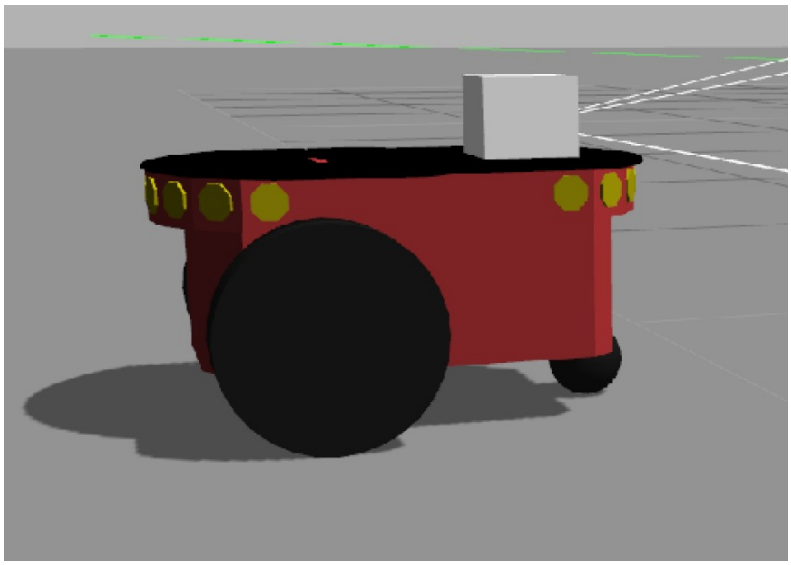


Figura 3.2: Pioneer 2DX en simulación [31].

Es habitual encontrar proyectos de iniciación con Gazebo que utilizan modelos digitales del Pioneer 2DX como el de la figura 3.2, o alguno de sus sucesores, para resolver tareas de una sencillez relativa a nivel de simulación, pero no hemos podido ubicar investigaciones actuales que trabajen a nivel físico con el robot. Es por ello que consideramos imperante la comprobación del hardware antes de avanzar con el proyecto. Para la operación del robot se trató de instalar ROS en una Raspberry Pi 3, pero una cascada de dependencias sin resolver demostró la incompatibilidad de esta placa con nuestro proyecto. Antes de reintentar la configuración en una Raspberry Pi 4, más compatible con ROS, decidimos conectar el Pioneer 2DX directamente a un ordenador para comprobar el estado de su sistema base. Ahí descubrimos que los controladores del robot se encuentran completamente desfasados, y por tanto este ya no es capaz de operar correctamente. Al buscar su manual de operaciones, puede verse que fue publicado por ActivMedia Robotics en 2001 [29]. En un campo como la robótica, que ha avanzado tanto en los últimos años, no es de extrañar que una tecnología con más de dos décadas de uso a sus espaldas quede fuera de juego. Ante esta evidencia, nos vimos en la necesidad de buscar otro robot en el que centrar nuestros esfuerzos.

3.1.2. Xiao Rover 2

Después de enfrentarnos a las diversas dificultades que surgieron al intentar utilizar el Pioneer 2DX para este proyecto, nos vimos en la necesidad de cambiar de enfoque y dirigir nuestro desarrollo hacia un segundo robot. La elección recayó en el Xiao Rover 2, un robot móvil modular diseñado específicamente para la navegación terrestre. Este robot se caracteriza por contar con piezas impresas en 3D, lo que le confiere una mayor flexibilidad y versatilidad en su estructura.

A diferencia del Pioneer 2DX, el Xiao Rover 2 presenta una configuración de cuatro ruedas motorizadas, lo que le proporciona una mayor estabilidad y capacidad de movimiento en diferentes tipos de terreno. Además, este robot viene equipado con un pequeño ordenador que utiliza el sistema operativo Ubuntu, al cual es posible acceder de manera remota a través de SSH. Esta característica nos brinda la posibilidad de controlar y configurar el robot de manera más eficiente, permitiéndonos realizar ajustes y ejecutar comandos de forma remota sin necesidad de estar físicamente conectados al robot.

3.1.2.1. Robot físico

El Xiao Rover 2 presenta un diseño modular, el cual permite aprovechar al máximo su naturaleza como robot impreso en 3D. Tal y como puede apreciarse en la figura 3.3, el núcleo central del robot está conformado por un módulo principal que alberga su batería, da soporte sobre él a la placa base que conecta todos los componentes del robot, y aguanta en la parte trasera al ordenador encargado de su gestión. Este ordenador cuenta con una pequeña pantalla táctil integrada, convenientemente anclada a la pared trasera de la bahía que aloja el ordenador, y gracias a la cual es posible verificar el estado del robot fácilmente sin necesidad de utilizar equipos externos. Sin embargo, para un uso físico del robot, se recomienda realizar las operaciones a través de SSH desde otro dispositivo.

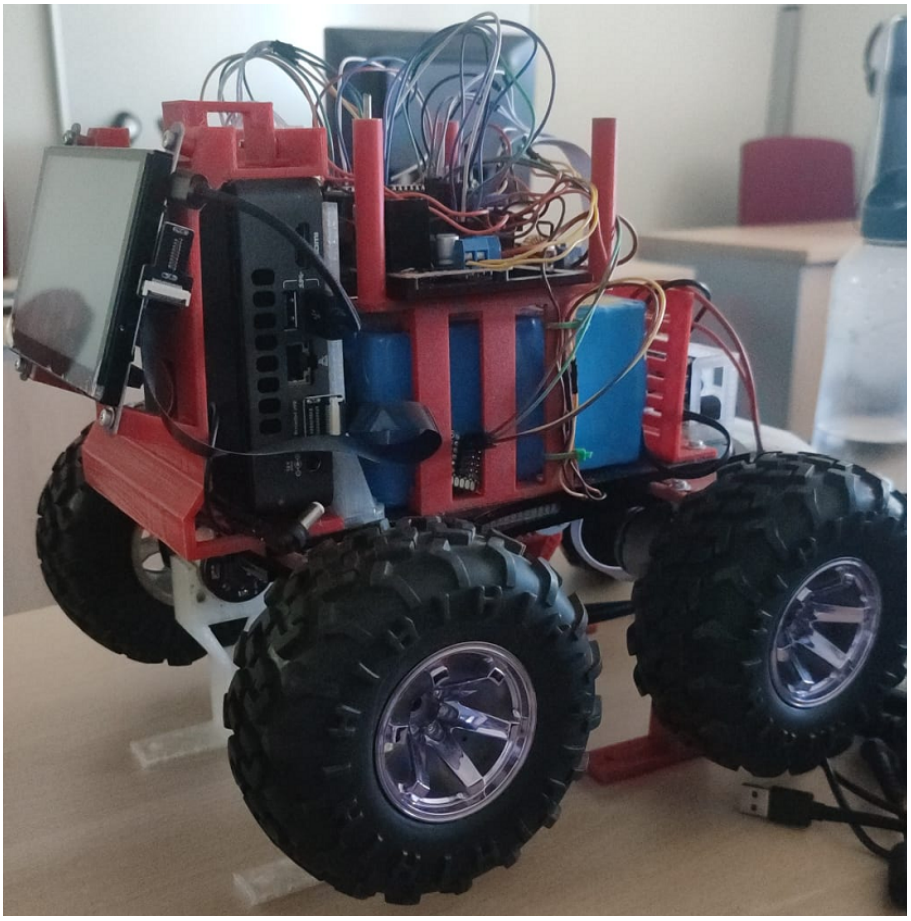


Figura 3.3: Xiao Rover 2 durante una revisión de sus componentes.

Justo debajo del módulo principal se encuentran los motores que impulsan las cuatro ruedas de este robot, y en la parte superior del robot existe la posibilidad de instalar un módulo adicional que albergue un sensor LiDAR. Sin embargo, en el contexto de nuestro proyecto, hemos optado por utilizar un módulo frontal diseñado para sostener una *webcam*. Esta cámara desempeñará un papel fundamental como sensor único para la red neuronal que regirá las acciones del robot, proporcionando información visual que permita al robot tomar decisiones y ejecutar tareas específicas de forma inteligente.

3.1.2.2. Modelo digital

El modelo digital del Xiao Rover 2 utilizado en nuestras simulaciones no es una réplica exacta del robot original, pero representa una aproximación suficiente para cumplir nuestros objetivos durante los entrenamientos. Para lograr esto, utilizamos ROS 2 en su distribución Galactic y creamos un archivo URDF, es decir en formato unificado de descripción robótica. Este archivo es una especificación XML para describir la configuración física completa de un robot, estableciendo la relación entre todas sus articulaciones y elementos rígidos [38]. Esto lo obtenemos a partir de los archivos ".xacro", o XML macros, que contienen la configuración de las diferentes partes del robot de una manera corta y legible mediante el uso de macros que pueden expandirse para formar expresiones XML complejas [37].

Ya con este archivo de configuración compilado, reconstruimos el robot en Unity utilizando un plugin para importar archivos URDF [11]. Sin embargo, nos encontramos con un desafío inicial relacionado con la compatibilidad de los modelos de las piezas, ya que estaban en formato ".stl", el cual no es soportado directamente por Unity. Para solucionar este inconveniente, convertimos los modelos de cada pieza al formato ".obj" y actualizamos sus referencias en los archivos de configuración ".xacro". Una vez realizada esta conversión, finalmente logramos importar todos los modelos 3D de los componentes individuales del robot en Unity, colocándolos en la escena de manera que conforman un esqueleto funcional del robot en el entorno virtual.

Este proceso nos permitió recrear con precisión las características físicas y estructurales del Xiao Rover 2 en un entorno virtual, lo cual resulta fundamental para llevar a cabo las simulaciones y entrenamientos requeridos para nuestro proyecto. Aunque el modelo no es una copia exacta, su aproximación suficiente, como puede verse en la Figura 3.4, nos brinda las herramientas necesarias para alcanzar nuestros objetivos de manera efectiva y eficiente en el entorno simulado.

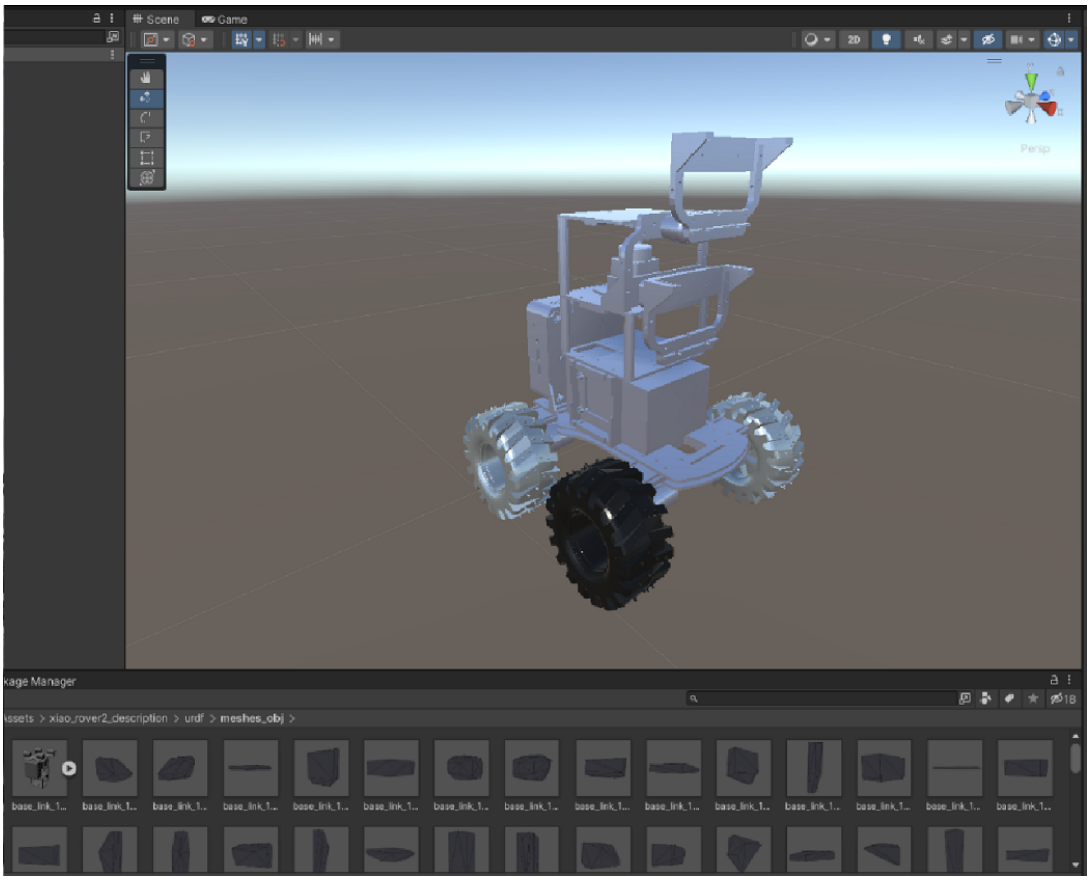


Figura 3.4: Modelo del Xiao Rover 2 importado a Unity3D, con algunos de sus componentes a la vista en la lista de archivos.

3.2. Preparación del entorno simulado

Dado que queremos entrenar al modelo digital de nuestro robot para que pueda funcionar correctamente en el mundo real, es crucial establecer una base sólida en el simulador de manera que el cambio de entorno suponga una transición lo menos problemática posible.

En primer lugar, preparamos un entorno básico en el simulador, que consiste en una plataforma blanca de cinco metros de ancho por cinco de largo que serviría más adelante como base de todos los escenarios, de manera similar a la habitación que se ve en la figura 3.5, pero sin paredes. En este entorno, utilizamos un cubo equipado con una cámara, que representa de manera abstracta a nuestro robot. El objetivo inicial es que el cubo pueda localizar y dirigirse hacia otro cubo de color naranja ubicado en un punto aleatorio de la habitación. Con el fin de acelerar los entrenamientos, creamos nueve de estas plataformas con sus respectivos agentes individuales, permitiendo así realizar múltiples pruebas simultáneamente en un solo entrenamiento. Sin embargo, dado que en el mundo real el robot no debería enfrentar el problema de caer al vacío y considerando que la disposición de las plataformas puede generar una superposición de información visual entre los agentes, transformamos las plataformas en cubículos.

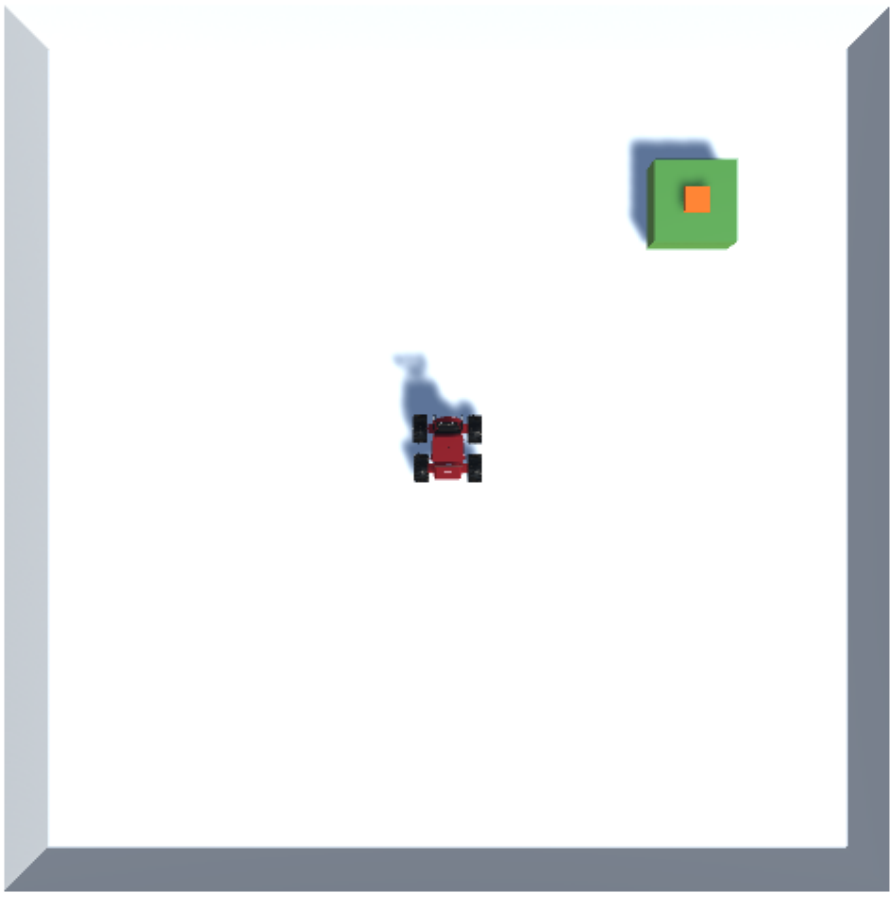


Figura 3.5: Entorno de entrenamiento con cámara cenital y habitación vacía.

Una vez que los primeros entrenamientos demuestran ser exitosos, comenzamos a realizar un avance progresivo hacia un entorno más complejo. Importamos el modelo del robot al simulador, aunque físicamente sigue siendo representado por un cubo. Puede verse en la figura 3.5 como el agente de color rojo que se encuentra en el centro de la habitación. El objetivo ahora es dirigirse hacia un cubo verde con un pequeño cubo naranja en la parte superior, de manera que sea una forma fácil de identificar. También realizamos pruebas utilizando tanto una cámara montada en el agente como una vista cenital fija.

A partir de ahí, realizamos un primer aumento de la complejidad visual sin modificar la física, decorando las paredes con fotos de habitaciones amuebladas reales como puede verse en la figura 3.6. Esto introduce un cierto grado de ruido visual que nos sirve para comprobar si los agentes siguen detectando correctamente su objetivo o se requieren ajustes en el sensor de la cámara montada en el robot.

A continuación, restablecemos el color blanco en las paredes y decoramos la habitación con mobiliario típico de oficina de forma estática, tal y como se ilustra en la figura 3.7. Se ha elegido este estilo de mobiliario dado que es algo que el agente podría encontrarse con facilidad en una ejecución en el mundo físico. Utilizando un enfoque de aprendizaje curricular, el mobiliario se va agregando gradualmente en sus ubicaciones predeterminadas a medida que el agente avanza en un entrenamiento y refina sus políticas para obtener una mayor recompensa.

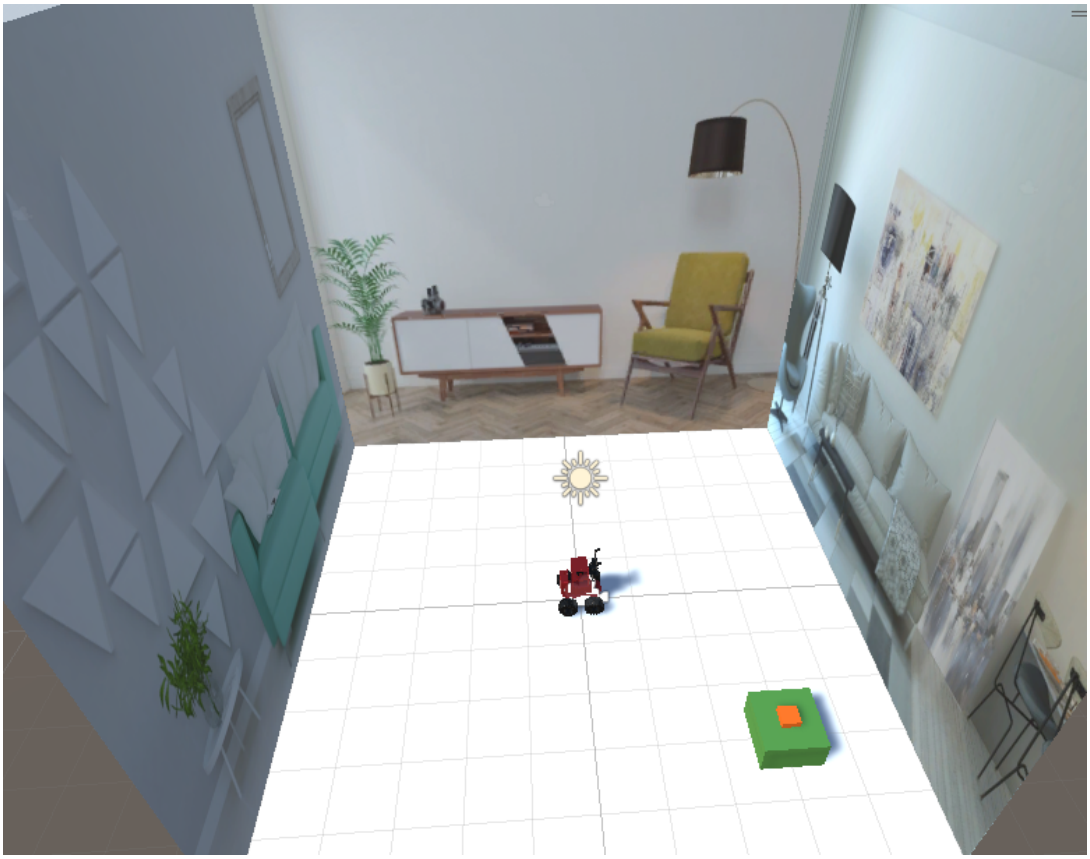


Figura 3.6: Entorno de entrenamiento con habitación con ruido visual.

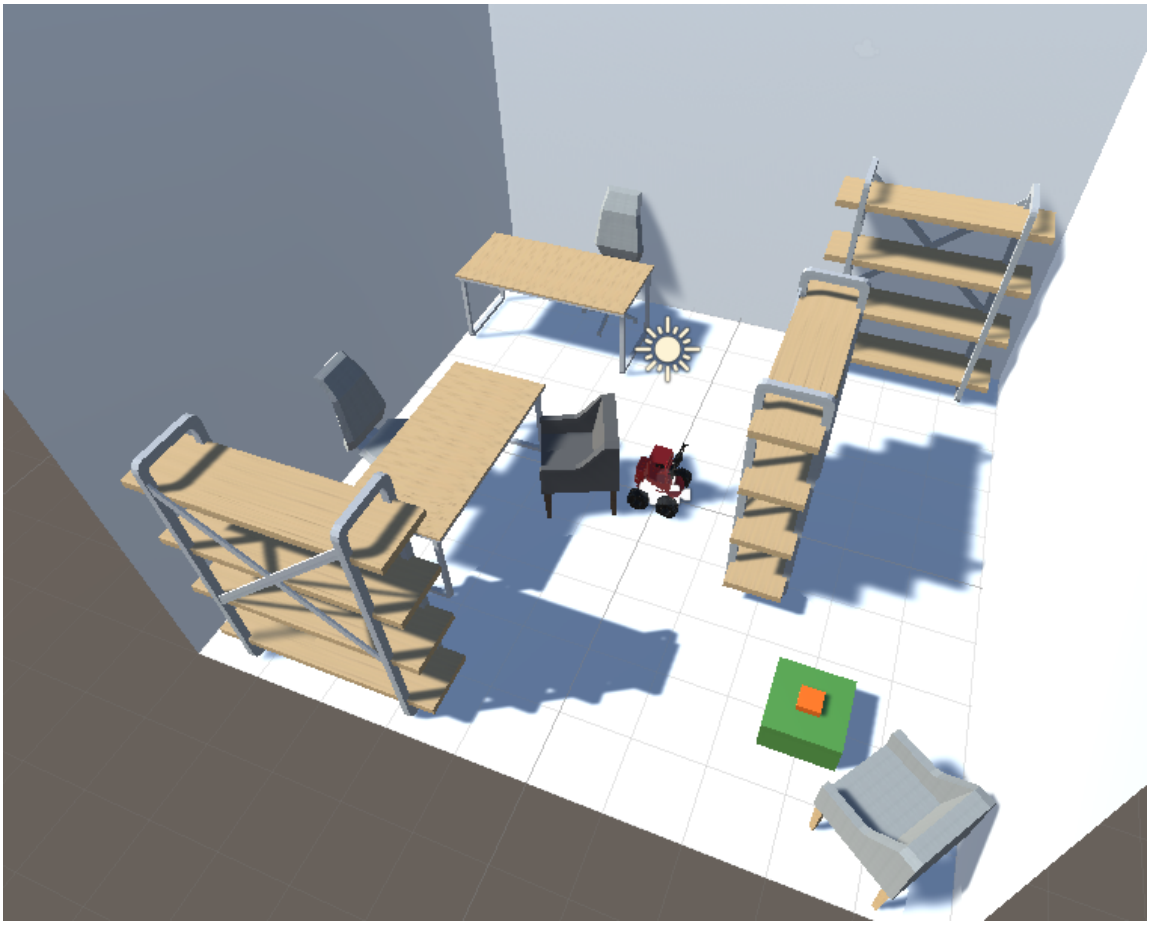


Figura 3.7: Entorno de entrenamiento con habitación con mobiliario estático completo.

Finalmente, creamos un generador de habitaciones aleatorias que proporciona al agente acceso a casi 1300 habitaciones diferentes para su entrenamiento, las cuales entran en juego en el entrenamiento de forma gradual como parte del aprendizaje curricular una vez que el agente completa con éxito su objetivo navegando la habitación estática con todo su mobiliario. Estas habitaciones se generan mediante la combinación de seis distribuciones diferentes de piezas de mobiliario, que se ubican aleatoriamente en cada episodio en cada uno de los cuatro sectores en los que se divide la habitación, pudiendo repetirse. Una de las distribuciones posibles puede verse en la figura 3.8. El conjunto de piezas de mobiliario seleccionadas tienen una complejidad tanto geométrica como visual mucho mayor a aquellas encontradas en la habitación estática, de manera que el agente desarrolle estrategias de navegación más precisas y termine de pulir su percepción visual. En algunos casos se aumenta la complejidad de los objetos combinándolos entre ellos, como es el caso de mesas con ordenadores portátiles sobre ellas o una mesa de café con dos plantas sobre ella y un conjunto de libros en su contenedor inferior. También se añaden distracciones como alfombras, puertas y cuadros.

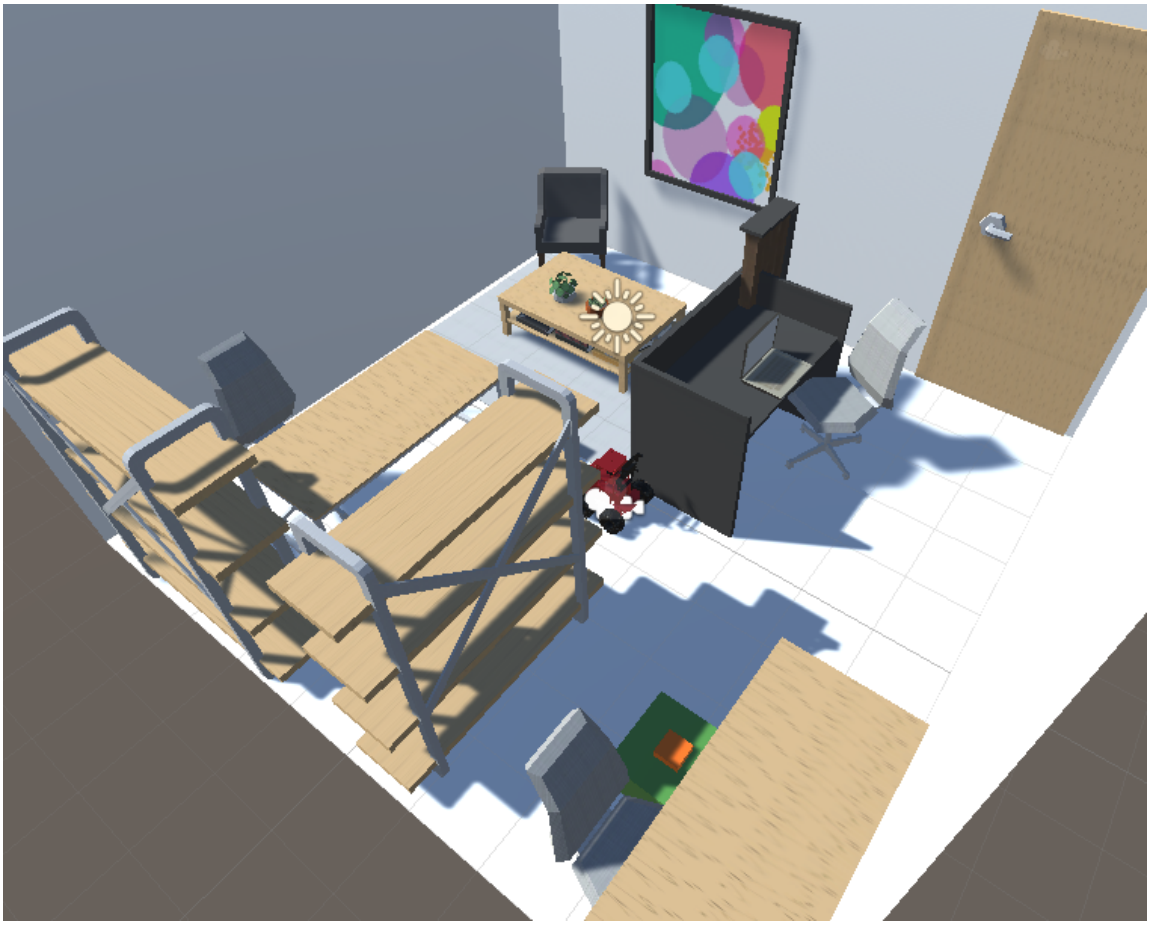


Figura 3.8: Entorno de entrenamiento con habitación con mobiliario aleatorio en una de sus disposiciones posibles.

Ya en este punto, disponemos de un entorno refinado que permitirá entrenar al agente de manera efectiva. Gracias a su variedad, aseguramos que el modelo tenga cierta capacidad de generalización y no experimente un deterioro significativo en su rendimiento al realizar la transición al robot físico, de manera que sea capaz de enfrentar los desafíos del mundo real de manera exitosa. No obstante, si se quiere poder afirmar que el modelo entrenado es capaz de generalizar, será necesario ponerlo a prueba en un contexto desconocido. Es por ello que tras definir los experimentos a realizar construimos una habitación adicional, pensada para ejecutar el modelo entrenado en inferencia y desafiar al agente.

Esta habitación de inferencia contiene muchos de los elementos utilizados en las habitaciones de entrenamiento, pero también incluye dos elementos nuevos. El primero de ellos es un taburete situado en el centro de la habitación, por lo que será un obstáculo recurrente para el agente, y el segundo son dos macetas que utilizan exactamente los mismos materiales que el objetivo. Concretamente, la maceta en sí utiliza el naranja del detalle superior del objetivo, y la planta por su parte el mismo tono de verde que cubre el objetivo casi en su totalidad. Con esto pretendemos poner a prueba la percepción visual del agente. De resto, nos aseguramos de disponer el mobiliario de manera que el agente se vea obligado a realizar giros complicados y seguir rutas poco intuitivas.

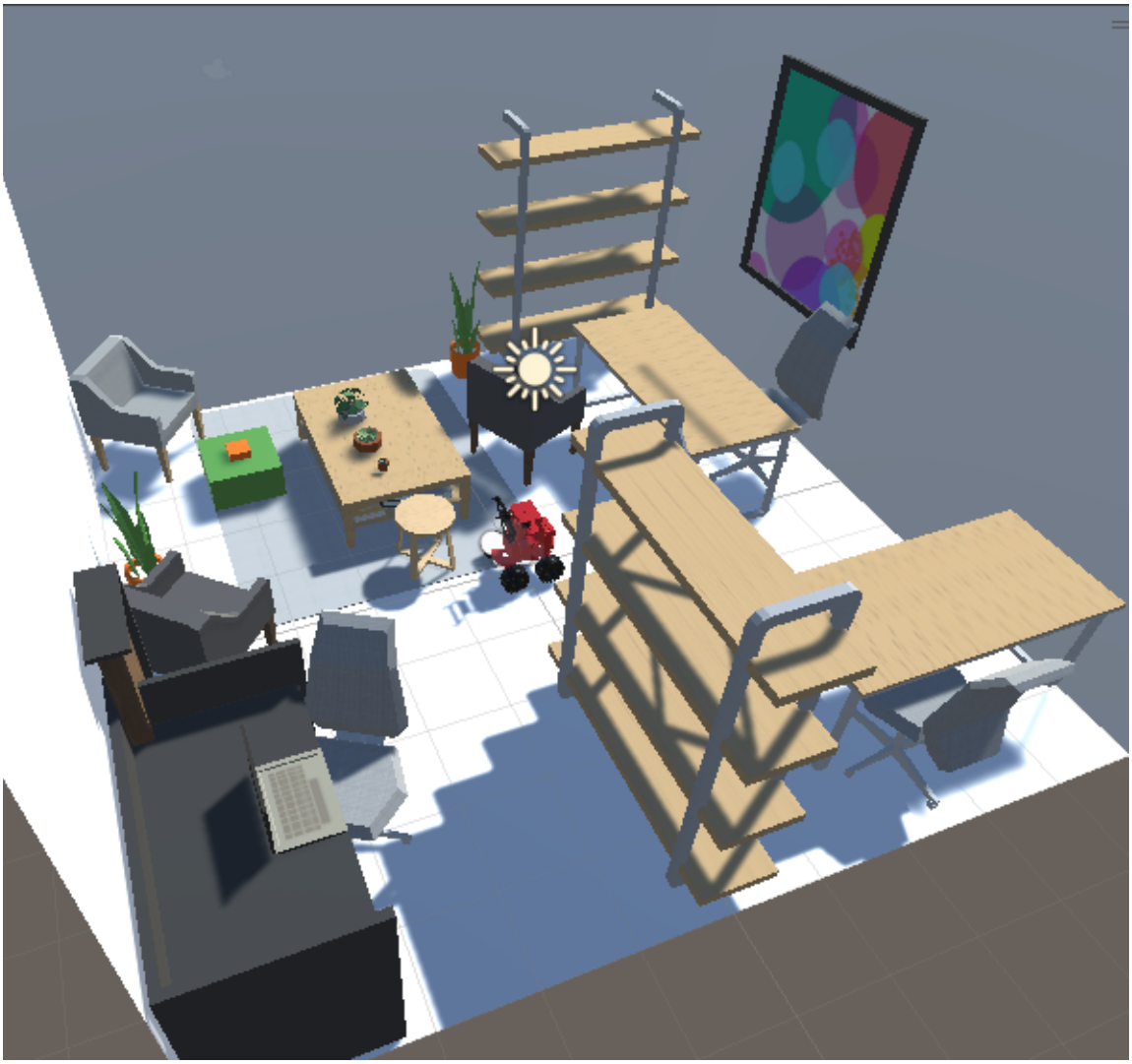


Figura 3.9: Disposición de la habitación de inferencia.

3.3. Parámetros de la red y sensores del agente

Una vez establecido el entorno en el que operan los agentes, pasamos a definir el funcionamiento de la red neuronal sobre la que operarán los mismos, así como sus observaciones, acciones y sistema de recompensas.

Las observaciones del agente se limitan al último fotograma recibido por la cámara montada en la parte frontal del agente. Esta cámara captura imágenes en escala de grises con una resolución de 36x36 píxeles, como en el fotograma que ilustra la figura 3.7. Aunque esta resolución pueda parecer excesivamente pequeña, a la hora de incluir datos visuales como entrada de un entrenamiento de aprendizaje por refuerzo debe acotarse el tamaño de estos lo máximo posible para reducir las computaciones necesarias, de manera que la información procesada sea la mínima que permita la compleción de la tarea. Una buena manera de medir esto es que una persona intente completar la tarea teniendo acceso únicamente al mismo conjunto de datos que el agente que se desea entrenar. El tamaño de imagen elegido implica que el número de canales de entrada de la red neuronal es de 1296, ya que cada píxel se asigna a

un canal utilizando el valor de su nivel de gris. Si hiciéramos uso de una imagen a color, el número de canales de entrada se triplicaría, dado que para cada píxel habría que representar su canal rojo, verde y azul, complicando excesivamente la entrada de la red. También se realizan pruebas utilizando una cámara idéntica pero estática, ubicada en una vista cenital de la habitación.

Para que el agente pueda hacer un uso efectivo de esa información de entrada, es necesario interpretarla correctamente. Con este propósito, al comienzo de la red neuronal se utilizan tres capas convolucionales, siguiendo la propuesta de Volodymir Mnih et al. [33]. Tras pasar por estas capas convolucionales, se encuentran en la red otras tres capas, esta vez completamente conectadas y cada una compuesta de 256 canales, que finalmente desembocan en la salida de la red.

La salida de la red está compuesta por dos únicos canales, cada uno de los cuales genera una señal continua, es decir, un número de punto flotante comprendido entre 1 y -1. Estas señales continuas se utilizan para controlar el desplazamiento del agente en el eje Z y la rotación del agente respecto al eje Y. En cuanto a los hiperparámetros que determinan el funcionamiento del algoritmo de entrenamiento, inicialmente utilizamos los valores más comunes, los cuales se encuentran explicados en la documentación de ML-Agents [32], y realizamos los ajustes necesarios para adaptarlos a nuestro caso específico.

Figura 3.10: Fotograma tal y como se envía a la red neuronal.

Es esencial que la red neuronal pueda procesar de manera eficiente las observaciones del agente y generar salidas adecuadas para controlar su movimiento. A medida que avanzamos en el entrenamiento y ajustamos los hiperparámetros, buscamos mejorar el rendimiento y la capacidad del agente para enfrentar los desafíos del entorno de manera más efectiva. El objetivo es lograr un equilibrio óptimo entre el procesamiento de la información visual y las acciones generadas por la red neuronal, permitiendo al agente desarrollar habilidades de navegación y toma de decisiones inteligentes.

3.4. Función de recompensa

Finalmente, habiendo establecido la configuración del entorno y el funcionamiento de la red neuronal, es crucial definir el sistema de recompensas para evaluar el desempeño de nuestros agentes en inferencia, así como guiar sus entrenamientos. Este aspecto es especialmente delicado y personalizado en un proyecto de esta naturaleza, ya que está estrechamente relacionado con el entorno en el que se llevará a cabo la tarea, el objetivo que debe cumplir el agente y las capacidades propias del agente para realizar acciones que lo acerquen a dicho objetivo. Además, la forma en que se recompensen las acciones del agente puede favorecer ciertos comportamientos sobre otros, y una configuración inadecuada podría propiciar la aparición de malos hábitos.

Existen aproximaciones al aprendizaje por refuerzo que incluyen el aprendizaje de funciones de recompensa efectivas a través de la interacción y observación de agentes humanos, pero creemos que una implementación como esa complicaría demasiado el proyecto. Es por ello que decidimos definir manualmente la función de recompensa que rija los entrenamientos de nuestros agentes. Tras considerar diversas opciones, como recompensar gradualmente la proximidad al objetivo o mantenerlo en el campo de visión de la cámara, decidimos simplificar el sistema de recompensas. En este caso, optamos por recompensar al agente únicamente al completar la tarea y aplicar una penalización leve en cada paso del entrenamiento para fomentar la finalización de la tarea en el menor tiempo posible. De manera formal, podemos expresar la función de recompensa utilizada como:

$$R = (5 * G + \sum_{i=1}^{S} \frac{-1}{S}) * (1 - B) - B$$

Donde R representa la recompensa, G es un marcador binario que indica si se ha completado o no la tarea, y S es el número máximo de pasos que el agente puede realizar durante un episodio de entrenamiento. De esta manera, la recompensa oscilará entre 4 y 5 en función del tiempo que el agente tarde en completar la tarea, mientras que se establecerá en -1 si el agente agota todos los pasos máximos permitidos para un episodio.

Adicionalmente, hemos incluido una penalización rígida que establece la recompensa en -1 en caso de que el agente, ya sea por aprovechar debilidades del entorno virtual o debido a un fallo del mismo, salga de la habitación designada. En caso de que esto ocurra, la variable B de la función de recompensa tomaría valor 1, siendo 0 en caso contrario. Esta penalización busca desalentar cualquier intento de explotar inconsistencias o errores en el entorno y garantizar que el agente se mantenga dentro de los límites establecidos.

El diseño de un sistema de recompensas adecuado es esencial para guiar el comportamiento del agente y promover su aprendizaje óptimo. A medida que avancemos en el entrenamiento y evaluemos el desempeño del agente, estaremos atentos a ajustar las recompensas y las penalizaciones en caso de ser necesario, con el objetivo de mejorar su rendimiento y fomentar la adquisición de habilidades de navegación y toma de decisiones inteligentes en el entorno virtual.

Capítulo 4

Experimentos y Resultados

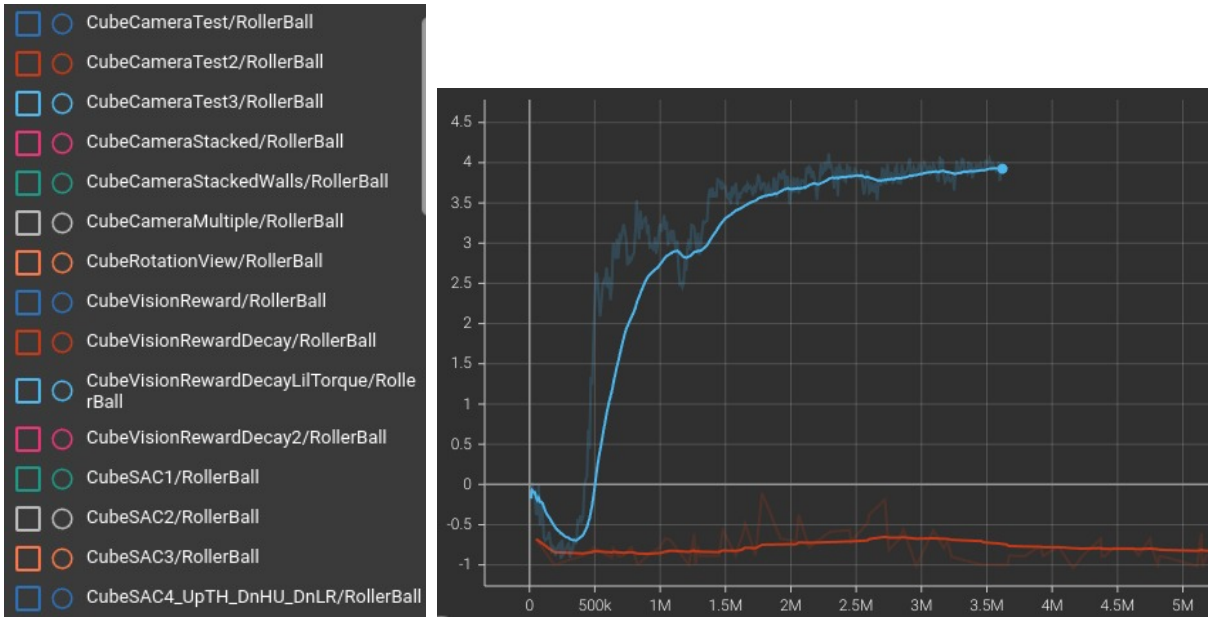
Tras plantear la evolución de todos los componentes virtuales del proyecto en el pasado capítulo, en este exponemos los motivos de esos cambios y el avance del proyecto a medida que ejecutamos distintos experimentos en estos entornos. En todos los experimentos el agente aprende desde cero, tras haber cambiado su configuración, su entorno o ambos a la vez.

Para el desarrollo de las diversas pruebas, se compiló el entorno de simulación en un ejecutable que permite la ejecución simultánea de varios entornos idénticos, lo que resulta beneficioso durante el entrenamiento. Al ejecutar 8 instancias de un entorno con 9 agentes cada una, podemos llevar a cabo pruebas con un total de 72 agentes al mismo tiempo, lo que reduce considerablemente el tiempo de ejecución. Además, con la escala de tiempo por defecto se consiguen simular veinte segundos por cada segundo en el mundo real.

4.1. Habitación vacía

En primera instancia realizamos una serie de pruebas utilizando *proximal policy optimization* (PPO) como algoritmo de entrenamiento, las cuales no arrojan buenos resultados. Tomando como referencia el trabajo de Wu et al. [12] con un robot Sphero Ollie, en cuyas pruebas de navegación utilizando una cámara como único sensor se utiliza un algoritmo de actor crítico, adaptamos el entrenamiento para que utilice el *soft actor critic* implementado en ML-Agents. Esto implicó realizar diversos cambios en los hiperparámetros, que requieren valores específicos para este segundo algoritmo, además de introducir algunos parámetros completamente nuevos. Sin embargo, incluso con este enfoque, las pruebas iniciales tampoco arrojaron resultados satisfactorios, y es que la red no consigue definir ninguna política realmente útil. Además, surgieron problemas durante la ejecución del proceso de entrenamiento, como bloqueos repentinos y ocasiones en que se obtienen valores de pérdida cercanos al infinito, lo cual no es computable.

Después de realizar varias iteraciones de prueba y error, descubrimos las causas que impedían el progreso de los entrenamientos. En primer lugar, se identificó una mala implementación de las fuerzas aplicadas para mover al agente, lo cual fue consecuencia de no cambiar la función utilizada en el entorno de prueba inicial en el que se basó el desarrollo del proyecto. En este entorno de prueba, el agente era de forma esférica,



(a) Pruebas iniciales (b) Resultados de estas primeras pruebas

Figura 4.1: Diversos entrenamientos de prueba (a) realizados para ajustar el entorno, así como el entrenamiento y el comportamiento del agente, y los resultados finales de esta primera fase (b) en las pruebas con comprobación visual (rojo) y *bumper* (azul).

y la función utilizada para moverlo no aplicaba fuerzas adecuadas para desplazar un cubo, como es el caso de la geometría física que le asignamos a nuestros agentes. Esto resultaba en un movimiento errático que a simple vista parecía ser causado por una mala salida del modelo. Después de implementar la función "AddForce", la cual en vez de aplicar una fuerza sobre el agente le añade aceleración en la dirección especificada, el agente comenzó a moverse adecuadamente.

La segunda causa de error se originó por una mala comprobación de la finalización de la tarea, que como ya hemos comentado consiste en alcanzar el cubo objetivo con la parte frontal del agente. Este fallo en la comprobación no permitía que el agente pudiera aprender de manera efectiva. Para solucionar este problema, se cambió por completo la comprobación, pasando de verificar si el objetivo se encuentra en el campo de visión del agente y a una distancia menor que un valor predefinido, a una simple comprobación de contacto mediante un *bumper* frontal, un enfoque que requiere menos cálculos y presenta una mayor fiabilidad. Es importante destacar que este *bumper* es una herramienta puramente digital que no trascenderá a la implementación física.

Con todos los errores solucionados, los entrenamientos comenzaron a completarse en un número razonable de episodios, al tiempo que desaparecen los errores de ejecución observados anteriormente, probablemente asociados a entrenamientos demasiado largos, puesto que se estaban alcanzando entrenamientos del orden de decenas de millones de pasos. En la figura 4.1 puede apreciarse cómo completamos un entrenamiento con éxito, asentándose la recompensa media obtenida alrededor de los cuatro puntos, una recompensa alta acorde a la baja dificultad de esta primera tarea.

4.2. Cámara cenital y ruido visual

Tras comprobar el funcionamiento del robot al utilizar una cámara montada, surgió la idea de explorar la posibilidad de utilizar una cámara cenital estática. Este enfoque permitiría que el agente tuviera en todo momento información acerca de la habitación al completo, facilitando la detección del objetivo y la planificación de una ruta hacia él. Sin embargo, esta ventaja conlleva el desafío de que el modelo debe comprender la orientación del agente, además de la posibilidad de perder de vista al robot o al objetivo debido a objetos como una mesa. Además, la implementación física se vuelve más complicada. También es pertinente apuntar que para beneficiarnos realmente de la vista cenital sería necesario aumentar la resolución de la cámara, lo que aumentaría significativamente el número de canales de entrada en la red neuronal. Esto, a su vez, incrementaría la carga computacional y dificultaría la convergencia del modelo en el entrenamiento. Teniendo en cuenta estos puntos y después de comprobar que los resultados de las pruebas no mostraban mejoras significativas, se decidió continuar el desarrollo utilizando la cámara montada en el frontal del agente.

Encontrar el objetivo en una habitación blanca es una tarea relativamente sencilla, y tras su consecución y haber descartado el uso de una cámara cenital, se propuso comprobar la correcta configuración de la cámara montada mediante la introducción de ruido visual. Para ello, se aplicaron cuatro materiales, creados a partir de imágenes de habitaciones amuebladas reales, a las cuatro paredes del entorno de entrenamiento. De esta manera, el agente debe aprender a diferenciar su objetivo de otros objetos, sin la dificultad añadida de tener que navegar sorteando obstáculos. Esta prueba muestra un proceso de aprendizaje algo más lento en comparación con la habitación vacía, como se puede comprobar en la figura 4.2, y es que para alcanzar valores de recompensa media similares el agente en habitación con ruido visual necesita unos dos millones y medio de pasos adicionales. Esto era claramente previsible teniendo en cuenta el aumento en la complejidad de la tarea. Sin embargo, los resultados obtenidos fueron alentadores desde un principio, y con ellos iteramos hacia el siguiente problema.

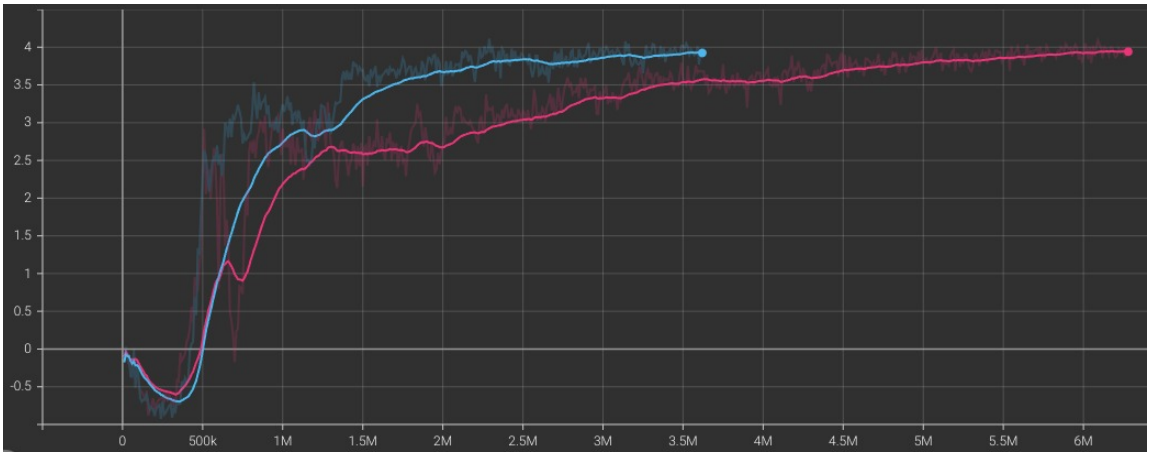


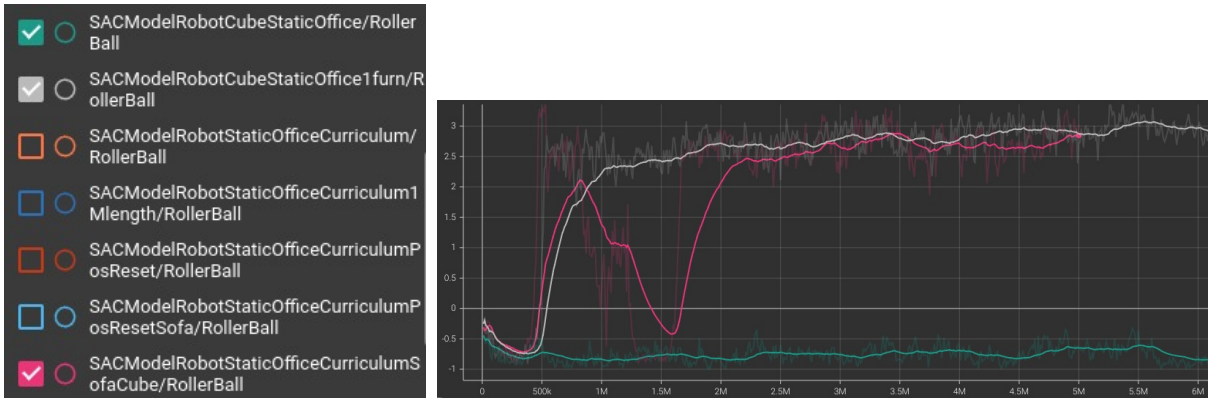
Figura 4.2: Gráfica de resultados en habitación vacía (azul) y con ruido visual (rosa)

4.3. Habitación amueblada con aprendizaje curricular

En este punto del desarrollo ya empezamos a enfrentar una situación más cercana a la realidad que enfrentaría un agente robótico en el mundo físico. El agente debe navegar por una habitación desconocida con diversos obstáculos que pueden encontrarse fácilmente en una oficina, y encontrar en algún lugar dentro de la misma a su objetivo.

Las primeras pruebas en este entorno arrojan resultados desalentadores, puesto que el agente no logra aprender a navegar por la habitación de manera efectiva, independientemente del número de episodios que se simulen. El problema radica en que aprender a navegar por un entorno complejo desde cero, sin siquiera haber aprendido a manejarse a sí mismo eficientemente primero, resulta una tarea demasiado difícil, abrumadora para el agente. Es por ello que decidimos implementar *curriculum learning*, o aprendizaje curricular, un estilo de aprendizaje consistente en dividir el entrenamiento de manera que el agente se enfrente a situaciones que vayan escalando en dificultad a medida que va aprendiendo.

De esta manera modificamos el archivo de configuración del entrenamiento para establecer cinco fases distintas. En la primera fase, el agente debe alcanzar el objetivo en una habitación vacía, de manera que comience aprendiendo a interpretar la información visual proveniente de la cámara montada, controlar su movimiento de manera efectiva y localizar el cubo objetivo. A partir de ahí, incrementamos gradualmente la complejidad de la navegación insertando primero las estanterías, luego las mesas, tras ellas las sillas y finalmente los sillones de la habitación. Así, el agente va adquiriendo la habilidad de evitar obstáculos cada vez más variados en un entorno cada vez más restringido.



(a) Lista de pruebas (b) Gráfica de resultados

Figura 4.3: Diversos entrenamientos de prueba (a) realizados para diagnosticar problemas y refinar el entorno de habitación amueblada. Los resultados finales (b) con aprendizaje curricular (rosa) demuestran la capacidad del agente para mantener en un entorno complejo el mismo nivel de recompensa que en un entorno con un solo mueble (gris), lo cual sin aprendizaje curricular (verde) no sería posible.

Teóricamente esto debería haber funcionado sin problemas, pero en la práctica, rara vez las cosas resultan tan sencillas. Es así que al ejecutar observamos un estancamiento de los entrenamientos en el momento en que se incluyen las sillas de oficina. Después de explorar diversas opciones relacionadas con el entrenamiento y la configuración del agente, notamos durante una ejecución del modelo por inferencia en tiempo real que los agentes se quedan atascados en las sillas, impidiéndoles completar su objetivo no solo en el episodio actual, sino también en los episodios siguientes. Esto se debía a que el agente, para que los episodios tengan todos posiciones iniciales distintas, no está programado para reiniciar su posición, excepto en caso de salirse de su entorno de entrenamiento o tras la inclusión de un nuevo elemento en el escenario, esto último para evitar esta clase de problemas de estancamiento.

Para solucionar este inconveniente, realizamos una modificación del *collider* (la geometría utilizada por Unity para calcular colisiones entre objetos físicos) de las sillas, simplificándolo. Esto permitió que los agentes no tuvieran mayores problemas con las sillas y pudieran completar su entrenamiento con éxito, como se puede comprobar en la figura 4.3, donde la recompensa media obtenida con la habitación completa se consigue equiparar a la obtenida previamente con un único mueble. De esta manera aprenden a navegar la oficina para encontrar su objetivo. Sin embargo existe otro problema, y es que muy probablemente el conocimiento del agente no sería fácilmente transferible a otras oficinas, y mucho menos al mundo real, puesto que podríamos decir que el modelo está especializado en esta única habitación. Siendo nuestro objetivo conseguir un agente lo más general posible, debemos variar los entornos utilizados durante el entrenamiento.

4.4. Habitaciones aleatorias con aprendizaje curricular

Habiendo demostrado la capacidad del agente para aprender a navegar una oficina, ahora queremos conseguir que pueda hacer lo mismo en cualquier habitación en la que se encuentre. Con este objetivo en mente, creamos un generador de habitaciones aleatorias el cual nos permite añadir otras cuatro fases adicionales al aprendizaje curricular del agente, a lo largo de las cuales la habitación generada puede volverse progresivamente más desafiante. Para lograr esto, amueblamos únicamente un sector de la habitación en la primera fase, dos sectores en la segunda y así sucesivamente, hasta generar habitaciones completamente amuebladas en la cuarta fase. Este enfoque nos permite exponer a los agentes a una variedad de configuraciones de habitaciones y obstáculos de forma gradual y controlada, fomentando su capacidad de adaptación para navegar en diferentes entornos.

Sin embargo, las primeras pruebas en este nuevo paradigma resultaron en que los agentes terminaban cayendo en la más absoluta inopia. Desde el momento en que completaban el entrenamiento en la habitación con una disposición fija de mobiliario, comienzan a olvidar lo aprendido sin llegar a cumplir nunca los requisitos para pasar a la siguiente etapa del aprendizaje curricular con habitaciones aleatorias. Conseguimos solventar este problema rápidamente aumentando el número máximo de pasos por episodio de entrenamiento, y es que llegamos a la conclusión de que los agentes no disponían del suficiente tiempo para cumplir su objetivo con la recompensa necesaria, por lo que intentaban mejorar sus políticas con un enfoque exploratorio que realmente no podía aspirar a resultados mejores.

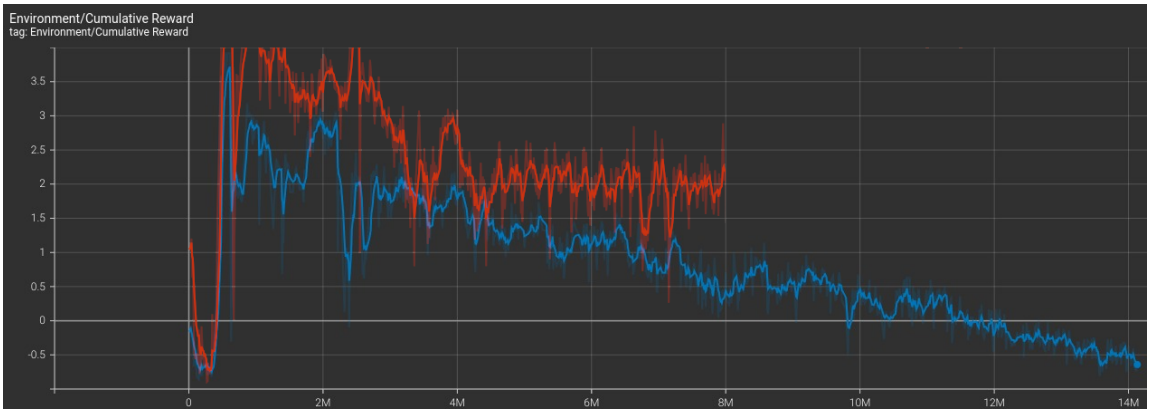


Figura 4.4: Gráfica de recompensa acumulada en habitación aleatoria con un máximo de 200 pasos por episodio (azul) y 500 (rojo)

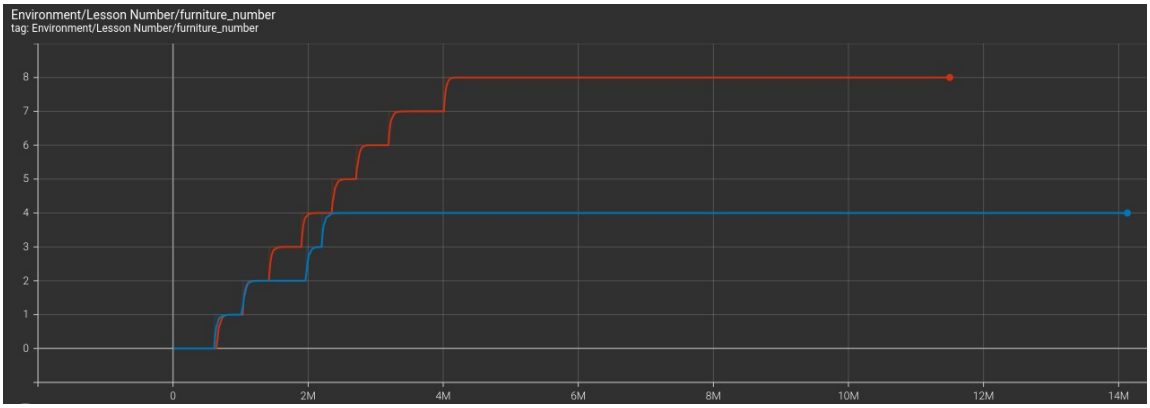


Figura 4.5: Gráfica de avance del aprendizaje curricular en habitación aleatoria con un máximo de 200 pasos por episodio (azul) y 500 (rojo)

Con este ajuste, los agentes no solo son capaces de empezar el entrenamiento con habitaciones aleatorias, cosa que ocurre después de casi 2.5 millones de pasos como puede verse en la figura 4.5, sino que logran completar el entrenamiento navegando con éxito las habitaciones con los cuatro sectores amueblados. Estos resultados pueden apreciarse en la figura 4.4, en la que se ve claramente cómo los agentes se estabilizan en una obtención consistente de alrededor de dos puntos de media de recompensa por episodio a partir de los 4 millones de pasos. El entrenamiento se da por finalizado tras 8 millones de pasos y 8 horas 45 minutos de cómputo, algo más de siete días en tiempo de simulación.

4.5. Inferencia en habitación desconocida

Tras llevar a cabo todo este proceso incremental de entrenamiento, podríamos asumir que contamos con un agente generalista especializado en navegación de interiores. Sin embargo, es necesaria una prueba final para respaldar esta afirmación, y con este propósito creamos una habitación completamente nueva a partir del mobiliario utilizado anteriormente, junto a algunos elementos no vistos antes por el agente como una planta y una maceta que usan exactamente los mismos materiales que el objetivo. Al ubicar los distintos objetos que forman parte de la habitación con la dis-

posición ilustrada en la figura 4.6, nos aseguramos de que esta presente un desafío para el agente, pero sin dejar de ser navegable.

Para realizar esta prueba de inferencia utilizamos como base el mejor modelo del que disponemos hasta el momento, el cual estudiamos en el apartado anterior. De esta manera, ML-Agents registra los resultados como una continuación del entrenamiento, pero sin realizar actualizaciones del modelo. Podemos ver estos resultados en la figura 4.7 en color rojo, más concretamente a partir del pico en los once millones y medio de pasos. Con un nuevo aumento de dificultad, la recompensa media obtenida sufre una vez más como ocurría anteriormente en los cambios de lección del aprendizaje curricular. No obstante, esta se mantiene con un valor estable superior un punto, lo cual demuestra la capacidad del agente de localizar su objetivo en esta habitación completamente desconocida sin necesidad de entrenamiento adicional.

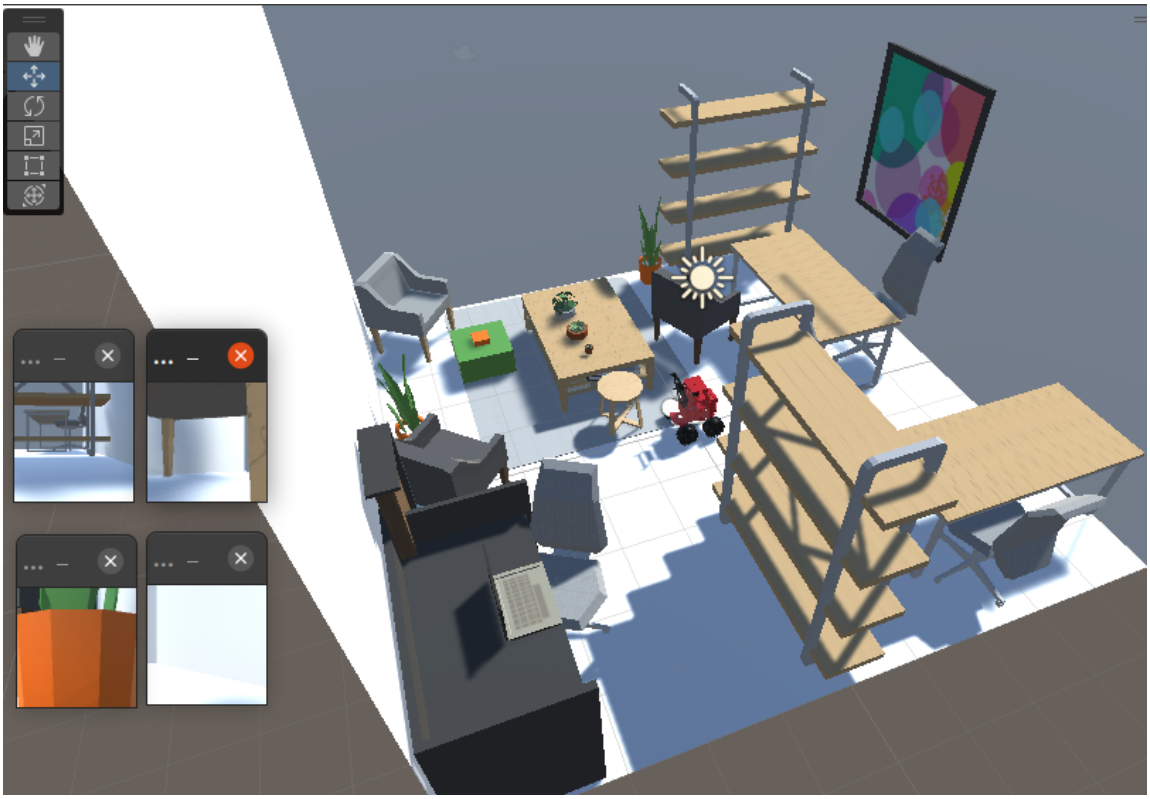


Figura 4.6: Disposición de la nueva habitación. A la izquierda, cuatro vistas de distintos agentes durante el proceso de inferencia. Recordemos que los agentes procesan estas imágenes en una escala de grises y a resolución 36x36.

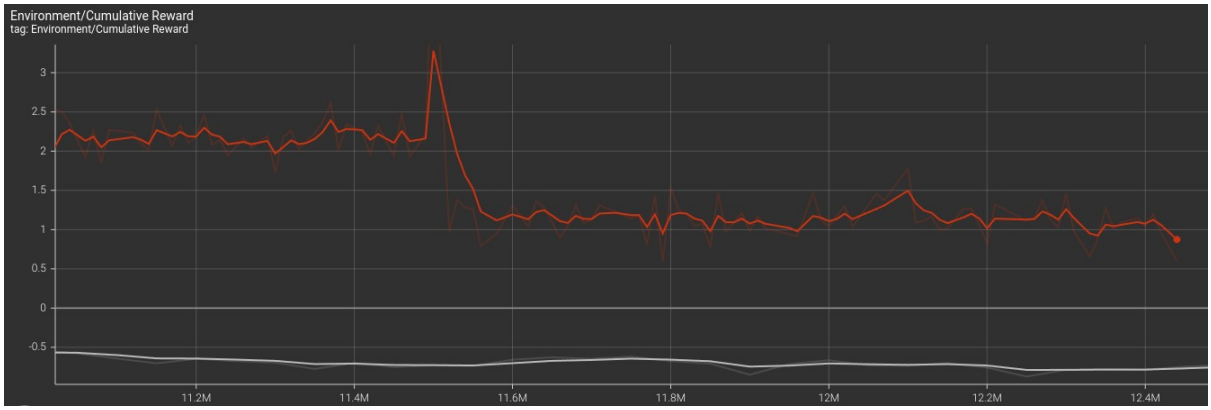


Figura 4.7: Inferencia en habitación aleatoria hasta los 11.5M de pasos, donde se pasa a inferencia en habitación desconocida (rojo) junto a resultados de un entrenamiento que fue incapaz de superar la habitación estática (gris). El pico en la recompensa obtenida al cambiar de ejecución es un error de medición que ocurre siempre al iniciar una nueva ejecución, por lo que no representa los resultados obtenidos.

Capítulo 5

Conclusiones

Antes de hacer ninguna afirmación sobre la consecución de nuestro objetivo principal, creemos de rigor realizar una revisión de los subobjetivos que nos debían llevar a alcanzar este.

- La selección de un robot adecuado tomó más tiempo del esperado dados los problemas derivados del estado actual del Pioneer 2DX, pero conseguimos un candidato viable con el Xiao Rover 2, que dispone de todos los elementos necesarios para completar con éxito tareas de navegación y búsqueda de objetivos, además de la capacidad de añadir componentes adicionales en caso de necesitarlo gracias a su naturaleza modular.
- La digitalización del Xiao Rover 2 fue un éxito mediante el uso de las herramientas provistas por ROS 2 para procesar sus archivos de configuración ".xacro" y generar un archivo ".urdf" con el que importarlo a Unity.
- La construcción del entorno virtual también fue lograda, con el diseño de 5 tipos de habitaciones (vacía, con ruido visual, estática, aleatoria y de inferencia) en las que apoyar diferentes experimentos. Además, los componentes de estas habitaciones son bastante comunes en entornos interiores, lo cual prepara al agente para enfrentarse al mundo real.
- Preparar una lógica adecuada tras los entrenamientos ha sido uno de los retos más grandes. La parametrización de la red neuronal quedó prácticamente asentada tras la primera serie de pruebas, pero la gestión de los entrenamientos presentó mayores dificultades al tener que realizar distintos experimentos. Aún así, se fueron implementando las mecánicas necesarias, como fue el caso del aprendizaje curricular en el experimento con habitaciones aleatorias, y se lograron completar con éxito todos los experimentos propuestos.
- Finalmente, la evaluación de los resultados que se han ido obteniendo ha sido clave a la hora de poder ir avanzando en los experimentos. Estos resultados han sido la manera en que nos hemos podido comunicar con el modelo para ir entendiendo sus necesidades ante el incremento de dificultad de sus tareas.

Con todo esto y tras estudiar los resultados conseguidos en el capítulo anterior, podemos afirmar que hemos cumplido el objetivo principal del proyecto. Habiendo cumplido todos los subobjetivos planteados en el primer capítulo de esta memoria, hemos logrado obtener un modelo robusto con la capacidad de navegar entornos desconocidos, con elementos no vistos anteriormente y en disposiciones completamente nuevas sin necesidad de reentrenamiento.

La primera conclusión extraída del desarrollo de este proyecto surge en las primeras fases del mismo, y es la necesidad de realizar pruebas con un método heurístico tras implementar nuevas mecánicas que afecten a los entrenamientos. Problemas con los cálculos físicos de la simulación o la lógica detrás de esta pueden retrasar un proyecto innecesariamente, siendo el tiempo perdido por no realizar estas pruebas inadmisible en un proyecto de mayor escala. Dada la rápida iteración entre unas pruebas y otras no se consideró especialmente necesario para este caso, pero es cierto que habría acelerado sobre todo las primeras etapas de experimentación.

La segunda conclusión tiene más relación con el kit de herramientas ML-Agents en sí. Su implementación del aprendizaje curricular es una manera excelente de abordar tareas complejas, pero los métodos que incluye en este momento para comprobar la correcta finalización de una tarea no son suficientes si se busca entrenar agentes que puedan generalizar correctamente. Comprobar si se mantiene la obtención de una recompensa mínima durante una cantidad determinada de episodios es insuficiente si tenemos en cuenta el fenómeno del *grokking* [41], por el cual un modelo de aprendizaje por refuerzo va a memorizar primero su conjunto de test y, tras entrenar por mucho más tiempo, finalmente aprenderá a generalizar. Para abordar esto sería necesario que la condición para avanzar en lecciones que buscan generalizar el conocimiento fuese completar con éxito una ejecución en inferencia en un contexto desconocido, la cual se incluiría periódicamente en el flujo del entrenamiento, puesto que no podemos predecir con exactitud en qué momento el modelo desarrollará la capacidad de generalizar, aunque existen ya algunas líneas de investigación al respecto [42].

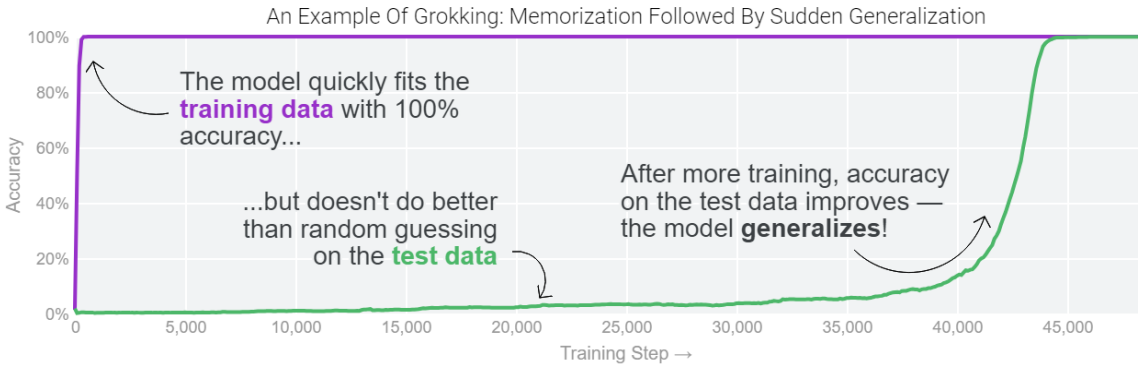


Figura 5.1: Ejemplo de *grokking* [41]

Finalmente, surge una tercera conclusión, y es que dada la falta de un modelo del mundo nuestro agente no es capaz de hacer una planificación de su ruta, y únicamente actúa en base a los estímulos más inmediatos. Esto tiene un impacto directo sobre la recompensa obtenida a través de su penalización por número de pasos realizados, al necesitar desarrollar estrategias más dirigidas a la exploración ciega del entorno hasta encontrar un camino directo al objetivo, lo cual requiere de más tiempo que planificar una ruta efectiva. Sin embargo, una reestructuración de la red neuronal para que el agente cuente con cierto contexto temporal podría ser una solución eficaz que no necesite de un modelo del mundo, al permitir al agente realizar maniobras más complejas. Igualmente, la falta de un modelo del mundo siempre presentará una limitación bajo los paradigmas actuales de sistemas autónomos.

Teniendo estas conclusiones en cuenta y atendiendo a los resultados obtenidos, surgen otras cuestiones, aunque estas toman principalmente la forma de ideas para futuras líneas de trabajo.

5.1. Futuras líneas de trabajo

Este trabajo ha sido realmente emocionante. Podríamos continuar desarrollando experimentos nuevos con los que desafiar a nuestros agentes, pero creemos que es el momento de parar, evaluar lo aprendido y tratar de implementarlo en una versión mejor desde su base. Siendo esto así, hemos identificado diversas maneras en que se podrían tratar de mejorar los resultados obtenidos. En este momento se escapan al alcance del proyecto, pero las creemos posibles para trabajos futuros. A continuación presentamos algunas de ellas.

En primer lugar y partiendo de la base de la tercera conclusión presentada anteriormente, proponemos mejorar la toma de decisiones del agente enriqueciendo los datos a procesar por la red. Esto se puede conseguir dotando de recurrencia a la red mediante la implementación de un sistema de $Long - Short\ Term\ Memory$ (LSTM) [36], de manera que los estados anteriores se procesen como una entrada más. Un sistema así permitiría que la red tuviera cierta memoria de las decisiones recientes y los motivos tras estas, otorgando al agente la capacidad de tomar decisiones coherentes con cierto contexto temporal. Ya se ha utilizado esta técnica anteriormente en trabajos de seguimiento activo de objetos [35], e incluso métodos de recurrencia mucho más simples como establecer la última acción realizada como una entrada más han sido de gran utilidad en el entrenamiento de sistemas autónomos como el publicado recientemente por Kaufmann et al. [46], cuyo sistema de control puede verse en la figura 5.2. En nuestro caso, la implementación de una red con LSTM ayudaría a nuestros agentes en la navegación de entornos desconocidos al proporcionarles una dimensión de información completamente nueva que pueden utilizar para desarrollar estrategias de navegación más precisas, controlar mejor la velocidad y realizar giros más eficientes, potencialmente preparándolos incluso para navegar entornos no estructurados.

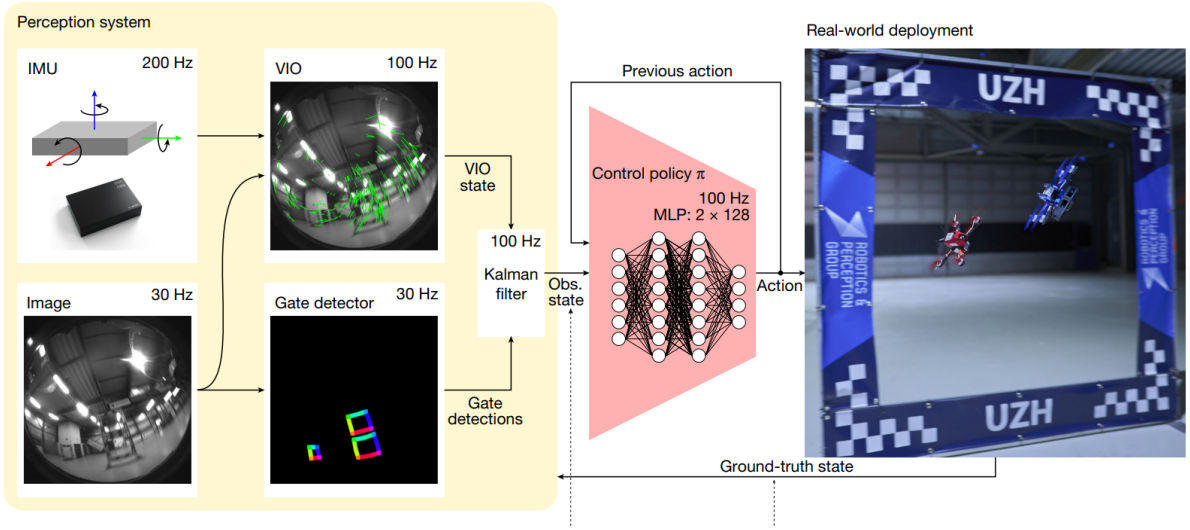


Figura 5.2: Sistema de control implementado recientemente en un dron autónomo de carreras [46]. Puede observarse como la red utiliza su última acción realizada como una entrada más.

Nuestra segunda propuesta está más relacionada con las acciones de los agentes y la precisión de la simulación. En el estado actual del proyecto, las acciones elegidas por el modelo activan funciones físicas propias de Unity que añaden fuerzas de aceleración de mayor o menor magnitud a un cubo que encapsula al robot, lo cual nos sirve para realizar pruebas pero tiene claras limitaciones en cuanto a la precisión de la simulación y las acciones que puede realizar un agente. Es por ello que para suavizar y afinar los movimientos del agente sería necesario realizar una implementación completa del sistema robótico del Xiao Rover 2 en la simulación con ROS 2, de manera que las acciones pasen a activar directamente los motores del robot. Este cambio también traería de manera indirecta un mejor cálculo de colisiones, al realizar una representación física precisa de la geometría del robot, aunque debe implementarse con cuidado de que el agente no aprenda como límite de sus movimientos el choque de sus partes con el entorno, puesto que esto podría suponer graves daños al sistema.

Para asegurar e incluso mejorar la generalización del modelo, tenemos una tercera propuesta relacionada con el fenómeno del *grokking* comentado anteriormente. Si bien una modificación de la implementación del aprendizaje curricular de ML-Agents sería posible, no es la solución más elegante, dado que podría entrar en conflicto con futuras actualizaciones de la herramienta. No obstante, siguiendo el trabajo de Notsawo Jr et al. [42] se podría tratar de predecir la aparición del fenómeno mediante el estudio de la pérdida obtenida durante el entrenamiento, aunque es cierto que este método requiere aún de una extensión de su marco teórico.

A partir de ahí, las posibilidades se extienden al contexto en el que se use este modelo, o uno similar. Aún teniendo cierta capacidad de generalización, pues tampoco creemos que sea infalible en cualquier entorno, un modelo especializado en un determinado entorno siempre rendirá mejor. Es por ello que para igualar o incluso superar un modelo así se podría realizar un entrenamiento paralelo a la ejecución física, al estilo de Daydreamer [12] y su uso de un modelo del mundo, de manera que el agente

aprenda de su entorno a medida que lo explora físicamente y simula diversas acciones posibles.

De momento, se pretende continuar estudiando e investigando en el campo de la robótica y los sistemas autónomos, de manera que sea realmente posible llevar la idea principal de este proyecto mucho más allá.

Bibliografía

[1] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. Science Robotics, 7(66), eabm6074.

[2] Koenig, N., & Howard, A. (2004, September). Design and use paradigms for gazebo, an open-source multi-robot simulator. In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566) (Vol. 3, pp. 2149-2154). IEEE.

[3] ”Government & Aerospace”. Unity Technologies. https://unity.com/solutions/government-aerospace

[4] ”Unity Robotics Hub”. Unity Technologies. Github. https://github.com/Unity-Technologies/Unity-Robotics-Hub

[5] Juliani, A., Berges, V. P., Teng, E., Cohen, A., Harper, J., Elion, C., ... & Lange, D. (2018). Unity: A general platform for intelligent agents. arXiv preprint arXiv:1809.02627.

[6] Borkman, S., Crespi, A., Dhakad, S., Ganguly, S., Hogins, J., Jhang, Y. C., ... & Yadav, N. (2021). Unity perception: Generate synthetic data for computer vision. arXiv preprint arXiv:2107.04259.

[7] "Plataforma Omniverse". Nvidia. https://www.nvidia.com/es-es/omniverse/

[8] Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., ... & State, G. (2021). Isaac gym: High performance gpu-based physics simulation for robot learning. arXiv preprint arXiv:2108.10470.

[9] "ML-Agents Toolkit Overview". Unity Technologies. Github. https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/ML-Agents-Overview.md

[10] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[11] ”URDF-Importer". Unity Technologies. Github. https://github.com/Unity-Technologies/URDF-Importer

[12] Wu, P., Escontrela, A., Hafner, D., Goldberg, K., & Abbeel, P. (2022). Daydreamer: World models for physical robot learning. arXiv preprint arXiv:2206.14176.

[13] Guss, W. H., Houghton, B., Topin, N., Wang, P., Codel, C., Veloso, M., & Salakhutdinov, R. (2019). Minerl: A large-scale dataset of minecraft demonstrations. arXiv preprint arXiv:1907.13440.

[14] "MineRL". Carnegie Mellon University. https://minerl.io

[15] "Gymnasium Documentation". OpenAI. https://gymnasium.farama.org

[16] ”OpenAI gym tutorial". Mr Ko. Github. https://ai-mrkogao.github.io/reinforcement%20learning/openaigymtutorial/

[17] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017, October). CARLA: An open urban driving simulator. In Conference on robot learning (pp. 1-16). PMLR.

[18] Rohmer, E., Singh, S. P., & Freese, M. (2013, November). V-REP: A versatile and scalable robot simulation framework. In 2013 IEEE/RSJ international conference on intelligent robots and systems (pp. 1321-1326). IEEE.

[19] "Tutorial: How to simulate Franka robot with CoppeliaSim". Francois Pasteau, Fabien Spindler, Gatien Gaumerais, Alexander Oliva. ROS documentation. http://docs.ros.org/en/noetic/api/visp_ros/html/tutorial-franka-coppeliasim.html

[20] Slamet, B., & Pfingsthorn, M. (2006). Manifoldslam: a multi-agent simultaneous localization and mapping system for the robocup rescue virtual robots competition. Master's thesis, Universiteit van Amsterdam.

[21] "What is Isaac Sim? - Omniverse Robotics Documentation". Nvidia. https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim/overview.html

[22] Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). Trust region policy optimization. In International conference on machine learning (pp. 1889-1897). PMLR.

[23] Kakade, S. M. (2001). A natural policy gradient. Advances in neural information processing systems, 14.

[24] ”Example Learning Environments - ML-Agents Documentation”. Unity Technologies. Github. https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Examples.md

[25] ”Soft Actor Critic—Deep Reinforcement Learning with Real-World Robots”. Berkeley Artificial Intelligence Research (BAIR). https://bair.berkeley.edu/blog/2018/12/14/sac/

[26] ”Maximum Entropy RL (Provably) Solves Some Robust RL Problems”. Berkeley Artificial Intelligente Research (BAIR). https://bair.berkeley.edu/blog/2021/03/10/maxent-robust-rl/

[27] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[28] Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. Advances in neural information processing systems, 29.

[29] ActivMedia Robotics, LLC. (2001). Pioneer 2 / PeopleBot Operations Manual.

[30] "Java Implementation of an indoor robot navigation - Pioneer 2DX". Dr. Paul Muntean. YouTube. (2012) https://www.youtube.com/watch?v=v8s15-7WqnM

[31] "How to 'glue' a camera to a pioneer 2DX robot". VansFannel. Gazebo Answers. https://answers.gazebosim.org//question/19062/how-to-glue-a-camera-to-a-pioneer-2dx-robot/

[32] "Training Configuration File - ML-Agents Documentation". Unity Technologies. Github. https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Training-Configuration-File.md

[33] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.

[34] Bjorck, J., Gomes, C. P., & Weinberger, K. Q. (2021). Is high variance unavoidable in rl? a case study in continuous control. arXiv preprint arXiv:2110.11222.

[35] Luo, W., Sun, P., Zhong, F., Liu, W., Zhang, T., & Wang, Y. (2018, July). End-to-end active object tracking via reinforcement learning. In International conference on machine learning (pp. 3286-3295). PMLR.

[36] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[37] "Xacro - ROS Index". Open Robotics. https://index.ros.org/p/xacro/

[38] "URDF - ROS Wiki". Open Robotics. http://wiki.ros.org/urdf

[39] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2021). Nerf: Representing scenes as neural radiance fields for view synthesis. Communications of the ACM, 65(1), 99-106.

[40] Li, S., Li, C., Zhu, W., Yu, B., Zhao, Y., Wan, C., ... & Lin, Y. (2023, June). Instant-3D: Instant Neural Radiance Field Training Towards On-Device AR/VR 3D Reconstruction. In Proceedings of the 50th Annual International Symposium on Computer Architecture (pp. 1-13).

[41] Pearce, A., Ghandeharioun, A., Hussein, N., Thain, N., Wattenberg, M. & Dixon, L. (2023, August). Do Machine Learning Models Memorize or Generalize? People + AI Research (PAIR).

[42] Notsawo Jr, P., Zhou, H., Pezeshki, M., Rish, I., & Dumas, G. (2023). Predicting Grokking Long Before it Happens: A look into the loss landscape of models which grok. arXiv preprint arXiv:2306.13253.

[43] Michaud, E. J., Gleave, A., & Russell, S. (2020). Understanding learned reward functions. arXiv preprint arXiv:2012.05862.

[44] Kerbl, B., Kopanas, G., Leimkühler, T., & Drettakis, G. (2023). 3d gaussian splatting for real-time radiance field rendering. ACM Transactions on Graphics (ToG), 42(4), 1-14.

[45] Khandate, G., Shang, S., Chang, E. T., Saidi, T. L., Adams, J., & Ciocarlie, M. (2023). Sampling-based Exploration for Reinforcement Learning of Dexterous Manipulation. arXiv preprint arXiv:2303.03486.

[46] Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V., & Scaramuzza, D. (2023). Champion-level drone racing using deep reinforcement learning. Nature, 620(7976), 982-987.

Anexo

.1. Archivo de configuración utilizado en el modelo final

```
default_settings: null
behaviors:
  RollerBall:
    trainer_type: sac
    hyperparameters:
      learning_rate: 0.0001
      learning_rate_schedule: constant
      batch_size: 256
      buffer_size: 100000
      buffer_init_steps: 0
      tau: 0.005
      steps_per_update: 9.0
      save_replay_buffer: false
      init_entcoef: 0.5
      reward_signal_steps_per_update: 9.0
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 3
      vis_encode_type: nature_cnn
      memory: null
      goal_conditioning_type: hyper
      deterministic: false
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
    init_path: null
    keep_checkpoints: 5
    checkpoint_interval: 500000
    max_steps: 20000000
    time_horizon: 1000
    summary_freq: 10000
    threaded: true
    self_play: null
    behavioral_cloning: null
env_settings:
  env_path: builds/SACModelFINAL.x86_64
  env_args: null
  base_port: 5005
  num_envs: 4
  num_areas: 1
  seed: -1
```

```
52   max_lifetime_restarts: 10
53   restarts_rate_limit_n: 1
54   restarts_rate_limit_period_s: 60
55 engine_settings:
56   width: 84
57   height: 84
58   quality_level: 5
59   time_scale: 20.0
60   target_frame_rate: -1
61   capture_frame_rate: 60
62   no_graphics: false
```