UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

TRABAJO FIN DE MÁSTER

DISEÑO E IMPLEMENTACIÓN DE UN ENTORNO SIMULADO PARA EXPERIMENTOS EN LA INTERACCIÓN HUMANO-ROBOT CON UN BRAZO HUMANO Y UN MANIPULADOR ROBÓTICO

MÁSTER UNIVERSITARIO EN INGENIERÍA MECATRÓNICA

AUTOR: ÓSCAR PONS FERNÁNDEZ

TUTOR: JUAN ANTONIO FERNÁNDEZ MADRIGAL

COTUTOR: VICENTE MANUEL ARÉVALO ESPEJO

MÁLAGA, JUNIO 2025

ESCUELA DE INGENIERÍAS INDUSTRIALES, UNIVERSIDAD DE MÁLAGA

DISEÑO E IMPLEMENTACIÓN DE UN ENTORNO SIMULADO PARA EXPERIMENTOS EN LA INTERACCIÓN HUMANO-ROBOT CON UN BRAZO HUMANO Y UN MANIPULADOR ROBÓTICO

Autor: Óscar Pons Fernández

Tutor: Juan Antonio Fernández Madrigal

Cotutor: Vicente Manuel Arévalo Espejo

Departamento: Ingeniería de Sistemas y Automática

Titulación: Máster en Ingeniería Mecatrónica

Palabras clave: Aprendizaje por refuerzo, simulación, robótica, SAC, Q-learning.

Resumen

Este Trabajo Fin de Máster presenta el desarrollo de un entorno de simulación orientado al entrenamiento de agentes mediante aprendizaje por refuerzo profundo (*Deep Reinforcement Learning* o DRL) para tareas de interacción física entre un manipulador robótico y un brazo humano simulado. El objetivo principal ha sido crear un *framework* modular y escalable que permita experimentar con estrategias de control aplicables en el ámbito de la rehabilitación física asistida.

Se ha integrado el manipulador *Franka Emika Panda* y el modelo biomecánico `MyoArm` utilizando el motor de simulación física *MuJoCo*, junto con librerías como `dm_robotics_panda`, `myo_sim` y `Stable Baselines3`. El sistema se ha validado realizando el entrenamiento de un agente DRL utilizando el algoritmo *Soft Actor-Critic* (SAC), el cual es capaz de seguir trayectorias predefinidas respetando unos umbrales de dolor o incomodidad física, medidos en magnitudes de fuerza y par.

Este trabajo de fin de estudios ha sido financiado por el proyecto de investigación "Tyrell: Time-Ready Reinforcement Learning for Robotic Skills and Tasks", código PID2023-147392NB-I00, por MICI-U/AEI/10.13039/501100011033 y los fondos FEDER de la Unión Europea, así como por los proyectos "CONCERTO" (PID2021-PID2021-127221OB-I00) y "RollGrip" (AT21_00051).

ESCUELA DE INGENIERÍAS INDUSTRIALES, UNIVERSIDAD DE MÁLAGA

DESIGN AND IMPLEMENTATION OF A SIMULATED ENVIRONMENT FOR DEVELOPING HUMAN-ROBOT INTERACTION EXPERIMENTS WITH BOTH A HUMAN AND A ROBOTIC ARM

Author: Óscar Pons Fernández

Supervisor: Juan Antonio Fernández Madrigal

Co-supervisor: Vicente Manuel Arévalo Espejo

Department: Systems Engineering and Automation

Degree: Master's Degree in Mechatronics Engineering

Keywords: Reinforcement learning, simulation, robotics, SAC, Q-learning.

Abstract

This Master Thesis presents the development of a simulation environment oriented to agent training through Deep Reinforcement Learning (DRL) for physical interaction tasks between a robotic manipulator and a simulated human arm. The main objective has been to create a modular and scalable framework that allows experimenting with control strategies applicable in the field of assisted physical rehabilitation.

The *Franka Emika Panda* manipulator and the biomechanical model `MyoArm` have been integrated using the *MuJoCo* physics engine, along with libraries such as `dm_robotics_panda`, `myo_sim`, and `Stable Baselines3`. The system was validated by training a DRL agent using the *Soft Actor-Critic* (SAC) algorithm, which is capable of following predefined trajectories while respecting thresholds of pain or physical discomfort, measured in terms of force and torque magnitudes.

This Master Thesis has been funded by the research project "Tyrell: Time-Ready Reinforcement Learning for Robotic Skills and Tasks", code PID2023-147392NB-I00, by MICI-U/AEI/10.13039/501100011033 and RDEF of the European Union, as well as by the projects "CONCERTO" (PID2021-PID2021-127221OB-I00) y "RollGrip" (AT21_00051).

Acrónimos

CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EII	Escuela de Ingenierías Industriales
HRI	Human-Robot Interaction
ML	Machine Learning
NN	Neural Network
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SB3	Stable Baselines 3
TFM	Trabajo Fin de Máster
UMA	Universidad de Málaga

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción y visión general

Contenido

En este capítulo se describirá la motivación y los objetivos que han guiado este proyecto. Previamente, se realizará una breve descripción del área del aprendizaje automático (*Machine Learning* o ML) denominado aprendizaje por refuerzo (*Reinforcement Learning* o RL), el cual se abordará con detalle en futuros capítulos. Finalmente, se expondrán las fases en que se ha dividido el trabajo y la estructura de la memoria.

1.1. Preliminares

El aprendizaje por refuerzo es un enfoque de aprendizaje automático inspirado en la psicología conductista. La característica distintiva del RL es el aprendizaje por ensayo y error sin requerir datos previamente recopilados ni etiquetados como necesitan otros enfoques del ML [16].

Los ejemplos de inteligencia artificial capaces de aprender entran dentro del marco del aprendizaje automático. A su vez, el *machine learning* comprende los siguientes subcampos [16]:

- **Aprendizaje supervisado** (*Supervised Learning* o SL): consiste en entrenar modelos a partir de datos etiquetados. Un ejemplo típico es el reconocimiento de dígitos escritos a mano, donde se proporcionan imágenes con sus respectivas etiquetas para que el modelo aprenda a clasificar nuevas correctamente.
- **Aprendizaje no supervisado** (*Unsupervised Learning* o UL): se basa en el uso de datos no etiquetados, con el objetivo de descubrir patrones. Un caso común es la segmentación de clientes, donde el modelo agrupa a los usuarios en función de similitudes en los datos.
- **Aprendizaje por refuerzo** (*Reinforcement Learning* o RL): se fundamenta en el aprendizaje mediante prueba y error, sin necesidad de datos etiquetados ni recopilados previamente por humanos. Su objetivo principal es aprender a actuar, como en el caso de un modelo que aprende a jugar a un videojuego interactuando con un emulador y observando las consecuencias de sus acciones.

En el caso de emplearse aproximaciones de funciones no lineales multicapa, generalmente redes neuronales, en cualquiera de los subcampos de ML mencionados, se estaría entrando en el contexto del aprendizaje profundo (*Deep Learning* o DL). La rama que unifica las técnicas de aprendizaje por refuerzo con las redes neuronales para la resolución de tareas de aprendizaje automático es denominada aprendizaje por refuerzo profundo (*Deep Reinforcement Learning* o DRL). Se recomienda visualizar la figura 1.1 para comprender con mayor claridad el contexto descrito.

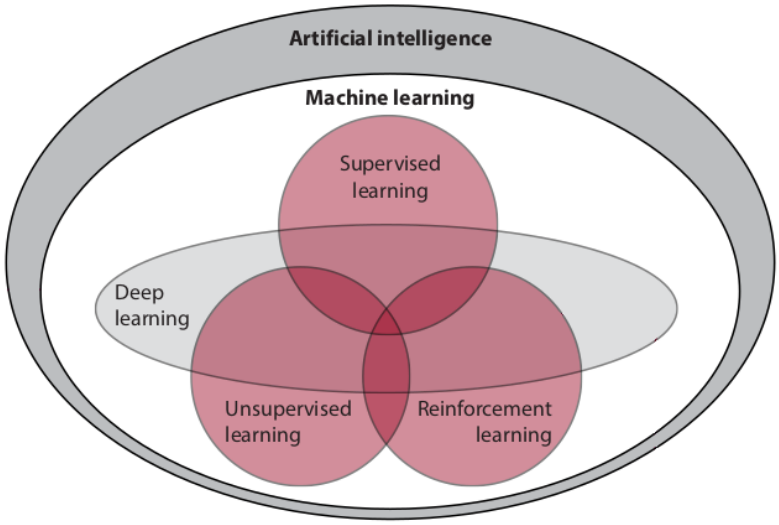


Figura 1.1: Subcampos del aprendizaje automático [16].

1.2. Objetivos

El objetivo de este trabajo es el desarrollo de un entorno de simulación que permita estudiar y experimentar la interacción física entre un brazo humano y un manipulador robótico. Este entorno servirá como plataforma para entrenar y validar estrategias de control basadas en aprendizaje por refuerzo, con especial interés en aplicaciones de asistencia física y rehabilitación.
De forma preliminar se describen los siguientes objetivos específicos:

- Integración de las diferentes librerías a emplear para el desarrollo del entorno de simulación.
- Recreación de un entorno de simulación que imite la estructura y la física de entorno real.
- Entrenar y validar un agente de DRL capaz de abordar aplicaciones dentro del ámbito de la asistencia física y la rehabilitación.

1.3. Motivación

La rehabilitación física asistida por robots representa una prometedora aplicación dentro del campo de la robótica colaborativa, especialmente en contextos donde la demanda de atención terapéutica supera la disponibilidad de personal médico especializado [26]. La interacción con extremidades humanas ha sido responsabilidad exclusiva de fisioterapeutas y médicos, lo que limita su escalabilidad y continuidad fuera del entorno clínico. Esto ha generado un interés importante en torno a la posibilidad de planificar de manera autónoma trayectorias para rehabilitación.

Sin embargo, el aprendizaje por refuerzo ha tenido un éxito limitado en el marco de la interacción humano-robot (*Human Robot Interaction* HRI) [23] en comparación con otros ámbitos de la robótica. Especialmente en tareas que requieren que el robot colabore físicamente con humanos.

Uno de los principales retos de la aplicación del RL a la HRI consiste en recopilar experiencias interactivas realistas con humanos, las cuales no pueden obtenerse directamente en el mundo real debido al elevado número de horas que requiere el entrenamiento de un agente y la peligrosidad que envuelve el proceso. De estos hechos subyace la necesidad de construir modelos humanos de alta fidelidad para simulaciones. Esta necesidad encaja perfectamente con el objetivo

descrito de desarrollar un entorno de simulación para estudiar la interacción física entre un brazo humano y un manipulador robótico.

El uso de DRL permite al robot adaptar sus movimientos en función del estado actual del paciente, lo cual es especialmente útil en contextos de rehabilitación [26], donde la condición física del paciente varía notablemente, incluso, en una misma sesión, en el rango de movimiento o en el nivel de fuerza, lo cual resulta difícil de modelar mediante métodos clásicos. Además, como se ha comentado, al integrar la interacción física entre un manipulador robótico y un brazo humano dentro de un entorno simulado, es posible experimentar, ajustar y validar estrategias de control en condiciones seguras antes de trasladarlas al entorno real.

1.4. Fases

El desarrollo del proyecto se ha estructurado en las siguientes fases:

- Estudio de la estructura, funcionamiento y capacidades de las diferentes librerías que se integrarán para conformar el entorno de simulación.
- Integrar con éxito el brazo manipulador junto con el humano en un entorno de simulación físicamente realista. Para ello, se empleará una programación orientada a objetos de forma que el código generado sea portable y escalable.
- Validación de la correcta funcionalidad del sistema mediante una demo básica antes de comenzar a plantear los entrenamientos de agentes empleando un enfoque basado en DRL.
- Estudio del RL poniendo el foco en su variante "*deep*", es decir, aquella que emplea redes neuronales profundas. También se revisará el estado del arte en el marco de la interacción humano-robot, centrándose en tareas de rehabilitación con manipuladores robóticos.
- Entrenar un agente empleando DRL capaz de ejecutar ejercicios consistentes en seguir una trayectoria predefinida, teniendo en cuenta la correcta ejecución del ejercicio por parte del paciente y su bienestar.
- Validar el modelo y recopilar datos para la redacción de la memoria de trabajo.
- Elaboración de manuales de despliegue y uso, junto con la preparación de los repositorios públicos que contendrán todos los recursos necesarios para

la replicación del sistema, incluyendo además código de ejemplo sobre el cual basar nuevas implementaciones.

1.5. Estructura de la memoria

La memoria se ha estructurado en seis capítulos, incluyendo el presente capítulo de introducción, y un anexo. Cada sección desarrolla de forma detallada un aspecto concreto del trabajo realizado. A continuación, se presenta un resumen conciso del contenido que se aborda en los capítulos posteriores:

- **Capítulo 2: Aprendizaje por refuerzo profundo.** En este capítulo se presentan las definiciones, conceptos y algoritmos clave que conforman el marco teórico del Aprendizaje por Refuerzo Profundo (DRL), con un recorrido progresivo que va desde métodos tabulares hasta algoritmos modernos como DDPG y SAC.
- **Capítulo 3: Desarrollo del entorno de simulación físicamente realista.** Se describe el diseño e implementación del entorno de simulación. Se detallan las librerías utilizadas, como `dm_robotics_panda`, `myo_sim`, `rl_spin_decoupler` y `Stable Baselines3`, así como la arquitectura del sistema y su diseño.
- **Capítulo 4: Prueba de uso del *framework*.** En este capítulo se formaliza la tarea de interacción físico-colaborativa entre el brazo humano y el manipulador robótico, que será resuelta mediante el entrenamiento de un agente basado en el algoritmo SAC. Se establecen las métricas de evaluación a partir del análisis de los umbrales de fuerza y par medidos en la muñeca del manipulador simulado. A continuación, se validan los resultados obtenidos a través de experimentos basados en trayectorias predefinidas, y se detallan las observaciones, acciones y la función de recompensa utilizadas en el entrenamiento del agente.
- **Capítulo 5: Análisis de resultados.** Se presentan los resultados obtenidos durante el entrenamiento del agente, analizando los hiperparámetros empleados y evaluando el comportamiento del modelo entrenado.
- **Capítulo 6: Conclusiones y líneas futuras.** Se exponen las conclusiones del trabajo y se proponen posibles líneas de trabajo futuro para ampliar y mejorar el sistema desarrollado.

- **Apéndices.** Se incluye un manual de uso detallado, que describe los pasos necesarios para replicar el entorno de simulación, entrenar al agente y visualizar los resultados.

Capítulo 2

Aprendizaje por refuerzo profundo

Contenido

En este capítulo se presentan las definiciones y formalismos necesarios para comprender el problema del aprendizaje por refuerzo. Posteriormente, se expone una evolución progresiva de algoritmos fundamentales en el campo del *Deep Reinforcement Learning* (DRL), partiendo desde enfoques tabulares hasta métodos modernos adaptados a entornos continuos como los que se encuentran en robótica.

Se comienza con *Q-learning* [27], un método clásico basado en valores que permite aprender la función óptima de acción-valor en dominios discretos y sin incertidumbre. A continuación, se introduce DQN (*Deep Q-Network*) [15], que extiende *Q-learning* con el uso de redes neuronales para tratar problemas de alta dimensionalidad y con ruido. Este algoritmo marca la entrada al DRL como tal.

Posteriormente, se aborda DDPG (*Deep Deterministic Policy Gradient*) [13], un algoritmo que permite trabajar en espacios de acción continuos mediante una arquitectura actor-crítico (*actor-critic*), que constituye un puente clave entre los métodos basados en valores y los basados en política.

Finalmente, se analiza SAC (*Soft Actor-Critic*) [10], un método moderno que incorpora técnicas de aprendizaje con máxima entropía para lograr políticas más estables y exploratorias, representando el estado del arte en control continuo dentro del DRL.

2.1. Definiciones y conceptos

2.1.1. Definiciones

A continuación, se presentan algunas de las definiciones más comunes dentro del ámbito del RL y a las que se hará referencia con frecuencia a lo largo de la memoria. Con el fin de ilustrar algunas definiciones, se supondrá que se está entrenando un manipulador robótico para realizar una tarea de recogida y colocación (*pick and place*).

- **Agente:** es alguien o algo que interactúa con el entorno ejecutando acciones que modifican el entorno en el que se encuentra, recibiendo unas observaciones y eventuales recompensas por ello [12]. En el ejemplo propuesto, sería la parte del código encargado de decidir el movimiento del manipulador.
- **Entorno:** es todo aquello que no incluye al propio agente. El agente no tiene control sobre el entorno, interactúa con él a través de las acciones [12]. En el ejemplo propuesto, el entorno serían los objetos a manipular, la mesa de trabajo e incluso el propio brazo robótico.
- **Observaciones:** información sobre el estado del entorno que el agente puede recibir y emplear para tomar sus decisiones [12]. En el ejemplo propuesto, podría tratarse de las imágenes obtenidas por una cámara o datos de otros sensores.
- **Estados:** es el conjunto de valores que describen el sistema en un instante temporal concreto [16]. A su vez, el espacio de estados es el conjunto de variables empleadas para describirlos y todos los valores posibles que pueden tomar. El espacio de estados puede ser finito o infinito y continuo o discreto. En el ejemplo propuesto podrían ser la velocidad y posición del

efecto final, el número, forma y ubicación de los objetos dispuestos en el espacio de trabajo, etc. Se denotará el estado con la letra s y el espacio de estados con la letra S.

- **Acciones:** permiten al agente interactuar con el entorno. A su vez, el espacio de acciones es el conjunto de posibles acciones. Estas pueden variar de un estado a otro, pero la convención es utilizar el mismo conjunto para todos los estados [16]. Además, el espacio de acciones puede ser continuo o discreto. En el ejemplo propuesto, el espacio de acciones, continuo, podrían ser las velocidades cartesianas del efector final. En la literatura es común hacer referencia a las acciones con la letra a, y al espacio de acciones con A.

- **Señal de recompensa:** codifica el objetivo que debe alcanzar el agente. La señal de recompensa, en su forma más general, asigna un escalar que indica la bondad de la transición a un estado s' desde un estado s al ejercer la acción a [16]. Se denominará a la recompensa con la letra r. En el ejemplo propuesto, la recompensa podría ser +1 cuando coloca un objeto de forma exitosa y 0 en caso contrario.

- **Función de transición:** indica cómo evoluciona el entorno y cómo reacciona a las acciones del agente. Más concretamente, la función de transición vincula un estado (un estado siguiente s') a un par estado-acción, y define la probabilidad de alcanzar ese estado futuro dado el par estado-acción [16].

- **Paso (*step*):** las interacciones entre el agente y el entorno se prolongan durante varios pasos o ciclos. Cada ciclo tiene una duración en los sistemas físicos (*time step*). En cada paso, el agente observa el entorno, actúa y recibe una nueva observación y recompensa [16].

- **Experiencia (*experience*):** es como se denomina al conjunto o tupla formada por un estado, una acción, una recompensa y el siguiente estado $[s, a, r, s']$ [16].

- **Episodio (*episode*):** La tarea que el agente intenta resolver puede tener, o no, un final natural. Las tareas que tienen un final natural, como un juego, se denominan tareas episódicas. Por el contrario, las tareas que no lo tienen se denominan tareas continuas, como aprender a caminar hacia delante. La secuencia de pasos temporales desde el principio hasta el final de una tarea episódica se denomina episodio [16].

- **Factor de descuento (*discount factor*):** es un valor real positivo inferior o igual a uno empleado para descontar exponencial o linealmente el valor de las recompensas futuras. Cuanto más lejano sea el horizonte temporal en que se recibe la recompensa, menos valiosa se puede considerar que es en el presente. El factor de descuento es por tanto empleado para ajustar la importancia de las recompensas a lo largo del tiempo [16]. Se denominará este valor escalar como γ.

- **Retorno (*return*):** es el sumatorio de las recompensas obtenidas en los diferentes *steps* de un episodio multiplicado por el factor de descuento [16]. Se denominará con la letra G:

$$G_0 = \sum_{t=0}^{\infty} \gamma^t r_t$$

- **Política (*policy*):** Encontrar la mejor es la tarea principal del agente. Las políticas son planes que cubren todos los estados posibles. Estas pueden ser estocásticas o deterministas, en función de si devuelve una distribución de probabilidad para cada acción o un valor concreto dado un estado u observación [16]. En el ejemplo propuesto, la política sería aquella función que dado un estado es capaz de inferir las velocidades cartesianas a ejecutar en el efector final del manipulador. Se hará referencia a este concepto como $\pi(s)$.

- **Función de valor (*value function* o *V-function*):** precisa la expectativa $\mathbb{E}$ de retorno (*returns* o G) que el agente recibirá si sigue una política π desde un estado s [16]. Se define como:

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid S_t = s \right]$$

- **Función de acción-valor (*action-value function* o *Q-function*):** precisa la expectativa de retorno si el agente sigue la política π tras tomar la acción a en un estado s [16]. Se define como:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid S_t = s, A_t = a \right]$$

2.1.2. Tipos de agentes

Los agentes se pueden clasificar atendiendo a la función que tratan de aproximar (su función objetivo) [16]:

- Los agentes diseñados para aproximar políticas se denominan basados en políticas (*policy-based*).
- Los agentes diseñados para aproximar funciones de valor se denominan basados en valores (*value-based*).
- Los agentes diseñados para aproximar tanto políticas como funciones de valor se denominan actor-crítico (*actor-critic*).

2.1.3. Aprendizaje por refuerzo

En la figura 2.1 se muestra una iteración de un bucle de aprendizaje por refuerzo. Este ciclo comienza con el agente recibiendo las observaciones y la recompensa de la iteración anterior; con ellos se alimenta al algoritmo encargado de hacerle mejorar en la tarea a aprender. El siguiente paso es decidir qué acción tomar y ejecutarla, lo que hace que el entorno transicione a otro estado. A partir de este punto, el bucle se repite hasta que finalice el episodio o se alcance una situación de terminación.

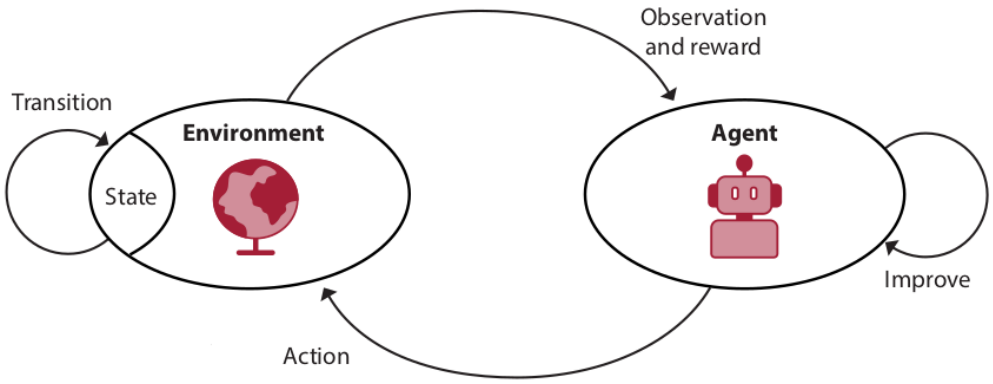


Figura 2.1: Ilustración de una iteración del bucle de aprendizaje por refuerzo [16].

Con las definiciones mostradas y el esquema de la figura 2.1 se empieza a discernir que la clave del aprendizaje por refuerzo está en que el agente encuentre una función capaz de mapear unas entradas (estado u observaciones) a unas salidas (acciones). Es decir, encontrar una política. Además, para ello solo

puede interactuar con el entorno de manera repetida. Esto claramente plantea un nuevo problema: es necesario establecer una métrica que indique cómo de buena o mala es la política del agente. La métrica empleada será el retorno que el agente obtendrá siguiendo esa política, es decir, la *V-function* o la *Q-function*. Estas funciones generalmente no son conocidas por el agente, por lo que tiene que estimarlas mientras aprende la política.

A continuación, se van a plantear dos métodos que tratan de obtener la mejor estimación posible del retorno esperado siguiendo una política, que, por el momento, se supondrá conocida (π) :

Método de Monte Carlo *(Monte Carlo Learning o MC)*:
Los métodos de Monte Carlo se basan en recolectar experiencias completas, es decir, trayectorias desde el estado inicial hasta que se alcanza un estado terminal. Cada una de estas trayectorias se denomina un episodio. La experiencia se representa como una tupla de la forma: `[estado, acción, recompensa, estado siguiente, final]`, donde `final` es un valor booleano que indica si se ha llegado al estado terminal.

Se define una tasa de aprendizaje o *learning-rate* (α) que puede ser una constante o un valor decreciente (exponencial o lineal). Una vez finalizado un episodio, que tiene lugar en el intervalo de tiempo $[t, T]$,se calcula el retorno obtenido ($G_{t:T}$). Es un método iterativo, que, "en infinito", converge a la $v_\pi(S)$ real. Si se itera un número finito de episodios, lo que se obtiene es una aproximación ($V_T(S)$).

El problema de los métodos basados en MC es que requieren de la ejecución de un episodio completo para recalcular la función de valor (V_T), por ello las variaciones entre las estimaciones de una iteración y otra son notables y el método oscila bastante, especialmente al inicio del entrenamiento. Por el contrario, no tiene sesgo en sus estimaciones.

El método de actualización de $V_T(S)$ ejecutado tras cada episodio se define a continuación:

$$V_T(S) = V_{T-1}(S) + \alpha \left(G_{t:T} - V_{T-1}(S)\right)$$

dónde el retorno ($G_{t:T}$) sería el objetivo del método (*MC target*) y el error de estimación o (*MC error*) sería $G_{t:T} - V_{T-1}(S)$.

Método de diferencias temporales *(Temporal-Difference Learning o TD)*:
Para minimizar la oscilación surgen los métodos TD, que actualizan la *V-function* en cada iteración. Para ello, estando en el estado s_t, el agente realiza una acción a_t, obtienen la recompensa r_{t+1} de ese paso. Con esto se predice el retorno que obtendrían a partir del estado siguiente estado ($V_t(s_{t+1})$). Así se consigue optimi-

zar la *state-value function* en cada iteración de cada episodio. Además, no tienen tanta varianza en sus estimaciones porque en cada paso se introduce menos información y, por tanto, ruido.

El problema de estos métodos es que convergen más lento y su estimación suele presentar un sesgo respecto al valor ideal a alcanzar, debido a que estiman los valores futuros utilizando otras estimaciones en lugar de las verdaderas recompensas futuras. Es decir, usan un valor estimado para predecir otro valor estimado. Este enfoque se llama *bootstraping*.

El método de actualización de la *state-value function* ejecutado tras cada paso se define a continuación:

$$V_{t+1}(s_t) = V_t(s_t) + \alpha \left(r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)\right)$$

dónde el objetivo (*TD target*) recoge la recompensa del paso actual y la expectativa de retorno desde el estado $t+1$: $r_{t+1} + \gamma V_t(s_{t+1})$. El error de estimación o *TD error* sería el *TD target* menos la estimación desde el estado actual: $r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$.

2.1.4. El dilema de la exploración contra la explotación

Quedaría por definir cómo el agente decide qué acción tomar en métodos basados en valor. Una opción sería tomar aquella acción que para un estado s_t maximice la expectativa de retorno: $a = \arg\max_a Q(s, a)$. Esta sería una estrategia que favorece la explotación (maximizar el retorno) pero que no da lugar a la exploración (experimentar nuevas acciones en el entorno). El caso contrario sería una estrategia de exploración, donde en cada paso se escoge una acción aleatoria. Esto plantea uno de los grandes dilemas dentro del RL: el balance entre exploración y explotación. Es aconsejable que el agente busque nuevas alternativas (tomar acciones diferentes), ya que de esta forma hallará qué acciones son más eficaces en cada estado, pero una vez las encuentre se quiere que utilice ese conocimiento adquirido (explotación).

Existen múltiples formas de abordar este problema; en este trabajo se hablará de la estrategia *epsilon greedy*, ya que, por su simplicidad y eficacia, es comúnmente empleada. El planteamiento es simple, como se muestra en el algoritmo 1.

La variable ϵ es un valor real positivo inferior o igual a uno que decae exponencial o linealmente en cada episodio. De esta forma, en los primeros instantes del aprendizaje se favorece la exploración, mientras que en los últimos episodios se favorece la explotación.

Algorithm 1 Selección de acción con estrategia ϵ-greedy

Input: Estado actual s, función de valor $Q(s, a)$, parámetro ϵ
Generar un número aleatorio $x \sim \mathcal{U}(0, 1)$
if $x < \epsilon$ **then**
    Seleccionar una acción a al azar (exploración)
else
    Seleccionar $a = \arg\max_a Q(s, a)$ (explotación)
end if
return a

2.2. *Q-Learning*

Q-Learning [27] es un algoritmo de aprendizaje por refuerzo basado en valores (*value-based method*), ampliamente utilizado por su simplicidad y eficacia en entornos discretos. Se encuadra dentro de los métodos de aprendizaje por diferencias temporales *(Temporal-Difference Learning o TD)*.

Se trata de un algoritmo *off-policy*, lo que significa que el agente aprende la política óptima independientemente de la política que sigue para explorar el entorno. Esto permite que el agente utilice una política de comportamiento ($\epsilon - greedy$) para balancear la exploración y la explotación, mientras actualiza su estimación de valores según la mejor acción posible en el siguiente estado, no necesariamente la acción que realmente toma.

Q-Learning trabaja en entornos donde el espacio de estados y acciones es finito y discreto. La política óptima se construye indirectamente mediante la iteración sobre una función *Q*, que se va actualizando en función de las recompensas observadas y la estimación futura de recompensa.

Durante el proceso de aprendizaje, el agente interactúa con el entorno a lo largo de múltiples episodios. En cada paso, selecciona una acción usando una estrategia de exploración ($\epsilon - greedy$), ejecuta dicha acción, y observa la recompensa y el nuevo estado. Posteriormente, se actualiza el valor de $Q(s, a)$ aplicando la siguiente regla de aprendizaje:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \underbrace{\Big(\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{TD target}} - \underbrace{Q(s,a)}_{\text{Estimación actual}} \Big)}_{\text{TD error}}$$

Para ilustrar los conceptos descritos, se muestra el pseudocódigo del algorit-

mo *Q-Learning* en el algoritmo 2.

Algorithm 2 Algoritmo Q-learning

Require: Tasa de aprendizaje α, factor de descuento γ, probabilidad de exploración ϵ, número de episodios M

Inicializar la función de valor $Q(s, a)$ arbitrariamente
for episodio = 1 hasta M **do**
    Inicializar el estado s
    while el estado s no es terminal **do**
        Seleccionar una acción a con ϵ-greedy.
        Ejecutar la acción a, observar la recompensa r y el nuevo estado s'
        Actualizar:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

        Establecer $s \leftarrow s'$
    end while
end for

2.3. *Deep Q-Network* (DQN)

DQN [15] es una extensión del ya descrito algoritmo *Q-learning*, modificado para emplear redes neuronales profundas que aproximan la función de valor $Q(s, a)$. Es eficaz en entornos con espacios de estado extensos o continuos que resultan difíciles de manejar mediante una tabla. Pese a esto, el espacio de acciones debe ser finito y discreto. Al igual que *Q-Learning*, DQN es un método *off-policy*, lo que significa que el agente aprende una política distinta de la política que sigue para explorar el entorno.

DQN tuvo una gran repercusión cuando fue descrito en el artículo original [15]. En este se presentó un agente capaz de jugar a un nivel humano (incluso superior al humano en ciertas pruebas) a siete diferentes juegos de la videoconsola Atari 2600 [28]. Para ello se empleó el banco de pruebas de RL, *The Arcade Learning Environment (ALE)* [1], que presenta a los agentes una entrada visual de alta dimensión (vídeo RGB de 210 × 160 a 60 Hz) y un conjunto diverso de tareas que fueron diseñadas para ser difíciles para los jugadores humanos. Este nivel de capacidad se consiguió empleando una red neuronal convolucional (CNN) para estimar la *Q-function*, es decir, al agente se le proporcionaban *frames* en bruto procedentes de los videojuegos, de los cuales, tras ser propagados

por una CNN, se inferían las acciones a ejecutar en los videojuegos, emulando lo que haría un jugador humano.

Por muy prometedor que parezca, el hecho de introducir una red neuronal profunda aumenta la inestabilidad durante el aprendizaje; DQN propone dos mecanismos para mejorar la estabilidad [16]:

1. ***Replay Buffer*:** Se genera un *buffer* circular al que en cada paso se añade una experiencia (s, a, r, s'). En cada paso de actualización, DQN toma un *minibatch* aleatorio (siguiendo una distribución uniforme) de este *buffer* para entrenar la red. Con esta modificación los *batches* de experiencias empleados dejan de estar ordenados por instantes temporales correlativos $(t, t+1, t+2...)$; es decir, el *replay buffer* es un método empleado para combatir el problema de la correlación temporal entre iteraciones.

2. **Red Objetivo (*Target Network*):** DQN emplea dos redes neuronales: una red principal (*online network*) $Q(s, a; \theta)$ y una red objetivo (*target network*) $Q'(s, a; \theta^-)$, que se actualiza periódicamente con los parámetros de la red principal. La *target network* es la empleada para estimar el valor de $Q(s', a')$, lo que contribuye a la estabilidad y mejora la convergencia evitando el denominado problema de "seguir tu propia cola" o "*chasing your own tail*" que puede verse claramente descrito en la figura 2.2.

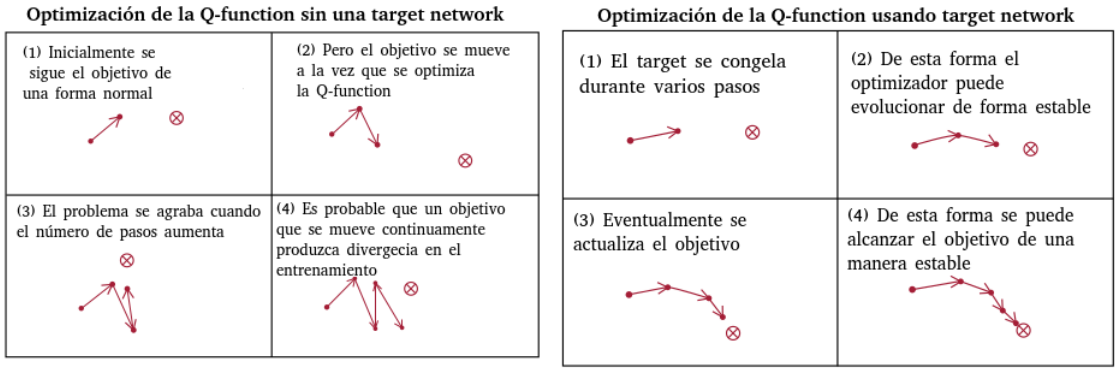


Figura 2.2: Ilustración del problema de "seguir tu propia cola" y cómo solventarlo mediante el empleo de la red objetivo [16].

Las variables de estado se propagan por una red neuronal (suele ser densa o convolucional); la red, a su salida, proporciona el valor de la *Q-function* para las distintas acciones posibles en el estado introducido. El proceso de entrenamiento es el siguiente:

1. Construir la red neuronal, sus distintas capas y funciones de activación.
2. Establecer una métrica de error (suele usarse el error cuadrático medio o MSE) y un optimizador (generalmente *RMSprop* [19] [15] [16]).
3. Decidir el método de exploración ($\epsilon - greedy$).
4. Realizar un número mínimo de iteraciones (`batch_size`) para añadir experiencias al *buffer*. Durante estos primeros pasos el modelo no se actualizará.
5. Tras `batch_size` iteraciones se pasa a entrenar la red neuronal con el *batch* de experiencias extraído del *buffer*. Este proceso de optimización se repite durante un número determinado de épocas y consta de los siguientes pasos:
   a) Se calcula $\max_a Q'(S_{t+1}, a)$ para todos los estados $t+1$ de las experiencias seleccionadas. En este paso se usa el *target-model* para garantizar la estabilidad.
   b) Con lo anterior se calcula el *TD-Target*: $R_{t+1} + \gamma \max_a Q'(S_{t+1}, a)$.
   c) Se calcula $Q(S_t, A_t)$ empleando el *online-model*.
   d) Se calcula el error como la diferencia de Q y el *target*:

$$Error = Q(S_t, A_t) - \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a)\right]$$

   e) A partir del conjunto de errores individuales, se calcula el error cuadrático medio (MSE) sobre el *batch*, integrando todas las experiencias seleccionadas del *replay buffer*.
   f) Se hace *backpropagation* [29] y se actualizan los pesos de la red *online* para disminuir dicho error.
6. Si han pasado un número de iteraciones determinado (`update_target_model`) se iguala el modelo *target* al *online*.
7. El proceso de entrenamiento se itera durante un número de episodios determinado.

En el algoritmo 3 se muestra el pseudocódigo del algoritmo DQN.

Algorithm 3 Deep Q-Network (DQN)

Input: Tasa de aprendizaje α, factor de descuento γ, exploración ϵ, episodios M, tamaño del minibatch `batch_size`, frecuencia de actualización de red objetivo `update_target_model`
Inicializar la red online con parámetros θ
Inicializar la red target Q' con parámetros $\theta^- \leftarrow \theta$
Inicializar la memoria de experiencia D vacía
for episodio = 1 hasta M **do**
    Inicializar estado s_1
    for paso = 1 hasta T **do**
        Seleccionar una acción a_t con ϵ-greedy.
        Ejecutar acción a_t, observar recompensa r_t y siguiente estado s_{t+1}
        Almacenar transición (s_t, a_t, r_t, s_{t+1}) en D
        Muestrear minibatch aleatorio de tamaño B desde D
        for all experiencia (s, a, r, s') en el minibatch **do**
            $target \leftarrow r + \gamma \max_{a'} Q'(s', a'; \theta^-)$
            Actualizar θ minimizando $(y - Q(s, a; \theta))^2$
        end for
        if (paso $\%$ `update_target_model`) = 0 **then**
            Actualizar red objetivo: $\theta^- \leftarrow \theta$
        end if
    end for
end for

En esta sección se ha descrito con detalle el algoritmo de aprendizaje por refuerzo profundo DQN, enfatizando las novedades como el uso de redes neuronales, un *buffer* de experiencias y el concepto de red objetivo (*target model*). Resulta intuitivo ver la similitud de DQN con el previamente estudiado *Q-learning*, pues la intención de estas secciones es ir construyendo conocimiento sobre los algoritmos previos. Para terminar, se muestra un diagrama de lo que sería el proceso de aprendizaje en DQN (ver figura 2.3).

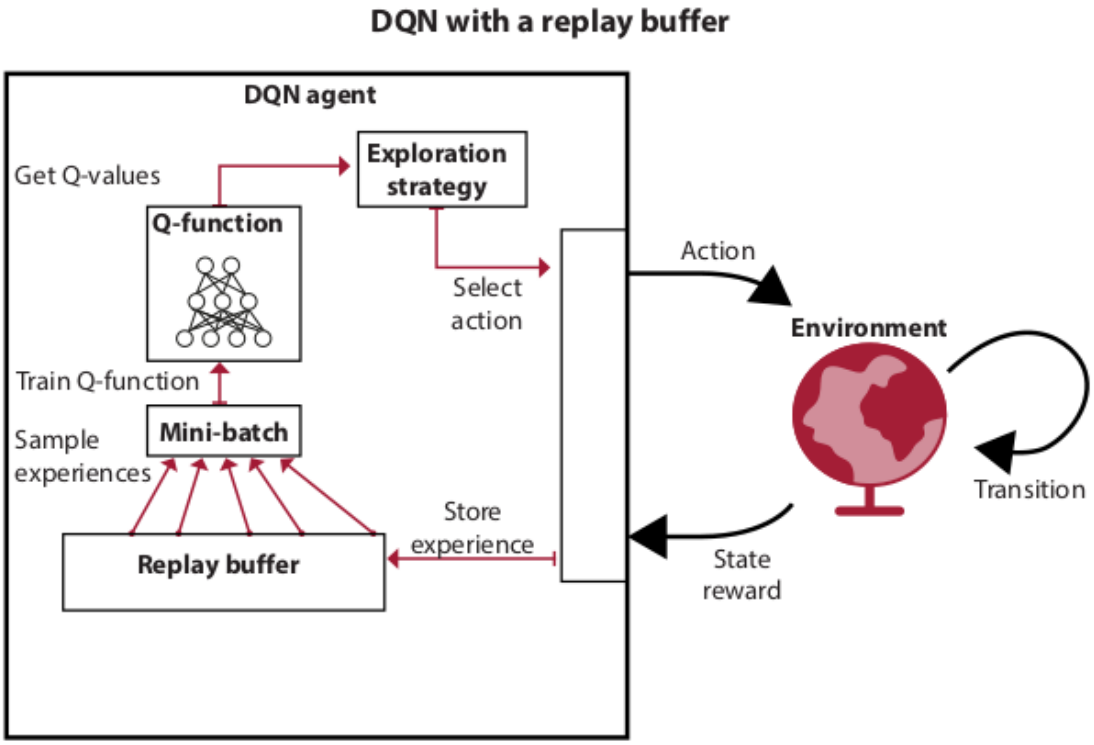


Figura 2.3: Diagrama del proceso de aprendizaje en DQN con *buffer* de experiencias [16].

2.4. *Deep Deterministic Policy Gradient* (DDPG)

DDPG (Deep Deterministic Policy Gradient) [13] es un método *actor-critic off-policy* diseñado específicamente para entornos con espacios de acción continuos. Recuérdese que, el concepto *actor-critic* hace referencia a que el agente tiene la capacidad de aprender tanto políticas como funciones de valor.

A diferencia de DQN, que solo puede aplicarse a espacios de acción discretos, DDPG permite el aprendizaje de políticas deterministas que mapean directamente estados (reales y continuos) a acciones (reales y continuas), de nuevo mediante el uso de redes neuronales profundas.

En este contexto, las redes utilizadas en DDPG son:

- **Red actor** $\mu(s|\theta^{\mu})$ (*actor network*), utilizada para predecir las acciones dado un estado. Para las pérdidas se calcula la media del retorno estimado por la red crítico *online* para las acciones inferidas por la red actor *online*. Este

valor se multiplica por -1 para que el optimizador, al tratar de disminuir la pérdida, realmente esté maximizando el retorno esperado. En este caso, el optimizador recomendado en el artículo original [13], y adoptado por otros autores [16] es ADAM [11].

- **Red crítico** $Q(s, a|\theta^Q)$ (*critic network*), utilizada para estimar el valor de la acción tomada. Su función de pérdidas es igual que en DQN, el error cuadrático medio. De nuevo, el optimizador recomendado en el artículo original [13] y adoptado por otros autores [16], es ADAM [11], a diferencia de *RMSprop* empleado en DQN.

En la figura 2.4 se puede observar la estructura de las redes actor-crítico empleadas en DDPG. Por mantener la nomenclatura de la figura y la simplicidad en la escritura, se hará referencia a la red actor como *Policy-NN* y a la crítico como *Q-NN*.

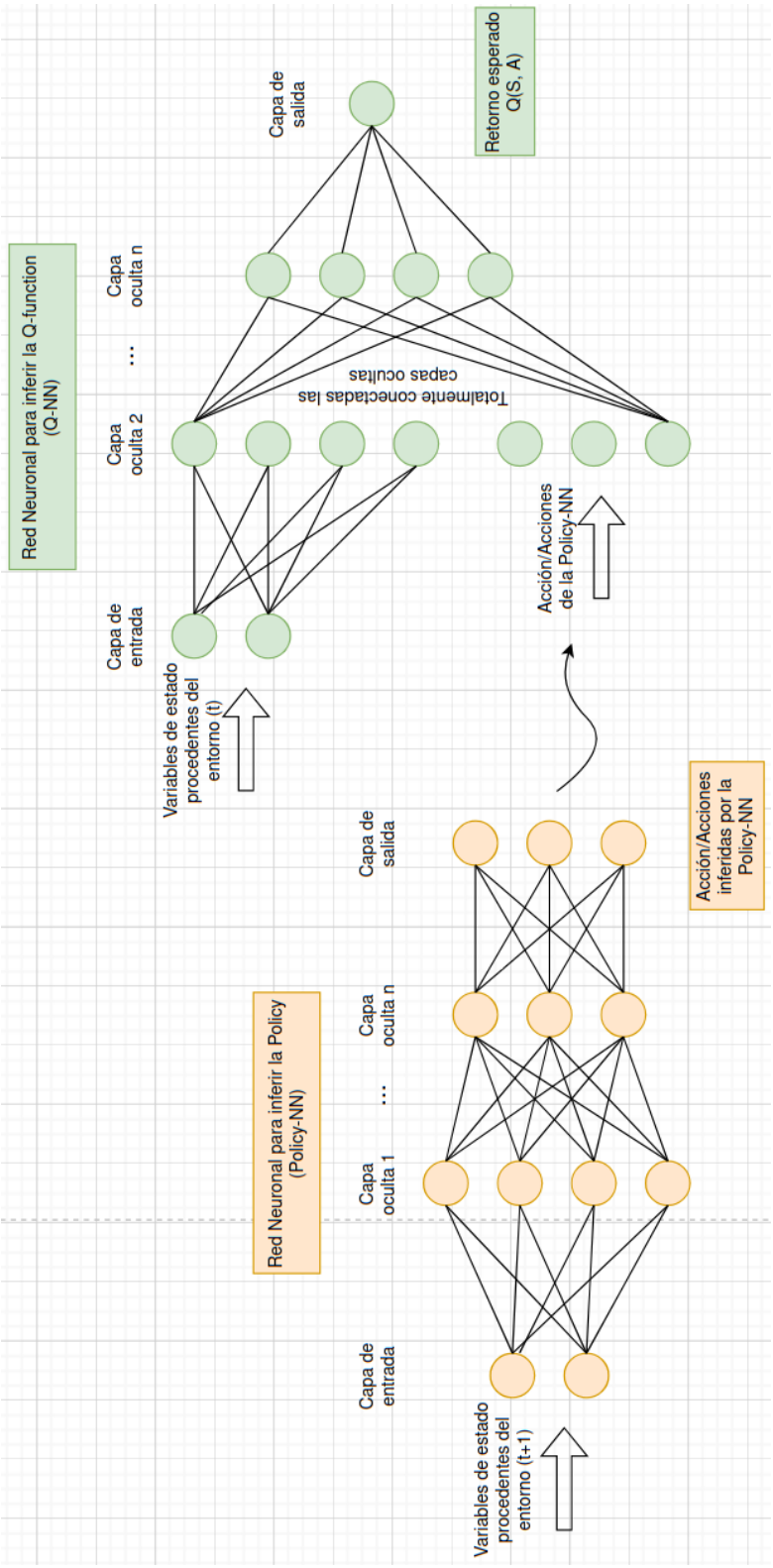


Figura 2.4: Estructura de las redes actor-crítico empleada en DDPG.

DDPG mantiene la técnica de red objetivo, tanto para la actor como la crítico, por lo que existirán un par de redes neuronales objetivo (*Target-Policy-NN* y *Target-Q-NN*) y un par *online* (*Online-Policy-NN* y *Online-Q-NN*). Hay que señalar que en la capa de entrada de las *Q-NN* solo se introducen las variables de estado; es en la primera capa oculta que se introducen los valores de las acciones para estimar el retorno. Todas las capas están totalmente conectadas a excepción de las neuronas de la primera capa oculta que hacen de entrada para las acciones y que no están conectadas con la capa de entrada. DDPG mantiene el *buffer* de experiencias o *replay buffer* introducido en DQN.

Finalmente, es importante notar que en este algoritmo no se decide un método de elección de acción, ya que la red actor se encarga aproximar la política. El problema reside en que esta red trata de maximizar el retorno (explotación), por lo que es necesario añadir una componente de exploración. Para ello, una práctica habitual es sumar una componente de ruido a las acciones inferidas por la red; esto ayuda a que el modelo "se equivoque" y explore nuevas acciones. En el artículo original se opta por utilizar un proceso Ornstein-Uhlenbeck [13]. Un enfoque más sencillo es añadir un ruido Gaussiano cuya componente disminuye en los sucesivos pasos; de esta forma se favorece la exploración en las primeras iteraciones y la explotación en las últimas [16].

A continuación se describe el proceso de entrenamiento para DDPG:

1. Construir las redes neuronales, sus distintas capas, funciones de activación. Configurar las funciones de pérdidas y optimizadores para la red del actor y para la del crítico.

2. Antes de comenzar con el entrenamiento, es necesario realizar un número mínimo de iteraciones (`batch_size`) para añadir experiencias al *buffer*; durante estos primeros pasos el modelo no se actualizará.

3. Tras `batch_size` iteraciones se pasa a entrenar la red neuronal con el *batch* de experiencias extraído del *buffer* (S_t, A, R_t, S_{t+1}). Este proceso de optimización se repite durante un número determinado de épocas y consta de los siguientes pasos:

   a) Se estiman las acciones para los estados futuros (S_{t+1}) con la *Target-Policy-NN* ($A'_{t+1} = \mu(S_{t+1}|\theta^{\mu-})$) y esto junto con los estados futuros se introduce en la *Target-Q-NN* para estimar el retorno. Con el retorno estimado de la red y las recompensas se calcula el *TD-Target*:

$$TD_{Target} = R_t + \gamma Q'(S_{t+1}, A'_{t+1}|\theta^{Q-})$$

b) Se estima el retorno esperado para los pares estado-acción de la experiencia con la *Online-Q-NN* ($Q(S_t, A|\theta^Q)$). Se calcula el error cuadrático medio de la resta del *TD-Target* y esta estimación para optimizar la *Online-Q-NN*.

$$TD_{Error} = Q(S_t, A|\theta^Q) - TD_{Target}$$

$$L_Q = \frac{1}{N}\sum_i (TD_{Error})^2$$

c) Lo siguiente es calcular las acciones con la *Online-Policy-NN* para los estados $t+1$ ($A_{t+1} = \mu(S_t, \theta^\mu)$). Estas acciones y los respectivos estados se introducen en la *Online-Q-NN* y se estiman los retornos ($Q(S_t, A_{t+1}|\theta^Q)$). La media de los retornos del *batch* multiplicada por -1 se emplea para optimizar la red *Online-Q-NN*.

$$Loss_{policy} = \frac{-1}{N}\sum_i Q(S_t, A_{t+1}|\theta^Q)$$

d) En DQN se actualizaba la red objetivo cuando pasaban un número de iteraciones determinado (`update_target_model`). Como se comentó anteriormente, daba estabilidad al entrenamiento al hacer que el *target* no oscilase, pero introduce un nuevo problema, y es que los valores del *target* se hacen más imprecisos iteración tras iteración hasta que se actualizan los modelos abruptamente igualando los pesos ($\theta^- = \theta$). En DDPG se hace uso de una solución que aporta progresividad y estabilidad, es el denominado *Polyak Averaging* [16] [13]. Con este método se fusionan los pesos de las redes progresivamente, lo cual no llega a hacer inestable el entrenamiento:

$$\theta^{Q-} = \tau\theta^Q + (1-\tau)\theta^{Q-}$$

$$\mu^{Q-} = \tau\mu^Q + (1-\tau)\mu^{Q-}$$

En el algoritmo 4 se muestra el pseudocódigo de DDPG.

Algorithm 4 Algoritmo DDPG

Inicializar las redes crítica $Q(s, a|\theta^Q)$ y actor $\mu(s|\theta^\mu)$ con pesos θ^Q y θ^μ
Inicializar las redes objetivo Q' y μ' con pesos $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Inicializar el buffer de repetición R
for episodio = 1 hasta M **do**
    Inicializar un proceso aleatorio $\mathcal{N}$ para la exploración de acciones
    Recibir el estado inicial s_1
    for $t = 1$ hasta T **do**
        Seleccionar acción $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ según la política actual y ruido de exploración
        Ejecutar acción a_t, observar recompensa r_t y nuevo estado s_{t+1}
        Almacenar la transición (s_t, a_t, r_t, s_{t+1}) en R
        Muestrear un minibatch aleatorio de N transiciones (S_t, A, R_t, S_{t+1}) desde R
        Calcular $TD_{target} = R_t + \gamma Q'(S_{t+1}, \mu'(s_{t+1}|\theta^{\mu-})|\theta^{Q-})$
        Actualizar el crítico minimizando la pérdida:
        $L_Q = \frac{1}{N}\sum_i (TD_{target} - Q(S_t, A|\theta^Q))^2$
        Actualizar la política del actor:
        $Loss_{policy} = \frac{-1}{N}\sum_i Q(S_t, \mu(S_t, \theta^\mu)|\theta^Q)$
        Actualizar las redes objetivo:
        $\theta^{Q-} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q-}$
        $\theta^{\mu-} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu-}$
    end for
end for

2.5. *Soft Actor-Critic* (SAC)

Soft Actor-Critic (SAC) [10] es un algoritmo de tipo *off-policy* basado en el enfoque *actor-critic*, diseñado para el aprendizaje en espacios de acción continuos. A diferencia de DDPG, que utiliza políticas deterministas, SAC emplea una política estocástica que maximiza tanto la recompensa esperada como la entropía de la política. El objetivo es fomentar la exploración efectiva mediante la incorporación de una componente de incertidumbre. La entropía se define como:

$$\mathcal{H}(X) = -\sum_{x \in X} p(x) \cdot \log(p(x))$$

Conceptualmente, la entropía proporciona la medida de incertidumbre en una distribución de probabilidad, permitiendo cuantificar la cantidad de “sorpresa” de

un conjunto de datos. Por ejemplo, en una distribución 50 % acción A y 50 % acción B, la entropía es muy alta ($\mathcal{H} = 0{,}301$) ya que no se puede saber (no es predecible) lo que va a pasar. Por el contrario, en una distribución 99 % acción A y 1 % acción B, la entropía es muy baja ($\mathcal{H} = 0{,}024$) ya que resulta sencillo predecir que se tomará la acción A [30].

A diferencia de la mayoría de algoritmos de RL, donde se trata de maximizar el retorno obtenido, en SAC se añade a este objetivo el concepto de entropía sobre la política. De esta forma se anima al agente a mantener una política estocástica, favoreciendo la exploración y ayudando a evitar mínimos locales.

$$J(\pi) = \sum_{t} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right]$$

donde el parámetro α es el denominado coeficiente de temperatura, usado para controlar el equilibrio entre exploración (entropía alta) y explotación (retorno alto).

En SAC se utilizan las siguientes redes neuronales:

- **Red actor** $\pi(a|s;\theta^\pi)$: estima una política estocástica mediante una distribución Gaussiana. A la salida de la red se obtienen, para cada acción, los parámetros (media y desviación estándar) de dicha distribución.
- **Red de valor objetivo** $V(s|\theta^V)$: en SAC se utilizan dos redes de valor independiente, una *target* y otra *online*.
- **Redes crítico** $Q_1(s,a|\theta^{Q_1})$ y $Q_2(s,a|\theta^{Q_2})$: SAC emplea dos redes Q para minimizar el *offset* o *bias* positivo en las estimaciones de valor; esta estrategia se toma del algoritmo de RL *Double Q-learning*) [25]. Se toma el menor de ambos valores en cada iteración cuando se actualiza la red de valor objetivo. Para la pérdida de las propias redes crítico se toma la media de ambas estimaciones.

La entropía se calcula directamente a partir de la política Gaussiana. A menudo, el parámetro α se ajusta automáticamente durante el entrenamiento siguiendo una pérdida definida sobre la entropía objetivo, lo que permite adaptarse dinámicamente a distintas fases del aprendizaje.

Una vez descritas las redes neuronales que conforman la parte del aprendizaje profundo en el algoritmo SAC y el concepto de entropía, se procede a mostrar la estructura de las diferentes redes en la figura 2.5. Se espera que de esta forma resulte claro el flujo entrada-salida de cada red, facilitando la compresión a la hora de describir la fase de aprendizaje.

En el algoritmo 5 se muestra directamente el pseudocódigo de SAC, proporcionando en el mismo los comentarios pertinentes para su comprensión. Esta implementación se basa en la descrita en el libro *Grokking Deep Reinforcement Learning* de Miguel Morales [16] pero incorporando la nomenclatura del artículo original [10].

Algorithm 5 Soft Actor-Critic (SAC)

Input: Buffer de experiencia D, tasa de aprendizaje α, factor de descuento γ, peso de entropía α
Inicializar redes Q: $Q_{\theta_1}(s,a)$, $Q_{\theta_2}(s,a)$
Inicializar red de valor: $V_\psi(s)$ y su red objetivo: $V_{\bar{\psi}}(s)$ con $\bar{\psi} \leftarrow \psi$
Inicializar política estocástica $\pi_\phi(a|s)$
for cada episodio **do**
    Inicializar estado s_0
    for cada paso en el episodio **do**
        Seleccionar acción $a_t \sim \pi_\phi(a_t|s_t)$ y ejecutarla en el entorno
        Observar recompensa r_t y siguiente estado s_{t+1}
        Almacenar transición (s_t, a_t, r_t, s_{t+1}) en D
        Muestrear minibatch de N transiciones (s_i, a_i, r_i, s_{i+1}) de D
        for all transición del minibatch **do**
            Muestrear $a_i' \sim \pi_\phi(s_i)$
            Estimar el valor objetivo:

$$y_i = \min_{j=1,2} Q_{\theta_j}(s_i, a_i') - \alpha \log \pi_\phi(a_i'|s_i)$$

            Actualizar la red de valor minimizando:

$$L_V(\psi) = \frac{1}{N} \sum_i \left(V_\psi(s_i) - y_i\right)^2$$

            Estimar objetivo para críticos:

$$y_i^Q = r_i + \gamma V_{\bar{\psi}}(s_{i+1})$$

            Actualizar críticos minimizando:

$$L_Q = \frac{1}{2N} \sum_i \left(Q_{\theta_1}(s_i, a_i) - y_i^Q\right)^2 + \frac{1}{2N} \sum_i \left(Q_{\theta_2}(s_i, a_i) - y_i^Q\right)^2$$

            Actualizar actor maximizando:

$$J_\pi = \frac{1}{N} \sum_i [y_i]$$

        end for
        Actualizar red de valor objetivo:

$$\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$$

    end for
end for

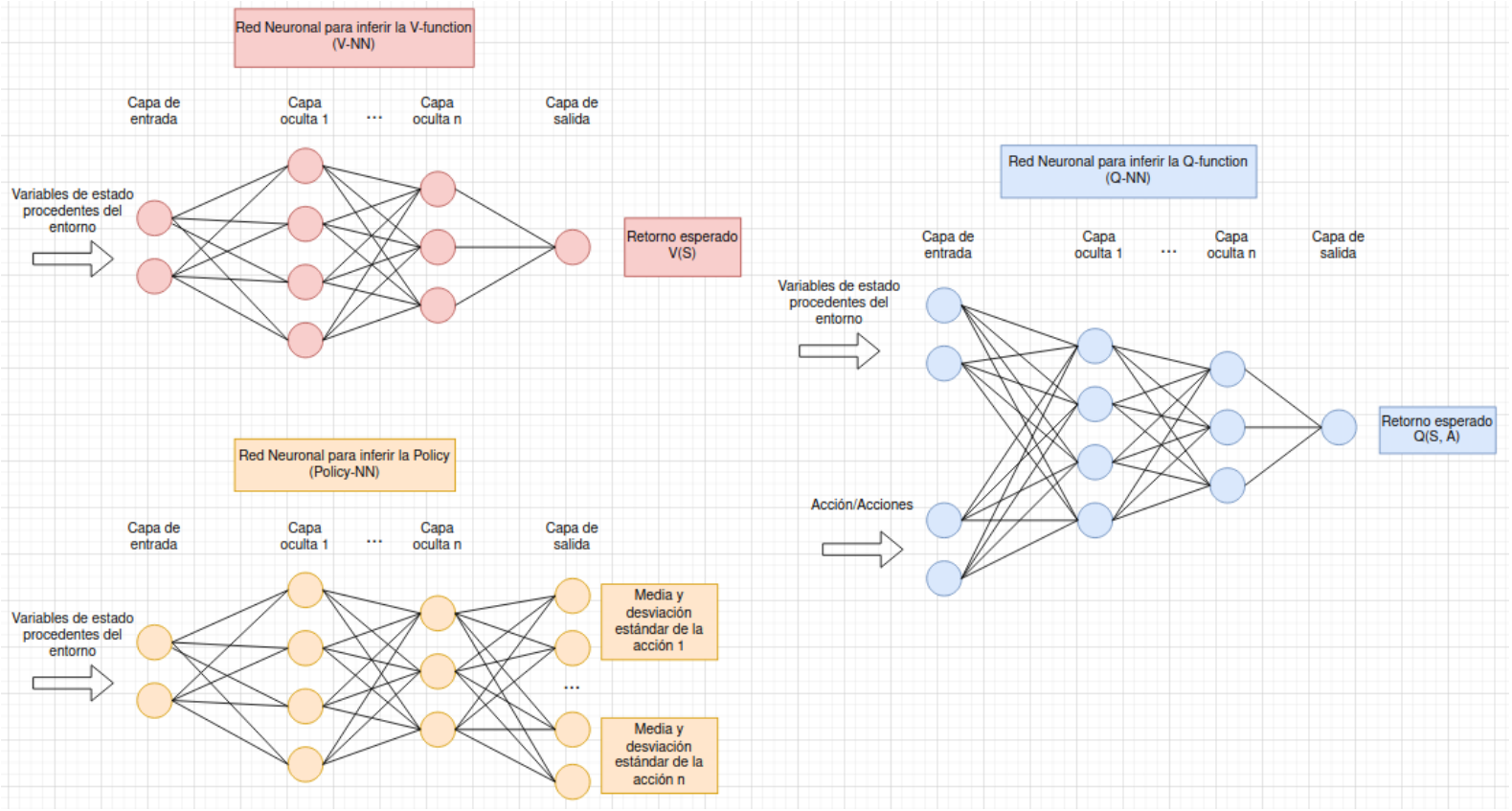


Figura 2.5: Redes neuronales que conforman la parte del aprendizaje profundo en el algoritmo SAC (*Policy-NN, V-NN y Q-NN*).

Capítulo 3

Desarrollo del entorno de simulación físicamente realista

Contenido

En este capítulo se abordan los conceptos relacionados con la implementación del entorno de simulación. Por ello, se discuten aspectos referentes a las librerías empleadas y se da una visión global del sistema, acompañada por diagramas donde se detallan los métodos e interfaces más relevantes.

3.1. Concepción del sistema

La motivación principal de este trabajo es el desarrollo de un *framework* de simulación en el que realizar experimentos de interacción humano-robot que involucren un robot manipulador y un brazo humano. Concretamente, el manipulador empleado es el Franka Emika Panda [8], el cual consta de siete grados de

libertad, un alcance de 855 mm, sensores de par en sus articulaciones y una capacidad de carga de hasta 3 kg, entre otras características (ver figura 3.1).

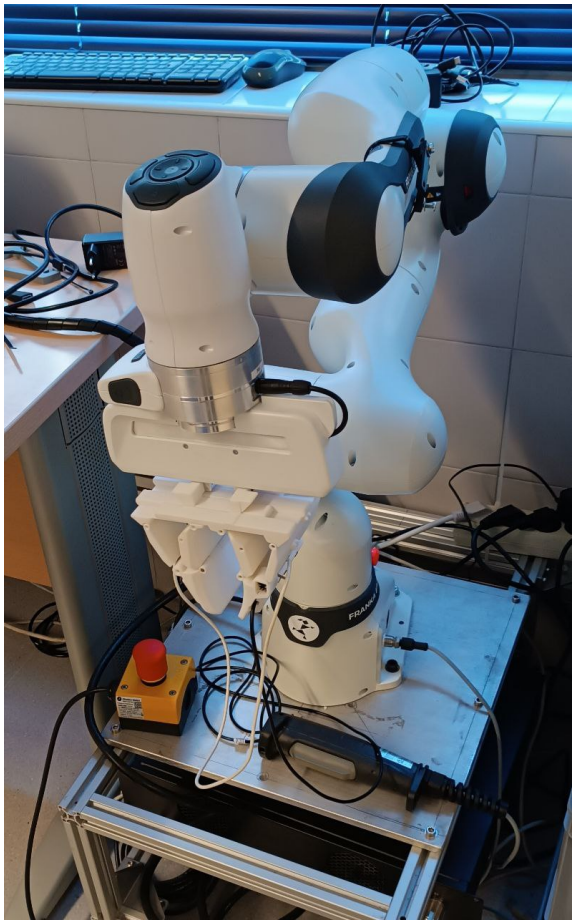


Figura 3.1: Fotografía del robot manipulado Franka Emika Panda ubicado en el laboratorio 2.005 de la Escuela de Ingenierías Industriales de la Universidad de Málaga.

3.2. Librerías

En esta sección se presentan las principales librerías que han permitido la construcción del entorno de simulación y el entrenamiento del agente en tareas de interacción humano-robot mediante aprendizaje por refuerzo.

Previamente a su integración, se ha llevado a cabo una revisión de las principales librerías y herramientas actualmente disponibles en el ámbito del aprendizaje por refuerzo aplicado a robótica. Las seleccionadas en este trabajo han sido

elegidas por varios motivos: su carácter de código abierto, lo que facilita la personalización e integración en arquitecturas específicas; su amplio uso dentro de la comunidad científica, lo que aporta confianza sobre su eficacia y validez en contextos experimentales complejos; y la disponibilidad de documentación y soporte activo por parte de los desarrolladores. Estos factores han permitido construir un entorno robusto, modular y reproducible.

3.2.1. dm_robotics_panda

`dm_robotics_panda` [4] es una librería desarrollada para facilitar la simulación y el control del brazo robótico Franka Emika Panda. Esta herramienta proporciona una interfaz que integra el entorno de simulación de *MuJoCo* [3] con el mencionado manipulador, permitiendo realizar experimentos de control y aprendizaje por refuerzo tanto reales como simulados.

MuJoCo es un motor de simulación física de propósito general diseñado para facilitar la investigación y el desarrollo en robótica, biomecánica o aprendizaje automático, entre otras áreas. Para ello permite el modelado de entornos y sistema mediante archivos en formato XML.

La librería `dm_robotics_panda` presenta una estructura modular que brinda una amplia flexibilidad en el desarrollo, permitiendo integrar entornos y sistemas propios descritos en archivos XML. Además, incluye ejemplos que conforman un sólido punto de partida para desarrollos propios, como es el caso de este trabajo.

Se destaca principalmente la clase `agent`, y su método `step`, el cual se itera cada 0.1 s. En este método se reciben las observaciones que el agente (el robot manipulador) ha recopilado tras interactuar con el entorno, y se retorna la acción a ejecutar por el mismo siendo las posibles actuaciones velocidades cartesianas del efector final, velocidades de rotación en las diferentes articulaciones o una interacción háptica entre un robot Panda físico y el simulado.

3.2.2. myo_sim

`myo_sim` [2] es una librería diseñada para proporcionar una plataforma de investigación centrada en el control motor humano, a través de modelos biomecánicos fisiológicamente realistas del codo, la muñeca, la mano y las piernas. Estos modelos, dotados de capacidades de interacción física, permiten simular tareas complejas y precisas en entornos reales. Todos estos modelos se encuentran descritos en archivos XML interpretables por el motor de simulación *MuJoCo*.

De todos los modelos proporcionados por la librería `myo_sim` se hará uso del

módulo `MyoArm`, que permite simular el comportamiento de un brazo humano con un total de 27 grados de libertad y 63 músculos. `myo_sim` proporciona un entorno, en el que se puede cargar el brazo humano simulado (se utilizará el término `myoarm` para hacer referencia al brazo). Un entorno no es más que un archivo XML que integra texturas para modelar un ambiente de consulta clínica junto con el propio `myoarm`, el cual puede verse en la figura 3.2. Este archivo se empleó como punto de partida, y sobre él se realizaron las modificaciones pertinentes para recrear un entorno con las características espaciales similares a las que se encuentran en el laboratorio 2.005 de la Escuela de Ingenierías Industriales de la Universidad de Málaga. En la figura 3.3 se puede ver en la fila superior unas fotos del laboratorio real con el robot manipulador Franka Emika Panda, y, bajo estas, unas capturas del entorno de simulación.

En las pruebas reales la persona se encuentra sentada en la silla y el robot manipulador toma su brazo a la altura de la muñeca para realizar la tarea de rehabilitación pertinente. Puesto que `myo_sim` no proporciona un modelo de humano sentado; se ha decidido en su lugar alzar la base sobre la que se coloca el manipulador para que quede a una altura relativa similar a la que se encuentra el real respecto a una persona sentada.

De la librería `myosim`, concretamente de su módulo `myoarm`, se han tomado los archivos XML interpretables por el motor de simulación *MuJoCo* para así construir un entorno con las características espaciales similares a las que se encuentran en el laboratorio real. Con este brazo humano interactuará el manipulador Franka Emika Panda. Este último es incorporado a la simulación a través de la librería `dm_robotics_panda`.

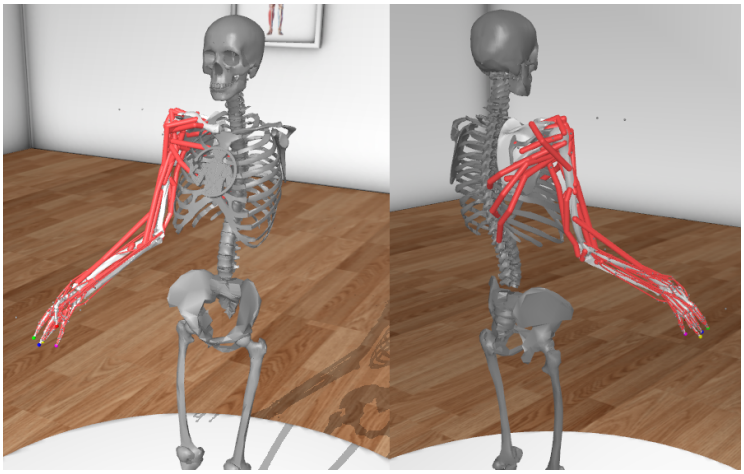


Figura 3.2: Ilustraciones del modelo `myoarm` de la librería `myo_sim`.

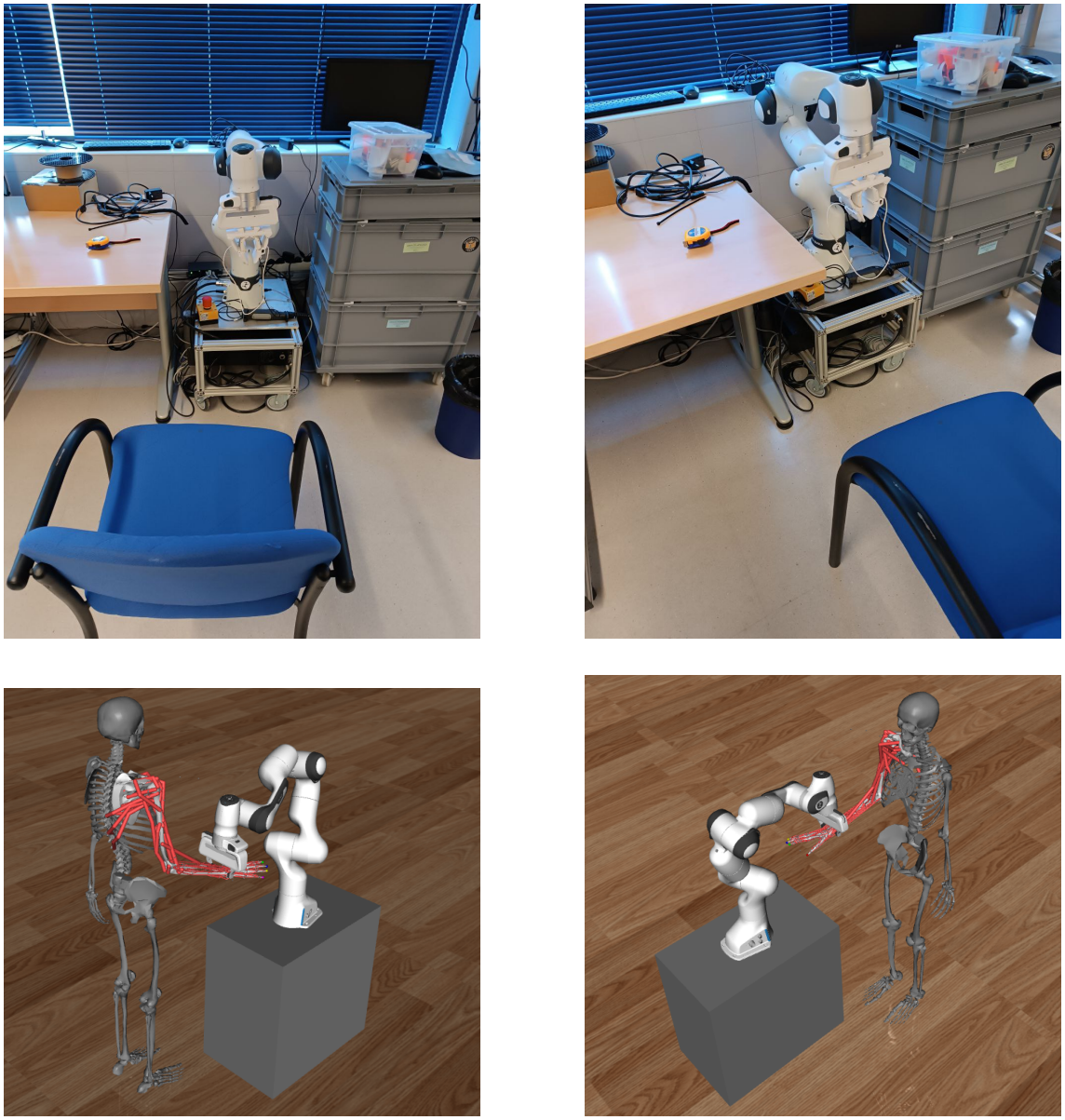


Figura 3.3: Ilustraciones del entorno real en laboratorio y su recreación en simulación mediante la integración de las librerías `myo_sim` y `dm_robotics_panda`.

3.2.3. Stable Baselines 3

`Stable Baselines 3` (SB3) [18] es una implementación en PyTorch [17] de algoritmos de aprendizaje por refuerzo. Proporciona una interfaz sencilla y consistente para entrenar, evaluar y desplegar agentes de RL en diversos entornos

de simulación. SB3 incluye implementaciones de siete algoritmos de RL y DRL:

1. **Advantage Actor Critic (A2C)** [14].
2. **Deep Deterministic Policy Gradient (DDPG)** [13]: Ha sido comentado en el capítulo anterior.
3. **Deep Q Network (DQN)** [15]: Ha sido comentado en el capítulo anterior.
4. **Heterogeneous Relational Deep Reinforcement Learning (HeR-DRL)** [31].
5. **Proximal Policy Optimization (PPO)** [20].
6. **Soft Actor-Critic (SAC)** [10]: Ha sido comentado en el capítulo anterior.
7. **Twin Delayed Deep Deterministic Policy Gradient (TD3)** [9].

Esta recopilación de algoritmos es ampliamente utilizada en la investigación y aplicación de RL. La librería está diseñada para ser modular y extensible, facilitando su integración con diferentes entornos y simuladores.

De la librería `Stable Baselines 3` se ha tomado la implementación del algoritmo de DRL SAC [21]. Además del mencionado algoritmo, se debe destacar que SB3 proporciona una clase virtual de la que heredar para integrar el proceso de aprendizaje por refuerzo en cualquier simulador de una forma intuitiva. Esta clase es `gym.Env`; dentro de ella se definen el espacio de acciones y las observaciones. Su método principal es `step`; en este se reciben las acciones que el agente deberá realizar en el paso actual. Dentro de este método se deberán conformar las observaciones y recompensa.

`Stable Baselines 3` proporciona una cómoda abstracción de conceptos como el *replay buffer* o el método de optimización de las redes neuronales que conforman el algoritmo. Pese a que la configuración de hiperparámetros proporcionada por defecto pueda resultar efectiva [21], se recomienda ajustarla en caso de ser necesario, como se hizo en este trabajo.

3.2.4. rl_spin_decoupler

La librería `rl_spin_decoupler` [24] está diseñada para desacoplar la ejecución de un algoritmo de aprendizaje por refuerzo, como los proporcionados por `Stable Baselines 3`, de la ejecución de un agente físico o simulado que requiere operar en su propio ciclo de control o `spin loop`.

Si bien esto generalmente no es necesario, pudiéndose implementar en la misma clase del entorno todo lo necesario tanto para RL como para el agente, hay marcos de aplicación, como este, en que eso no es posible. En este caso particular, la simulación del agente debe ejecutarse de forma independiente, operando en su propio ciclo de control (`spin loop`) con una frecuencia constante y requisitos temporales estrictos que no pueden ser garantizados si se comparte el mismo hilo o proceso con la lógica del aprendizaje.

La arquitectura de esta librería se basa en un modelo cliente-servidor que utiliza *sockets* para la comunicación entre procesos. Un proceso ejecuta el algoritmo de RL y utiliza la clase `RLSide` del módulo, mientras que otro proceso ejecuta el agente (la simulación del entorno con el brazo robótico junto al humano) y emplea la clase `AgentSide`. Esta separación permite un control fino de los tiempos de ejecución de acciones por parte del agente.

Los métodos principales de la clase `AgentSide` son:

- `readWhatToDo()`: recibe las indicaciones que dictan al agente si debe recibir la acción, resetear el entorno y mandar una observación o finalizar su ejecución.
- `stepSendObs()`: para mandar las observaciones recopiladas por el agente al proceso que implementa el RL.
- `resetSendObs()`: para mandar las observaciones obtenidas tras el reinicio del entorno al proceso que implementa el RL.

Los métodos principales de la clase `AgentSide` son:

- `stepSendActGetObs()`: para mandar las acciones a ejecutar al agente y recibir las observaciones recopiladas por el mismo.
- `resetGetObs()`: para recibir las observaciones obtenidas tras el reinicio del entorno por parte del agente.

La librería `rl_spin_decoupler` está diseñada para su sencilla integración en las implementaciones de los algoritmos de aprendizaje por refuerzo de SB3.

3.3. Estructura del sistema

Se han comentado anteriormente las librerías que forman parte del *framework* de simulación para la realización de experimentos de interacción humano-robot

que se ha desarrollado en este trabajo. Si bien resulta complejo describir estas librerías sin anticipar conceptos referentes a la estructura del sistema, en esta sección se va a detallar la misma en profundidad. En primer lugar, en la figura 3.4 se presenta un diagrama general.

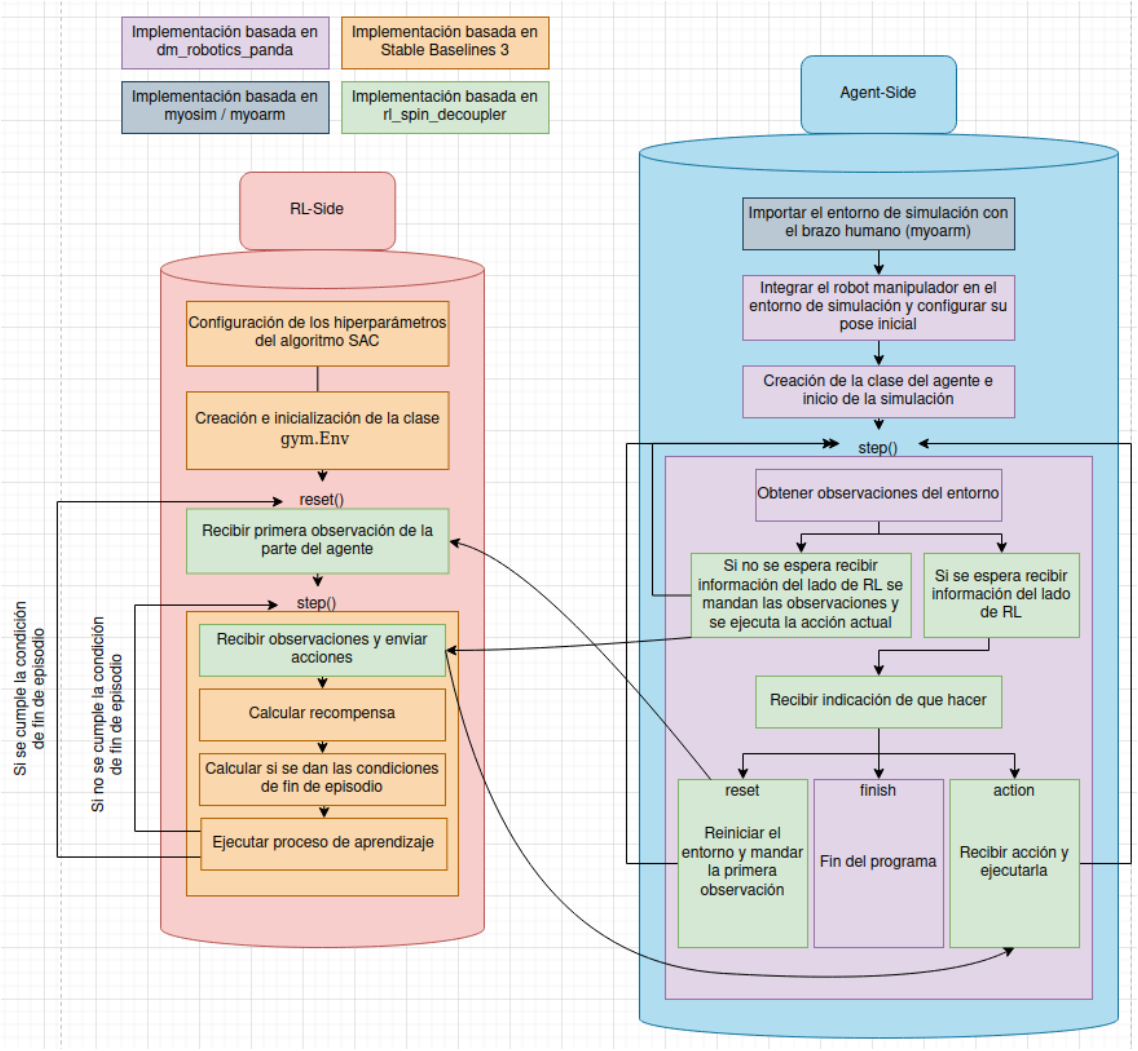


Figura 3.4: Estructura del sistema, detallando las etapas esenciales de cada lado (`AgentSide` y `RLSide`) y su comunicación.

Empleando como apoyo la figura 3.4, procedemos a comentar la implementación.

El primer paso en el lado del aprendizaje (`RLSide`) es configurar el algoritmo de RL a emplear, en este caso SAC. Es recomendable ajustar los hiperparáme-

tros; para ello será necesario basarse en la guía proporcionada por la documentación de SB3 [21], en el conocimiento que se tenga del algoritmo (por ello fue descrito en el capítulo anterior) y la propia experimentación. Tras esto se hereda de la clase `gym.Env` para desarrollar una propia denominada `CustomEnv` con los mismos métodos que la clase padre. Estos son (i) `reset()`, en el que se recibe la observación procedente del reinicio de la simulación en la parte del agente, y (ii) `step()`, que comienza recibiendo las observaciones por parte del agente y enviándole las acciones a ejecutar en el paso actual. Con las observaciones se calcula la recompensa del *step* y además se comprueba si se dan las condiciones de fin de episodio. Estas condiciones son superar el límite de iteraciones máximas que se contemplan en un episodio o haber recibido unas observaciones que demuestran un comportamiento peligroso por parte del agente. Bajo ambas condiciones se envía una señal de reinicio al lado del agente; en caso contrario, se itera otro paso o *step*. Es importante comentar que en el constructor de la clase se deben definir las dimensiones y límites del espacio de acciones y del espacio de estados.

En el lado del agente (`AgentSide`) se comienza cargando el entorno de simulación que incluye el brazo humano empleando los ficheros XML creados a partir de la librería `myosim`. Tras esto, se incorpora a la simulación el robot manipulador Franka Emika Panda y se configura su pose inicial empleando la librería `dm_robotics_panda`. Se inicia la simulación, y en cada iteración del método `step` se reciben las observaciones fruto de la interacción (mediante las acciones) del agente con el entorno. Tras esto se emplean las funciones o métodos descritas de la librería `rl_spin_decoupler`. Al final de cada iteración del método `step` (excepto las interrumpidas por el reinicio de la simulación) se devuelve la acción a ejecutar por el agente. Como se ha comentado, esta acción es configurable, pudiendo tratarse de velocidades cartesianas del efector final, velocidades de rotación en las diferentes articulaciones o una interacción háptica entre un robot Panda físico y el simulado.

3.4. Diseño modular y flexible del sistema

En la sección anterior se ha detallado la estructura del sistema, además de sus métodos e interfaces principales. Hasta el momento no se ha descrito la tarea a desarrollar en ningún momento, lo cual no es casualidad. Se pretende que se perciba la versatilidad del sistema, el cual permite la experimentación de cualquier tarea que involucre el control del manipulador durante su interacción con un brazo humano y dentro del contexto del aprendizaje por refuerzo. Por ello no se ha querido ligar la presentación del sistema a una tarea concreta.

En caso de querer realizar una implementación propia, bastaría con replicar el sistema siguiendo la guía que se detalla en el anexo A. Tras esto, se podría acceder al código fuente que desarrolla la parte del `RLSide` con unos sencillos cambios

- Cambiar el algoritmo SAC por cualquiera de los que incorpora SB3 o bien mantenerlo y modificar sus hiperparámetros, si el usuario lo considera oportuno, para adaptarlo a su tarea.
- Modificar en el constructor de la clase `CustomEnv` la definición de los espacios de acciones y estados.
- Modificar el método de cálculo de recompensa (`calculate_reward`) implementando en el mismo la función de recompensa pertinente basada en las observaciones establecidas.
- Modificar el método de comprobación de fin de episodio (`end_episode`) en base a las necesidades del marco de aplicación.

En la parte del agente (`AgentSide`) se deberían implementar los siguientes cambios:

- Configurar la acción a ejecutar por el agente, pudiendo tratarse de velocidades cartesianas del efector final, velocidades de rotación en las diferentes articulaciones o una interacción háptica entre un robot Panda físico y el simulado.
- Definir las observaciones que se recabarán para enviar al `RLSide` mediante la modificación del método `format_obs`.

Con estas pequeñas modificaciones se podría comenzar a entrenar un agente para el control del manipulador Franka Emika Panda en su interacción con un brazo humano. Esto conforma un *framework* generalista dentro del ámbito de la interacción humano-robot (Human Robot Interaction HRI) que permite recabar experiencias realistas en simulación. Esta contribución se alinea con los desafíos clave identificados para la aplicación del aprendizaje por refuerzo en el ámbito del HRI, tal como se expone en el artículo [23].

Capítulo 4

Prueba de uso del *framework*

Contenido

4.1. Definición formal de la tarea

En capítulos anteriores se ha realizado una revisión del campo del aprendizaje por refuerzo profundo, describiendo tanto los fundamentos teóricos como algunos de los algoritmos más relevantes en la historia de la materia.

Tras esto se ha descrito el entorno de simulación, abordando desde su caracterización cualitativa hasta los aspectos técnicos relacionados con su implementación. Con estos elementos introducidos, se dispone del marco conceptual y técnico necesario para el desarrollo de un agente autónomo, capaz de planificar trayectorias orientadas a tareas de rehabilitación, mediante la interacción física entre un brazo humano y un manipulador robótico.

El hito que se trata de alcanzar en esta etapa es la generación de un agente capaz de seguir una trayectoria arbitraria desplazando conjuntamente un brazo humano, evitando alcanzar estados que puedan provocar al usuario sensaciones de dolor o resistencia mecánica percibida como forcejeo. Para ello, se deben definir unas métricas de calidad que posteriormente se traducirán en una función de recompensa en el momento del entrenamiento.

4.2. Estudio de los umbrales de fuerza y par

Como se ha mencionado previamente, el agente desarrollado debe ser capaz de seguir trayectorias de referencia, desplazando de forma conjunta un brazo humano y evitando alcanzar configuraciones espaciales que generen en el usuario sensaciones de dolor o incomodidad, comúnmente denominadas *ban points* [26]. Las métricas que se emplearán para determinar si el usuario ha alcanzado un *ban point* son las fuerzas y pares medidos en la muñeca del manipulador robótico simulado.

Con el objetivo de determinar los umbrales de dichas magnitudes físicas que pueden inducir una percepción de dolor o resistencia en el usuario, se ha llevado a cabo una serie de experimentos controlados. En estos ensayos se ha comandado al manipulador una velocidad cartesiana constante, registrándose los valores de fuerza y par en cada uno de los ejes del espacio cuando, a través de observación visual, se ha evidenciado que el movimiento inducido podría producir malestar físico en el brazo humano acoplado.

Los experimentos realizados consistieron en la inicialización del entorno de simulación y la aplicación de una velocidad cartesiana nula en todos los ejes espaciales, exceptuando un eje en el que se estableció una velocidad lineal no nula. De este modo, se evaluaron los límites de fuerza y par generados en la muñeca del manipulador robótico para cada uno de los ejes de traslación de forma individual. Una vez finalizado el análisis de los efectos inducidos por velocidades lineales, se estudió el impacto de imponer velocidades angulares sobre el efector final, aplicando de forma aislada rotaciones tipo *roll*, *pitch* y *yaw*. En la figura 4.2 se pueden ver ilustradas algunas de las pruebas.

A partir de los datos obtenidos en estas pruebas, se determinó que la percepción de dolor por parte del usuario se produce cuando se exceden los 25 N en magnitud de fuerza o los 55 Nm en magnitud de par medido en la muñeca del manipulador, en cualquiera de los ejes espaciales. En consecuencia, toda observación que supere dichos umbrales será interpretada como indicativa de la inducción de dolor al usuario por parte del agente.

Asimismo, se establecieron umbrales intermedios para representar niveles de molestia. En concreto, se definió un umbral de 15 N para las fuerzas y de 35 Nm para los pares, por encima de los cuales se considerará que el agente está generando incomodidad, aunque sin alcanzar el nivel de dolor.

Cabe destacar que la percepción de dolor e incomodidad es altamente subjetiva y varía considerablemente entre individuos. Por este motivo, aunque los umbrales definidos en este estudio ofrecen una referencia inicial útil, sería conveniente realizar estudios más amplios y personalizados para caracterizar adecuadamente la respuesta de diferentes usuarios ante fuerzas y pares aplicados. No obstante, debido a las limitaciones del presente trabajo, no ha sido posible llevar a cabo un estudio con usuarios reales, por lo que los umbrales aquí definidos se utilizan como una primera aproximación para el entrenamiento del agente.

En la figura 4.2 se presentan seis configuraciones espaciales representativas que formaron parte del ensayo experimental. Se recomienda prestar atención a la figura 4.1 para visualizar el sistema de referencia *world* sobre el que se comandan las velocidades del efector final del manipulador.

En la parte superior de las diferentes capturas del simulador tomadas durante los experimentos (ver figura 4.2) se pueden apreciar tres gráficas. De izquierda a derecha, describen la magnitud a medir sobre cada eje, fuerzas o pares, seguido de la acción comandada, en este caso la velocidad y finalmente la recompensa, que se ha fijado a cero, ya que no se está realizando aprendizaje. Destacar que las fuerzas y pares se miden sobre la muñeca del manipulador, no sobre el sistema de referencia *world* mostrado en la figura 4.1.

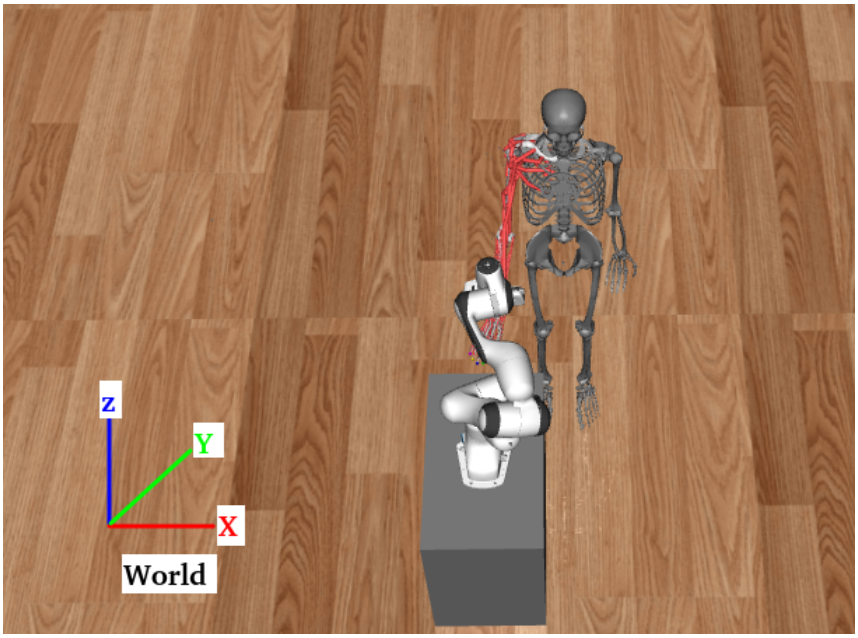
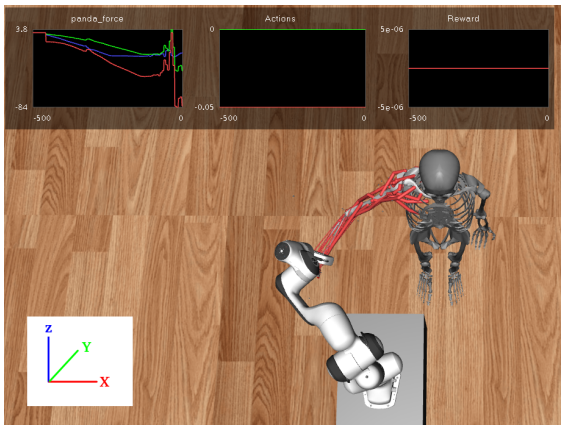
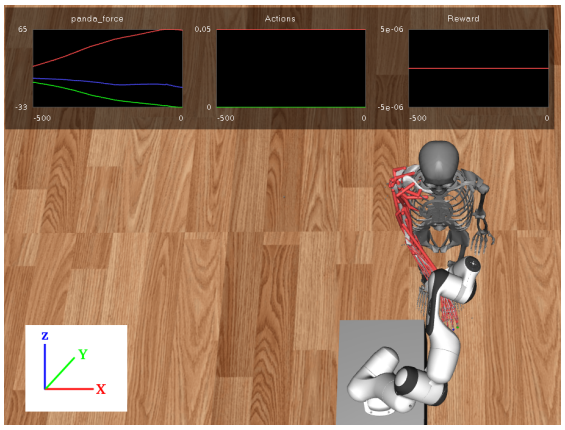


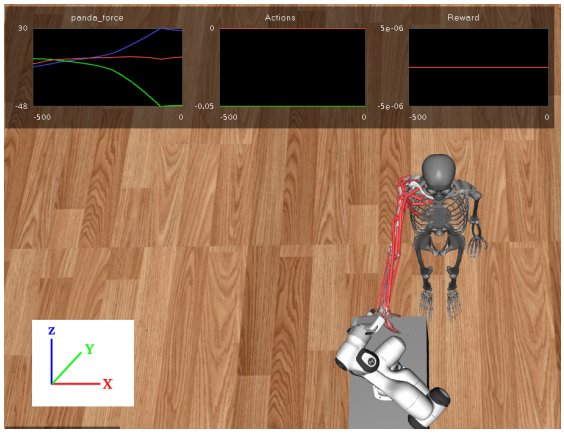
Figura 4.1: Ilustración del sistema de referencia *world* sobre el que se comandan las velocidades del efector final del manipulador en el simulador.



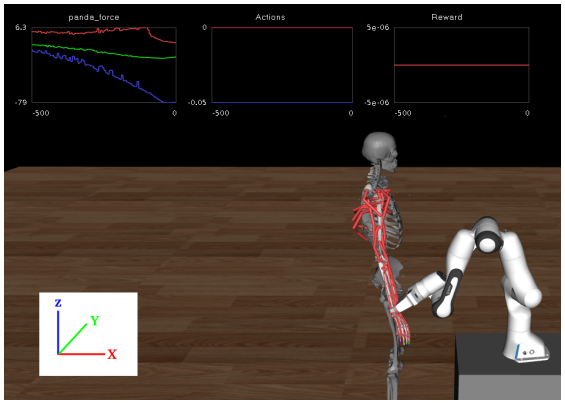
(a) Estudio de los umbrales de fuerza comandando una velocidad lineal negativa sobre el eje espacial X.



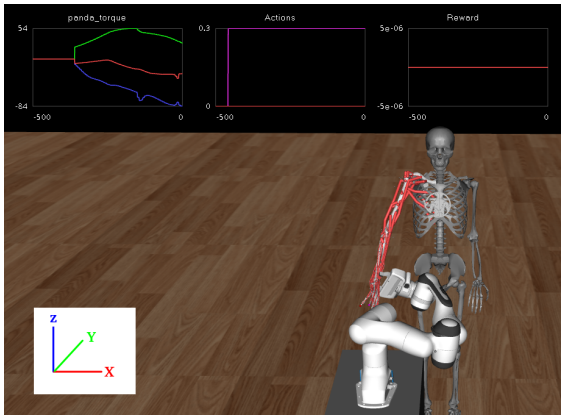
(b) Estudio de los umbrales de fuerza comandando una velocidad lineal positiva sobre el eje espacial X.



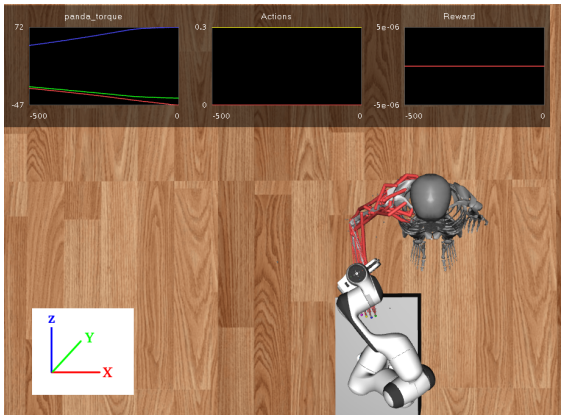
(c) Estudio de los umbrales de fuerza comandando una velocidad lineal negativa sobre el eje espacial Y.



(d) Estudio de los umbrales de fuerza comandando una velocidad lineal negativa sobre el eje espacial Z.



(e) Estudio de los umbrales de par comandando una velocidad angular de tipo *pitch*.



(f) Estudio de los umbrales de par comandando una velocidad angular de tipo *yaw*.

Figura 4.2: Configuraciones espaciales representativas utilizadas en el experimento.

4.3. Estudio de trayectorias completas

Una vez realizado el estudio de fuerzas y pares, se ha implementado una máquina de estados sobre el método `step` del agente. Esta se encarga de conmutar la velocidad cartesiana comandada sobre el efector final en función del tiempo. Con este sencillo comportamiento se consigue que el manipulador siga una trayectoria, aunque de momento, sin atender a ninguna observación proveniente del entorno. Se realizan dos nuevos experimentos, uno siguiendo una trayecto-

ria cuadrada a una velocidad de 0.05 m/s y otro describiendo una trayectoria triangular.

4.3.1. Trayectoria cuadrada

En la figura 4.3 se aprecia la comparativa entre la trayectoria ideal y la que ha realizado el manipulador desplazando conjuntamente un brazo humano. La discrepancia entre ambas se debe a que el brazo humano induce una resistencia al movimiento que desvía de la trayectoria ideal al manipulador. Esto se aprecia claramente en la figura 4.6, donde se comparan las velocidades lineales medidas en el efector final y las programadas. A pesar de que es imposible describir un pulso ideal como el que se asigna, es importante destacar que en los ejes XY no se puede alcanzar la velocidad ideal debido a, como se ha mencionado, la resistencia al movimiento por parte del brazo humano.

Por otra parte, en las figuras 4.4 y 4.5 se pueden apreciar las fuerzas y pares medidos sobre la muñeca del manipulador. Los picos de estas magnitudes que se muestran en las gráficas concuerdan con los fijados en el estudio anterior, entrando dentro del rango de dolor, ya que el manipulador fuerza un movimiento que el brazo no puede seguir.

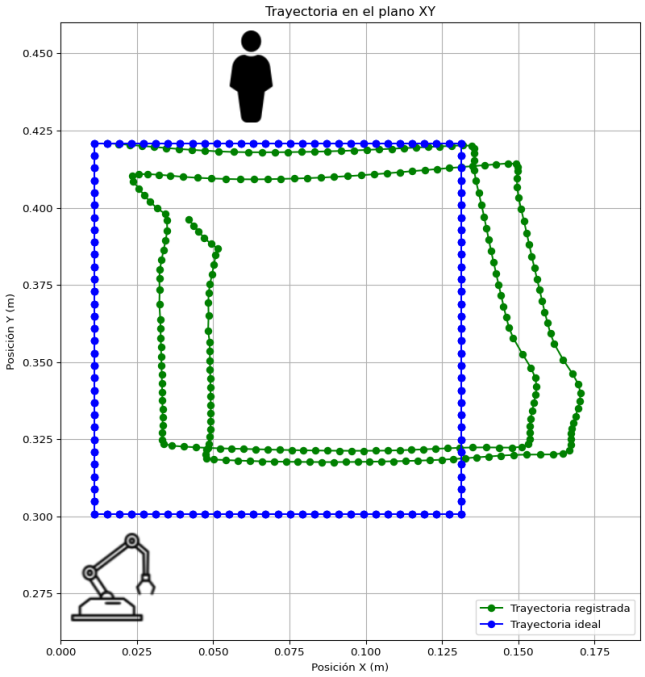


Figura 4.3: Comparación de la trayectoria descrita por el manipulador sobre el plano XY contra la ideal (dos vueltas).

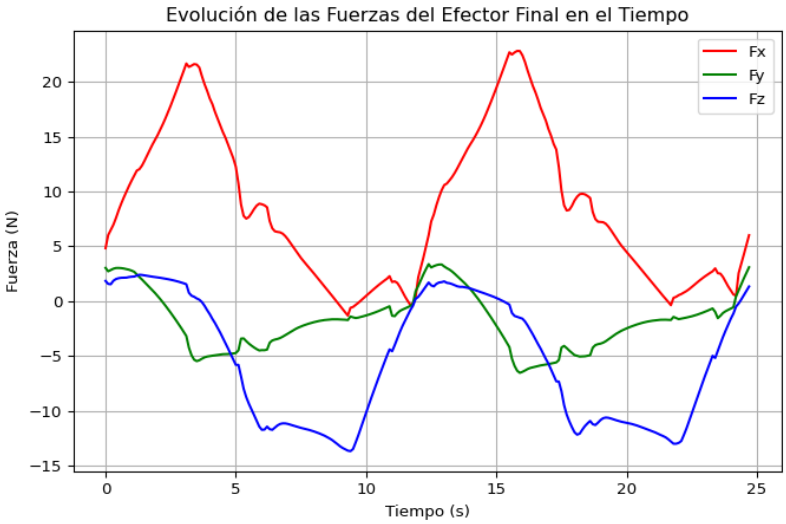


Figura 4.4: Fuerzas medidas en el efector final del manipulador durante la descripción de la trayectoria cuadrada (dos vueltas).

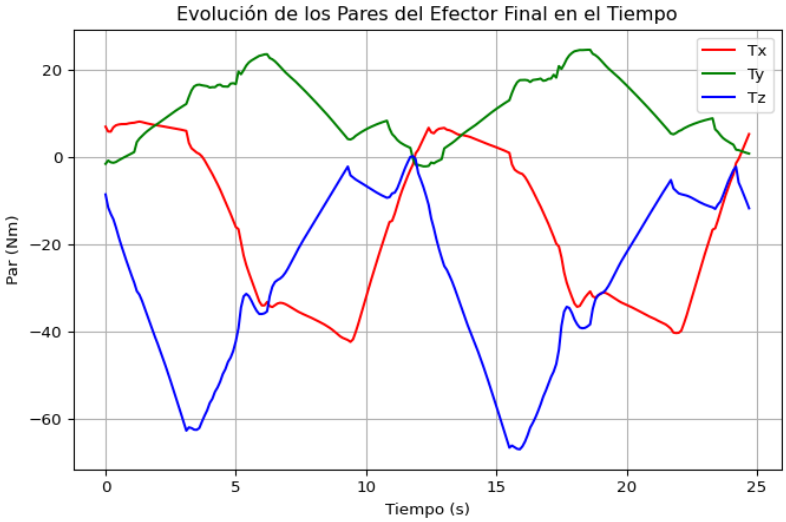


Figura 4.5: Pares medidos en el efector final del manipulador durante la descripción de la trayectoria cuadrada (dos vueltas).

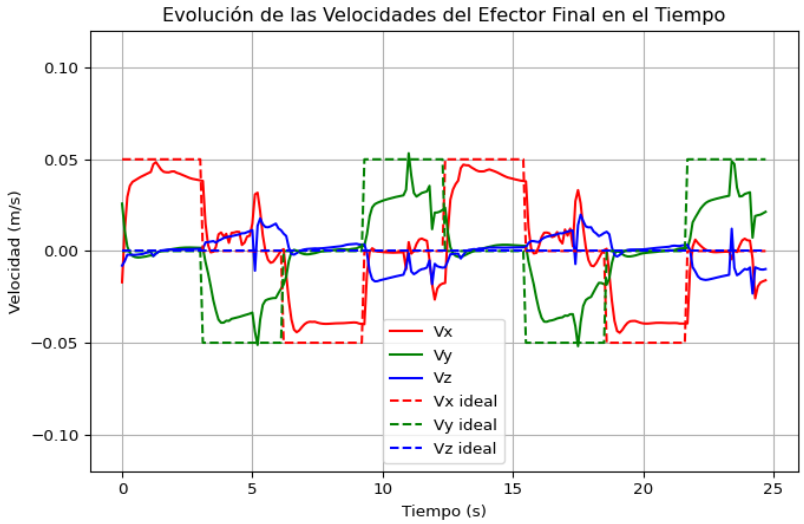


Figura 4.6: Comparación de las velocidades medidas en el efector final del manipulador y las ideales durante la descripción de la trayectoria cuadrada (dos vueltas).

4.3.2. Trayectoria triangular

De nuevo, en la figura 4.7 se muestra una comparativa de la trayectoria ideal frente a la ejecutada. Resulta esclarecedor percatarse de como en los instantes $T = 6s$ y $T = 15s$, la velocidad lineal medida en Y es notablemente inferior a la programada (ver figura 4.10), coincidiendo con un pico en la fuerza medida sobre el eje Z y otro en el par sobre el eje X de la muñeca del manipulador (ver figuras 4.8 y 4.9). Se prueba, por tanto, la hipótesis de que la discrepancia en las trayectorias se debe a la resistencia al movimiento por parte del brazo humano.

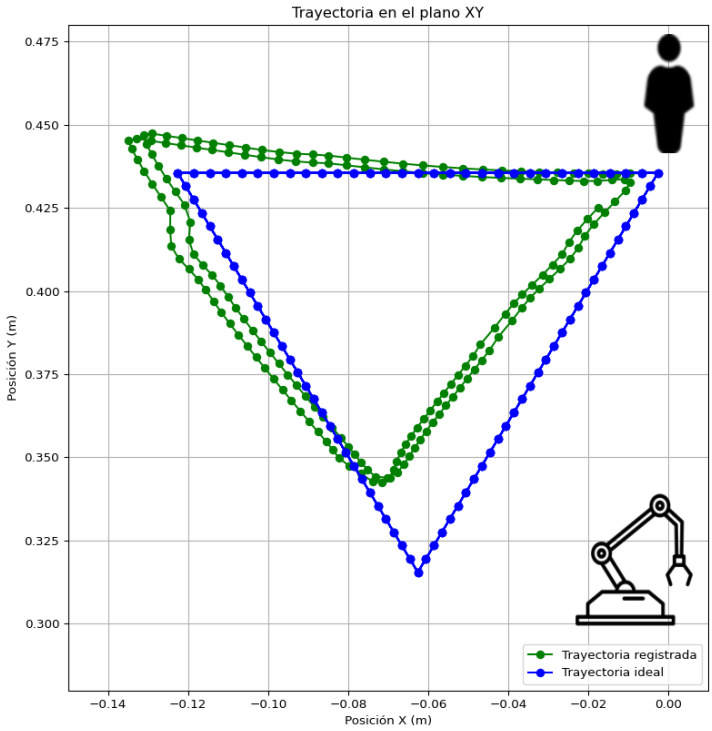


Figura 4.7: Comparación de la trayectoria descrita por el manipulador sobre el plano XY contra la ideal (dos vueltas).

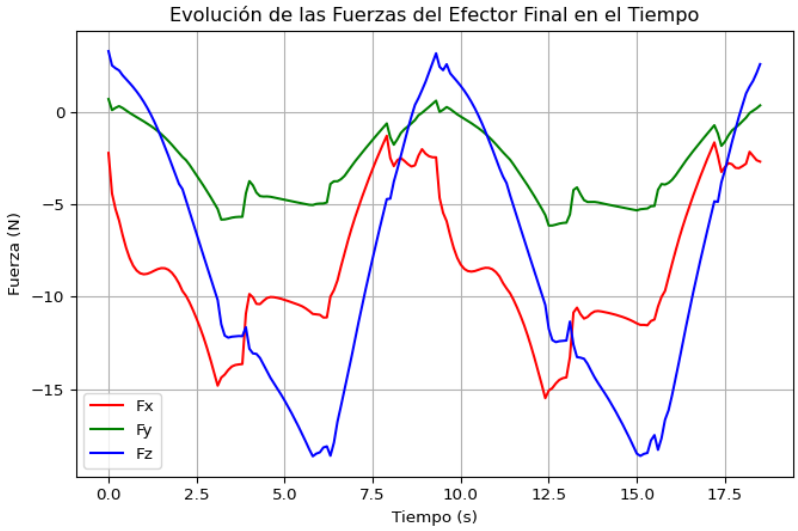


Figura 4.8: Fuerzas medidas en el efector final del manipulador durante la descripción de la trayectoria triangular (dos vueltas).

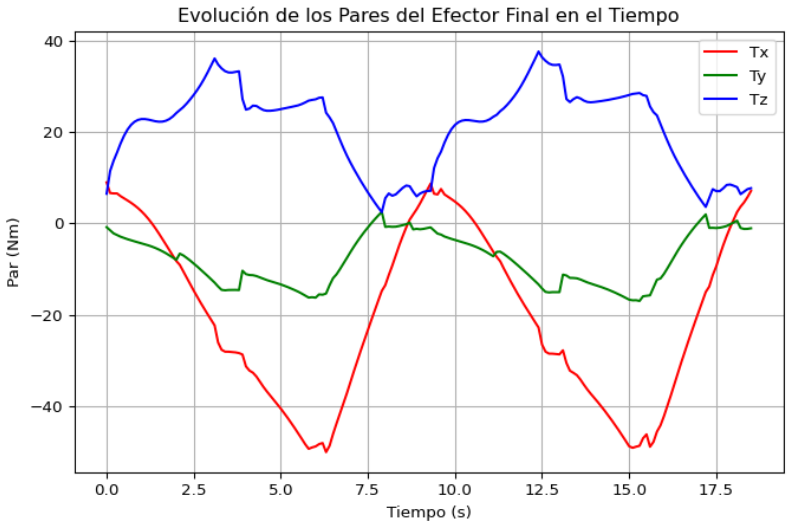


Figura 4.9: Pares medidos en el efector final del manipulador durante la descripción de la trayectoria triangular (dos vueltas).

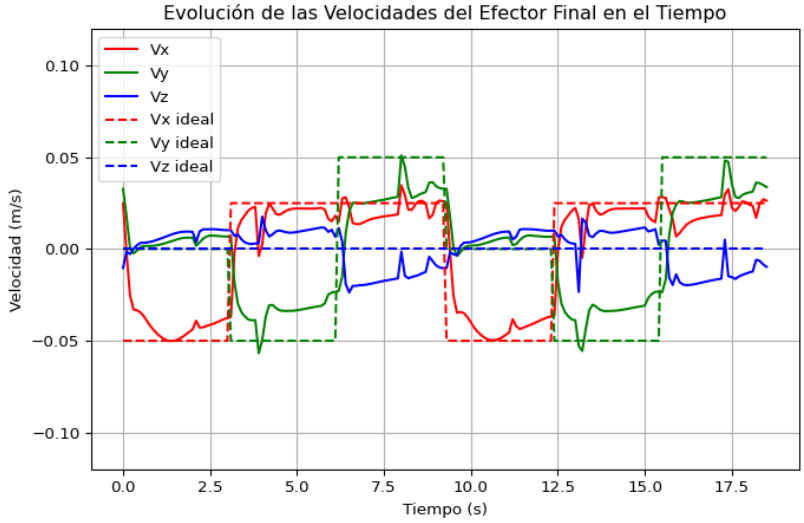


Figura 4.10: Comparación de las velocidades medidas en el efector final del manipulador y las ideales durante la descripción de la trayectoria triangular (dos vueltas).

4.4. Observaciones, acciones y función de recompensa

Definidos los límites operativos de seguridad a través del estudio de fuerzas y pares, y una vez validado el entorno mediante la ejecución de trayectorias predeterminadas, es posible abordar el diseño del agente desde una perspectiva de aprendizaje por refuerzo. Para ello, resulta imprescindible especificar tres elementos clave: el conjunto de observaciones que describen el estado del entorno, el espacio de acciones mediante el cual el agente puede interactuar con el entorno, y la función de recompensa, que actuará como guía para el aprendizaje de una política de control. En general esta política debe ser capaz de seguir trayectorias arbitrarias, desplazando conjuntamente un brazo humano, de forma efectiva y segura. En este caso, la acción que el agente puede realizar para interactuar con el entorno es la velocidad cartesiana de su efector final.

4.4.1. Espacio de observaciones

Como se comentó en el capítulo anterior, el agente se encarga de interactuar con el entorno y recibir observaciones. Concretamente, este proceso se lleva a cabo en el método `step` de la clase `agent`, procedentes de la librería `dm_robotics_panda`. A continuación, se detalla el conjunto de observaciones disponible, empleado para describir el estado del robot manipulador y su interacción con el entorno. Las descripciones de los diferentes parámetros se han obtenido de la documentación de `dm_robotics_panda` [6] y la inspección del código fuente [5].

- `panda_joint_pos`: Posición de cada una de las 7 articulaciones del manipulador.
- `panda_joint_vel`: Velocidad articular.
- `panda_joint_torques`: Par aplicado en cada articulación.
- `panda_tcp_pos`: Posición cartesiana del efector final (TCP) respecto de la base del robot.
- `panda_tcp_quat`: Orientación del TCP en formato cuaternión, respecto de la base del robot.
- `panda_tcp_rmat`: Matriz de rotación que describe la orientación del TCP respecto de la base del robot.

- `panda_tcp_pose`: Combinación de posición y orientación del TCP, es un vector que concatena las observaciones de `panda_tcp_pos` y `panda_tcp_quat`.
- `panda_tcp_vel_world`: Velocidad del TCP en el sistema de referencia global.
- `panda_tcp_vel_relative`: Velocidad del TCP en el sistema de referencia de la base del robot.
- `panda_tcp_pos_control`, `panda_tcp_quat_control`, `panda_tcp_rmat_control`, `panda_tcp_pose_control`, `panda_tcp_vel_control`: Variables que indican el objetivo de control actual en términos de posición, orientación y velocidad del TCP.
- `panda_force` y `panda_torque`: Fuerzas y pares medidos en la muñeca del manipulador.
- `panda_gripper_width` y `panda_gripper_state`: Estado del efector final en cuanto a la apertura que presenta (en metros) y un indicador del estado abierto/cerrado.
- `panda_twist_previous_action`: Acción previa ejecutada por el agente, expresada como un vector de velocidades lineales y angulares.
- `time`: Marca temporal del episodio.

En la práctica no se emplea un conjunto tan amplio de observaciones para el aprendizaje. Fruto de la experimentación se han ido añadiendo y eliminando parámetros observables por el agente hasta llegar a la selección final, la cual se detalla a continuación:

- `F_wristX`, `F_wristY`, `F_wristZ`: Fuerza medida en la muñeca del manipulador en los ejes X, Y y Z respectivamente.
- `T_wristX`, `T_wristY`, `T_wristZ`: Par (torque) medido en la muñeca del manipulador en los ejes X, Y y Z respectivamente.
- `Vx`, `Vy`, `Vz`: Velocidad cartesiana del efector final en los ejes X, Y y Z respectivamente (en metros por segundo).
- `roll`, `pitch`, `yaw`: Velocidades angulares del efector final alrededor de los ejes X, Y y Z (en radianes por segundo).

- `X_dif`, `Y_dif`, `Z_dif`: Diferencia entre la posición actual del efector final y la posición ideal de la trayectoria en los ejes X, Y y Z.
- `euclidean_dist`: Distancia euclídea entre la posición actual del efector final y la posición ideal de la trayectoria.
- `X_follow_vector`, `Y_follow_vector`, `Z_follow_vector`: Vector que indica la dirección de la trayectoria ideal en el paso actual en los ejes X, Y y Z. Este vector se obtiene de la diferencia entre el punto de la trayectoria a alcanzar y el actual.

Destacar, que todas las observaciones se han normalizado en el rango $[-1, 1]$ para mejorar la convergencia durante el aprendizaje.

Finalmente, comentar que la formación de estas observaciones a emplear durante el aprendizaje es lo que se ha codificado en el método `format_obs` de la clase `agent`, que se encarga de la interacción con el entorno (`AgentSide`).

4.4.2. Función de recompensa

La recompensa se definió en el segundo capítulo como un escalar que indica la bondad de la transición a un estado s' desde un estado s al ejercer la acción a. Es decir, codifica el objetivo que debe alcanzar el agente. La función de recompensa es aquella que, dados unos parámetros de entrada, proporciona a la salida el mencionado escalar que mide lo bien o mal que lo ha hecho el agente.

En este caso, se premiará únicamente lo cerca que se encuentre la posición del efector final del manipulador del punto de la trayectoria que le corresponde en dicho instante temporal. A este término se denominará R_{dist}. Por el contrario, el agente deberá ser penalizado cuando su actuación provoque dolor o malestar en el brazo del sujeto que se mueve de manera solidaria a él. Por tanto, se penalizará cuando el robot excede los umbrales de fuerza o par definidos como molestos. A este término se denominará R_{safety}. También se penalizará al agente si ejecuta unas velocidades lineales o angulares elevadas. A estos términos se denominará $R_{linealVel}$ y $R_{angularVel}$ respectivamente.

A continuación, se realiza una definición formal de los diferentes términos que componen la función de recompensa:

- R_{dist}: Recompensa positiva basada en la cercanía del efector final a la posición ideal en la trayectoria. Se define mediante un decaimiento exponencial en función de la distancia euclídea d a la posición ideal:

$$R_{dist} = G \cdot e^{-\alpha d}$$

donde α es un parámetro de ganancia que regula la pendiente del decaimiento y G el valor máximo de recompensa, alcanzable cuando $d = 0$.

- $R_{linealVel}$: Penalización proporcional a la norma de la velocidad lineal del efector final, normalizada por una velocidad máxima $\mathbf{v}_{max}$:

$$R_{linealVel} = \beta \frac{\|\mathbf{v}_{lin}\|}{\|\mathbf{v}_{max}\|}$$

  donde β es un coeficiente de penalización y $\mathbf{v}_{lin}$ es el vector de velocidades lineales normalizado en el rango $[-1, 1]$. El vector $\mathbf{v}_{max}$ se define como el valor máximo de velocidad normalizada que se puede registrar, es decir $\mathbf{v}_{max} = [1, 1, 1]$.

- $R_{angularVel}$: Penalización proporcional a la norma de la velocidad angular del efector final, también normalizada por $\mathbf{v}_{max}$:

$$R_{angularVel} = \beta \frac{\|\mathbf{v}_{ang}\|}{\|\mathbf{v}_{max}\|}$$

  donde $\mathbf{v}_{ang}$ es el vector de velocidades angulares normalizado en el rango $[-1, 1]$.

- R_{safety}: Penalización fija si se exceden los umbrales máximos tolerables de fuerza o par medidos en la muñeca del manipulador:

$$R_{safety} = \begin{cases} G & \text{si } \max_i |F_{wrist,i}| \geq F_{threshold} \text{ o } \max_i |T_{wrist,i}| \geq T_{threshold} \\ 0 & \text{en otro caso} \end{cases}$$

La función de recompensa total R se define como:

$$R = R_{dist} - R_{linealVel} - R_{angularVel} - R_{safety}$$

Finalmente, comentar, que esta función de recompensa total es la que se ha codificado en el método `calculate_reward` llamado durante la ejecución del método `step` de la clase `CustomEnv` que está en el lado en que se realiza el aprendizaje (`RLSide`).

4.5. Obtención del sistema de referencia base

Como se ha mencionado anteriormente, las fuerzas y torques registrados en la muñeca del manipulador Franka Emika Panda están referenciados con respecto al sistema de coordenadas de dicha muñeca. Una de las observaciones

del método `step` de la clase `agent` heredada del `dm_robotics_panda` es la matriz de rotación que transforma este sistema al sistema de referencia de la base del manipulador. Esto facilitaría notablemente la tarea de no ser por la falta de documentación acerca de cuál es el sistema de la base y cómo se relaciona con el sistema global (`world`), en el que se especifican las poses del efector final y se comandan las velocidades cartesianas.

Hasta el momento no ha sido necesario transformar las fuerzas y pares al sistema global para el análisis de los datos. Sin embargo, en capítulos posteriores esto se volverá indispensable. Por este motivo, se considera oportuno documentar en esta sección la definición del sistema base y su correspondencia con el sistema global, así como el procedimiento experimental empleado para determinar dicha relación.

El estudio ha consistido en una serie de experimentos, de los cuales se presentan a continuación cuatro casos representativos:

- Mover el manipulador con velocidad fija en X positivo (hacia la derecha).
- Mover el manipulador con velocidad fija en X negativo (hacia la izquierda).
- Mover el manipulador con velocidad fija en Y positivo (hacia la adentro).
- Mover el manipulador con velocidad fija en Y negativo (hacia la afuera).

Para cada uno de estos esfuerzos se supone que el brazo humano que interactúa con el manipulador, en un momento de máxima contracción o estiramiento, según el caso, se opondrá de forma notable al sentido de movimiento. En ese instante se debería registrar un vector de fuerza en sentido contrario al impuesto por el manipulador.

Para cada experimento, se recogieron las fuerzas, pares y matrices de rotación, y se procedió a transformar dichos valores a distintos sistemas de referencia base, hasta encontrar aquel en el que los resultados coincidieran con la hipótesis planteada. En la figura 4.11 se muestra una imagen del entorno de simulación en la que se han representado los sistemas de referencia global (`world`) y base deducidos. Asimismo, en la figura 4.12 se presentan los resultados obtenidos en las cuatro pruebas descritas, evidenciando cómo los vectores de fuerza transformados desde el sistema de la muñeca al sistema base y posteriormente al sistema global, se oponen efectivamente al sentido del movimiento, tal y como se esperaba.

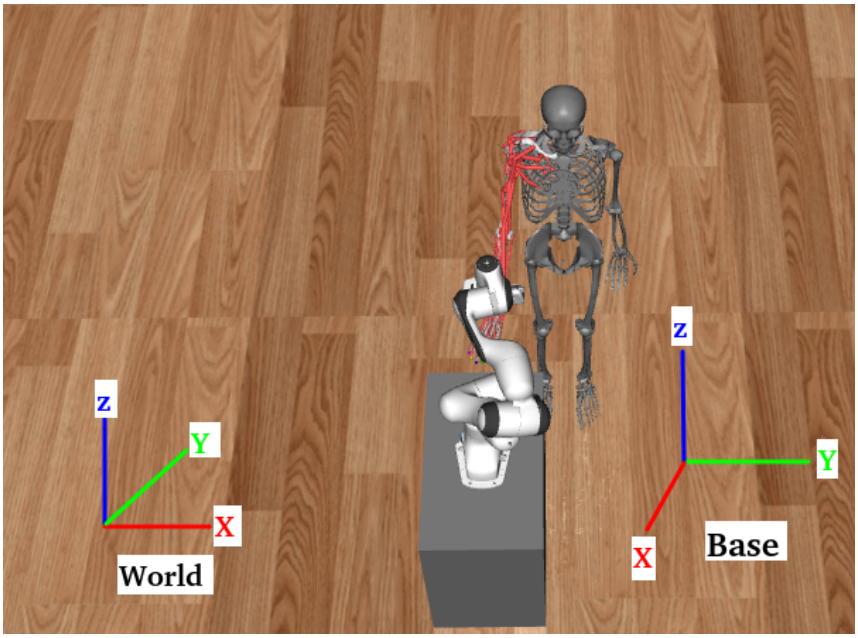
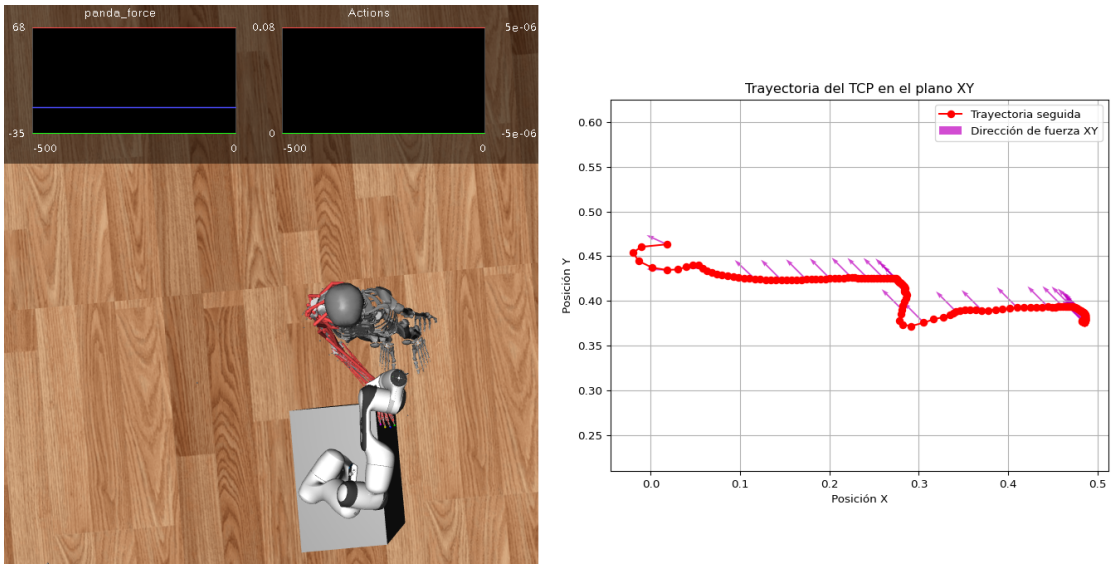
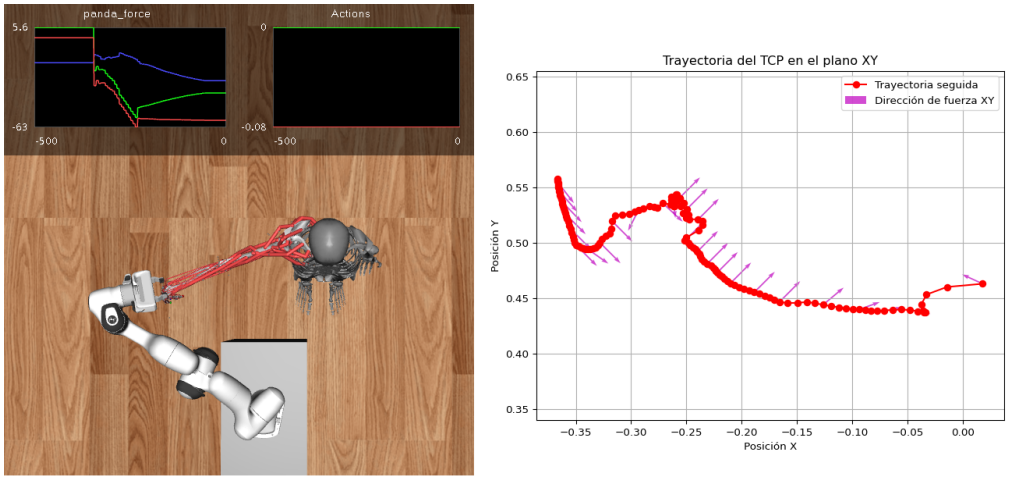


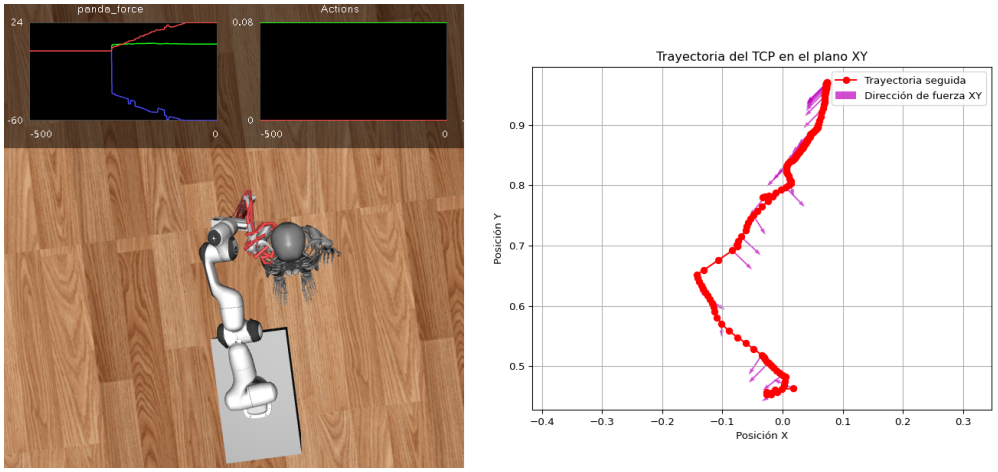
Figura 4.11: Ilustración del entorno de simulación en la que se han representado los sistemas de referencia global (`world`) y base.



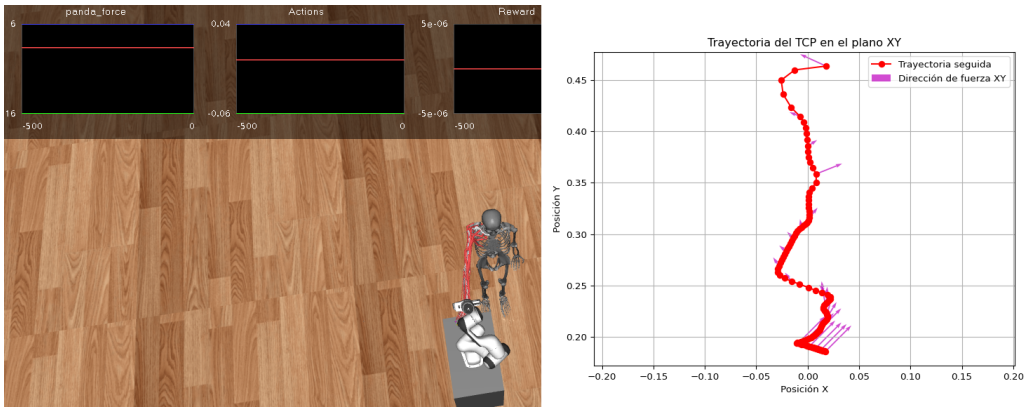
(a) Resultado de la prueba basada en mover el manipulador con velocidad fija en X positivo, comandada sobre sistema *world* (hacia la derecha).



(b) Resultado de la prueba basada en mover el manipulador con velocidad fija en X negativo, comandada sobre sistema *world* (hacia la izquierda).



(c) Resultado de la prueba basada en mover el manipulador con velocidad fija en Y positivo, comandada sobre sistema *world* (hacia adentro).



(d) Resultado de la prueba basada en mover el manipulador con velocidad fija en Y negativo, comandada sobre sistema *world* (hacia afuera).

Figura 4.12: Conjunto de resultados experimentales para la obtención del sistema de referencia base. Las direcciones de las fuerzas mostradas están referidas al sistema de referencia *world*.

Capítulo 5

Resultados del aprendizaje por refuerzo

Contenido

5.1. Análisis de hiperparámetros

El entrenamiento del agente basado en el algoritmo *Soft Actor-Critic* (SAC), proporcionado por la librería `Stable Baselines 3` ha requerido un cuidadoso ajuste de hiperparámetros con el fin de garantizar una convergencia estable durante el entrenamiento para la función de recompensa definida.

La función de recompensa R ha sido diseñada para equilibrar dos objetivos fundamentales: el seguimiento preciso de una trayectoria deseada por parte del

efector final del manipulador, y la garantía de seguridad y confort en la interacción física con el brazo humano simulado. A continuación, se describen los distintos términos que integran la función de recompensa, con el fin de detallar la configuración de hiperparámetros adoptada en cada caso. Para una comprensión más precisa de la formulación de dichos términos, se remite al capítulo 4.

- **Recompensa de proximidad** (R_{dist}): incentiva al agente a mantener la posición del efector final próxima a la trayectoria ideal. El parámetro α regula la pendiente del decaimiento y se configura a $\alpha = 2$. La variable G controla el valor máximo de recompensa y se establece a $G = 10$, alcanzable cuando la distancia euclídea entre el efector final del manipulador y el punto correspondiente de la trayectoria es $d = 0$. La recompensa se encuentra acotada en el rango $[-10, 10]$, ya que en el artículo original de SAC se demuestra que con este escalado el algoritmo converge alcanzando la solución más óptima en el menor tiempo, en comparación a otros escalados [10].
- **Penalizaciones por velocidad** ($R_{linealVel}$ y $R_{angularVel}$): penalizan movimientos bruscos o no deseados, considerando las componentes lineales y angulares de la velocidad del efector final. El coeficiente β regula la penalización máxima y se configura a $\beta = 0{,}75$ en ambos casos.
- **Penalización por seguridad** (R_{safety}): impone una penalización fija si se exceden los umbrales definidos de fuerza o par medidos en la muñeca del manipulador. Las variables $F_{threshold}$ y $T_{threshold}$ son respectivamente los umbrales que definen la sensación de incomodidad física en el sujeto, configurados a $F_{threshold} = 15N$ y $T_{threshold} = 35Nm$. Ambos resultados fueron obtenidos de la experimentación descrita en el capítulo 4. El valor de G se establece a 10, por los motivos comentados en el término de recompensa por proximidad.

El comportamiento del agente ha sido evaluado en un entorno con trayectorias variables, alternando entre configuraciones (i) cuadradas, en sentido horario y antihorario, (ii) circulares, en sentido horario y antihorario, (iii) triangulares, únicamente en sentido horario. Estas trayectorias se alternan dinámicamente cada 100 episodios para fomentar la generalización de la política aprendida. Es decir, cada 100 episodios se proporciona una trayectoria diferente al agente, tratando así de evitar que ajuste su política a las trayectorias proporcionadas en vez de generalizar su comportamiento ante trayectorias nuevas.

La longitud máxima de los episodios se ha fijado en 400 pasos, pudiendo finalizar antes en caso de exceder alguno de los umbrales de fuerza o par fijados

para la sensación de dolor (25 N en magnitud de fuerza o los 55 Nm en magnitud de par).

El tiempo de acción se ha fijado a 0.7 segundos. Durante este periodo, el agente (`AgenSide`) mantiene constante la velocidad cartesiana del efector final inferida por la política. Transcurrido dicho intervalo, se transmite la última observación recopilada al lado del aprendizaje (`RLSide`).

Anteriormente, se ha descrito la configuración de los hiperparámetros referentes al la función de recompensa. A continuación, se procede a detallar aquellos relacionados con la configuración del algoritmo SAC:

- `policy = "MlpPolicy"`: Tipo de arquitectura de política empleada por el agente. En este caso, se utiliza una red neuronal multicapa (Multi-Layer Perceptron, MLP).
- `learning_rate = 0.0005`: Tasa de aprendizaje del optimizador Adam [11]. Este valor se emplea en todas las redes: actor, de valor objetivo y crítico.
- `learning_starts = 10000`: Número de pasos iniciales antes de comenzar el entrenamiento del agente. Durante estos pasos, el modelo solo recopila experiencias y las almacena en el *replay buffer*.
- `batch_size = 256`: Tamaño del *minibatch* utilizado en cada paso de optimización.
- `gamma = 0.9999`: Factor de descuento (γ).
- `train_freq = (1, "step")`: Frecuencia con la que se actualiza el modelo. En este caso, se entrena el modelo tras cada paso de simulación.
- `gradient_steps = 1`: Número de pasos de gradiente a realizar tras cada ciclo de entrenamiento. En este caso, se realiza una sola actualización por paso.

La descripción de los diferentes hiperparámetros ha sido extraída de la documentación que proporciona SB3 para su implementación del algoritmo SAC [21].

Con esta configuración se ha logrado entrenar un agente capaz de seguir diversas trayectorias mientras desplaza de forma conjunta un brazo humano, evitando transicionar a estados que puedan inducir sensaciones de dolor o generar resistencia mecánica percibida por el usuario.

Alcanzar esta configuración ha sido el fruto de más de 3300 horas de entrenamiento, llevadas a cabo en cuatro dispositivos diferentes. Entrenar por un número de horas tan elevado, que equivale unos 140 días, ha sido posible gracias a disponer de 4 ordenadores con capacidad para realizar múltiples entrenamientos en paralelo. A modo de recapitulación, se recoge en la tabla 5.1 el *hardware* de cada dispositivo, el número total de entrenamientos realizados y el número de horas que se ha utilizado para entrenar.

CPU	**GPU**	**Nº Entrenamientos**	**Horas Totales**
Intel Core i5-14600KF	NVIDIA GTX 1080	45	1562
Intel Core i9-10900KF	NVIDIA GT 710	17	928
Intel Core i5-13400F	NVIDIA RTX 3060	9	621
Intel Core i5-8300H	NVIDIA RTX 1050 Laptop	6	279

Tabla 5.1: Resumen de recursos computacionales utilizados para el entrenamiento de los agentes.

Cabe destacar que, no sólo se ha empleado el algoritmo SAC en la totalidad de los entrenamientos: también se llevaron a cabo experimentos utilizando el algoritmo *Twin Delayed Deep Deterministic Policy Gradient* (TD3) [9], el cual, aunque no ha sido descrito en el capítulo 2, puede considerarse una evolución del algoritmo DDPG previamente estudiado [13].

Pronto se evidenció que los entrenamientos con SAC resultaban más eficientes, gracias a su capacidad de fomentar la exploración efectiva, al incorporar una política estocástica que maximiza tanto la recompensa esperada como la entropía de la propia política.

5.2. Resultados del aprendizaje

Con el objetivo de evaluar la robustez y reproducibilidad de la solución propuesta, se llevaron a cabo cuatro entrenamientos independientes bajo condiciones idénticas: mismos hiperparámetros, configuración del entorno, arquitectura de red y una semilla aleatoria. Todos los agentes se entrenaron hasta completar aproximadamente 4.5 millones de pasos de interacción con el entorno.

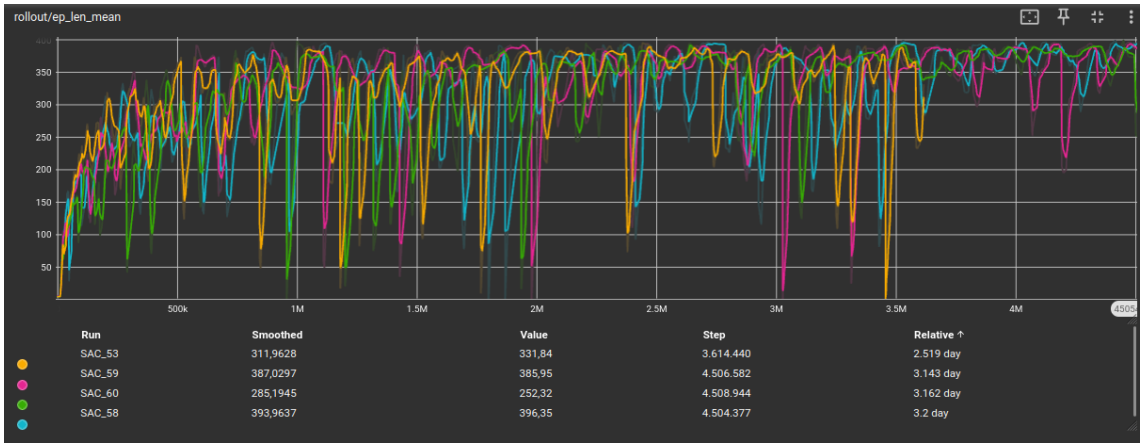


Figura 5.1: Evolución de la longitud media del episodio (`episode_length_mean`) durante el entrenamiento de cuatro agentes independientes con configuración idéntica.

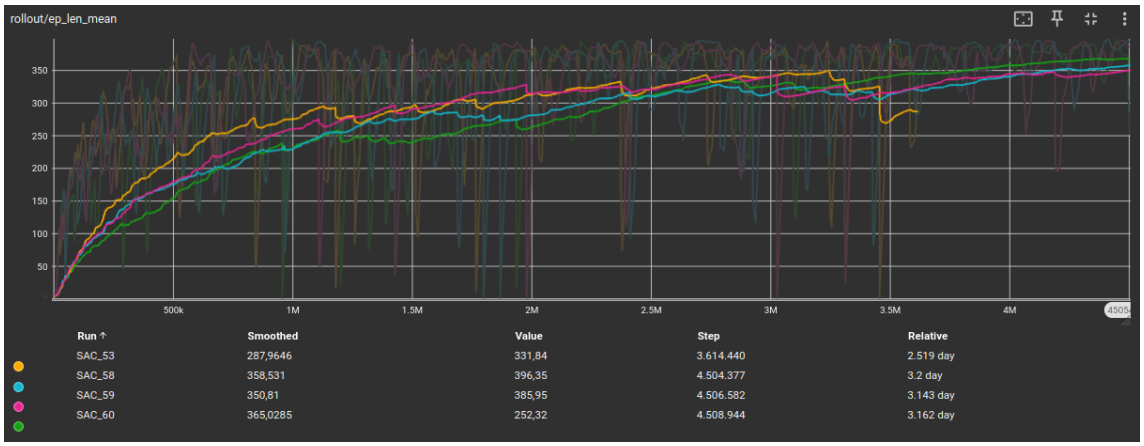


Figura 5.2: Evolución de la longitud media del episodio (`episode_length_mean`) durante el entrenamiento de cuatro agentes independientes con configuración idéntica. Las curvas han sido suavizadas para resaltar las tendencias.

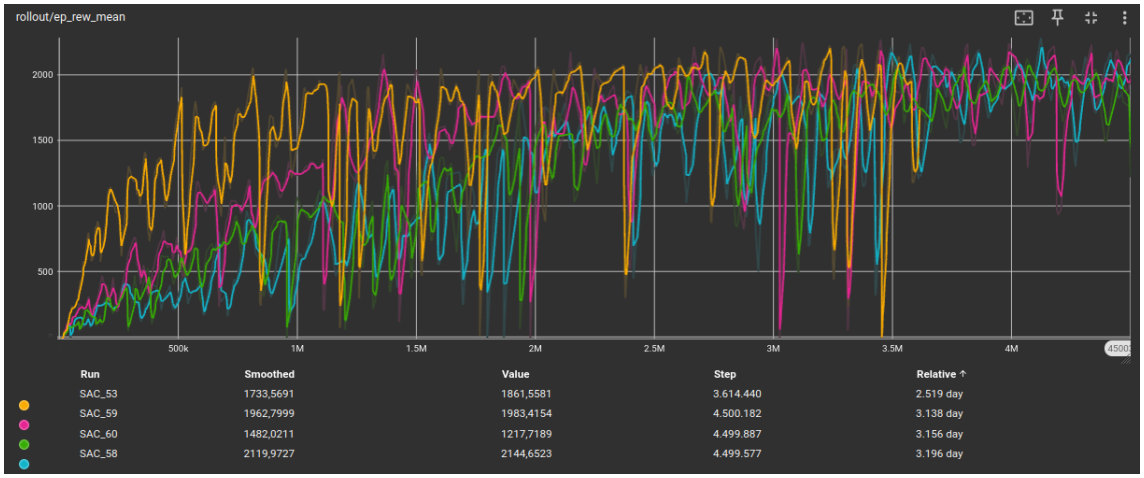


Figura 5.3: Evolución de la recompensa media por episodio (`episode_reward_mean`) durante el entrenamiento de cuatro agentes independientes con configuración idéntica.

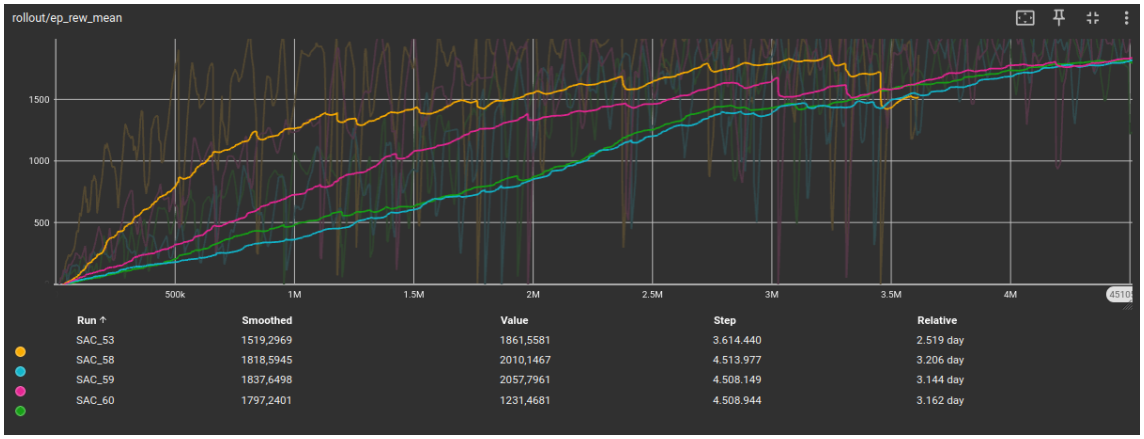


Figura 5.4: Evolución de la recompensa media por episodio (`episode_reward_mean`) durante el entrenamiento de cuatro agentes independientes con configuración idéntica. Las curvas han sido suavizadas para resaltar las tendencias.

Pese al carácter ruidoso de métricas como `episode_reward_mean` y `episode_length_mean`, tal como se visualiza en las gráficas 5.1 y 5.3 generadas mediante *TensorBoard* [22], al observar las figuras 5.2 y 5.4 (con suavizado aplicado) se aprecia que los cuatro agentes convergen hacia una misma solución en términos de comportamiento y retorno esperado alrededor de los 4 millones de pasos. Este patrón sugiere que, si bien los procesos de entrenamiento individuales pueden diferir localmente, la política aprendida es globalmente estable y reproducible, validando así la consistencia del diseño del entorno, de la función

de recompensa y de los hiperparámetros seleccionados.

En la gráfica mostrada en la figura 5.2 se observa que la longitud media de los episodios converge hacia los 360 pasos, valor que se aproxima significativamente a la longitud máxima permitida por episodio (400 pasos). Este comportamiento indica que el agente es capaz de completar la trayectoria sin generar movimientos bruscos que excedan los umbrales de fuerza o par que pondrían fin al episodio de manera temprana.

Por otro lado, en la figura 5.4 se aprecia que el retorno medio por episodio converge en torno a un valor de 1800. Considerando que la media de pasos en un episodio es de 360, se deduce que la recompensa media por paso es de aproximadamente 5. Considerando que la recompensa máxima definida por el término R_{dist} es 10, este valor representa el 50 % de dicho máximo. Si se despeja el valor de la distancia d en la expresión de R_{dist}, teniendo en cuenta que dicha distancia está normalizada en el rango $[-1, 1]$ para un máximo de 15 cm, se obtiene que la distancia euclídea media entre el efector final del manipulador y el punto ideal de la trayectoria en dicho paso es de aproximadamente 5 cm.

5.3. Resultados del mejor entrenamiento

En esta sección se va a realizar un análisis de las capacidades del agente resultante. Para ello, se ha sometido al agente, una vez entrenado, al seguimiento de cuatro trayectorias, dos experimentadas durante el aprendizaje y otras dos desconocidas. Las trayectorias sobre las que se ha evaluado son las siguientes (i) trayectoria cuadrada en sentido antihorario, (ii) trayectoria triangular en sentido horario, (iii) trayectoria triangular en sentido antihorario, (iv) trayectoria pentagonal en sentido antihorario.

5.3.1. Trayectoria cuadrada en sentido antihorario

En la figura 5.5 se representan, a la derecha, cuatro iteraciones de la trayectoria cuadrada sobre el plano XY con sentido de giro antihorario. Como puede apreciarse, el resultado no se ajusta a la perfección, pero la forma de la figura es clara. Además, la reproducibilidad de las trazas es notable. Esto denota que el agente no ha podido mejorar la descripción de la trayectoria debido a fuerzas o pares que generarían sensación de malestar en el sujeto, lo cual puede verse en la gráfica de la izquierda, donde se muestra la trayectoria descrita en el espacio tridimensional; sobre esta se ha mostrado el sentido y módulo del vector fuerza transformado al sistema global en diferentes instantes. En los laterales

derecho e izquierdo de la trayectoria descrita se aprecia claramente cómo las fuerzas medidas apuntan al centro, lo que quiere decir que el brazo humano se resiste al movimiento en esa área, impidiendo la perfecta ejecución. Para apreciar la trayectoria y vectores de la fuerza con mayor claridad se han representado únicamente dos iteraciones de la trayectoria.

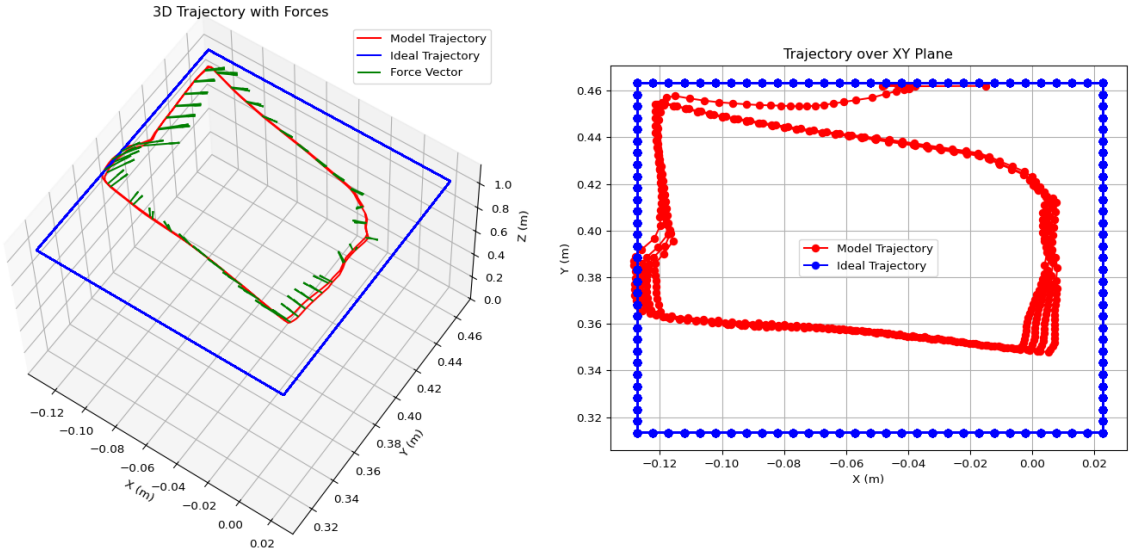


Figura 5.5: Resultados para la trayectoria cuadrada. Comparación de la trayectoria seguida por el agente respecto a real sobre el plano XY (derecha). Comparación de la trayectoria seguida por el agente respecto a real en el espacio tridimensional ilustrando sentido de la fuerza en diferentes pasos de la misma (izquierda).

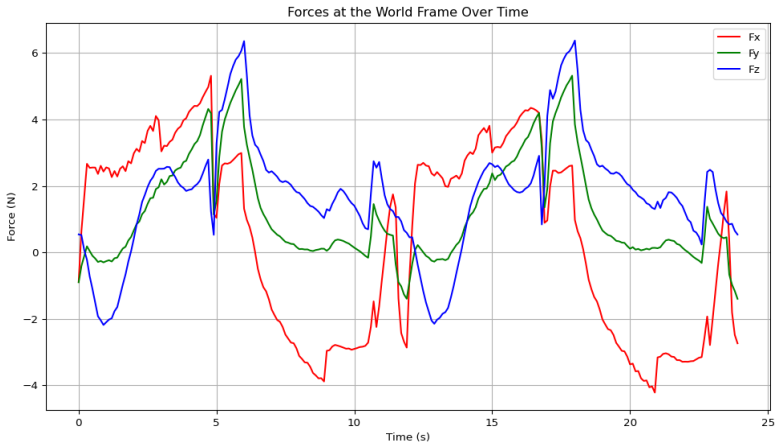


Figura 5.6: Fuerzas del efector final en la ejecución de la trayectoria cuadrada, transformadas al sistema global.

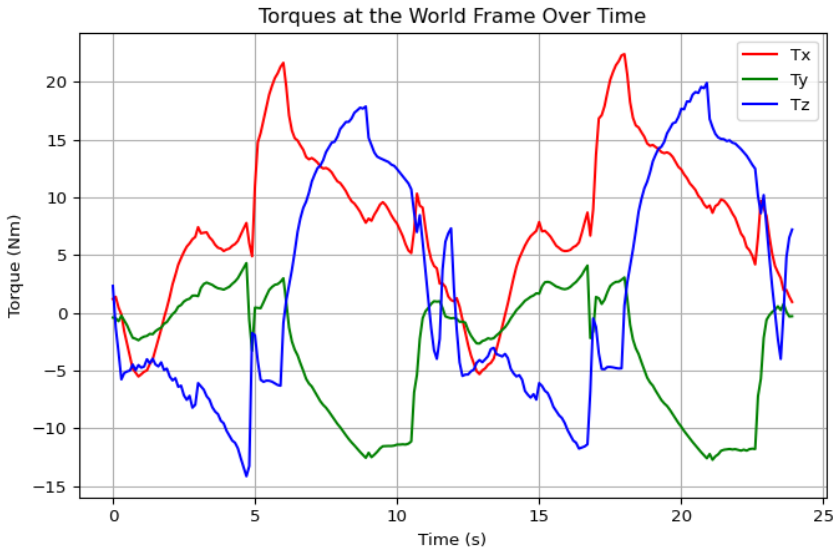


Figura 5.7: Pares del efector final en la ejecución de la trayectoria cuadrada, transformados al sistema global.

Las figuras 5.6 y 5.7 muestran las fuerzas y pares medidos en la muñeca del manipulador tras haber sido transformados al sistema de referencia global durante dos iteraciones. Es importante notar cómo en ningún momento se superan los umbrales de molestia física.

La trayectoria cuadrada tiene un tiempo estipulado de 12 segundos para su recorrido (por iteración), correspondiendo 3 segundos por lado. En la gráfica 5.7 se aprecia cómo entre los 3 s y los 6 s se produce un pico en el par registrado sobre el eje X; esto se corresponde con el intervalo temporal en que se está describiendo el lado inferior del cuadrado, que es precisamente el que presenta un mayor error en términos de distancia euclídea respecto a la trayectoria ideal. Estos resultados demuestran que el agente prioriza la seguridad y bienestar del sujeto frente a la ejecución de la trayectoria ideal, lo cual está alineado con las métricas de recompensa establecidas para el entrenamiento.

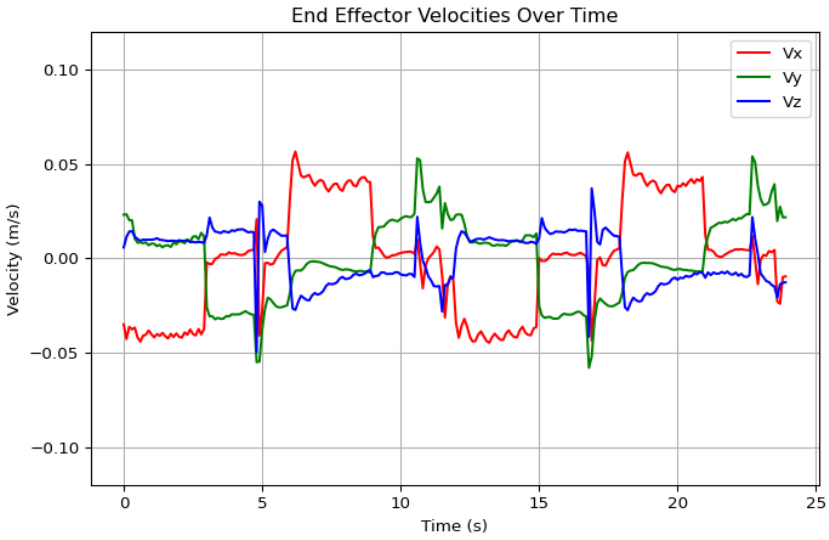


Figura 5.8: Velocidades lineales del efector final en la ejecución de la trayectoria cuadrada, transformadas al sistema global.

En la figura 5.8 se pueden apreciar las velocidades lineales del efector final para cada eje espacial referenciadas al sistema global. En ningún momento se supera el umbral de los 0.25 m/s establecidos como máximo. De hecho, se queda muy por debajo de este valor. Esto es gracias a haber introducido los términos $R_{linealVel}$ y $R_{angularVel}$ en la función de recompensa para penalizar velocidades elevadas que se traducirían en movimientos bruscos por parte del manipulador. Además, se aprecia cómo las velocidades se mantienen contenidas alrededor de los 0.05 m/s, que se recuerda, es la velocidad a la que se ha configurado la trayectoria ideal.

5.3.2. Trayectoria triangular en sentido horario

En la figura 5.9 se representan, a la derecha, cuatro iteraciones de la trayectoria triangular, en sentido horario, sobre el plano XY. Como puede apreciarse, el resultado sigue sin ajustarse a la perfección a la trayectoria ideal, siendo especialmente errático en la primera vuelta. La reproducibilidad de la trayectoria descrita se recobra de mediados de la segunda vuelta hasta el final de la cuarta. En la figura de la derecha se puede observar, por el sentido y la magnitud de los vectores de fuerza, por qué se desvía de la trayectoria al describir el lateral derecho del triángulo y por qué no llega a alcanzar el lateral izquierdo. Esto evidencia claramente la presencia de la resistencia del brazo humano simulado durante la ejecución de esta trayectoria.

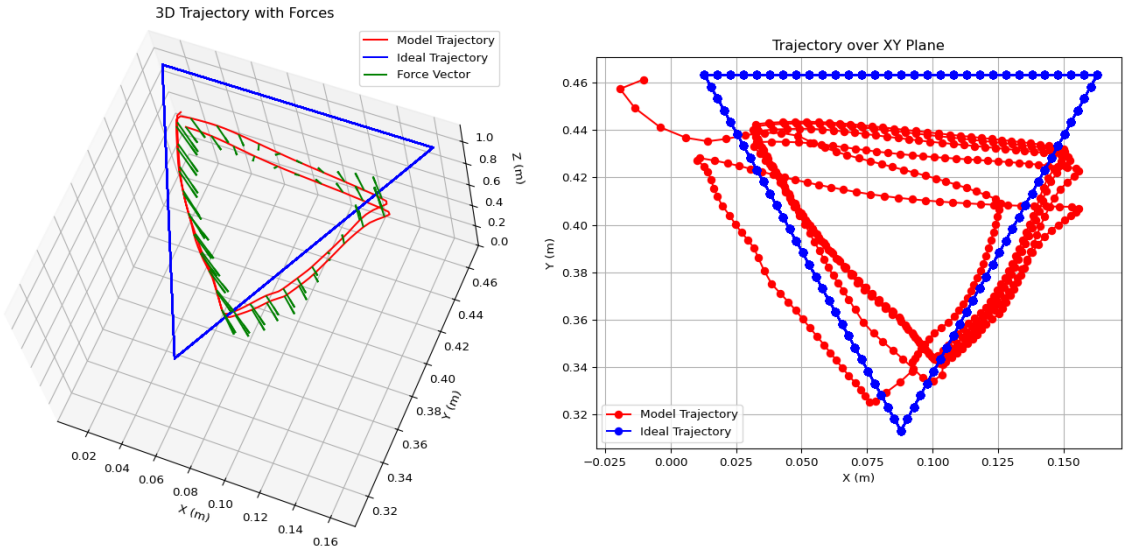


Figura 5.9: Resultados para la trayectoria triangular en sentido horario. Comparación de la trayectoria seguida por el agente respecto a la real sobre el plano XY (derecha). Comparación de la trayectoria seguida por el agente respecto a la real en el espacio tridimensional, ilustrando el sentido de la fuerza en diferentes pasos de la misma (izquierda).

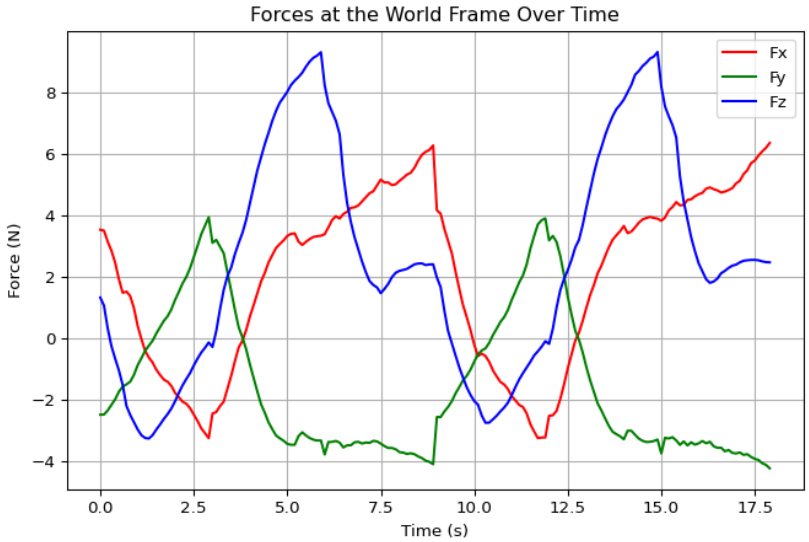


Figura 5.10: Fuerzas del efector final en la ejecución de la trayectoria triangular en sentido horario, transformadas al sistema global.

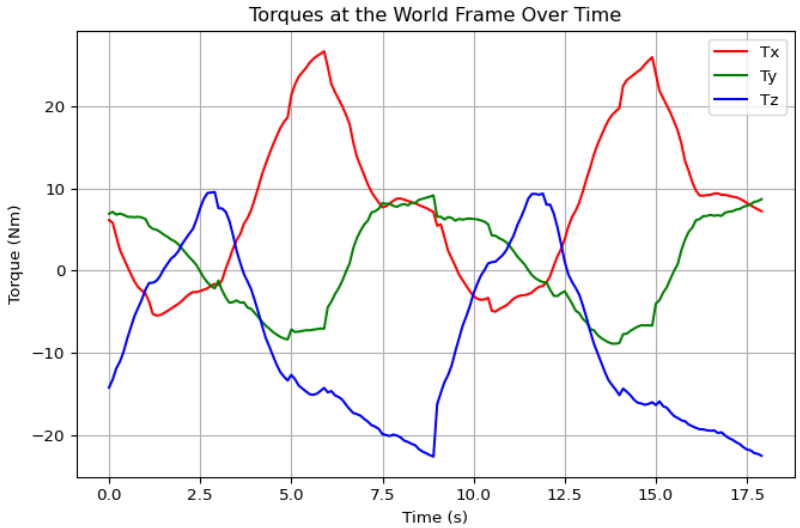


Figura 5.11: Pares del efector final en la ejecución de la trayectoria triangular en sentido horario, transformados al sistema global.

Las figuras 5.10 y 5.11 muestran las fuerzas y pares registrados y transformados a sistema global. De nuevo, el agente se mantiene por debajo de los umbrales de fuerza y par. La trayectoria triangular está configurada para ser recorrida en 9 segundos, tres por lado. Entre los 3 s y los 6 s se experimentan picos en la fuerza (eje Z) y los pares (ejes X y Z), lo cual impide al agente describir el lado izquierdo del triángulo de manera correcta, viéndose forzado a desplazarse hacia el centro de la figura para no exceder los umbrales establecidos. Al representarse dos vueltas, los picos se vuelven a manifestar durante la segunda iteración, entre los 12 s y los 15 s.

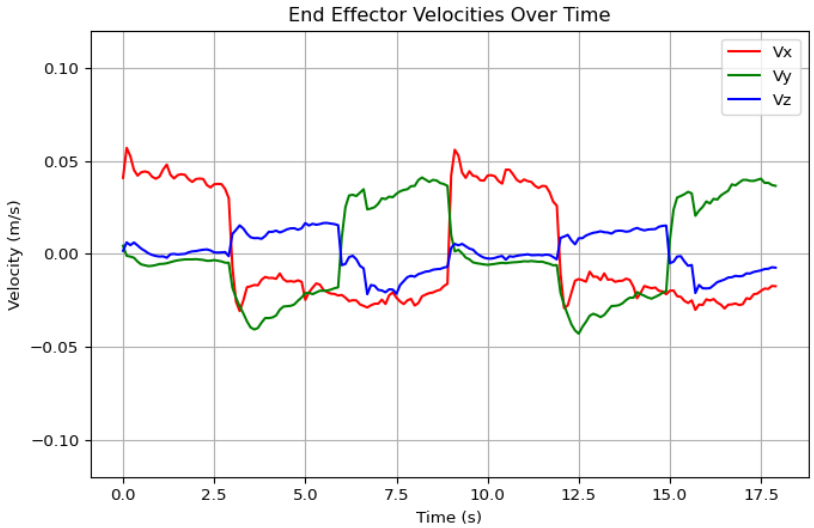


Figura 5.12: Velocidades lineales del efector final en la ejecución de la trayectoria triangular en sentido horario, transformadas al sistema global.

En la figura 5.12 se pueden apreciar las velocidades lineales del efector final, que de nuevo se mantienen notablemente por debajo del máximo establecido.

5.3.3. Trayectoria triangular en sentido antihorario

En la figura 5.13 se representan, a la derecha, cuatro iteraciones de la trayectoria triangular en sentido antihorario sobre el plano XY. Siendo notable que la trayectoria descrita se encuentra ligeramente desplazada a la derecha respecto de la ideal. Es importante destacar que esta trayectoria era desconocida para el agente, es decir, no se había enfrentado con ella. Se nota una clara reproducibilidad en la ejecución. Si se visualiza la figura de la izquierda, es fácil percatarse de que el sentido de la fuerza apunta a la derecha en gran parte de la trayectoria, coincidiendo con el desplazamiento que se percibía visualmente.

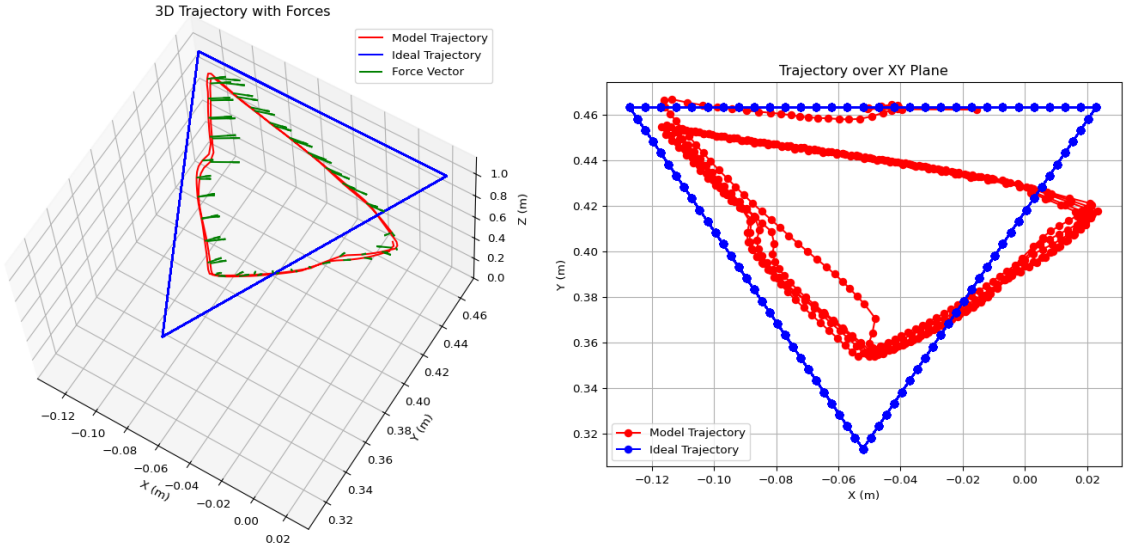


Figura 5.13: Resultados para la trayectoria triangular en sentido antihorario. Comparación de la trayectoria seguida por el agente respecto a la real sobre el plano XY (derecha). Comparación de la trayectoria seguida por el agente respecto a la real en el espacio tridimensional, ilustrando el sentido de la fuerza en diferentes pasos de la misma (izquierda).

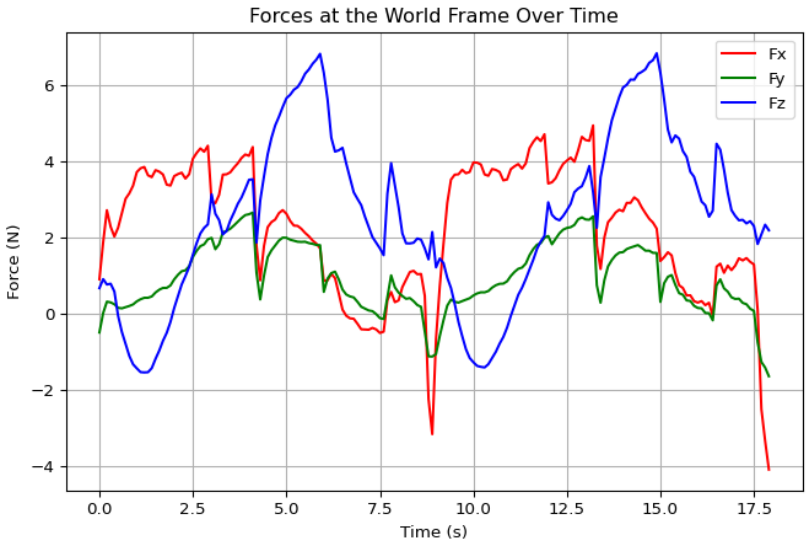


Figura 5.14: Fuerzas del efector final en la ejecución de la trayectoria triangular en sentido antihorario, transformadas al sistema global.

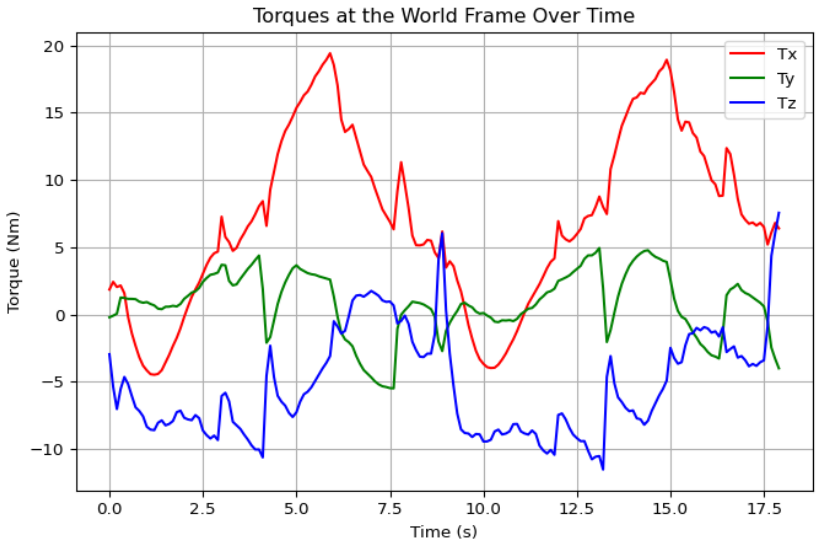


Figura 5.15: Pares del efector final en la ejecución de la trayectoria triangular en sentido antihorario, transformados al sistema global.

Las figuras 5.14 y 5.15 muestran las fuerzas y pares registrados y transformados a sistema global. De nuevo, pese a enfrentarse a una trayectoria desconocida, el agente es capaz de seguirla manteniéndose por debajo de los umbrales establecidos para las magnitudes de fuerza y par.

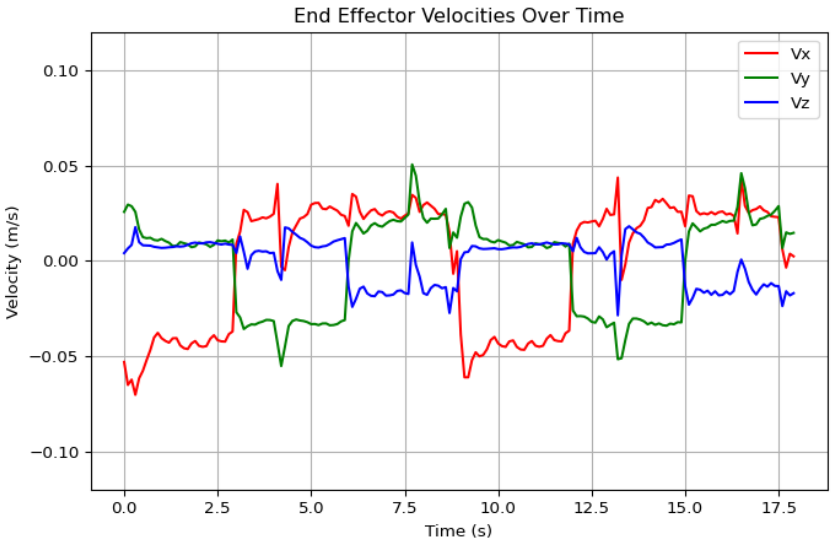


Figura 5.16: Velocidades lineales del efector final en la ejecución de la trayectoria triangular en sentido antihorario, transformadas al sistema global.

De igual modo, la figura 5.16 muestra las velocidades lineales del efector final. Sobre los ejes X e Y, se puede observar cómo se mantiene en torno a -0.05 m/s por 3 s para pasar a 0.025 m/s por 6 s, coincidiendo con las velocidades ideales con las que se ha generado la trayectoria.

5.3.4. Trayectoria pentagonal en sentido antihorario

En la figura 5.17 se representan, a la derecha, cuatro iteraciones de la trayectoria pentagonal en sentido antihorario sobre el plano XY. Esta es de nuevo una trayectoria desconocida para el agente, en este caso no solo por el sentido de giro, sino también por la forma. De nuevo, los resultados no son perfectos, pero se ajusta a la forma en términos generales y la reproducibilidad de la ejecución es notable. Una vez más, parece que la trayectoria descrita se encuentra desplazada hacia la derecha, hipótesis que se contrasta visualizando el sentido de la fuerza en la figura de la izquierda.

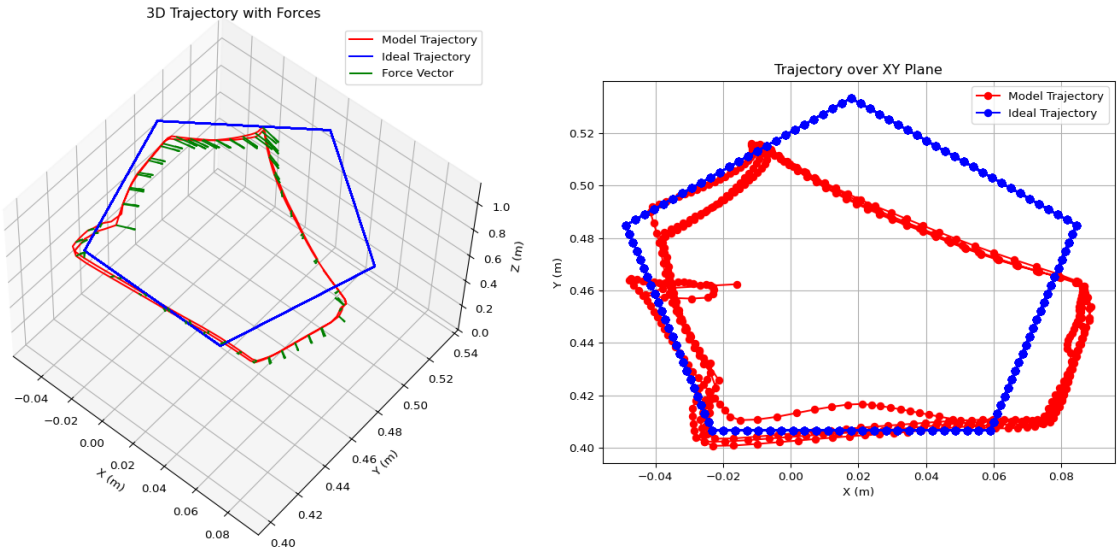


Figura 5.17: Resultados para la trayectoria pentagonal en sentido antihorario. Comparación de la trayectoria seguida por el agente respecto a la real sobre el plano XY (derecha). Comparación de la trayectoria seguida por el agente respecto a la real en el espacio tridimensional, ilustrando el sentido de la fuerza en diferentes pasos de la misma (izquierda).

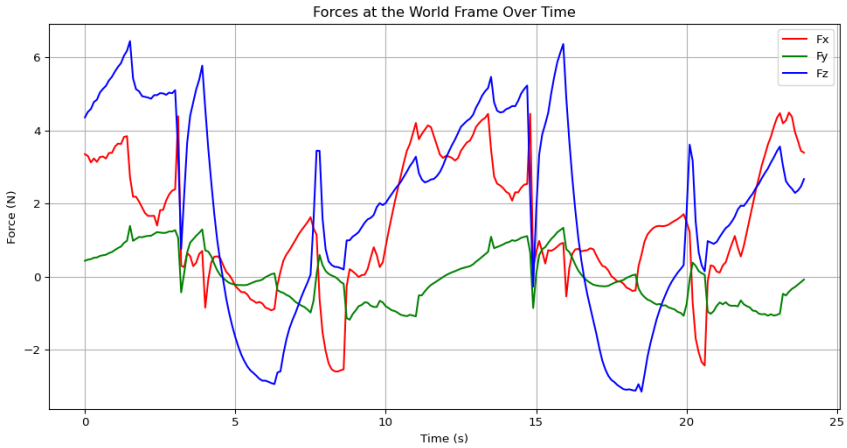


Figura 5.18: Fuerzas del efector final en la ejecución de la trayectoria pentagonal en sentido antihorario, transformadas al sistema global.

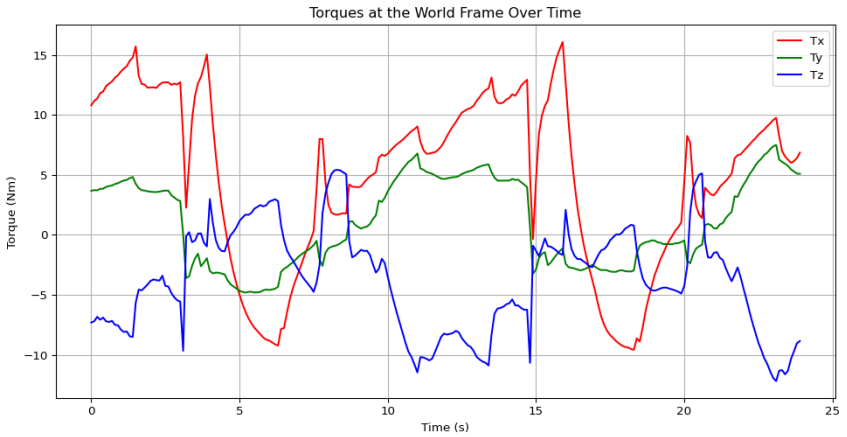


Figura 5.19: Pares del efector final en la ejecución de la trayectoria pentagonal en sentido antihorario, transformados al sistema global.

Las figuras 5.18 y 5.19 muestran las fuerzas y pares medidos durante el experimento y transformados a sistema global. Una vez más, pese a enfrentarse a una trayectoria desconocida, en este caso en concepto de forma, el agente es capaz de seguirla manteniéndose por debajo de los umbrales establecidos para las magnitudes de fuerza y par. Además, entre los 8 s y los 15 s se registran unos picos notables en la fuerza sobre los ejes X y Z, lo cual se corresponde con el intervalo temporal en el que se está describiendo el lateral derecho del pentágono, donde el error es mayor. La trayectoria pentagonal está configurada para ser descrita en 12 segundos, por lo que entre los 8 s y los 15 s se está llevando

a cabo el final de la primera vuelta y el inicio de la segunda.

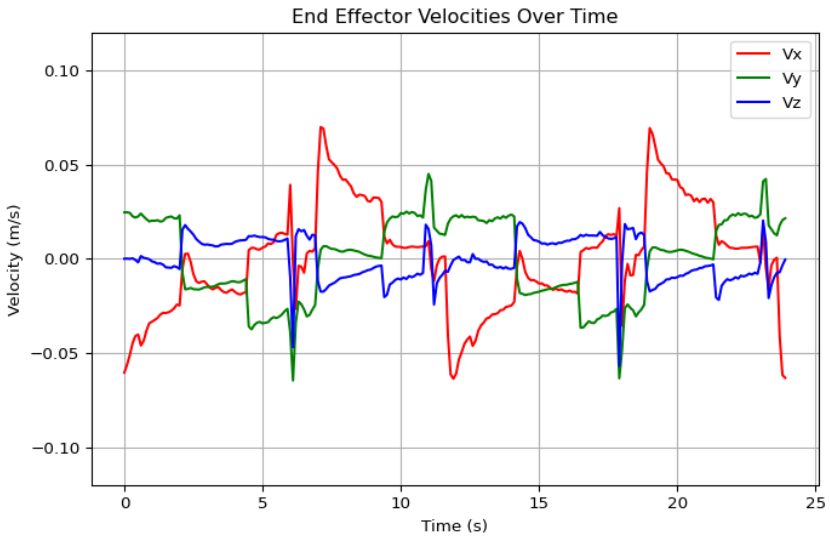
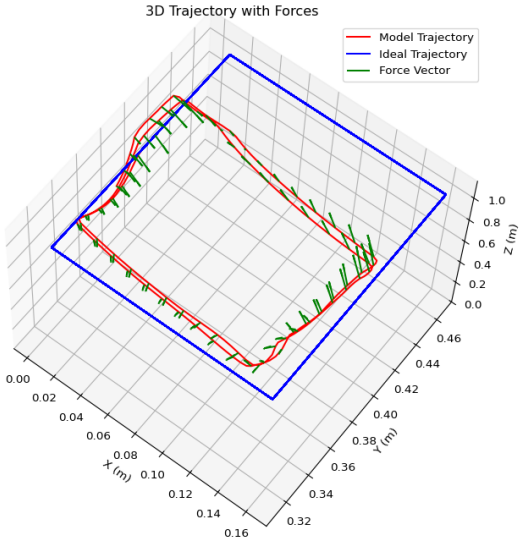


Figura 5.20: Velocidades lineales del efector final en la ejecución de la trayectoria pentagonal en sentido antihorario, transformadas al sistema global.

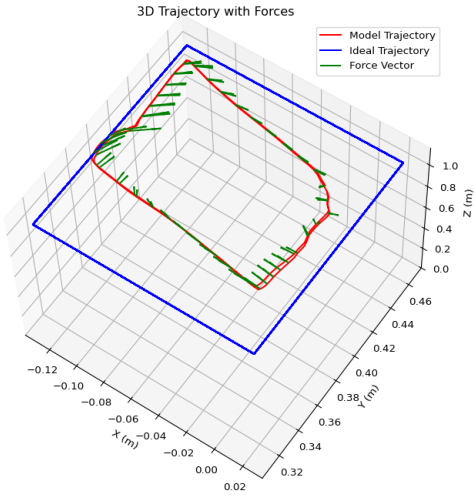
Una vez más, la figura 5.20 se pueden apreciar las velocidades lineales del efector final, que se mantienen notablemente por debajo del máximo establecido, demostrando la capacidad de generalizar del modelo ante nuevos estados.

5.3.5. Resumen de resultados

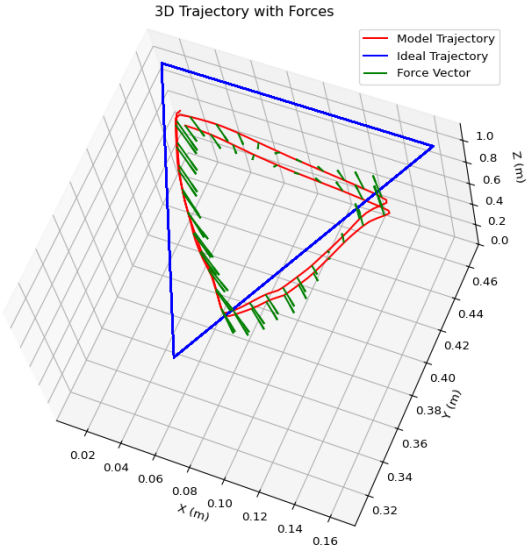
En la figura 5.21 se presenta la trayectoria recorrida por el agente durante dos iteraciones consecutivas para las distintas configuraciones evaluadas. Sobre esta trayectoria se han representado el sentido y módulo del vector fuerza cada 3 pasos. Cada una de las trayectorias adopta la forma de una figura geométrica básica proyectada sobre el plano XY, pero situada a diferentes niveles a lo largo del eje Z, dentro del rango $[0{,}925, 1{,}075]m$. Esta variación en altura busca simular la adaptación de la tarea de rehabilitación a usuarios con distintas estaturas, lo que puede implicar cambios en la posición del punto de agarre del manipulador.



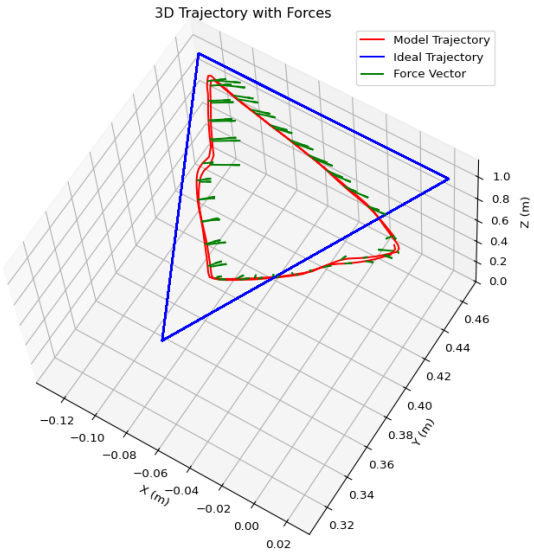
(a) Trayectoria cuadrada sobre el espacio tridimensional en sentido horario.



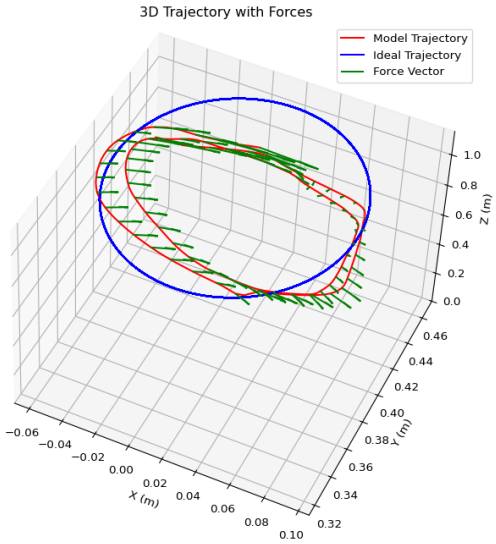
(b) Trayectoria cuadrada sobre el espacio tridimensional en sentido antihorario.



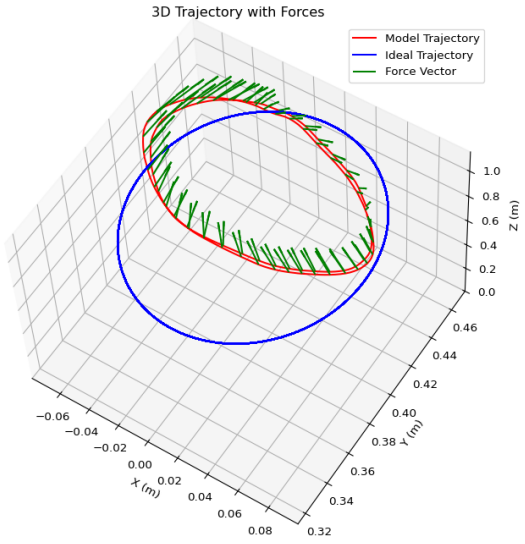
(c) Trayectoria triangular sobre el espacio tridimensional en sentido horario.



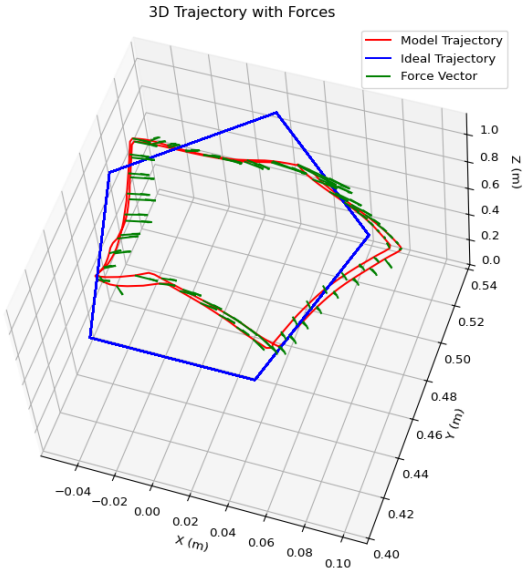
(d) Trayectoria triangular sobre el espacio tridimensional en sentido antihorario.



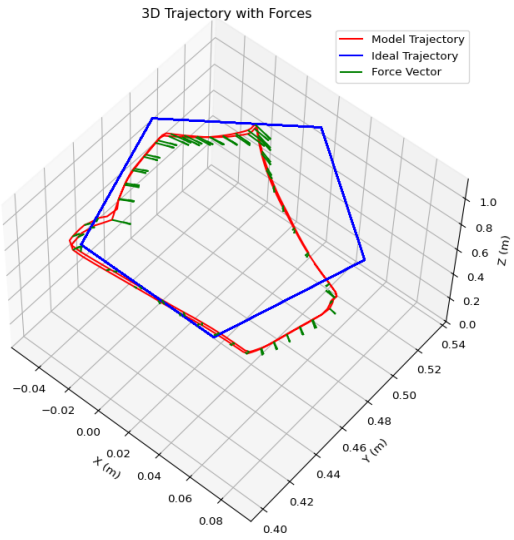
(e) Trayectoria circular sobre el espacio tridimensional en sentido horario.



(f) Trayectoria circular sobre el espacio tridimensional en sentido antihorario.



(g) Trayectoria pentagonal sobre el espacio tridimensional en sentido horario.



(h) Trayectoria pentagonal sobre el espacio tridimensional en sentido antihorario.

Figura 5.21: Comparación de las trayectorias seguidas por el agente en el espacio tridimensional para distintas formas y sentidos de giro.

Cabe recordar que el agente había experimentado previamente, durante el entrenamiento, cinco de las trayectoria:

- Trayectoria cuadrada en sentido horario.
- Trayectoria cuadrada en sentido antihorario.
- Trayectoria circular en sentido horario.
- Trayectoria circular en sentido antihorario.
- Trayectoria triangular en sentido horario.

Por otro lado, las tres trayectorias restantes eran desconocidas para el agente:

- Trayectoria triangular en sentido antihorario.
- Trayectoria pentagonal en sentido horario.
- Trayectoria pentagonal en sentido antihorario.

Tras haber evaluado en detalle las capacidades del agente resultante, se concluye que ha aprendido la tarea. Esto se evidencia en su capacidad para seguir tanto trayectorias previamente experimentadas como nuevas, sin exceder en ningún momento los umbrales establecidos para las magnitudes de fuerza y par, lo que garantiza la seguridad física del sujeto. Sin embargo, la forma de algunas trayectorias ejecutadas resulta insatisfactoria. Esta discrepancia respecto a las trayectorias ideales puede explicarse al analizar los vectores de fuerza sobre el recorrido o los registros de fuerza y par durante la prueba. Dichos análisis revelan que el agente adapta su comportamiento frente a la resistencia del brazo humano simulado, evitando provocar sensaciones de dolor o forcejeo.

Cabe destacar que la gestión del dolor en contextos de rehabilitación es especialmente compleja. Aunque los registros sugieren que el agente evita generar sensaciones de dolor o forcejeo, es importante tener en cuenta que la percepción del dolor varía considerablemente entre individuos. En este sentido, lo que desde un punto de vista técnico puede considerarse seguro o aceptable, podría no serlo desde la perspectiva del paciente. Esta variabilidad subraya la necesidad de incorporar evaluaciones con usuarios reales, para validar plenamente la adecuación del comportamiento del agente en entornos clínicos.

5.4. Impacto del número de trayectorias en el desempeño del agente

Durante la fase de entrenamiento, se observó que la exposición del agente a múltiples trayectorias promovía una mejora significativa en su desempeño general, favoreciendo su capacidad de generalización ante trayectorias no vistas previamente.

No obstante, en la fase de evaluación se formuló la hipótesis de que dicha generalización podría comprometer la precisión del agente al seguir trayectorias específicas, debido a la diversidad de experiencias acumuladas durante el aprendizaje.

Con el objetivo de contrastar esta hipótesis, se procedió a entrenar un agente manteniendo la misma configuración de hiperparámetros previamente descrita, restringiendo en este caso el entrenamiento a una única trayectoria: un cuadrado en sentido antihorario.

En la figura 5.22 se muestran dos vueltas completas a la trayectoria cuadrada en sentido antihorario ejecutadas por el agente entrenado únicamente con esta trayectoria. A pesar de haber sido entrenado específicamente con ella, el agente no consigue ajustarse con precisión al recorrido ideal, presentando desviaciones notables, especialmente en el lateral izquierdo del cuadrado. Esto sugiere que la resistencia ofrecida por el brazo humano al movimiento sigue teniendo un impacto relevante en el desempeño del agente

Los resultados relacionados con las fuerzas y pares aplicados durante la ejecución se muestran en las figuras 5.23 y 5.24. En ellas se aprecia que el agente no excede en ningún momento los límites establecidos para dichas magnitudes. Este resultado confirma la efectividad del término R_{safety} en la función de recompensa, que actúa de manera consistente sin depender del número o tipo de trayectorias vistas durante el entrenamiento.

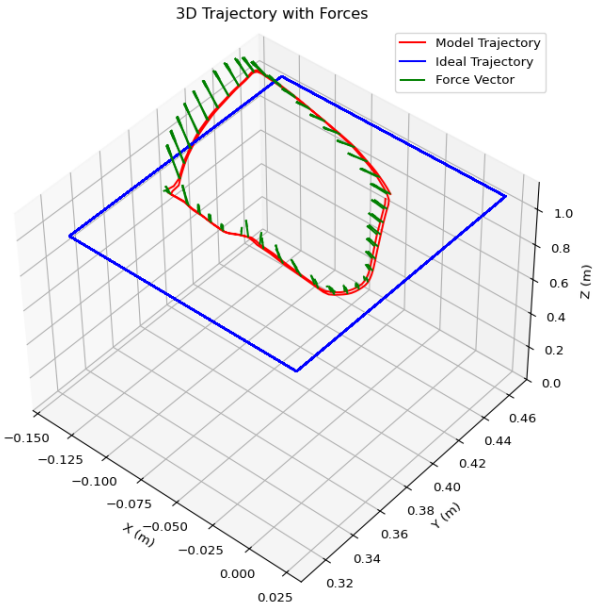


Figura 5.22: Resultados para la trayectoria cuadrada en sentido antihorario para el caso del agente entrenado con una única trayectoria. Comparación de la trayectoria seguida por el agente respecto a la real en el espacio tridimensional, ilustrando el sentido de la fuerza en diferentes pasos de la misma.

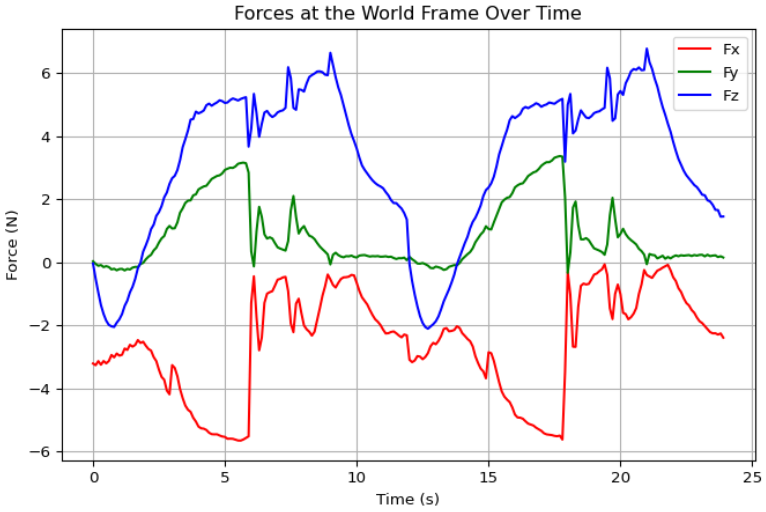


Figura 5.23: Fuerzas del efector final en la ejecución de la trayectoria cuadrada en sentido antihorario, transformadas al sistema global (para el caso del agente entrenado con una única trayectoria).

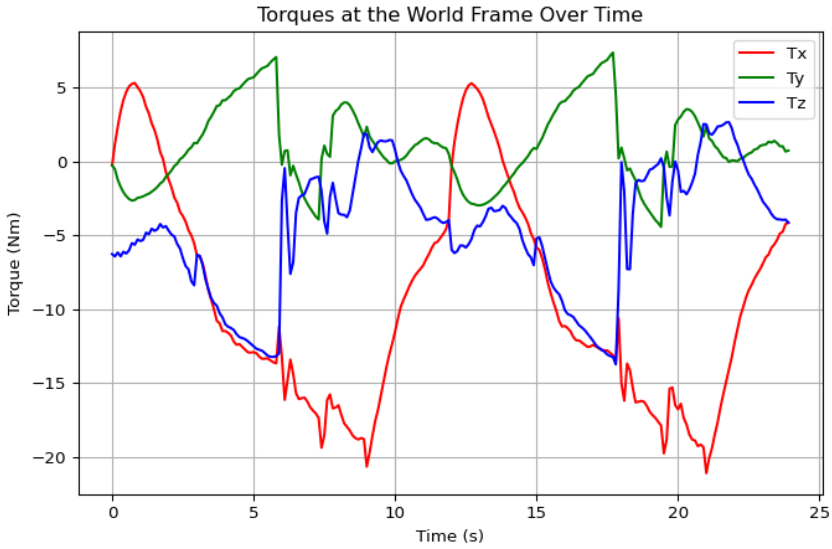
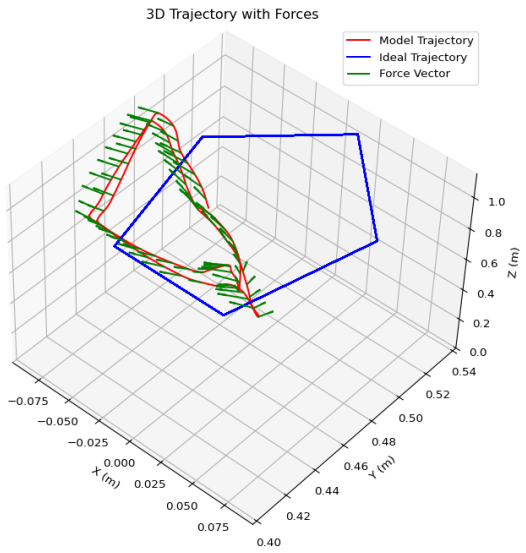
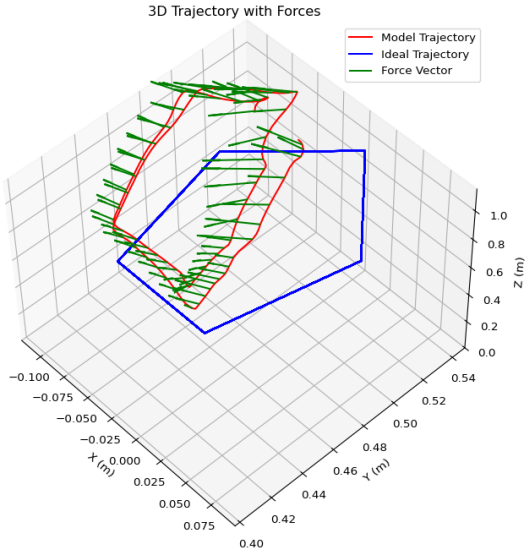


Figura 5.24: Pares del efector final en la ejecución de la trayectoria cuadrada en sentido antihorario, transformados al sistema global (para el caso del agente entrenado con una única trayectoria).

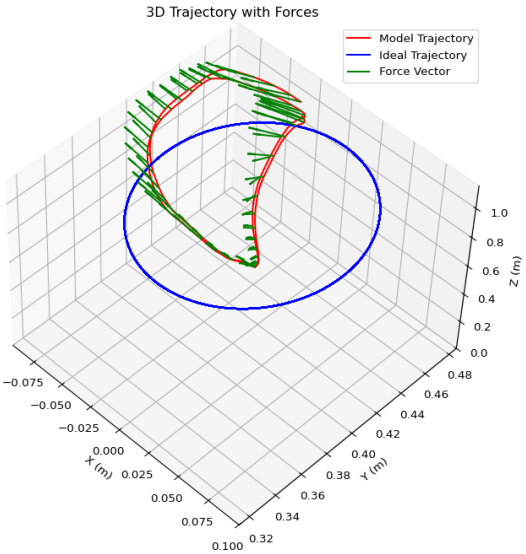
Finalmente, se evaluó el comportamiento del agente frente a trayectorias no experimentadas durante el entrenamiento. Para ello, se seleccionaron cuatro recorridos distintos: (i) trayectoria pentagonal en sentido horario, (ii) trayectoria triangular en sentido horario, (iii) trayectoria circular en sentido antihorario y (iv) trayectoria pentagonal en sentido antihorario. Como se aprecia en la figura 5.25, en todos los casos, el agente mostró dificultades para ajustarse a la trayectoria ideal, evidenciando una falta de capacidad de generalización. En algunos ensayos se observa incluso un comportamiento inestable o errático. Estos resultados refuerzan la importancia de emplear un conjunto diverso de trayectorias durante el proceso de entrenamiento para promover un comportamiento generalizable y robusto del agente.



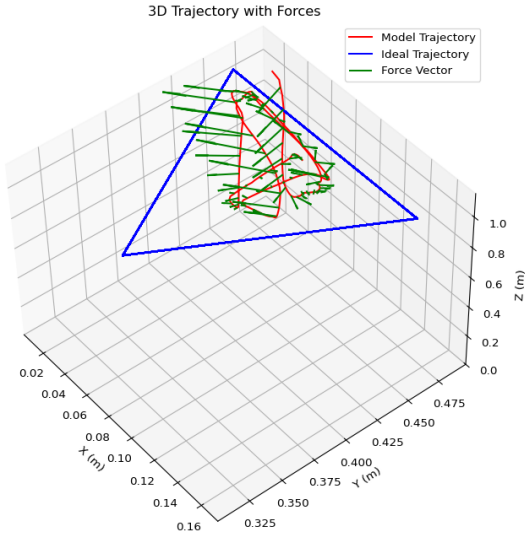
(a) Trayectoria pentagonal sobre el espacio tridimensional en sentido horario.



(b) Trayectoria pentagonal sobre el espacio tridimensional en sentido antihorario.



(c) Trayectoria circular sobre el espacio tridimensional en sentido antihorario.



(d) Trayectoria triangular sobre el espacio tridimensional en sentido horario.

Figura 5.25: Comparación de las trayectorias seguidas por el agente en el espacio tridimensional para distintas formas y sentidos de giro (para el caso del agente entrenado con una única trayectoria).

Capítulo 6

Conclusiones y líneas futuras

Contenido

6.1. Conclusiones

La principal aportación del presente Trabajo Fin de Máster ha sido la generación de un *framework* de simulación que permita la realización de experimentos de interacción humano-robot que involucren un robot manipulador, concretamente el Franka Emika Panda, y un brazo humano. Como se describió en el capítulo 1, uno de los principales retos de la aplicación del RL a la interacción humano-robot consiste en recopilar experiencias interactivas realistas con humanos. Debido a la complejidad y riesgo que involucra realizar entrenamientos en el mundo real, numerosos esfuerzos se concentran en la construcción de entornos de simulación que integren modelos humanos de alta fidelidad [23].

Para materializar el entorno de simulación propuesto, se ha llevado a cabo la integración de diversas librerías clave para la simulación física y el entrenamiento de agentes de aprendizaje por refuerzo profundo. En particular, se ha empleado la librería `dm_robotics_panda` para modelar el manipulador Franka Emika Panda, junto con `myo_sim`, que permite simular de manera realista las físicas de un brazo humano. A esto se suman `Stable Baselines3` para el entrenamiento de políticas de DRL, y `rl_spin_decoupler` como herramienta de apoyo para comunicar los procesos de aprendizaje y simulación.

Todo el entorno ha sido construido bajo una arquitectura modular orientada a objetos, lo que garantiza una estructura de código extensible, reutilizable y fácilmente integrable. Este enfoque no solo facilita el desarrollo de experimentos personalizados, sino que también permite una abstracción clara de los componentes clave involucrados en la interacción física humano-robot.

El resultado es un *framework* de simulación realista y versátil, diseñado específicamente para abordar los desafíos únicos que presenta la interacción física colaborativa entre humanos y robots en contextos de rehabilitación. El entorno permite replicar condiciones experimentales complejas de forma segura y controlada, ofreciendo un banco de pruebas fiable para la validación de estrategias de control basadas en DRL antes de su implementación en sistemas reales.

Además, se ha desarrollado un conjunto de scripts y ejemplos que facilitan el uso del *framework*, junto con un repositorio público documentado y un manual de instalación paso a paso. Este esfuerzo busca fomentar la reproducibilidad y reutilización del entorno por parte de la comunidad investigadora.

Para validar el entrenamiento se realizaron multitud de experimentos, como aquellos sirvieron para definir los umbrales de fuerza y par o el análisis de los datos resultantes de comandar trayectorias mediante máquinas de estados. Todos estos experimentos fueron detallados en el capítulo 4. Aunque la prueba final consistió en entrenar un agente basado en el algoritmo *Soft Actor-Critic* (SAC) capaz de seguir una trayectoria arbitraria desplazando conjuntamente un brazo humano, evitando alcanzar estados que puedan provocar al usuario sensaciones de dolor o resistencia mecánica percibida como forcejeo. Para ello fue necesario elaborar una función de recompensa y definir los espacios de observaciones y acciones.

De esta forma se validó el correcto funcionamiento del sistema y quedó contrastado el potencial que presenta el *framework* como base para la generación de agentes de DRL mediante el entrenamiento en un entorno de simulación con un modelo de brazo humano de alta fidelidad.

En términos generales, se puede confirmar que el agente, ha aprendido la tarea propuesta, ya que se le ha sometido al seguimiento de trayectorias experimentadas y no experimentadas, demostrando ser capaz abordar la tarea sin generar esfuerzos sobre el brazo humano. Pese a ello, los resultados pueden ser mejorables, especialmente en lo referente al seguimiento de la trayectoria de manera fidedigna.

6.2. Líneas futuras

A partir de los resultados alcanzados en este trabajo, se identifican diversas líneas de investigación y desarrollo susceptibles de ser exploradas en el futuro:

- **Mejora de los resultados obtenidos:** Como se ha comentado, pese a que el agente resultante ha sido validado con trayectorias no experimentadas, los resultados son aceptables pero mejorables. Siendo clara la necesidad de contar con mayor tiempo para realizar nuevos entrenamientos que permitiesen afinar el tuneado de hiperparámetros para alcanzar un mejor desempeño, especialmente en la fidelidad de la trayectoria ejecutada respecto a la ideal.
- **Validación en entorno físico real:** Una extensión natural de este trabajo sería trasladar el agente entrenado al entorno físico real, empleando el manipulador Franka Emika Panda ubicado en el laboratorio 2.005 de la Escuela de Ingenierías Industriales de la Universidad de Málaga. Esta prueba permitiría verificar la generalización del modelo validando lo que se denomina como *sim-to-real transfer* [23].
- **Ampliación del conjunto de tareas:** Explorar tareas más complejas, como ejercicios de rehabilitación reales diseñados por un terapeuta, debiendo realizarse un nuevo estudio del espacio de observaciones y función de recompensa. Esto daría mayor fuerza al carácter generalista del *framework*.

El trabajo desarrollado constituye una base sólida sobre la cual se pueden construir investigaciones futuras con aplicaciones potenciales tanto en el ámbito clínico como en la robótica colaborativa.

Apéndice

Manual de uso

Contenido

Este anexo tiene por objetivo describir de forma detallada el proceso de instalación y puesta en marcha del entorno de simulación desarrollado. Dicha información es esencial para cualquier usuario que desee reproducir o extender los experimentos realizados, y se basa en el contenido del repositorio público del proyecto [7].

A.1. Requisitos previos

Antes de comenzar, se asume que el usuario dispone de un sistema operativo basado en Unix (preferentemente Ubuntu 22.04 o superior) con Python 3.10 o superior y acceso a internet para la descarga de dependencias. Asimismo, es recomendable contar con conocimientos básicos sobre entornos virtuales en Python y control mediante terminal.

A.2. Clonación del repositorio

El primer paso consiste en clonar el repositorio del proyecto. Para ello, se puede emplear el siguiente conjunto de comandos:

```
git clone https://github.com/opfernandez/panda_myoarm.git
cd panda_myoarm
python3 -m venv venv
source venv/bin/activate
git submodule init
git submodule update --remote --merge
pip3 install -r requirements.txt
```

En el repositorio se hace uso de submódulos de Git, por lo que es necesario inicializarlos y actualizarlos. Estos submódulos se corresponden con las librerías externas detalladas en el Capítulo 3, como `dm_robotics_panda` o `myo_sim`. Por otro lado, el archivo `requirements.txt` contiene las dependencias necesarias para la ejecución del entorno, incluyendo librerías de simulación, control, y aprendizaje por refuerzo. Para no tener problemas de dependencias durante la instalación se recomienda crear un entorno virtual de Python con el comando `python3 -m venv venv`.

A.3. Entrenamiento del modelo

El entrenamiento del modelo basado en aprendizaje por refuerzo profundo, en este caso basado en SAC, se realiza mediante la ejecución simultánea de dos scripts en terminales independientes. Se recuerda que un proceso se encarga de interactuar con el entorno y recopilar experiencias (`AgentSide`). El otro proceso recibe esas experiencias para realizar el proceso de aprendizaje e inferir las acciones a ejecutar por el agente (`RLSide`). La librería `rl_spin_decoupler` se encarga de desacoplar ambos procesos y comunicarlos empleando sockets.

1. Lanzamiento del algoritmo de aprendizaje (`RLSide`)

Desde el directorio de ejemplos, se ejecuta el siguiente script:

```
cd dm_robotics_panda/examples/
python3 rl_side_panda_myoarm.py -p <puerto>
```

Donde `<puerto>` es el puerto que se usará para la comunicación mediante sockets, debe ser un número entero, por ejemplo 49055. Este proceso ejecuta

el algoritmo SAC de `Stable Baselines 3`, que se comunica vía sockets con el entorno simulado.

2. Lanzamiento del entorno simulado (`AgentSide`)

En una segunda terminal, se debe iniciar el entorno de simulación, que se encargará de generar observaciones a partir de las acciones enviadas por el algoritmo:

```
python3 agent_side_panda_myoarm.py -p <puerto>
```

El valor de `<puerto>` debe coincidir con el empleado en el script anterior.

3. Monitorización del proceso de entrenamiento

Para visualizar en tiempo real el progreso del entrenamiento y las métricas relevantes (recompensas, pérdidas, etc.), se recomienda utilizar `TensorBoard` [22]:

```
tensorboard --logdir ./train_logs --host 0.0.0.0
```

Este comando habilita una interfaz accesible desde el navegador en la dirección `http://localhost:6006/`. A continuación se muestra una captura de la interfaz resultante (ver figura A.1).

Figura A.1: Interfaz accesible desde el navegador para monitorizar el entrenamiento empleando `TensorBoard` [22].

A.4. Evaluación del modelo entrenado

Una vez finalizado el proceso de entrenamiento (ya sea de forma automática o manual), los modelos entrenados se almacenan en la carpeta `/checkpoints` en formato `.zip`.

La evaluación se realiza mediante el siguiente script, que permite cargar el modelo y definir la trayectoria a seguir durante la prueba:

```
python3 test_trained_DRL_model.py -m <ruta_al_modelo.zip> -t <trayectoria>
```

Entre las trayectorias disponibles se encuentran:

- Trayectoria circular en sentido horario y antihorario, respectivamente: `h-circle`, `ah-circle`.
- Trayectoria cuadrada en sentido horario y antihorario, respectivamente: `h-square`, `ah-square`
- Trayectoria triangular en sentido horario y antihorario, respectivamente: `h-triangle`, `ah-triangle`
- Trayectoria pentagonal en sentido horario y antihorario, respectivamente: `h-pentagon`, `ah-pentagon`

Durante este proceso, se almacenan automáticamente en ficheros `.csv` las métricas clave: fuerzas, torques, velocidades del efector final y la propia trayectoria seguida. Estos datos se utilizan a posteriori para inspeccionar la calidad del modelo resultante.

A.5. Visualización de resultados

Para representar gráficamente los datos obtenidos durante las pruebas, se ha implementado un script específico que genera las siguientes gráficas:

- Trayectoria que ha seguido el efector final del manipulador respecto a la programada o ideal. Ambas trayectorias se muestran sobre el plano XY.
- Evolución temporal de las fuerzas medidas en la muñeca del manipulador.
- Evolución temporal de los pares medidos en la muñeca del manipulador.

- Evolución temporal de las fuerzas medidas en el sistema de referencia global (`world`).
- Evolución temporal de los pares medidos en el sistema de referencia global (`world`).
- Trayectoria que ha seguido el efector final del manipulador respecto a la programada o ideal. Ambas trayectorias se muestran en el espacio tridimensional, sobre la trayectoria seguida por el manipulador se grafican los vectores de fuerza transformados al sistema global.

Una vez se han generado los ficheros `.csv` con los datos de la prueba, simplemente se debe ejecutar el siguiente comando:

```
python3 plot_data.py
```

A.6. Conclusión

En este anexo se han descrito de forma estructurada los pasos necesarios para clonar, configurar, ejecutar, entrenar y evaluar modelos dentro del *framework* de simulación desarrollado. Siguiendo estas instrucciones, cualquier usuario podría reproducir los resultados alcanzados en este trabajo o adaptar el entorno a nuevas tareas de una forma cómoda gracias a la abstracción en la programación y los ejemplos proporcionados. Además, para facilitar la tarea, se han elaborado dos códigos de ejemplo `example_rl_side_panda_myoarm.py` y `example_agent_side_panda_myoarm.py` que contienen los métodos principales y explicaciones de cada uno de ellos en forma de comentarios, conformando un punto de partida sólido e intuitivo para construir nuevas tareas. De nuevo se quiere destacar el carácter generalista y abierto del *framework* implementado.

Bibliografía

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013.

[2] Vittorio Caggiano, Huawei Wang, Guillaume Durandau, Massimo Sartori, and Vikash Kumar. Myosuite – a contact-rich simulation suite for musculoskeletal motor control, 2022.

[3] DeepMind Technologies Limited. Mujoco — advanced physics simulation, 2025. Última consulta: 19 de mayo de 2025.

[4] Jean Elsner. Taming the panda with python: A powerful duo for seamless robotics programming and integration. *SoftwareX*, 24:101532, 2023.

[5] Jean Elsner. dm_robotics_panda: Panda model for dm_robotics, 2025. Última consulta: 24 de mayo de 2025.

[6] Jean Elsner. Tutorial: dm_robotics_panda. `https://jeanelsner.github.io/dm_robotics_panda/tutorial.html`, 2025. Última consulta: 24 de mayo de 2025.

[7] Oscar P. Fernández. panda_myoarm: Integración de franka emika panda con myoarm para aprendizaje por refuerzo, 2025. Última consulta: 26 de mayo de 2025.

[8] Franka Robotics GmbH. Franka robotics – empowering innovators, enabling breakthroughs, 2025. Última consulta: 19 de mayo de 2025.

[9] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[12] M. Lapan. *Deep Reinforcement Learning Hands-On*. Expert insight. Packt Publishing Ltd, 2024.

[13] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[14] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[16] Miguel Morales. *Grokking deep reinforcement learning*. Manning, 2020.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[18] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of machine learning research*, 22(268):1–8, 2021.

[19] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[21] Stable-Baselines3 Contributors. Soft actor-critic (sac) — stable-baselines3 documentation, 2025. Última consulta: 19 de mayo de 2025.

[22] Stable-Baselines3 Contributors. Tensorboard integration — stable-baselines3 documentation, 2025. Última consulta: 24 de mayo de 2025.

[23] Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes, 2024.

[24] UNCORe Team. rl_spin_decoupler: Spin decoupler for rl/agent loops, 2025. Última consulta: 20 de mayo de 2025.

[25] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[26] Xusheng Wang, Jiexin Xie, Shijie Guo, Yue Li, Pengfei Sun, and Zhongxue Gan. Deep reinforcement learning-based rehabilitation robot trajectory planning with optimized reward functions. *Advances in Mechanical Engineering*, 13(12):16878140211067011, 2021.

[27] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[28] Wikipedia contributors. Atari 2600 — wikipedia, la enciclopedia libre, 2025. Consultado el 10 de mayo de 2025.

[29] Wikipedia contributors. Backpropagation — wikipedia, la enciclopedia libre, 2025. Consultado el 11 de mayo de 2025.

[30] Wikipedia contributors. Entropy (information theory)— wikipedia, la enciclopedia libre, 2025. Consultado el 13 de mayo de 2025.

[31] Xinyu Zhou, Songhao Piao, Wenzheng Chi, Liguo Chen, and Wei Li. Her-drl:heterogeneous relational deep reinforcement learning for decentralized multi-robot crowd navigation, 2024.