unir
LA UNIVERSIDAD
EN INTERNET

Universidad Internacional de la Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Deep Reinforcement Learning aplicado a un módulo experimental del acelerador de IFMIF-DONES

Trabajo Fin de Máster

Presentado por: Galo Gallardo Romero

Dirigido por: Guillermo Torralba Elipe

Ciudad: Tudela de Navarra

Fecha: 19 Septiembre 2023

Índice de Contenidos

Índice de Ilustraciones

Índice de Tablas

Resumen

Aunque la fusión nuclear tiene el potencial de ser una fuente de energía renovable y limpia, su extracción se enfrenta a desafíos importantes, como el estudio de los efectos de la irradiación de neutrones en los materiales del reactor. Con el objetivo de resolver este problema, el proyecto IFMIF-DONES busca construir una instalación para generar un espectro de neutrones similar al de una reacción de fusión nuclear. Dichos neutrones se producen por la colisión de un haz de deuterones acelerado a altas energías contra un blanco de litio. En este trabajo, se investiga la aplicación de la Inteligencia Artificial al control autónomo del acelerador lineal de IFMIF-DONES. En concreto, se desarrolla y valida la metodología necesaria para el control de la posición y las configuraciones geométricas del haz mediante algoritmos Deep Reinforcement Learning. Los resultados obtenidos en esta fase inicial del proyecto son alentadores y respaldan la efectividad de este enfoque, ya que los agentes entrenados logran alcanzar cualquier configuración geométrica de manera óptima y dentro de las tolerancias predefinidas.

Palabras Clave: Reinforcement Learning, Inteligencia Artificial, aceleradores lineales, fusión nuclear, control autónomo

Abstract

Although nuclear fusion has the potential to be a renewable and clean energy source, its extraction faces significant challenges, such as the study of the effects of neutron irradiation on reactor materials. In order to solve this problem, the IFMIF-DONES project aims to build a facility to generate a neutron spectrum similar to that of a nuclear fusion reaction. These neutrons are produced by the collision of a deuteron beam accelerated to high energies against a lithium target. In this work, the application of Artificial Intelligence to the autonomous control of the IFMIF-DONES linear accelerator is investigated. Specifically, the methodology required for the control of the position and geometrical configurations of the beam by means of Deep Reinforcement Learning algorithms is developed and validated. The results obtained in this initial phase of the project are encouraging and support the effectiveness of this approach, since the trained agents manage to achieve any geometric configuration optimally and within the predefined thresholds.

Keywords: Reinforcement Learning, Artificial Intelligence, linear accelerators, nuclear fusion, autonomous control

Capítulo 1

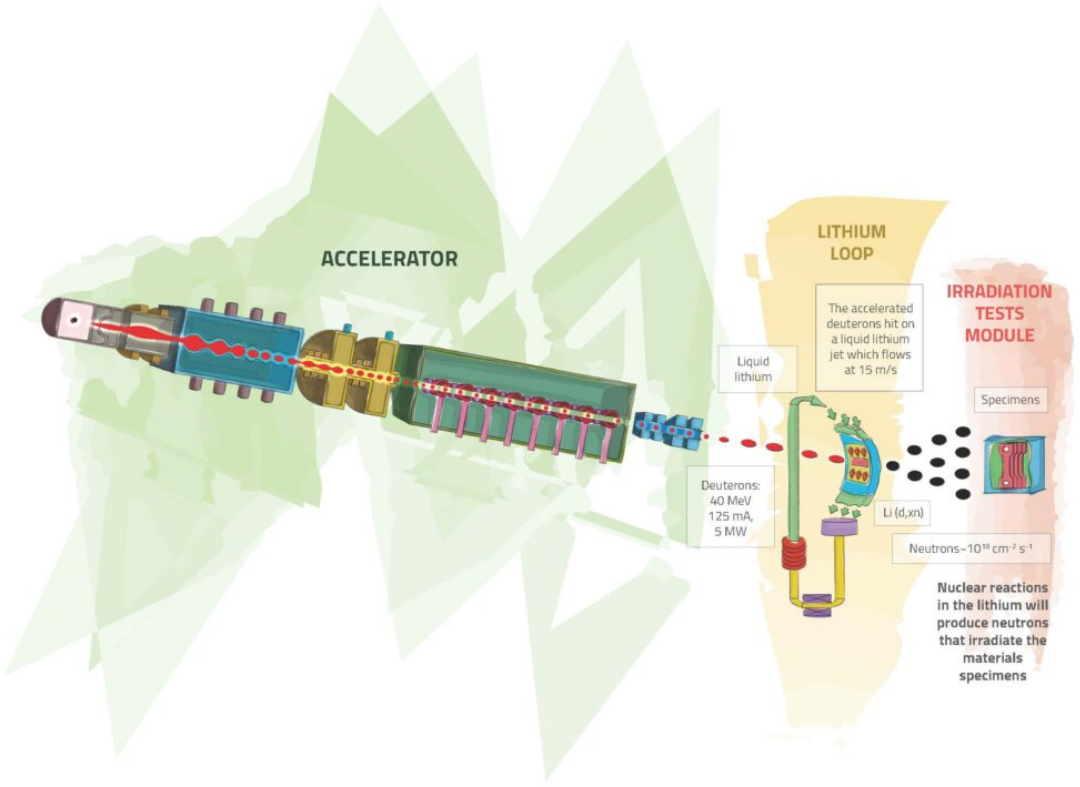
Introducción

La creciente demanda energética a nivel mundial plantea problemas de sostenibilidad con las técnicas de producción de energía actuales (Rafiee and Khalilpour, 2019). Por este motivo, la comunidad científica ha dedicado un esfuerzo importante a la investigación en el campo de la fusión nuclear durante las últimas décadas. Los procesos de fusión nuclear proporcionarán una fuente de energía renovable y limpia. Esto la convierte en una candidata potencial para suavizar los efectos del cambio climático y ayudar al desarrollo sostenible del planeta (Toschi, 1997; Wu and Şahin, 2018). A pesar de su inmenso potencial, la construcción de este tipo de centrales no está exenta de problemas, se enfrenta a numerosos retos importantes (Ciattaglia et al., 2017).

En primer lugar, alcanzar las condiciones de temperatura y presión necesarias para iniciar la reacción de fusión y además mantenerla sigue siendo un gran desafío. En segundo lugar, hay que controlar y confinar el plasma de manera estable, minimizando sus inestabilidades y perturbaciones (Degrave et al., 2022). Estos dos procesos son complejos y requieren de mucha energía, por lo tanto se tiene que alcanzar un cierto nivel de eficiencia para que la instalación posea el balance energético adecuado. Finalmente, hay que gestionar de manera adecuada la radiación de neutrones, los cuales pueden degradar y causar daños estructurales importantes en los materiales del reactor (Rapp, 2017).

Debido a este último problema, uno de los requisitos para la construcción de plantas de fusión es el estudio de los efectos de la irradiación de neutrones en los materiales de la instalación. Se necesita entonces una fuente de neutrones con un espectro similar al de una reacción de fusión nuclear. Esto ha dado lugar al proyecto IFMIF-DONES (The International Fusion Materials Facility-DEMO-Oriented Neutron Source) (Bernardi et al., 2022) en el que se pretende construir una planta para generar dicho flujo de neutrones y así estudiar y evaluar materiales para reactores de fusión nuclear, como ITER y DEMO (Wan et al., 2019; Federici et al., 2019).

Figura 1.1: Estructura del experimento IFMIF-DONES. La planta se divide en tres módulos: acelerador lineal para el flujo de deuterones, circuito del flujo de litio y zona de irradiación de neutrones. Imagen tomada de: `https://ifmif-dones.es/dones-program/dones-facility/`



El objetivo principal del proyecto IFMIF-DONES es generar un espectro de neutrones que cubra el rango de una reacción nuclear Deuterio-Tritio (D-T). Esto se consigue mediante una reacción de stripping (Timofeyuk and Johnson, 2020) generada por colisión de un haz de deuterones (iones de deuterio) con partículas de litio

(D-Li) bajo condiciones muy específicas. Los neutrones producidos en esta reacción son utilizados en una cámara de pruebas para ver sus efectos sobre distintas muestras de materiales. La instalación consta de varias partes (véase Figura 1.1), a saber, un acelerador lineal para el haz de deuterones, una cámara con un circuito de litio donde se produce la reacción y la zona de irradiación de neutrones (Bernardi et al., 2022). El acelerador lineal es el responsable de crear, transportar, acelerar y moldear el haz de deuterones a la geometría deseada para la colisión con el blanco de litio (Podadera et al., 2021a).

Por otra parte, la Inteligencia Artificial (IA) ha experimentado un desarrollo exponencial en la última década. Definida como la ciencia de programar algoritmos que aprenden de los datos, se trata de una tecnología revolucionaria que se ha aplicado ya a diversos retos y se ha implementado en la mayoría de sectores. Son algoritmos capaces de llevar a cabo tareas que normalmente son resueltas con la inteligencia humana, como pueden ser el reconocimiento de imágenes o sonidos, la producción del habla, la toma de decisiones, la planificación o el control de sistemas. La IA tiene el potencial de avanzar en campos como la medicina, la robótica, la industria y la investigación. Uno de los principales paradigmas de la IA es el Reinforcement Learning (RL) (Jokanović, 2022).

Los algoritmos RL son utilizados para resolver problemas de toma de decisiones secuenciales. Un agente interactúa con un entorno (normalmente simulado) que le proporciona feedback en forma de recompensas o castigos dependiendo de sus acciones. El objetivo del agente es aprender una forma de actuar óptima maximizando las recompensas obtenidas. El Deep Reinforcement Learning (DRL) combina algoritmos RL con redes neuronales profundas como núcleos de los agentes, lo que les permite desempeñar tareas más complejas. Hoy en día, estos algoritmos han demostrado ser herramientas versátiles y potentes, capaces de resolver problemas en entornos complejos y dinámicos como los videojuegos, la robótica, los vehículos autónomos, las finanzas o sistemas de control en general (Graesser and Keng, 2019).

Los sistemas que controlan los aceleradores de partículas presentan importantes desafíos, siendo uno de ellos el desarrollo de un sistema de control eficiente. Los

controladores tradicionales funcionan pero requiere un esfuerzo sustancial a nivel de ingeniería. Esto es debido a que la dinámica del haz presenta un comportamiento no lineal y con dependencia temporal por las pérdidas energéticas. Además, la geometría del haz requiere un ajuste cuidadoso de todos los parámetros físicos de los que depende (Nagaitsev et al., 2021; Ibarra et al., 2019; Kaiser et al., 2022; St. John et al., 2021).

Debido a esto, varios autores están considerando la posibilidad de aplicar algoritmos de DRL en el control y diseño de aceleradores de partículas, aunque siguen en fases iniciales. En conjunto, estos trabajos muestran el potencial del RL para automatizar y mejorar la eficiencia de varios experimentos futuros (Pang et al., 2020; Eichler et al., 2021; Shin et al., 2018; Kain et al., 2020). Además, recientemente, investigadores de Deep Mind han conseguido controlar de manera estable el plasma en un tokamak (máquina con forma toroidal que confina el plasma mediante campos magnéticos) aplicando técnicas DRL (Degrave et al., 2022). Por lo tanto, en esta tésis, se plantea el uso de este paradigma para diseñar la arquitectura de un controlador para un módulo experimental del acelerador lineal de IFMIF-DONES.

La estructura de este trabajo se presenta de la siguiente manera: en la sección 2 se describe el marco teórico, que abarca las bases matemáticas de las técnicas DRL. En la sección 3, se lleva a cabo un estudio de los algoritmos de Estado del Arte en este campo, así como un análisis de trabajos recientes relacionados con sistemas de control aplicados a instrumentación física. En la sección 4, se especifica la arquitectura del controlador y se valida mediante una implementación a nivel de software, mostrando todos los resultados obtenidos. En la sección 5, se presentan las conclusiones y se indican los pasos a seguir en las siguientes etapas del proyecto.

Capítulo 2

Marco teórico

El RL es una rama de la IA enfocada en diseñar algoritmos para solucionar problemas en entornos donde existen tomas de decisiones secuenciales. En estos algoritmos, uno o varios agentes aprenden a elegir acciones basándose en señales de recompensa que recibe tras interactuar con el entorno. A través de ensayo y error, los agentes desarrollan la capacidad de tomar decisiones que maximizan una recompensa esperada. Estos algoritmos se inspiran en los estudios experimentales del aprendizaje animal llevados a cabo en el campo de la psicología (Sutton and Barto, 2018), y su estudio empezó con Richard Bellman en la década de 1950 (Bellman, 1958). Sin embargo, no ha sido hasta estos últimos años que se ha podido aplicar de manera eficiente debido al avance en la potencia de cómputo (Graesser and Keng, 2019).

Este paradigma ha logrado grandes avances en aplicaciones como:

- Los videojuegos, donde los agentes son capaces de competir a nivel mundial en juegos como Dota 2 (Berner et al., 2019).
- Los juegos de mesa, donde Alpha Go logró vencer al campeón mundial de Go (Silver et al., 2017).
- La conducción autónoma, donde se está investigando su uso a través de la información recibida de los sensores de los coches, para tomar decisiones adecuadas en tiempo real (Chen et al., 2019).

- La robótica, donde se utiliza para entrenar robots para realizar tareas complejas y adaptarse a entornos cambiantes. Un ejemplo destacado es la resolución del cubo de Rubik por una mano robótica (OpenAI et al., 2019).
- Sistemas de control, donde se ha utilizado para el control y optimización de procesos industriales y científicos, como el control del plasma en un tokamak (Degrave et al., 2022).
- Las finanzas, donde se ha utilizado para la toma de decisiones de inversión y gestión (Wu et al., 2020).

En la sección 2.1, se estudia cómo se plantea un problema en RL, cuáles son sus componentes principales y cómo es el ciclo de aprendizaje. Todas las explicaciones están basadas en los libros de Graesser and Keng (2019) y Sutton and Barto (2018). En la Sección 2.2, se definen las técnicas y conceptos fundamentales necesarios para comprender el funcionamiento de todos los algoritmos RL. También se examina el algoritmo Deep Q-Networks, cuya arquitectura es utilizada en algoritmos recientes. Todos estos conceptos son importantes para entender el funcionamiento de algoritmos más complejos utilizados hoy en día y que se exponen en el Capítulo 3.

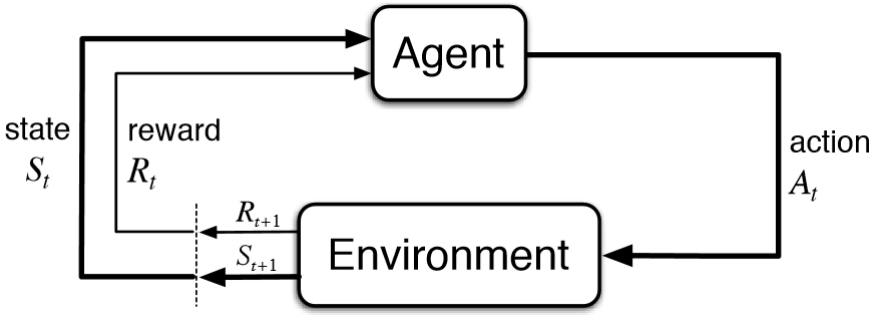
2.1. Definiendo un problema Deep Reinforcement Learning

2.1.1. Formulación del problema como un proceso de Markov

En la resolución de un problema RL, se pueden identificar dos elementos fundamentales: el agente y el entorno. El entorno es el sistema que se desea modelar y resolver, y su información está descrita mediante estados. Por su parte, el agente es la entidad que observa el estado del entorno y toma decisiones en base a la información recopilada. Estas decisiones son acciones que hacen que el entorno avance al siguiente estado y que generan una recompensa para el agente, la cual puede ser positiva o negativa. Cada ciclo de este proceso, que se lleva a cabo en iteraciones de tiempo t denominados pasos temporales (*"time steps"*), se representa en la Figura 2.1. Para

un tiempo t, una iteración de este bucle define una tupla (s_t, a_t, r_t) conocida como *experiencia*, donde s_t es el estado, a_t es la acción y r_t es la recompensa.

Figura 2.1: El loop principal de un algoritmo RL. A tiempo $t = 0$, el agente percibe el estado inicial S_0 del entorno. Basándose en la información proporcionada por ese estado, el agente toma la acción A_0. El entorno avanza al siguiente time step y produce el estado S_1 y la recompensa R_1. Imagen tomada de Sutton and Barto (2018).



De manera resumida, un problema de RL es un sistema en el que un agente interactúa con un entorno siguiendo este bucle. En cada iteración del ciclo, se obtiene una experiencia que es utilizada para el aprendizaje del agente. Por otra parte, un episodio es un conjunto de experiencias secuenciales que ocurren desde $t = 0$ hasta $t = T$. El momento T es el tiempo en el que el agente cumple su objetivo o el entorno alcanza una condición terminal. La secuencia de experiencias durante un episodio se conoce como trayectoria, la cual se puede expresar mediante la siguiente fórmula:

$$\tau = (s_0, a_0, r_0), (s_1, a_1, r_1), ..., (s_T, a_T, r_T) \tag{2.1}$$

Un entorno dado define el espacio de estados, el espacio de acciones y las funciones de recompensa compatibles. El espacio de estados, $\mathcal{S}$, es el conjunto de todos los posibles estados s_t disponibles en el entorno. De manera similar, el espacio de acciones, $\mathcal{A}$, es el conjunto de todas las acciones posibles a_t que el agente puede ejecutar. Ambos espacios dependen del problema a resolver y del investigador, ya que la información de los estados y las acciones pueden ser almacenados como distintos tipos de estructuras de datos según el contexto. Por otro lado, la función de recompensa $\mathcal{R}(s_t, a_t, s_{t+1})$ es una función que devuelve un escalar en la transición del entorno

debido a una acción. Este escalar es la recompensa obtenida por el agente y debe reflejar la diferencia entre los estados s_{t+1} y s_t. Finalmente, la transición de estados es formulada como un proceso de decisión de Markov (MDP) (Bellman, 1957), que describe la evolución del entorno en base a las decisiones tomadas.

Figura 2.2: Algoritmo del bucle de control de un problema RL. Dado un agente y un entorno, en cada episodio y time step, el agente selecciona acciones. El entorno genera una recompensa y el siguiente estado para la acción ejecutada. Con la información obtenida, el agente aprende (línea 8 del pseudo-código, el método update). Imagen tomada de Graesser and Keng (2019).

```
1: Given an env (environment) and an agent:
2: for episode = 0, ..., MAX_EPISODE do
3:     state = env.reset()
4:     agent.reset()
5:     for t = 0, ..., T do
6:         action = agent.act(state)
7:         state, reward = env.step(action)
8:         agent.update(action, state, reward)
9:         if env.done() then
10:            break
11:        end if
12:    end for
13: end for
```

Un MDP es un marco de trabajo que modela matemáticamente un problema de decisiones secuenciales. Una vez tomada una acción a tiempo t, el entorno evoluciona a un nuevo estado s_{t+1} siguiendo una distribución de probabilidad

$$s_{t+1} \sim P\left(s_{t+1} | (s_0, a_0), (s_1, a_1), ..., (s_t, a_t)\right) \tag{2.2}$$

En esta ecuación, se observa que el estado en el tiempo $t+1$ depende de todos los estados y acciones previos del episodio. Esta dependencia hace que la función no sea práctica. Por lo tanto, los MDPs introducen la suposición de que el siguiente estado solo depende del estado actual. Esta simplificación, conocida como la propiedad de Markov, permite que los problemas de RL sean manejables:

$$s_{t+1} \sim P\left(s_{t+1} | s_t, a_t\right) \tag{2.3}$$

A pesar de la simplicidad de esta formulación matemática, los algoritmos RL han sido capaces de resolver problemas altamente complejos. Esto es debido a que los estados de un sistema pueden definirse de tal manera que cumplan la propiedad de Markov y posean la información necesaria para calcular los siguientes elementos de la secuencia. Sin embargo, es importante destacar que existen entornos en los que el estado observado por el agente posee solo una parte de la información de los mismos. Este tipo de problemas, en los que no es posible conocer el estado completo del entorno, se conocen como Problemas de Decisión de Markov Parcialmente Observables (POMDP) (Kaelbling et al., 1998).

El objetivo de un agente, que es resolver un problema formulado en el marco del RL, es equivalente a maximizar las recompensas obtenidas. Esto se define matemáticamente a través de la función $R(\tau)$, que depende de la trayectoria recorrida por el agente en un episodio (o en un conjunto de iteraciones temporales):

$$R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + ... + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t r_t \tag{2.4}$$

Donde $\gamma \in [0, 1]$ es un hiperparámetro conocido como factor de descuento. La Ecuación 2.4 se interpreta como la suma de recompensas descontadas obtenidas en cada instante a lo largo de una trayectoria. El factor γ es muy importante en RL puesto que cambia el valor de las recompensas a lo largo del tiempo, y representa la importancia que el agente le da a las recompensas futuras. Mientras más grande sea este factor, más tiene en cuenta el agente las recompensas a largo plazo. Si este valor es nulo, solo importa la recompensa inicial r_0.

En base a todo esto, se define el objetivo del agente, $J(\tau)$, como el valor esperado de la función $R(\tau)$ calculado sobre múltiples trayectorias para una política π (del inglés *"policy"*) dada:

$$J(\tau) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right] = \mathbb{E}_{\tau\sim\pi}\left[R(\tau)\right] \tag{2.5}$$

Donde π es una función que guía la conducta del agente, al utilizarla para muestrear las trayectorias en el proceso de entrenamiento. Resolver un problema de RL implica encontrar una política óptima de acciones, lo que es equivalente a maximizar la

Ecuación 2.5. Para ello, el bucle descrito en la Figura 2.1 se trata como un problema MPD, que es implementado computacionalmente en el algoritmo descrito en la Figura 2.2.

2.1.2. Funciones de aprendizaje y tipos de algoritmos

En la línea 8 del pseudo-código de la Figura 2.2, aparece el método *update*, que está relacionado con el algoritmo de aprendizaje del agente. Esto sucede después de recopilar experiencias en cada iteración[1], con las cuales el agente actualiza los parámetros de una o varias funciones. El objetivo del agente, como se ha mencionado anteriormente, es seleccionar acciones que maximizan la función de recompensa. Dichas acciones son producidas por una política $\pi(s)$, la cual toma estados como entrada. Esta función es el cerebro del agente, ya que define su comportamiento indicándole qué acción tomar en cada estado. Esta es una de las funciones que el agente puede aprender en RL y forma parte de la mayoría de algoritmos que se ven en el Capítulo 3.

Existen otras dos funciones que el agente puede aprender: las funciones de valor. Éstas proporcionan información acerca del objetivo y evalúan la calidad de los estados y las acciones. La Ecuación 2.6 muestra la función de valor V^{π} que evalúa un estado $s_0 = s$ para todas las posibles acciones, suponiendo que el agente empieza en dicho estado y continúa actuando de acuerdo a π:

$$V^{\pi}(s) = \mathbb{E}_{s_o=s,\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t\right] \tag{2.6}$$

Por otra parte, la función de valor Q^{π} de la a ecuación 2.7 evalúa un estado $s_0 = s$ para una acción determinada $a_0 = a$.

$$Q^{\pi}(s,a) = \mathbb{E}_{s_o=s,a_0=a,\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t\right] \tag{2.7}$$

Existen tres enfoques que clasifican los algoritmos RL basados en estas funciones:

[1]Es importante mencionar que el término *iteración"* depende del contexto y del algoritmo, ya que puede ser un paso temporal, un batch de experiencias o un episodio completo.

- Métodos basados en el aprendizaje de π: los parámetros de la función política se aprenden directamente.
- Métodos basados en el aprendizaje de funciones de valor: en vez de aprender la política directamente se aprende una función de valor que evalúa la calidad de cada estado.
- Métodos combinados: se aprenden dos funciones, la función política y una función de valor.

Además, una distinción importante para el estudio de algoritmos RL que determina completamente el aprendizaje es si son on-policy u off-policy:

- On policy: el agente aprende de los datos generados por la función política con la que elige acciones.
- Off policy: se utiliza una función política diferente para actuar y otra para entrenar (actualizar los valores de los parámetros).

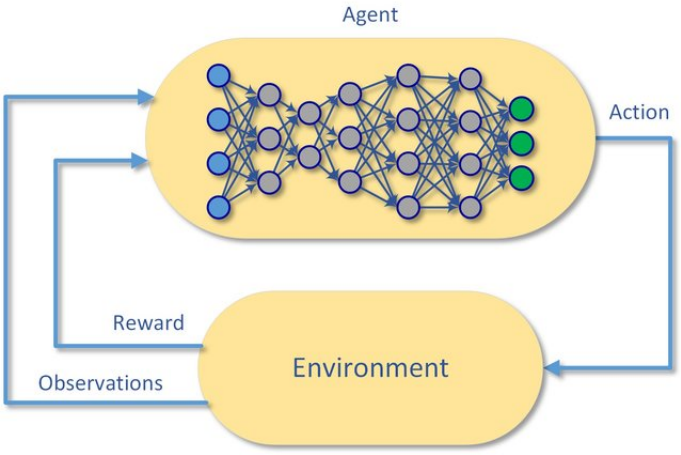
En los primeros, los datos tienen que ser descartados después del entrenamiento ya que se utilizan solo los datos de la política actual. Esto los hace menos eficientes en términos de muestreo. En cambio, en los algoritmos off-policy, cualquier experiencia previa puede ser utilizada para el entrenamiento. Finalmente, hay que mencionar que en este trabajo se exponen y utilizan algoritmos en los cuales el agente no tiene acceso a un modelo del entorno que predice las transiciones de estado según la Ecuación 2.3. Los algoritmos que utilizan un modelo se denominan métodos basados en modelos, aunque no son tan populares debido a su dificultad de implementación. Ver, por ejemplo, el trabajo de Nagabandi et al. (2017).

Existe también otra rama de algoritmos RL que siguen una metodología diferente a los que se mencionan en esta sección, conocida como *offline learning*. En este campo, más parecido al aprendizaje supervisado, los agentes aprenden de un conjunto de datos de experiencias obtenido previamente, sin necesidad de interactuar con el entorno (Levine et al., 2020).

2.1.3. Deep Reinforcement Learning

El Deep Learning (DL) es una disciplina de la IA que busca crear modelos capaces de formar representaciones complejas a partir de conceptos más simples. Los pilares fundamentales de estos modelos son las redes neuronales, las cuales son algoritmos inspirados en el funcionamiento de las neuronas del cerebro humano. Estas redes están compuestas por capas de neuronas (perceptrones) (Rosenblatt, 1958), las cuales procesan la información desde la entrada hasta la salida. Los conceptos complejos son aprendidos de manera jerárquica a medida que la información avanza en profundidad por sus capas (Geron, 2019). La utilidad de estas redes reside en el teorema de aproximación universal, que establece que una red suficientemente compleja puede aproximar cualquier función matemática (Hornik et al., 1989; Yarotsky, 2018). En otras palabras, las redes neuronales pueden modelar cualquier relación entre los datos de entrada y los datos de salida de un problema.

Figura 2.3: Bucle de control de un problema RL donde la función policy es una red neuronal. Las observaciones son utilizadas como valores de entrada de la red, mientras que las acciones son los valores de salida. Imagen tomada de Mohammadi and Al-Fuqaha (2018).



El DRL combina el paradigma del RL con redes neuronales profundas (Graesser and Keng, 2019; Geron, 2019), lo que permite a los agentes aprender cualquiera de las funciones mencionadas en la Sección 2.1.2. Cada función queda representada por el conjunto de parámetros θ que controlan las conexiones de las neuronas en la red. El proceso de aprendizaje implica encontrar los valores óptimos para dichos

parámetros. Esto se lleva a cabo minimizando una función pérdida $L(f(x,\theta),y)$, la cual evalúa la salida predicha por el modelo $f(x,\theta)$, comparándola con sus valores reales y.

Dado un conjunto de datos de entrenamiento (x,y), se calcula primero la salida de la red $f(x,\theta)$ para un entrada x con valores θ inicializados de manera aleatoria (ver, por ejemplo, el artículo de Boulila et al. (2021) para un resumen de métodos de inicialización de parámetros). A continuación se evalúa la función pérdida con los valores reales y los valores obtenidos. En el siguiente paso se calcula el gradiente de esta función , $\nabla_\theta L$, utilizando técnicas del cálculo diferencial. Finalmente, se actualizan los parámetros θ de la red mediante alguna técnica de optimización como *stochastic gradient descent* (SGD) (Amari, 1993):

$$\theta \leftarrow \theta - \alpha \nabla_\theta L \tag{2.8}$$

donde α es un hiperparámetro que controla la tasa de aprendizaje. Este proceso de entrenamiento se repite de manera iterativa hasta alcanzar el mínimo global. En DRL, el conjunto de valores (x,y) son obtenidos por el agente mediante interacciones con el entorno. Debido a esto, los datos son generados de manera secuencial en cada paso temporal.

2.2. Conceptos fundamentales en Deep Reinforcement Learning

2.2.1. Aprendizaje de funciones de valor

En algoritmos que aprenden solamente funciones de valor, la política óptima que mapea estados a acciones óptimas, denominada π^*, no se obtiene entrenando directamente π. En su lugar, se emplean las funciones de valor para evaluar acciones. Como se ha mencionado antes en la Sección 2.1.2, la función $Q^\pi(s,a)$ evalúa la calidad de un par (s,a) para una política determinada. De manera similar, la función $V^\pi(s)$ mide el valor de un estado s para cualquier acción, según una política. Esta función es el valor esperado o promedio de $Q^\pi(s,a)$ para todas las acciones posibles.

Una de las razones para elegir $Q^{\pi}(s,a)$ sobre $V^{\pi}(s)$ es que con la primera función es trivial diseñar la selección de acciones del agente (Graesser and Keng, 2019; Sutton and Barto, 2018). Por ejemplo, se puede calcular $Q^{\pi}(s,a)$ para todas las acciones en un estado y tomar el valor máximo, $Q^{*}(s,a)$, el cual produce una política óptima (Sutton and Barto, 2018):

$$\pi^{*} = \arg\max_{a} Q^{*}(s,a) \tag{2.9}$$

Por otra parte, si las transiciones de estado en el entorno son estocásticas (una acción puede dar lugar a distintos estados), con la función $V^{\pi}(s)$ se pueden necesitar más muestras para estimar dicha función de manera adecuada.

Por estos motivos, se utiliza $Q(s,a)$ en algoritmos basados en funciones de valor. Esto se conoce como Q-Learning (Clifton and Laber, 2020). Se tienen tablas de valores de $Q^{\pi}(s,a)$ para todos los pares (s,a) y a medida que se explora el entorno se van actualizando dichos valores. Cabe mencionar, a modo de desventaja, que el aprendizaje de la función $Q^{\pi}(s,a)$ es más costoso a nivel computacional ya que requiere muchos más datos para cubrir el espacio de todos los pares (s,a) en vez del espacio de estados s.

2.2.2. Aprendizaje por diferencia temporal

El aprendizaje por diferencia temporal (del inglés "*temporal difference*", TD) es la técnica mediante la que se aproximan los valores $Q^{\pi}(s,a)$ y es también uno de los conceptos fundamentales en todos los algoritmos DRL (Sutton, 1988). En esencia, el aprendizaje TD se basa en la idea de actualizar el valor estimado de la función de valor con dos componentes: la recompensa recibida y otra estimación para la función de valor en el siguiente estado. En este sentido, la técnica TD es bootstrapping porque utiliza estimaciones para estimar otros valores (Efron and Tibshirani, 1993). El agente aprende entonces la función de valor actualizando su estimación mediante la diferencia entre el valor estimado del estado actual y el valor estimado del siguiente estado. Esta diferencia se denomina error TD:

$$TD_{error} = r + \gamma \max_{a} Q(s',a') - Q(s,a) \tag{2.10}$$

donde las variables (s', a') denotan el estado y la acción a tiempo $t+1$ y se ha omitido π de la notación (de ahora en adelante se supone que las funciones de valor están definidas para una política en concreto). La actualización de la función se lleva a cabo mediante:

$$Q(s,a) \leftarrow Q(s,a) + \alpha TD_{error} \tag{2.11}$$

donde α es la tasa de aprendizaje que determina cuánto cambia el valor de $Q(s,a)$ en cada iteración. Por otra parte, la ecuacion de Bellman (Bellman, 2010) está relacionada con las dos ecuaciones anteriores y es la ecuación que permite calcular de manera iterativa, mediante programación dinámica, los valores necesarios para el error:

$$Q(s,a) \approx r + \gamma \max_{a} Q(s',a') = Q_{target}(s,a) \tag{2.12}$$

$$TD_{error} = Q_{target}(s,a) - Q(s,a) \tag{2.13}$$

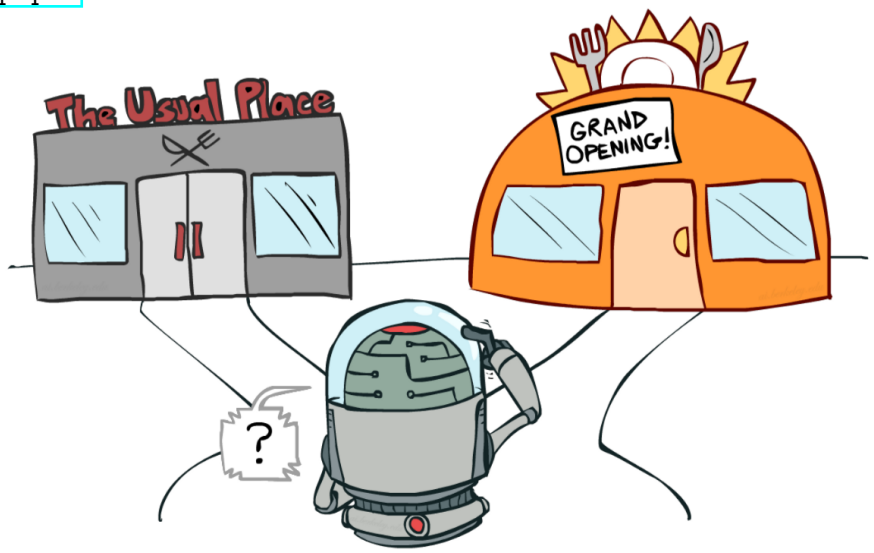
Para poder implementar estas ecuaciones es necesario utilizar una red neuronal para aproximar la función de valor $Q(s,a)$. Esta red, parametrizada por un conjunto de valores θ, se emplea en la estimación de $Q(s,a)$ y de $Q(s',a')$ en la Ecuación 2.10. Una vez obtenido el error, se utiliza como función de pérdida para minimizar las distancias entre $Q_{target}(s,a)$ (calculado con la recompensa y la estimación de la red a tiempo $t+1$) y $Q(s,a)$ (estimado por la red a tiempo t). Los parámetros de la red se actualizan mediante la Ecuación 2.8.

El aprendizaje TD funciona porque emplea la recompensa recibida en el siguiente estado tras ejecutar la acción. Por lo tanto, señales de recompensa un paso en el futuro son enviadas al presente en cada actualización. Uno de los primeros algoritmos en emplear esta técnica fue SARSA (Rummery and Niranjan, 1994), aunque ha sido implementada en todos los algoritmos posteriores en los que se calculan funciones de valor. También hay que mencionar que TD espera solamente un paso temporal para llevar a cabo las actualizaciones, pero existe también $TD(n)$, que hace uso de n pasos futuros. Finalmente, se ha demostrado que esta técnica de aprendizaje converge a la función óptima solo para el caso de aproximadores lineales de funciones, aunque en la práctica se alcanzan buenos resultados para funciones no lineales (Tsitsiklis and Van Roy, 1997; Tesauro, 1995; Sutton, 1988).

2.2.3. El balance entre exploración y explotación

Hasta ahora se ha explicado el aprendizaje de funciones de valor y la técnica TD, que son la base de casi todos los algoritmos en DRL, pero no se ha mencionado la toma de acciones. En estos algoritmos no se aprende la función política directamente, se entrena el agente actualizando la función de valor pero las acciones se ejecutan con una política diferente. En esta sección se explica cómo se puede transformar la función de valor $Q(s, a)$ a una política de acciones. La respuesta más sencilla es que, dada dicha función, el agente puede seleccionar siempre la acción que corresponde al máximo de cada estado según la Ecuación 2.9. Se dice entonces que el agente actúa de manera codiciosa (*greedy*).

Figura 2.4: El problema de la exploración y la explotación. Los conceptos de explotación y exploración se refieren a la toma de decisiones ante opciones de calidad conocida frente a otras de calidad desconocida. Explotar implica elegir repetidamente una opción conocida y buena, aunque pueda haber alternativas mejores. Por el contrario, la exploración implica arriesgarse a probar novedades con potencial, a pesar de la posibilidad de un resultado negativo. Imagen tomada del curso de IA de Berkeley: `https://inst.eecs.berkeley.edu/~cs188/sp20/assets/lecture/lec15_6up.pdf`



No obstante, si las acciones del agente son siempre codiciosas y solo busca explotar la información ya conocida de la función $Q(s, a)$, el espacio (s, a) no se explorará de

manera adecuada. Pueden existir muchos pares estado-acción con resultados mejores que el agente no probará. Hace falta definir una política de acciones que explore y muestree todo el espacio (s, a) de tal manera que el agente no se quede atascado en un óptimo local. La Figura 2.4 muestra una analogía del balance entre exploración y explotación.

Una de las soluciones más utilizadas para abordar este problema es la política ϵ-greedy (Rawson and Balan, 2021):

- Explotación: el agente selecciona la acción óptima (Ecuación 2.9) con probabilidad $1 - \epsilon$, donde ϵ es un hiperparámetro.
- Exploración: El agente toma una acción aleatoria con probabilidad ϵ.

Explorar acciones aleatorias puede reducir la velocidad del algoritmo al principio, pero abre la posibilidad de descubrir acciones que llevan a un óptimo global. Para que el agente aprenda de manera adecuada, el hiperparámetro ϵ no puede ser una constante. A medida que el agente va descubriendo el entorno, su valor tiene que decaer para favorecer la exploración al principio y la explotación al final. Se suele empezar con $\epsilon = 1$ y terminar con valores muy pequeños $\approx$0.001.

2.2.4. Algoritmo Deep-Q Networks

El algoritmo Deep-Q Networks (DQN) se basa en aprender la función de valor $Q(s, a)$ mediante la técnica TD. A diferencia de SARSA, este algoritmo aprende la función óptima Q^* vista en la Ecuación 2.9 (Mnih et al., 2013). En otras palabras, utiliza el máximo sobre todas las posibles acciones a' del estado s'. Una consecuencia importante de esto es que DQN es off-policy, la función que se aprende no depende de la política usada para actuar en el entorno y generar experiencias. Esto se aprecia en la Ecuación 2.9, donde la parte derecha solo depende r y s', que están relacionadas con el entorno y no con el agente o su política. Además, se puede demostrar que la función óptima Q^* está relacionada con la función política óptima, π^* (Sutton and Barto, 2018)

El empleo del algoritmo DQN presenta beneficios significativos en estabilidad y

velocidad de convergencia en comparación con algoritmos anteriores que aprenden la función $Q(s, a)$. Dichos algoritmos, tabulares (no utilizan redes neuronales) en su mayoría, tenían problemas para lidiar con espacios de altas dimensiones, que son comunes en los problemas de control actuales. El uso de redes neuronales en DQN (de ahí la palabra *deep*) aumenta su capacidad para resolver entornos complejos y de alta dimensión, mejora la generalización y se beneficia del teorema de aproximación universal de funciones.

Otra característica fundamental que implementa DQN es el *experience replay* (reproductor de experiencias) (Lin, 1992). Esta técnica, utilizada por algoritmos off-policy, consiste en almacenar experiencias pasadas del agente en una memoria para su uso posterior. Se almacenan las últimas k experiencias y se van descartando las más antiguas conforme la memoria se llena. Esto permite una mayor eficiencia a la hora de entrenar agentes, mejorando así el proceso de aprendizaje. Cada lote de experiencias utilizado puede contener muestras de distintos episodios y distintas políticas, ayudando así a romper las correlaciones entre experiencias y reducir la varianza de los parámetros que se actualizan. Finalmente, para la selección de acciones y el balance de exploración-explotación, DQN puede utilizar la política ϵ-greedy o una política de Boltzmann (Cesa-Bianchi et al., 2017). En este último caso, las acciones se muestrean de una distribución de probabilidad de Boltzmann donde el parámetro temperatura τ juega un rol análogo a ϵ. A mayor temperatura, la distribución tiende a ser más uniforme.

2.2.5. Policy gradient

La mayoría de los algoritmos que solo utilizan funciones de valor (como DQN) son algoritmos off-policy. No se aprende directamente la política y se utiliza otra política para actuar (por ejemplo, ϵ-greey). El objetivo de un algoritmo on-policy es aprender directamente la función policy $\pi_\theta(s)$ que se utiliza para acumular experiencia. Esta función tiene que ser óptima, es decir, tiene que maximizar la recompensa acumulada esperada a lo largo del tiempo dada por la Ecuación 2.5. En la práctica, las políticas se representan por redes neuronales $\pi_\theta(s)$, donde encontrar una política óptima es

equivalente a hallar los mejores parámetros θ.

La recompensa esperada de una trayectoria viene da por la Ecuación 2.4. El objetivo del agente es maximizar dicha recompensa sobre todas las posibles trayectorias según la Ecuación 2.5. El valor esperado de esta ecuación se calcula ahora sobre trayectorias $\tau \sim \pi_\theta$, por lo tanto, el problema consiste en maximizar $J(\pi_\theta)$:

$$\max_\theta J(\pi_\theta) = \max_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \tag{2.14}$$

donde se necesita aplicar SGD para encontrar el máximo de dicha función, de acuerdo con la Ecuación 2.8,

$$\theta \leftarrow \theta + \alpha\nabla_\theta J(\pi_\theta) \tag{2.15}$$

Para entrenar la red, es necesario conocer un método que permita calcular el término con gradiente de esta ecuación. Afortunadamente, el teorema *"policy gradient"* transforma este término en algo computable:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} R_t(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{2.16}$$

donde ahora hay que calcular el gradiente del logaritmo de $\pi_\theta(a_t|s_t)$, que es la probabilidad de que el agente tome la acción a_t dado el estado s_t a tiempo t. Ver, por ejemplo, el libro de Graesser and Keng (2019) para una demostración del teorema.

La Ecuación 2.16 es la base de todos los algoritmos policy gradient (PG) que serán analizados en el Capítulo 3. REINFORCE (Williams, 1992) fue el primer algoritmo que implementó esta técnica, ayudándose de técnicas Monte Carlo (MC) para estimar el gradiente. Los métodos MC utilizan muestreo aleatorio para generar datos con el objetivo de aproximar cualquier función (Kroese et al., 2014). Es importante destacar que los algoritmos que utilizan muestreo MC necesitan generar trayectorias de episodios completos para actualizar los parámetros de la red puesto que la Ecuación 2.16 se estima sobre trayectorias muestreadas.

Capítulo 3

Estado del Arte

3.1. Algoritmos Deep Reinforcement Learning

3.1.1. Mejoras para DQN

Target Networks

En el algoritmo DQN original, se inicializa primero la red que aproxima la función $Q(s, a)$. Después, con la política ϵ-greedy, se recogen y almacenan experiencias (s, a, r, s'). Dichas experiencias son utilizadas para estimar el error de diferencia temporal mediante la ecuación de Bellman (Ec. 2.12). Este error se utiliza como función de pérdida para actualizar los parámetros θ de la red neuronal. Sin embargo, esta técnica tiene el inconveniente de que el valor Q_{tar} se modifica en cada actualización de θ. Este valor cambiante puede desestabilizar el entrenamiento porque hace que esté mucho menos claro establecer a qué valores debe intentar acercarse la red.

Para abordar esta cuestión, se ha propuesto la utilización del método denominado "Target Networks"(Mnih et al., 2015). Este enfoque consiste en introducir una segunda red con el objetivo de reducir los cambios en los valores de Q_{tar} entre las diferentes iteraciones, lo que impide que éstos se desplacen durante un número determinado de pasos. La red *target* es una copia de la red original con con sus propios

parámetros (al igual que θ, son pesos y sesgos), denotados por ϕ:

$$Q^{\pi_\phi}(s,a) \leftarrow Q^{\pi_\theta}(s,a) \tag{3.1}$$

Es mediante esta segunda red que se estiman los valores del error de diferencia temporal. La red original se actualiza constantemente, mientras que la red *target* se actualiza solo después de un número determinado de iteraciones. La frecuencia de actualización es un hiperparámetro que depende del problema en cuestión. Si se establece muy alta, el proceso de entrenamiento puede volverse inestable, mientras que si se establece muy baja, la convergencia se ralentiza. Por último, es importante destacar que existen otros métodos de actualización para ϕ, como el método de actualización de Polyak, que calcula un promedio entre θ y ϕ (Piché et al., 2022).

Double DQN

Otro de los desafíos asociados al algoritmo DQN base, es la sobrestimación de los valores de Q_{tar}. En la parte derecha de la Ecuación 2.12, se requiere calcular el máximo de los valores $Q^{\pi_\theta}(s',a')$ sobre el espacio de acciones. Ahora bien, se ha demostrado que el máximo valor de un conjunto de estimaciones con algo de ruido tiene un sesgo positivo (Graesser and Keng, 2019). Dado que las funciones utilizadas en DRL son aproximaciones mediante redes neuronales, siempre existirán errores en las estimaciones, lo que resulta en una sobreestimación de los valores calculados por la ecuación de Bellman. Si el sesgo fuera constante para todos los valores de la función, no habría problemas significativos. Sin embargo, debido a la política de acciones utilizada comúnmente (ϵ-greedy), el espacio (s,a) no se explora de manera uniforme y unos valores se sobrestiman más que otros. Esto puede conducir a políticas que no son óptimas y afectar negativamente al rendimiento del agente.

Para hacer frente a este problema, se introdujo el algoritmo Double DQN (DDQN) (van Hasselt et al., 2015), que separa la selección de acciones de la estimación de valores al utilizar dos redes neuronales independientes. El algoritmo aprende entonces dos funciones $Q(s,a)$, mediante dos redes diferentes con parámetros θ y φ. La ecuación utilizada es la siguiente:

$$Q^{\pi}_{\text{tar:DDQN}}(s,a) = r + \gamma Q^{\pi_\varphi}\left(s', \max_{a'} Q^{\pi_\theta}(s',a')\right) \tag{3.2}$$

En esta ecuación, la primera red, $Q^{\pi_\theta}(s', a')$, se utiliza para calcular la acción a' que maximiza Q^{π_θ} en el estado s'. A continuación, esta acción es empleada por la segunda red, $Q^{\pi_\varphi}(s', a')$, para estimar el valor de Q_{tar}. En la práctica, se suelen utilizar las redes *target* como segunda red en DDQN y aplicar ambas técnicas para mejorar la eficiencia de entrenamiento y la política obtenida.

Prioritized Experience Replay

Finalmente, otra técnica que ha contribuido a mejorar el algoritmo ha sido la repetición de experiencias prioritarias (PER, Prioritized Experience Replay) (Schaul et al., 2016) . En la Sección 2.2.4, se explicó cómo DQN puede hacer uso de un reproductor de experiencias debido a ser un algoritmo off-policy. En el enfoque PER, en lugar de muestrear las experiencias de manera aleatoria, se lleva a cabo un proceso basado en una distribución de probabilidad construida en función de la calidad de las experiencias. Se considera que una experiencia es mejor si es más informativa y transmite más información al agente durante el proceso de aprendizaje.

Las experiencias que presentan un error TD notable implican una gran discrepancia entre $Q^{\pi}_{tar}(s, a)$ y $Q^{\pi_\theta}(s, a)$. Estas son precisamente las experiencias que el agente utiliza para aprender de manera más efectiva (hay un cambio más grande en los parámetros de la red). Por lo tanto, se considera el valor del error TD (definido en la Ecuación 2.10) como medida de la prioridad para calcular la relevancia de las experiencias. Una de las ventajas de utilizar esta cantidad es que que es muy sencillo calcular en el algoritmo DQN.

Uno de los enfoques para asignar probabilidades a cada experiencia a partir de estos valores es utilizar la siguiente ecuación:

$$P(i) = \frac{(w_i + \epsilon)^\eta}{\sum_j (w_j + \epsilon)^\eta} \tag{3.3}$$

donde w_i denota el error TD de experiencia i, ϵ es una constante positiva y pequeña que garantiza que todas las experiencias sean muestreadas, y η es un hiperparámetro que controla la distribución de probabilidades.

Es importante destacar que esta técnica introduce un sesgo en el proceso de entre-

namiento al modificar la distribución del muestreo de experiencias. En el algoritmo original, las experiencias se seleccionan de manera uniforme, lo que garantiza que todas las experiencias tengan la misma probabilidad de ser elegidas para el entrenamiento. Sin embargo, con PER, las experiencias priorizadas son muestreadas con mayor frecuencia, lo que puede resultar en políticas poco óptimas. Para solucionar este problema se multiplica los errores TD por un conjunto de pesos. Esto se conoce como *importance sampling*.

3.1.2. Advantage Actor-Critic

Arquitectura Actor-Crítico

La base de los algoritmos PG es el teorema *Policy Gradient*, el cual se describe en la Ecuación 2.16. Este teorema hace posible el cálculo del gradiente de la función objetivo $J(\pi_\theta)$, que debe maximizarse mediante ascenso de gradiente para encontrar los parámetros θ que generan la función política óptima. Sin embargo, el algoritmo base REINFORCE que utiliza esta técnica presenta el problema de tener una varianza elevada, debido a muestrear trayectorias estocásticas completas que pueden dar lugar a distintos resultados. Una posible solución a este problema es aumentar el número de trayectorias utilizadas para la estimación del gradiente, pero esto reduce la eficiencia computacional del proceso de aprendizaje (Barto et al., 1983).

El algoritmo Advantage Actor-Critic (A2C) es uno de los algoritmos que se benefician del teorema PG, a la vez que busca solucionar el problema de las varianzas elevadas. Para ello, en lugar de aprender directamente la función política, ésta es reforzada por una señal que proviene de una función de valor. La arquitectura A2C consta de dos componentes principales: el actor y el crítico. El actor es responsable de aprender una política parametrizada, mientras que el crítico se encarga de aprender una función de valor que actúa como una señal de refuerzo para el actor. La idea principal detrás de esta arquitectura es utilizar al crítico para guiar al actor y mejorar la eficiencia del aprendizaje. El uso de señales de refuerzo no solo reduce la varianza de los resultados, sino que además estas señales pueden ser más informativas para la política porque pueden disminuir la dispersión de las recompensas. Sin

embargo, el proceso de entrenamiento se vuelve más complejo al tener que aprender dos funciones.

Función ventaja

La función de ventaja elegida por estos algoritmos ha de tener en cuenta la calidad de una acción relativa al resto de acciones en el mismo estado, por lo tanto se utiliza comunmente:

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t) \tag{3.4}$$

Esta ecuación cuantifica, para cada estado s_t, qué tan buena es una acción en comparación con el promedio, y es utilizada por el crítico para evaluar pares (s, a). Una vez obtenida, es utilizada por el actor para calcular el gradiente del objetivo de la siguiente manera:

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_t\left[A_t^{\pi} \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t)\right] \tag{3.5}$$

Donde se ha sustituido la función objetivo por la función ventaja en la Ecuación 2.16.

Para calcular la función de ventaja definida en la Ecuación 3.4, es necesario estimar previamente las funciones de la parte derecha, $Q^{\pi}(s, a)$ y $V^{\pi}(s)$. En la práctica, se aprende solamente la función $V(s)$ y se estima la función $Q(s, a)$. Existen dos métodos para estimar esta segunda función a partir de la primera: n-Step Returns (NSTEP) (Sutton and Barto, 2018) y Generalized Advantage Estimation (GAE) (Schulman et al., 2018), representadas en las siguientes ecuaciones:

$$A^{\pi}_{\text{NSTEP}}(s_t, a_t) \approx r_t + \gamma r_{t+1} + \cdots + \gamma^n r_{t+n} + \gamma^{n+1} \hat{V}^{\pi}(s_{t+n+1}) - \hat{V}^{\pi}(s_t) \tag{3.6}$$

$$A^{\pi}_{\text{GAE}}(s_t, a_t) \approx \sum_{\ell=0}^{\infty} (\gamma\lambda)^{\ell} \delta_{t+l}, \text{ donde } \delta_t = r_t + \gamma \hat{V}^{\pi}(s_{t+1}) - \hat{V}^{\pi}(s_t) \tag{3.7}$$

La primera técnica reescribe la función Q como una suma de las recompensas esperadas para n pasos temporales y el valor estimado de $V^{\pi}(S_{n+1})$, donde n es un hiperparámetro. La segunda técnica es una extensión de la primera, donde se lleva a cabo una media ponderada de las funciones de ventaja para todos los n pasos temporales. Al igual que n, el hiperparámetro λ controla el balance entre sesgo y varianza del estimador.

El valor estimado de la función $V(s)$, necesario para calcular la ventaja, se obtiene mediante aprendizaje por diferencia temporal, explicado en la Sección 2.2.2.

Algoritmo A3C

El algoritmo Asynchronous Advantage Actor Critic (A3C) (Mnih et al., 2016) utiliza una arquitectura actor-crítico, como se ha discutido en esta sección, junto con una variante del descenso de gradiente conocida como descenso de gradiente estocástico asíncrono (ASGD, asynchronous stochastic gradient descent) para actualizar los parámetros de la política y la función de valor. Este enfoque permite paralelizar el aprendizaje con el fin de acelerar el entrenamiento. Además, al igual que A2C, A3C es un algoritmo on-policy, lo que significa que el gradiente se calcula utilizando trayectorias generadas por la misma política.

3.1.3. Proximal Policy Optimization

Colapso de rendimiento

Los algoritmos PG presentan principalmente dos problemas: ineficiencia en el muestreo y la posibilidad de experimentar un fenómeno conocido como colapso de rendimiento. El primer problema es común a todos estos algoritmos, ya que son *on-policy* y no pueden utilizar experiencias almacenadas de políticas anteriores. En cuanto al segundo problema, se observa que un agente puede comenzar a generar trayectorias subóptimas, lo que lleva a una disminución en las recompensas promedio durante el entrenamiento. Este colapso se debe a que las ecuaciones utilizadas en estos algoritmos buscan encontrar la política óptima actualizando los valores de los parámetros de la red, pero las distancias en este espacio de parámetros $\Theta = \{\theta_k\}$ no necesariamente son congruentes[1] con las distancias en el espacio de políticas $\Pi = \{\pi_i\}$ (Schulman et al., 2017).

El problema es que no se sabe el cambio $\Delta\pi$ en el espacio de políticas cuando se

[1] La distancia entre dos puntos en el espacio de parámetros Θ no es la misma que la distancia entre sus políticas en el espacio Π.

lleva a cabo una actualización de los parámetros, dada por la ecuación:

$$\Delta\theta = \alpha \nabla_\theta J(\pi_\theta) \tag{3.8}$$

Aquí, α representa un hiperparámetro que determina la tasa de aprendizaje. Si su valor es demasiado alto, una sola iteración puede alejar a la política de su óptimo, lo que lleva al agente a generar trayectorias cada vez peores. Esto implica que encontrar tasas de aprendizaje óptimas sin provocar un colapso en el rendimiento se convierte en un desafío.

Objetivo subrogado

Para abordar el problema del colapso de rendimiento, es necesario garantizar una mejora monótona al actualizar los parámetros de la política. En primer lugar, se define un objetivo subrogado que codifica la información sobre cómo una nueva política π' mejora con respecto de la anterior:

$$J_\pi^{\text{CPI}}(\pi') \approx \mathbb{E}_{\tau\sim\pi}\left[\sum_{t\geq 0} A^\pi(s_t, a_t) r_t(\theta)\right] = J(\pi') - J(\pi) \tag{3.9}$$

En esta ecuación, el superíndice CPI significa *conservative policy iteration* (Vieillard et al., 2020) y $r_t(\theta) = \frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)}$ es el *importance sampling ratio*, necesario para que se cumpla la aproximación. Este cociente mide cuánto ha cambiado la política actual en comparación con la anterior. Los detalles de derivación de esta ecuación se han omitido por su longitud y complejidad [2].

En segundo lugar, esta aproximación debe satisfacer $J(\pi') - J(\pi) \geq 0$ y estar limitada en tamaño, por lo que hay que modificar la función objetivo de la Ecuación 3.9. La manera más sencilla de conseguirlo es mediante el uso de la técnica de *clipping*:

$$J^{CLIP}(\theta) = \mathbb{E}_t\Big[\text{mín}\left(r_t(\theta) A_t^{\pi_{\theta_{\text{old}}}}, \text{clip}\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right) A_t^{\pi_{\theta_{\text{old}}}}\right)\Big] \tag{3.10}$$

En esta expresión, se utiliza la política anterior (parametrizada por θ_{old}) para calcular la ventaja. El hiperparámetro ϵ define la región de *clipping*. Por otra parte, se restringe el cociente $r_t(\theta)$ al rango $[1-\epsilon, 1+\epsilon]$ (se penalizan cambios que que

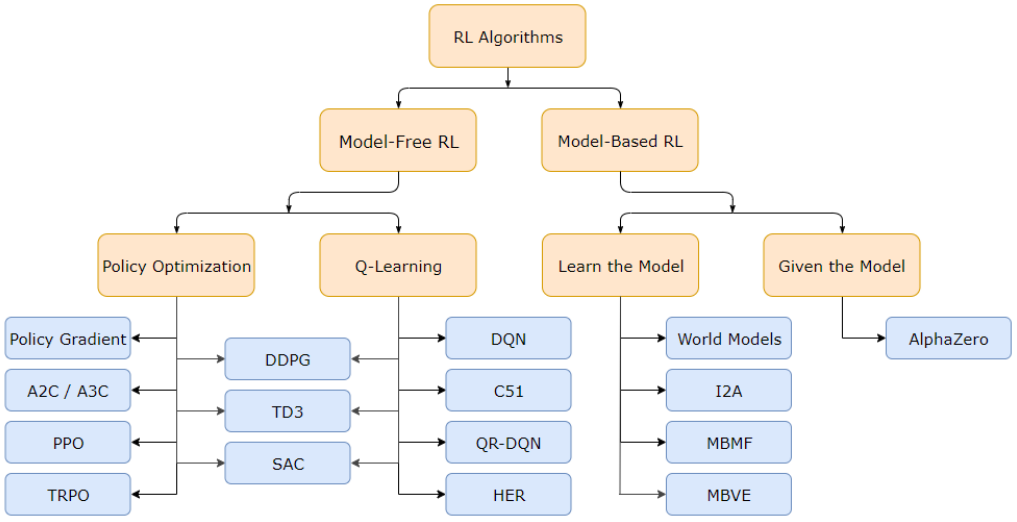
[2] El desarrollo completo se puede encontrar en (Graesser and Keng, 2019).

se alejen del valor unitario) utilizando la función *clip*. Esto evita la degradación del rendimiento y mejora la estabilidad del entrenamiento, impidiendo que las actualizaciones de la política sean demasiado grandes.

El algoritmo Proximal Policy Optmization (PPO) hace uso del objetivo subrogado visto en la Ecuación 3.10, el cual garantiza mejoras monótonas y pequeñas a la función política.

3.1.4. Otros algoritmos

Figura 3.1: Resumen de algoritmos más populares en DRL. Solo se consideran los algoritmos que no aprenden el modelo del proceso de Markov, si no solamente funciones de valor, la política o ambas. Esta imagen ha sido tomada de la Wiki de OpenAI SpinningUp: `https://spinningup.openai.com/en/latest/index.html`



En este capítulo, se han presentado los algoritmos más utilizados en el campo del DRL, específicamente PPO y las variantes de DQN. Sin embargo, también existen otros algoritmos más recientes que están ganando popularidad, como se puede ver en la Figura 3.1. Uno de los más prominentes en sistemas de control con espacios continuos es Soft Actor-Critic (SAC) (Haarnoja et al., 2018). Este algoritmo se basa en una arquitectura Actor-Crítico, donde el actor maximiza una función de entropía que busca equilibrar la exploración y la explotación, intentando obtener las recompensas máximas mientras actúa de manera estocástica. Una gran ventaja de

este algoritmo es su eficiencia en el muestreo, ya que es off-policy y mejora el proceso de recolección de datos.

Otro algoritmo adicional de gran interés es Maximum a Posteriori Policy Optimisation (MPO) (Abdolmaleki et al., 2018). Este enfoque de aprendizaje utiliza el método de estimación máxima a posteriori para incorporar conocimiento previo sobre la política en la optimización. MPO resulta especialmente útil en problemas de alta dimensionalidad, donde los datos generados por los simuladores pueden ser lentos, como es común en la mayoría de los simuladores físicos. Finalmente, el algoritmo Twin Delayed Deep Deterministic (TD3) (Fujimoto et al., 2018), el cual se basa en una mejora del conocido algoritmo DDPG (Lillicrap et al., 2019), incrementa también la estabilidad y rendimiento en ciertos problemas. Al igual que SAC y MPO, TD3 es también un algoritmo off-policy.

En resumen, tanto SAC, MPO o TD3 son algoritmos que expanden las capacidades de la optimización de políticas en DRL. Al igual que PPO, estos tres algoritmos representan importantes avances en problemas complejos de control continuo y muestran potencial para abordar el desafío del acelerador de IFMIF-DONES. Es importante mencionar que ningún algoritmo es mejor que otro a priori, por lo que hay que comparar sus resultados una vez aplicados al problema en concreto.

3.2. Deep Reinforcement Learning aplicado al control de instrumentación física

El paradigma DRL ha demostrado ser exitoso en diversas aplicaciones de control de sistemas. En el ámbito de la Física, merece especial atención el trabajo realizado por investigadores de Google DeepMind (Degrave et al., 2022), quienes aplicaron el algoritmo MPO para controlar las configuraciones del plasma en el Tokamak TCV. Los resultados se pueden ver en las Figuras 3.2, 3.3 y 3.4, donde se muestra la arquitectura utilizada y un ejemplo de la geometría alcanzada por el plasma. Los resultados de este experimento representaron un gran logro y un avance significativo tanto en el campo de la fusión nuclear como en el de la IA, dado el alto grado de

complejidad del problema.

Por otro lado, se han realizado varios trabajos aplicando DRL al ámbito de los aceleradores de partículas, aunque no alcanzan el mismo nivel de complejidad que el caso del tokamak. En su mayoría, estos estudios se han centrado en secciones experimentales de módulos de aceleradores. Uno de los ejemplos destacados es el estudio llevado a cabo por investigadores de DESY y KIT para desarrollar aceleradores autónomos (Eichler et al., 2021). En dicho estudio, lograron enfocar y centrar el haz de partículas en una pantalla objetivo mediante el uso de DRL. Los resultados y el bucle de control se presentan en las Figuras 3.5 y 3.6, respectivamente.

Ambos trabajos pueden considerarse ejemplos a seguir en el diseño de arquitecturas DRL para el control de sistemas físicos de última generación. Este enfoque implica la identificación del problema específico a resolver, la definición de los espacios de acciones y observaciones del sistema, el establecimiento de una función de recompensa adecuada para guiar el proceso de aprendizaje, y el diseño de redes neuronales para aproximar las funciones utilizadas por los algoritmos. Una vez que el agente DRL ha sido entrenado, puede lograr un control autónomo del instrumento, siendo suficiente que un experto especifique el objetivo a alcanzar.

Figura 3.2: Esquemas del entorno y de la política utilizados en el trabajo llevado a cabo por DeepMind (Degrave et al., 2022). Izquierda: Entorno simulado del tokamak que utiliza las ecuaciones de Grad-Shafranov. El entorno toma como input una acción y devuelve una medida y una recompensa, además de pasar al siguiente estado. Derecha: La red neuronal que representa la política de control. Las entradas son los parámetros objetivo y las medidas, mientras que las salidas son las acciones del agente.

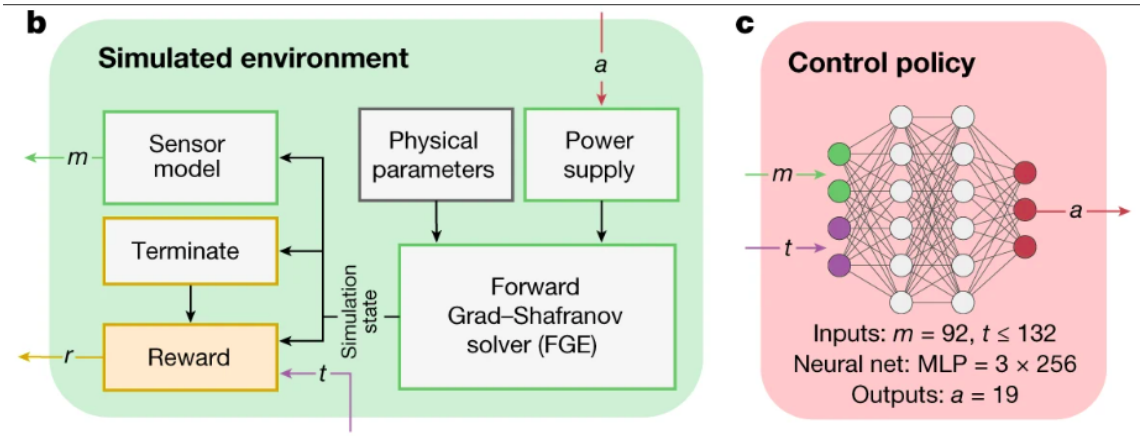


Figura 3.3: Esquema de la arquitectura en el trabajo llevado a cabo por DeepMind (Degrave et al., 2022). La política de control toma una configuración objetivo y el estado del plasma en el tokamak para encontrar las acciones óptimas.

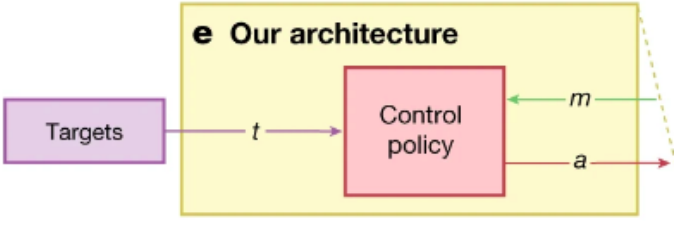


Figura 3.4: Esquema del despliegue del controlador en el trabajo llevado a cabo por DeepMind (Degrave et al., 2022). Izquierda: tokamak TCV con todos las bobinas actuadoras. Derecha: sección del tokamak donde se puede ver la conguración alcanzada con el plasma y algunos parámetros relevantes.

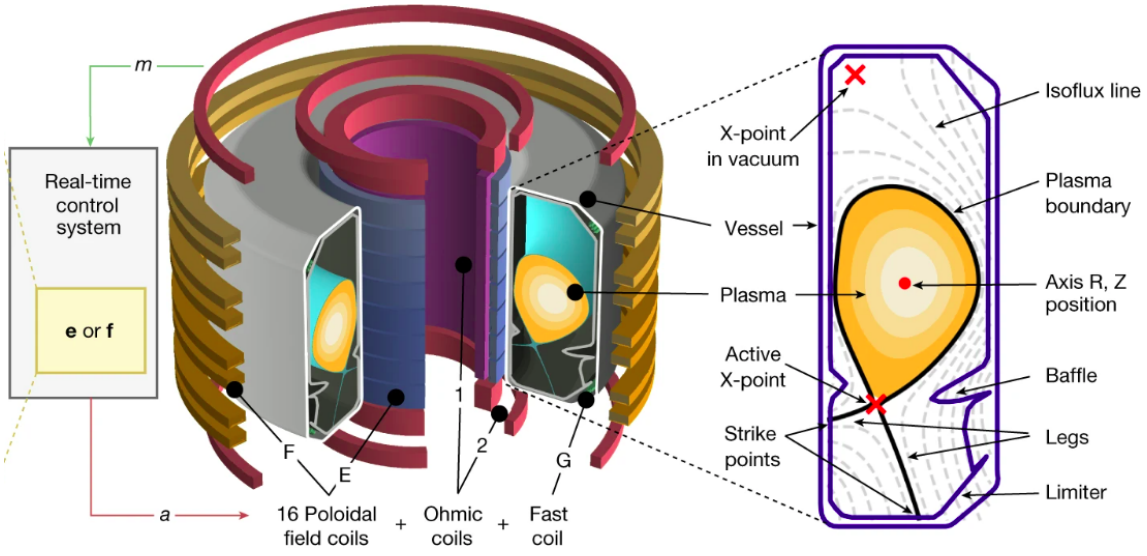


Figura 3.5: Lazo de control del área experimental del acelerador ARES. Se muestran los actuadores (imanes correctores y cuadrupolos en la parte inferior), la pantalla que recoge la información del haz, y las entradas y salidas del agente. La función objetivo es calculada con los parámetros objetivo y los parámetros alcanzados en cada estado. Imagen tomada de Eichler et al. (2021).

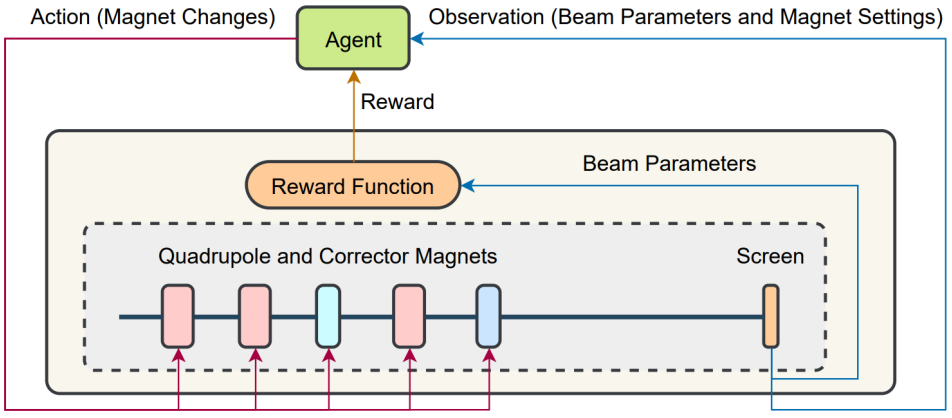
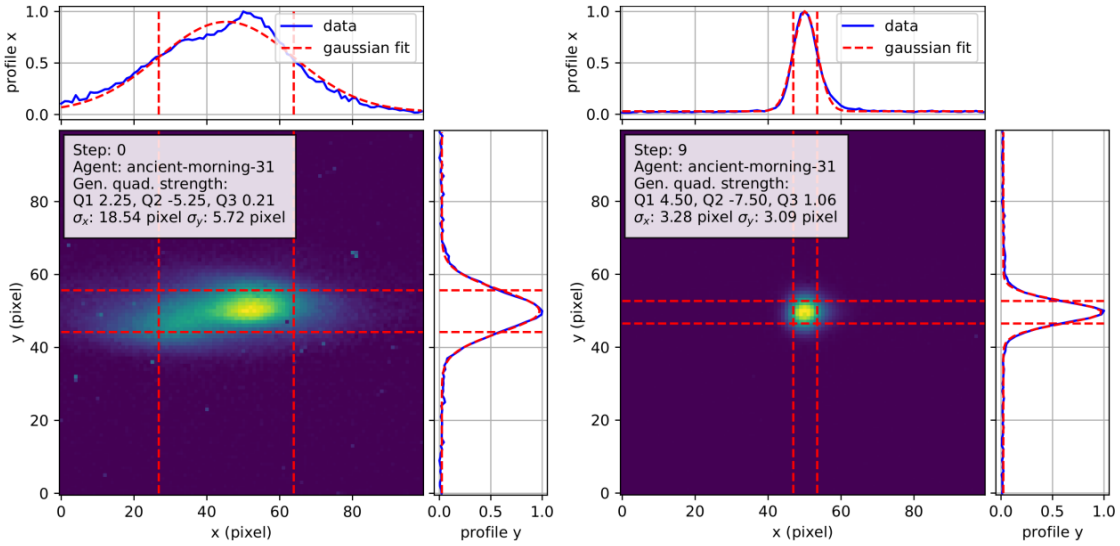


Figura 3.6: Resultados obtenidos mediante un sistema de control DRL aplicado al acelerador ARES (Eichler et al., 2021). Izquierda: estado inicial. Derecha: objetivo cumplido por el agente, que utiliza el algoritmo DDPG.



Capítulo 4

Diseño de la arquitectura del controlador

Diseñar controladores para sistemas físicos puede resultar un desafío de considerable complejidad, especialmente en casos como los aceleradores de partículas. En estos sistemas, se pueden encontrar espacios de alta dimensionalidad, la dinámica del haz tiene un comportamiento no lineal, y existe una incertidumbre inherente en las simulaciones. Debido a esto, los enfoques de control convencionales a menudo se ven limitados para abordar estos problemas de manera efectiva. Sin embargo, como se ha detallado en la sección anterior, el DRL surge como una alternativa capaz de encontrar secuencias de acciones óptimas que resuelven de manera eficaz este tipo de situaciones.

No obstante, la intervención de un experto en el campo sigue siendo indispensable para definir los objetivos precisos que se pretenden alcanzar con el haz de partículas. La experiencia y el conocimiento de la materia son fundamentales para establecer las metas y restricciones específicas que guiarán el proceso de aprendizaje. En este sentido, resulta crucial determinar las características deseadas del haz, como su energía, intensidad y estabilidad, ya que esto contribuye en gran medida al éxito de la aplicación de los algoritmos DRL.

En este capítulo, se detalla el conjunto de pasos que se siguen para diseñar agen-

tes inteligentes destinados al control de un módulo experimental en un acelerador de partículas. Cada uno de estos puntos describe etapas clave en el desarrollo del proyecto:

- **Análisis del Estado del Arte de Algoritmos DRL**: se examina la literatura actual en algoritmos DRL, como se ha hecho ya en el Capítulo 3. Dado que este es un problema de control continuo, se eligen los algoritmos PPO y SAC. No obstante, se considera también la posibilidad de probar otros enfoques discretizando los espacios de observaciones y acciones.
- **Estudio del acelerador de partículas**: se analiza la dinámica de haces de partículas cargadas para entender el funcionamiento del entorno. Esto se lleva a cabo en la Sección 4.1, donde se describen las partes fundamentales del acelerador lineal de IFMIF-DONES y se explica el funcionamiento básico de los aceleradores de partículas.
- **Elección de un Framework DRL**: se lleva a cabo un estudio comparativo de los frameworks DRL más populares, como se puede ver en la Sección 4.2.
- **Selección de un Simulador**: es el núcleo del entorno. Un simulador recibe acciones y cambia el estado del acelerador. Para esta etapa, se opta por el simulador *open source* Ocelot (Agapov et al., 2014) que se explica en la Sección 4.2.
- **Definición del Problema**: el objetivo es controlar el haz de partículas en un acelerador mediante la modificación de los imanes actuadores. Los detalles específicos de la formulación del problema de Markov se abordan en la Sección 4.3.
- **Creación del Entorno**: a partir del simulador, se construye una interfaz compatible con Gymnasium (Towers et al., 2023), tal como se describe en la Sección 4.2. Además de ofrecer entornos predefinidos para el entrenamieto de agentes DRL, esta librería proporciona las herramientas necesarias para construir entornos personalizados desde cero.
- **Esquema de la arquitectura del controlador**: se resumen todas las com-

ponentes de los agentes. Esto se realiza en la Sección 4.4

- **Validación de la Metodología**: finalmente, se inicia la fase de entrenamiento de los agentes DRL, prestando especial atención a las curvas de aprendizaje a través de Tensorboard (Abadi et al., 2015). Una vez los agentes están entrenados, se procede a evaluar su rendimiento. La Sección 4.5 explica todos los experimentos llevados a cabo.

4.1. El acelerador lineal de IFMIF-DONES

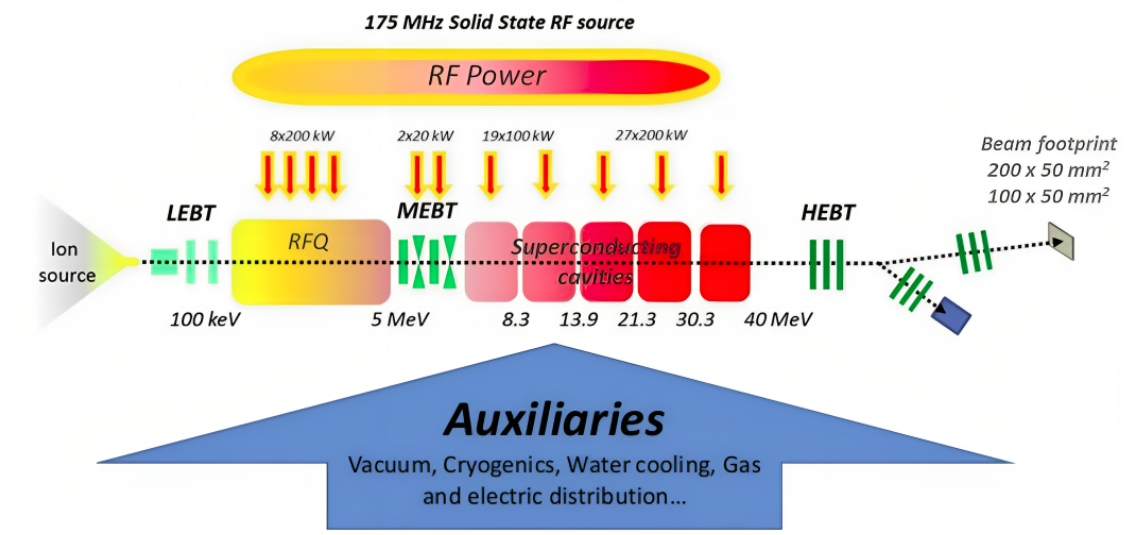
4.1.1. Módulos del acelerador

Los sistemas del acelerador lineal de IFMIF-DONES tienen el objetivo de acelerar iones de deuterio en modo de onda continua (CW, del inglés *continuous wave*) hacia el blanco de litio (Podadera et al., 2021a; Ibarra et al., 2019). Este proceso se lleva a cabo mediante el uso de varios módulos, los cuales crean, transportan y configuran el haz para que alcance los parámetros deseados en la colisión con el blanco. Las distintas partes del acelerador, como se observan en la Figura 4.1, son las siguientes:

- Sistema Inyector: se encarga de generar y extraer un haz de deuterones a una corriente de 140 mA y una energía de 100 keV.
- Cuadrupolo de radiofrecuencia (RFQ): Este dispositivo agrupa las partículas cargadas en un *bunch* y acelera el haz hasta una energía de 5 MeV.
- Línea de transporte del haz de energía media (MEBT): acondiciona el haz para su procesamiento posterior.
- Módulo de radiofrecuencia superconductora (SRF): este componente acelera el haz hasta alcanzar su energía final de 40 MeV.
- Línea de Transporte del Haz de Alta Energía (HEBT): Esta línea es responsable de dar forma al perfil del haz y direccionarlo hacia el flujo de litio.

Adicionalmente, a lo largo del acelerador se distribuyen dispositivos de diagnóstico del haz y herramientas de instrumentación. Estos elementos tienen la función de

Figura 4.1: Esquema del acelerador lineal de IFMIF-DONES (Podadera et al., 2021b). De izquierda a derecha: fuente de iones, LEBT, RFQ, MEBT, SRF, HEBT y cámara de colisión. Los sistemas auxiliares necesarios para el funcionamiento del acelerador se muestran en la parte inferior.



medir diversos parámetros como la corriente del haz, su perfil, o las pérdidas de energía. Dichas magnitudes son utilizadas por los sistemas DRL para construir el espacio de estados e intentar llegar al objetivo deseado. En la Figura 4.2, se puede observar las geometrías objetivo para el haz de deuterones en la cámara de colisión.

4.1.2. Física de aceleradores

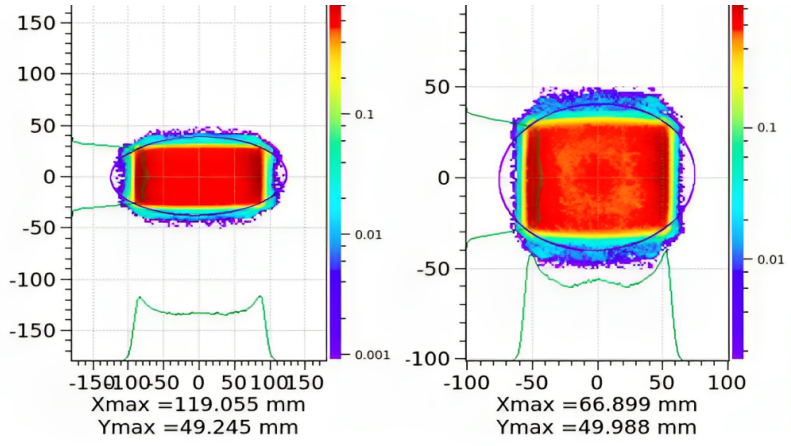
Un acelerador de lineal tiene como objetivo incrementar la energía de un haz de partículas cargadas (como electrones o protones) a la vez que alcanza una cierta geometría en su envolvente, como se puede ver en la Figura 4.2. La energía y la trayectoria de estas partículas son modificadas por distintos tipos de elementos magnéticos colocados en en el interior de los módulos vistos en la Fig. 4.1. Algunos de los elementos más utilizados son:

- **Cuadrupolos**: se emplean para controlar el tamaño y la forma del haz, enfocando partículas en un plano y desenfocándolas en el otro.
- **Sextupolos y octupolos**: elementos de enfoque de orden superior para co-

rregir efectos no lineales.

- **Bends**: campos magnéticos que curvan la trayectoria del haz.
- **Correctores**: proporcionan pequeños ajustes a la posición del haz.
- **Solenoides**: genera un campo magnético uniforme a lo largo de su eje, a menudo utilizado para enfocar o colimar partículas cargadas.
- **Cavidades**: estructuras para aumentar la energía de las partículas cargadas.
- **Monitores**: sensores para obtener mediciones del haz en un punto específico del entramado.

Figura 4.2: Geometrías del haz deseadas para la colisión con el litio (Ibarra, 2019).

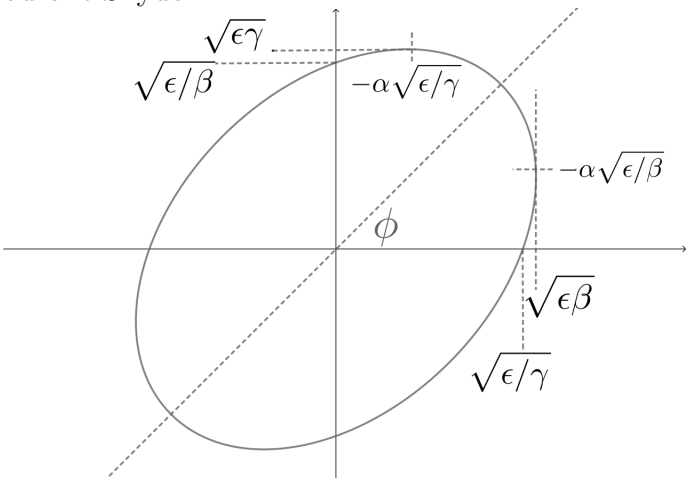


Mediante la combinación correcta de estos elementos (y sus parámetros), se pueden alcanzar las energías y geometrías deseadas en IFMIF-DONES. Además, dichos elementos pueden ser utilizados como actuadores en la definición de un problema DRL.

Es importante también conocer el funcionamiento de estos aceleradores a nivel matemático, para el uso correcto de los simuladores que modelan los entornos. En física de aceleradores, los parámetros Twiss (Courant-Snyder) son esenciales para describir las características de un haz de partículas cargadas en diferentes puntos de su trayectoria (Holzer, 2014). Dichos parámetros permiten calcular la evolución general del haz a través de ecuaciones matriciales de la óptica en vez de simular partículas a nivel individual. Un ejemplo de envolvente del haz para una dimensión en el espacio

de fases (posición-velocidad) puede verse en la Fig. 4.3.

Figura 4.3: Elipse del haz en el espacio de fases para una coordenada, $x - x'$. Los parámetros Twiss son α, β y γ. La emitancia del haz ϵ se refiere al área de la elipse. Estos parámetros se pueden calcular para los ejes x, y y z . Fuente: Wikipedia, parámetros de Courant-Snyder.



En aceleradores circulares se suelen buscar soluciones periódicas y estables de los parámetros Twiss. No obstante, en el caso de aceleradores lineales como es IFMIF-DONES, el objetivo principal es llevar el haz a una cierta configuración para la colisión con el objetivo. En estas situaciones, no es necesario que el acelerador tenga una disposición magnética periódica ni que se busquen soluciones estables para el haz.

4.2. Software

4.2.1. Frameworks de DRL

Debido al éxito del DRL, han surgido numerosas librerías para facilitar el desarrollo y la experimentación de algoritmos de este paradigma. Estos frameworks, en su mayoría *open source*, proporcionan a desarrolladores e investigadores los recursos necesarios para diseñar, entrenar y evaluar agentes de DRL de manera eficaz. En esta sección, se exploran algunas de las principales bibliotecas para el DRL, cada una de las cuales ofrece características y capacidades únicas. Se ha seleccionado un

subconjunto de frameworks en función de diversas características:

- **Popularidad, comunidad y soporte**: los frameworks con una fuerte presencia y trayectoria tienen más probabilidades de recibir actualizaciones continuas, mejoras y apoyo de la comunidad.
- **Simplicidad**: se busca una API intuitiva con diferentes niveles de abstracción. Un framework fácil de usar proporcionará abstracciones de alto nivel, facilitando la definición de agentes, entornos y procesos de entrenamiento sin necesidad de un conocimiento profundo de los algoritmos subyacentes o de los detalles de implementación.
- **Flexibilidad**: posibilidad de implementar nuevas técnicas en caso de ser necesario.
- **Documentación**: una documentación adecuada, tutoriales y ejemplos también son esenciales para facilitar el uso de una librería.
- **Algoritmos implementados**: se buscan librerías que incorporen de serie los algoritmos requeridos para el proyecto, lo que evita gastar tiempo en implementaciones de baja abstracción.
- **Compatibilidad con entornos**: se verifica si la librería ofrece compatibilidad con una amplia gama de entornos de simulación.
- **Cobertura de código**: se refiere al grado en que el código base del framework se verifica mediante pruebas automatizadas. Mayor cobertura del código indica mayor confiabilidad y reducción de errores.
- **Escalabilidad**: las capacidades de multiprocesamiento y distribución permiten interacciones en entornos paralelos. Esto puede acelerar el entrenamiento de los algoritmos y facilitar el estudio de los hiperparámetros.
- **Eficiencia computacional**: la velocidad a la que el framework ejecuta tareas de entrenamiento e inferencia, utilizando optimizaciones como GPU, procesamiento paralelo y algoritmos eficientes.
- **Soporte multi-agente**: capacidad del framework para manejar escenarios con

múltiples agentes, interactuando y aprendiendo simultáneamente.

- **Offline learning**: posibilidad de entrenar agentes utilizando conjuntos de datos previamente recopilados sin interacciones en tiempo real con el entorno. Esto es últil cuando los simuladores son demasiado lentos para el entrenamiento.

Stable Baselines 3 (SB3) es una librería *open source* diseñada para facilitar la replicación de resultados en el ámbito del DRL, utilizando PyTorch como backend (Raffin et al., 2021). Este framework ofrece una implementación sólida de algoritmos esenciales de DRL, junto con una interfaz simple y documentación exhaustiva que simplifican el entrenamiento y la comparación de agentes. SB3 se enfoca en algoritmos de aprendizaje sin modelo para un solo agente, priorizando la estabilidad de las implementaciones por encima de la inclusión de nuevas características o algoritmos. Las propiedades clave de SB3 incluyen:

- **Interfaz intuitiva:** el entrenamiento de agentes se logra con código simple y fácil de entender.
- **Documentación completa:** el código está exhaustivamente documentado y se provee de una guía del usuario y tutoriales.
- **Implementaciones de alta calidad:** las curvas de aprendizaje se comparan con resultados publicados, y el código está respaldado por pruebas unitarias.
- **Versatilidad:** SB3 incluye algoritmos Estado del Arte y una variedad de características independientes del algoritmo.
- **Entorno experimental:** SB3 Zoo brinda scripts para entrenar, evaluar agentes y ajustar hiperparámetros, entre otras funcionalidades.
- **SB3 Contrib:** las funcionalidades experimentales se implementan en un repositorio separado para no afectar la estabilidad del repositorio principal.

Respecto a los algoritmos incorporados en SB3, éstos abarcan espacios de acción tanto continuos como discretos, además de admitir entornos multiproceso que permiten la ejecución en paralelo en múltiples núcleos para agilizar el proceso de entrena-

miento. Ejemplos notables son: A2C, DDPG, DQN, PPO, SAC y TD3. De manera similar, SB3 Contrib constituye un paquete experimental que sirve como repositorio para algoritmos y herramientas de RL considerados en fase experimental.

En lo que respecta al proyecto, SB3 incluye algoritmos que podrían ser aplicados al diseño de sistemas con espacios continuos, como los del acelerador de IFMIF-DONES. Además, SB3 se enfoca en la estabilidad y la simplicidad, lo que hace que esta librería sea útil para entrenar agentes en las primeras etapas del proyecto. No obstante, a pesar de sus ventajas, SB3 no admite entrenamiento multiagente, y su capacidad de entrenamiento distribuido es menos escalable en comparación con otros frameworks. Por otra parte, algoritmos como MPO no están implementados.

Ray RLlib es una biblioteca *open source* diseñada para entrenar agentes DRL a nivel industrial. Proporciona un soporte robusto para aplicaciones y ofrece APIs sencillas (Liang et al., 2018). Este framework permite la personalización de varios aspectos de los flujos de trabajo, ofreciendo flexibilidad a los desarrolladores. Algunas características clave de RLlib incluyen:

- Soporte para los frameworks TensorFlow y PyTorch.
- Capacidades de entrenamiento en paralelo, permitiendo un entrenamiento más rápido utilizando múltiples CPUs o nodos simultáneamente.
- Desarrollo de entornos RL multiagente.
- Soporte para entrenamiento offline/histórico.
- Incluye Ray Tune, una herramienta para el estudio de hiperparámetros.

Por otra parte, esta librería ofrece una amplia gama de algoritmos implementados como A2C, DQN, PPO, o SAC , ideales para problemas de control en espacios continuos. Finalmente, el framework se ha aplicado con éxito a problemas de control reales y complejos en ámbitos como la robótica y el control industrial. Ofrece ventajas de escalabilidad y velocidad en comparación con SB3, especialmente cuando se combina con Ray Clusters para el entrenamiento distribuido. Como aspecto negativo, cabe mencionar que la API es mucho menos intuitiva que la SB3.

Tianshou es una biblioteca de Python para DRL basada en el backend de PyTorch, la cual presenta una infraestructura flexible, compatible tanto con entrenamiento online como offline. Este framework implementa también múltiples algoritmos clásicos bajo una interfaz unificada (Weng et al., 2022). Una de las características centrales de Tianshou es la modularidad. Proporciona componentes para DRL en lugar de scripts de entrenamiento, permitiendo prototipado rápido mediante factores de infraestructuras comunes y ajustes mínimos de variables para aplicar técnicas como el muestreo paralelo de datos.

En el diseño de sistemas de control para IFMIF-DONES, la elección de frameworks depende de las necesidades específicas y objetivos de control. SB3 destaca por su sencillez y estabilidad, ideal para problemas DRL sin múltiples agentes[1] ni entrenamiento distribuido. Tianshou es flexible y de uso fácil, con modularidad y soporte para offline learning y entornos multi-agente. RLlib destaca por su escalabilidad y compatibilidad con TensorFlow y PyTorch. Una comparación de los frameworks más importantes se puede ver en la Figura 4.4. Los que han sido utilizados en más proyectos son los que se han explicado en esta Sección.

Figura 4.4: Comparación de frameworks DRL. Imagen tomada de Raffin et al. (2021).

	SB3	OAI Baselines	PFRL	RLlib	Tianshou	Acme
Backend	PyTorch	TF	PyTorch	PyTorch/TF	PyTorch	Jax/TF
User Guide / Tutorials	✓/ ✓	✗/ –	–/ ✓	✓/ ✓	–/ ✓	–/ ✓
API Documentation	✓	✗	✓	✓	✓	✗
Benchmark	✓	✓	✓	✓	–	–
Pretrained models	✓	✗	✓	✗	✗	✗
Test Coverage	95%	49%	?	?	94%	74%
Type Checking	✓	✗	✗	✓	✓	✓
Issue / PR Template	✓	✗	✗	✓	✓	✗
Last Commit (age)	< 1 week	> 6 months	< 1 month	< 1 week	< 1 month	< 1 week
Approved PRs (6 mo.)	75	0	13	222	85	5

Finalmente, en el contexto de este proyecto, merece la pena resaltar el estudio de otros dos marcos de trabajo. La biblioteca **d3rlpy** proporciona implementaciones de algoritmos DRL offline, resultando particularmente relevantes en situaciones donde los simuladores presentan limitaciones de velocidad (Seno and Imai, 2022). Por otra

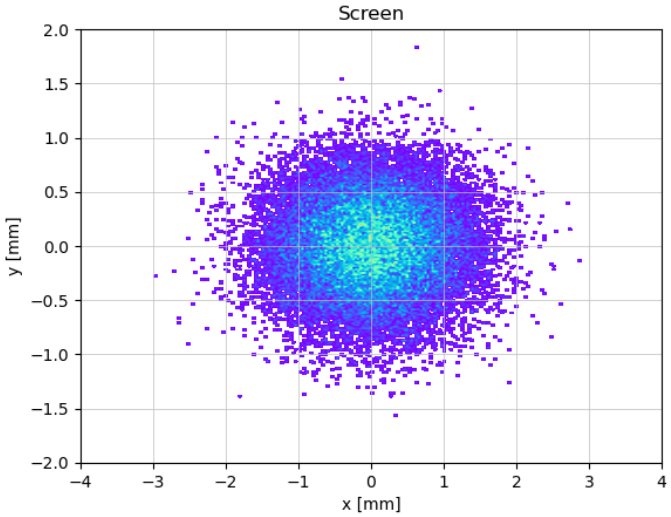
[1] Se pueden entrenar múltiples agentes para distintos objetivos y coordinarlos posteriormente.

parte, el framework **ACME** (Hoffman et al., 2022) incorpora la implementación del algoritmo MPO, el cual ha sido empleado en el estudio de control de plasma en un tokamak desarrollado por Deep Mind.

4.2.2. Simulador Ocelot

Ocelot[2] es un conjunto de herramientas escritas en Python con el propósito de simular haces de electrones y fuentes de luz basadas en anillos de almacenamiento (Agapov et al., 2014). Hace uso extensivo de las bibliotecas NumPy y SciPy para cálculos numéricos y científicos eficientes, así como de matplotlib para generar figuras de alta calidad. Ocelot es un proyecto de código abierto desarrollado por físicos de instituciones como el European XFEL, DESY (Alemania) y el Instituto NRC Kurchatov (Rusia).

Figura 4.5: Distribución de electrones (gaussiana bidimensional) que llegan al final de la línea del acelerador. Fuente: elaboración propia utilizando la API de Ocelot.



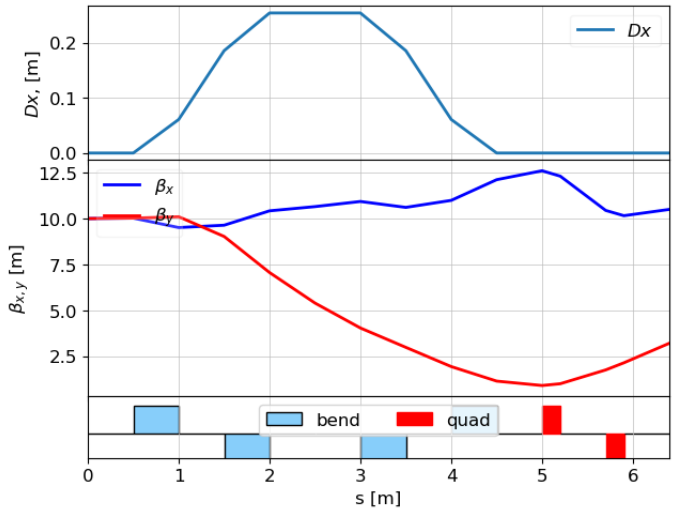
Además de la inclusión de todos los elementos magnéticos relevantes para un acelerador lineal, vistos en la Sección 4.1, Ocelot incorpora:

- **Dinámica de Partículas Cargadas en Aceleradores (CPBD):** aborda la dinámica de partículas cargadas en aceleradores.
- **Seguimiento (Tracking):** simula las trayectorias del haz de partículas.

[2] https://github.com/ocelot-collab/ocelot

- **Ajuste (Matching):** busca ajustar los parámetros del haz para optimizar el rendimiento del acelerador.
- **Efectos Colectivos (Collective Effects):** estudia el comportamiento colectivo e interacciones de partículas.
- **Carga Espacial (Space Charge):** modela el efecto de las interacciones entre partículas cargadas en la dinámica del haz resolviendo la ecuación de Laplace en tres dimensiones.

Figura 4.6: Arriba: evolución de la función de dispersión en la coordenada x a lo largo del acelerador. Mitad: evolución de los parámetros beta Twiss en las coordenadas (x, y) para una geometría específica. Abajo: elementos magnéticos presentes en el acelerador. Fuente: elaboración propia utilizando la API de Ocelot.



En las Figuras 4.5 y 4.6 se puede apreciar algunos de los usos de Ocelot, como obtener los parámetros Twiss en la línea del acelerador o la distribución de electrones que llegan al objetivo.

4.2.3. Gymnasium

Gymnasium (anteriormente conocida como OpenAI Gym) es una biblioteca *open source* desarrollada por la fundación Farama, la cual brinda acceso a una amplia gama de entornos predefinidos destinados a entrenar algoritmos DRL (Towers et al., 2023). Estos entornos abarcan desde juegos clásicos hasta desafíos más complejos, y

se diseñan para simular situaciones del mundo real. Esta herramienta es esencial en la resolución de problemas DRL gracias a su capacidad de integración con la mayoría de frameworks DRL.

Por otra parte, Gymnasium ofrece la posibilidad de crear wrappers de compatibilidad para entornos simulados, lo que otorga a los investigadores la flexibilidad de ajustar los entornos existentes a sus necesidades particulares o incluso de crear desde cero entornos personalizados para abordar desafíos específicos. En la Figura 4.7 se muestra el proceso de creación del wrapper utilizado en este trabajo.

Figura 4.7: Proceso de elaboración del wrapper de compatibilidad de Gymnasium para la creación de un entorno de un acelerador de partículas utilizando Ocelot. En primer lugar, se toma la API de Ocelot y se construye un entramado del acelerador. A continuación, se crea una clase en Python que tiene el papel de simulador con inputs y outputs. Finalmente, se escribe una clase (hereda de Gymnasium) que representa el wrapper o entorno compatible. Fuente: elaboración propia.

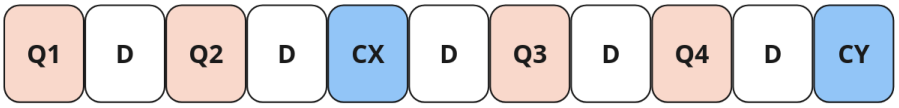
Para este proyecto, se han concebido y elaborado dos wrappers individuales. Cada uno de estos wrappers constituye un entorno que se integra sin problemas con los frameworks de entrenamiento y aborda un problema específico dentro del ámbito del DRL. Dado que es crucial formular el problema en el contexto de un proceso de Markov y precisar la función de recompensa correspondiente, los detalles relativos a ambas configuraciones son detallados en la Sección 4.3.

4.3. Definición del problema

4.3.1. Módulo experimental y objetivos

El objetivo central de este proyecto consiste en desarrollar uno o múltiples agentes DRL con la capacidad de controlar de manera autónoma (con ayuda de un experto) haces de partículas cargadas en aceleradores lineales. Controlar un haz significa alcanzar unos parámetros objetivo mediante una secuencia de acciones óptima. Para este propósito, y como una primera aproximación al problema, se ha procedido a implementar el siguiente módulo experimental de la Figura 4.8 en Ocelot.

Figura 4.8: Módulo experimental del acelerador lineal. Se muestran los distintos elementos del entramado: cuadrupolos (Q_j), zonas de deriva sin interacciones (D) e imanes correctores para las coordenadas transversales (C_k). El haz se inyecta desde la parte izquierda y se propaga por las distintas zonas. Fuente: elaboración propia.



Para cumplir el objetivo de alcanzar configuraciones específicas del haz en cuanto a posición y deformación, se plantea un sistema multi-agente. Esto es debido a que los efectos de los cuadrupolos están desacoplados de los efectos de los imanes correctores, por lo que los agentes pueden actuar de manera independiente aunque reciban las mismas medidas. El primer agente se encarga de controlar la posición del haz con los imanes correctores (C_x, C_y), modificando los ángulos (θ_x, θ_y) correspondientes a cada actuador en incrementos limitados. El segundo agente se encarga de cambiar la geometría del haz manipulando los campos magnéticos de primer orden k_1 [3] de los cuatro cuadrupolos (Q_1, Q_2, Q_3, Q_4). Por lo tanto, para cada agente se construye un entorno de Gymnasium distinto, puesto que los espacios de acciones y observaciones no son iguales.

[3] Ocelot permite también modifcar el campo de segundo orden y la longitud del elemento.

4.3.2. Espacios de acciones y medidas

Es necesario formalizar matemáticamente el sistema como un problema de Markov. Para ello hay que definir un espacio de acciones y un espacio de observaciones para cada agente del controlador. Para el agente que controla la posición del haz, los elementos del espacio de acciones son vectores continuos de dimensión 2 que se corresponden con los dos imanes correctores. De esta manera, un vector de este espacio posee incrementos en los ángulos:

$$a = (\Delta\theta_{Cx}, \Delta\theta_{Cy}) \tag{4.1}$$

Cada estado del espacio de observaciones debe contener la información suficiente para permitir que los algoritmos sean capaces de resolver el problema en cuestión. Se requiere un esfuerzo considerable en la selección de las variables a observar. Si estas variables son muchas, la complejidad del problema podría aumentar considerablemente, haciendo que el agente tenga dificultades en el aprendizaje. Si se opta por un número limitado de variables, el espacio de estados podría ser no completo, resultando en un problema irresoluble. Por lo tanto, los algoritmos DRL requieren conocimiento de un experto humano en su diseño para crear representaciones eficaces. En este problema, se ha elegido el siguiente espacio de estados de dimensión 6 y valores continuos:

$$s = ((\theta_{Cx}, \theta_{Cy}), (\mu_x, \mu_y), (T\mu_x, T\mu_y)) \tag{4.2}$$

donde se han incluido los valores totales actuales de los ángulos de los imanes correctores, las medias de la distribución de partículas cargadas al final del acelerador (μ_j) y los parámetros objetivo a alcanzar en dichas medias ($T\mu_j$). Cabe destacar que el espacio de observaciones inicial posee una dimensión igual al número de partículas inyectadas en el haz, cuyas propiedades geométricas se resumen con los parámetros de una Gaussiana bidimensional.

Los espacios para el segundo agente que controla la deformación del haz se definen de manera similar. El espacio de acciones es un espacio de valores continuos y dimensión 4, mientras que su correspondiente espacio de estados viene dado por los valores de

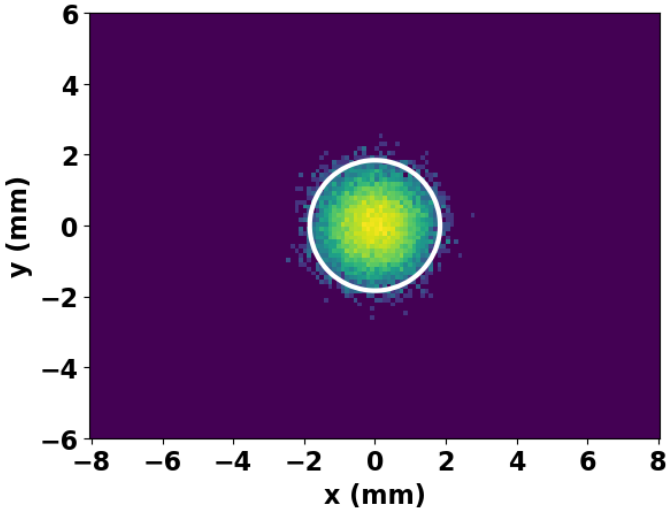
los cuadrupolos, las medidas actuales de las desviaciones típicas del haz y sus valores objetivo:

$$a = (\Delta k_{1,Q1}, \Delta k_{1,Q2}, \Delta k_{1,Q3}, \Delta k_{1,Q4}) \tag{4.3}$$

$$s = ((k_{1,Q1}, k_{1,Q2}, k_{1,Q3}, k_{1,Q4}), (\sigma_x, \sigma_y), (T\sigma_x, T\sigma_y)) \tag{4.4}$$

Los rangos de cada variable, para ambos problemas, se pueden observar en la Tabla A.1 del Apéndice A, donde se resumen los valores implementados en el wrapper de Gymnasium.

Figura 4.9: Distribución de electrones al final del módulo experimental utilizado en este trabajo. La línea blanca representa el contorno de nivel 2σ de una superficie Gaussiana de parámetros $(\mu_x = 0, \mu_y = 0, \sigma_x = 0{,}75, \sigma_y = 0{,}75)$. Los valores de todos imanes actuadores se inicializan a cero para esta configuración inicial. Fuente: elaboración propia.



La Figura 4.9 muestra el estado inicial del haz para los elementos magnéticos elegidos. Cada estado es una distribución de partículas [4] que impacta al final del módulo del acelerador. Esta distribución de partículas se puede interpretar como una distribución de probabilidad Gaussiana bidimensional de parámetros $(\mu_x, \mu_y, \sigma_x, \sigma_y)$, donde las medias son utilizadas por un agente para controlar la posición y las desviaciones típicas son empleadas por el otro agente para controlar la geometría del haz en la colisión.

[4] En cada time step se considera el paso de un *bunch* de 20000 partículas.

Finalmente, es importante mencionar que los parámetros $(\mu_x, \mu_y, \sigma_x, \sigma_y)$ utilizados para construir el espacio de estados, son calculados con un vector que contiene las posiciones finales de todas las partículas inyectadas. Es decir, los sensores utilizados son de posición. El vector inicial de observaciones tiene dimensión 20000 (tantas como número de partículas se inyectan), que se reduce a cuatro después de extraer los rasgos de interés (los cuatro parámetros de la distribución gaussiana bidimensional que forma el haz en pantalla)[5].

4.3.3. La función recompensa

Las funciones de recompensa tienen un papel crucial en el aprendizaje de los algoritmos DRL, al brindar la retroalimentación necesaria para dirigir al agente hacia su objetivo. Esto se debe a que la recompensa que obtiene por cada acción puede ser beneficiosa o perjudicial para lograr la meta establecida. Por ende, seleccionar una función de recompensa adecuada puede hacer que el problema se resuelva satisfactoriamente, que el agente aprenda cosas distintas o que directamente no aprenda.

En el contexto de este proyecto, las funciones de recompensa se han diseñado teniendo en cuenta distancias en espacios de configuraciones, como se puede ver en la Figura 4.10. En esta gráfica, las dos componentes (x, y) son las observaciones del haz para cada agente ((μ_x, μ_y) para el control de la posición y (σ_x, σ_y) para el control de la geometría). Se define también un threshold (o tolerancia) en este espacio para la condición de victoria que termina el episodio. Además, en caso de no alcanzar el objetivo, también se implementa una duración máxima de 200 time steps por episodio[6]. Teniendo en cuenta esto, la función de recompensa utilizada es:

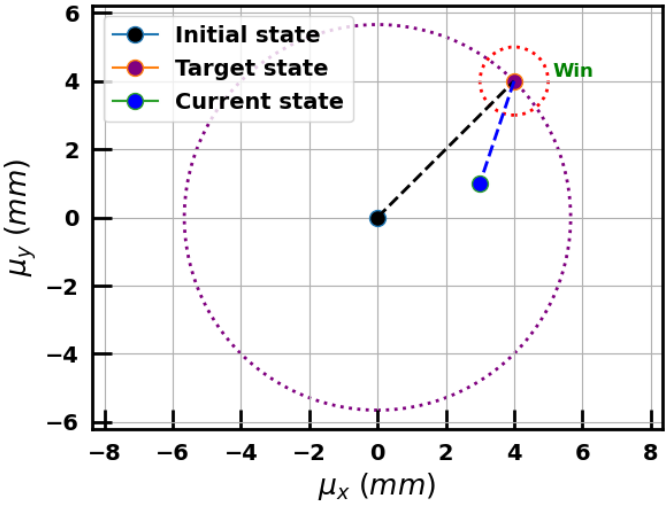
$$r = \begin{cases} -\frac{\sqrt{(x-x_t)^2+(y-y_t)^2}}{\sqrt{(x_0-x_t)^2+(y_0-y_t)^2}}, & \text{si} \quad \frac{\sqrt{(x-x_t)^2+(y-y_t)^2}}{\sqrt{(x_0-x_t)^2+(y_0-y_t)^2}} > \text{threshold}, \\ 10, & \text{si} \quad \frac{\sqrt{(x-x_t)^2+(y-y_t)^2}}{\sqrt{(x_0-x_t)^2+(y_0-y_t)^2}} \leq \text{threshold}, \end{cases} \tag{4.5}$$

En la Ecuación 4.5, los valores (x, y) representan las observaciones actuales del haz para cada agente, (x_t, y_t) denota el objetivo que se busca alcanzar, y (x_0, y_0) corres-

[5] *Feature engineering* en la construcción de espacios en DRL.

[6] Esto es otro hiperparámetro que depende de la complejidad del problema.

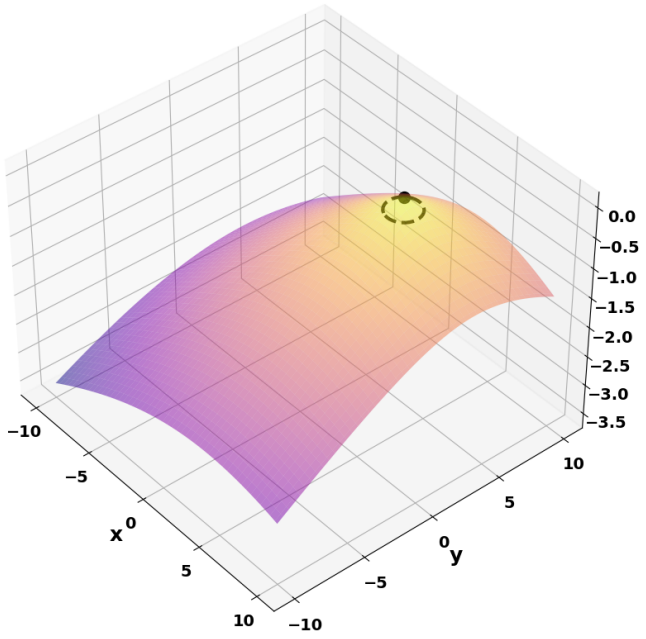
Figura 4.10: Explicación del espacio de configuraciones para la posición. Se muestra el estado inicial del haz, el estado objetivo y un estado intermedio. La línea punteada roja denota la condición de alcanzar el objetivo. El espacio de configuraciones para el control de la geometría es similar, pero utilizando las desviaciones típicas. Ver la Figura 4.11 para una representación tridimensional de la componente negativa de función de recompensa, calculada con la Ecuación 4.5. Fuente: elaboración propia.



ponde al estado inicial antes de que se tomen acciones. El threshold es un hiperparámetro a estudiar (se toman valores entre 0.2 y 0.1) que regular la precisión de los agentes. Mientras menor es este valor, más difícil se vuelve alcanzar la solución debido a que los agentes tienen que hilar más fino. Después de llevar a cabo diversas pruebas, se ha optado por esta función de recompensa por diversas razones. Al comenzar desde el estado inicial, el valor de la recompensa es -1 y gradualmente converge hacia 0 a medida que el agente se aproxima al objetivo. La inclusión de recompensas negativas motiva al agente a buscar de manera proactiva la solución óptima (en número de acciones), ya que de lo contrario continuará acumulando valores negativos. Además, esta función de recompensa comparte el mismo rango de valores para cualquier objetivo definido, lo que proporciona una base uniforme para la evaluación. Cuando el agente alcanza la región de éxito, el episodio se da por terminado y la recompensa obtenida es +10, un valor que, si bien está en la misma escala que los anteriores, refleja un logro sustancialmente mayor.

Es importante destacar que la función de recompensa utilizada no es única. Aunque existen directrices generales y metodologías comunes que guían la creación de estas funciones, en última instancia, su diseño estará ligado a las particularidades de cada problema. En muchas ocasiones, como es el caso presente, la construcción de funciones de recompensa se desarrolla mediante un proceso iterativo de ensayo y error. Esto permite ajustar y refinar la función dependiendo del comportamiento del agente y la consecución de los objetivos específicos.

Figura 4.11: Representación tridimensional de la componente negativa de función de recompensa del agente de control de la posición. El punto negro indica el objetivo a alcanzar y la línea punteada es el threshold para la condición de victoria. Fuente: elaboración propia.



4.4. Arquitectura del controlador

Para la etapa de entrenamiento, representada en la Figura 4.12, se han seleccionado los algoritmos PPO y SAC, los cuales están implementados dentro del marco de trabajo SB3. La elección de estos algoritmos Estado del Arte es debida a que han sido empleados para abordar una diversidad de problemas de control continuo, con resultados positivos.

En este proceso, el agente interactúa con un entorno simulado, representado en la Figura 4.13, construido a través de la combinación de las herramientas Ocelot y Gymnasium, con el propósito de aprender la función política óptima. Dicha política, que se aprecia en la Figura 4.14, puede ser desplegada en sistemas físicos una vez aprendida. En cada paso temporal, la política toma el estado del haz al final de la línea, lo compara con el objetivo a alcanzar, y ejecuta una serie de acciones.

Cabe destacar que todos estos esquemas son iguales tanto para el agente de control de la posición como para el agente que controla la deformación del haz, modificando simplemente los espacios de entrada y de salida. Incluso la función de recompensa es la misma a nivel matemático para ambos casos, pero en distintos espacios de configuraciones. Finalmente, todos los hiperparámetros elegidos se pueden ver en el Apéndice B.

Figura 4.12: Proceso de entrenamiento para los dos problemas. Los agentes (algoritmos) implementados en SB3 aprenden interactuando en un entorno simulado del acelerador. Es el loop de entrenamiento de la Figura 2.1, particularizado a este problema. La única diferencia es que en este bucle se toman de input los parámetros objetivo de la configuración a alcanzar. Dichos parámetros son definidos por el operador y son utilizados tanto en la política como en la función de recompensa. Fuente: elaboración propia.

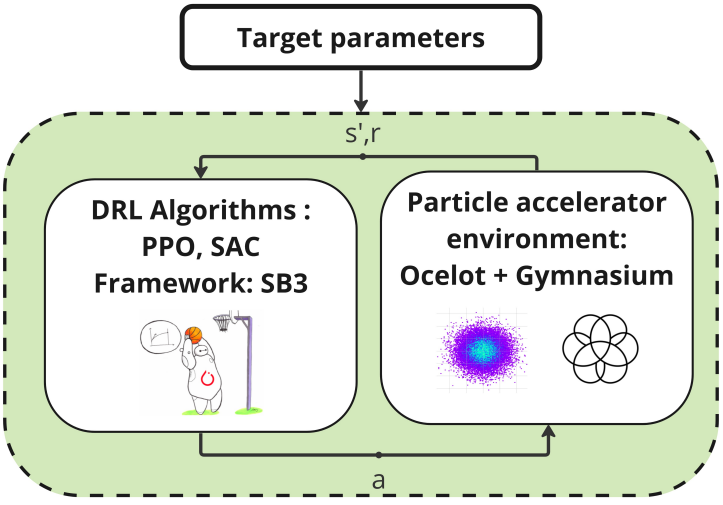


Figura 4.13: Esquema del entorno creado para el módulo experimental del entramado del acelerador. Dicho entorno toma acciones del agente y devuelve el siguiente estado y la recompensa. Los espacios de acciones, estados y la función recompensa han sido definidos en la Sección 4.3. Fuente: elaboración propia.

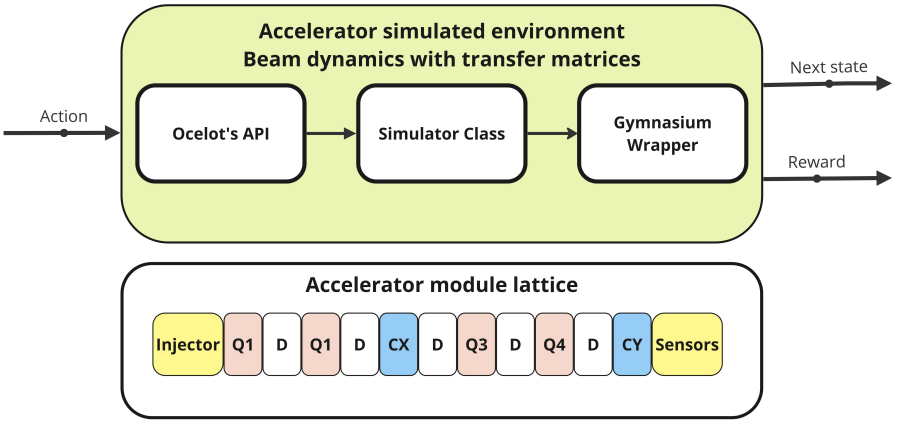
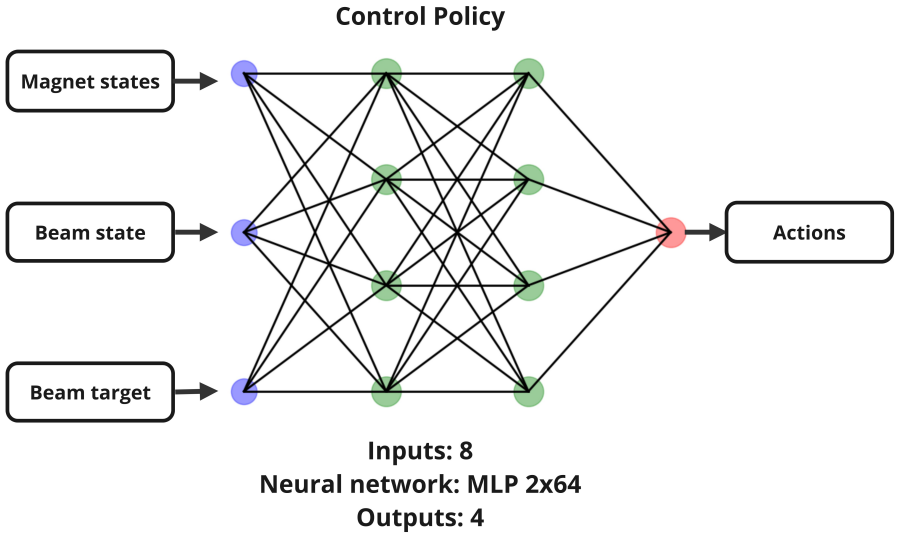


Figura 4.14: Arquitectura de la función política (es igual para ambos problemas). La capa de entrada (azul) tiene dimensión igual a la del espacio de estados. La capa de salida (rojo) tiene una neurona por actuador. Todos estos espacios han sido definidos en la Sección 4.3. Las capas ocultas (verde) poseen 64x64 neuronas con función de activación ReLU. Una vez aprendida, puede ser desplegada en un sistema físico para el control del haz. Fuente: elaboración propia.



4.5. Validación de la metodología

Para validar la metodología seguida, se han entrenado distintos agentes [7]. Para cada agente elegido se han llevado a a cabo los siguientes experimentos:

- **Entrenamiento con un objetivo:** Se entrena la política con un solo objetivo. El agente encuentra la solución óptima pero es incapaz de generalizar resultados a otros estados del haz. Este enfoque se utiliza como prueba de entrenamiento y tiene utilidad solamente en el ámbito de la optimización, no del control. Por ejemplo, se puede encontrar una solución que optimice todos los parámetros (longitudes, ángulos, fuerzas...) de cada elemento del acelerador.
- **Entrenamiento con múltiples objetivos:** Se discretiza el espacio de configuraciones a alcanzar y se emplean varios objetivos para entrenar al agente. Para el control de la posición se emplean ocho puntos, mientras que para el control de la geometría se emplean cuatro. Estos agentes son capaces de generalizar a casi cualquier objetivo, siendo eficaces en control.
- **Reducción de threshold:** Los dos pasos anteriores se realizan con un threshold de 0.2. Para que los agentes sean más precisos, se repite el entrenamiento con un valor de 0.1. Es posible que al reducir este umbral sea necesario que el agente explore más configuraciones durante el entrenamiento (aunque no ha sido el caso en este trabajo).
- **Cambio de algoritmo:** Se emplea SAC en vez de PPO para ver cómo evoluciona el proceso de entrenamiento. Esto se lleva a cabo con el threshold de 0.1 y con varios objetivos.

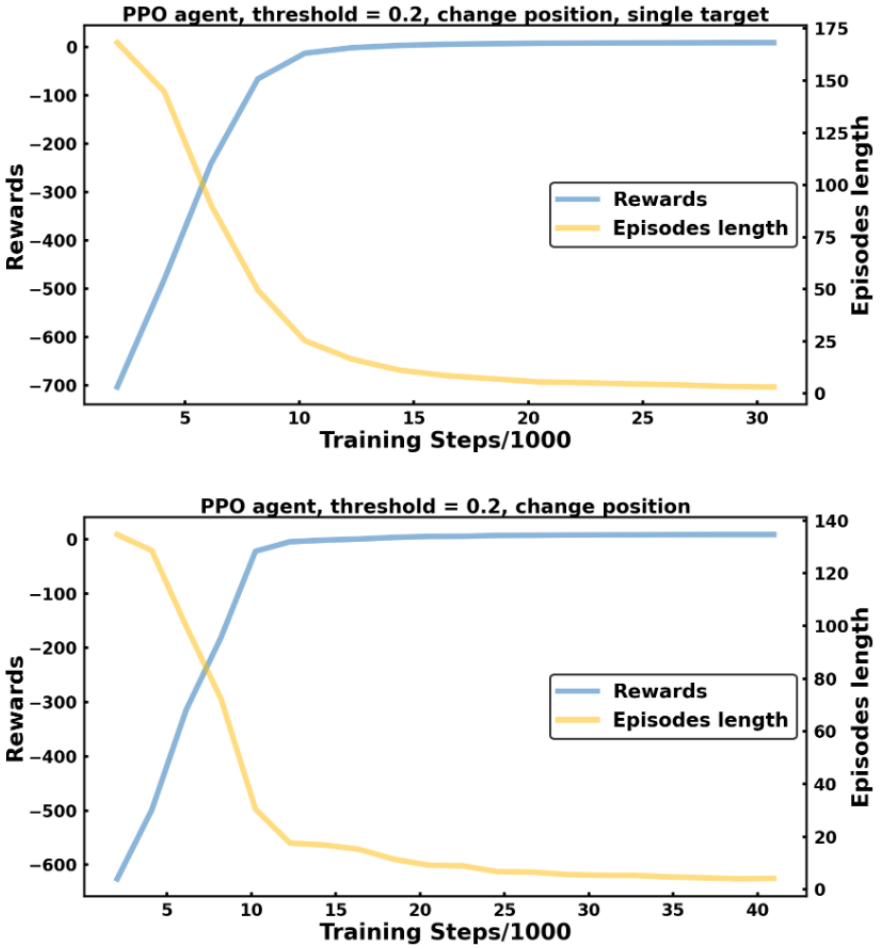
4.5.1. Agentes de control de la posición del haz

En primer lugar, se entrenan los agentes de control de la posición debido a que es un problema lineal (los efectos de los ángulos son lineales en la posición del haz) con dos actuadores, mientras que los efectos de modificar los cuadrupolos son no

[7] En este sentido, se trata de un trabajo híbrido entre metodología y desarrollo de código en Python.

lineales. Los resultados de las curvas de aprendizaje [8] se muestran en las siguientes gráficas.

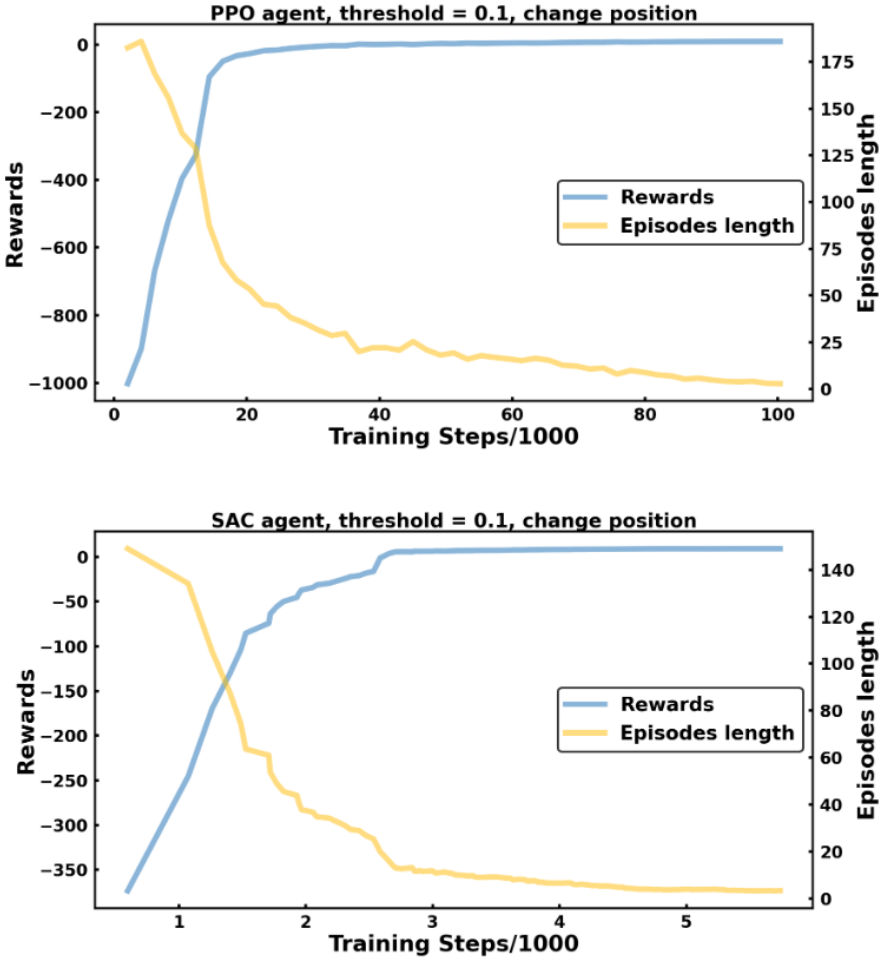
Figura 4.15: Curvas de aprendizaje de los agentes de control de posición del haz con el algoritmo PPO (threshold 0.2). Arriba: entrenamiento con una configuración. Abajo: Entrenamiento con ocho configuraciones distintas. Azul: curva de recompensas. Naranja: curva de duración de episodios. Fuente: elaboración propia.



En la Figura 4.15, se observa una comparación del entrenamiento de dos agentes. Las curvas enseñadas son útiles para evaluar el aprendizaje porque representan la evolución de los promedios de recompensas y longitud de los episodios durante las iteraciones del algoritmo. En todos los casos, las recompensas mejoran mientras que la longitud de los episodios decrece, ya que el algoritmo busca constante soluciones que maximicen la recompensa planteada.

[8] Obtenidas de los logs de Tensorboard.

Figura 4.16: Curvas de aprendizaje de los agentes de control de posición del haz con dos algoritmos y un threshold de 0.1. Arriba: entrenamiento con el algoritmo PPO. Abajo: Entrenamiento con el algoritmo SAC. Azul: curva de recompensas. Naranja: curva de duración de episodios. Fuente: elaboración propia.

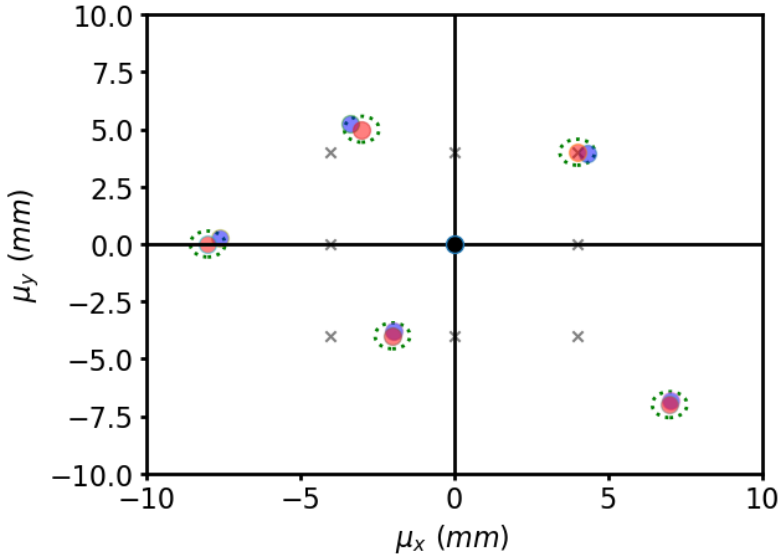


En cuanto al número de objetivos, el primer agente aprende la solución de acciones óptima utilizando solamente una configuración de entrenamiento. No obstante, este agente no es capaz de generalizar a más configuraciones ya que la red neuronal solo ha tenido un valor de entrada en el aprendizaje (la entrada *Beam Target* de la Figura 4.14). Este tipo de entrenamiento es interesante para optimizar los distintos parámetros de secciones enteras del acelerador. Se puede puede emplear en lugar de, por ejemplo, algoritmos evolutivos.

En la Figura 4.16, se muestra el proceso de aprendizaje para otros dos agentes, uno con el algoritmo PPO y otro con el algoritmo SAC. Ambos encuentran la solución

óptima del problema y dan lugar a políticas similares, pero SAC converge más rápido. No obstante, cabe mencionar que las gráficas no pueden ser comparadas ya que los tiempos en las iteraciones de los algoritmos son distintos.

Figura 4.17: Resultados de la evaluación del agente de control de la posición (threshold de 0.1, 8 configuraciones de entrenamiento). Azul: posiciones alcanzadas. Rojo: posiciones objetivo. Verde: thresholds. Cruces: configuraciones de entrenamiento. Fuente: elaboración propia.

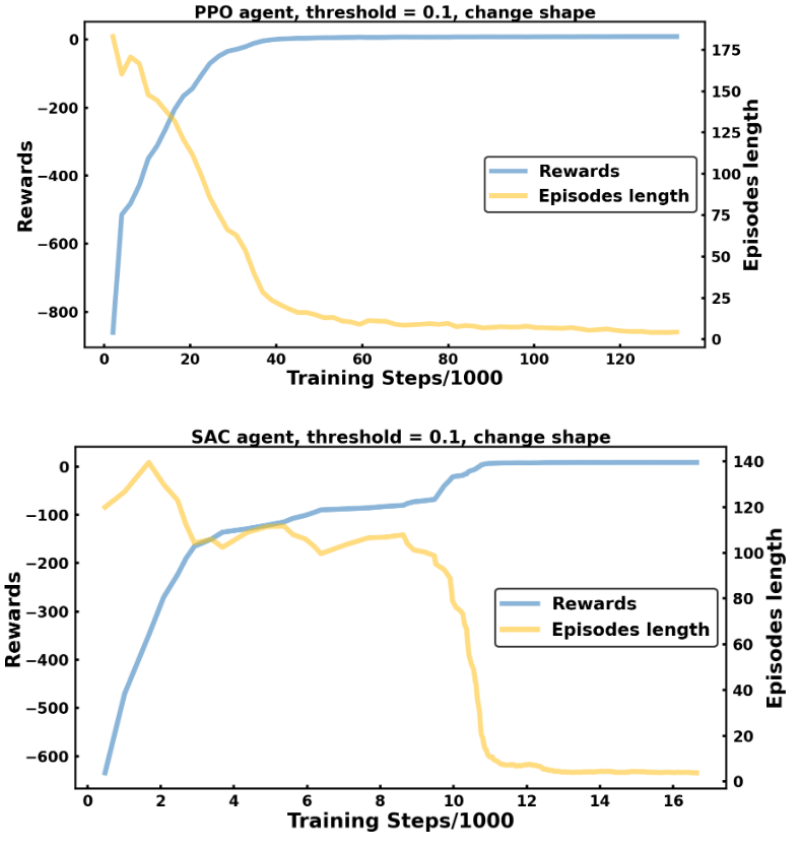


El agente final de control de posición (entrenado con varios objetivos, el algoritmo PPO y un threshold de 0.1), visto en la parte superior de la Figura 4.16, puede controlar el haz para llegar a cualquier estado. La evaluación de dicho agente se puede ver en el espacio de configuraciones de la Figura 4.17, donde se muestran las posiciones promedio de las partículas del haz. Los puntos evaluados han sido elegidos de tal manera que cuatro de ellos no pertenezcan al conjunto de entrenamiento. En base a estos resultados, se puede concluir que el agente es capaz de controlar de manera autónoma la posición del haz al final del acelerador incluso para configuraciones que no han sido vistas durante el proceso de aprendizaje.

4.5.2. Agentes de control de la geometría del haz

Los agentes de control de la geometría del haz aprenden a modificar los valores de la fuerza de las lentes de los cuadrupolos con el objetivo de deformarlo hasta lograr una configuración específica. Para ello se han entrenado dos agentes (PPO y SAC) con los mismos cuatro objetivos. Las curvas de aprendizaje se pueden apreciar en la Figura 4.18. Al tratarse de un problema más complejo (el espacio de acciones es de dimensión superior y los cambios en los cuadrupolos afectan a todas las partículas), el tiempo de entrenamiento aumenta con respecto al caso anterior. El algoritmo SAC también tiene más dificultados en encontrar el mínimo global, como se puede ver en las líneas más irregulares.

Figura 4.18: Curvas de aprendizaje de los agentes de control de la geometría del haz con dos algoritmos y un threshold de 0.1. Arriba: entrenamiento con el algoritmo PPO. Abajo: Entrenamiento con el algoritmo SAC. Azul: curva de recompensas. Naranja: curva de duración de episodios. Fuente: elaboración propia.



A diferencia del control de la posición, el espacio de posibles valores (Q_j) no tiene so-

lución en todos los rangos de (σ_x, σ_y), como se aprecia en el Apéndice C. Por lo tanto, hay que tomar en cuenta estos límites para elegir configuraciones de entrenamiento y evaluación. Los resultados de la evaluación fuera del conjunto de entrenamiento se pueden ver en la Figura 4.19, donde la política entrenada alcanza siempre el objetivo en el espacio de configuraciones con una tolerancia de 0.1. Por otra parte, la Figura 4.20 muestra las distribuciones de electrones alcanzadas al final del módulo del acelerador. Todos estos resultados demuestran que es posible controlar de manera autónoma, con ayuda de un experto en el dominio, haces de partículas cargadas dentro de un acelerador lineal.

Figura 4.19: Resultados de la evaluación del agente de control de la geometría del haz (threshold de 0.1, 4 configuraciones de entrenamiento). Azul: configuraciones alcanzadas. Rojo: configuraciones objetivo. Verde: thresholds. Cruces: puntos de entrenamiento. Fuente: elaboración propia.

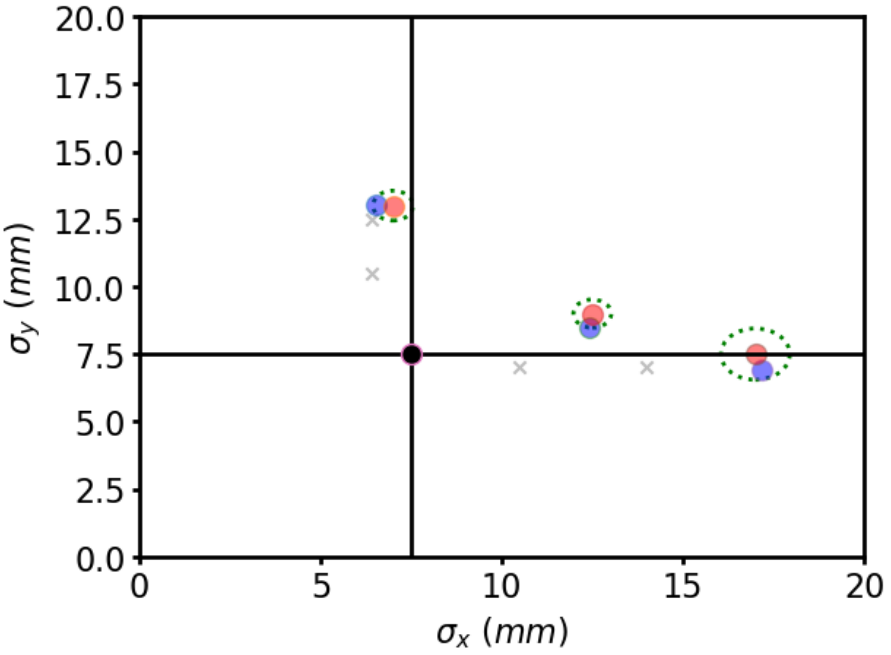
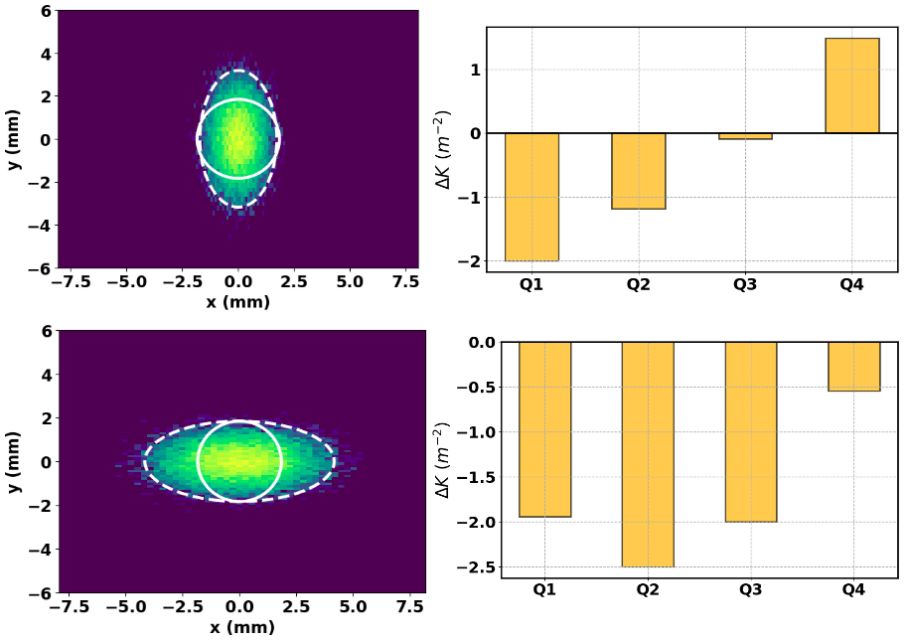


Figura 4.20: Resultados de evaluación para dos configuraciones. Izquierda: distribución de electrones alcanzada (verde), configuración inicial (línea blanca sólida) y configuración objetivo (línea blanca punteada). Derecha: estados finales de los cuadrupolos. Fuente: elaboración propia.



Para el módulo final, el controlador inteligente asociado tomará como entradas los parámetros objetivo del problema, definidos por el experto, y las medidas de los sensores obtenidas en cada paso de un conjunto de partículas. Una vez que los objetivos se han establecido, los agentes se encargan de alcanzarlos mediante la selección de acciones óptimas dadas por la política de la Figura 4.14. Estos controladores simplifican el diseño de sistemas tradicionales y, al mismo tiempo, aumentan su eficiencia y flexibilidad, ya que las metas se logran sin requerir demasiado trabajo manual.

Capítulo 5

Conclusiones y trabajo futuro

La extracción de energía de fusión nuclear se enfrenta a desafíos significativos, entre los cuales destaca la necesidad de estudiar los efectos de la irradiación de neutrones en los materiales del reactor. Para abordar este problema, el proyecto IFMIF-DONES ha surgido con el propósito de construir una instalación capaz de generar un espectro de neutrones similar al producido en reacciones nucleares, logrado mediante la colisión de haces de deuterones y litio en un acelerador lineal. Además, el proyecto DONES-FLUX se dedica a investigar la optimización de los flujos dentro de IFMIF-DONES, con el objetivo de crear una instalación inteligente y eficiente que mejore la gestión y el control de todos sus componentes.

En este trabajo, se ha investigado el potencial de algoritmos DRL para la optimización y el control de una sección experimental del acelerador. Concretamente, se ha desarrollado y validado una metodología para tratar el control de un haz de partículas cargadas utilizando el marco de los procesos de Markov y técnicas de DRL. Para ello, se han definido matemáticamente todas las componentes del problema y, tras un análisis exhaustivo, se han seleccionado las herramientas necesarias para llevar a cabo su implementación. El sistema multi-agente entrenado aprende a controlar de manera independiente la posición y la geometría del haz, mediante la selección de acciones óptimas con imanes correctores y cuadrupolos, respectivamente. Para las tolerancias elegidas, los resultados obtenidos han sido muy positivos ya que se alcanza cualquier configuración desde el estado inicial de manera general. Esto de-

muestra la viabilidad de este paradigma de la IA en control autónomo (guiado por un experto) de aceleradores lineales.

En cuanto a trabajo futuro, los agentes aún se pueden mejorar de varias maneras. En primer lugar, se intentará reducir el threshold para alcanzar el objetivo y aumentar así la precisión de los resultados. También se aumentará el número de objetivos de entrenamiento y se añadirán estados iniciales aleatorios para que los agentes aprendan a saltar de un estado arbitrario a otro sin tener que pasar por la configuración inicial. Esto conlleva una ampliación del espacio de estados y un incremento en la complejidad del proceso de entrenamiento. Por este motivo también se llevará a cabo un estudio sistemático de los hiperparámetros para mejorar la eficiencia y convergencia de los algoritmos. En este ámbito, se propone el uso de herramientas como Optuna (Akiba et al., 2019), las cuales emplean técnicas de muestreo y podado para encontrar los valores óptimos de entrenamiento. Asimismo, se planea también entrenar agentes con el framework Ray RLlib, reconocido por su eficiencia y escalabilidad en comparación con SB3.

También es importante señalar que todo este estudio ha sido realizado exclusivamente con electrones, ya que Ocelot no es compatible con otras partículas en este momento. En la próxima fase del proyecto, se tiene previsto emplear otros simuladores compatibles con deuterones, como OPAL (Adelmann et al., 2019) o TraceWin (Uriot and Pichoff, 2015). También es relevante mencionar que no se han incluido fenómenos físicos de segundo orden en la simulación, como campos de estela (del inglés *Wake fields*) (Ferrario et al., 2016). Dichos fenómenos, además de aumentar la complejidad del problema, incrementan también el tiempo de computación en órdenes de magnitud, haciendo que el entrenamiento online no sea viable.

Para abordar el problema de los tiempos de ejecución de los simuladores, se consideran dos soluciones: el uso de Deep Learning Surrogate Models (DLSM) (Haghighat et al., 2021)o las ya mencionadas técnicas de entrenamiento *offline*. Cabe destacar que, en relación a la primera alternativa y a este trabajo, la empresa HI-Iberia (líder del proyecto DONES-FLUX) está investigando varios modelos subrogados para su aplicación en el acelerador de IFMIF-DONES. Un ejemplo es el control de la presión

en la cámara de vacío que alberga el circuito de litio mediante la inyección de argón al final de la línea (Torregrosa-Martin et al., 2023).

Finalmente, futuros módulos experimentales incluirán actuadores para modificar la energía del haz de partículas, además de la posición y la geometría. De esta manera, se entrenarán agentes con el objetivo de controlar las cavidades de radiofrecuencia. También se implementarán imanes que permitan geometrías más complicadas, como las vistas en la Figura 4.2, donde la forma que adquiere el haz antes de la colisión es rectangular. En las etapas finales del proyecto, se contempla la posibilidad de implementar la política de acciones óptimas en hardware real y probarla en un módulo experimental real del acelerador.

Agradecimientos

Quiero expresar mi sincero agradecimiento al Programa Europeo de Fusión Nuclear por su fundamental participación en el desarrollo de ITER y DEMO, contribuyendo al avance hacia un futuro sostenible en la energía de fusión. También, agradezco al Centro para el Desarrollo Tecnológico Industrial (CDTI) por su generosa financiación de los proyectos IFMIF-DONES y DONES-FLUX. Además, deseo reconocer y agradecer a las diversas instituciones que han desempeñado un papel crucial en este trabajo, incluyendo el consorcio IFMIF-DONES, así como las universidades de Granada, Extremadura y Sevilla, y las valiosas colaboraciones de las empresas DEM, EDAIR, LEADING, SUPRASYS y ASE Optics. También quiero expresar mi profundo agradecimiento al equipo de HI-Iberia por su dedicación y orientación durante mi aprendizaje, ya que sus conocimientos y esfuerzos han sido fundamentales para obtener los resultados expuestos en este trabajo. Finalmente, no puedo dejar de agradecer a UNIR por hacer posible esta tésis y a mi director por su apoyo y comentarios.

Apéndice A

Variables y rangos

Tabla A.1: Rangos de todas las variables del espacio de estados y del espacio de acciones. Incluyen factores para tener las escalas en valores razonables para los frameworks DRL (ya que se utilizan de entradas para redes neuronales).

Variable	Mínimo	Máximo
$\theta_{Cj} \times 10^3$ (rad)	-10	10
$\Delta\theta_{Cj} \times 10^3$ (rad)	-1	1
$k_{1,Qj}$ $\times 2$ (m^{-2})	-5	5
$\Delta k_{1,Qj} \times 2$ (m^{-2})	-1	1
μ_x (mm)	-20	20
μ_y (mm)	-20	20
σ_x (mm)	0	25
σ_y (mm)	0	25
(x, y) (mm)	-20	20

Apéndice B

Valores de los hiperparámetros utilizados

Tabla B.1: Hiperparámetros utilizados por el algoritmo PPO (SB3).

Hiperparámetro	**Valor**
learning_rate	0.0003
n_steps	2048
batch_size	64
n_epochs	10
gamma	0.99
gae_lambda	0.95
clip_range	0.2
normalize_advantage	True
ent_coef	0.0
vf_coef	0.5
max_grad_norm	0.5

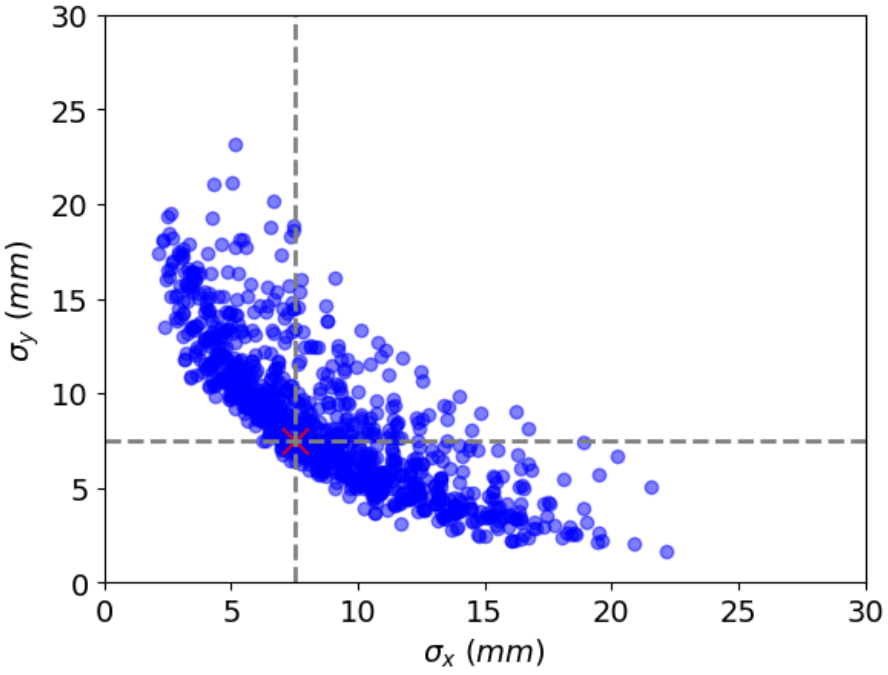
Tabla B.2: Hiperparámetros utilizados por el algoritmo SAC (SB3).

Hiperparámetro	**Valor**
learning_rate	0.0003
buffer_size	1000000
learning_starts	100
batch_size	256
tau	0.005
gamma	0.99
train_freq	1
gradient_steps	1
action_noise	None
replay_buffer_class	None
replay_buffer_kwargs	None
optimize_memory_usage	False
ent_coef	'auto'
target_update_interval	1
target_entropy	'auto'

Apéndice C

Espacio de soluciones permitidas para el caso de los cuadrupolos

Figura C.1: Muestreo de posibles soluciones con distintas configuraciones de cuadrupolos. Fuente: elaboración propia.



Bibliografía

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. (2018). Maximum a posteriori policy optimisation.

Adelmann, A., Calvo, P., Frey, M., Gsell, A., Locans, U., Metzger-Kraus, C., Neveu, N., Rogers, C., Russell, S., Sheehy, S., Snuverink, J., and Winklehner, D. (2019). Opal a versatile tool for charged particle accelerator simulations.

Agapov, I., Geloni, G., Tomin, S., and Zagorodnov, I. (2014). Ocelot: A software framework for synchrotron light source and fel studies. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 768:151–156.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework.

Amari, S. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4):185–196.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846.

Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684.

Bellman, R. (1958). Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228–239.

Bellman, R. E. (2010). *Dynamic Programming*. Princeton University Press, Princeton.

Bernardi, D., Ibarra, A., Arbeiter, F., Arranz, F., Cappelli, M., Cara, P., Castellanos, J., Dzitko, H., García, A., Gutiérrez, J., Królas, W., Martin-Fuertes, F., Micciché, G., Muñoz Roldan, A., Nitti, F., Pinna, T., Podadera, I., Pons, J., Qiu, Y., and Román, R. (2022). The ifmif-dones project: Design status and main achievements within the eurofusion fp8 work programme. *Journal of Fusion Energy*, 41:24.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680.

Boulila, W., Driss, M., Al-Sarem, M., Saeed, F., and Krichen, M. (2021). Weight initialization techniques for deep learning algorithms in remote sensing: Recent trends and future perspectives. *CoRR*, abs/2102.07004.

Cesa-Bianchi, N., Gentile, C., Lugosi, G., and Neu, G. (2017). Boltzmann exploration done right.

Chen, J., Yuan, B., and Tomizuka, M. (2019). Model-free deep reinforcement learning for urban autonomous driving. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2765–2771.

Ciattaglia, S., Federici, G., Barucca, L., Lampasi, A., Minucci, S., and Moscato, I. (2017). The european demo fusion reactor: Design status and challenges from balance of plant point of view. In *2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC / ICPS Europe)*, pages 1–6.

Clifton, J. and Laber, E. (2020). Q-learning: Theory and applications. *Annual Review of Statistics and Its Application*, 7(1):279–301.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., Casas, D., Donner, C., Fritz, L., Galperti, C., Huber, A., Keeling, J., Tsimpoukelli, M., Kay, J., Merle, A., Moret, J.-M., and Riedmiller, M. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602:414–419.

Efron, B. and Tibshirani, R. (1993). An Introduction to the Bootstrap.

Eichler, A., Burkart, F., Kaiser, J., Kuropka, W., Stein, O., Xu, C., Bründermann, E., and Santamaria Garcia, A. (2021). First steps toward an autonomous accelerator, a common project between desy and kit. In *12th International Particle Accelerator Conference : virtual edition, May 24th-28th, 2021, Brazil : proceedings volume / IPAC2021. Ed.: R. Picoreti*, pages 2182–2185. JACoW Publishing. 54.11.11; LK 01.

Federici, G., Bachmann, C., Barucca, L., Baylard, C., Biel, W., Boccaccini, L., Bustreo, C., Ciattaglia, S., Cismondi, F., Corato, V., Day, C., Diegele, E., Franke, T., Gaio, E., Gliss, C., Haertl, T., Ibarra, A., Holden, J., Keech, G., Kembleton, R., Loving, A., Maviglia, F., Morris, J., Meszaros, B., Moscato, I., Pintsuk, G., Siccinio, M., Taylor, N., Tran, M., Vorpahl, C., Walden, H., and You, J. (2019). Overview of the demo staged design approach in europe. *Nuclear Fusion*, 59(6):066013.

Ferrario, M., Migliorati, M., and Palumbo, L. (2016). Wakefields and instabilities in linear accelerators. *CERN*.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods.

Geron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2nd edition.

Graesser, L. and Keng, W. (2019). *Foundations of Deep Reinforcement Learning*. Addison-Wesley Data & Analytics Series. Pearson Education.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

Haghighat, E., Raissi, M., Moure, A., Gomez, H., and Juanes, R. (2021). A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 379:113741.

Hoffman, M. W., Shahriari, B., Aslanides, J., Barth-Maron, G., Momchev, N., Sinopalnikov, D., Stańczyk, P., Ramos, S., Raichuk, A., Vincent, D., Hussenot, L., Dadashi, R., Dulac-Arnold, G., Orsini, M., Jacq, A., Ferret, J., Vieillard, N., Ghasemipour, S. K. S., Girgin, S., Pietquin, O., Behbahani, F., Norman, T., Abdolmaleki, A., Cassirer, A., Yang, F., Baumli, K., Henderson, S., Friesen, A., Haroun, R., Novikov, A., Colmenarejo, S. G., Cabi, S., Gulcehre, C., Paine, T. L., Srinivasan, S., Cowie, A., Wang, Z., Piot, B., and de Freitas, N. (2022). Acme: A research framework for distributed reinforcement learning.

Holzer, B. J. (2014). Introduction to transverse beam dynamics.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

Ibarra, A. (2019). Ifmif-dones. In *6th DEMO Workshop*, Moscow, Russia.

Ibarra, A., Arbeiter, F., Bernardi, D., Krolas, W., Cappelli, M., Fischer, U., Heidinger, R., Martin-Fuertes, F., Micciché, G., Cervantes, A., Nitti, F., Pinna, T., Aiello, A., Bazin, N., Chauvin, N., Chel, S., Devanz, G., Gordeev, S., Regidor, D.,

and Schwab, F. (2019). The european approach to the fusion-like neutron source: the ifmif-dones project. *Nuclear Fusion*, 59.

Jokanović, V. (2022). *Artificial Intelligence*, chapter 1. Taylor and Francis Group.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134.

Kain, V., Hirlander, S., Goddard, B., Velotti, F. M., Della Porta, G. Z., Bruchon, N., and Valentino, G. (2020). Sample-efficient reinforcement learning for cern accelerator control. *Phys. Rev. Accel. Beams*, 23:124801.

Kaiser, J., Stein, O., and Eichler, A. (2022). Learning-based optimisation of particle accelerators under partial observability without real-world training. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 10575–10585. PMLR.

Kroese, D., Brereton, T., Taimre, T., and Botev, Z. (2014). Why the monte carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems.

Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I. (2018). Rllib: Abstractions for distributed reinforcement learning.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2019). Continuous control with deep reinforcement learning.

Lin, L. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

Mohammadi, M. and Al-Fuqaha, A. (2018). Enabling cognitive smart cities using big data and machine learning: Approaches and challenges. *IEEE Communications Magazine*, 56(2):94–101.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. (2017). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *CoRR*, abs/1708.02596.

Nagaitsev, S., Huang, Z., Power, J., Vay, J. L., Piot, P., Spentzouris, L., Rosenzweig, J., Cai, Y., Cousineau, S., Conde, M., Hogan, M., Valishev, A., Minty, M., Zolkin, T., Huang, X., Shiltsev, V., Seeman, J., Byrd, J., Hao, Y., Dunham, B., Carlsten, B., Seryi, A., and Patterson, R. (2021). Accelerator and beam physics research goals and opportunities.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. (2019). Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113.

Pang, X., Thulasidasan, S., and Rybarcyk, L. (2020). Autonomous control of a particle accelerator using deep reinforcement learning.

Piché, A., Thomas, V., Marino, J., Marconi, G. M., Pal, C., and Khan, M. E. (2022). Beyond target networks: Improving deep q-learning with functional regularization.

Podadera, I., Arbeiter, F., Bazin, N., Bellan, L., Chauvin, N., Chel, S., Duglue, G., Dumas, J., Dzitko, H., Fagotti, E., Grabowski, W., Hauer, V., Ibarra, A., Jaksic, M., Jimenez-Rey, D., Królas, W., López, R., Mollá, J., Muñoz, A., Nomen, O., Oliver, C., Palmieri, A., Pisent, A., Plouin, J., Prieto, C., Regidor, D., Sanmartí, M., Esteban, F. S., Seguí, L., Singh, B., Sánchez-Herranz, D., Tadic, T., Varela, R., Wysocka-Rabin, A., and de la Morena, C. (2021a). The Accelerator System of IFMIF-DONES Multi-MW Facility. In *Proc. IPAC'21*, number 12 in International Particle Accelerator Conference, pages 1910–1913. JACoW Publishing, Geneva, Switzerland. https://doi.org/10.18429/JACoW-IPAC2021-TUPAB211.

Podadera, I. et al. (2021b). The Accelerator System of IFMIF-DONES Multi-MW Facility. In *12th International Particle Accelerator Conference*.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Rafiee, A. and Khalilpour, K. R. (2019). Chapter 11 - renewable hybridization of oil and gas supply chains. In Khalilpour, K. R., editor, *Polygeneration with Polystorage for Chemical and Energy Hubs*, pages 331–372. Academic Press.

Rapp, J. (2017). The challenges of plasma material interactions in nuclear fusion devices and potential solutions. *Fusion Science and Technology*, 72(3):211–221.

Rawson, M. G. and Balan, R. (2021). Convergence guarantees for deep epsilon greedy policy learning. *CoRR*, abs/2112.03376.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408.

Rummery, G. and Niranjan, M. (1994). On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2018). High-dimensional continuous control using generalized advantage estimation.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

Seno, T. and Imai, M. (2022). d3rlpy: An offline deep reinforcement learning library.

Shin, S. S., Chai, J.-S., and Ghergherehchi, M. (2018). Using Deep Reinforcement Learning for Designing Sub-Relativistic Electron Linac. In *Proc. 9th International Particle Accelerator Conference (IPAC'18), Vancouver, BC, Canada, April 29-May 4, 2018*, number 9 in International Particle Accelerator Conference, pages 4720–4722, Geneva, Switzerland. JACoW Publishing. https://doi.org/10.18429/JACoW-IPAC2018-THPML032.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

St. John, J., Herwig, C., Kafkes, D., Mitrevski, J., Pellico, W. A., Perdue, G. N., Quintero-Parra, A., Schupbach, B. A., Seiya, K., Tran, N., Schram, M., Duarte, J. M., Huang, Y., and Keller, R. (2021). Real-time artificial intelligence for accelerator control: A study at the fermilab booster. *Phys. Rev. Accel. Beams*, 24:104601.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3(1):9–44.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68.

Timofeyuk, N. and Johnson, R. (2020). Theory of deuteron stripping and pick-up reactions for nuclear structure studies. *Progress in Particle and Nuclear Physics*, 111:103738.

Torregrosa-Martin, C., Ibarra, A., Aguilar, J., Ambi, F., Arranz, F., Arbeiter, F., Bagnasco, A., Becerril, S., Bernardi, D., Bolzon, B., Botta, E., Brenneis, B., Cappelli, M., Cara, P., Castellanos, J., Cosic, D., De la Morena, C., Diez, A., Ericsson, G., García, A., García, M., Garcinuño, B., Gutiérrez, J., Gutiérrez, V., Jimenez-Rey, D., Dezsi, T., Ferreira, M. J., Fiore, S., Krolas, W., Lorenzo, R., Luque, M., Maciá, L., Marroncle, J., Martin-Fuertes, F., Marugán, J., Maestre, J., Meléndez, C., Micciché, G., Mollá, J., Moreno, A., Nitti, F., Núñez, C., Ogando, F., Pinna, T., Oliver, C., Podadera, I., Prieto, C., Prokopowicz, R., Qiu, Y., Rapisarda, D., Regidor, D., Rodríguez, E., Sabogal, A., Sánchez-Herranz, D., Sanmarti, M., Seguí, L., Serikov, A., Tadić, T., Talarowska, A., Wiacek, U., Weber, M., Valenzuela, J., and Zsakai, A. (2023). Overview of ifmif-dones diagnostics: Requirements and techniques. *Fusion Engineering and Design*, 191:113556.

Toschi, R. (1997). Nuclear fusion, an energy source. *Fusion Engineering and Design*, 36(1):1–8.

Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Shen, A. T. J., and Younis, O. G. (2023). Gymnasium.

Tsitsiklis, J. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690.

Uriot, D. and Pichoff, N. (2015). Status of tracewin code. In *TraceWin*.

van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning.

Vieillard, N., Pietquin, O., and Geist, M. (2020). Deep conservative policy iteration.

Wan, B., Liang, Y., Gong, X., Xiang, N., Xu, G., Sun, Y., Wang, L., Qian, J., Liu, H., Zeng, L., Zhang, L., Xinjun, Z., Ding, B., Zang, Q., Lyu, B., Garofalo, A.,

Ekedahl, A., Li, M., Ding, F., and Xia, T. (2019). Recent advances in east physics experiments in support of steady-state operation for iter and cfetr. *Nuclear Fusion*, 59.

Weng, J., Chen, H., Yan, D., You, K., Duburcq, A., Zhang, M., Su, Y., Su, H., and Zhu, J. (2022). Tianshou: a highly modularized deep reinforcement learning library.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Wu, X., Chen, H., Wang, J., Troiano, L., Loia, V., and Fujita, H. (2020). Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*, 538:142–158.

Wu, Y. and Şahin, S. (2018). 3.13 fusion energy production. In Dincer, I., editor, *Comprehensive Energy Systems*, pages 538–589. Elsevier, Oxford.

Yarotsky, D. (2018). Universal approximations of invariant maps by neural networks. *CoRR*, abs/1804.10306.

Deep Reinforcement Learning aplicado a un módulo experimental del acelerador de IFMIF-DONES

Galo Gallardo Romero

Universidad Internacional de la Rioja, Logroño (España)

06-09-2023

RESUMEN

Uno de los desafíos más importantes de las futuras centrales de fusión nuclear es el estudio de los efectos de la irradiación de neutrones sobre los materiales del reactor. Con el objetivo de resolver este problema, el proyecto IFMIF-DONES busca construir una instalación para generar un espectro de neutrones similar al de una reacción de fusión nuclear mediante la colisión de un haz de deuterones y un circuito de litio. En este trabajo, enmarcado dentro del proyecto DONES-FLUX, se investiga la aplicación de la Inteligencia Artificial al control autónomo del acelerador lineal de IFMIF-DONES. En concreto, se desarrolla y valida la metodología para implementar algoritmos de Deep Reinforcement Learning a un módulo experimental del acelerador, el cual tiene como objetivo modificar las configuraciones del haz mediante imanes actuadores. Los resultados obtenidos en esta primera iteración del proyecto son positivos y respaldan este enfoque, ya que los agentes entrenados alcanzan de manera óptima cualquier configuración geométrica dentro de las tolerancias establecidas.

PALABRAS CLAVE

Reinforcement Learning, Inteligencia Artificial, aceleradores lineales, fusión nuclear, control autónomo

I. INTRODUCCIÓN

La comunidad científica ha invertido un esfuerzo considerable en la investigación del campo de la fusión nuclear, con el propósito de descubrir nuevas fuentes de energía que contribuyan a un futuro sostenible [1]. No obstante, la construcción de centrales de fusión nuclear se enfrenta a desafíos importantes [2]. Uno de ellos es el estudio de los efectos de la irradiación de neutrones sobre los materiales del reactor, puesto que estas partículas pueden causar daños estructurales. Para abordar este problema, surge el proyecto IFMIF-DONES (The International Fusion Materials Facility-DEMO-Oriented Neutron Source) [3], con el objetivo de producir un flujo de neutrones con un espectro similar al de una reacción de fusión.

La futura instalación de IFMIF-DONES consta de un acelerador lineal que se encarga de producir, transportar y configurar un haz de deuterones para la colisión con un blanco de litio bajo unas condiciones específicas [4]. De las reacciones de *stripping* [5] producidas, se genera un flujo de neutrones que cubre el rango espectral de una reacción de fusión nuclear Deuterio-Tritio. Dicho flujo de neutrones interactúa con distintas muestras de materiales que son irradiadas en la cámara de pruebas.

Por otra parte, el proyecto DONES-FLUX (enmarcado dentro de IFMIF-DONES), tiene el objetivo de investigar la optimización de las

instalaciones mediante el uso de técnicas de Inteligencia Artificial (IA). Uno de los campos de más éxito de la IA es el Deep Reinforcement Learning (DRL), empleado para resolver problemas de tomas de decisiones secuenciales [6]. En este paradigma, un agente interactúa con un entorno (normalmente simulado) que le proporciona *feedback* en forma de recompensas o castigos dependiendo de sus acciones. De esta manera, dicho agente aprende una forma de actuar óptima maximizando las recompensas obtenidas.

Este enfoque está teniendo un impacto significativo en diversas áreas, como la robótica [7], los juegos de mesa [8], los videojuegos [9] o los sistemas de control. Los controladores basados en DRL pueden mejorar el desarrollo y la eficiencia de los sistemas tradicionales, permitiendo que los agentes alcancen objetivos establecidos por expertos en lugar de llevar a cabo ajustes manuales de manera constante. El estudio más importante en este campo es el del control magnético de plasmas en tiempo real dentro de un tokamak, llevado a cabo por investigadores de Deep Mind [10].

En este trabajo, se estudia la aplicación de los algoritmos DRL al control autónomo de un módulo experimental[1] del acelerador lineal de IFMIF-DONES. En la Sección II, se resume el funcionamiento de las técnicas DRL Estado del Arte empleadas. En la Sección III, se define el problema y se desarrolla la metodología para abordarlo. En la Sección IV, se exponen los resultados obtenidos de la evaluación de los agentes entrenados. En la Sección V, se discute la importancia de los resultados. Finalmente, en la Sección VI, se aportan las conclusiones a las que se ha llegado.

II. ESTADO DEL ARTE

Un problema de DRL se compone de dos elementos fundamentales: el entorno y el agente. El entorno representa el modelo del problema a resolver, con estados s_t que contienen información sobre el sistema a tiempo t. Por otro lado, el agente es un algoritmo que utiliza la información obtenida del entorno para tomar decisiones. Una vez ejecutada la acción a_t, el entorno avanza al siguiente estado s_{t+1} y una función de recompensa $r(s_t, a_t, s_{t+1})$ evalúa si el agente se acerca o aleja del objetivo [11]. Este ciclo se puede observar en la Figura 1.

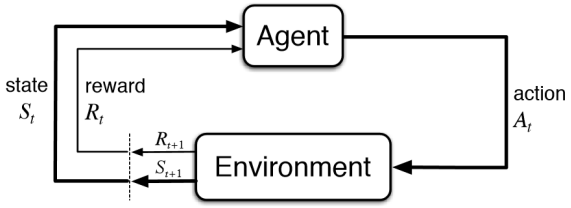


Figura 1: Bucle de aprendizaje de un algoritmo DRL, donde se muestra la interacción del agente con el entorno [11].

Estos problemas se formulan como procesos de Decisión de Markov (MDPs) [6], donde el aprendizaje se lleva a cabo mediante las ecuaciones de Bellman [12] y las experiencias obtenidas por el agente (s_t, a_t, r_t) durante los episodios (un episodio se define como una secuencia de experiencias hasta alcanzar la condición de derrota o victoria). El objetivo principal del agente es aprender una función de política $\pi(s_t)$ que determina las acciones óptimas a partir de los estados observados. Esta función, cerebro del agente, se aproxima utilizando redes neuronales con parámetros θ, lo que permite una mejor generalización a espacios de alta dimensionalidad. El proceso de aprendizaje implica actualizar de manera iterativa los parámetros de la red (mediante *gradient descent*) hasta encontrar la política óptima.

Uno de los algoritmos que está actualmente siendo utilizado por sus resultados es el de Proximal Policy Optimization (PPO) [13], el cual opera con espacios de estados y acciones continuos. Se trata de un algoritmo avanzado del tipo *policy gradient* que aborda el problema del colapso de rendimiento de los algoritmos actor-crítico (A2C) [14] y mejora su eficiencia. PPO logra cumplir esto haciendo uso de un objetivo subrogado que limita las modificaciones en la política para que no se aleje demasiado de la política anterior, evitando cambios drásti-

[1] No representativo de la implementación final de la instalación.

cos que podrían llevar a la inestabilidad en el aprendizaje. Otro algoritmo relevante en este contexto es el Soft Actor-Critic (SAC) [15], el cual mejora el proceso de recolección de datos debido a su eficiencia de muestreo.

III. OBJETIVOS Y METODOLOGÍA

El objetivo de este trabajo es diseñar la arquitectura de un controlador DRL para modificar la posición y la deformación de un haz de partículas cargadas en el módulo experimental definido en la Figura 2. El entramado consta de un inyector de electrones, cuadrupolos (Q_i), zonas de deriva sin interacciones (D), imanes correctores (C_j) y un conjunto de sensores de posición al final de la línea. Para resolver este problema, se plantea un sistema multi-agente donde un agente controla la posición del haz y otro la deformación. Esto es posible porque los efectos de los actuadores correspondientes son independientes.

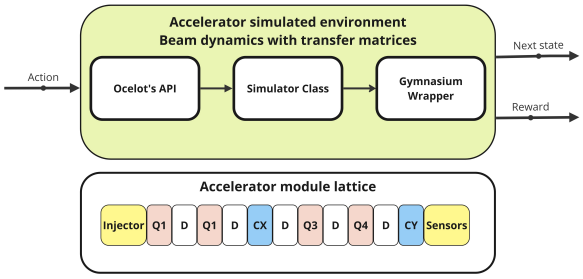


Figura 2: Arriba: implementación del entorno simulado. Abajo: entramado del acelerador y sus correspondientes elementos.

El espacio de estados, inicialmente las posiciones finales (x, y) de las 20000 partículas inyectadas, se reduce mediante *feature engineering* a los parámetros $(\mu_x, \mu_y, \sigma_x, \sigma_y)$ de una distribución gaussiana bidimensional. El agente de control de la posición toma las medias, mientras que el agente de deformación toma las desviaciones típicas. A este espacio también se incorporan los parámetros objetivo (definidos por el operador) y los estados absolutos de los imanes.

Por otra parte, las acciones vienen definidas por los incrementos de los actuadores. El agente de control de la posición modifica los ángulos de los imanes correctores en las dos coordenadas $(\Delta\theta_x, \Delta\theta_y)(rad)$, mientras que el agente de deformación modifica las fuerzas de los cuatro cuadrupolos $(\Delta k_1, \Delta k_2, \Delta k_3, \Delta k_4)(m^{-2})$. Todos estos espacios se pueden observar como las entradas y las salidas de la función política (red neuronal representada en la Figura 3).

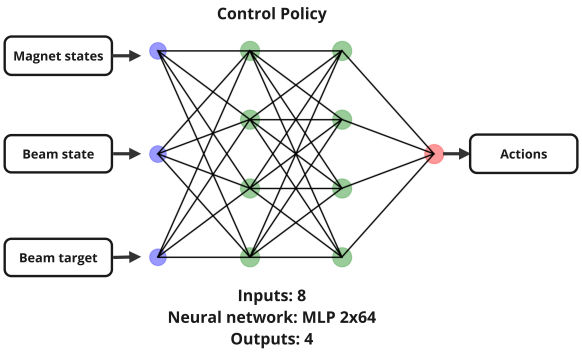


Figura 3: Red neuronal de la política de control. Se representa el espacio de estados, las capas ocultas (64x64) y la capa de salida (con dimensión igual al número de actuadores).

En cuanto a la función de recompensa, crucial en la retroalimentación necesaria para el aprendizaje del agente, se ha tomado la distancia euclídea normalizada negativa sobre el espacio de configuraciones considerando las observaciones de cada agente. Esta función asigna un valor de recompensa negativo que disminuye a medida que el agente se acerca al objetivo, motivándolo a buscar una solución óptima en términos de acciones. Cuando el agente alcanza con éxito la región objetivo, la recompensa es $+10$, indicando un logro sustancialmente mayor.

Para implementar el entorno, se ha empleado el conjunto de herramientas Ocelot[2], las cuales llevan a cabo cálculos de la dinámica de haces de electrones [16]. Mediante su API, se ha construido un simulador con acciones, estados y transiciones. Finalmente, se ha utilizado Gymnasium [17] para construir un *wrapper* que garantiza la compatibilidad del entorno con la

[2] https://github.com/ocelot-collab/ocelot

mayoría de frameworks DRL. Todo este proceso se ve resumido en la parte superior de la Figura 2.

Tras examinar la literatura, se ha escogido el algoritmo PPO debido a su popularidad en problemas de control con espacios continuos. Por otra parte, se ha llevado a cabo un estudio comparativo de los frameworks DRL más populares, donde al final se ha elegido Stable Baselines 3 [18] por su simplicidad, documentación y estabilidad. Por último, para validar la metodología, se ha llevado a cabo el entrenamiento de diversos agentes DRL, prestando especial atención a las curvas de aprendizaje a través de Tensorboard [19]. Tras el entrenamiento, dichos agentes se han evaluado en situaciones completamente nuevas.

IV. RESULTADOS

A. Entrenamiento

Se han realizado múltiples experimentos de entrenamiento, modificando algunos de los hiperparámetros de los agentes hasta alcanzar la convergencia. En esta sección, se exponen solamente los resultados obtenidos para los algoritmos PPO y SAC, empleando un threshold de 0.1, y con múltiples objetivos de entrenamiento aleatorios predefinidos para cada episodio. Las curvas de aprendizaje se pueden ver en las Figuras 5 y 6 del Apéndice A. Para todos dos agentes mostrados, cabe destacar que la recompensa promedio mejora con el tiempo y el número de acciones decrece, hasta llegar a las secuencias óptimas.

B. Evaluación

Los resultados de la evaluación se pueden ver en la Figura 7 del Apéndice A, donde se muestran los espacios de configuraciones para ambos agentes, tanto para el control de la posición como la deformación geométrica del haz. Todos los estados obtenidos entran dentro de la tolerancia establecida. Para terminar, dos de las configuraciones geométricas del agente de control de la deformación y los valores de sus cuadrupolos se exponen en la Figura 4.

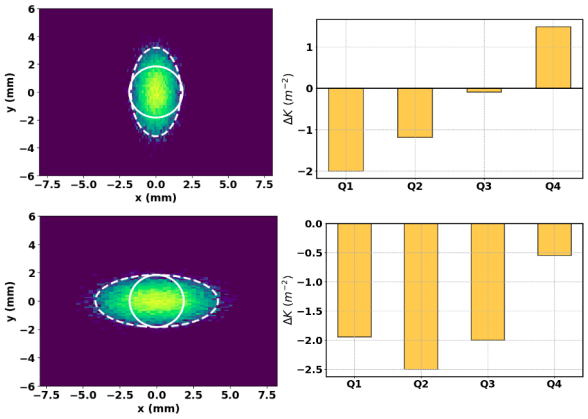


Figura 4: Resultados obtenidos para 2 configuraciones. Izquierda: distribuciones espaciales de los electrones al final del módulo acelerador. La línea blanca sólida representa la configuración inicial, mientras que la punteada es el objetivo establecido (nivel 2σ). Derecha: estados finales de los imanes actuadores.

V. DISCUSIÓN

Los resultados obtenidos por los agentes entrenados son bastante positivos (dentro de las tolerancias establecidas), ya que con un número limitado de configuraciones de entrenamiento son capaces de generalizar a cualquier otro objetivo. Además, el número de acciones que toman es mínimo para la mayoría de casos probados con los dos agentes. Estos resultados son similares a los que se pueden ver en el trabajo de Eichler A. et al. [20], aunque su entorno tiene un actuador menos y su metodología considera un único agente con los algoritmos Deep Deterministic Policy Gradient (DDPG) [21] y Twin Delayed DDPG (TD3) [22].

Por otra parte, hay que mencionar que la intervención de un experto en el campo sigue siendo indispensable para definir los objetivos precisos que se pretenden alcanzar con el haz. La experiencia y el conocimiento de la materia son fundamentales para establecer las metas y restricciones específicas que guiarán el proceso de aprendizaje. En este sentido, resulta crucial determinar las características deseadas del haz, ya que esto contribuye en gran medida al éxito de la aplicación de los algoritmos DRL.

VI. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo, se ha explorado la aplicación de algoritmos de DRL en una sección experimental del acelerador lineal de IFMIF-DONES. La metodología desarrollada ha demostrado la viabilidad y el potencial de utilizar estos agentes para controlar la posición y la geometría de un haz de electrones, ya que logran llegar a cualquier configuración establecida mediante una selección de acciones óptima.

En la siguiente iteración del proyecto, se planean distintas estrategias para mejorar los agentes. En cuanto al entrenamiento, se reducirán las tolerancias, se aumentarán los objetivos de entrenamiento, y se añadirán saltos entre configuraciones. Además de probar otros algoritmos y frameworks, se llevará también a cabo un estudio sistemático de los hiperparámetros con Optuna [23].

Finalmente, se utilizará el simulador OPAL [24] (debido a su compatibilidad con deuterones), se incluirán fenómenos físicos de segundo orden como los campos de estela [25] y se implementará la arquitectura exacta de cada módulo del acelerador de IFMIF-DONES. No obstante esto puede ocasionar tiempos de ejecución demasiado largos para cada paso temporal del ciclo de aprendizaje, por lo que también será necesario explorar soluciones Deep Learning Surrogate Models (DLSM) [26] o algoritmos DRL *offline* [27].

Referencias

[1] R Toschi. Nuclear fusion, an energy source. *Fusion Engineering and Design*, 36(1):1–8, 1997.

[2] Sergio Ciattaglia, Gianfranco Federici, Luciana Barucca, Alessandro Lampasi, Simone Minucci, and Ivo Moscato. The european demo fusion reactor: Design status and challenges from balance of plant point of view. In *2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC / ICPS Europe)*, pages 1–6, 2017.

[3] Davide Bernardi, Angel Ibarra, Frederik Arbeiter, F. Arranz, Mauro Cappelli, Philippe Cara, J. Castellanos, Hervé Dzitko, A. García, J. Gutiérrez, W. Królas, Francisco Martin-Fuertes, G. Micciché, Antonio Muñoz Roldan, F. Nitti, Tonio Pinna, Ivan Podadera, J. Pons, Yuefeng Qiu, and Raquel Román. The ifmif-dones project: Design status and main achievements within the eurofusion fp8 work programme. *Journal of Fusion Energy*, 41:24, 10 2022.

[4] I. Podadera, F. Arbeiter, N. Bazin, L. Bellan, N. Chauvin, S. Chel, G. Duglue, J. Dumas, H. Dzitko, E. Fagotti, W.C. Grabowski, V. Hauer, A. Ibarra, M. Jaksic, D. Jimenez-Rey, W. Królas, R. López, J. Mollá, A. Muñoz, O. Nomen, C. Oliver, A. Palmieri, A. Pisent, J. Plouin, C. Prieto, D. Regidor, M. Sanmartí, F.J. Saura Esteban, L. Seguí, B.K. Singh, D. Sánchez-Herranz, T. Tadic, R. Varela, A. Wysocka-Rabin, and C. de la Morena. The Accelerator System of IFMIF-DONES Multi-MW Facility. In *Proc. IPAC'21*, number 12 in International Particle Accelerator Conference, pages 1910–1913. JACoW Publishing, Geneva, Switzerland, 08 2021. https://doi.org/10.18429/JACoW-IPAC2021-TUPAB211.

[5] N.K. Timofeyuk and R.C. Johnson. Theory of deuteron stripping and pick-up reactions for nuclear structure studies. *Progress in Particle and Nuclear Physics*, 111:103738, 2020.

[6] L. Graesser and W.L. Keng. *Foundations of Deep Reinforcement Learning*. Addison-Wesley Data & Analytics Series. Pearson Education, 2019.

[7] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell,

Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019.

[8] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[9] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[10] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602:414–419, 02 2022.

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[12] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

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

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

[16] I. Agapov, G. Geloni, S. Tomin, and I. Zagorodnov. Ocelot: A software framework for synchrotron light source and fel studies. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 768:151–156, 2014.

[17] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[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] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit

Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[20] Annika Eichler, Florian Burkart, Jan Kaiser, Willi Kuropka, Oliver Stein, Chenran Xu, Erik Bründermann, and Andrea Santamaria Garcia. First steps toward an autonomous accelerator, a common project between desy and kit. In *12th International Particle Accelerator Conference : virtual edition, May 24th-28th, 2021, Brazil : proceedings volume / IPAC2021. Ed.: R. Picoreti*, pages 2182–2185. JACoW Publishing, 2021. 54.11.11; LK 01.

[21] 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, 2019.

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

[23] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

[24] Andreas Adelmann, Pedro Calvo, Matthias Frey, Achim Gsell, Uldis Locans, Christof Metzger-Kraus, Nicole Neveu, Chris Rogers, Steve Russell, Suzanne Sheehy, Jochem Snuverink, and Daniel Winklehner. Opal a versatile tool for charged particle accelerator simulations, 2019.

[25] Massimo Ferrario, Mauro Migliorati, and Luigi Palumbo. Wakefields and instabilities in linear accelerators. *CERN*, 01 2016.

[26] Ehsan Haghighat, Maziar Raissi, Adrian Moure, Hector Gomez, and Ruben Juanes. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 379:113741, 06 2021.

[27] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

A. RESULTADOS DE ENTRENAMIENTO Y EVALUACIÓN

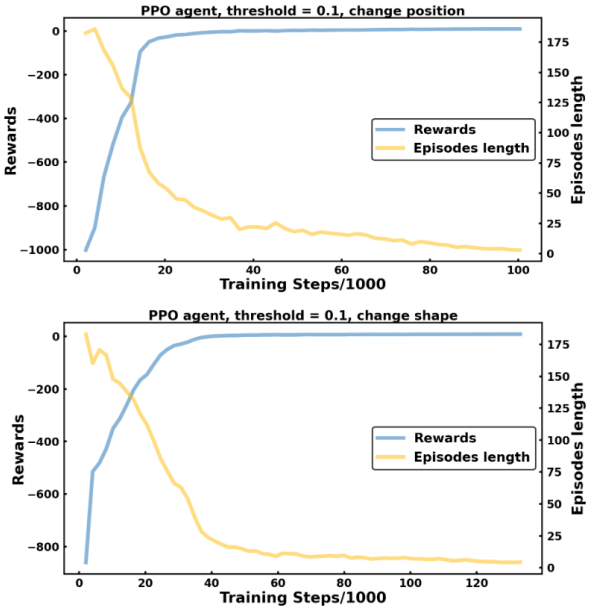


Figura 5: Curvas de aprendizaje con el algoritmo PPO. Arriba: agente de posicionamiento. Abajo: agente de deformación del haz. Líneas azules: recompensas promedio. Líneas naranjas: duración promedio de los episodios.

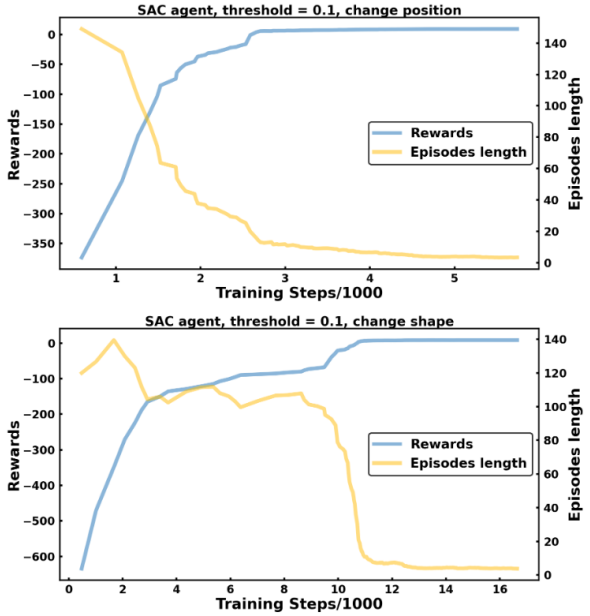


Figura 6: Curvas de aprendizaje con el algoritmo SAC. Arriba: agente de posicionamiento. Abajo: agente de deformación del haz. Líneas azules: recompensas promedio. Líneas naranjas: duración promedio de los episodios.

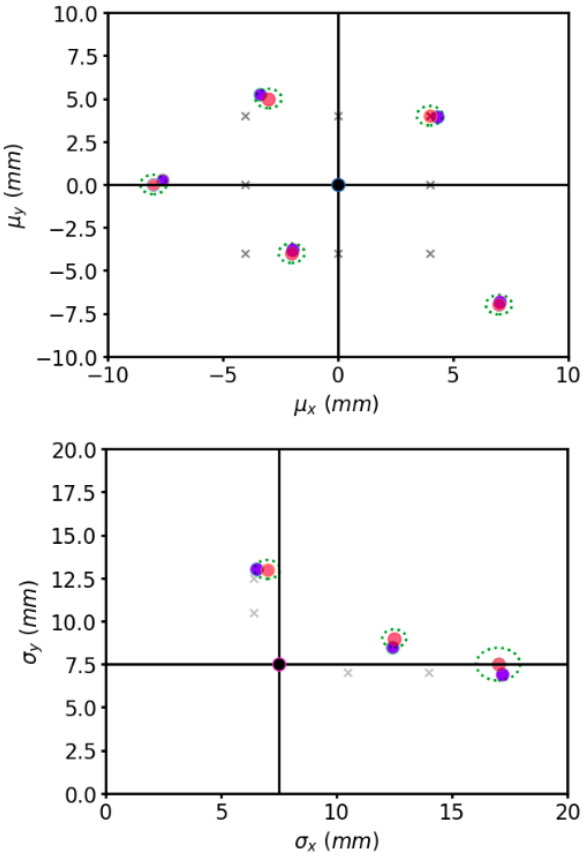


Figura 7: Resultados de la evaluación de ambos agentes (algoritmo PPO) en su correspondiente espacio de configuraciones. Arriba: agente de posición. Abajo: agente de deformación. Puntos rojos: objetivos. Puntos azules: estados alcanzados. Líneas verdes: tolerancia. Puntos cruz: objetivos de entrenamiento.