Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Control adaptable de brazos robóticos usando aprendizaje por refuerzo profundo

T E S I S

Que para optar por el grado de

Maestro en Ciencia e Ingeniería de la Computación

P R E S E N T A :

Luis Alejandro Lara Patiño

Director de tesis:

Dr. Gibrán Fuentes Pineda

IIMAS-UNAM

Ciudad Universitaria, Cd. Mx. noviembre 2019

Agradecimientos

A mi madre y a mi padre, cuyos sacrificios posibilitaron mis estudios universitarios y de posgrado. Esta tesis es la culminación de todos estos esfuerzos y es tan suya como mía.

A mis tíos y primos, quienes me brindaron sus consejos y ayuda incondicional, considerándome como su propio hijo o hermano.

A Gibrán, mi asesor, quien fue y seguirá siendo una fuente de inspiración y admiración. Espero algún día llegar a saber y conocer tanto como él.

A mis amigos y compañeros de licenciatura y maestría, que me acompañaron durante este viaje y me proporcionaron su apoyo.

A la UNAM, el Instituto de Investigaciones Aplicadas y Sistemas y el Posgrado en Ciencia e Ingeniería de la Computación por proveerme de conocimientos que serán muy valiosos en mi vida profesional.

Al Consejo Nacional de Ciencia y Tecnología por la beca recibida durante este posgrado.

Índice general

Índice de figuras

Índice de tablas

Notación

En este trabajo, una letra sin estilo (x) representa un valor escalar, una letra minúscula en negrita ($\boldsymbol{x}$) representa un vector y una matriz (de dos o más dimensiones) se escribe mediante una letra mayúscula en negrita ($\boldsymbol{A}$).

Un elemento individual de un vector, matriz o tensor se denota con una serie de subíndices separados por comas. Por ejemplo, $\boldsymbol{x}_i$ es el i-ésimo elemento del vector $\boldsymbol{x}$, y $\boldsymbol{A}_{i,j}$ es el elemento en la i-ésima fila y j-ésima columna de la matriz $\boldsymbol{A}$.

Redes neuronales

m	Número de ejemplos en el lote
n_x	Tamaño de la entrada
$n^{[l]}$	Número de neuronas en la capa l
$\boldsymbol{X}$	Matriz de entrada
$\boldsymbol{x}^{(i)}$	i-ésimo ejemplo de entrada
$\boldsymbol{Y}$	Matriz de etiquetas
$y^{(i)}$	Etiqueta correspondiente al i-ésimo ejemplo
$\hat{\boldsymbol{Y}}$	Matriz de salidas
$\hat{y}^{(i)}$	Etiqueta correspondiente al i-ésimo ejemplo
$\boldsymbol{W}^{[l]}$	Matriz de pesos de la capa l
$\boldsymbol{b}^{[l]}$	Vector de sesgos de la capa l
$g^{[l]}$	Función de activación de la capa l
$J(\boldsymbol{\theta})$	Función de pérdida

Aprendizaje por refuerzo

G_t	Retorno esperado en el tiempo t
$V_\pi(s)$	Función de valor de estados bajo la política π
$Q_\pi(s, a)$	Función de valor de acciones bajo la política π

Capítulo 1

Introducción

Es indiscutible que los robots han dado forma al mundo en el que vivimos: desde el tornillo más pequeño hasta las más grandes obras de infraestructura, la mayoría de productos con los que interactuamos a diario han sido creados con ayuda de robots.

Pero aunque su éxito en el ámbito industrial es bastante claro, la participación de los robots en la vida cotidiana de las personas deja todavía mucho que desear; aún estamos lejos del día en el que los robots caminen en la calle junto a los humanos, ayudándolos en sus tareas cotidianas. Existen una gran variedad de razones por las cuales esto no ha sucedido y una de las principales es que los robots son todavía máquinas muy delicadas y peligrosas y requieren un entorno completamente controlado para funcionar adecuadamente.

Con el fin de lograr la inclusión de los robots en el ámbito doméstico es necesario dotarlos de la capacidad de operar en entornos impredecibles sin la necesidad de programarlos específicamente para cada tarea que realizarán. En resumen, introducir a los robots la capacidad de adaptación a nuevas circunstancias de manera automática o con una participación mínima del ser humano.

Existen muchas maneras diferentes en las que un robot puede adaptarse a su entorno; puede cambiar la configuración de sus articulaciones para realizar su tarea más eficientemente, repararse a sí mismo cuando recibe un daño significativo o aprender a partir de la observación de su entorno. Esta tesis se enfoca en la adaptación del robot a cambios en su dinámica. Es decir, si un motor falla o sucede alguna otra circunstancia que modifique la manera en la que el robot se mueve, que éste pueda aprender el nuevo comportamiento para

seguir cumpliendo con su tarea.

En particular, hay dos maneras en las que la dinámica de un robot puede modificarse. la primera de ellas corresponde a un cambio en alguno de los parámetros del modelo del sistema. Por ejemplo, que una pieza del robot disminuya su masa o que el coeficiente de fricción en una junta aumente. Estos cambios muchas veces se deben al desgaste normal del robot, aunque también surgen debido a procesos propios de la tarea del robot; si éste sujeta un objeto con su pinza, la sección que está en contacto con el objeto parecerá más pesada desde el punto de vista del controlador. El segundo caso de cambios en la dinámica del robot es cuando la estructura de su modelo se altera, lo cual sucede generalmente cuando el robot sufre modificaciones importantes, como una elongación en una de sus secciones o una falla en sus motores.

Hasta ahora hemos hablado de robots en general, pero la variedad de configuraciones que existen es inmensa y constantemente se desarrollan otras completamente diferentes. Sin embargo, uno de los tipos de robot más utilizados en la actualidad es el manipulador. Éste consiste en una serie de secciones, llamadas eslabones, que se conectan entre sí mediante motores (ver figura 1.1). La manera en la que se conectan es secuencial, de manera que cada eslabón está unido a otros dos. El primer eslabón está fijo y las secciones restantes se mueven respecto a la anterior, formando una configuración en serie. En el último eslabón se puede colocar una herramienta, que puede ser una pinza, una punta soldadora, un cortador, etc. Dado que es el tipo de robot más empleado en la industria y que son relativamente simples, gran parte de la investigación en robótica se dirige a ellos. Ésta será también la configuración que se tratará en este trabajo.

Figura 1.1: Ejemplo de robot manipulador. Imagen tomada del sitio web de ABB™ [1].

Por otro lado, las técnicas basadas en inteligencia artificial recientemente han atraído la atención del público debido a su capacidad de resolver problemas muy complejos en ámbitos

muy dispares. En el caso específico de la robótica, el campo denominado aprendizaje por refuerzo es uno de los más utilizados.

Una de las más grandes bondades del aprendizaje por refuerzo, que lo vuelven muy atractivo para el trabajo con robots es que la tarea que éstos realizarán puede definirse de manera muy intuitiva. Muchas otras técnicas, incluidas las redes neuronales, requieren conocer las respuestas correctas para un cierto número de casos de ejemplo, para que posteriormente el modelo propuesto pueda generalizar a partir de dichos ejemplos. En contraste, el aprendizaje por refuerzo requiere solamente definir una función de recompensa, la cual puede verse como una medida para indicarle a un agente inteligente qué es bueno hacer y qué no. De esta manera, no es necesario conocer de antemano casos de ejemplo para el entrenamiento, los cuales pueden ser muy complicados de obtener para tareas robóticas.

1.1. Planteamiento del problema

Al desarrollar un trabajo como el presente surgen una variedad de retos, en parte originados por el hecho de trabajar con robots, mientras otros son inherentes al trabajo con aprendizaje por refuerzo y redes neuronales. Los siguientes párrafos profundizan en dichos desafíos.

1.1.1. Modelado del sistema

Uno de los primeros pasos para diseñar un controlador por medios tradicionales es obtener el modelo del sistema. En el caso de un robot manipulador, el modelo consiste en un conjunto de 3 sistemas de ecuaciones diferenciales: la cinemática directa, la cinemática inversa y la ecuación dinámica. Cada uno de ellos tiene tantas ecuaciones como motores tenga el robot, por lo general, seis. Por lo tanto, el modelo de un robot puede tener 18 ecuaciones en total, incluso más cuando se estudian ciertas configuraciones.

No obstante, el modelo del robot por sí solo es insuficiente para diseñar un controlador dinámico. Es necesario de igual modo modelar la tarea que el robot realizará. Muchas veces esto se reduce a encontrar ecuaciones que describan la trayectoria que seguirá la herramienta del robot. Comúnmente se especifican posiciones, velocidades y aceleraciones de la misma.

Al considerar estas variables para cada una de las 6 coordenadas espaciales[1], el número de ecuaciones para describir una tarea de seguimiento de trayectoria asciende a 9. Mas aún, tareas más complejas podrían requerir un número mayor de ecuaciones y ser más complicadas de modelar. En consecuencia, no hay un límite en la complejidad del modelo de una tarea en particular.

El número de ecuaciones o la complejidad de un modelo no deberían representar un problema si el método para obtenerlas fuera estructurado y perfectamente claro. Éste es el caso de la cinemática directa; existe un método con pasos perfectamente establecidos que por convención se sigue en toda investigación en robótica, denominado Denavit-Hartenberg [2]. Por el contrario, el resto del modelo del robot y el modelo de la tarea no tienen un método establecido o una convención en su estructura; cada persona aplica su propio criterio y experiencia para obtener el modelo. Dada la dificultad de encontrar un modelo adecuado, éste es un paso que sería deseable eliminar, ya que consume recursos que de otra manera estarían dirigidos a la creación del controlador.

Debe considerarse además el hecho de que todo modelo es limitado, sólo una aproximación a la realidad. De este modo, es posible que el modelo obtenido no represente fielmente la dinámica completa del sistema. Potencialmente, un modelo incompleto puede traer consecuencias indeseables en el comportamiento del controlador. Por poner un ejemplo, si se tiene un modelo de un robot que no considera el fenómeno de la fricción seca, el controlador diseñado a partir de éste puede tener problemas en el posicionamiento fino de la herramienta debido a que ésta se detendría antes de lo previsto.

1.1.2. Operación segura del robot

Los robots manipuladores, especialmente los de tipo industrial, son capaces de desarrollar una gran cantidad de fuerza en sus motores. Esta capacidad es muy útil a la hora de llevar a cabo su tarea, pero puede llegar a ser peligrosa si no se utiliza de manera adecuada. En caso de una colisión, un robot puede causar daños severos a otros objetos o a sí mismo, resultando en grandes pérdidas de dinero. Asimismo, un golpe dado por un robot a una persona fácilmente puede romper huesos o incluso derivar en la muerte. Por tanto es imprescindible mantener

[1]Se requieren tres coordenadas para la posición y tres más para la orientación de la herramienta.

protocolos de seguridad en todo momento de operación de un robot.

Por otro lado, una de las bases del aprendizaje por refuerzo es la exploración: un agente debe realizar acciones que no había probado con anterioridad para evaluar potenciales caminos para realizar su tarea de una mejor manera. En el caso que nos ocupa esto significa que el robot realizará movimientos aleatorios, especialmente al aprender una nueva dinámica. Inmediatamente salta a la vista que existe un problema de seguridad con este enfoque; no es posible que un manipulador ejecute movimientos completamente aleatorios, ya que muy probablemente desencadenará una colisión.

Por consiguiente, un controlador por aprendizaje por refuerzo precisa una estrategia que permita la exploración de nuevas acciones sin poner en riesgo al robot y sus alrededores.

1.1.3. Tiempo de aprendizaje

Todo proceso de aprendizaje toma un cierto tiempo en llevarse a cabo, y los agentes basados en aprendizaje por refuerzo no son la excepción. Además, durante este periodo, puede considerarse que el agente no lleva a cabo trabajo útil, ya que el comportamiento del robot consiste mayormente en movimientos aleatorios y, en el mejor de los casos, subóptimos.

Independientemente de la tarea que desempeñe el robot, o si se encuentra en un entorno industrial o doméstico, siempre es deseable maximizar el tiempo productivo del mismo, lo cual puede lograrse reduciendo al mínimo el tiempo de aprendizaje del controlador. Esta estrategia cobra mayor importancia si se considera que un robot, especialmente uno de servicio, potencialmente aprenderá cientos o miles de tareas. Si tarda más que unos pocos minutos en asimilar cada una de ellas, su tiempo productivo se reducirá dramáticamente.

Finalmente, es conveniente mencionar que otra estrategia que puede ayudar a aliviar este problema es procurar que el tiempo de aprendizaje no sea completamente inútil. Aún cuando se realice una tarea parcialmente o de manera subóptima, es mejor que no llevar a cabo ninguna acción. Por supuesto, esta táctica debe considerarse para cada caso particular, dado que ciertas tareas requieren realizarse completamente y sin errores.

1.2. Objetivos

El objetivo general de este trabajo es implementar un controlador de posición para un manipulador robótico serial. Dicho controlador debe tener la capacidad de adaptarse a cambios en la dinámica del robot.

1.2.1. Objetivos específicos

- Examinar las técnicas de aprendizaje por refuerzo utilizadas hasta el momento para el control de robots, analizando sus ventajas y desventajas.
- Construir diversos entornos simulados para realizar pruebas de forma segura.
- Diseñar un modelo de aprendizaje por refuerzo capaz de aprender la dinámica de un manipulador robótico.
- Valorar la capacidad de adaptación del controlador al introducir perturbaciones en los entornos simulados.
- Evaluar el rendimiento del controlador propuesto mediante la inspección del comportamiento dinámico del robot.

1.3. Hipótesis

Este trabajo se basa en la hipótesis de que un agente de aprendizaje por refuerzo libre de modelo puede aprender una política para el control de posición del efector final de un brazo robótico y, tras la aparición de una falla en el mismo, adaptarse para continuar su tarea. Esta suposición se basa en primer lugar en el hecho de que el aprendizaje por refuerzo ha sido utilizado en varias ocasiones para el control robótico [3], [4], [5]. Se sustenta también en la idea de que, cuando un robot falla, la nueva dinámica puede considerarse como un robot completamente nuevo. Luego, como mínimo, debe ser posible entrenar un nuevo agente desde cero en el nuevo entorno.

Esta tesis considera, además, que el aprendizaje en línea es indispensable para lograr la adaptación a perturbaciones y si se implementa un mecanismo que regule el balance entre

exploración y explotación mientras el controlador está en ejecución, un agente de aprendizaje por refuerzo basado en gradiente de política puede adaptarse para funcionar en línea.

Adicionalmente, se ha encontrado que el uso de una estructura de datos para almacenar la experiencia del agente impacta positivamente en la estabilidad de los agentes de aprendizaje por refuerzo [6]. Sin embargo, este trabajo considera la posibilidad de que al reducir temporalmente el tamaño de esta estructura cuando sucede una perturbación puede acelerarse la recuperación del agente ante ella. Esto se deriva de la idea de que, tras un cambio en la dinámica del sistema, es deseable reemplazar lo más rápido posible la experiencia acumulada de la dinámica anterior por experiencia generada con la nueva dinámica. Un almacén más pequeño lograría este propósito. Sin embargo, para mantener la estabilidad una vez aprendida la nueva dinámica, se requeriría incrementar de nuevo el tamaño del mismo.

Finalmente, el método propuesto en este trabajo se evaluará bajo una variedad de perturbaciones, donde cada una representa una desviación de la dinámica original de diferente magnitud. Se espera que, cuando sucedan perturbaciones que no afecten en gran medida al modelo del robot, el agente logre una adaptación de manera más rápida y que su rendimiento tras la recuperación sea muy similar al anterior. Por otro lado, frente a perturbaciones donde el modelo del robot cambia drásticamente, se espera que el agente logre una adaptación, pero que ésta no sea tan buena como la anterior.

1.4. Contribuciones

Los principales aportes de la presente tesis son:

- **El código de un agente de aprendizaje por refuerzo para el control adaptable de brazos robóticos.** El código generado durante este trabajo es público y está diseñado de manera modular para facilitar el uso de entornos diferentes a los aquí empleados. El repositorio del proyecto se encuentra en: `https://github.com/LuisLaraP/OnlineLearningController`.
- **Un mecanismo para regular el aprendizaje basado en el comportamiento del sistema.** Este mecanismo regula la medida en la cual el agente concentra su atención

en realizar su tarea o en su aprendizaje. La regulación del aprendizaje es indispensable para la capacidad de adaptación del agente.

- **Una colección de entornos simulados con perturbaciones para evaluar la adaptabilidad de controladores.** Para la evaluación del agente implementado aquí se creó una variedad de entornos con diferentes perturbaciones. Anteriormente ya existían entornos de robots en simulación, pero ninguno se enfocaba a estudiar la adaptabilidad de controladores. Los entornos creados en este trabajo se diseñaron para evaluar algunas formas específicas de adaptación.

1.5. Organización de la tesis

Los capítulos de este trabajo se organizan como sigue:

- En el capítulo 2 se revisan cuatro de los métodos más utilizados en la actualidad para el control de brazos robóticos y se incluyen algunos de los trabajos más relevantes de cada uno de ellos.
- En el capítulo 3 se presentan las bases teóricas en la cuales se sustenta esta tesis. Está dividida en tres secciones: la primera trata sobre las redes neuronales y sus métodos de entrenamiento, la segunda introduce los conceptos básicos del aprendizaje por refuerzo y los métodos actor-crítico y la tercera es una breve explicación de las métricas que se utilizarán para evaluar el controlador desde la perspectiva del Control Automático.
- En el capítulo 4 se detalla la arquitectura de todos los elementos que conforman el agente propuesto en este trabajo, así como el funcionamiento del algoritmo utilizado para su entrenamiento.
- En el capítulo 5 se describen los entornos implementados para la evaluación del agente, así como las perturbaciones aplicadas a cada uno de ellos. Asimismo se explica el método a través del cual se obtuvieron las métricas que se reportan en este trabajo.
- En el capítulo 6 se reportan y comentan los resultados que arrojaron los diversos experimentos realizados.

- En el capítulo 7 se presentan las conclusiones finales del trabajo, analizando el rendimiento y el comportamiento de ambos agentes implementados. También se sugieren algunas líneas de trabajo que podrían ser prometedoras en el futuro.

Capítulo 2

Estado del arte

El problema del control de sistemas dinámicos precede por mucho a la invención de los robots, comenzando a utilizarse desde el siglo XIX para regular la velocidad de las máquinas de vapor. A su vez, las técnicas de control tienen una aplicación bastante amplia, incluyendo sistemas mecánicos, eléctricos y térmicos; incluso procesos químicos y nucleares.

Tradicionalmente los sistemas de control automático se han construido utilizando técnicas matemáticas y diseños creados manualmente para cada problema en particular. Sin embargo, las técnicas actuales de Inteligencia Artificial han dado origen a un gran número de enfoques novedosos para la implementación de controladores con nuevas propiedades que anteriormente eran muy complicadas o imposibles de obtener.

De estos nuevos enfoques, uno de los más prometedores es el del aprendizaje por refuerzo, dado que ha sido aplicado con éxito en una gran variedad de situaciones. La literatura relacionada a este campo, aunque extensa, permite identificar varias líneas de investigación que están activas actualmente e intentan resolver el mismo problema de maneras distintas. Aquí se agrupan algunos de los trabajos más importantes según la línea de investigación a la que pertenecen y se analizan las ventajas y desventajas que ofrece cada uno de los métodos que se emplean.

2.1. Programación Dinámica Aproximada

La programación dinámica es una técnica para optimizar la ejecución de algoritmos, que consiste en dividir el problema en otros más pequeños y calcular las soluciones de éstos de manera anticipada. Fue desarrollada por Richard Bellman a mediados del siglo pasado [7].

Originalmente, esta técnica se diseñó para utilizarse en sistemas con espacios de acciones y de estados discretos. Sin embargo, los sistemas comúnmente encontrados en la robótica poseen estados y acciones continuas. Por lo tanto es imposible la aplicación de la programación dinámica en estas tareas, por lo menos en su forma original. Para aliviar esta problemática, la programación dinámica *aproximada* [8] utiliza funciones paramétricas que aprenden los resultados de los subproblemas. De igual forma, pueden aproximar resultados para casos en los cuales no fueron entrenadas.

Un algoritmo básico de Programación Dinámica Aproximada fue desarrollado en [9]. Se basa en la técnica de iteración de valor, en la cual se calcula la utilidad de cada estado posible en el sistema de manera iterativa. Este trabajo en particular emplea este algoritmo para la creación de controladores óptimos para sistemas no lineales en tiempo discreto. Para ello, se requiere tener un modelo matemático del sistema a controlar, con el cual se calcula la utilidad de todos los estados antes de la ejecución del controlador.

Por otro lado, Song [10] desarrolló un controlador también para sistemas no lineales, que llamó .ªprendizaje por refuerzo integral", con la principal diferencia que no requería el modelo del sistema. Este algoritmo se sustenta en el esquema de control tipo PI, el cual calcula la integral del error. En consecuencia, el controlador desarrollado puede aplicarse a cualquier sistema del cual no se conoce la dinámica y funciona bajo perturbaciones igualmente desconocidas.

2.2. Gradiente de Política

Un problema que presentaban los algoritmos de aprendizaje por refuerzo originales, tales como la programación dinámica aproximada, es que la función de valor de acciones se representaba de forma tabular. Por tanto, era imposible aplicarlos a sistemas de acciones

continuas sin aplicar algún tipo de discretización. Es por este motivo que Sutton y otros obtuvieron la expresión analítica del gradiente de una política [11]. Posteriormente, el equipo de DeepMind en Google obtuvo la forma determinista del gradiente [12], la cual puede calcularse de forma más eficiente.

En este mismo trabajo se presenta una estructura novedosa para los algoritmos de aprendizaje por refuerzo: el actor-crítico, donde se parametrizan tanto la política como la función de valor, permitiendo ejecutar algoritmos en sistemas con espacios de acción y estados continuos.

Empleando esta estructura, Lillicrap y otros investigadores, igualmente en el grupo DeepMind, crearon un controlador basado en el algoritmo Q-Learning, llamado DDPG por sus siglas en inglés [13]. Este controlador utiliza dos redes neuronales: un actor y un crítico. El controlador obtenido no requería conocer el modelo del sistema y funcionaba con estados y acciones continuos. Adicionalmente, mediante la inclusión de capas convolucionales en ambas redes se logró el control de diversos entornos simulados a partir únicamente del video de la tarea.

El algoritmo DDPG alcanzó una gran popularidad y se ha vuelto prácticamente ubicuo dentro de la línea de investigación del gradiente de política. Los trabajos más recientes son principlamente modificaciones y mejoras a dicho algoritmo. Un ejemplo es el trabajo realizado por Banerjee y Chatterjee [14], donde extienden el algoritmo DDPG con un sistema de detección de errores en tiempo real. Cuando se detecta una discrepancia entre el comportamiento del sistema y el esperado, el controlador se vuelve a aprender. Como normalmente el cambio necesario es muy pequeño, el aprendizaje se lleva a cabo rápidamente.

2.3. Búsqueda Guiada de Políticas

La búsqueda de políticas surgió como uno de los primeros intentos de aplicar aprendizaje por refuerzo a problemas con espacios de acción continuos. Se basa en emplear una forma parametrizada de la política y encontrar los mejores parámetros para dicha política mediante una búsqueda [15, 16].

Las parametrizaciones que se emplearon en los primeros trabajos alrededor de la búsqueda

de políticas eran bastante sencillas; se trataba generalmente de pequeñas redes neuronales de menos de 10 neuronas. Sin embargo, conforme éstas se hicieron más complejas, su número de parámetros aumentó y la búsqueda simple dejó de ser eficiente para encontrar los valores óptimos de los parámetros. Por esta razón Levine y Koltun mejoraron la búsqueda original agregando un paso de optimización, resultando en la búsqueda guiada de políticas [17].

Aunque éste es un avance relativamente reciente, existen ya varios trabajos que utilizan la búsqueda guiada de políticas para el control de robots. Tal es el caso del realizado por Levine et.al. [18], en el cual entrenan redes neuronales para realizar varias tareas de manipulación de objetos en un robot PR2. Una de las mayores aportaciones de este trabajo fue que no existía ningún componente del controlador diseñado a mano: la tarea se aprendía completamente dentro de la red neuronal, la cual recibía el video de las cámaras del robot y los ángulos de las articulaciones y devolvía los torques que se requería aplicar a cada uno de los motores del robot.

2.4. Función de ventaja normalizada

La gran mayoría de algoritmos de aprendizaje por refuerzo requieren obtener en cada instante de tiempo la acción que represente la mayor recompensa en un momento dado. En un sistema con un espacio de acciones continuo, encontrar este resultado de manera exacta es intratable, ya que al haber un número infinito de acciones, se requiere realizar una optimización completa, cuyo cómputo lleva mucho tiempo.

Como respuesta a este problema, el equipo de Google Deepmind encontró una manera de parametrizar una política que además permite encontrar la acción con el valor más alto de manera inmediata, sin realizar una optimización [19]. Además, diseñó un algoritmo para ejecutar y entrenar la política simultáneamente, el cual se basa en Q-Learning. Para acelerar el proceso de aprendizaje, en este trabajo se utiliza el modelo aprendido del sistema para generar muestras de entrenamiento sintéticas, las cuales se incorporan a las generadas en el sistema real. El resto del algoritmo es muy similar y emplea las mismas técnicas que DDPG. El algoritmo creado en este trabajo se evaluó sobre una variedad de entornos simulados de robótica. Sus resultados muestran una mejora respecto al algoritmo DDPG original, apren-

diendo una política adecuada de manera más rápida y en general obteniendo una mayor recompensa.

Posteriormente, el mismo equipo que creó el algoritmo lo puso a prueba utilizando robots físicos [20]. Para ello realizaron modificaciones a su algoritmo para permitir a varios robots operar en paralelo realizando la misma tarea y así acelerar el aprendizaje. Asimismo implementaron diversas medidas de seguridad para evitar daños a los robots o a sus operadores. Se evaluaron dos tareas: control de posición y apertura de puertas. Sus resultados indican que el método de función de ventaja normalizada es lo suficientemente eficiente para implementarse en una plataforma física y que el utilizar varios agentes funcionando en paralelo acelera significativamente el aprendizaje.

Capítulo 3

Marco Teórico

En este capítulo se introducen algunos conceptos que son clave para comprender completamente esta tesis.

3.1. Redes neuronales

El capítulo 2 menciona varios trabajos que utilizan redes neuronales artificiales en conjunción con un algoritmo de aprendizaje por refuerzo. En efecto, esta es la manera más popular de parametrizar una política en el mundo académico actual. Por este motivo se repasarán aquí los fundamentos teóricos de las redes neuronales que tengan relevancia para el presente trabajo.

3.1.1. La neurona artificial

Una red neuronal artificial (RNA) es un modelo computacional, comúnmente utilizado para la aproximación de funciones muy complejas. Pese a que están inspiradas en la estructura del sistema nervioso de los animales, las RNA son una simplificación de éste y su desarrollo ha seguido su propio curso desde entonces. En consecuencia, los modelos computacionales están ahora muy poco relacionados con sus contrapartes biológicas.

Las RNA están conformadas por un gran número de unidades de cómputo llamadas neuronas artificiales. Cada una de ellas recibe un vector de entrada $\boldsymbol{x}$ y produce una salida

escalar a. La capacidad de aprendizaje de las neuronas está dada por un conjunto de parámetros ajustables $\boldsymbol{w}$, también llamados *pesos*, así como un parámetro extra b, denominado *sesgo*. La salida de una neurona está definida entonces como:

$$a = g\left(\sum_{i=1}^{n} \boldsymbol{w}_i \boldsymbol{x}_i + b\right) \tag{3.1}$$

donde $\boldsymbol{x}_i$ y $\boldsymbol{w}_i$ son el i-ésimo componente del vector de entradas y el vector de pesos, respectivamente. La función g es llamada *función de activación*, generalmente no lineal.

Las funciones de activación son indispensables para proporcional a la red neuronal la capacidad de aproximar cualquier función [21]. Existe una gran variedad de funciones de activación y cada una de ellas es adecuada para un tipo de aplicación diferente. En la tabla 3.1 se muestran algunas de las más importantes y utilizadas, algunas de las cuales se emplearán en este trabajo. La más simple de ellas es la activación lineal, que es equivalente a no utilizar ninguna función. La siguiente es la función tangencial hiperbólica (tanh), cuyo rango es el intervalo $[-1, 1]$. Por este motivo puede usarse para limitar la salida de una neurona a dicho intervalo. La función localizada hasta abajo es llamada comúnmente ReLu y es muy utilizada en redes neuronales profundas. Aunque en su parte positiva es igual a la lineal, introduce una no linealidad al ser cero en toda su parte negativa.

3.1.2. Redes neuronales multicapa

La manera más simple de organizar neuronas para formar redes neuronales es en capas. A manera de ejemplo, la figura 3.1 muestra una RNA con tres capas. En esta figura, las entradas a la red se encuentran en la parte izquierda del diagrama, mientras que la salida está en la derecha. Nótese que las entradas a la red están representadas como neuronas. Sin embargo, no realizan ninguna clase de cómputo; en realidad se dibujan de esta manera por comodidad en la notación. A la primera capa de una RNA, formada por estas neuronas se le llama *capa de entrada*. Análogamente, la última capa de una red es comúnmente llamada *capa de salida*. Todas las capas restantes que no son de entrada o de salida son llamadas *capas ocultas*.

Dentro de una misma capa, todas las neuronas comparten sus entradas, aunque los pesos

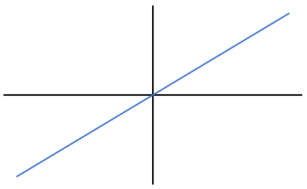
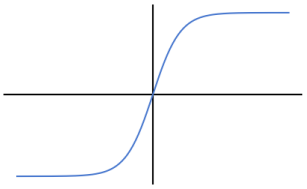
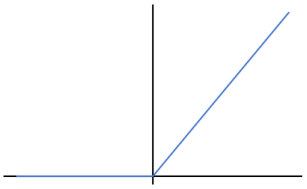
Nombre	Ecuación	Gráfica
Lineal	$g(x) = x$	
Tangente hiperbólica (tanh)	$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	
Lineal rectificada (ReLu)	$g(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$	

Tabla 3.1: Algunas funciones de activación comunes.

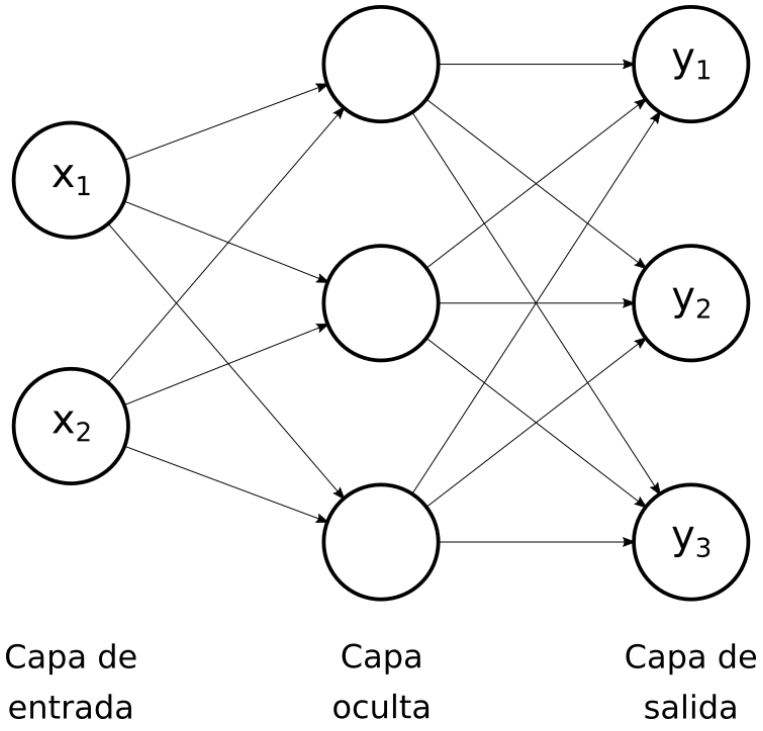


Figura 3.1: Red neuronal artificial con tres capas.

y los sesgos de cada una son independientes. La salida de cada neurona en una capa se conecta a una entrada de todas las neuronas de la capa siguiente. La conectividad es exclusivamente entre capas adyacentes; no existen conexiones hacia capas más adelante, ni hacia capas previas. Al conectar varias capas de neuronas en cascada es posible aprender funciones que con una sola capa sería imposible. De hecho, se ha demostrado que una RNA con una sola capa oculta puede aproximar cualquier función continua de sus entradas [22], siempre y cuando ésta cuente con suficientes neuronas en dicha capa.

En la práctica es más eficiente considerar una capa como una sola entidad, en lugar de pensar en ella como neuronas separadas. De esta manera, se tiene una sola matriz de pesos W para toda la capa, la cual está formada por los vectores de pesos de cada una de las neuronas. La matriz de pesos tiene un tamaño de $n^{[l]} \times n^{[l-1]}$, donde $n^{[l]}$ es el número de neuronas en la capa actual y $n^{[l-1]}$ es el número de neuronas en la capa previa. Asimismo se tiene un vector de sesgo $\boldsymbol{b}$ de longitud $n^{[l]}$, formado por el sesgo de todas las neuronas. Finalmente, la salida de la capa es un vector de tamaño $n^{[l]}$, es decir, con una salida por cada neurona. Tomando todo lo anterior en consideración, la salida de una sola capa de una RNA se puede escribir como:

$$\boldsymbol{a}^{[l]} = g^{[l]}\left(\boldsymbol{W}^{[l]}\boldsymbol{x} + \boldsymbol{b}^{[l]}\right) \tag{3.2}$$

donde x es el vector de entradas a la capa.

A la hora de conectar varias capas para formar una red neuronal, el diseñador tiene la libertad de elegir cuántas capas utilizar, así como la cantidad de neuronas en cada una de ellas. De la misma manera, es posible elegir una función de activación diferente para cada capa. Estos factores constituyen la *arquitectura* de la red. Normalmente se parte de estructuras preestablecidas, probadas en tareas similares y se realizan modificaciones a la misma mediante un método de búsqueda.

3.1.3. Entrenamiento

El entrenamiento de una red neuronal corresponde al proceso de encontrar los valores adecuados de sus parámetros (pesos y sesgos), de modo que se aproxime una función deseada.

A lo largo de la historia de las RNA han surgido una gran cantidad de métodos y algoritmos de entrenamiento y se han adaptado algunos de otras áreas de la computación. Los primeros algoritmos propuestos eran bastante simples, basados en búsquedas. Sin embargo, las redes neuronales en la actualidad tienen un número de parámetros en el orden de millones, lo que imposibilita la aplicación de este tipo de técnicas. Las soluciones más populares están basadas en el algoritmo del *descenso del gradiente*, el cual se explica a continuación.

Función de pérdida

Uno de los componentes más importantes del algoritmo de entrenamiento por descenso del gradiente es la función de pérdida, también llamada función de costo. Su finalidad es la de evaluar el rendimiento de la red en la realización de una tarea específica. Como su nombre sugiere, el valor que toma para una determinada respuesta representa de alguna manera el costo asociado a dicha respuesta. De esta manera, si la salida de una red neuronal no es muy relevante para su tarea, la función de pérdida tomará un valor alto. En caso de que la red neuronal sea muy apta en la realización de su tarea, la función de pérdida tendrá un valor bajo.

La forma de la función de costo depende fuertemente de la tarea que se busca que la red neuronal lleve a cabo, de modo que es necesario diseñarla para cada caso. No obstante, algunas tareas se presentan muy a menudo y existen funciones de pérdida diseñadas para ellas que han sido probadas en varias ocasiones y funcionan muy bien. Algunos ejemplos de estas tareas son la regresión, clasificación binaria y clasificación multiclase. La tabla 3.2 lista algunas funciones de costo que se emplean en estos casos.

Tarea	Función de pérdida
Regresión	Error cuadrático medio
Clasificación binaria	Entropía cruzada binaria
Clasificación multiclase	Entropía cruzada categórica

Tabla 3.2: Funciones de pérdida para algunas tareas populares.

Descenso del gradiente

Una vez diseñada la función de pérdida, el entrenamiento de la RNA se reduce a un problema de optimización. De acuerdo con lo mencionado en la sección anterior, la función de pérdida toma valores bajos cuando la red neuronal realiza correctamente su tarea. Luego, el objetivo del entrenamiento es minimizar esta función, ya que en el punto mínimo de la misma el rendimiento de la red es el mejor posible. Uno de los muchos métodos de optimización que existen, y el más utilizado para el entrenamiento de redes neuronales, es el descenso del gradiente. Este algoritmo requiere obtener la derivada de la función de pérdida respecto a todos los parámetros entrenables de la red, es decir los pesos y los sesgos.

Las funciones de pérdida, en su forma general, son funciones escalares de variable vectorial, es decir, tienen varias entradas y una sola salida. En estos casos, su gradiente toma la forma de un vector formado por la derivada parcial de la salida con respecto a cada una de las entradas. La dirección en la cual apunta este vector es aquella en la que la función cambia más rápidamente, o para ponerlo de una manera más intuitiva, la dirección en la que la pendiente de la función es más pronunciada.

El algoritmo del descenso del gradiente toma ventaja de este hecho; la idea general es ajustar iterativamente los parámetros de la red neuronal en la dirección contraria a la que apunta el gradiente, hacia donde la función decrece más rápidamente. Eventualmente, los parámetros convergerán al valor en el cual la función de pérdida es mínima.

La estructura general del descenso del gradiente se muestra en el algoritmo 1. En esta figura, los parámetros de la red están representados por $\boldsymbol{\theta}$, que es un vector de números reales de tamaño n.

Algoritmo 1: Descenso del gradiente.

$\boldsymbol{\theta}_0 \leftarrow \boldsymbol{k} \in \mathbb{R}^n$
$t \leftarrow 0$
repetir
  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta})$
  $t \leftarrow t + 1$
hasta que *se alcance convergencia*

En primer lugar, los parámetros son inicializados con un valor genérico. Muchas veces se

utiliza el cero como valor inicial. Sin embargo, en las redes neuronales esto no es recomendable, ya que éstas sufren de un problema denominado *simetría*. Éste consiste básicamente en el hecho de que, como todas las neuronas de una capa se conectan con todas las neuronas de la capa siguiente, es posible intercambiar de lugar dos neuronas en la misma capa y la red seguirá produciendo la misma salida. En otras palabras, todas las neuronas de una capa dada son iguales. Por lo tanto, si se inicializan los pesos de la red a cero, no se podrá diferenciar las diferentes neuronas, y todas recibirán el mismo gradiente. En consecuencia, al finalizar el entrenamiento todas ellas tendrán los mismos pesos. Es por ello que en la práctica los pesos de las redes neuronales comúnmente se inicializan con valores aleatorios muy pequeños. De esta manera el gradiente toma un valor ligeramente diferente para cada una de ellas y se vuelve posible diferenciarlas.

Posteriormente, los parámetros de la red se actualizan de forma iterativa mediante la regla de actualización:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{3.3}$$

donde J es la función de pérdida y α es un hiperparámetro llamado *tasa de aprendizaje*. Un *hiperparámetro* es una variable cuyo valor se fija antes de la ejecución de un algoritmo. El descenso del gradiente solamente tiene uno; la tasa de aprendizaje. En pocas palabras, ésta controla la velocidad del aprendizaje; si toma un valor pequeño, el aprendizaje será lento. Por otro lado, si α es alta, el aprendizaje será más rápido.

Debe tenerse cuidado a la hora de determinar el valor de la tasa de aprendizaje, ya que valores extremos de la misma pueden causar problemas durante la ejecución del descenso del gradiente. Un valor demasiado bajo ralentizará el proceso de aprendizaje, pudiendo llegar al punto de volverse intratable. Por otro lado, un valor demasiado alto de α puede provocar que el algoritmo diverja en lugar de converger.

El proceso de aprendizaje se detiene a través de un criterio de convergencia, el cual es la condición de salida del ciclo de actualización de los parámetros. El criterio que se utilice puede variar dependiendo de la tarea, pero algunos que se utilizan frecuentemente incluyen:

- **Número de iteraciones.** El algoritmo se detiene tras haber transcurrido una cierta cantidad de pasos.

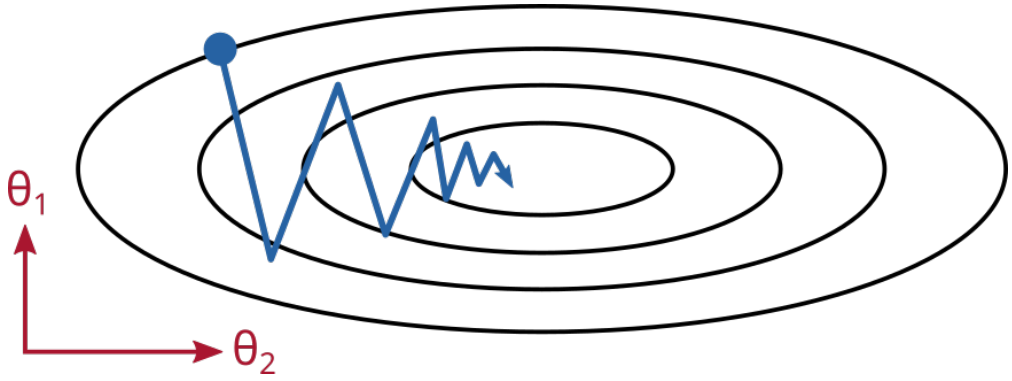


Figura 3.2: Ejemplo de una función de pérdida cuya magnitud cambia drásticamente de una dimensión a otra.

- **Razón de cambio de la función de pérdida.** Cuando la función de pérdida disminuye muy lentamente, podría significar que ya se está suficientemente cerca de un mínimo. Hay que tener cuidado al utilizar este criterio, ya que un gradiente muy pequeño puede también significar que la función de pérdida en la región actual es simplemente muy plana, aunque no haya un mínimo cerca. Existen modificaciones al algoritmo original diseñadas para tratar con este problema.

- **Distancia a cero de la función de pérdida.** Algunas funciones de pérdida tienen mínimos en los cuales el valor del error es cero. Por lo tanto, puede revisarse si el error alcanza un valor cercano a cero, y si el error cae debajo de éste, puede detenerse el aprendizaje.

3.1.4. Optimizador Adam

Adam es un algoritmo que incorpora modificaciones al algoritmo original del descenso del gradiente que en muchas ocasiones mejoran la estabilidad y reducen el tiempo de aprendizaje de un modelo.

Uno de los principales problemas que pueden encontrarse a la hora de entrenar un modelo de aprendizaje automático es que la forma de la función de pérdida varíe fuertemente en cada una de sus dimensiones. Recordemos que ésta es una función de los parámetros del modelo. Entonces, puede ocurrir que la función de pérdida sea muy angosta respecto a algunos parámetros, pero muy amplia respecto a otros (véase figura 3.2). En este escenario,

una tasa de aprendizaje adecuada para el parámetro θ_2 es demasiado grande para el otro parámetro y puede llegar a causar divergencia en el proceso de aprendizaje. El diseñador del modelo está obligado entonces a reducir la tasa de aprendizaje para adecuarla al parámetro θ_1, lo que en general incrementa el tiempo de entrenamiento.

Para aliviar este problema se propuso agregar *momentum* al algoritmo del descenso del gradiente [23]; en lugar de actualizar los parámetros del modelo a partir del gradiente de la función de pérdida directamente, como en la ecuación 3.3, se calcula una media móvil pesada exponencialmente (MME) de este gradiente, denotado por $V_{d\boldsymbol{\theta}}$:

$$V_{d\boldsymbol{\theta}} = \beta_1 V_{d\boldsymbol{\theta}} + (1 - \beta_1)\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{3.4}$$

Una media móvil es un tipo de filtro paso bajas de una señal; es decir, sigue la tendencia de una variable pero ignora las oscilaciones rápidas que ésta presente. En particular, la media móvil exponencial es más sensible a las observaciones más recientes y presta menor atención a los datos vistos en el pasado. El hiperparámetro $\beta_1 \in [0, 1]$ controla en qué grado se presta atención a los datos pasados.

La cantidad calculada en la ecuación 3.4 se utiliza entonces para actualizar los pesos del modelo, como sigue:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha V_{d\boldsymbol{\theta}} \tag{3.5}$$

En el ejemplo de la figura 3.2, el uso del *momentum* tendrá el efecto de atenuar las oscilaciones en la dirección de θ_1, mientras que el gradiente en la dirección de θ_2 se mantendrá prácticamente igual, por lo que será posible emplear una tasa de aprendizaje mayor y reducir el tiempo de entrenamiento.

Otro problema que surge durante el entrenamiento de muchos modelos es que comúnmente se requiere definir una tasa de aprendizaje diferente para cada uno de los parámetros del modelo, e incluso la tasa de aprendizaje óptima puede variar en diferentes etapas del entrenamiento. Sin embargo, establecer manualmente una tasa de aprendizaje para cada parámetro resultaría muy complicado. En su clase en Coursera, Geoffrey Hinton [24] propuso una técnica para adaptar automáticamente la tasa de aprendizaje de cada parámetro de un modelo independientemente. De manera similar al momentum, se calcula la MME del

cuadrado de la derivada de la función de pérdida respecto a los parámetros ($S_{d\boldsymbol{\theta}}$):

$$S_{d\boldsymbol{\theta}} = \beta_2 S_{d\boldsymbol{\theta}} + (1 - \beta_2) \left(\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) \right)^2 \tag{3.6}$$

donde la operación de exponenciación se realiza por cada elemento en el vector de parámetros.

La cantidad calculada en la ecuación 3.6 se utiliza para escalar la tasa de aprendizaje, como sigue:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{S_{d\boldsymbol{\theta}}} + \epsilon} \frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{3.7}$$

Nótese el uso de un número pequeño, ϵ, para evitar dividir entre cero.

Al aplicar esta técnica, los parámetros que consistentemente presenten un gradiente muy alto en la función de pérdida resultarán en una MME también alta. En el cálculo de la ecuación 3.7, la tasa de aprendizaje será dividida por esta cantidad y al final su magnitud disminuirá. Por otro lado, los parámetros con gradientes consistentemente bajos verán incrementada su tasa de aprendizaje. Adicionalmente, estos efectos cambiarán si la forma del gradiente de la función de pérdida se modifica, lo que vuelve esta técnica muy útil en problemas no estacionarios.

Un pequeño problema que presenta el cálculo de las medias móviles exponenciales es que, en el inicio del proceso de entrenamiento, cuando hay pocas muestras, exhiben un sesgo que las aleja del valor real de la media. Conforme más muestras son recabadas este sesgo disminuye exponencialmente, hasta que es prácticamente cero. Como este fenómeno solamente ocurre al inicio del entrenamiento y permanece por un tiempo relativamente corto, muchas implementaciones no hacen nada por solucionarlo. No obstante, en aquellos casos donde la media móvil sea importante, puede aplicarse la siguiente corrección de sesgo:

$$\widehat{MME}(x) = \frac{MME(x)}{1 - \beta^t} \tag{3.8}$$

donde t es el número de muestras que se han visto hasta el momento.

Finalmente, el algoritmo Adam [25] incorpora todas las mejoras mencionadas arriba. La forma del algoritmo es la misma que en el descenso del gradiente; lo único que cambia es la

ecuación para la actualización de los parámetros:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\widehat{S}_{d\boldsymbol{\theta}}} + \epsilon} \widehat{V}_{d\boldsymbol{\theta}} \tag{3.9}$$

El resultado es un algoritmo robusto ante entornos ruidosos y no estacionarios, que funciona bien con modelos que exhiben un gradiente no convexo y altamente variable respecto a cada uno de sus parámetros.

3.2. Aprendizaje por refuerzo

En esta sección se introduce el campo del aprendizaje por refuerzo y las técnicas basadas en el gradiente de la política, de las cuales forma parte este trabajo.

El aprendizaje por refuerzo es un campo de la inteligencia artificial que se ocupa de seleccionar acciones para que un agente inteligente maximice una medida de rendimiento. Al contrario que en el aprendizaje supervisado, este tipo de técnicas no reciben ejemplos de acciones deseadas, sino que encuentran por sí mismas las mejores acciones a realizar, siendo uno de sus puntos críticos el balance entre la exploración de los conocimientos adquiridos y la exploración de acciones nuevas.

3.2.1. El modelo del aprendizaje por refuerzo

En un escenario común de aprendizaje por refuerzo, un agente interactúa con su entorno en pasos discretos de tiempo. En cada instante t, el agente recibe una observación que le da alguna idea del estado en el que se encuentra el entorno s_t. De acuerdo con esta información, el agente selecciona una acción a_t del conjunto de acciones posibles A. La ejecución de esta acción a su vez influye sobre el entorno, provocando que éste cambie su estado. En el siguiente instante de tiempo, el entorno comunica al agente el valor de la acción que acaba de realizar mediante una *recompensa* r_{t+1}, la cual es un valor escalar. Junto con esta información, el agente recibe además la observación correspondiente al siguiente estado del entorno, s_{t+1} [22].

Un sistema de aprendizaje por refuerzo completo, además del agente y el entorno, contiene cuatro elementos principales [26]:

- **Política (π).** Es un mapeo de observaciones del agente a acciones. La forma de una política puede ser tan simple como una tabla de consulta, o puede involucrar una gran cantidad de cómputo, como una red neuronal o un proceso de búsqueda.

- **Señal de recompensa.** Es un valor escalar que define el objetivo del problema de aprendizaje por refuerzo. En términos generales indica qué tan buenas o malas son las acciones que toma el agente.

- **Función de valor ($V(s)$).** Especifica qué tan deseable es encontrarse en un determinado estado en el largo plazo. De manera intuitiva, si un estado es muy cercano al estado objetivo, por lo general su valor será alto. Por el contrario, si cualquier acción en un determinado estado solamente aleja al agente de su objetivo, el valor de dicho estado será bajo. De manera análoga existe una función de valor para acciones, $Q(s, a)$, la cual indica qué tan beneficioso sería para el agente ejecutar la acción a en el estado s.

- **Modelo del entorno.** Este elemento puede o no estar presente. Es cualquier entidad que permita al algoritmo de aprendizaje por refuerzo obtener información acerca del comportamiento del entorno. La presencia o ausencia de este elemento permite hacer una categorización de los algoritmos de aprendizaje por refuerzo: aquellos que requieren un modelo del entorno son llamados *basados en modelo*, mientras que los que prescinden del mismo se denominan *libres de modelo*.

El objetivo del aprendizaje por refuerzo es, entonces, encontrar una política tal que maximice la recompensa recibida a largo plazo. Se debe tomar en cuenta que tanto la política como el entorno son, en el caso general, estocásticos. Una política estocástica implica que en un mismo estado, el agente puede ejecutar una de entre un conjunto de acciones posibles, cada una de las cuales tiene una probabilidad asociada de ser seleccionada. De este modo, una política se vuelve un mapeo de estados a distribuciones de probabilidad de acciones. Por otro lado, un entorno estocástico significa que, al ser ejecutada una determinada acción en un estado dado, el entorno puede cambiar a algún nuevo estado determinado mediante un proceso aleatorio.

3.2.2. Funciones de valor

Como se mencionó anteriormente, el objetivo del aprendizaje por refuerzo es maximizar la recompensa que un agente recibe a largo plazo. Formalmente, esta medida se denomina *retorno esperado*. Si el agente se encuentra actualmente en el instante de tiempo t, el retorno esperado a partir de dicho momento se define como la suma de recompensas obtenidas para cada acción futura:

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{3.10}$$

donde T es el instante de tiempo final. Sin embargo, existen casos en los cuales no existe un tiempo final y la tarea se extiende en el tiempo indefinidamente, lo cual causa que la suma anterior se vuelva infinita. Para asegurar convergencia a un valor finito, se introduce un *factor de descuento* γ, donde $0 \leq \gamma \leq 1$, resultando en la siguiente forma:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.11}$$

La forma descontada, además de asegurar un valor finito para el retorno, provoca que el agente preste mayor atención a las recompensas más inmediatas y que siga considerando, aunque en menor medida, las recompensas lejanas. El balance entre estos dos extremos se controla mediante el valor del factor de descuento. Nótese que, si $\gamma = 1$, la forma descontada es igual a la ecuación 3.10, por lo cual se utiliza comúnmente la forma de la ecuación 3.11.

La mayoría de algoritmos de aprendizaje por refuerzo utilizan las llamadas *funciones de valor* para su funcionamiento. Existen dos tipos: funciones de valor de estados $V(s)$ y de acciones $Q(s, a)$. Intuitivamente, la función de valor indica qué tan deseable es estar en el estado s (en el caso de $V(s)$) o ejecutar la acción a en el estado s (en el caso de $Q(s, a)$). Evidentemente, un estado o acción será más deseable si puede obtenerse una mayor cantidad de recompensa en el futuro. Por tanto, la definición de estas funciones está dada en función del retorno:

$$V_\pi(s) \doteq \mathbb{E}_\pi[G_t | s_t = s] \quad \forall s \in S \tag{3.12}$$

$$Q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | s_t = s, a_t = a] \quad \forall s \in S, \forall a \in A \tag{3.13}$$

Adicionalmente, puede verse que el retorno que se recibirá depende de las acciones que se tomen en el futuro y que éstas, a su vez, dependen de la política. En consecuencia las funciones de valor se definen con respecto a una política determinada y equivalen al retorno recibido tras visitar el estado s, ejecutar la acción a y seguir la política π después. En las ecuaciones de arriba, $\mathbb{E}_\pi[\cdot]$ denota el valor esperado de una variable aleatoria si el agente sigue la política π.

Podemos sustituir la expresión para el cálculo del retorno (ecuación 3.11):

$$\begin{aligned} V_\pi(s_t) &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t \right] \\ &= \mathbb{E}_\pi \left[r_t + \gamma \sum_{k=1}^{\infty} \gamma^k r_{t+k+1} | s_t \right] \\ &= \mathbb{E}_\pi \left[r_t + \gamma V_\pi(s_{t+1}) | s_t \right] \end{aligned} \tag{3.14}$$

y de igual manera para la función de valor de acciones:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[r + \gamma Q_\pi(s', a') | s \right] \tag{3.15}$$

donde s' es el estado siguiente a s y a' es la acción a ejecutar en dicho estado, de acuerdo con la política π. Se observa, entonces que ambas funciones de valor cumplen con una propiedad de recursividad, ya que el valor de una acción o un estado puede calcularse a partir del valor del estado o acción siguientes. A las ecuaciones 3.14 y 3.15 se les conoce como *ecuaciones de Bellman.*

Mediante las ecuaciones de Bellman es posible calcular las funciones de valor mediante un método conocido por la palabra inglesa *bootstrapping.* Éste consiste en asignar inicialmente un valor arbitrario a todos los estados y acciones (que puede ser cero). Posteriormente, conforme el agente visita los diferentes estados y realiza distintas acciones en ellos, el valor de los mismos se actualiza de acuerdo a la recompensa recibida y el valor del siguiente estado o acción. De esta manera, la función de valor calculada tenderá eventualmente a los valores reales de estados y acciones para la política actual. El proceso a través del cual se calcula la función de valor para una política es denominado *evaluación de la política.*

3.2.3. Métodos actor-crítico

Mediante el uso de la técnica de evaluación de la política se puede construir una primera aproximación de algoritmo de aprendizaje por refuerzo. Para ello se inicia con una política arbitraria; puede ser incluso completamente aleatoria. Posteriormente, el agente interactúa con el entorno utilizando la política actual. Cuando se haya reunido una cantidad suficiente de experiencia, al término de un episodio, por ejemplo, se realiza un proceso de evaluación de la política. La información que contiene la función de valor recién calculada puede ser empleada para hacer una mejora de la política actual. Si se desea maximizar la recompensa a largo plazo, entonces tal vez la mejor manera de lograrlo es modificar la política para ejecutar siempre las acciones con el mayor valor:

$$\pi(s) \leftarrow argmax_a Q(s,a) \quad \forall s \in S \tag{3.16}$$

Éste de hecho, es uno de los algoritmos más populares de aprendizaje por refuerzo, llamado *Q-Learning* [27].

En teoría, la cantidad de experiencia necesaria para hacer una correcta evaluación de una política es infinita. Sin embargo, se ha demostrado que incluso con una sola acción que realice el agente es suficiente para asegurar la convergencia de *Q-Learning* a la política óptima.

La forma básica de *Q-Learning* asume que la función de valor está dada en forma tabular, es decir, que existe una tabla con una entrada por cada una de las parejas acción-valor posibles en el problema, donde se almacenan los valores de cada una de ellas. No obstante, conforme crece el tamaño de los espacios de estados y de acciones, el tamaño de esta tabla también lo hace. En el caso en el que éstos son continuos, se tiene un número infinito de estados y de acciones y la forma tabular se vuelve imposible. Como solución a este problema, se utiliza una forma aproximada de la función de valor, en la cual no se representa como una tabla, sino como una función parametrizada por un vector $\boldsymbol{\theta}^Q$. Nótese que esta forma es solamente una aproximación de la función real de valor, ya que normalmente se utiliza un número de parámetros mucho menor al número de estados, por lo que al modificar un elemento de $\boldsymbol{\theta}^Q$, se afecta el valor de varios estados o acciones.

Regresando a la mecánica básica de *Q-Learning* (ecuación 3.16), se tiene que la política es

actualizada para que se ejecuten las acciones que tienen el valor más alto en cada estado. Sin embargo, al utilizar una función de valor parametrizada, la extracción de las acciones más valiosas deja de ser sencillo. Para ello se requeriría realizar una optimización de la función de valor, lo cual lleva mucho tiempo y puede tener un impacto considerable en el rendimiento del algoritmo en general, en especial si la mejora de la política se realiza en cada iteración del mismo. En este caso es beneficioso también utilizar una forma parametrizada de la política mediante un vector $\boldsymbol{\theta}^{\mu}$. De esta manera es posible entrenar la política utilizando un método que no involucre a la función de valor. De hecho, al emplear una política de forma paramétrica, la función de valor se vuelve prescindible, aunque en muchas ocasiones se mantiene para guiar el entrenamiento de la política. A los métodos que utilizan parametrizaciones tanto de la función de valor como de la política se les llama *métodos actor-crítico*, donde el actor $\mu(s)$ se encarga de seleccionar las acciones del agente, representando a la política, y el crítico $Q(s, a)$ evalúa las acciones realizadas, que es el trabajo de la función de valor.

3.2.4. Gradiente de política

Los métodos actor-crítico requieren, además del algoritmo de aprendizaje por refuerzo, de técnicas para entrenar tanto la función de valor como la política; es decir, para encontrar los valores adecuados de sus parámetros. Afortunadamente, algunas de las ideas que se trataron para las redes neuronales son de utilidad en el aprendizaje por refuerzo.

La función de valor es la más sencilla de entrenar, ya que la tarea que realiza es un caso de regresión: recibe una entrada multidimensional y regresa un valor escalar. Además, los ejemplos de entrenamiento son fácilmente calculables; a la recompensa recibida por ejecutar una acción se suma el valor del estado siguiente, el cual se obtiene de la misma función de valor, aplicando el factor de descuento. Por lo tanto, pueden emplearse los métodos de entrenamiento vistos en la primera sección de este capítulo.

En el caso del actor, el entrenamiento es más complicado, porque no se pueden generar ejemplos de entrenamiento para la política. En su lugar, es necesario encontrar una función de pérdida que permita evaluar el rendimiento de la política y actualizar los pesos mediante alguna técnica basada en gradiente. Aunque por mucho tiempo se creyó que tal cosa no era posible, hoy en día se cuenta con un método para entrenar al actor gracias al *teorema del*

gradiente de política.

Para encontrar una medida de rendimiento adecuada, debe recordarse que el objetivo del aprendizaje por refuerzo es maximizar la recompensa recibida a largo plazo. Por lo tanto, puede utilizarse el valor del estado inicial como medida de rendimiento. Esto funciona porque el valor de un estado es precisamente el valor esperado de las recompensas futuras recibidas al seguir una determinada política. Luego, el valor del estado inicial es el valor esperado de la recompensa para todo el episodio. Con el fin de considerar la posibilidad de múltiples estados iniciales, se calcula el valor esperado del retorno para todos ellos:

$$J(\boldsymbol{\theta}^{\mu}) = \mathbb{E}[V(s_0)] \tag{3.17}$$

A partir de la ecuación anterior puede expandirse la definición del valor esperado:

$$J(\boldsymbol{\theta}^{\mu}) = \int_S \rho^{\pi}(s) \int_A \pi_{\boldsymbol{\theta}^{\mu}}(s,a) r(s,a) da ds \tag{3.18}$$

donde $\rho^{\pi}(s)$ es la distribución de probabilidad de visitar cada estado bajo la política π, en la cual ya se incluye el descuento de la recompensa. Si la política es determinista (como es el caso de este trabajo), puede eliminarse la integral interna:

$$J(\boldsymbol{\theta}^{\mu}) = \int_S \rho^{\pi}(s) r(s, \mu(s)) ds \tag{3.19}$$

En [12], Silver et.al demuestran que la ecuación anterior puede escribirse de la siguiente forma:

$$J(\boldsymbol{\theta}^{\mu}) = \int_S \rho^{\pi}(s) \nabla_{\boldsymbol{\theta}} \mu(s) \nabla_a Q(s, \mu(s)) ds \tag{3.20}$$

que, nuevamente, toma la forma de un valor esperado:

$$J(\boldsymbol{\theta}^{\mu}) = \mathbb{E}[\nabla_{\boldsymbol{\theta}} \mu(s) \nabla_a Q(s, \mu(s))] \tag{3.21}$$

Si se aplica la ecuación 3.21 para entrenar al agente puede pensarse que los parámetros del actor se mueven en dirección del gradiente de la función de valor, es decir, en la dirección en la que las acciones tienen un mayor valor. Con el tiempo, las acciones seleccionadas por

el actor tenderán a ser las óptimas. De igual manera, en la práctica es posible ignorar el cómputo de la distribución de probabilidad de los estados $\rho^{\pi}(s)$ y llevar a cabo solamente un muestreo de la misma, de tal manera que la actualización de pesos mediante el descenso del gradiente resulta de la siguiente forma:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} + \alpha \frac{1}{N} \sum_{i} \nabla_{\boldsymbol{\theta}} \mu(s_i) \nabla_a Q(s_i, \mu(s_i)) \tag{3.22}$$

donde el gradiente se calcula a partir de N muestras.

3.3. Generalidades de control automático

Ahora se abordarán algunos conceptos acerca de la respuesta en tiempo de sistemas dinámicos, los cuales se utilizarán más adelante para evaluar el rendimiento del modelo desarrollado en esta tesis.

La respuesta en tiempo de un sistema dinámico es la forma en la cual evoluciona su salida en función del tiempo. Esta respuesta está influenciada por la dinámica interna del sistema y por la entrada que éste recibe. En el caso de que el sistema no reciba ninguna entrada, su respuesta se conoce como *respuesta natural*. Si, por el contrario solamente se analiza la respuesta generada por la entrada, sin considerar la dinámica del sistema, se está hablando de la *respuesta forzada*. En este trabajo se tratará solamente la *respuesta completa*, formada por la suma de la respuesta natural y la respuesta forzada.

La entrada de un sistema dinámico puede consistir en cualquier señal en función del tiempo, pero la que más se utiliza para el análisis de sistemas de control es la función escalón. Esta función está definida como:

$$H(x) = \begin{cases} 0 & \text{si } x < 0 \\ c & \text{si } x \geq 0 \end{cases} \tag{3.23}$$

donde c es una constante que define la amplitud del escalón. En el caso particular de que $c = 1$, la función recibe el nombre de escalón unitario.

En los sistemas de control, la entrada del sistema representa la referencia de la variable

a controlar. En otras palabras, es el valor que se desea tome la variable a controlar. Una referencia del tipo escalón significa que, a partir de un tiempo inicial (normalmente 0), se quiere que la salida del sistema tome un valor determinado y se mantenga allí. Éste es precisamente el caso para la tarea de regulación de posición de un robot, que es el tema de este trabajo.

3.3.1. Sistemas de segundo orden

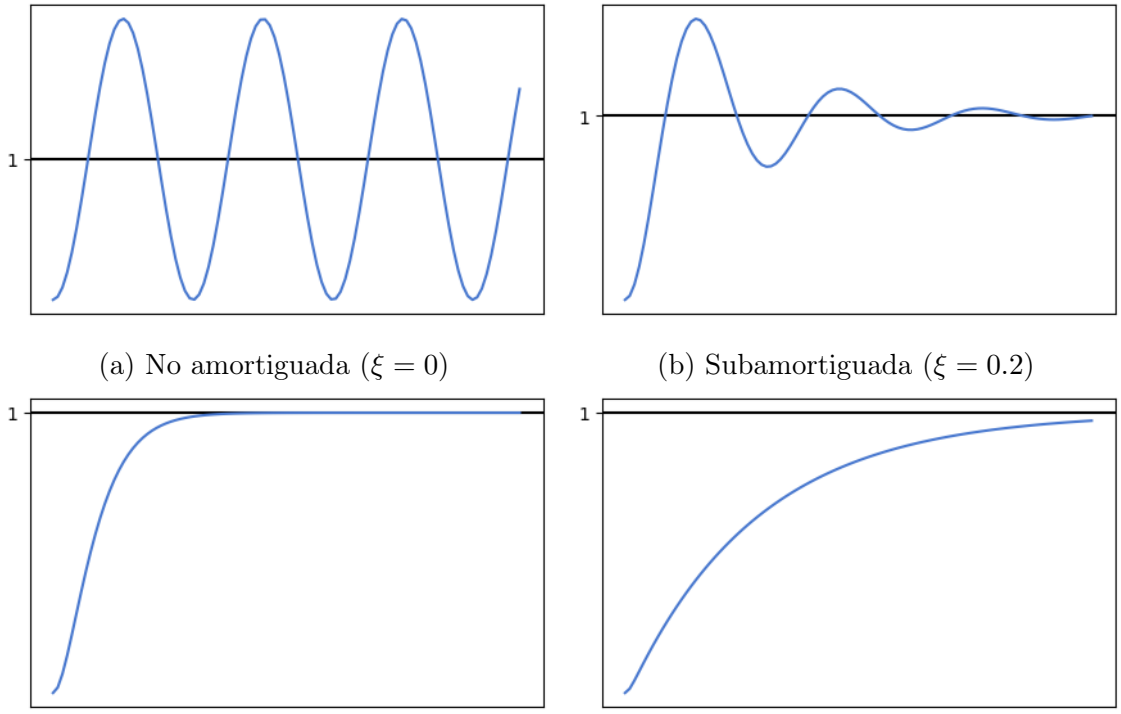
Los sistemas de segundo orden son aquellos que, como su nombre sugiere, pueden modelarse mediante ecuaciones diferenciales de segundo orden.

Un sistema de segundo orden en realidad es una simplificación; los procesos que ocurren en el mundo real normalmente son de órdenes superiores. La importancia de estudiar sistemas de orden 2 radica en que todos los sistemas de orden superior pueden aproximarse como uno de segundo orden. Éstos presentan todos los tipos de respuesta que un sistema dinámico puede tener y los órdenes más grandes solamente varían ligeramente respecto a dichas respuestas.

Existen cuatro tipos de respuestas que puede presentar un sistema estable de segundo orden: no amortiguada, subamortiguada, críticamente amortiguada y sobreamortiguada. Las gráficas de salida contra tiempo de todas ellas se muestran en la figura 3.3. El tipo de respuesta que tendrá un sistema dinámico dado depende de un parámetro llamado *coeficiente de amortiguamiento* ξ, el cual, de manera intuitiva, es una medida de la manera en la que las oscilaciones de un sistema decaen con el tiempo. Entre más alto sea este valor, las oscilasciones decaen más rápidamente. El valor exacto de este parámetro puede obtenerse de la ecuación diferencial que modela al sistema.

Cuando $\xi = 0$, se presenta la respuesta no amortiguada. En este caso, el sistema permanece oscilando alrededor de un valor central y nunca llega a estabilizarse en un punto. Por esta razón, los sistemas que presentan esta tipo de respuesta no son deseables en un sistema de control, puesto que en ningún momento alcanzan la referencia dada.

Al incrementar el coeficiente de amortiguamiento, $(0 < \xi < 1)$, la respuesta oscilatoria comienza a exhibir un decaimiento exponencial, debido al cual el sistema eventualmente se estabilizará en el valor central, dando lugar a la respuesta subamortiguada. Dependiendo del tipo de aplicación, este tipo de sistemas puede o no ser aceptable en un sistema de control,



(a) No amortiguada ($\xi = 0$)

(b) Subamortiguada ($\xi = 0.2$)

(c) Críticamente amortiguada ($\xi = 1$)

(d) Sobreamortiguada ($\xi = 3$)

Figura 3.3: Tipos de respuesta de sistemas de segundo orden.

ya que el sobrepaso inicial puede resultar en daños a equipo o personas.

En el caso particular de $\xi = 1$, el decaimiento de la respuesta es tan fuerte que toda oscilación de la respuesta desaparece. Este tipo de respuesta se llama críticamente amortiguada, ya que es el punto crítico del factor de amortiguamiento en el cual el sistema se estabiliza en el valor final en tiempo más corto. En sistema de control éste es el escenario más deseable debido a que el sistema alcanza el valor de referencia en el menor tiempo físicamente posible y no presenta oscilaciones que pudieran llegar a ser perjudiciales.

Si el factor de amortiguamiento continúa incrementándose ($\xi > 1$), se presenta el caso de la respuesta sobreamortiguada, muy similar a la situación anterior, con la diferencia de que el tiempo que tarda el sistema en alcanzar su valor final es mayor. Conforme aumenta ξ, este tiempo va siendo mayor, aunque en todos los casos el sistema eventualmente alcanza el valor final. Este tipo de respuesta también es deseable en sistemas de control porque tiene las mismas características que la respuesta críticamente amortiguada.

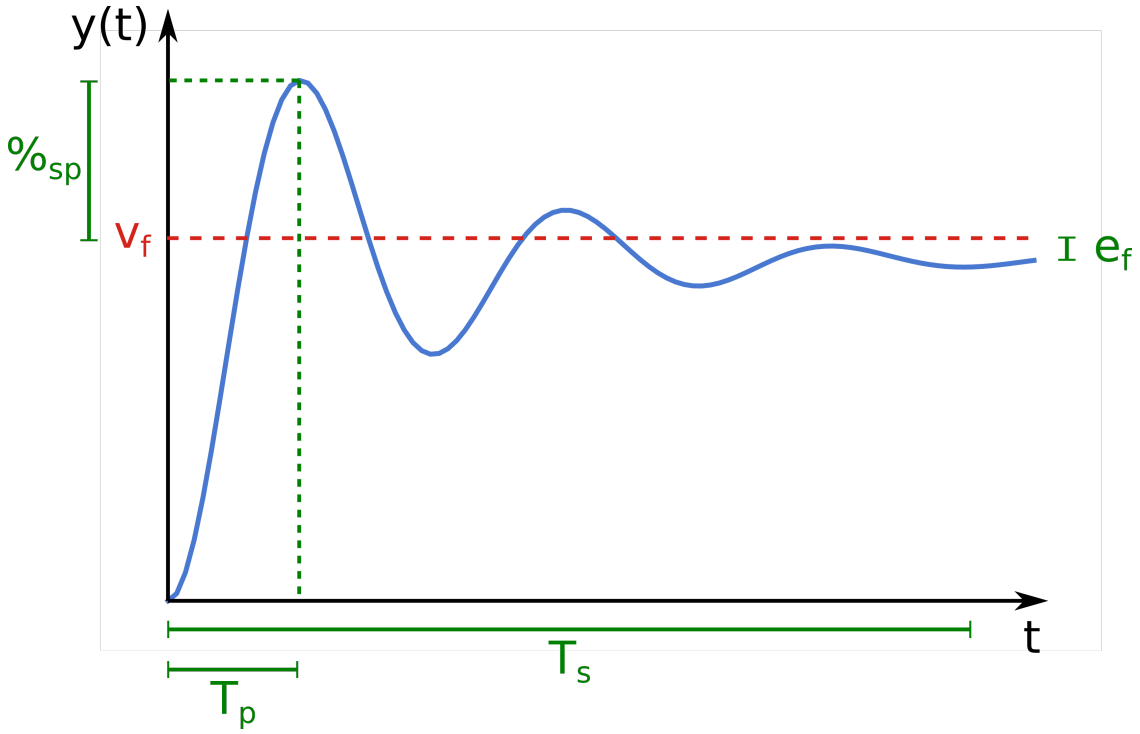


Figura 3.4: Visualización de algunos parámetros que describen a la respuesta general de segundo orden.

3.3.2. El sistema general de segundo orden

La forma de la respuesta de los sistemas de segundo orden puede describirse mediante un conjunto de parámetros que pueden calcularse a partir del modelo del sistema. Además, estos parámetros son fácilmente observables y medibles en la gráfica de la respuesta del sistema, por lo que su significado es muy intuitivo. A continuación se explican algunos de los más importantes, los cuales también pueden verse representados en la figura 3.4.

- **Frecuencia natural (ω_n).** Es la frecuencia de las oscilaciones del sistema sin amortiguamiento. Esta medida solamente puede observarse en sistemas no amortiguados y subamortiguados, que son los únicos que presentan oscilación.

- **Valor final (v_f).** Es el valor en el cual el sistema se estabiliza cuando $t \to \infty$. Nótese que este parámetro no tiene sentido en el caso no amortiguado, ya que el sistema nunca se estabiliza.

- **Error en estado estable (e_f).** En el caso de sistemas de control, es la diferencia

entre el valor final de la respuesta y la referencia. Es deseable que este parámetro sea cero, para que el sistema alcance la referencia dada.

- **Tiempo de asentamiento (T_s).** Es el tiempo requerido para que la respuesta alcance y se mantenga dentro de un rango de $\pm 2\,\%$ del valor final.

- **Porcentaje de sobrepaso ($\%_{sp}$).** La cantidad que la respuesta excede el valor final en su punto más alto, dada como porcentaje del valor final. Esta medida solamente es válida en el caso subamortiguado, porque es la única que exhibe oscilaciones y se estabiliza en un valor final.

- **Tiempo pico (T_p).** El tiempo que tarda la respuesta en alcanzar su primer pico. De igual manera, solamente aplica en el caso subamortiguado.

Capítulo 4

Un agente para el control de brazos robóticos

En este capítulo se detalla el modelo propuesto para el control adaptativo de brazos robóticos, así como su método de entrenamiento.

4.1. Arquitectura del agente

En esta sección se explican los componentes que conforman al agente inteligente y cómo éstos interactúan entre sí. Esta estructura está basada en la presentada con el algoritmo DDPG en [13], el cual es un método actor-crítico. Esto significa que consta de dos elementos principales: el actor, que aproxima la política del agente y el crítico, que aprende el valor de las acciones seleccionadas por el actor.

4.1.1. Actor

Formalmente, el actor, como parametrización de la política, tiene como objetivo seleccionar una o más acciones a realizar dependiendo del estado en el que se encuentre el mundo. En este trabajo, la política es determinista; por lo tanto, el actor es una función de la forma $\mu : S \rightarrow A$, parametrizada por un vector $\boldsymbol{\theta}^{\mu}$.

En este trabajo, la forma del actor seleccionada fue una red neuronal cuya entrada es

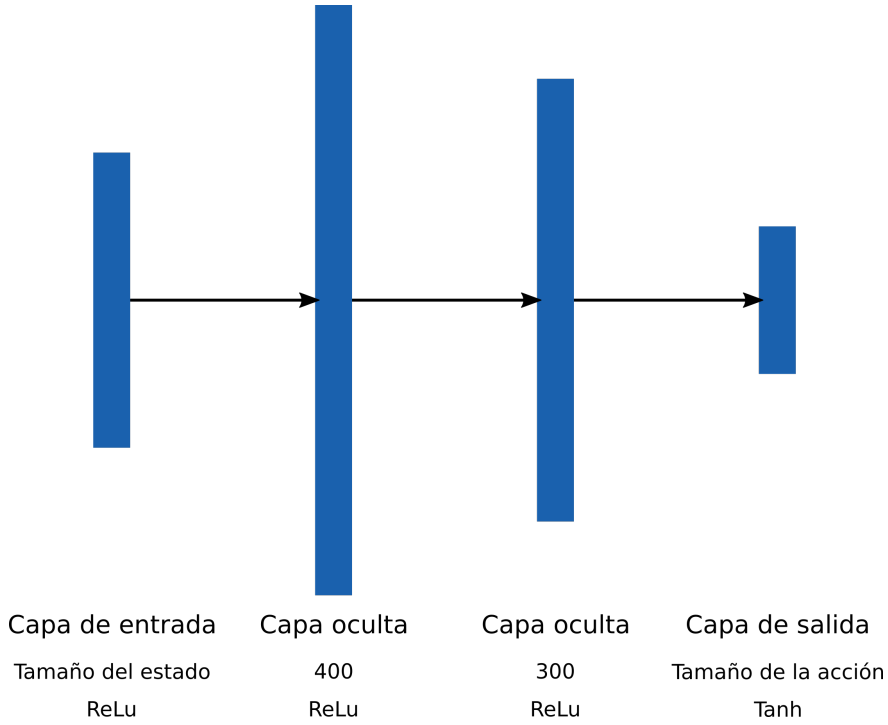


Figura 4.1: Arquitectura del actor.

el vector de estado actual del entorno y cuya salida es el vector de acción seleccionado. La arquitectura concreta se muestra en la figura 4.1. Ésta consta de una capa de entrada, una de salida y dos capas ocultas, donde en todas ellas existe una conectividad completa; es decir, todas las neuronas de la capa se conectan con todas las neuronas de la capa siguiente. La capa de entrada del actor es la que representa el estado del entorno. Por lo tanto, el número de neuronas en esta capa dependerá del tamaño del vector de estado. Esta cantidad está fija para un entorno determinado, aunque varía de un ambiente a otro. Las capas ocultas tienen un tamaño de 400 y 300 unidades, respectivamente. Dichas cantidades fueron sugeridas en el artículo original y, aunque se ha visto que tamaños menores también ofrecen buenos resultados, con dinámicas más complejas presentan problemas para aprender una política aceptable. Finalmente, la capa de salida del actor tiene el mismo tamaño que el vector de acciones para el entorno dado. De igual manera que el estado, este tamaño es una propiedad del entorno y por lo general corresponde con el número de grados de libertad del robot.

La capa de entrada y ambas capas ocultas tienen una activación lineal rectificada, mientras que a la capa de salida se le asignó una activación tangencial hiperbólica. Esto es porque

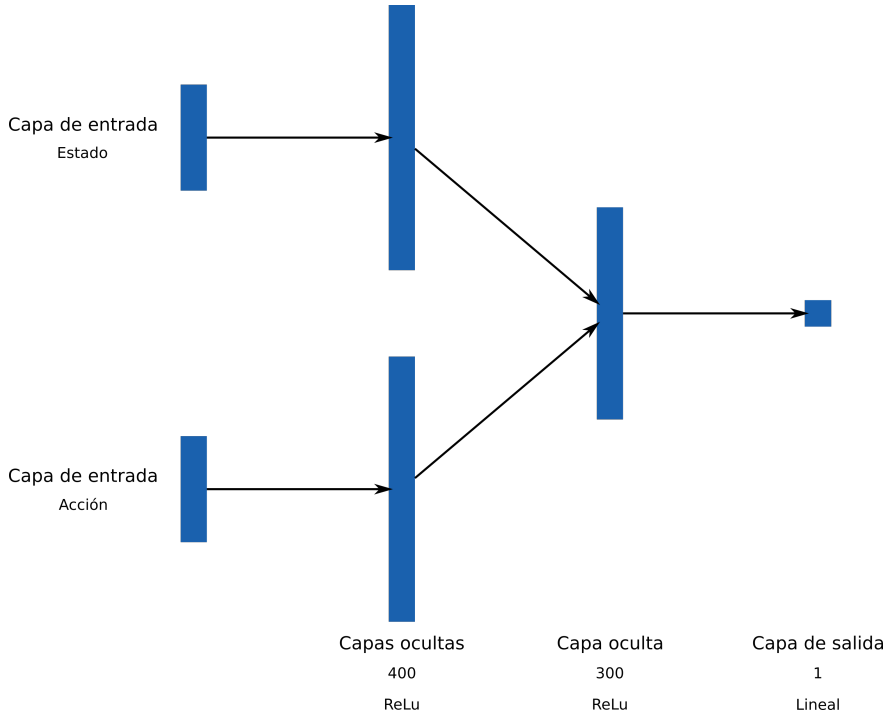


Figura 4.2: Arquitectura del crítico.

los espacios de acciones están acotados en todas sus dimensiones y un entorno dado no puede recibir una acción que esté por debajo del límite mínimo o por arriba del límite máximo. Esto se corresponde con las condiciones de un robot real, donde no puede aplicarse a un motor un voltaje más grande del especificado por el fabricante en ninguna de las direcciones.

4.1.2. Crítico

El crítico es el encargado de aproximar la función de valor de acciones del agente, lo cual significa que debe relacionar una pareja de estado-acción con un valor escalar. Debido a que tiene más de una entrada, la arquitectura del crítico es un poco más compleja que la del actor.

En primer lugar, la red neuronal crítica tiene dos capas de entrada: una para recibir el estado del entorno y la segunda destinada a la acción del agente. Cada una de estas capas de entrada se conecta de manera independiente a una capa densa de 400 neuronas con función de activación lineal rectificada. Después de esta capa ambos vectores de características tienen el mismo tamaño, por lo que es posible sumarlos. De esta manera se tiene un solo vector de

400 elementos, el cual se alimenta a la segunda capa densa, la cual es única. Esta segunda capa oculta tiene 300 unidades y una función de activación lineal rectificada. Finalmente, el vector resultante pasa por la capa de salida de una sola unidad con una función de activación lineal. Esto es porque la salida del crítico debe ser escalar y además no puede estar acotada, ya que no existe un límite al valor que puede tener una determinada acción.

El mismo criterio se aplica tanto para el actor como el crítico para la inicialización de los pesos; para todas las capas excepto la última, los pesos iniciales se obtienen al muestrear una distribución de probabilidad uniforme en el intervalo $\left[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}\right]$, donde f es el número de neuronas en la capa anterior. En contraste, la última capa de ambas redes se inicializa a partir de una distribución uniforme en un intervalo muy cercano a cero. Esto es para asegurar que las acciones al inicio del entrenamiento del agente sean también muy pequeñas. Para el actor el intervalo es de $[-3 \times 10^{-3}, 3 \times 10^{-3}]$, mientras que los pesos del crítico se obtienen del rango $[-3 \times 10^{-4}, 3 \times 10^{-4}]$.

4.1.3. Proceso de ruido

En el aprendizaje por refuerzo es muy importante mantener un equilibrio entre la exploración de acciones que no se han intentado antes y la explotación de la experiencia que se ha adquirido hasta el momento. En este trabajo la forma en la que se realiza exploración de nuevas estrategias es mediante la perturbación de la acción seleccionada con ruido sintético. En otras palabras, se genera un vector con números aleatorios del mismo tamaño que el vector de acción. Posteriormente, a la acción obtenida por parte del actor se le suma este vector de ruido, lo que resulta en una acción similar a la seleccionada por la política, pero aun así novedosa. Por supuesto, la distribución de la cual se muestrea el vector de ruido debe ser cuidadosamente diseñada, de tal manera que la magnitud de la acción original no sea opacada por el ruido.

Naturalmente, cada entorno específico requiere de acciones en escalas distintas; un robot pequeño puede requerir torques de 10 Nm, pero un robot más grande podría desarrollar torques de 1000 Nm en sus motores. En un principio esto implica que la escala del ruido debe ser elegida específicamente para cada entorno. En este trabajo se adopta una estrategia para evitar este problema: debido a que la última capa del actor tiene una activación tangencial

hiperbólica, su salida estará siempre dentro del rango $(-1, 1)$. Antes de ejecutar la acción seleccionada se realiza una operación de escala para adaptar la salida del actor al rango adecuado de acción para el entorno. Si el ruido se genera y se agrega a la acción antes de realizar esta operación de escala, entonces ésta se encargará de adaptar el ruido al rango adecuado para cada entorno. Entonces la magnitud del ruido se conceptualiza no en sus unidades finales, sino como una fracción del rango total posible.

Cuando se trabaja con entornos robóticos existen ciertas restricciones prácticas que se deben atender a la hora de diseñar el agente. Una de las más importantes es el minimizar los impactos que reciben los motores, es decir evitar cambios bruscos en el torque que se aplica a los mismos, ya que esto puede dañarlos. Es por este motivo que en el presente trabajo se utilizó un proceso de ruido correlacionado temporalmente para la exploración de acciones en lugar de muestrear una distribución estática, como una distribución normal. Existen múltiples procesos estocásticos, pero el empleado aquí es el proceso Ornstein-Uhlenbeck [28], que se basa en el movimiento de partículas en suspensión, llamado *Browniano*. Este proceso se define mediante la ecuación diferencial:

$$dx_t = \theta(\mu - x_t)dt + \sigma\mathcal{N}(0, dt) \tag{4.1}$$

donde θ y σ son parámetros y $\mathcal{N}(\mu_N, \sigma_N)$ denota la distribución normal con media μ_N y desviación estándar σ_N. La variable x representa el estado del sistema; puede ser un escalar o un vector. La letra μ denota otro parámetro del proceso, que indica el valor central de la variable x: ésta se moverá alrededor de este punto. Se encontró que los valores $\theta = 0.15$, $\sigma = 0.2$ funcionan bien para todos los entornos.

4.1.4. Búfer de repetición

La amplia mayoría de métodos de optimización para redes neuronales, incluido el descenso del gradiente, requieren que los ejemplos que se utilicen para el entrenamiento sean independientes e idénticamente distribuidos. Lo anterior significa que no debe existir correlación entre ellos o, en otras palabras, que la información contenida en un ejemplo de entrenamiento dado no debe permitir deducir cómo será el siguiente.

Al analizar con detenimiento el problema tratado en este trabajo puede verse cómo es que se rompe la condición del párrafo anterior. Supóngase que, en un momento dado, el robot se encuentra en un estado x. Debido a la frecuencia con la cual se toman los ejemplos de entrenamiento, que están en el orden de los milisegundos, en el instante de tiempo siguiente el robot estará en un estado ubicado en una vecindad $x + \Delta x$ del estado anterior porque éste solamente podrá moverse una pequeña distancia en el tiempo que tarda en tomarse la siguiente muestra. Se tiene, entonces, que el estado actual proporciona mucha información acerca de cuál será el siguiente.

El búfer de repetición es una estructura de datos en la cual se almacena la información vista en anteriores instantes de tiempo. Concretamente, cada entrada en el búfer consta de cinco elementos: el estado en el que se encontraba el mundo en ese momento ($\boldsymbol{s}_t$), la acción que se seleccionó ($\boldsymbol{a}_t$), la recompensa recibida por ejecutar dicha acción (r_t), el estado del mundo después de ejecutar la acción ($\boldsymbol{s}_{t+1}$) y, finalmente, una variable booleana que indica si el instante de tiempo en cuestión fue el último del episodio o no (f_t). Como el agente puede ser ejecutado potencialmente por un periodo muy largo de tiempo, se limita el número de entradas que puede almacenar el búfer. Cuando éste alcanza su capacidad máxima, se eliminan los datos más antiguos para dejar espacio a los nuevos.

Durante la ejecución del agente, las observaciones obtenidas en cada uno de los instante de tiempo se almacenan en el búfer de repetición. Cuando llega el momento de entrenar las redes neuronales, se selecciona un conjunto de entradas del búfer de repetición al azar, con una distribución de probabilidad uniforme. La cantidad de ejemplos seleccionados es constante para todos los pasos de entrenamiento y depende del tamaño de minilote, el cual es un hiperparámetro del algoritmo. Nótese que puede darse el caso de que el búfer no tenga las suficientes entradas para completar un lote de entrenamiento. En tal circunstancia, se omite el entrenamiento en ese instante de tiempo.

El uso del búfer de repetición permite romper la dependencia temporal entre los ejemplos de entrenamiento ya que, al ser obtenidos aleatoriamente, no se alimentan al algoritmo de entrenamiento en secuencia y un ejemplo de entrenamiento dado no contiene información del siguiente. De esta manera es posible entonces aplicar los algoritmos mencionados en el capítulo 3.

4.1.5. Redes objetivo

La ecuación 3.22 es utilizada para la actualización de los pesos del actor. Nótese que en dicha expresión deben evaluarse tanto el crítico como el actor para obtener el nuevo valor de los parámetros. Sin embargo, durante el entrenamiento la salida de ambas redes neuronales cambia mucho de una iteración a otra y con ellas, las actualizaciones de los pesos. En consecuencia, el proceso de aprendizaje puede tener problemas para converger o incluso puede llegar a divergir.

Para evitar inestabilidades durante el entrenamiento, se crean copias de ambas redes neuronales en el agente (actor y crítico). A estas copias se les llama *redes objetivo*. Los pesos de estas redes se calculan como una media móvil exponencial de los pesos de las redes originales, de manera que la regla de actualización de pesos para las redes objetivo es:

$$\boldsymbol{\theta'} \leftarrow \tau\boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta'} \tag{4.2}$$

donde $\boldsymbol{\theta'}$ representa los parámetros de la red objetivo y τ es un hiperparámetro que controla la importancia que se le da a las muestras más recientes. El valor seleccionado para este hiperparámetro en este trabajo fue de 0.001.

El uso de la media móvil exponencial atenúa el ruido en los pesos de las redes objetivo, ya que se emplean varias muestras para calcular su valor en un instante de tiempo dado. Además, al estar el promedio pesado exponencialmente, los pesos tienen un retraso significativo respecto a las redes originales, como ocurriría de haberse utilizado una media común. Todas estas propiedades mejoran la estabilidad en la generación de los ejemplos de entrenamiento tanto para el actor como para el crítico y permiten la aplicación directa de las expresiones expuestas en el capítulo 3.

4.2. Algoritmo de entrenamiento

El método utilizado en este trabajo para entrenar el agente se basa igualmente en el propuesto en [13], con algunas modificaciones. La estructura general del mismo se muestra en el algoritmo 2.

Algoritmo 2: Algoritmo de entrenamiento del agente.

Inicializar el actor $\mu(\boldsymbol{s}|\boldsymbol{\theta}^{\mu})$ y el crítico $Q(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}^{Q})$ con pesos aleatorios
Crear las redes objetivo μ' y Q' copiando los pesos de las redes del agente
Crear el búfer de repetición
para *época* $= 1$ **a** E **hacer**
  para *episodio* $= 1$ **a** M **hacer**
    Inicializar el proceso de ruido aleatorio $\mathcal{R}$
    Recibir el estado inicial del entorno $\boldsymbol{s}_1$
    para $t = 1$ **a** T **hacer**
      Seleccionar la acción $\boldsymbol{a}_t = \mu(\boldsymbol{s}_t) + \mathcal{R}_t$
      Ejecutar la acción $\boldsymbol{a}_t$ y recibir la recompensa r_t y el nuevo estado $\boldsymbol{s}_{t+1}$
      Agregar el conjunto $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}, f_t)$ al búfer de repetición
    fin
  fin
  para $p = 1$ **a** P **hacer**
    Seleccionar aleatoriamente N ejemplos $(\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}_{i+1}, f_i)$ del búfer de repetición
    $y_i \leftarrow r_i + \gamma Q'(\boldsymbol{s}_{i+1}, \mu'(\boldsymbol{s}_{i+1}))$ Entrenar el crítico mediante descenso del gradiente con pérdida:

$$L = \frac{1}{N} \sum_i (y_i - Q(\boldsymbol{s}_i, \boldsymbol{a}_i))^2$$

    Entrenar el actor mediante el gradiente de la política:

$$\nabla_{\boldsymbol{\theta}^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(\boldsymbol{s}, \boldsymbol{a})|_{\boldsymbol{s}=\boldsymbol{s}_i, \boldsymbol{a}=\mu(\boldsymbol{s}_i)} \nabla_{\boldsymbol{\theta}^{\mu}} \mu(\boldsymbol{s})|_{\boldsymbol{s}_i}$$

    Actualizar las redes objetivo:

$$\boldsymbol{\theta}^{\mu'} \leftarrow \tau \boldsymbol{\theta}^{\mu} + (1 - \tau) \boldsymbol{\theta}^{\mu'}$$
$$\boldsymbol{\theta}^{Q'} \leftarrow \tau \boldsymbol{\theta}^{Q} + (1 - \tau) \boldsymbol{\theta}^{Q'}$$

  fin
fin

Este algoritmo se basa en la estructura básica que se presentó en la sección 3.2.3. Ésta se compone de dos etapas principales: la evaluación de la política y la actualización de la política, las cuales pueden distinguirse fácilmente en el algoritmo 2. A cada iteración del algoritmo, que contiene una instancia de cada una de las etapas se le conoce en este trabajo como *época*.

En cada época se ejecuta en primer lugar la etapa de evaluación de la política, que en este caso consiste en ejecutar el agente dentro del entorno simulado de acuerdo a la política que ha aprendido hasta el momento. Durante esta etapa los mecanismos de aprendizaje se desactivan, de tal manera que la política permanezca estática durante la evaluación. Es importante resaltar que a las acciones ejecutadas durante esta etapa se les agrega el ruido de exploración correspondiente, explicado en la sección 4.1.3. Todos las acciones seleccionadas y las transiciones de estado vistas se agregan al búfer de repetición para ser utilizadas durante el entrenamiento.

Posterior a la fase de evaluación se ejecuta la etapa de entrenamiento del agente, en la cual no se ejecuta ninguna acción; se utilizan los ejemplos guardados en el búfer de repetición para obtener un lote con el cual se actualizan el actor y el crítico de acuerdo a las ecuaciones presentadas en el capítulo 3. Asimismo se actualizan las redes objetivo con los nuevos valores de los pesos de las redes originales.

Durante una época se ejecutan varias iteraciones de evaluación de la política y un número diferente de iteraciones de entrenamiento del agente. La cantidad de pasos para cada una de estas etapas son parámetros ajustables del algoritmo. Para propósitos de este trabajo se utilizaron 100 iteraciones de evaluación y 50 iteraciones de entrenamiento, a diferencia del algoritmo original, que realizaba una iteración de cada fase. El balance entre estos dos parámetros impacta en el tiempo de ejecución del algoritmo ya que, por lo general, el entrenamiento del agente es una operación más costosa que la mera evaluación de la política. Así, al emplear más pasos de evaluación que de entrenamiento puede reducirse el tiempo de todo el proceso.

4.3. Regulación del entrenamiento del agente

Uno de los requisitos indispensables para el correcto funcionamiento de un agente de aprendizaje por refuerzo es la exploración; es tan importante que, si no se considera la exploración de nuevas acciones por parte del agente, es imposible que éste pueda aprender. En este caso, la exploración está dada por el ruido que se agrega a las acciones. Sin embargo, las acciones ruidosas son indeseables cuando el agente está ya realizando su tarea después de haber sido entrenado. La manera convencional en la que se evita este problema es mediante la separación de las fases de entrenamiento y producción del agente: en primera instancia éste se entrena mediante algún algoritmo de aprendizaje por refuerzo y una vez que se alcanza un rendimiento aceptable se extrae el actor (que contiene la política aprendida) y se implementa en el entorno de producción, descartando las partes de la arquitectura que posibilitan el aprendizaje. En este momento, como el aprendizaje se detiene, se vuelve innecesario el uso del ruido de exploración.

No obstante, el objetivo principal de este trabajo es la creación de un controlador que pueda adaptarse a entornos desconocidos y para lograrlo es necesario continuar el aprendizaje aún en el entorno de producción. En consecuencia, en este trabajo se propone un método para regular el balance exploración-explotación en línea, es decir, simultáneamente a la ejecución del controlador, de manera que se incremente el ruido de exploración cuando el agente aún esté aprendiendo la dinámica del entorno y se reduzca una vez que el proceso de aprendizaje termine.

4.3.1. Confianza del agente

El grado de exploración del agente se regula a través de una *medida de confianza*, que se actualiza en cada iteración de la fase de ejecución. Esta métrica es una valor escalar, cuyo valor varía entre cero y uno, donde cero representa una política de exploración agresiva y uno indica que el agente ya alcanzó un valor óptimo y la exploración puede detenerse.

El cálculo de la confianza se basa en la recompensa recibida, ya que es una de las formas más directas de medir el rendimiento del agente. A diferencia de los algoritmos de aprendizaje por refuerzo que trabajan de forma episódica, en este caso no es posible construir una curva

de aprendizaje[1], debido a que para ello se requiere crear un entorno específicamente para realizar la evaluación del agente, eliminando el ruido de exploración y el aprendizaje. Como alternativa, se utilizó una media móvil exponencial de la recompensa recibida en cada instante de tiempo. Este promedio ponderado puede verse como una versión en tiempo continuo de la curva de aprendizaje, ya que es proporcional a la recompensa acumulada en los instantes de tiempo más recientes. El método empleado para calcular la media móvil exponencial es análogo al utilizado en el optimizador Adam (ecuación 3.4), solamente reemplazando el gradiente por la recompensa. Para calcular el factor de suavizado para la MME se consideró una expresión utilizada para el análisis de activos financieros:

$$\beta = \frac{2}{T+1} \tag{4.3}$$

donde T es el número de muestras que influyen de manera significativa en el promedio. El valor de β utilizado aquí fue de 0.0001, que corresponde a considerar las últimas 20000 muestras.

A grandes rasgos, la confianza del agente debería disminuir en tanto el aprendizaje continúa. Cuando el agente domina su tarea y el aprendizaje termina, la confianza entonces subirá. En este orden de ideas, este trabajo se basa en la hipótesis de que la recompensa aporta la información suficiente acerca del estado del aprendizaje del agente; cambios fuertes en la tendencia de la recompensa indican un proceso de aprendizaje en curso, mientras que la estabilización de la misma alrededor de un valor significa que el agente alcanzó un punto óptimo en su aprendizaje y éste puede detenerse.

Para detectar la tendencia de la recompensa podría utilizarse una derivada numérica. Sin embargo, este tipo de soluciones resultan demasiado ruidosas para ser utilizadas adecuadamente. En su lugar se empleó un algoritmo de detección de cambios por suma acumulada (CUSUM, por sus siglas en inglés) [29], en el cual se mantiene una señal de cambio S,

[1]En la curva de aprendizaje se grafica la recompensa acumulada de cada episodio contra el número de episodio, de tal forma que se observa la evolución del aprendizaje del agente en el tiempo.

actualizándose en cada instante de tiempo mediante las siguientes ecuaciones:

$$S_0 = 0 \tag{4.4}$$

$$S_{t+1} = \text{máx}(0, S_t + x_t - w_t) \tag{4.5}$$

donde x_t es la muestra vista en el tiempo t y w es un valor de referencia. El cambio se calculará relativo a esta referencia. En este caso se utilizó como referencia la MME de la recompensa; de este modo se calcula el cambio de dicha señal respecto a sí misma, es decir su tendencia. Cuanto más alto es el valor de S, más agresivo es el cambio, ya sea porque la tendencia es muy marcada, o bien porque se ha mantenido por mucho tiempo.

Nótese que las expresiones dadas arriba detectan solamente cambios en la dirección positiva. Para la detección de cambios negativos simplemente debe cambiarse la operación de máximo por un mínimo. Entonces, para detectar ambos tipos de cambio se deben mantener dos señales S separadas: una para cada dirección. Para obtener la magnitud del cambio total se sumó el valor absoluto de ambas señales. La cantidad resultante indica la magnitud del cambio total, sin importar su dirección. Ahora bien, en la definición original del algoritmo CUSUM una vez que la señal de cambio sube, ésta no vuelve a descender sino hasta que se presenta un cambio en la dirección contraria. Este comportamiento es indeseable en este trabajo, ya que muchas veces se presentan varios cambios consecutivos en la misma dirección. Para adecuar la señal a las necesidades del proyecto, se aplicó un suavizado exponencial a la señal S, con el mismo factor de suavizado utilizado para la MME de la recompensa, quedando la expresión final para la actualización de S como sigue:

$$S_{t+1} = 0.0001\,\text{máx}(0, S_t + x_t - w_t) \tag{4.6}$$

Esta señal se utiliza para ajustar la confianza del agente. Para ello se compara con un umbral; si la suma cusum es mayor al dicho umbral significa que está ocurriendo un cambio fuerte en la recompensa. Entonces la confianza se ajusta a la baja restándole una cantidad pequeña en cada instante de tiempo. Además, la confianza se limita en cero para que no tome valores negativos. Por otro lado, si la suma acumulada se encuentra debajo del umbral

establecido, a la confianza se le suma la misma cantidad en cada instante de tiempo y se limita para que no tome valores mayores a 1. La cantidad que se le suma o resta a la confianza en cada instante de tiempo se fijó en 1×10^{-5}, lo que significa que la confianza puede ir de 0 a 1 en 100,000 instantes de tiempo; suficiente para evitar afectar el aprendizaje del agente al realizar cambios muy bruscos.

El umbral adecuado para la suma acumulada cambia según el entorno en el cual actúe el agente. En particular, depende de la función de recompensa utilizada. Para encontrar el valor ideal, la estrategia que se siguió en este trabajo se basó en la prueba y error: por lo general el valor cusum se estabiliza después de algún tiempo de entrenamiento. Los valores de umbral se decidieron de manera que el valor final de cusum quede por debajo del mismo y lo más cerca posible, de modo que cualquier cambio en el entorno dispare el aprendizaje en el agente.

En este trabajo se exploran dos mecanismos a través de los cuales la confianza afecta el aprendizaje del agente. El primero de ellos como ya se dijo es la magnitud del ruido de exploración. El segundo es el tamaño del búfer de repetición. En las siguientes secciones se profundiza en la base lógica y el funcionamiento de ambos.

4.3.2. Ruido de exploración

Aunque necesario durante el aprendizaje, el ruido de exploración es un obstáculo para el correcto funcionamiento del robot y debe ser desactivado durante el funcionamiento normal del mismo. Además, una adecuada evaluación de la política actual debe también ser efectuada en un entorno sin ruido. En consecuencia, el valor de la confianza se utiliza para regular la cantidad de ruido de exploración que se suma a las acciones seleccionadas por la política. En particular, la amplitud del ruido es inversamente proporcional a la confianza del agente: cuando la confianza es 0 la amplitud del ruido es máxima y cuando la confianza tiene un valor de 1 el ruido de exploración desaparece completamente.

Parte de la definición de un entorno de aprendizaje por refuerzo es el espacio de acciones, es decir, cuántas dimensiones tiene el vector de acción y cuáles son los límites superior e inferior de cada una de esas dimensiones. Respetar estos límites es muy importante para evitar daños al robot cuando el agente se implemente en entornos reales. Por lo tanto,

adicionalmente a la escala del ruido de exploración también se escala la acción seleccionada por el agente, de la siguiente manera:

$$\boldsymbol{a}_t = \frac{1}{2}(1 + C) \cdot \mu(\boldsymbol{s}_t) + \frac{1}{2}(1 - C) \cdot \mathcal{R}_t \tag{4.7}$$

donde C es la confianza del agente. De acuerdo con esta expresión, cuando la confianza es máxima ($C = 1$), la política toma un factor de escala de 1, mientras que el ruido es multiplicado por cero. Conforme disminuye la confianza, la escala de la política disminuye y el ruido aumenta. Cuando la confianza es igual a cero, la política conforma la mitad de la acción resultante y el ruido de exploración la otra mitad. Es importante que la política siempre tenga un impacto sobre la acción final, ya que el aprendizaje requiere una cierta cantidad de explotación. Durante el desarrollo de este proyecto se encontró que el límite de 1/2 en la escala de la acción de la política funciona bien para todos los entornos considerados.

Debido a la activación tangencial hiperbólica, la salida del actor está acotada y en ninguna de sus dimensiones su valor superará la unidad. Por otro lado, el ruido, al ser un proceso estocástico, no puede acotarse de manera estricta. Sin embargo, sí puede mantenerse entre -1 y 1 con una muy alta probabilidad. La definición dada en este mismo capítulo obedece esta restricción. Entonces puede verse que, al efectuar la operación de la ecuación 4.7 y no importando el valor de C, la acción tampoco superará la unidad. Luego, al escalar la acción para adecuarla al espacio de acciones del entorno, no se superarán los límites establecidos.

4.3.3. Tamaño del búfer de repetición

La experiencia del agente se almacena en el búfer de repetición conforme éste realiza su tarea para posteriormente ser utilizada para entrenar al actor y al crítico. En cierto sentido, entonces, puede decirse que el aprendizaje del agente va retrasado respecto a su experiencia, ya que potencialmente una muestra dada de experiencia puede utilizarse varios miles de pasos después de haberse generado. En consecuencia, al producirse una alteración en el entorno y comenzarse a generar muestras con la nueva dinámica, pasará un tiempo considerable antes de que éstas comiencen a ser seleccionadas para el entrenamiento. El resultado final será que el agente tardará mucho en aprender la nueva dinámica del entorno y en adaptarse a ella.

Para acelerar el aprendizaje posterior a una perturbación se propone utilizar la confianza para ajustar el tamaño del búfer de repetición, aprovechando el hecho de que las nuevas muestras reemplazan a las antiguas cuando el búfer se encuentra lleno. El tamaño del búfer se calcula directamente a partir del valor de la confianza, mediante la siguiente expresión:

$$N_{buf} = 50,000 + (1,000,000 - 50,000) \cdot C \tag{4.8}$$

El búfer está diseñado para que, cuando la confianza se reduzca a cero, tenga un tamaño de 50,000. Esto es parar descartar rápidamente las muestras antiguas cuando se produce una perturbación, que ya no representan la dinámica del sistema y así agilizar el aprendizaje. Cuando la confianza vuelve a aumentar, el tamaño del búfer se incrementa igualmente, hasta un máximo de 1,000,000, con el fin de estabilizar el controlador en la nueva dinámica.

Capítulo 5

Entornos simulados

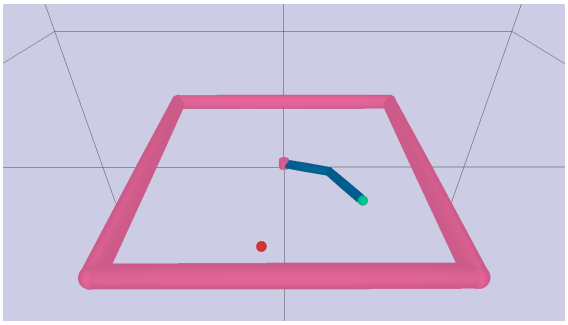
El agente descrito en el capítulo 4 se evaluó en tres diferentes entornos con brazos robóticos simulados. El simulador sobre el cual se implementaron los entornos es Roboschool, propuesto en [30]. Este simulador es de código abierto [1] y es compatible con varias bibliotecas y otros simuladores utilizados en aprendizaje por refuerzo. El motor físico que utiliza es Bullet, también de código abierto [2].

Todos los entornos consisten en un brazo robótico, cuya tarea es posicionar el extremo de su último eslabón (la herramienta, o el efector final) lo más cerca posible de un punto objetivo dado y mantenerlo en dicha posición. Tanto la posición inicial del robot como la posición del punto objetivo se seleccionan aleatoriamente al comenzar cada episodio, los cuales duran 100 pasos. Los entornos varían según si el robot está contenido en un espacio bidimensional o tridimensional y en el número de grados de libertad del mismo, o GDL. Adicionalmente, y con el objetivo de evaluar la capacidad de adaptación del controlador, para cada uno de los entornos se implementaron varios escenarios de perturbación, los cuales varían en intensidad: desde un cambio en los parámetros del sistema, como la modificación de una longitud, hasta la alteración completa de la forma del modelo, como la eliminación de un motor.

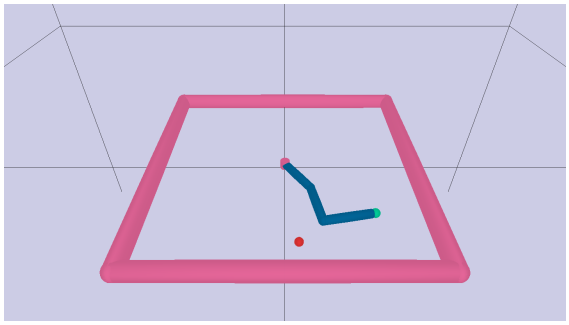
El código de todos los entornos se encuentra en el mismo repositorio que el agente, disponible en `https://github.com/LuisLaraP/OnlineLearningController`.

En la figura 5.1 se muestran imágenes renderizadas de los tres entornos implementados,

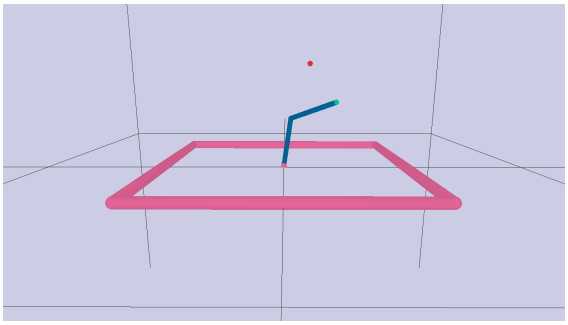
[1] Disponible en `https://github.com/openai/roboschool`
[2] Disponible en `https://github.com/bulletphysics/bullet3`



(a) Bidimensional 2 GDL.



(b) Bidimensional 3 GDL.



(c) Tridimensional 3 GDL.

Figura 5.1: Entornos virtuales utilizados para evaluar el agente.

que a continuación se describen.

5.1. Bidimensional 2 GDL

El primero y más simple de los entornos implementados es un robot de 2 grados de libertad que está contenido en un plano (figura 5.1a). Consiste en dos eslabones conectados en serie a un pivote central. La primera articulación puede rotar libremente, dándole al robot la capacidad de alcanzar puntos a todo su alrededor. La segunda articulación está limitada a rotar en el rango [-180°, 180°]. Ambos eslabones tienen una longitud de 0.1 unidades.

En este entorno el estado del mundo está dado por un vector con los siguientes elementos:

- La posición del punto objetivo (2 coordenadas).
- El vector desde el efector final del robot hasta el objetivo (2 coordenadas).
- El seno del ángulo de la primera articulación.
- El coseno del ángulo de la primera articulación.

- La velocidad angular de la primera articulación.
- La posición y velocidad angular de la segunda articulación.

5.2. Bidimensional 3 GDL

Este entorno se basa en el explicado arriba y se muestra en la figura 5.1b. Se trata igualmente de un robot contenido en un plano, pero en este caso es de tres grados de libertad. Esto significa que, para efectos de control de posición, el robot teóricamente puede cumplir su tarea aún si falla uno de sus motores, puesto que tiene más grados de libertad que variables a controlar.

El vector de estado en este entorno es el mismo que en el bidimensional 2 GDL, con el siguiente añadido:

- La posición y velocidad angular de la tercera articulación.

5.3. Tridimensional 3 GDL

En este entorno, ilustrado en la figura 5.1c el robot ya no está contenido en un plano, sino que existe en un espacio de tres dimensiones. Éste tiene también tres articulaciones, dispuestas de la siguiente manera: el primer eslabón del robot está conectado al pivote central mediante dos articulaciones, la primera de las cuales le permite rotar en el plano horizontal de manera continua, sin restricciones. La segunda articulación permite al mismo eslabón rotar hacia arriba, levantándose del plano horizontal. Esta junta está restringida a girar hasta 180°, de modo que el primer eslabón no puede alcanzar puntos con coordenada z negativa. La tercera articulación del robot conecta el segundo eslabón con el primero y está restringida a rotar en el rango [-180°, 180°]. Ambos eslabones tienen una longitud de 0.2 unidades.

Otra diferencia de este entorno con los dos anteriores es el vector de estado, que en este caso es más simple y no contiene información redundante. Consiste solamente en los siguientes elementos:

- La posición del punto objetivo (3 coordenadas).
- La posición angular de todas las articulaciones (3 elementos).
- La velocidad angular de todas las articulaciones (3 elementos).

5.4. Función de recompensa

La función de recompensa es una parte muy importante de la definición de un entorno de aprendizaje por refuerzo. En el caso de los entornos utilizados en este trabajo, tienen todos la misma función de recompensa. Esta forma de función de recompensa, propuesta en [30], tiene la ventaja de que es fácilmente generalizable y escalable a otros problemas de control robótico de aprendizaje por refuerzo.

En un problema de control de posición de brazos robóticos se pueden identificar varios objetivos que deben cumplirse simultáneamente: en primer lugar el efector final del robot debe alcanzar el objetivo, pero también debe hacerlo de la manera más económica posible, es decir, consumiendo la menor cantidad de energía. De esta manera el agente no sólo intentará alcanzar el objetivo lo más rápido que le permitan los motores, sino que tenderá a seguir trayectorias más suaves y con aceleraciones más pequeñas, cosa que siempre es deseable en cualquier robot para alargar la vida útil del mismo.

Como base para el planteamiento de la función de recompensa se propone la siguiente ecuación, que sirve como una medida del «potencial» del robot, donde esta medida es más grande cuanto más cerca se encuentra el efector final del objetivo:

$$E = -100 \cdot \|\boldsymbol{p}_{obj} - \boldsymbol{p}_{ef}\| \tag{5.1}$$

es decir, el potencial se calcula como la norma del vector existente entre el efector final del robot ($\boldsymbol{p}_{ef}$) y el punto objetivo ($\boldsymbol{p}_{obj}$), multiplicada por una constante negativa. El valor de dicha constante es principalmente para adecuar las magnitudes de las diferentes unidades que se ocupan para medir distancias, aunque es importante que sea negativa para que el potencial sea más alto conforme el robot se acerca al objetivo.

A partir del concepto de potencial, y atendiendo el primer objetivo de control mencionado, se puede plantear la siguiente función de recompensa:

$$r_t = E_t - E_{t-1} \tag{5.2}$$

Esta función asigna una mayor recompensa a las acciones que contribuyan de mayor medida a la consecución del objetivo. Además, tiene la ventaja añadida de que es densa; en todos los instantes de tiempo el agente recibe información que ayuda a su aprendizaje.

No obstante, esta función no ataca el segundo objetivo de un sistema de control. Para ello, se agrega a la ecuación anterior un término que aproxima la cantidad de electricidad que consumiría el robot si éste fuera real:

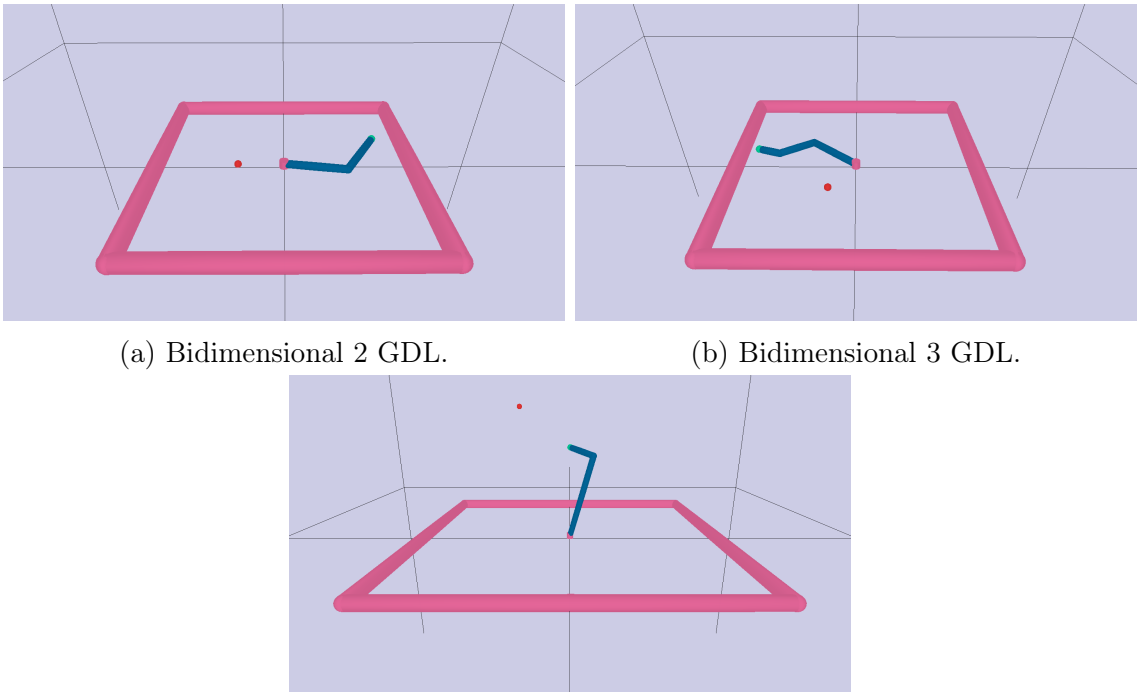
$$r_t = E_t - E_{t-1} - P_t \tag{5.3}$$

donde P_t es la potencia (energía por unidad de tiempo) que consumió el robot en un instante de tiempo dado al realizar su acción. Esta cantidad está dada por:

$$P_t = 0.1 \cdot \sum_{i=0}^{J} \omega_t^i \cdot \tau_t^i + 0.01 \cdot \sum_{i=0}^{J} |\tau_t^i| \tag{5.4}$$

donde la potencia de un motor está dada como el producto de su velocidad angular ω y su torque τ. Además se agrega un segundo término que representa el torque ejercido en una junta cuando ésta se encuentra inmóvil. Ambas cantidades se calculan para todas las juntas y se suman ponderadas por dos contantes que, nuevamente, sirven para ajustar las magnitudes de sus unidades. Nótese que la electricidad aparece también con un signo negativo, con el fin de que la recompensa sea mayor cuando el robot consuma menos energía.

Finalmente, se agrega un término más a la función de recompensa para evitar que las articulaciones se bloqueen, lo que puede causar un consumo excesivo de energía o incluso daños a un robot real. Para ello se implementó una variable B^i que asume un valor de 1 si la articulación i se encuentra cerca de cualquiera de los extremos de su rango de movimiento y



(a) Bidimensional 2 GDL. (b) Bidimensional 3 GDL.

(c) Tridimensional 3 GDL.

Figura 5.2: Perturbaciones de longitud de todos los entornos.

cero en caso contrario. De esta manera, la función de recompensa final resulta como sigue:

$$r_t = E_t - E_{t-1} - P_t - \sum_{i=0}^{J} B_t^i \tag{5.5}$$

5.5. Perturbaciones

Para evaluar la capacidad de adaptación del controlador se implementó una serie de perturbaciones comunes a todos los entornos, las cuales introducen una variación en los parámetros o la forma del modelo del robot. Estos escenarios están pensados para ejecutarse una vez que el agente ha aprendido una política para la versión sin perturbar, pero también pueden aprenderse desde cero.

La primera forma de perturbación consiste en modificar la longitud de uno o más eslabones del robot, tal como se muestra en la figura 5.2 para todos los entornos. En este caso el modelo del robot sigue siendo el mismo, solamente cambia el parámetro de longitud. Por lo tanto, es el escenario en el cual es más sencillo para el agente recuperarse. Los detalles de

Entorno	Eslabón 1	Eslabón 2	Eslabón 3
Bidimensional 2 GDL	+0.05	+0.05	-
Bidimensional 3 GDL	+0.05	+0.0	-0.05
Tridimensional 3 GDL	+0.1	-0.1	-

Tabla 5.1: Cambios en la longitud de los eslabones en cada uno de los entornos.

cómo varían los entornos bajo esta perturbación se proporcionan en la tabla 5.1.

El segundo tipo de perturbación simula una falla en una articulación del robot, provocando que ésta rote libremente. El agente continúa recibiendo la información de la posición y velocidad angular de la junta en cuestión y puede continuar enviando una señal de control a la misma, pero ésta será ignorada. En un entorno dado solamente se aplica esta perturbación a una de las articulaciones, ya que si existiesen demasiadas fallas sería imposible para el robot continuar llevando a cabo su tarea. En particular, la falla se aplica en la segunda articulación en ambos robots bidimensionales. En el caso del entorno tridimensional no se implementó esta perturbación debido a que el robot perdería en gran medida su capacidad de realizar la tarea.

El tercer y último tipo de perturbación que se implementó emula una falla en un motor; una de las articulaciones del robot se deja fija en un ángulo determinado (en este caso 0°). Similarmente a la perturbación anterior, el agente sigue recibiendo la información del estado de la junta, pero no puede actuar sobre ella. Sin embargo, a diferencia del caso anterior, la junta no rota en absoluto; permanece fija. Este es un escenario más realista de lo que sucedería en un robot real si un motor falla. En el escenario tridimensional la junta seleccionada para fallar es la segunda, mientras que en los otros dos entornos falla la misma junta que en la perturbación anterior.

Capítulo 6

Evaluación experimental

En este capítulo se presentan los experimentos a través de los cuales se evaluó el agente propuesto. También se resumen y analizan los resultados obtenidos a partir de dichos experimentos.

En este trabajo se implementaron dos versiones del agente descrito en el capítulo 4: en la primera de ellas la confianza afecta solamente la amplitud del ruido de exploración, mientras que en la segunda versión la confianza afecta también el tamaño del búfer de exploración, de acuerdo a las reglas establecidas en el capítulo 4. Ambas versiones del agente se entrenaron en los mismos entornos de manera idéntica, de acuerdo con el procedimiento descrito a continuación.

En primer lugar, se entrenó el agente en la versión normal de cada uno de los entorno desde cero. El aprendizaje se prolongó, independiente de los resultados o del rendimiento que tuviera el agente, durante 500,000 instantes de tiempo. Posteriormente se tomó el agente entrenado en el entorno normal y se continuó el entrenamiento durante 500,000 pasos de tiempo más en cada uno de los entornos perturbados, de modo que al final se realizaron 3 experimentos para cada entorno (2 en el caso del robot tridimensional).

En cada experimento se recabaron las siguientes métricas, capturadas en todos los instantes de tiempo:

- Acción total, agregando el ruido de exploración.
- Valor de la función de pérdida utilizada para entrenar al crítico.

- Valor de la acción efectuada, es decir, la salida del crítico al ser evaluado con la acción actual.
- Confianza del agente.
- Recompensa observada.

En cada experimento se realizaron 5 ejecuciones diferentes e independientes. Los resultados que se reportan aquí se calcularon promediando los datos obtenidos en cada ejecución. Además del promedio, se obtuvo la desviación estándar de las diferentes métricas, que proporciona información acerca de la dispersión de los datos.

Finalmente, en adición a las métricas presentadas arriba, se calculó en cada experimento la curva de aprendizaje del agente. Debido a que el objetivo de esta curva es evaluar el rendimiento de la política aprendida por el agente, debe ser medida sin la intervención del ruido de exploración. Para ello se creó un entorno separado de evaluación, en el cual cada 100 episodios se pausó el entrenamiento del agente, eliminando el ruido de exploración, la actualización de la confianza y el búfer de repetición, para realizar 5 ejecuciones en el entorno seleccionado con el agente solamente. Estas 5 ejecuciones, de igual manera que las demás métricas, se promediaron y se calculó su desviación estándar para formar la curva de aprendizaje.

6.1. Comparación de agentes

Esta sección está dedicada a comparar el comportamiento durante el entrenamiento de las dos versiones del agente. El primer conjunto de gráficas fueron obtenidas a partir de la versión en la cual la confianza afecta solamente al ruido de exploración. En el segundo conjunto de resultados se muestra el comportamiento del agente que modifica tanto el ruido de exploración como el tamaño del búfer de repetición.

En particular, los resultados que se presentan aquí contienen tres métricas: la primera es la recompensa media (utilizando MME) recibida por el agente a lo largo de su entrenamiento, la segunda es la confianza del agente y la tercera consiste en la curva de aprendizaje. Recordemos que ésta última es la recompensa acumulada por episodio del agente y debe ser medida sin

ruido de exploración. Todas las métricas se midieron sobre las versiones sin perturbar de los tres entornos implementados. Además, en las gráficas aquí presentadas se grafica la media de las 5 ejecuciones llevadas a cabo para cada entorno y la región sombreada se extiende hasta una desviación estándar por arriba y por abajo de la media para dar una idea de la dispersión de la variable.

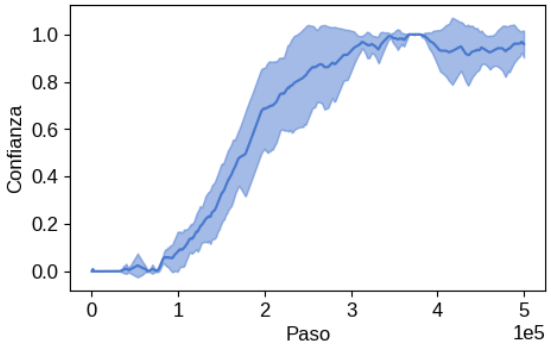
6.1.1. Búfer de repetición fijo

Durante el entrenamiento en el entorno bidimensional de 2 GDL (figura 6.1), puede verse que el rendimiento del agente en un inicio es muy malo, con recompensas mayormente negativas. Sin embargo, rápidamente aprende una buena política, que le permite obtener recompensas que, en promedio, son positivas. Tras aproximadamente 200,000 pasos el aprendizaje se estabiliza en una política aceptable.

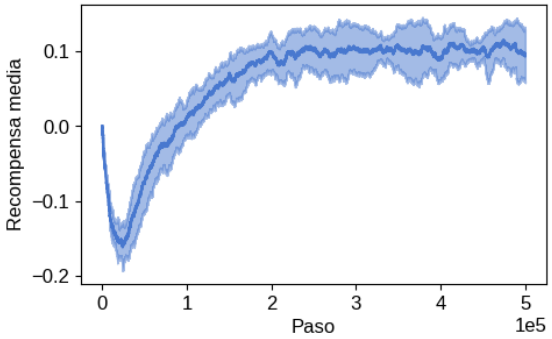
El comportamiento de la confianza del agente refleja el proceso de entrenamiento; al inicio tiene un valor de cero, ya que el agente aún no ha aprendido nada. Posteriormente y conforme la recompensa media se eleva la confianza aumenta también, hasta llegar al máximo valor posible de 1 tras 300,000 pasos aproximadamente. Nótese que existe un momento en el cual la recompensa promedio recibida por el agente se estabiliza, pero la confianza no alcanza su máximo nivel. Este fenómeno se puede observar también en otros experimentos y es deseable ya que, en un escenario en tiempo real, el agente no puede estar seguro de que la recompensa se ha estabilizado hasta después de un tiempo.

Asimismo, la recompensa media exhibe un comportamiento interesante: en los primeros pasos de entrenamiento su valor disminuye, para posteriormente aumentar. En realidad inicialmente la recompensa media recibida es mucho menor, pero para construir estas gráficas se utilizó un algoritmo de suavizado basado en la media móvil exponencial. Una de las debilidades de este algoritmo es que presenta una tendencia al sesgo cuando aún no se tienen suficientes datos. El comportamiento descendente de la gráfica refleja el proceso de adaptación de la media móvil al valor real de la variable medida.

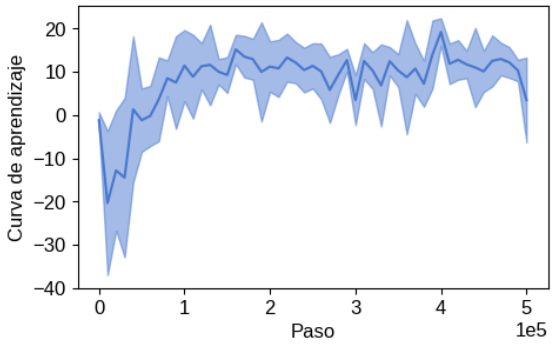
También se observa que, por lo general, existe una baja dispersión en los datos. En otras palabras, todas las ejecuciones son muy similares, lo que demuestra la robustez del algoritmo de aprendizaje, aun considerando su naturaleza estocástica.



(a) Confianza.

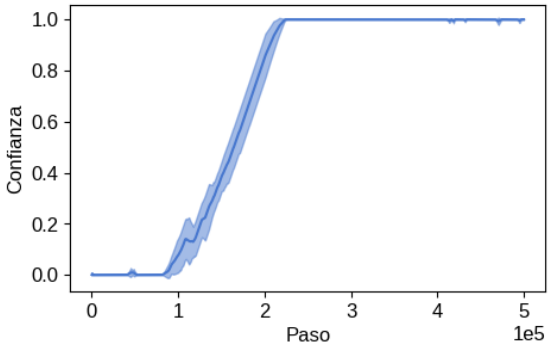


(b) Recompensa media.

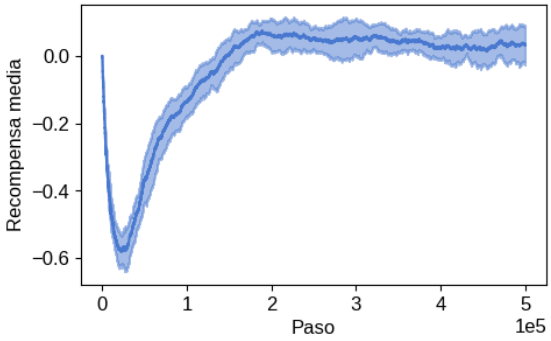


(c) Curva de aprendizaje.

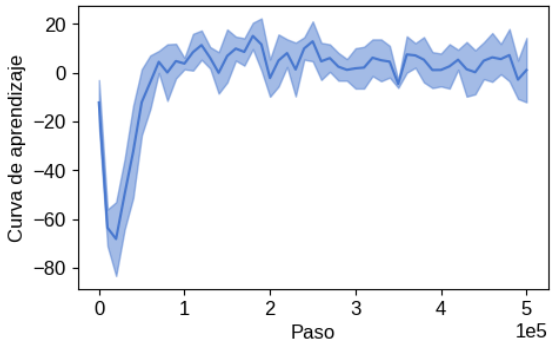
Figura 6.1: Entrenamiento del agente con búfer de repetición fijo en el entorno bidimensional de 2 GDL.



(a) Confianza.



(b) Recompensa media.



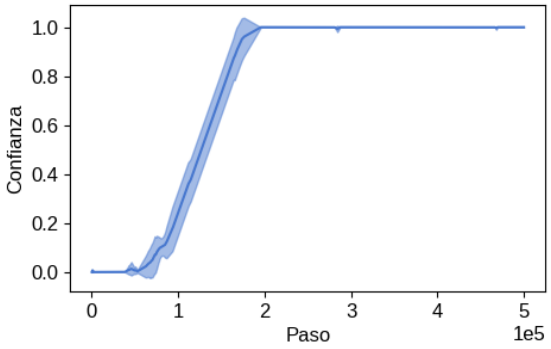
(c) Curva de aprendizaje.

Figura 6.2: Entrenamiento del agente con búfer de repetición fijo en el entorno bidimensional de 3 GDL.

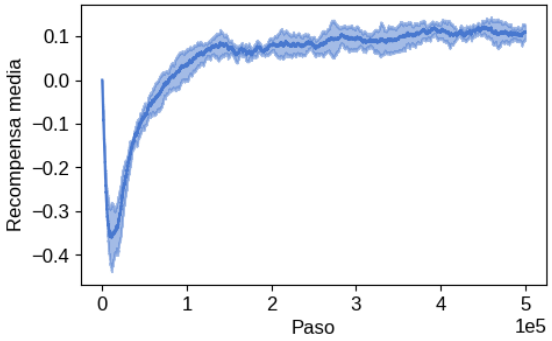
Pasando a un entorno un poco más complicado, el agente en el robot bidimensional de 3 GDL (figura 6.2) exhibe un comportamiento muy similar al anterior, donde inicialmente su desempeño es muy pobre, pero rápidamente se estabiliza en una política aceptable. Por otro lado, en este caso se observa una dispersión mucho más baja en el caso de la confianza y la manera en la que ésta se incrementa es más uniforme, presentando una tendencia casi completamente lineal y alcanzando en poco tiempo su valor máximo.

En este caso, además, la fase transitoria del aprendizaje es también más rápida, alcanzando el agente su política final tras sólo 100,000 pasos, de acuerdo con la curva de aprendizaje. Asimismo se muestra que el valor final de la recompensa media es más bajo que en el entorno de 2 GDL, pero la diferencia es mínima.

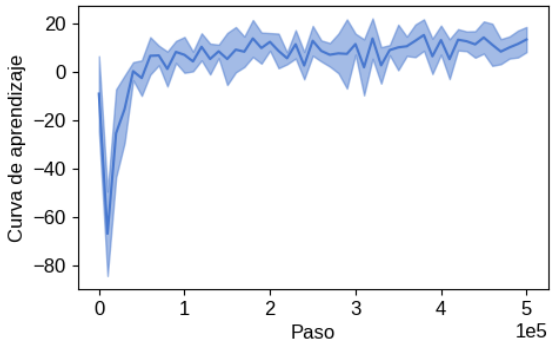
Finalmente, la figura 6.3 muestra los resultados obtenidos en el entorno tridimensional de 3 GDL, los cuales son más parecidos a los del caso bidimensional 3 GDL, posiblemente por el número de grados de libertad. No obstante, en este entorno los fenómenos menciona-



(a) Confianza.



(b) Recompensa media.



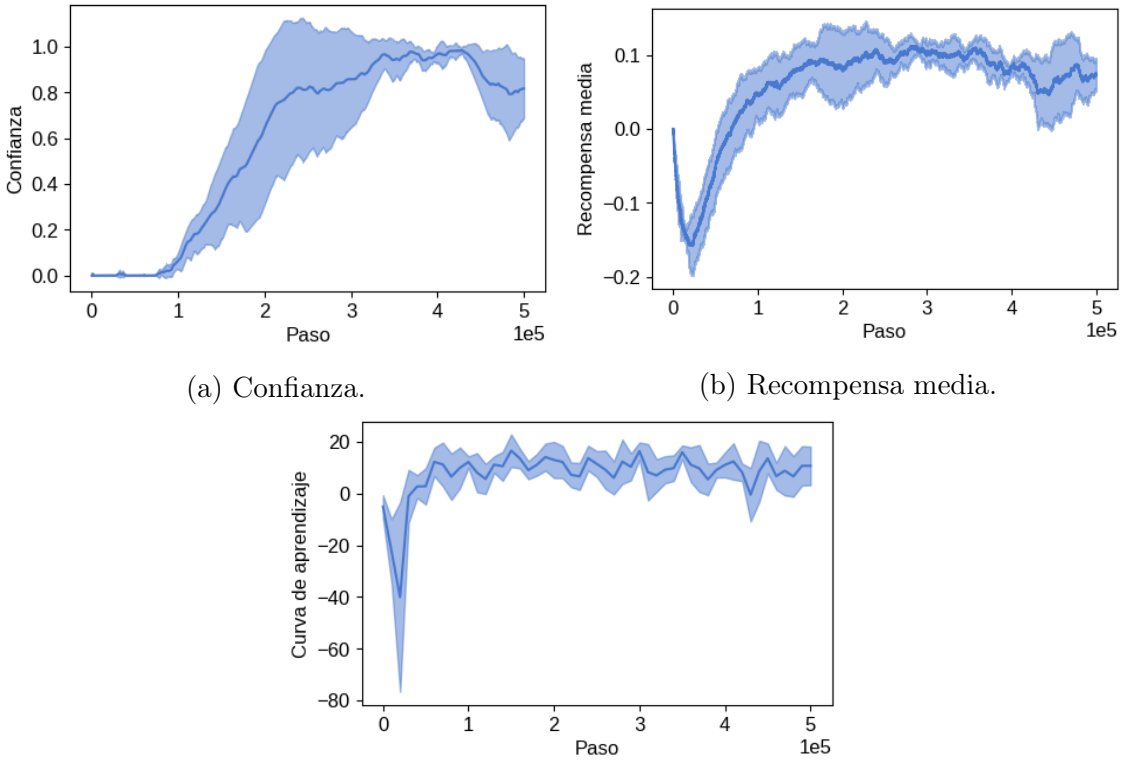
(c) Curva de aprendizaje.

Figura 6.3: Entrenamiento del agente con búfer de repetición fijo en el entorno tridimensional de 3 GDL.

dos anteriormente están más pronunciados: la política final se encuentra muy rápidamente (aproximadamente 50,000 pasos) y el comportamiento de la confianza es muy similar en las cinco ejecuciones que se realizaron. El valor final de la recompensa media es similar a los dos casos anteriores y la diferencia es imperceptible a simple vista.

Adicionalmente, este es el entorno que presenta menor grado de variabilidad entre las diferentes ejecuciones de todos los experimentos de este agente.

Como comentarios finales es importante agregar que en todos los entornos ensayados con este agente, una vez que la recompensa media y la curva de aprendizaje alcanzan su valor final, se mantienen bastante estables a partir de ese punto, mostrando variaciones mínimas. Las magnitudes que toman las variables presentadas en los tres entornos son también muy similares.



(a) Confianza.

(b) Recompensa media.

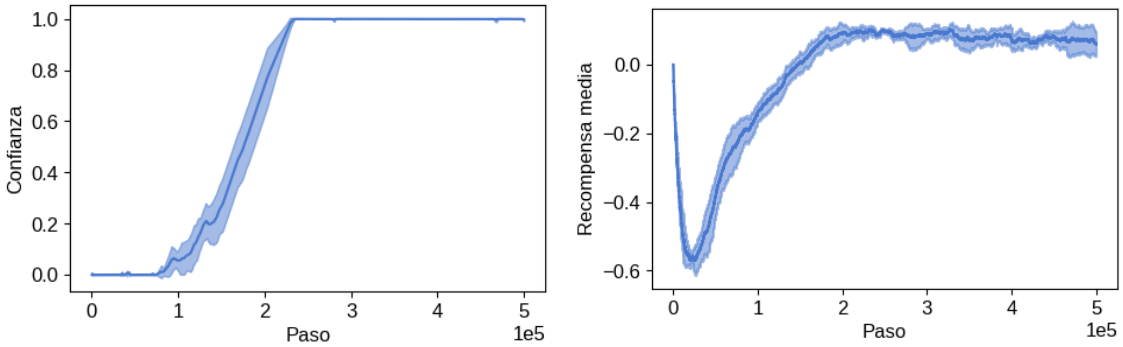
(c) Curva de aprendizaje.

Figura 6.4: Entrenamiento del agente con búfer de repetición variable en el entorno bidimensional de 2 GDL.

6.1.2. Búfer de repetición variable

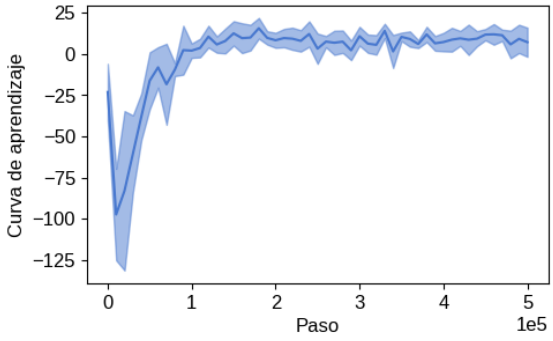
A continuación se revisan los resultados obtenidos por el agente con búfer de repetición variable en los mismos entornos que el agente anterior. En primer lugar se analiza el entorno bidimensional de 2 GDL, mostrado en la figura 6.4. A grandes rasgos tiene un comportamiento similar al agente anterior. No obstante, el aprendizaje se llevó a cabo de forma más rápida, de manera que la curva de aprendizaje llegó a su valor final tras aproximadamente 75,000 pasos. También se observa que, hacia el final del experimento, alrededor de los 450,000 pasos la recompensa media se desestabiliza ligeramente, lo cual provoca que la confianza baje. Asimismo, aunque la confianza sigue la misma tendencia general que en el agente anterior, en este caso se tiene una mayor variabilidad entre las ejecuciones y no se alcanza el valor máximo.

De acuerdo con la figura 6.5, los resultados de este agente en el entorno bidimensional



(a) Confianza.

(b) Recompensa media.

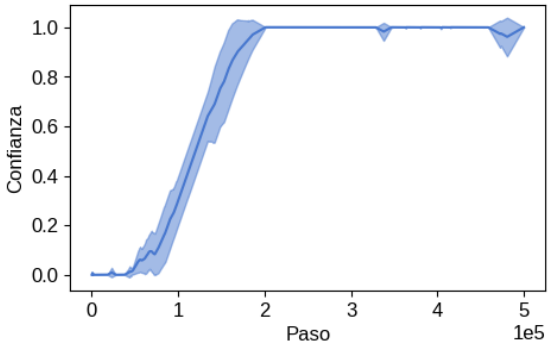


(c) Curva de aprendizaje.

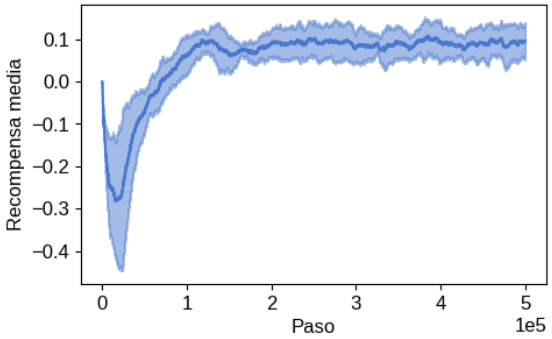
Figura 6.5: Entrenamiento del agente con búfer de repetición variable en el entorno bidimensional de 3 GDL.

de 3 GDL son muy similares a los obtenidos con el agente anterior. El valor final de la recompensa media es aproximadamente el mismo y el aprendizaje toma un tiempo similar. De igual manera la confianza exhibe una baja dispersión y un comportamiento muy cercano al lineal durante su incremento.

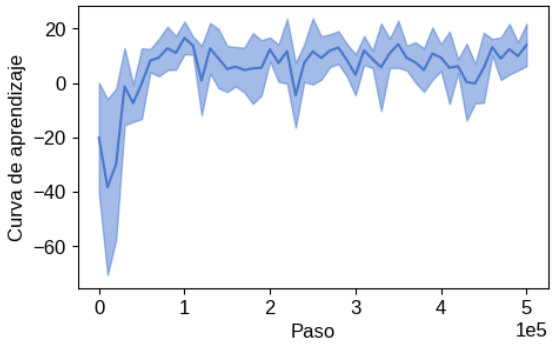
Finalmente, en el entorno tridimensional de 3 GDL se pueden observar algunas diferencias entre ambos agentes. Una de ellas es el hecho de que, aunque en ambos casos se alcanza aproximadamente el mismo valor final de recompensa media, en este caso existe una mayor variabilidad entre las diferentes ejecuciones, lo cual también se refleja en la curva de aprendizaje, que presenta un mayor nivel de ruido que el agente anterior. En el caso de la confianza también se puede ver que la desviación estándar de las ejecuciones es más grande, aun cuando la media comienza a elevarse y alcanza su nivel máximo aproximadamente al mismo tiempo. Además, este agente terminó su aprendizaje alrededor del paso 100,000, que es un tiempo significativamente mayor que el agente anterior.



(a) Confianza.



(b) Recompensa media.



(c) Curva de aprendizaje.

Figura 6.6: Entrenamiento del agente con búfer de repetición variable en el entorno tridimensional de 3 GDL.

En general el agente con búfer de repetición variable exhibe un comportamiento más volátil, con un mayor grado de dispersión entre sus ejecuciones, pero con la misma calidad de resultados que el agente con búfer de repetición fijo. En los entornos más sencillos muestra un aprendizaje más rápido y, conforme el entorno se vuelve más complejo, su aprendizaje desde cero se ralentiza. Sin embargo, sigue dándose en un tiempo razonable.

6.2. Perturbaciones

En esta sección se estudia el comportamiento del agente cuando, tras 500,000 pasos de realizar su tarea, se introduce una perturbación en su entorno que lo obliga a adaptarse. En primer lugar se presentan los resultados del agente completo, con búfer de repetición variable, en cada uno de los entornos y posteriormente se incluye una sección en la que se comparan ambos tipos de agentes y se analiza qué impacto tiene el búfer de repetición variable en la recuperación ante perturbaciones. Esta sección, sin embargo, no muestra resultados para todos los entornos.

Las figuras que siguen contienen las mismas métricas que la sección anterior. En las gráficas, la parte izquierda muestra el comportamiento del agente al aprender la política desde cero en el entorno sin perturbar. En la parte derecha se presenta el comportamiento del agente tras aplicar cada perturbación en dicho entorno. Para distinguir las diferentes líneas, se utiliza un color diferente para cada tipo de perturbación, indicado por la leyenda de la figura 6.7.

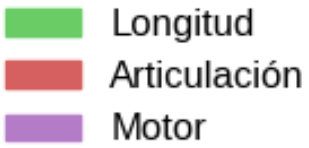


Figura 6.7: Código de colores para los resultados de los entornos perturbados.

La figura 6.8 muestra los resultados del entorno bidimensional de 2 GDL. Se observa que, en cualquier caso, la introducción de una perturbación resulta en una caída en la confianza del agente. Sin embargo, nunca llega a ser cero. Además, en dos de los tres casos (perturbaciones de motor y de articulación) la confianza del agente rápidamente vuelve a subir y establecerse en los niveles que tenía anteriormente. El hecho de que la perturbación de longitud haya resultado en la confianza final más baja es sorprendente, ya que se trata del

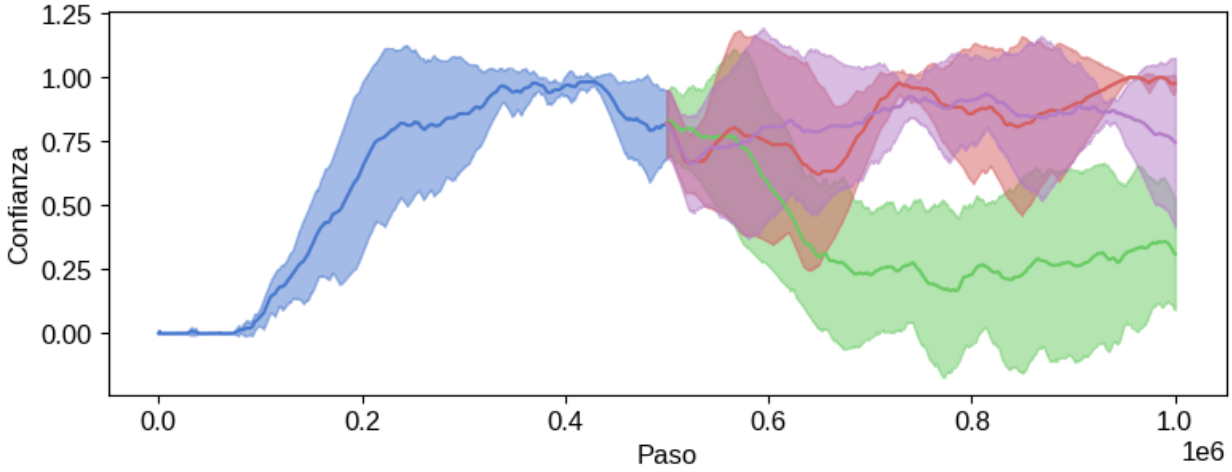
tipo de modificación de modelo más sencillo de los tres. También se aprecia que, en general, la desviación estándar de la confianza es mayor después de aplicar la perturbación, lo que implica una mayor volatilidad en el aprendizaje del agente, lo cual es de esperarse, ya que éste se encuentra en un entorno más complicado e incluso en ocasiones imposible de resolver perfectamente.

Por otro lado, la gráfica de la recompensa media muestra tendencias diferentes a la confianza. En este caso, la perturbación que termina con menor rendimiento tras un millón de pasos es la de articulación. Éste es el resultado esperado, ya que este tipo de perturbación cambia completamente el modelo del robot y su nueva forma tiene una gran cantidad de no linealidades. En cambio, la perturbación de motor muestra la recompensa final más alta, superando por un gran margen a la de longitud. La desactivación de un motor modifica también el modelo del robot, pero lo hace de una manera que no introduce elementos no lineales y, al contrario, simplifica el modelo del robot al eliminar un grado de libertad. Esto explica que tenga el rendimiento más alto de las tres perturbaciones.

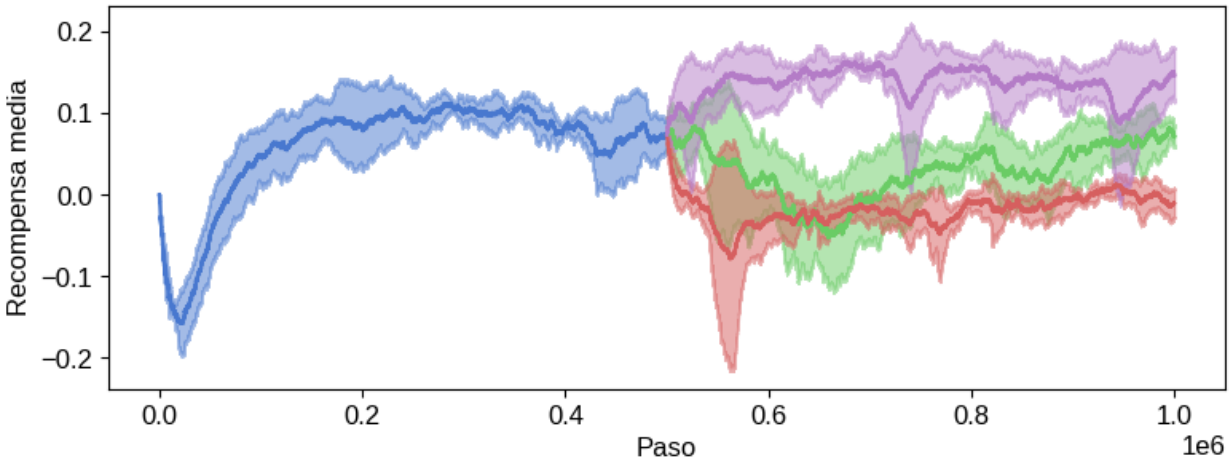
Asimismo, la recompensa media en cada una de las perturbaciones se mantiene relativamente estable después del evento, y su desviación estándar, aunque aumenta respecto al entorno original, no lo hace de manera significativa. Cabe notar que existen algunos picos en la recompensa que en varios casos coinciden en varias gráficas, pero su aparición parece aleatoria y no existe suficiente información para determinar el origen de los mismos.

Finalmente, la curva de aprendizaje muestra un patrón similar a la recompensa media en cuanto al rendimiento relativo del agente bajo las perturbaciones; es decir, la perturbación de articulación presenta el peor rendimiento y la de motor el mejor. Aun así, la curva de aprendizaje de los agentes es bastante regular; ésta se mantiene relativamente estable durante los 500,000 pasos que duraron los experimentos con una baja dispersión. Existen algunos picos hacia abajo, sobre todo en la perturbación de articulación y de longitud. Nótese que éstos picos coinciden con subidas o bajadas bruscas en la curva de recompensa media.

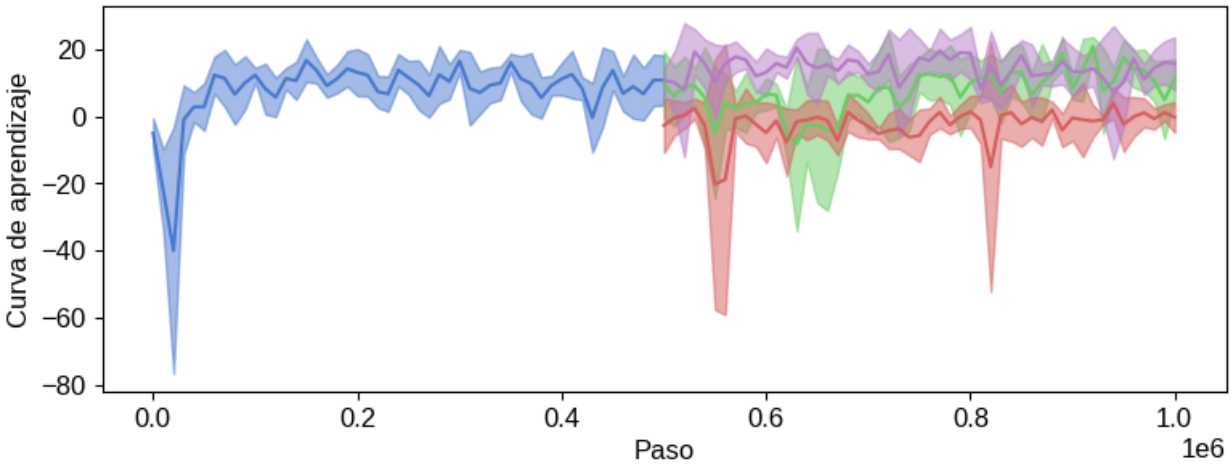
Por su parte, la figura 6.9 muestra el comportamiento del agente bajo las perturbaciones en el entorno bidimensional de 3 GDL. Se observa que, a pesar de que al iniciar la perturbación la confianza del agente era la máxima, en todos los casos ésta disminuyó momentáneamente y luego se recuperó. También se puede observar que hacia el final del episodio



(a) Confianza.



(b) Recompensa media.



(c) Curva de aprendizaje.

Figura 6.8: Resultados del entorno bidimensional de 2 GDL con perturbaciones.

la confianza baja en varios de los casos, a causa de inestabilidades en la recompensa media.

A su vez, tanto la recompensa media como la curva de aprendizaje siguen el mismo patrón que en el entorno de 2 GDL, siendo la perturbación de articulación la que provoca el peor rendimiento. De igual manera que en el entorno anterior, la curva de aprendizaje se mantiene estable y con una baja variación. Sin embargo, ambas gráficas, sobre todo la de recompensa media, muestran una baja en el rendimiento hacia el millón de pasos, que se aprecia de mayor manera en la perturbación de articulación.

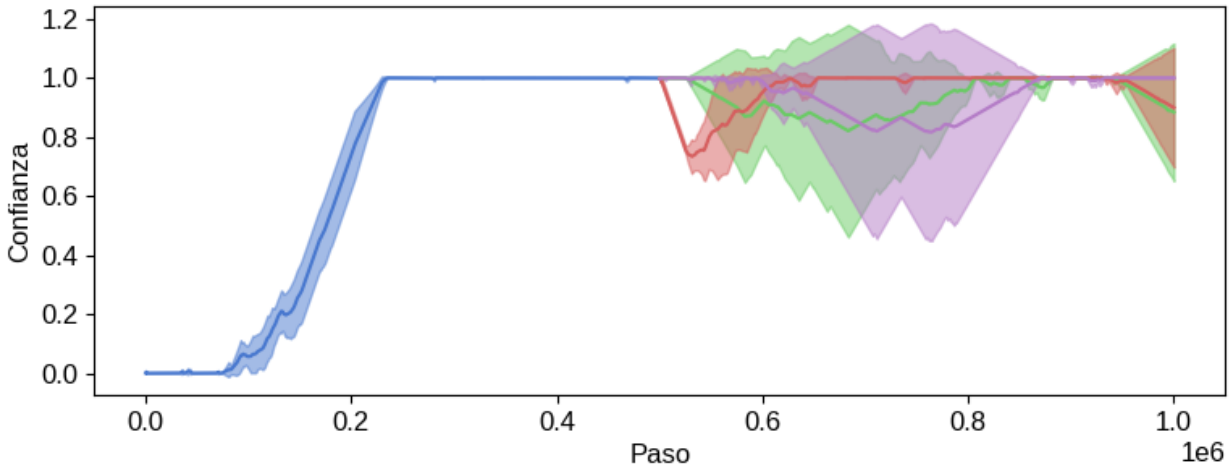
Finalmente, la figura 6.10 muestra los resultados de los experimentos en el entorno tridimensional de 3 GDL. Hay que recordar que en este entorno no se implementó la perturbación de articulación, por lo que solamente se presentan dos líneas. En este entorno, al igual que en los anteriores, el agente mantuvo estable la recompensa media y la curva de aprendizaje. Por otro lado, y a diferencia de los entornos anteriores, la perturbación de motor no alcanzó un valor final de recompensa media mayor al del entorno original; en cambio, la recompensa obtenida fue menor.

En cuanto a la confianza, el agente mostró una mayor estabilidad en este entorno que en los dos anteriores, ya que, aunque hubo picos negativos, éstos fueron más pequeños y esporádicos. Cabe destacar también que en este entorno no se presentó ninguna inestabilidad al final del entrenamiento.

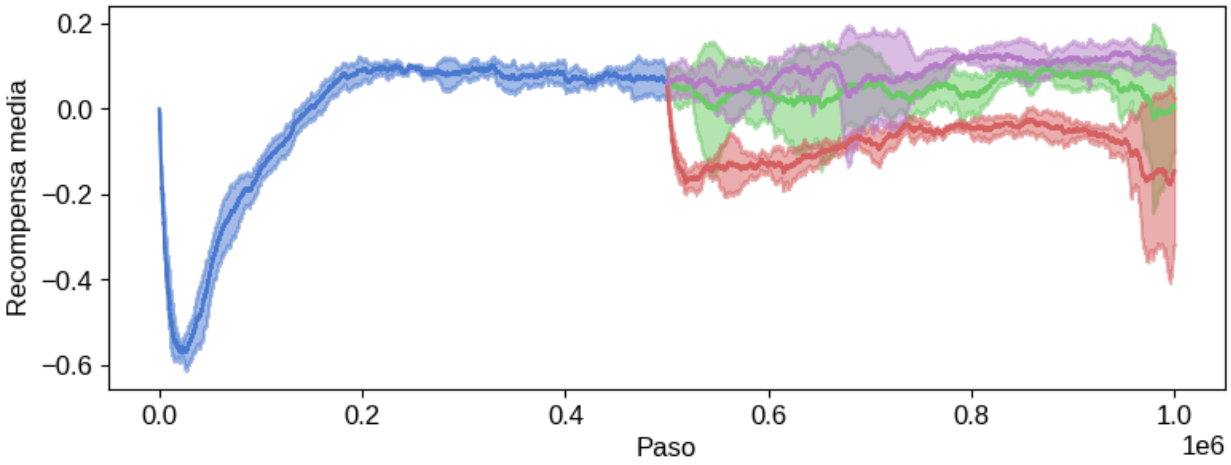
En resumen, el agente mantuvo exitosamente un rendimiento aceptable en todos los entornos perturbados. Al inicio del evento, tanto la recompensa media como la confianza cayeron, pero rápidamente se recuperaron y se estabilizaron en un valor final. Dicho valor depende del entorno específico en el que el agente se esté ejecutando, pero la dificultad relativa de las perturbaciones es la misma en todos ellos.

6.2.1. Impacto del búfer variable en la adaptación

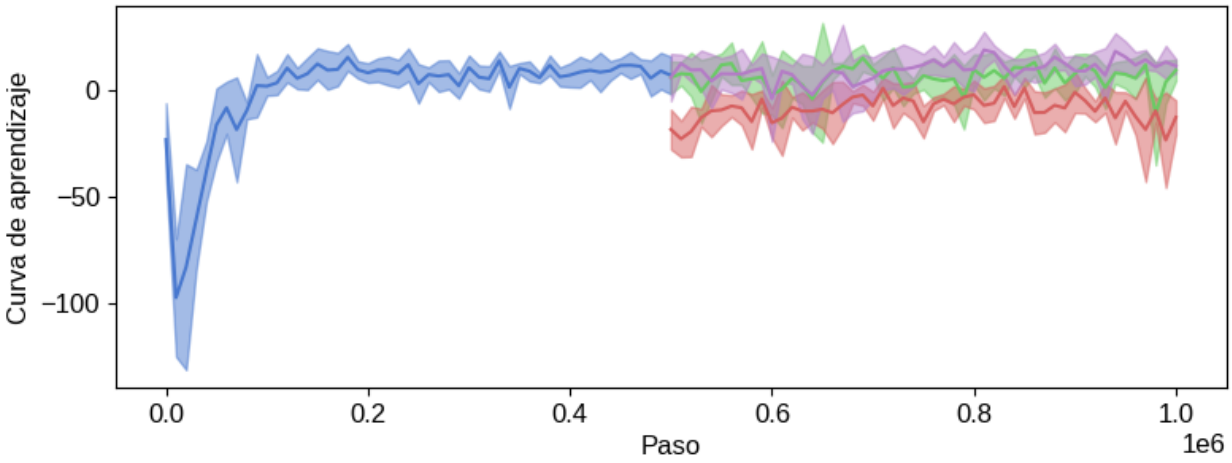
A continuación se analiza el comportamiento de cada agente en las variantes perturbadas del entorno bidimensional de 3 GDL. Se eligió este entorno porque el robot tiene más grados de libertad de los necesarios para realizar su tarea, por lo que se dice que está *sobreactuado*. De esta manera, si alguno de sus motores falla, aún estará en capacidades de funcionar adecuadamente.



(a) Confianza.

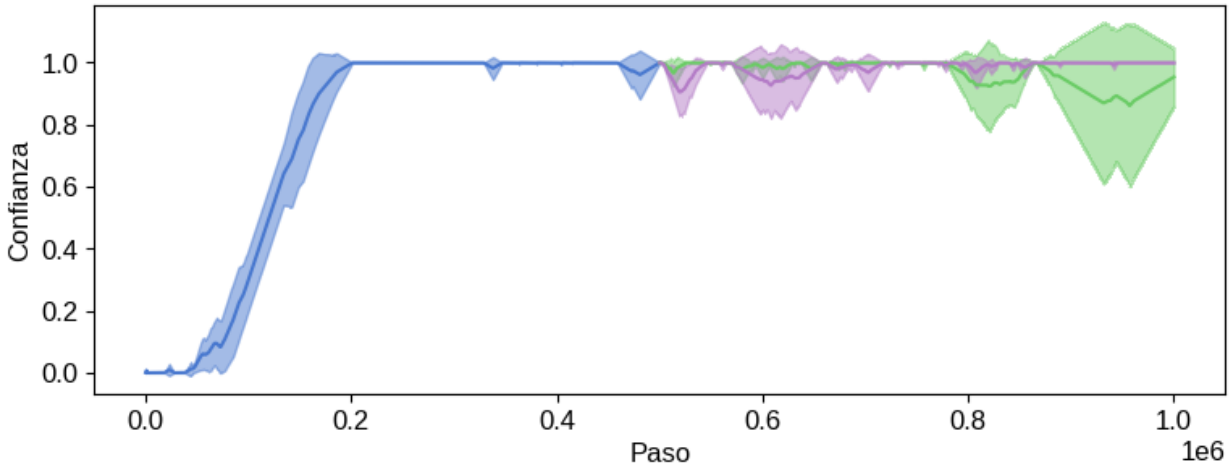


(b) Recompensa media.

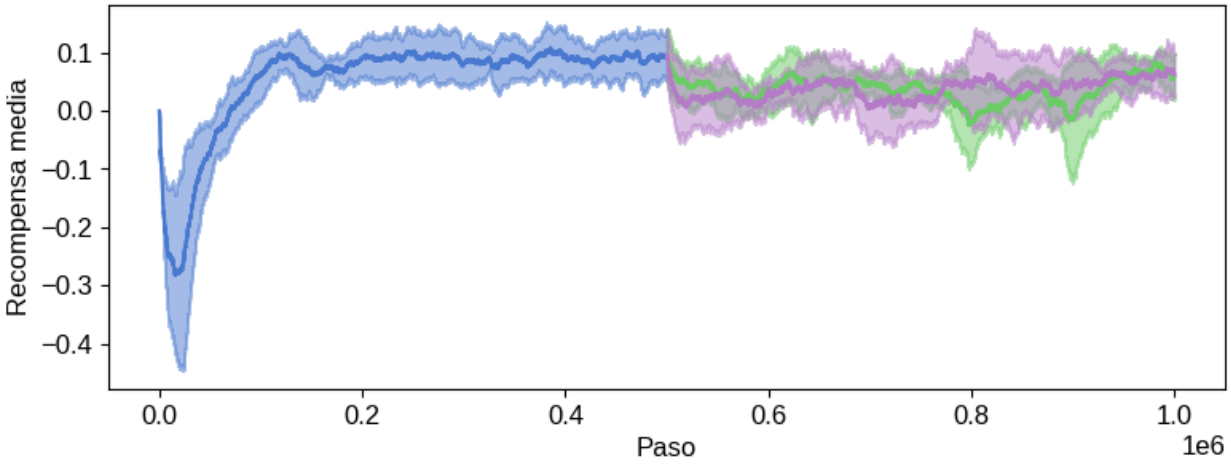


(c) Curva de aprendizaje.

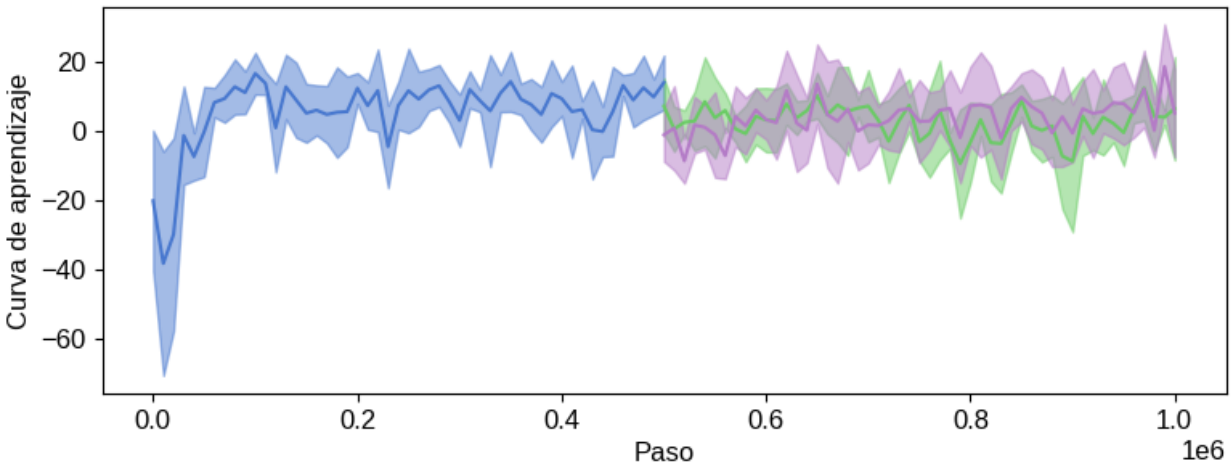
Figura 6.9: Resultados del entorno bidimensional de 3 GDL con perturbaciones.



(a) Confianza.

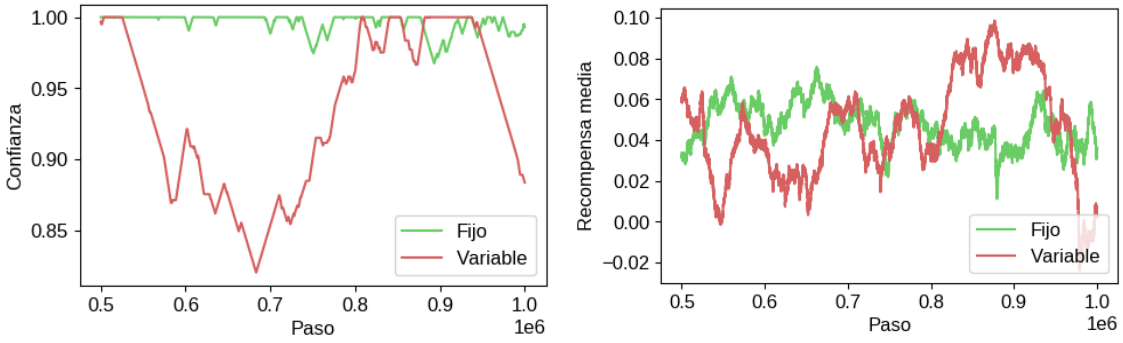


(b) Recompensa media.



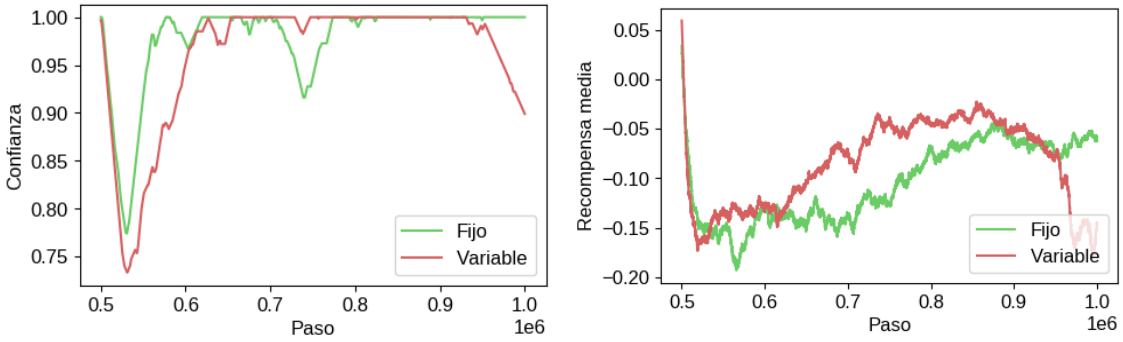
(c) Curva de aprendizaje.

Figura 6.10: Resultados del entorno tridimensional de 3 GDL con perturbaciones.



(a) Confianza. (b) Recompensa media.

Figura 6.11: Comparación de agentes en el entorno bidimensional de 3 GDL con perturbación de longitud.

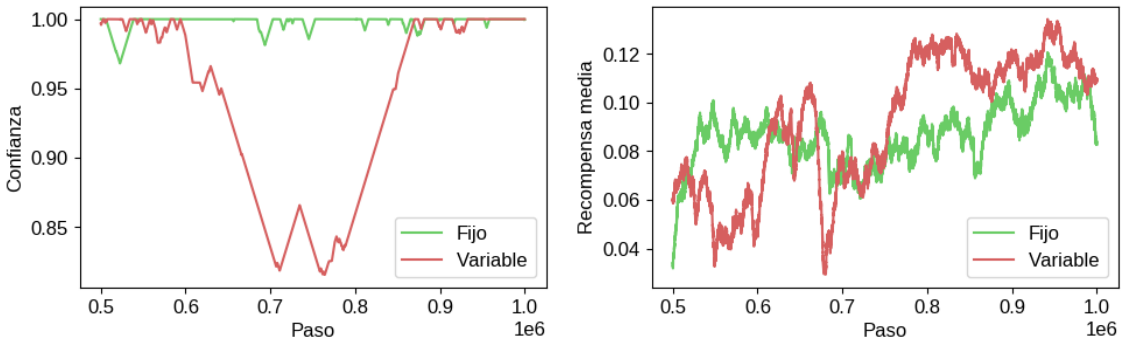


(a) Confianza. (b) Recompensa media.

Figura 6.12: Comparación de agentes en el entorno bidimensional de 3 GDL con perturbación de articulación.

Las figuras que siguen solamente contienen dos métricas: la confianza y la recompensa media. Cada gráfica contiene dos líneas: una para el agente con búfer de repetición fijo y la segunda para el búfer de repetición variable. Además, se omite la parte que contiene la información del entrenamiento desde cero en el entorno normal, ya que es irrelevante para este análisis.

Uno de los resultados más evidentes a partir de las figuras 6.11 a 6.13 es que el agente con búfer de repetición variable exhibe bajadas más pronunciadas en la confianza, sobre todo en las perturbaciones de longitud y de motor. En la perturbación de articulación este fenómeno no es tan notorio, pero el búfer fijo aún así presenta una confianza más alta. Este tipo de comportamiento es indicativo de un aprendizaje menos estable, ya que la recompensa media se mueve más rápidamente.



(a) Confianza. (b) Recompensa media.

Figura 6.13: Comparación de agentes en el entorno bidimensional de 3 GDL con perturbación de motor.

Otro fenómeno de interés es que la disminución más pronunciada en la confianza del agente no se da justamente después de la introducción de la perturbación, sino unos miles de pasos después. Esto es porque los movimientos en la confianza se dan cuando la recompensa media cambia de forma abrupta. Entonces, la confianza baja rápidamente cuando el aprendizaje es más intenso. Este momento llega cuando la cantidad de ejemplos de entrenamiento seleccionados que provienen de la nueva dinámica supera a la cantidad que proviene de la dinámica anterior. Esto, lógicamente, ocurre una vez han transcurrido una buena cantidad de pasos tras la introducción de la perturbación.

Por otro lado, las gráficas de recompensa media muestran que el agente con búfer de repetición variable presenta una mayor cantidad de oscilaciones en esta métrica, que junto con niveles de confianza más bajos, apuntan a que en este caso el aprendizaje es más volátil. Sin embargo, esto no debe tomarse como una desventaja; al contrario, señala que, aunque menos estable, el aprendizaje es también más rápido. Asimismo, el agente con búfer de repetición variable se mantiene por encima del otro agente a lo largo de la mayor parte del tiempo en recompensa media. Sobra decir que ésta es una evidencia de que el agente con búfer de repetición variable alcanza mejores rendimientos que su contraparte.

Se observa también que el rendimiento del agente con búfer de repetición variable está directamente ligado a su confianza; en la perturbación de articulación se muestra que la recompensa media recibida por este agente es superior a su contraparte mientras su confianza es máxima. En este experimento y en el de la perturbación de longitud se observa que en

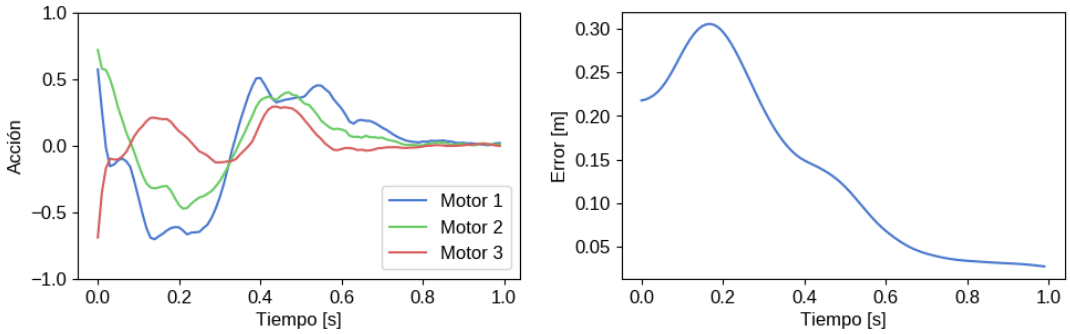
los últimos 100,000 pasos la recompensa media del agente cae muy rápidamente y que esto coincide con la baja en la confianza. El agente con búfer de repetición fijo, por otro lado, mantiene su rendimiento estable a lo largo del experimento, pero en un nivel menor.

6.3. Análisis dinámico del controlador

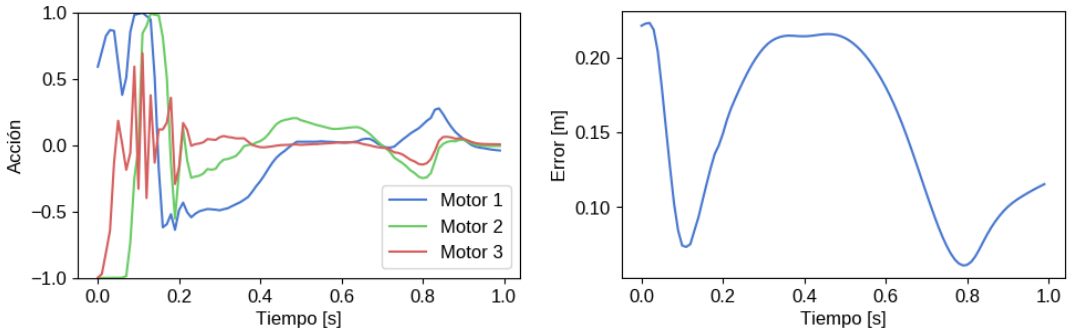
En esta sección se discute el comportamiento del controlador implementado desde el punto de vista del control automático. Para ello se tomó una de las ejecuciones del agente con búfer de repetición variable en el entorno bidimensional de 3 GDL y cada una de sus perturbaciones y se extrajo la información del último episodio, correspondiente a los 100 pasos de tiempo finales de la ejecución. Se consideró el último episodio porque se quiere evaluar el controlador final que aprende el agente, que se ejecutará en el robot real. Además, en este caso no se calculó un promedio porque la trayectoria que siga el robot depende de las condiciones iniciales y en cada episodio éstas se generan aleatoriamente. Por lo tanto la obtención del promedio causaría pérdidas de información al ser las ejecuciones completamente diferentes.

Para cada experimento se presentan dos gráficas: la primera contiene la acción ejecutada por cada uno de los motores del robot, mientras que la segunda muestra la evolución del error a lo largo del episodio. Recordemos que el rango de valores que pueden tomar las acciones del agente es $[-1, 1]$, que posteriormente son escaladas de acuerdo al espacio de acción del entorno específico en el cual se desenvuelve el agente. Por otro lado, el error, que está dado en metros, es la distancia medida entre el punto objetivo y el efector final del robot, por lo que tomará valores positivos solamente y la cota superior dependerá del espacio de trabajo del robot y de la posición del objetivo. Evidentemente, lo que se quiere es que el error eventualmente llegue a cero y se mantenga en este valor. Finalmente, el tiempo, graficado en el eje horizontal, se obtuvo multiplicando el número de pasos transcurridos por la duración de un instante de tiempo, que para todos los experimentos fue de 0.01 segundos.

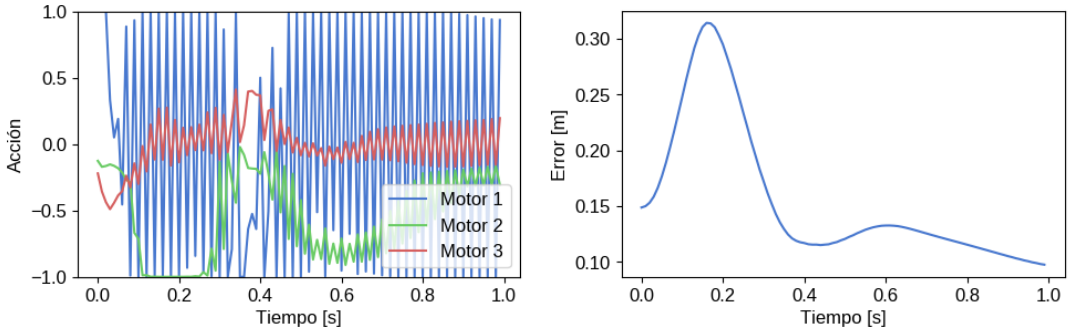
Comenzando con el entorno sin perturbar, la figura 6.14a muestra un comportamiento bastante bueno, ya que el controlador logra llevar el error a un valor muy cercano a cero. El objetivo además se alcanza en un tiempo de 0.8 segundos, que es adecuado para moverse



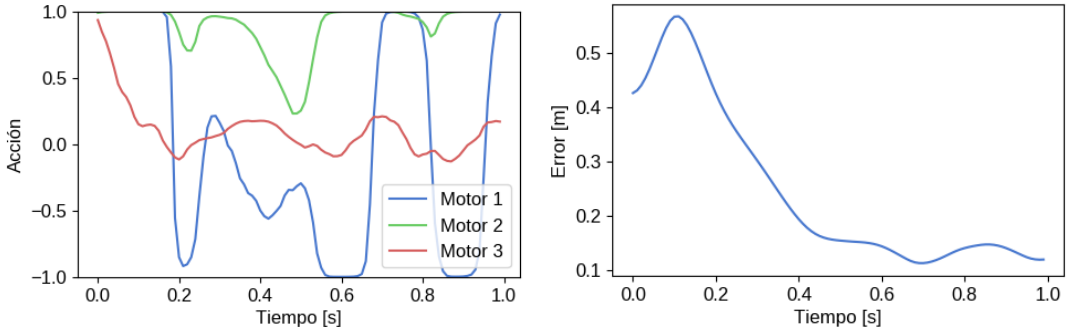
(a) Sin perturbaciones.



(b) Perturbación de longitud.



(c) Perturbación de articulación.



(d) Perturbación de motor.

Figura 6.14: Comportamiento dinámico del controlador en el entorno bidimensional de 3 GDL

varios centímetros. Después de alcanzar el objetivo, el controlador mantiene el efector final del robot en esta posición hasta que termina el episodio. Un comportamiento interesante es que, al inicio del episodio, el error aumenta ligeramente antes de descender, lo que indica que el efector final inicialmente se aleja del objetivo antes de acercarse a él.

En cuanto a las acciones realizadas por el agente, se observa que los torques más significativos se aplican al inicio del episodio y éstos se reducen en amplitud hasta quedar en cero al final. Éste es precisamente el comportamiento esperado: una vez que el robot alcanza su objetivo, no es necesario aplicar fuerzas a los motores, ya que hacerlo solamente lograría alejar la herramienta del robot del objetivo. Además, mantener la amplitud de los torques al mínimo es una política muy eficiente, ya que así se ahorra electricidad; recordemos que dentro de la función de recompensa de los agentes se encuentra una penalización al consumo energético. Por lo tanto, la política lograda por el agente en el entorno sin perturbar es muy deseable para la implementación en una planta real.

Pasando ahora a la figura 6.14b, que muestra los resultados en el entorno con perturbación de longitud, vemos que en este caso el controlador no encontró una política que logra mantener el efector final del robot en el punto objetivo. Aunque el error toma valores muy cercanos a cero en dos ocasiones, rápidamente vuelve a subir. La forma de esta curva sugiere que la herramienta pasó muy cerca del objetivo, pero lo hizo a una velocidad muy alta, por lo que el error vuelve a subir. Entonces, puede concluirse que el robot tuvo un comportamiento oscilatorio, que no es muy deseable. Sin embargo, las oscilaciones ocurrieron alrededor del punto objetivo y muy cerca del mismo. Además, puede verse que la amplitud de las mismas presenta una reducción, propia del tipo de respuesta subamortiguada, con lo que puede esperarse que, eventualmente, el error llegue y se mantenga en cero. Lamentablemente, el tiempo de un segundo no es suficiente para que esto ocurra.

Las acciones en este entorno presentan un patrón muy similar al anterior, donde al inicio la amplitud de las acciones es más grande y posteriormente éstas se atenúan. No obstante, existe un segundo pico en la acción, que coincide con la segunda reducción en el error. Este nuevo pico de acción logra reducir la amplitud de la oscilación, pero no eliminarla. Una rápida revisión de la gráfica de acción revela que es posible incrementar la amplitud de las acciones al inicio del episodio para evitar las oscilaciones y que este controlador, aunque

bueno, definitivamente puede mejorarse.

Otro punto interesante acerca de la acción es que presenta ruido de alta frecuencia hacia el inicio del episodio, denominado *chattering*, tanto en el motor 1 como en el motor 3. Este tipo de ruido puede llegar a ser perjudicial para el robot, ya que provoca pequeños impactos en el motor que, a la larga, reducen la vida útil del mismo. Sin embargo, si su frecuencia es suficientemente alta, la dinámica natural de los motores y del robot filtrará estas oscilaciones, permitiendo un funcionamiento normal. Inclusive, existen métodos de control cuya base de funcionamiento es este fenómeno.

Por su parte, la figura 6.14c muestra los resultados en la perturbación de articulación. Lo primero que salta a la vista es la gran cantidad de *chattering* que existe en esta ejecución. Aún así, es notoria una tendencia en los motores 2 y 3. El motor 1 oscila prácticamente en todo el rango posible, por lo que el valor medio es cero. Aproximadamente, el torque aplicado al motor 3 se mantiene también alrededor de cero. En cambio, el motor 2 presenta un sesgo hacia los movimientos negativos. En esta perturbación, el motor 2 es el que presenta el fallo, por lo que las acciones seleccionadas por el agente para éste son ignoradas. Es posible que por este motivo, y debido a la naturaleza estocástica del algoritmo, se hubiera creado el sesgo que se observa aquí. No obstante, lo más adecuado es que el agente aprendiera a apagar este motor para mejorar su recompensa.

Aunque la señal de acción puede mejorarse, la gráfica de error muestra que el objetivo de control de posición parece cumplirse: el error alcanza un valor más bajo que el que tenía inicialmente. El valor final del error es de 10 cm, que es alto para considerarse aceptable, pero la curva de error muestra al final del episodio una tendencia a la baja, que después de más tiempo puede llegar a niveles permisibles.

Finalmente, la figura 6.14d contiene los resultados para la perturbación de motor. Nuevamente, en este entorno se desactiva el motor número 2 y se deja rígido. Por este motivo, vemos que la acción correspondiente al mismo se satura la mayor parte del tiempo en un valor de 1 y, al igual que el caso anterior, lo mejor sería que fuera cero. Por el contrario, los motores 1 y 3 tienen un valor medio de cero a lo largo del episodio. Existe, empero, un comportamiento oscilatorio muy grande en el motor 1, que se satura en ambos extremos del rango de acción.

El error, por otra parte, después de una pequeña elevación, disminuye consistentemente hasta asentarse en un valor de alrededor de 1.5 cm. Considerando que la amplitud del movimiento deseado es de 45 cm, el valor final del error indica que el rendimiento de este controlador es bastante bueno. Existen unas pequeñas oscilaciones de alrededor de 0.75 cm de amplitud, causadas por el fenómeno explicado en el párrafo anterior. Estas dos evidencias sugieren que el robot se mantuvo en movimiento, pero de un modo que no afectó significativamente el error.

En conclusión, todos los controladores en el entorno bidimensional de 3 GDL tienen un comportamiento más parecido al subamortiguado, ya que presentan oscilaciones que van atenuándose con el tiempo. El coeficiente de amortiguamiento es, en la mayoría de los casos, muy pequeño y resulta en un movimiento cercano al críticamente amortiguado que, como se mencionó en el capítulo 3, es el ideal. Además, en todos los casos existe un pico hacia arriba en el error al inicio del episodio. Éste no encaja totalmente en el comportamiento subamortiguado, porque la señal de error no cruza el valor final. Más bien se trata de la forma general de la política aprendida por el agente. Podría argumentarse que este pico es necesario para lograr un mejor rendimiento en el resto del episodio, pero esta idea está basada puramente en la especulación. Sería necesario recabar más información para dar un veredicto definitivo.

El mejor comportamiento se tuvo en la versión sin perturbar del entorno, lo que es de esperarse, pero el agente logró aprender con éxito una nueva política para todas las versiones de perturbación. El valor final del error es en general bueno también, alcanzando valores menores a los 5 cm.

Capítulo 7

Conclusiones y trabajo futuro

En este trabajo se ataca el problema de la creación de un controlador de posición para brazos robóticos capaz de adaptarse a fallas graves y así continuar su tarea. La revisión del estado del arte reveló que existen varios algoritmos y arquitecturas que permiten aprender controladores robóticos desde cero, pero la mayoría no permite la adaptación.

El método propuesto en este trabajo se basa en la teoría del gradiente de política para crear un agente que, a partir de la experiencia pasada, aprende gradualmente en línea, permitiendo la adaptación. Se utilizó una arquitectura actor-crítico, muy empleada en modelos del estado del arte, donde ambos componentes se implementaron en forma de redes neuronales completamente conectadas. Se empleó también la técnica del búfer de repetición, que ha demostrado incrementar la estabilidad de los agentes tipo actor-crítico. Finalmente, una de las mayores aportaciones de este trabajo fue el desarrollo de un método de detección de fallos y regulación del aprendizaje en forma de una medida de confianza, la cual es clave para lograr el entrenamiento del agente en línea.

Se crearon dos versiones del agente: la primera, en la que la confianza regula solamente el ruido de exploración y la segunda en la que además afecta el tamaño del búfer de repetición, con el objetivo de acelerar la adaptación. Ambos agentes fueron evaluados en entornos simulados, donde se recrearon robots con diferente número de grados de libertad. La capacidad de adaptación del controlador se evaluó mediante la introducción de tres distintos tipos de perturbaciones a dichos entornos, todas diseñadas de acuerdo a los problemas que un robot real podría sufrir.

En primer lugar, se compararon ambas versiones de agente en las versiones originales de los entornos. Los resultados indican que los dos agentes logran aprender una política de control satisfactoria en todos los entornos que se evaluaron. Además, los agentes encuentran un controlador aceptable en un corto tiempo, adecuado para implementarse en una planta real. El agente con búfer de repetición variable muestra un aprendizaje en general más rápido, por lo que ésta fue la versión considerada para los experimentos siguientes. Sin embargo, es también menos estable, por lo que se recomienda evaluar cada caso particular donde se aplique este método para determinar qué versión del agente es la más adecuada.

A continuación se evaluó la capacidad del agente para recuperarse ante los diferentes tipos de perturbaciones. Aunque es afectado inicialmente por la nueva dinámica del sistema, el agente demostró ser capaz de encontrar una nueva política que mantiene la recompensa recibida en un valor estable. Se encontró también que las perturbaciones tienen una cierta dificultad relativa, indicada por el valor final que toma la recompensa recibida. Como se esperaba, la perturbación más complicada fue la de articulación, pero en contra de los pronósticos, la perturbación de motor fue la más sencilla. Adicionalmente se examinó el impacto que tiene el búfer de repetición variable en la capacidad de adaptación. Se encontró que el agente con tamaño de búfer variable logra valores de recompensa más altos, por lo que se concluye que presenta una mayor capacidad de adaptación.

Finalmente se realizó un análisis del comportamiento dinámico del controlador. De acuerdo con la teoría de control, el agente implementado exhibe un comportamiento propio de los sistemas subamortiguados, con ligeras oscilaciones. El tiempo de asentamiento en la mayoría de los casos es menor a un segundo, que es un tiempo adecuado en tareas domésticas. El error en estado permanente es menor a los 5 cm en general, que es significativamente mayor que el alcanzado por los controladores industriales, pero considerando que este método se propuso para aplicarse en entornos domésticos, es adecuado.

Es importante notar que, a pesar de que los controladores obtenidos cumplen su función de adaptar perturbaciones, en algunos casos es inevitable que el robot pierda la capacidad de realizar su tarea. Por ejemplo, un robot de 3 GDL puede posicionar su efector final en cualquier posición y orientación dentro de un plano. No obstante, si uno de sus motores falla, será imposible controlar las tres coordenadas de forma independiente. En consecuencia, es

mejor implementar el controlador descrito en este trabajo en un robot que está sobreactuado para la tarea que le concierne, como es el caso de la mayoría de los brazos de robots domésticos.

7.1. Trabajo futuro

El agente propuesto en este trabajo, aunque bueno, puede mejorarse. A continuación se presentan algunas líneas de investigación que, a partir de la experiencia y los resultados obtenidos pueden ser prometedoras:

- **Evaluar otras maneras de calcular la confianza.** La forma propuesta en esta tesis para calcular la confianza introduce un hiperparámetro adicional que requiere ajustarse para cada caso específico. Podría encontrarse un método para eliminar este hiperparámetro adicional al ajustar automáticamente la confianza en línea.

- **Explorar métodos para estabilizar el agente con búfer de repetición variable.** Como se mencionó arriba, este agente fue el que alcanzó el mejor rendimiento de los dos evaluados. Por otro lado también se observó que las disminuciones en la confianza pueden impactar negativamente el rendimiento del mismo. Algunos métodos posibles para evitar este problema incluyen desactivar por completo el aprendizaje en ciertos momentos o introducir un mecanismo de atención en el búfer de repetición.

- **Modificar la función de recompensa.** La forma de la función de recompensa influye en gran medida en el éxito del experimento, a veces incluso más que la misma arquitectura del agente. En particular, se sugiere explorar la introducción de términos que incentiven o penalicen ciertos aspectos del comportamiento dinámico del controlador, como un término integral que busque reducir el error en estado estable.

- **Adaptar el agente para trabajar con video.** Gran parte de la investigación actual en control robótico por inteligencia artificial se centra en diseñar agentes cuyo estado del mundo está representado por un video de la tarea. Como idea inicial, puede investigarse la posibilidad de reemplazar las redes neuronales completamente conectadas por un par de redes convolucionales, que admiten entradas de alta dimensionalidad.

Bibliografía

[1] "Irb 6620." https://new.abb.com/products/robotics/industrial-robots/irb-6620.

[2] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *Trans. of the ASME. Journal of Applied Mechanics*, vol. 22, pp. 215–221, 1955.

[3] S. Amarjyoti, "Deep Reinforcement Learning for Robotic Manipulation-The state of the art," *CoRR*, vol. abs/1701.0, 1 2017.

[4] M. P. Deisenroth, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2011.

[5] J. Kober and J. Peters, "Reinforcement Learning in Robotics: A Survey," in *Springer Tracts in Advanced Robotics*, vol. 97, pp. 9–67, 2014.

[6] S. Adam, L. Buşoniu, and R. Babuška, "Experience replay for real-time reinforcement learning control," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 42, no. 2, pp. 201–212, 2012.

[7] R. Bellman, "On the Theory of Dynamic Programming," *Proceedings of the National Academy of Sciences*, vol. 38, pp. 716–719, 8 1952.

[8] J. Murray, C. Cox, G. Lendaris, and R. Saeks, "Adaptive dynamic programming," *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 32, pp. 140–153, 5 2002.

[9] Q. Wei, D. Liu, and H. Lin, "Value Iteration Adaptive Dynamic Programming for Optimal Control of Discrete-Time Nonlinear Systems," *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 840–853, 2016.

[10] R. Song, F. L. Lewis, Q. Wei, and H. Zhang, "Off-Policy Actor-Critic Structure for Optimal Control of Unknown Systems With Disturbances," *IEEE Transactions on Cybernetics*, vol. 46, pp. 1041–1050, 5 2016.

[11] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *NIPS'99 Proceedings of the 12th International Conference on Neural Information Processing Systems* (Sora A. Solla, Todd K. Leen, and K. Müller, eds.), (Denver), pp. 1057–1063, MIT Press Cambridge, 1999.

[12] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *Proc. of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), (Beijing), p. 387–395, JMLR.org, 2014.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *ICLR*, 2016.

[14] S. Banerjee and A. Chatterjee, "Real-time self-learning for control law adaptation in nonlinear systems using encoded check states," in *2017 22nd IEEE European Test Symposium (ETS)*, (Limassol, Cyprus), pp. 1–6, IEEE, 5 2017.

[15] V. Gullapalli, "A stochastic reinforcement learning algorithm for learning real-valued functions," *Neural Networks*, vol. 3, no. 6, pp. 671–692, 1990.

[16] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 5 1992.

[17] S. Levine and V. Koltun, "Guided Policy Search," in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta), pp. 1–9, PMLR, 2013.

[18] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," *Journal of Machine Learning Research*, vol. 17, pp. 1–40, 4 2015.

[19] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), (New York), pp. 2829–2838, JMLR.org, 2016.

[20] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3389–3396, 2017.

[21] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 303–314, 12 1989.

[22] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[23] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, pp. 145–151, 1 1999.

[24] G. Hinton, N. Srivastava, and K. Swersky, "Neural Networks for Machine Learning," 2014.

[25] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *ICLR 2015*, pp. 1–15, 2014.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2 ed., 2018.

[27] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 5 1992.

[28] G. E. Uhlenbeck and L. S. Ornstein, "On the Theory of the Brownian Motion," *Physical Review*, vol. 36, pp. 823–841, 9 1930.

[29] E. S. PAGE, "CONTINUOUS INSPECTION SCHEMES," *Biometrika*, vol. 41, no. 1-2, pp. 100–115, 1954.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," pp. 1–12, 7 2017.