UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

APLICACIONES DEL APRENDIZAJE REFORZADO EN ROBÓTICA MÓVIL

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

KENZO IGNACIO LOBOS TSUNEKAWA

PROFESOR GUÍA:
SR. JAVIER RUIZ-DEL-SOLAR SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
SR. FELIPE TOBAR HENRÍQUEZ
SR. ANDRÉS CABA RUTTE

SANTIAGO DE CHILE
2018

RESUMEN DE LA MEMORIA PARA OPTAR AL TITULO DE INGENIERO CIVIL ELÉCTRICO
POR: KENZO IGNACIO LOBOS TSUNEKAWA
FECHA: 22/01/2018
PROF. GUÍA: DR. JAVIER RUÍZ-DEL-SOLAR SAN MARTÍN

APLICACIONES DEL APRENDIZAJE REFORZADO EN ROBÓTICA MÓVIL

En la última década se ha observado un aumento importante en las aplicaciones relacionadas a la robótica a nivel mundial. Adicionalmente, estas aplicaciones ya no se encuentran únicamente en laboratorios o en fábricas, donde se pueden mantener condiciones controladas, sino que también se presentan en distintas situaciones cotidianas. Entre las distintas categorías de robots, destaca fuertemente la robótica móvil, debido a su alto potencial de impacto social. Aquellos robots que forman parte de esta categoría, potencialmente pueden resolver una cantidad muy variada de problemas, debido a su capacidad de realizar interacciones complejas con un entorno dinámico, el cual puede incluir interacciones con seres humanos u otros agentes robóticos. Sin embargo, la capacidad de resolver problemas por parte de un agente robótico suele estar limitada por el conocimiento y las habilidades del diseñador. Se identifica entones la necesidad de incorporar metodologías generales que permitan a los agentes robóticos adquirir las habilidades necesarias para poder realizar las labores que les son asignadas.

En el presente trabajo se estudia el uso del Aprendizaje Reforzado como herramienta de uso general para que los agentes robóticos adquieran las habilidades necesarias para realizar su labor. Son objetivo de especial interés, no solo la capacidad de resolver problemas particulares, sino que además estudiar la capacidad de generalización de las soluciones, y la escalabilidad de ésta herramienta.

La metodología propuesta consiste en el uso del fútbol robótico como caso de estudio, debido a su complejidad como problema, al mismo tiempo de su facilidad de evaluación. Se identifican problemas de diversa complejidad y naturaleza en este contexto, identificando cuales son las características que son generales a distintos problemas, permitiendo extraer resultados de interés a otras aplicaciones. Para resolver los problemas identificados, se utilizan distintos algoritmos del Aprendizaje Reforzado, tanto tradicionales como modernos, haciendo hincapié en los beneficios de cada uno.

Los resultados permiten perfilar al Aprendizaje Reforzado como una herramienta útil en el contexto de la robótica móvil. Algoritmos tradicionales son capaces de solucionar problemas sencillos de manera altamente eficiente y utilizando bajos recursos. Por otro lado, las técnicas modernas permiten abordar problemas mucho más complejos, previamente considerados intratables de manera directa. Finalmente, el uso de esta metodología presenta un potencial todavía no explorado a profundidad, sin conocer todavía el límite en sus aplicaciones. Se identifica entonces un amplio campo de desarrollo para futuros trabajos e investigación.

Agradecimientos

Quiero agradecer al Laboratorio de Robótica de la Universidad de Chile por ofrecer un lugar idóneo para el aprendizaje durante este importante período de mi vida. Adicionalmente, quiero agradecer especialmente a José Miguel Yañez por todas sus enseñanzas, paciencia y amistad, a Leonardo Leottau por introducirme en esta emocionante área de estudios, y al Dr. Javier Ruiz-del-Solar por su confianza y apoyo a lo largo de mis estudios, y en particular durante la realización de esta memoria. Adicionalmente quisiera destacar que parte del trabajo de esta memoria fue realizado durante el curso *MA5203 - Aprendizaje de Máquinas Probabilístico* dictado por el profesor Felipe Tobar, y que fue parcialmente financiado por el proyecto FONDECYT 1161500.

Kenzo Lobos Tsunekawa

Tabla de contenido

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

La robótica es una disciplina que resulta de la combinación de otras áreas del conocimiento, tales como mecánica, electrónica, ciencias de la computación, control de sistemas, entre otras, siendo su objetivo el diseño, construcción, programación y operación de sistemas o agentes robóticos, que son dispositivos programables capaces de cumplir tareas previamente especificadas, de manera automática.

Las aplicaciones de robótica ya no son impensadas en nuestra sociedad. Desde hace hace décadas, robots están presentes en las industrias, principalmente en forma de manipuladores robóticos (e.g Fanuc, KUKA) encargados de labores de ensamblaje y manufactura, y de manera más reciente también en líneas de transporte de productos en bodegas (Amazon).
Adicionalmente, en la última década se ha observado un notorio aumento en la cantidad de aplicaciones robóticas que están presentes de manera visible en nuestra sociedad. Estas aplicaciones ya no se encuentran aisladas en un laboratorio, o en un espacio cerrado dentro de una fábrica, sino que se encuentran inmersos en ambientes dinámicos y complejos. Ejemplos de estos casos son los robots de limpieza (e.g Roomba), de servicio (e.g Bender), de acompañamiento (e.g Pepper) y educacionales (e.g NAO, Robi).

La mayoría de los casos anteriores caben dentro de la categoría de robótica móvil, la cual es un área de la robótica que estudia aquellos agentes robóticos que debido a sus capacidades de movimiento, deben ser capaces de resolver problemáticas complejas, tales como localización y navegación, mientras desempeñan su labor. Esta es la categoría de robots, que tiene un mayor potencial a futuro, debido a que su movilidad les permite abordar una mayor gama de problemas, que incluye labores en las que deba existir interacción y potencial colaboración, con seres humanos y otros agentes robóticos.

Pese al gran potencial de la robótica, sus implementaciones suelen estar limitadas por la capacidad del diseñador de implementar el conjunto de sistemas de *software* que les permitan realizar su labor de manera eficiente. A nivel general, el diseñador de un agente robótico debe implementar los siguientes sistemas: percepción, modelamiento, toma de decisiones y actuación. Un esquema que presenta los sistemas anteriormente mencionados y sus interacciones está presente en la Figura 1.1.

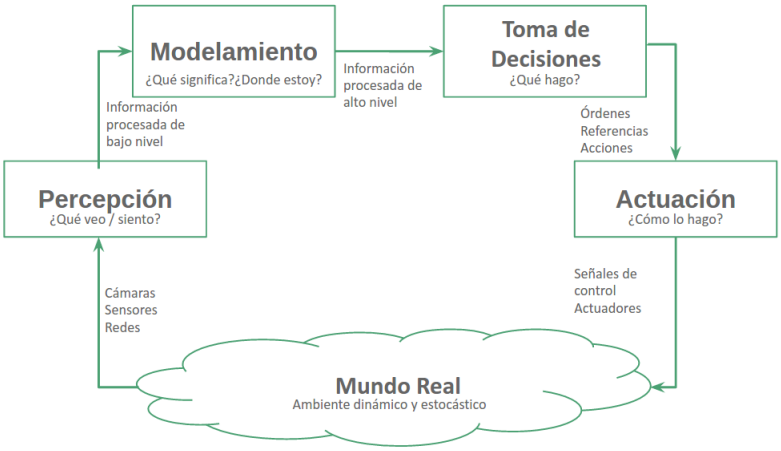


Figura 1.1: Distintos módulos que componen a un sistema robótico.

Debido a la alta complejidad de cada sistema, y a la gran diversidad de aplicaciones robóticas, es usual tener que desarrollar cada sistema de manera *ad-hoc* al problema particular. Esta metodología permite usualmente lograr desempeños aceptables o en algunos casos incluso sobresalientes. Sin embargo, no permite en la mayoría de los casos, una generalización de las soluciones desarrolladas, a otros problemas similares.

En este contexto, el trabajo de esta memoria busca evaluar el uso del Aprendizaje Reforzado, una herramienta proveniente del Aprendizaje de Máquinas, como metodología general de resolución o implementación de los sistemas anteriormente mencionados. En particular es habitual encontrar en la literatura aplicaciones de Aprendizaje Reforzado relacionadas a los sistemas de toma de decisiones o actuación. En este trabajo se evalúan dichos casos, además de utilizar el Aprendizaje Reforzado para implementar de manera simultánea más de un bloque, utilizando directamente la información sensorial.

1.1. Fútbol Robótico

El fútbol robótico es una competencia de naturaleza científica, que consiste en un entorno donde dos equipos de robots autónomos deben desempeñarse en la tarea de jugar fútbol, con reglas similares a su contraparte humana. Esta competencia forma parte de la *RoboCup (Robot Soccer World Cup)* [1], un evento anual, que incluye un simposio además de las competencias anteriormente mencionadas. El objetivo de la *RoboCup* es promover el desarrollo de la robótica y su investigación, principalmente a nivel universitario.

El fútbol robótico presenta apropiadas cualidades como objeto o caso de estudio en el contexto de este trabajo. En primer lugar, el fútbol robótico es una competencia que se ha estudiado por casi 20 años, por lo cual las implementaciones actuales están consolidadas y existen diferentes herramientas disponibles. Además, la naturaleza intrínseca del fútbol robótico lo convierte en un caso de estudio idóneo, debido que la problemática a nivel de implementación es compleja, y general a muchas otras aplicaciones robóticas, y sin embargo la evaluación del desempeño de las distintas aplicaciones en este contexto es sencilla, debido a su similaridad con el fútbol humano.

1.2. Estado del Arte

La formulación del Aprendizaje Reforzado [2] hace que su uso en robótica parezca como una consecuencia natural, debido a que el aprendizaje se realiza mediante prueba y error, en una secuencia de interacciones del robot con su entorno, buscando aprender cierto comportamiento. Es de esta manera que ciertos autores incluso llegan a comparar la relación entre la robótica y el Aprendizaje Reforzado, y la existente entre la Física y la Matemática, siendo en este caso el Aprendizaje Reforzado la herramienta que permite a la robótica alcanzar su mayor potencial [3]. Pese a lo anterior, existen numerosas limitaciones, tanto teóricas como prácticas, que dificultan el uso del Aprendizaje Reforzado de manera directa.

En particular, las principales limitaciones con respecto al uso del Aprendizaje Reforzado en robótica consisten en la complejidad computacional, y eficiencia de datos (cantidad de interacciones necesarias), a medida que la dificultad o dimensionalidad del problema se incrementa.

Uno de los problemas más complejos en robótica consiste en la generación de movimientos y control motor (actuación), desafíos que has sido objeto de estudio de diversas metodologías, entre las cuales se encuentra el Aprendizaje Reforzado. Este tipo de problemas están caracterizados por una dimensionalidad que varía usualmente entre 6 y 50 dimensiones, tanto para el espacio de estados como para el espacio de acciones (usualmente representados por los grados de libertad actuados del sistema), por una necesidad de utilizar la menor cantidad de interacciones posibles (experimentos caros y peligrosos para el sistema físico), y por la necesidad de ser capaces de ejecutarse en tiempo real (interacciones entre sistemas físicos).

En la literatura, existen diversas estrategias para abordar la problemática anteriormente mencionada. La eficiencia o complejidad de la metodología en cuanto a datos puede ser abor-

dada exitosamente mediante el uso de simuladores de los sistemas físicos involucrados. Sin embargo, el uso de simuladores para realizar el proceso de aprendizaje presenta en la práctica pobres resultados de transferencia directa (entrenamiento en un simulador, y prueba en el sistema físico), debido a que el denominado *reality gap* se hace presente fuertemente en el modelamiento físico de los actuadores. Esta nueva problemática genera que grandes esfuerzos de la comunidad científica sean destinados al diseño de simuladores cada vez más realistas, con el objetivo de poder disminuir los efectos del *reality gap*.
Por otro lado, realizar el proceso de aprendizaje directamente en el sistema físico (robot) suele ser impracticable. En estos casos el proceso de aprendizaje puede demorar días, en los cuales suele ser necesaria intervención humana continua, como por ejemplo, restablecer condiciones de entrenamiento al finalizar cada episodio. Finalmente, el sistema físico se desgasta durante el entrenamiento, y el continuo uso genera que el sistema no presente un proceso estacionario. Debido a estas limitaciones, se suelen utilizar en robótica solo ciertos tipos de algoritmos de aprendizaje reforzado que entran dentro de la categoría de *Policy Search*, que permiten realizar el proceso de entrenamiento como una optimización de parámetros sobre una política inicial de buen comportamiento. Pese a tener una alta eficiencia en cuanto a interacciones, necesita una política inicial que no siempre está disponible. En estos casos, el Aprendizaje Reforzado se utiliza como un simple optimizador de parámetros, y presenta resultados competitivos con otras metodologías similares, que son incluso más generales, como las presentes en la computación evolutiva.

Finalmente, en los últimos años se han desarrollado nuevas herramientas que han permitido resolver problemas previamente intratables. El principal ejemplo de este fenómeno es el Aprendizaje Profundo o *Deep Learning*, cuyas técnicas han permitido abordar diversos problemas como clasificación y predicción con resultados anteriormente impensables. En el área de Aprendizaje Reforzado, se han incorporado resultados y herramientas provenientes de *Deep Learning*, en lo que se denomina *Deep Reinforcement Learning*. *Deep Reinforcement Learning* ha generado un nuevo interés en el área del Aprendizaje Reforzado, debido a los logros que ha obtenido. En particular, mediante el uso del algoritmo *Deep Q-Learning* [4] se logró implementar un sistema con rendimientos comparables o incluso superiores a jugadores humanos en la consola *Atari* [5], demostrando una alta escalabilidad, al abordar de manera eficiente espacios de alta dimensionalidad. Por otro lado, a través del uso del algoritmo *Deep Deterministic Policy Gradients (DDPG)* [6], se logró resolver un conjunto de problemas físicos (simplificaciones de problemas recurrentes en robótica), con un rendimiento que en algunos casos resultó ser superior incluso a algoritmos de *planning* o control óptimo, con acceso completo al modelo físico subyacente. Es así como se observa una situación en la que en la que día a día se resuelven problemas anteriormente imposibles de abordar, y no se conocen aún cuales son las limitaciones reales de esta nueva metodología. De este modo, la comunidad científica se enfrenta con optimismo al desafío de desarrollar este campo, debido a las promesas que este presenta.

1.3. Objetivos

1.3.1. Objetivos Generales

El objetivo general de este trabajo consiste en la evaluación del uso de técnicas de Aprendizaje Reforzado como herramienta de resolución de problemas en aplicaciones de robótica, en particular de robótica móvil, utilizando como caso de estudio la problemática presente en el fútbol robótico. Son de especial importancia la evaluación o caracterización de cuales son los problemas que esta metodología puede resolver, y las potenciales ventajas y desventajas de esta metodología con respecto a otras.

Adicionalmente, también se busca realizar una comparación entre distintas técnicas de Aprendizaje Reforzado, con el objetivo de evaluar, en diferentes problemas, cuales son las condiciones bajo las cuales cada método resulta conveniente. En particular se busca evaluar tiempos de ejecución, recursos utilizados, estabilidad del proceso de aprendizaje, y tiempo de convergencia.

1.3.2. Objetivos Específicos

En vista de los objetivos generales planteados, y para el correcto desarrollo de este trabajo, se procede a enumerar los objetivos específicos o hitos de desarrollo de la metodología propuesta:

1. Realizar una revisión bibliográfica del Aprendizaje Reforzado, con énfasis en aplicaciones en robótica y metodologías del estado del arte que presenten potencial para dicha área.
2. Diseñar e implementar una librería de Aprendizaje Reforzado orientada a aplicaciones en robótica, donde se requiere un alto desempeño computacional, y una alta integrabilidad con entornos de desarrollo de aplicaciones robóticas tradicionales.
3. Implementar algoritmos y herramientas de Aprendizaje Reforzado, tanto tradicionales como modernas.
4. Analizar el caso de estudio presentado, identificando problemas que no puedan ser resueltos mediante técnicas tradicionales, y que puedan ser modelados mediante Aprendizaje Reforzado.
5. Modelar los problemas anteriormente identificados mediante la formulación del Aprendizaje Reforzado, con énfasis en diseños que permitan abordar los problemas tradicionales en aplicaciones en robótica
6. Analizar el desempeño de la metodología propuesta en los problemas anteriores, utilizando los distintos métodos implementados, identificando las cualidades de los resultados obtenidos que permitan concluir acerca de su implementabilidad en sistemas físicos, generabilidad de la metodología a otros problemas y definir futuras lineas de desarrollo.

1.4. Estructura de la Memoria

El presente informe contiene una estructura secuencial, que inicia con la presentación de las herramientas teóricas que se utilizan en este trabajo. Luego, se introduce formalmente el caso de estudio, detallando la problemática existente, indicando además la metodología planteada para resolver dichos problemas. Más adelante se presentan los diferentes experimentos realizados, detallando las condiciones bajo los que fueron efectuados. Finalmente se analizan los resultados, y se presentan las conclusiones relevantes extraídas del presente trabajo.

El Capítulo 2 consiste en el marco teórico de las herramientas utilizadas a lo largo de este trabajo. El capítulo comienza con una breve introducción al Aprendizaje Reforzado, la cual consiste principalmente en el contexto bajo el cual se propone esta metodología, su formulación básica y objetivos concretos. Posteriormente se detalla cuales han sido sus principales éxitos en el área de la robótica, así como cuales han sido sus principales limitaciones que previenen un uso masivo. Finalmente se desarrollan las diferentes técnicas e implementaciones que se utilizan durante este trabajo.

El Capítulo 3 presenta de manera formal el caso de estudio, y cuales son las herramientas que se utilizan para trabajar en esta problemática. Posteriormente se justifica la necesidad de implementar un conjunto de herramientas flexibles que permitan resolver problemas en robótica de manera sencilla mediante Aprendizaje Reforzado. Es este sentido, se presentan además las herramientas externas y el rol que juegan dentro de la librería de *software* a implementar. Finalmente se desarrollan los diferentes problemas identificados, la razón de su elección, y su modelamiento, que les permite ser resueltos mediante la metodología propuesta.

El Capítulo 4 presenta y analiza los resultados obtenidos para los distintos problemas identificados, utilizando las herramientas introducidas en el capítulo 2, e implementadas según el capítulo 3. En particular se estudian el rendimiento, beneficios y desventajas de cada método propuesto. Se analiza además la relevancia de cada una de las partes que componen las estrategias de aprendizaje reforzado, con el fin de identificar aquellos componentes que resultan críticos para el correcto desempeño de estas técnicas. Finalmente, se busca extender estos resultados a otros problemas, identificando similitudes, además de las posibles complicaciones para llevar estas metodologías a otros problemas.

Finalmente, el Capítulo 5 presenta las conclusiones del trabajo realizado, teniendo en consideración fuertemente la relación entre los objetivos planteados inicialmente y los resultado obtenidos, que junto con el análisis realizado permiten discutir acerca de los alcances de esta metodología, y establecer un mapa de desarrollo en investigación para futuros trabajos.

Capítulo 2

Marco Teórico

El presente capítulo introduce al lector los conceptos fundamentales del Aprendizaje Reforzado, centrado en aquellas herramientas utilizadas en el contexto de este trabajo. El objetivo de este capítulo consiste en entregar al lector las herramientas suficientes para poder comprender el trabajo realizado, identificar la importancia de los distintos elementos presentes en los experimentos realizados, y aprovechar la posterior discusión respecto a los mismos.

En primer lugar, la Sección 2.1 consiste en una introducción general al Aprendizaje Reforzado. Se detalla su contexto, motivación y parte de su historia. Posteriormente se desarrollan formalmente los conceptos asociados, incluyendo los objetivos generales del proceso de aprendizaje, y las diferentes naturalezas que pueden presentar las implementaciones de Aprendizaje Reforzado.

En segundo lugar, en la Sección 2.2 se abordan algunas de las implementaciones clásicas de Aprendizaje Reforzado, desarrollando además cuales han sido las modificaciones o extensiones realizadas a dichas implementaciones o lo largo de los años. Finalmente, en la Sección 2.3 se desarrollan técnicas modernas que estén enmarcadas dentro de la literatura de *Deep Reinforcement Learning* La distinción explícita entre los desarrollos previos a *Deep Reinforcement Learning* en el Aprendizaje Reforzado, y aquellos resulten posteriores, se realiza con el objetivo de cuantificar y evaluar la diferencia en cuanto a resultados de cada metodología, con el objetivo de ser una guía para futuros trabajos con respecto a las herramientas que resulten convenientes de utilizar.

2.1. Aprendizaje Reforzado

En esta sección se introduce el Aprendizaje Reforzado, una herramienta matemática proveniente del Aprendizaje de Máquinas, que permite modelar el proceso de aprendizaje mediante una sucesión de interacciones entre un agente de aprendizaje y su entorno, con el objetivo de una tarea específica. En primer lugar se presentan las motivaciones que dan origen al Aprendizaje Reforzado, a la vez que se lo compara con metodologías similares dentro del Aprendizaje de Máquinas. Posteriormente, se plantea la formulación matemática de este estilo de aprendizaje, identificando los conceptos clave o elementos esenciales del Aprendizaje

Reforzado, a la vez que se plantean formalmente los objetivos, suposiciones, entre otros. Finalmente se desarrollan temas particulares del área, tales como la taxonomía de las diferentes implementaciones de Aprendizaje Reforzado, y sus potenciales usos en robótica, pero sin entrar en detalles de implementaciones particulares, las cuales son debidamente desarrolladas en secciones independientes, debido a su extensión.

2.1.1. Motivación

El Aprendizaje de máquinas es un campo de estudio que busca otorgar a máquinas de cómputo la habilidad de aprender comportamientos sin que éste les sea explícitamente programado. El término de Aprendizaje de Máquinas fue establecido por primera vez en 1959 en el contexto de los juegos por computadora, y desde entonces ha evolucionado hasta estar presente en diferentes áreas de estudio tales como lo son el reconocimiento de patrones, predicciones de series de tiempo, clasificación, compresión de datos, reducción de dimensionalidad, etc.

Si bien existen diversos tipos de Aprendizaje de Máquinas, la formulación más utilizada en la actualidad corresponde a la del Aprendizaje Supervisado. Este tipo de aprendizaje busca realizar un mapeo entre un conjunto de entrada $\mathcal{X}$ y un conjunto de salida $\mathcal{Y}$. En el caso que el conjunto de salida es discreto, se suele hablar de problemas de clasificación, y en el caso continuo se habla de problemas de regresión. Para aprender este mapeo, es necesario la existencia de un conjunto de datos denominado de entrenamiento, que corresponden a tuplas $(x, y) \in \mathcal{X} \times \mathcal{Y}$. Los datos y son obtenidos mediante una entidad denominada como supervisor, que puede ser un ser humano (o varios), o algún otro sistema artificial. Con esta formulación, se busca que el sistema de aprendizaje sea capaz de aprender el conocimiento el supervisor con respecto a los datos.

El Aprendizaje Supervisado presenta excelentes resultados en diferentes tipos de aplicaciones. Por ejemplo, existen técnicas que permiten reconocer diferentes objetos en secuencias de video, identificar objetos entre mil clases, y predecir series de tiempo biológicas, con una alta precisión. Sin embargo la formulación del Aprendizaje Supervisado presenta numerosas fallas cuando se desea aplicar a problemas inmersos dentro de área de toma de decisiones secuenciales.

La primera limitación del Aprendizaje Supervisado en problemas de toma de decisiones secuenciales tiene relación con los supuestos bajo cuales se realiza el aprendizaje. En el aprendizaje supervisado, uno de los principales supuestos es la independencia temporal de los datos. Más aún, se asume que los datos de entrenamiento son independientes e idénticamente distribuidos (i.i.d), y que estos datos representan la distribución de los datos a los que se verá expuesto el sistema posteriormente al entrenamiento. Sin embargo, en los problemas de decisiones secuenciales, las acciones del sistema tienen consecuencias sobre su entorno, o dicho de otra manera, condicionan los datos futuros ($p(x_{t+1}) \neq p(x_{t+1}|a_t = a)$). Dicha dependencia sobre las acciones del sistema, produce que el entrenamiento sea mucho más complejo. Además, en caso de ignorar esta dependencia, el resultado del entrenamiento, pese

a lograr errores ínfimos durante el entrenamiento, produce pobres desempeños al momento de desplegar el resultado del entrenamiento sobre condiciones de validación.

Un segundo problema del Aprendizaje Supervisado en este tipo de problemas tiene relación con la existencia del supervisor. El Aprendizaje Supervisado se basa en la disponibilidad de dicho supervisor, y su objetivo es imitarlo. Sin embargo, en muchos problemas, no existe un supervisor que tenga el conocimiento experto necesario (como por ejemplo plantas de control excesivamente complejas), o incluso en el caso que dicho supervisor esté disponible, el conocimiento de dicho supervisor representa una cota en el desempeño del sistema de aprendizaje. Es decir, el objetivo del aprendizaje es obtener un rendimiento a la par que el obtenido por el supervisor, pero en ningún caso puede sobrepasar su desempeño (matemáticamente el error se plantea como la diferencia entre las acciones del sistema de aprendizaje con respecto a las realizadas por el supervisor, y en ningún caso se optimiza con respecto al problema real a resolver).

Para abordar este tipo de problemas, en los que el Aprendizaje Supervisado suele ser insuficiente, el Aprendizaje de Máquinas ofrece otras alternativas, dentro de las cuales una de las más generales corresponde al Aprendizaje Reforzado. A nivel general, el Aprendizaje Reforzado resuelve los dos principales problemas del Aprendizaje Supervisado en problemas de toma de decisiones secuenciales. En este caso no existe un supervisor que deba entregar la respuesta correcta frente a cada situación, pero si debe entregar una retroalimentación escalar de acuerdo al desempeño general del sistema. La ventaja de ésta formulación radica en que el diseñador no debe conocer en ningún caso la solución del problema, pero si debe ser capaz de evaluar el resultado final y comparar distintas situaciones. Por otra parte, la retroalimentación no debe ser inmediata, y se puede retroalimentar el resultado de una acción tras varios instantes de tiempo (recompensas retrasadas). Además, no existen datos de entrenamiento previamente al proceso de aprendizaje. Los datos se obtienen de manera secuencial mediante las decisiones del sistema de entrenamiento. De esta manera, el sistema puede aprender sobre la distribución real de datos a la que se verá enfrentado posteriormente.

Una revisión histórica completa del Aprendizaje Reforzado puede encontrarse en [2]. Sin embargo, es importante mencionar que desde sus inicios el Aprendizaje Reforzado ha estado relacionado con otras áreas del conocimiento. Por ejemplo, la formulación matemática y varios métodos específicos de aprendizaje están basados en la programación dinámica desarrollada por Bellman [7] décadas atrás. Por otra parte, la inspiración de un aprendizaje que está basado esencialmente en la prueba y error está fuertemente relacionada con el trabajo presente en el condicionamiento clásico (Pavlov), de donde se origina el aprendizaje por diferencias temporales (*TD-Learning*) [2]. Adicionalmente el desarrollo del Aprendizaje Reforzado ha estado continuamente de la mano con la neurociencia [8], justificando muchas de las técnicas mediante hipótesis desarrolladas en dicho campo, mientras que de manera paralela, la neurociencia valida algunas de sus hipótesis mediante los resultados exitosos del Aprendizaje Reforzado. Finalmente, por ser un método inmerso dentro del Aprendizaje de Máquinas, es común encontrar aplicaciones de aproximación funcional, procesos Gaussianos, métodos de optimización, entre otros, en aplicaciones de Aprendizaje Reforzado. La Figura 2.1 presenta un resumen de algunas de las disciplinas que interactúan constantemente con el Aprendizaje Reforzado.

Figura 2.1: Algunas de las disciplinas que interactúan con el Aprendizaje Reforzado

Finalmente, es importante dar a conocer algunos de los campos donde el Aprendizaje Reforzado ha sido especialmente exitoso, ya sea por el desempeño logrado, o por la capacidad de resolver problemas que otros métodos no logran solucionar. Lo anterior se debe a que para el lector ajeno al tema, el Aprendizaje Reforzado corresponde a una herramienta tanto poderosa como atípica, por lo que puede resultar difícil identificar los problemas a los que puede o debe aplicarse. En particular, el Aprendizaje Reforzado ha demostrado importantes aplicaciones en juegos de mesa [9] [10], videojuegos [4] [5] [11], locomoción robótica [12] [13] [14], manipulación robótica [15], inversiones de portafolios, etc.

2.1.2. Formulación del Aprendizaje Reforzado

El Aprendizaje Reforzado se plantea como una herramienta de resolución de problemas de toma de decisiones secuenciales, del cual se supone clásicamente que se rige mediante un proceso de decisión de *Markov*. En este caso, el proceso queda conformado por los elementos presentes en la Figura 2.2, y puede describirse mediante la tupla $MDP = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$. Adicionalmente, la propiedad de *Markov* en este caso queda descrita como se presenta en la ecuación 2.1.

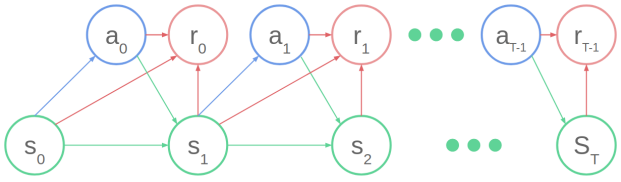


Figura 2.2: Proceso de decisión de *Markov* en toma de decisiones secuenciales.

$$\forall t \quad P(s_t|s_{t-1}, a_{t-1}) = P(s_t|s_{t-1}, ..., s_0, a_{t-1}, ..., a_0) \tag{2.1}$$

De la Figura 2.2, $\mathcal{S}$ y $\mathcal{A}$ representan al espacio de estados y acciones respectivamente. La formulación general no tiene restricciones con respecto a la naturaleza de dichos espacios, pero los trabajos clásicos y las principales garantías de convergencia existen solo para el caso de espacios discretos, en donde no existe necesidad de utilizar aproximadores funcionales. Por otro lado $\mathcal{R}$ representa a la recompensa escalar, pilar fundamental del Aprendizaje Reforzado. Existen diferentes definiciones de la misma, ya sea $\mathcal{R} : \mathcal{S} \to \mathbb{R}$, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ o incluso $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. Finalmente $\mathcal{P}$ representa a la función de transición, que usualmente es considerada estocástica, es decir: $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. Una de las ventajas del Aprendizaje Reforzado con respecto a otras metodologías como el control óptimo, es que precisamente no se asume $\mathcal{P}$ como conocida (y muchos casos nunca se aprende directamente).

Adicionalmente a la formulación anterior, es útil además considerar los elementos presentes en la Figura 2.3, pues permiten entender la separación de responsabilidades al momento de abordar un problema mediante Aprendizaje Reforzado. En esta Figura el ambiente corresponde al problema real a resolver (ya sea físico o virtual), el cual debe ser modelado por el diseñador. Por otro lado el intérprete corresponde al modelamiento del problema, el cual a partir de la información disponible del ambiente, debe inferir una formulación de estados que siga la propiedad de *Markov*, además de producir una señal de recompensa en base al objetivo a resolver. Finalmente, el agente corresponde al algoritmo de aprendizaje, en este caso de Aprendizaje Reforzado. En resumen, se debe identificar el problema a resolver (ambiente), posteriormente este debe ser modelado (intérprete), y finalmente se resuelve mediante el agente.

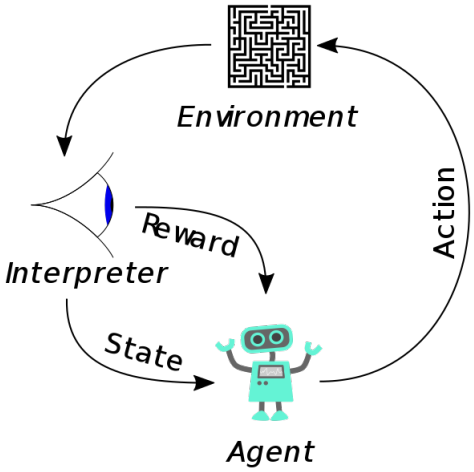


Figura 2.3: Elementos la formulación de Aprendizaje Reforzado (Fuente: *Wikimedia Commons*).

Finalmente, el producto del proceso de aprendizaje, el cuál es además el objeto matemático que directa o indirectamente se modifica durante el entrenamiento corresponde a la política π. La política corresponde a un mapeo entre el espacio de estados y el espacio de

acciones, el cual puede ser determinista $\pi : \mathcal{S} \to \mathcal{A}$ o estocástico $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. En la literatura se suele recomendar la utilización de políticas estocásticas en aquellos problemas en los que se presente observabilidad parcial. Sin embargo estudios recientes, algoritmos que aprenden políticas determinísticas han sido capaces de abordar problemas complejos con observabilidad parcial [16].

2.1.3. Objetivo del Aprendizaje Reforzado

El objetivo del Aprendizaje Reforzado es maximizar la recompensa obtenida por el agente mientras recorre el proceso de *Markov*. Sin embargo existen diversas formulaciones para esta maximización, dependiendo del tipo de problema que se desea resolver. Para aquellos problemas que no sean episódicos, es decir que no tengan condiciones de inicio y términos claras, y para aquellos problemas de larga o infinita duración, el uso de la recompensa acumulada $J = \sum_t r_t$ no es adecuada, debido a que es numéricamente inestable y puede presentar problemas se divergencia. Por otro lado, maximizar la recompensa inmediata produce acciones marginales, sin considerar recompensas futuras. Es por eso que usualmente se utiliza la formulación de la recompensa descontada (ecuación 2.2), que permite evaluar largos horizontes de tiempo, pero otorgando mayor importancia a recompensas cercanas en el tiempo, además de otorgar estabilidad numérica a la optimización.

$$J = \sum_{t=0} \gamma^t r(t) \tag{2.2}$$

El denominado factor de descuento $\gamma \in [0, 1)$ permite controlar el horizonte sobre el cual el agente busca maximizar la recompensa. Factores cercanos a cero generan un comportamiento marginal, asignándole importancia solo a eventos cercanos en el tiempo. Por otro lado, valores cercanos a 1 permiten abordar problemas en los que la recompensa esté fuertemente retrasada en el tiempo. Sin embargo, mientras largo el horizonte determinado por γ, el proceso de aprendizaje resulta más lento, y potencialmente se enfrenta a errores numéricos. En la literatura, para problemas episódicos de naturaleza sencilla, se suele utilizar un factor de descuento γ=0,99.

La formulación de recompensa descontada y su factor de descuento γ permiten abordar problemas en los que se busca generar buenos comportamientos transientes. Sin embargo existen otros problemas de naturaleza periódica como lo es el balance de un sistema o la locomoción de un robot, en los cuales se reportan mejores resultados cuando se utiliza como objetivo de maximización la recompensa promedio $J = \frac{1}{N} \sum_{t=0}^{N-1} r_t$ (con el objetivo de mejorar el comportamiento estacionario).

Las formulaciones anteriores no representan directamente la dependencia del funcional a optimizar con respecto a las acciones que genera la política actual, ni la potencial estocacidad del proceso. Es así que para el caso descontado, el funcional real a optimizar corresponde al presentado en la ecuación 2.3, en la cual $p(s_0) = p_0$ corresponde a la distribución inicial de estados.

$$J(\pi) = \mathbb{E}_{a_t \sim \pi(s_t), s_t \sim P(s_{t-1}, a_{t-1}), s_0 \sim p_0} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \tag{2.3}$$

Adicionalmente a la formulación del funcional a optimizar, es conveniente presentar las denominadas funciones de valor, que se desprenden de la definición de J, y que además son utilizadas frecuentemente en las distintas implementaciones. Estas son la función de valor de estado, presentada en la ecuación 2.4, la función de valor del par estado-acción o valor Q, presentada en la ecuación 2.5, y la función de ventaja, presentada en la ecuación 2.6.

$$V^{\pi}(s) = \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r_t | \pi, s_0 = s \right] \tag{2.4}$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r_t | \pi, s_0 = s, a_0 = a \right] \tag{2.5}$$

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s) \tag{2.6}$$

En las ecuaciones 2.4 y 2.5, τ representa a una trayectoria o también denominada *rollout*, y consiste en la secuencia $\tau = (s_0, a_0, ..., s_n)$, donde n marca el fin de la trayectoria particular. Es importante destacar que la duración de cada trayectoria es variable, y que la distribución de la trayectoria $p(\tau)$ está condicionada por la función de transición del sistema y por la política utilizada.

Es conveniente destacar en este contexto las ecuaciones recursivas de Bellman, que son un pilar fundamental en la totalidad de los algoritmos de Aprendizaje Reforzado que utilizan las funciones de valor. Estas propiedades se presentan en las ecuaciones 2.7 y 2.8, y permiten estimar iterativamente las funciones de valor mediante programación dinámica [7].

$$V^{\pi}(s_t) = \mathbb{E}_{\tau \sim p(\tau)} \left[r_{t+1} + \gamma V(s_{t+1}) \right] \tag{2.7}$$

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\tau \sim p(\tau)} \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) \right] \tag{2.8}$$

Como parte del proceso de aprendizaje, diversos algoritmos de Aprendizaje Reforzado buscan estimar alguna de las funciones de valor presentadas anteriormente. De éstas, la función de valor de estado V corresponde al caso más simple de estimar, debido a que solo depende del estado. Por otro lado, la función de valor del par estado-acción, depende tanto del estado como de la acción, lo que le permite derivar una política mediante la selección de la acción que maximice la función para un estado determinado. Finalmente, la función de ventaja suele ser estimada en aquellos algoritmos que utilizan la diferencia entre el valor Q y la función de valor de estado como parte del gradiente de optimización, y en aquellos casos donde se desee estimar los valores Q a partir de la función de valor de estado.

Las definiciones anteriores sin embargo están formuladas a partir de una política determinada (π), y son comúnmente utilizadas en algoritmos de tipo *on-policy*, que estiman

las funciones anteriores sobre la política ejecutada. En este sentido, también es conveniente definir las funciones de valor óptimas (ecuaciones 2.9, 2.10 y 2.11), que son utilizadas por algunos algoritmos *off-policy* que aprender funciones de valor sobre una política distinta a la ejecutada, en este caso una política óptima π^* (óptima en el sentido de J).

$$V^*(s) = V^{\pi^*}(s) = \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r_t | \pi^*, s_0 = s \right] \tag{2.9}$$

$$Q^*(s,a) = Q^{\pi^*}(s,a) = \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r_t | \pi^*, s_0 = s, a_0 = a \right] \tag{2.10}$$

$$A^*(s,a) = A^{\pi^*}(s,a) = Q^*(s,a) - V^*(s) \tag{2.11}$$

2.1.4. Taxonomía del Aprendizaje Reforzado

Existen diversas metodologías dentro del Aprendizaje Reforzado para poder optimizar el objetivo planteado en la ecuación 2.3. Sin embargo, a nivel general, las distintas implementaciones se pueden distinguir según dos criterios: el uso de un modelo, y el tipo de optimización. Esta categorización puede observarse en la Figura 2.4, mientras que revisiones sobre la literatura de los diferentes métodos y sus aplicaciones pueden encontrarse en [17] [18] y [19]. A continuación se realiza un pequeño resumen de las distintas formulaciones y sus principales características.

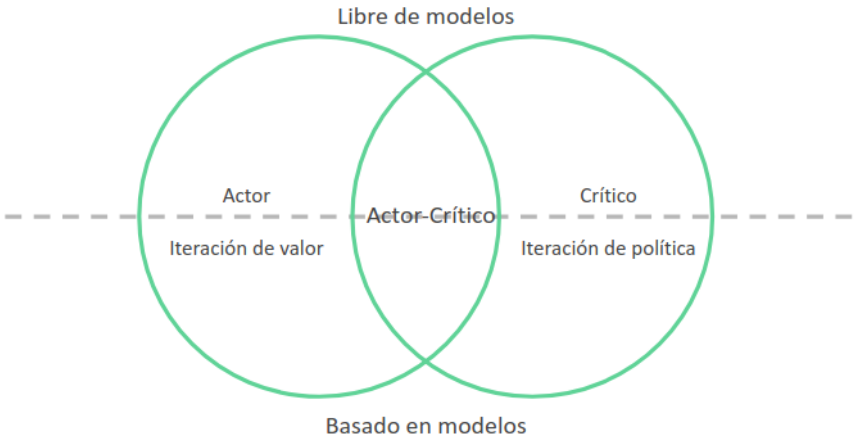


Figura 2.4: Clasificación de métodos de Aprendizaje Reforzado según su naturaleza.

La primera manera de clasificar los algoritmos de Aprendizaje Reforzado corresponde al uso de modelos. De esta manera se distinguen a los algoritmos basados en un modelo (*model-based*), y aquellos libres de modelo (*model-free*). Los algoritmos basados en modelos asumen una estructura parametrizada de la función de transición, la cual es inicialmente desconocida. El objetivo de este tipo de métodos consiste en lograr a partir de las experiencias obtenidas, identificar el modelo. Adicionalmente, de manera paralela o posterior, se busca encontrar

una política óptima (sobre el modelo estimado) para resolver el problema. Las principales ventajas de esta familia de métodos corresponden a una alta eficiencia con respecto al numero de iteraciones necesarias para alcanzar buenos rendimientos. La razón de este fenómeno se debe a que la parametrización del modelo por parte del diseñador suele tener en cuenta el conocimiento del mismo sobre el problema, y que por lo general, dicha parametrización es realizada sobre un espacio reducido de parámetros. Sin embargo, el entrenamiento suele tener que ser realizado de manera *off-line* debido a cómputos complejos (por ejemplo procesos *Gaussianos*). Por otro lado, los algoritmos libres de modelo no conocen, ni buscan conocer la función de transición, y basan su aprendizaje en la derivación, y mejora de una política directamente a partir de experiencias. Sus principales ventajas consisten en que no necesitan conocimiento experto para determinar la parametrización del modelo, y tampoco se ven limitadas por el mismo. Sin embargo, se ven afectados por una baja eficiencia con respecto al uso de datos, principalmente debido a que sus estimaciones se realizan en espacios de mayor dimensionalidad.

La segunda manera de categorizar los algoritmos de Aprendizaje Reforzado corresponde a como formulan la optimización de la ecuación 2.3. Una primera manera de optimizar la ecuación 2.3 es realizarlo directamente como un proceso de optimización sobre la política en la dirección del gradiente de J. Esta formulación asume que la política está parametrizada de manera diferenciable por un vector de parámetros θ, de tal manera las actualizaciones de la política se realizan siguiendo la dirección de $\nabla_\theta J$. Las ventajas de este método consisten en que se puede utilizar una política inicial para acelerar el proceso de aprendizaje a partir de un rendimiento base adecuado, además de poder resolver problemas de acciones continuas de manera natural. Sin embargo las desventajas de esta metodología radican en la estimación de $\nabla_\theta J$, en donde la mayoría de los estimadores presentan una alta varianza, dificultando el proceso de aprendizaje. La segunda categoría dentro de esta clasificación corresponden a los métodos de iteración por valor o *value-based*. Estos algoritmos estiman las funciones de valor presentadas en la Sección 2.1.3, y ejecutan accionas basadas en las predicciones de dichas funciones en conjunto con distintas estrategias de exploración. De esta manera son capaces de derivar políticas a partir de estas estimaciones, y a medida que se mejoran las estimaciones, indirectamente mejoran la política con el objetivo de mejorar J. La ventaja de estos métodos son buenas tasas de aprendizaje, además de tener una noción acerca del rendimiento de la política a partir de las estimaciones (también conocidas como predictores). Sin embargo estos métodos presentan dificultades en el entrenamiento, pues pequeñas perturbaciones en las estimaciones pueden producir grandes cambios en la política subyacente, lo que produce en consecuencias errores en la predicción o estimación de valores. Finalmente, la tercera categoría corresponde a la denominada familia de algoritmos de Actores-Críticos (*Actor-Critic*). Inicialmente formulados en [2], este tipo de algoritmos reconoce las fortalezas y debilidades de los métodos basados en política, los cuales en este contexto son denominados métodos de actor, y los métodos basados en funciones de valor, que en este contexto son conocidos como críticos. La formulación de este tipo de algoritmos plantea un apendizaje paralelo, en el cual se aprenden simultáneamente tanto el crítico como el actor. El crítico tiene como misión estimar o predecir el desempeño del actor, y la labor del actor es mantener una estructura independiente al crítico, que es capaz de realizar el mapeo correspondiente a una política. En este caso el aprendizaje del crítico se realiza de manera tradicional, usualmente mediante diferenciales temporales, mientras que el actor se modifica usualmente a partir de una gradiente derivado por el crítico (este proceso puede observarse en la la Figura 2.5). Las

ventajas de esta formulación mixta consisten en que se adquieren los beneficios de ambos métodos anteriores, aliviando sus desventajas de manera conjunta. En particular se obtiene una manera de abordar problemas de acciones continuas, se habilita la posibilidad de utilizar políticas iniciales provenientes de otros métodos, y se realizan modificaciones suaves sobre la política, facilitando la labor del crítico.

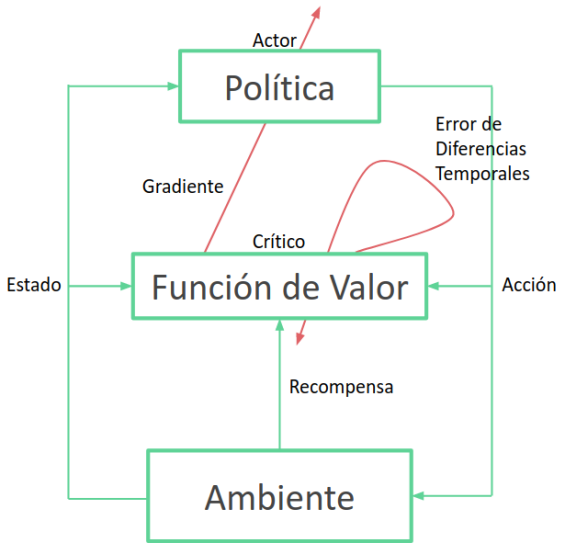


Figura 2.5: Diagrama del entrenamiento de algoritmos de Actor-Crítico.

2.1.5. Aprendizaje Reforzado en Robótica

En la Sección 1.2 se introduce el estado del arte con respecto al uso del Aprendizaje Reforzado en Robótica. Sin embargo, en esta sección se revisita dicho contenido, utilizando los contenidos desarrollados en la Sección 2.1.

El diagrama de la Figura 1.1 presenta los principales bloques, módulos o sistemas que son recurrentes en las diversas aplicaciones de robótica. La relación entre el Aprendizaje Reforzado y la robótica queda expuesta al comparar dicho diagrama con el presentado en la Figura 2.3. En esta comparación se puede apreciar la similitud entre ambos esquemas, en donde se puede identificar al mundo real como el ambiente, al módulo de modelamiento como el intérprete, y al módulo de toma de decisiones o el de actuación como el agente. Esta similitud ha generado que desde sus inicios, el Aprendizaje Reforzado haya estado vinculado fuertemente con la robótica, e incluso se ha llegado a plantear por parte de algunos autores, que la relación entre el Aprendizaje Reforzado y la robótica, es idéntica a la presentada por la Física y la Matemática [3], en cuanto a que la Matemática es la herramienta que permite el desarrollo de la Física.

A lo largo de la historia, se han desarrollado diversas aplicaciones de robótica que utilizan Aprendizaje Reforzado para implementar comportamiento de otra manera difíciles de diseñar o ajustar. En robótica, las aplicaciones más conocidas del Aprendizaje Reforzado han abordado problemas de locomoción de robots [12] [13] [14] y manipulación robótica [15]. Sin embargo, existen numerosas otras aplicaciones tales como la navegación sin mapas [20] [21],

el balance de robots [22], la minimización de energía de movimientos [23] y la generación de acciones de alto nivel [24] [25] [26].

Pese a existir una extensa literatura al respecto, la mayoría de los trabajos anteriormente mencionados realiza simplificaciones de los problemas, con tal de que las herramientas utilizadas sean capaces de abordarlos. Son clásicos las simplificaciones que buscan reducir tanto el espacio de estados como el de acción, utilizan políticas base, o utilizar fuertemente *reward shaping* con el objetivo de acelerar el proceso. Pese a esto, logros recientes dentro del área, tales como el presentado por [14], en donde se aprenden movimientos de locomoción sin *reward shaping*, y actuando sobre el problema de dimensionalidad completa (actuadores de los modelos), hacen pensar que se está frente a un gran cambio de paradigma al abordar problemas mediante Aprendizaje Reforzado. Este tipo de resultados convierten al Aprendizaje Reforzado en un campo activo de investigación en robótica, donde actualmente se estudia como abordar de manera eficiente problemas con observabilidad parcial, la transferencia de conocimiento desde simuladores a robots físicos, estrategias para enfrentar procesos no estacionarios, entre otros.

2.2. Algoritmos tradicionales de Aprendizaje Reforzado

En esta sección se presentan algunos de los enfoques tradicionales en el uso de Aprendizaje Reforzado. En primer lugar, la Sección 2.2.1 presenta la problemática introducida por los espacios continuos, y como se abordan clásicamente. Posteriormente las Secciones 2.2.2 y 2.2.3 presentan algunos de los algoritmos más conocidos en el área, con algunas de las modificaciones que se les han realizado a través de los años. Más adelante, la Sección 2.2.4 presenta un caso básico de algoritmo de actor crítico y sus propiedades. Finalmente, la Sección 2.2.5 presenta el Aprendizaje Reforzado Descentralizado, una técnica que utiliza conceptos de la teoría de sistemas multi-agentes, para formular una manera distinta de abordar problemas de espacios de acciones con más de una dimension.

2.2.1. Aproximación Funcional

La aproximación funcional está presente en casi todas las implementaciones reales de Aprendizaje Reforzado, excepto en las más sencillas. Si bien la formulación básica del Aprendizaje Reforzado no realiza supuestos mayores con respecto a la naturaleza de los espacios de estado y acción, muchos de los algoritmos clásicos los asumen de naturaleza discreta, como lo es el caso de *Q-Learning* [27] y *SARSA* [28]. Adicionalmente, la mayoría de los trabajos clásicos que estudian la convergencia de algoritmos de aprendizaje, solo garantizan cotas de convergencia para el caso discreto.

Sin embargo, en la práctica los problemas que resultan de interés, presentan espacios de estados continuos, y en muchas ocasiones también espacios de acciones continuos. Una manera sencilla de sobrellevar esta situación corresponde a discretizar dichos espacios en celdas, lo que permite utilizar algoritmos sobre espacios discretos. Si bien este enfoque es teóricamente correcto, suele obtener pobres desempeños y se vuelve rápidamente infactible según la dimensionalidad del problema.

Esta situación genera la necesidad de incorporar aproximaciones funcionales a los algoritmos de aprendizaje, siendo comúnmente los objetos de aproximación la política $\pi_\theta(s) = \pi(s, \theta)$, y las funciones de valor $V_\theta(s) = V(s, \theta)$ y $Q_\theta(s, a) = Q(s, a, \theta)$.

Uno de los tipos de aproximación funcional más utilizados en las implementaciones de Aprendizaje Reforzado tradicional consiste en los métodos de aproximación lineal, debido a las garantías de convergencia existentes [29], siendo el método más popular el basado en aproximaciones mediante funciones de base radial o *Radial Basis Functions(RBF)*, debido su propiedad de aproximador universal de funciones [30].

En en el caso de uso de aproximadores funcionales lineales, los objetos de estudio del aprendizaje reforzado pueden ser escritos como se presentan en las ecuaciones 2.12 y 2.13, en las cuales ϕ_i representan a las denominadas funciones base o características, mientras que θ y φ representan los vectores de parámetros de la aproximación funcional.

$$\pi_{\varphi}(s) = \phi(s)^T \varphi \tag{2.12}$$

$$V_{\theta}(s) = \phi(s)^T \theta \tag{2.13}$$

En la literatura se presentan distintas funciones base para realizar la aproximación funcional [31] [32], siendo ejemplos clásicos el uso de funciones de base radial, bases de *Fourier* [33], polinomios, entre otros. Sin embargo, pese a que el uso de aproximadores lineales presenta propiedades útiles en términos de implementación y produce procesos de entrenamiento estables, sufre fuertemente de la maldición de la dimensionalidad [7], permitiendo abordar únicamente problemas de relativa baja dimensionalidad (usualmente entre 1 y 5 dimensiones). Por esta razón, en aplicaciones prácticas es común el uso de *Tile Coding* como aproximación funcional, debido a que si bien también consiste en un aproximador lineal, genera un vector de funciones base o características de tipo *sparse*, lo que acelera notablemente lo cálculos computacionales. Sin embargo, *Tile Coding* corresponde un caso de características binarias, lo que reduce el poder de generalización del aproximador, y un método intermedio corresponde al uso de funciones base de soporte finito (*Finite Support Basis Functions*) [26], que son funciones base que generan características de tipo *sparse*, es decir computacionalmente eficientes, pero de una naturaleza similar a las de base radial, de las cuales existen estudios extensos acerca de sus propiedades de generalización.

Pese a poseer deseables propiedades matemáticas, los aproximadores funcionales lineales limitan en gran medida la cantidad de problemas a resolver. Es por dicha razón, que en la literatura existen diversos trabajos que buscan establecer el uso de aproximadores no lineales o incluso no paramétricos en el Aprendizaje Reforzado. El primer ejemplo corresponde a los procesos *Gaussianos*, que si bien obtienen excelentes resultados y son altamente eficientes con respecto al uso de datos, no son escalables en su dimensionalidad, y tampoco resultan prácticos, debido a que la ejecución de la política se realiza mediante la ejecución del proceso *Gaussianos* completo, que comprende exhaustivos cálculos computacionales. Otra alternativa corresponde al uso de aproximadores lineales con bases adaptativas [34]. Ese tipo de método busca utilizar funciones bases lineas dinámicas, es decir que modifican sus propiedades a lo largo del entrenamiento con el fin e cubrir los espacios de estado y/o acción de una manera eficiente, dado un presupuesto dado en cantidad de funciones base. Si bien esta metodología permite abordar muchos de los problemas mencionados anteriormente, los autores solo la validan con problemáticas sencillas y de baja dimensionalidad, sin dar pruebas claras de su aplicabilidad a situaciones más complejas. Un último caso de aproximación funcional no lineal corresponde al uso de redes neuronales, en especial aquellas de perceptrón multi capa (*Multi Layer Perceptron* o *MLP*), debido a que su capacidad de comprimir información, a la vez de tener un gran potencial de representación, le permiten en teoría escalar de manera adecuada en la dimensionalidad de los problemas. En [35] se realizan experimentos utilizando *MLP* como aproximador funcional, donde sin embargo se presentan resultados adversos a su uso en Aprendizaje Reforzado. El uso de *batch* cada vez más grandes dificulta su entrenamiento en situaciones reales, y el entrenamiento resulta inestable, presentando en algunos casos la divergencia de los estimadores de valores de estado.

2.2.2. Q-Learning

Q-Learning [27] es uno de los algoritmos más antiguos y conocidos en el área de Aprendizaje Reforzado y pertenece a las categorías libre de modelo y basado en funciones de valor. Está basado en el aprendizaje por diferencias temporales (*TD-Learning*) tanto para la labor de predicción (estimación de la función de valor), como para la acción de control (elección de acción). Debido a la necesidad de derivar una política a partir de las predicciones realizadas, se debe estimar directamente la función de valor del par estado-acción, en vez de simplemente estimar la función de valor de estado.

El objetivo de *Q-Learning* es estimar (o predecir) la función de valor del par estado-acción o valor Q en su formulación óptima (ver ecuación 2.10). El error de diferencia temporal en este caso queda dado por la ecuación 2.14, y la regla de actualización queda dada por la ecuación 2.15, en donde la predicción supone una ejecución de acciones (control) avaras en función de las estimaciones disponibles. Debido a que este algoritmo estima los valores provenientes a una política distinta a la ejecutada realmente, este algoritmo entra dentro de la categoría de algoritmos *off-policy*.

$$TD = r + \max_{a' \in \mathcal{A}} \gamma Q(s', a') - Q(s, a) \tag{2.14}$$

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha TD \tag{2.15}$$

Para el caso discreto existen fuertes garantías de convergencia de *Q-Learning* a Q^*. En particular, converge con probabilidad 1 si la tasa de aprendizaje α disminuye constantemente durante el entrenamiento y todos los pares $(s, a) \in \mathcal{S} \times \mathcal{A}$ se visitan de manera frecuente. Este último requisito requiere una exploración constante sobre la política ejecutada (distinta a la elección avara) que permita visitar todos los pares estado-acción. Sin embargo, debido a que ejecuta una acción distinta a la estimada, el rendimiento *online* de *Q-Learning* suele ser inferior al estimado, presentando potenciales situaciones peligrosas durante la exploración (en experimentos reales). Un esquema básico del funcionamiento de *Q-Learning* puede observarse en el Algoritmo 1.

El Algoritmo 1 permite resolver solo una pequeña cantidad de problemas (ver Sección 2.2.1). Sin embargo, el uso de aproximaciones funcionales para abordar espacios continuos resulta natural para *Q-Learning*. En este caso se busca estimar de manera iterativa la función de valor del par estado-acción, buscando minimizar el error cuadrático medio de la predicción sobre la distribución de los pares estado-acción inducidos por la política derivada de la estimación (ver ecuación 2.16).

$$MSE(\theta) = \mathbb{E}\left[Q(s, a) - \hat{Q}(s, a)\right]^2 \tag{2.16}$$

Luego, se puede utilizar cualquier método de optimización, siendo el más común el método de gradiente descendente, que en este caso tiene la forma presentada en la ecuación 2.17, la cual sin embargo suele ser descrita en términos del error de diferencia temporal (ecuación

Algoritmo 1 Q-Learning para el caso tabular

Entrada: N Número de episodios.
α Tasa de aprendizaje.
γ Factor de descuento.
π_{expl} Método de exploración (e.g $\epsilon - greedy$).
$\mathcal{S} \subseteq \mathbb{N}$ Espacio de estados.
$\mathcal{A} \subseteq \mathbb{N}$ Espacio de acciones.

Salida: Q.

Inicializar de manera arbitraria $Q \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$
para $n = 1$ hasta N **hacer**
  Inicializar s (condición inicial)
  mientras no *terminal* **hacer**
    $a \sim \pi_{epl}(Q(s, \cdot))$
    Ejecutar a. Observar r, s', *terminal*
    si terminal **entonces**
      $y \leftarrow r$
    si no
      $y \leftarrow r + \gamma \underset{a' \in \mathcal{A}}{max} Q(s', a')$
    fin si
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[y - Q(s, a)\right]$
    $s \leftarrow s'$
  fin mientras
fin para
devolver Q

2.18). Finalmente en el caso de aproximación lineal, la regla de actualización queda dada por la ecuación 2.19, donde se asume que el objetivo no depende del vector de parámetros. Pese a no estar justificada esta suposición, en la literatura se suele obviar este hecho.

$$\theta \leftarrow \theta - \frac{1}{2}\alpha\nabla_\theta \left[Q(s,a) - \hat{Q}(s,a)\right]^2 \tag{2.17}$$

$$\theta \leftarrow \theta + \alpha \left[r + \gamma \underset{a' \in \mathcal{A}}{max}\hat{Q}(s',a') - \hat{Q}(s,a)\right] \nabla_\theta \hat{Q}(s,a) \tag{2.18}$$

$$\theta_a \leftarrow \theta_a + \alpha \left[r + \gamma \underset{a' \in \mathcal{A}}{max}\hat{Q}(s',a') - \hat{Q}(s,a)\right] \phi(s) \tag{2.19}$$

Otra modificación que se le puede a realizar a *Q-Learning* es el uso de trazas de elegibilidad, provenientes de la formulación de $TD(\lambda)$ [2]. Esta formulación permite acelerar el proceso de aprendizaje considerablemente, mediante una mejora en la asignación de recompensas a acciones atrás en el tiempo. Si bien los métodos de diferenciales temporales tienen la capacidad innata de propagar estas asignaciones atrás en el tiempo, requieren una cantidad considerable de visitas a dichos estados, lo que produce en algunos casos nula o lenta convergencia. Por otro lado, el uso de trazas de elegibilidad o $TD(\lambda)$ permite encontrar un punto medio entre algoritmos de Monte Carlo y el aprendizaje por diferencias tradicional o en este contexto también denominado $TD(0)$, debido a que el aprendizaje se realiza en cada iteración sin necesidad de esperar al final del episodio para asignar créditos y realizar actualizaciones en los parámetros, pero a la vez permite actualizar más de un paso atrás en el tiempo, por cada iteración. La idea detrás de $TD(\lambda)$ puede observarse en la Figura 2.6, donde δ_t representa el error de diferencias temporales en el instante t que se propaga hacia pares de estado-acción anteriores, con un decaimiento exponencial sobre la magnitud del error de diferencias temporales.

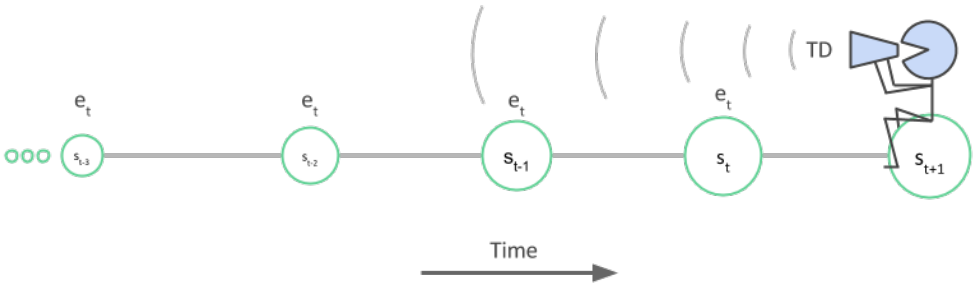


Figura 2.6: Idea detrás de $TD(\lambda)$.

Con las modificaciones realizadas por las aproximaciones funcionales, y el uso de trazas de elegibilidad, se puede formular una versión más avanzada del algoritmo *Q-Learning*, la cual es presentada en el Algoritmo 2.

Algoritmo 2 Q-Learning(λ) mediante trazas con reemplazo y aproximación funcional lineal

Entrada: N Número de episodios.
α Tasa de aprendizaje.
γ Factor de descuento.
λ Factor de decaimiento de las trazas de elegibilidad.
π_{expl} Método de exploración (e.g $\epsilon - greedy$).
f Extractor de características. $f : \mathcal{S} \to \mathbb{R}^m$
$\mathcal{S} \subseteq \mathbb{R}^S$ Espacio de estados.
$\mathcal{A} \subseteq \mathbb{N}$ Espacio de acciones.

Salida: $(\theta_0, ..., \theta_{|\mathcal{A}-1|})$.

Inicializar de manera arbitraria de los parámetros $\theta_a \ \forall a \in \mathcal{A}$
para $n = 1$ hasta N **hacer**
  Inicializar trazas de elegibilidad $\forall a \in \mathcal{A} \ e_a = 0_m$
  Inicializar s (condición inicial)
  $\phi \leftarrow f(s)$
  mientras no *terminal* **hacer**
    $\forall a \in \mathcal{A} \ Q_a \leftarrow \theta_a^T \phi'$
    $a \sim \pi_{epl}(Q)$
    Ejecutar a. Observar r, s', *terminal*
    $\phi' \leftarrow f(s')$
    $\forall a' \in \mathcal{A} \ Q'_{a'} \leftarrow \theta_{a'}^T \phi'$
    Cálculo de la predicción del valor estado-acción
    si terminal **entonces**
      $y \leftarrow r$
    si no
      $y \leftarrow r + \gamma \underset{a' \in \mathcal{A}}{max} Q'_{a'}$
    fin si
    Actualización de las trazas de elegibilidad
    $\forall a \in \mathcal{A} \ e_a \leftarrow \gamma \lambda e_a$
    $e_a \leftarrow e_a + \phi$
    Actualización de parámetros
    $\theta_a \leftarrow \theta_a + \alpha e_a \left[y - Q_a \right]$
    $\phi \leftarrow \phi'$
  fin mientras
fin para
devolver $(\theta_0, ..., \theta_{|\mathcal{A}-1|})$.

Una limitación en cuanto a la aproximación funcional posible en *Q-Learning* consiste en que únicamente se puede aproximar el espacio de estados. Este fenómeno se debe a que en la regla de predicción de *Q-Learning* se utiliza el valor máximo entre las distintas acciones, proceso que no resulta sencillo de determinar al usar acciones continuas. Si bien es teóricamente posible determinar dicho valor mediante optimización numérica de manera *online*, la complejidad extra genera que se prefieran otros algoritmos para tratar con espacios continuos. Una consecuencia directa de esta limitación, consiste en que para realizar la aproximación lineal entre un espacio continuo y uno discreto, se suele realizar un aproximador funcional

independiente para cada acción, lo que aumenta los costos computacionales al momento de ejecución pues se deben realizar $|\mathcal{A}|$ aproximaciones funcionales para poder determinar la acción a ejecutar.

2.2.3. Sarsa

Al igual que *Q-Learning*, *SARSA* [28] es un algoritmo libre de modelos, cabe dentro de la categoría de estimación de funciones de valor, y basa su aprendizaje en las diferencias temporales. La gran diferencia entre *Q-Learning* y *SARSA* consiste en que *SARSA* corresponde a un algoritmo *on-policy*, es decir, estima las funciones de valor correspondientes a la política que es realmente ejecutada (estima directamente Q^{π}). Es este caso, la fórmula de diferencias temporales queda determinada por la ecuación 2.20, donde a' corresponde a la acción que realmente se ejecuta.

$$TD = r + \gamma\, Q(s', a') - Q(s, a) \tag{2.20}$$

El aprendizaje realizado en *SARSA* corresponde a una secuencia iterativa entre mejorar las estimaciones de la función de valor estado-acción, y mejorar la política que se deriva de ella. Para lograr este fenómeno se deben utilizar estrategias (políticas) que ejecuten tanto acciones avaras (que maximice el valor del par estado-acción), como acciones aleatorias. De esta manera durante el entrenamiento se puede determinar que la acción producto de una acción aleatoria reporta mayores retornos que la acción previamente considerada como óptima, modificando la política para subsiguientes iteraciones.

Las garantías de convergencias existentes para *SARSA* son similares a las que presenta *Q-Learning*, es decir convergencia con probabilidad 1 al usar tasas de aprendizaje decrecientes, y una visita continua a todos los pares estado-acción. Adicionalmente, se requiere que el método de exploración degenere en acciones avaras a lo largo del proceso de entrenamiento (e.g decaimiento de ϵ en exploración $\epsilon - greedy$), para que Q^{π} converja de manera exitosa en Q^{*}.

Los Algoritmos 3 y 4 presentan los esquemas correspondientes a la versión básica de *SARSA* en espacios discretos, y a la versión que utiliza aproximación funcional lineal y trazas de elegibilidad respectivamente. Es importante destacar que a nivel de implementación, *Q-Learning* y *SARSA* presentan implementaciones muy similares, y son pequeñas sutilezas las que marcan la gran diferencia entre algoritmos *on-policy* y *off-policy*.

Pese a poseer implementaciones similares, las diferencias teóricas marcan importantes diferencias en su funcionamiento. Debido a que *Q-Learning* estima acciones óptimas, pero debe realizar exploración para lograr convergencia, la acciones realizadas por este algoritmo no tienen en consideración el posible riesgo que corre el agente al realizar dicha acción. Por otro lado, al ser *on-policy*, *SARSA* sí tiene en consideración los efectos del mecanismo de exploración dentro de su predicción, por lo que las acciones que ejecuta suelen tener en consideración los posibles riesgos que producen. Esta fenómeno es de suma importancia en

Algoritmo 3 *SARSA* para el caso tabular

Entrada: N Número de episodios.
α Tasa de aprendizaje.
γ Factor de descuento.
π_{expl} Método de exploración (e.g $\epsilon - greedy$).
$\mathcal{S} \subseteq \mathbb{N}$ Espacio de estados.
$\mathcal{A} \subseteq \mathbb{N}$ Espacio de acciones.
Salida: Q.

Inicializar de manera arbitraria $Q \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$
para $n = 1$ hasta N **hacer**
  Inicializar s (condición inicial)
  $a \sim \pi_{epl}(Q(s, \cdot))$
  mientras no *terminal* **hacer**
    Ejecutar a. Observar r, s', *terminal*
    $a' \sim \pi_{epl}(Q(s, \cdot))$
    si terminal **entonces**
      $y \leftarrow r$
    si no
      $y \leftarrow r + \gamma\, Q(s', a')$
    fin si
    $Q(s, a) \leftarrow Q(s, a) + \alpha\,[y - Q(s, a)]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  fin mientras
fin para
devolver Q

Algoritmo 4 $SARSA(\lambda)$ mediante trazas con reemplazo y aproximación funcional lineal

Entrada: N Número de episodios.
α Tasa de aprendizaje.
γ Factor de descuento.
λ Factor de decaimiento de las trazas de elegibilidad.
π_{expl} Método de exploración (e.g $\epsilon - greedy$).
f Extractor de características. $f : \mathcal{S} \to \mathbb{R}^m$
$\mathcal{S} \subseteq \mathbb{R}^S$ Espacio de estados.
$\mathcal{A} \subseteq \mathbb{N}$ Espacio de acciones.

Salida: $(\theta_0, ..., \theta_{|\mathcal{A}-1|})$.

Inicializar de manera arbitraria de los parámetros $\theta_a \ \forall a \in \mathcal{A}$
para $n = 1$ hasta N **hacer**
  Inicializar trazas de elegibilidad $\forall a \in \mathcal{A} \ e_a = 0_m$
  Inicializar s (condición inicial)
  $\phi \leftarrow f(s)$
  $\forall a \in \mathcal{A} \ Q_a \leftarrow \theta_a^T \phi'$
  $a \sim \pi_{epl}(Q)$
  mientras no *terminal* **hacer**
    Ejecutar a. Observar r, s', *terminal*
    $\phi' \leftarrow f(s')$
    $\forall a' \in \mathcal{A} \ Q'_{a'} \leftarrow \theta_{a'}^T \phi'$
    $a' \sim \pi_{epl}(Q')$
    Cálculo de la predicción del valor estado-acción
    si terminal **entonces**
      $y \leftarrow r$
    si no
      $y \leftarrow r + \gamma \, Q'_{a'}$
    fin si
    Actualización de las trazas de elegibilidad
    $\forall a \in \mathcal{A} \ e_a \leftarrow \gamma \lambda e_a$
    $e_a \leftarrow e_a + \phi$
    Actualización de parámetros
    $\theta_a \leftarrow \theta_a + \alpha e_a \left[y - Q_a \right]$
    $\phi \leftarrow \phi'$
    $a \leftarrow a'$
  fin mientras
fin para
devolver $(\theta_0, ..., \theta_{|\mathcal{A}-1|})$.

robótica [36], donde la seguridad del agente físico juega un rol importante, y razón para que en dicho campo se prefieran ampliamente los algoritmos de tipo *on-policy*.

2.2.4. Actor-Crítico Lineal

Los algoritmos *Q-Learning* y *SARSA* son ejemplos clásicos de métodos de tipo crítico, y con las modificaciones presentadas en las Secciones 2.2.2 y 2.2.3, dichos algoritmos son capaces de enfrentarse a problemas de estados continuos, pero solo permiten utilizar acciones discretas. En la práctica esto produce dificultad al abordar problemas de naturaleza continua, como lo es usual en el ámbito de la robótica, debido a la necesidad de discretizar el espacio de acciones, y como se discute en la Sección 2.2.2, induce en altos costos computacionales, debido a a la necesidad de realizar una aproximación funcional independiente para cada posible acción discreta.

En esta sección se introduce una implementación de algoritmo de tipo Actor-Crítico, que cumple la arquitectura presentada en la Figura 2.5, y está basado en la implementación propuesta en [37], pero utiliza trazas de elegibilidad tal como se propone en [19].

El fundamento de este algoritmo se basa en la estimación de una política (actor) determinista (aunque en [37] se menciona que puede ser estocástica), la cual se perturba constantemente, usualmente con ruido *Gaussiano* para términos de exploración. La labor del crítico en este caso es estimar de manera *on-policy* la función de valor de estados de la política ejecutada por el actor, la cual se aprende utilizando diferencias temporales, mientras que el actor aprende mediante una estimación del gradiente proveniente del crítico, en conjunto con diferencias temporales. Las principales ventajas de este algoritmo es que permite utilizar acciones continuas y potencialmente multidimensionales, al solo estimar la función de valor de estado. Además, es computacionalmente eficiente, debido a que solo deben realizar una aproximación funcional por cada dimensión del espacio de acciones de la política, y una aproximación funcional para el crítico. El esquema completo de esta implementación se presenta en el Algoritmo 5, donde es común modular la amplitud del ruido a medida que progresa el aprendizaje (en el caso de exploración *Gaussiana*, se disminuye paulatinamente su varianza).

2.2.5. Aprendizaje Reforzado Descentralizado

En aplicaciones de robótica, uno de los problemas o limitaciones recurrentes consiste en el gran numero de interacciones con el ambiente para obtener buenos rendimientos, sobre todo en espacios de estados y acciones multi-dimenisonales. Para abordar este problema, en [38] se propone el uso de Aprendizaje Reforzado Descentralizado, que consiste en una formulación distinta a la tradicional basada en la teoría de sistemas multi-agentes, en la que cada dimensión del espacio de acciones (continua o discreta) es tratada por un agente individual de manera independiente.

Esta formulación permite diseñar la solución o agente para cada dimensión del espacio de estados de manera independiente, pudiendo utilizar en algunos casos incluso agentes de aprendizaje correspondientes a metodologías fuera del Aprendizaje Reforzado. Además, el

Algoritmo 5 Actor-Crítico, trazas con reemplazo y aproximación funcional lineal

Entrada: N Número de episodios.
α_C Tasa de aprendizaje para el crítico.
α_A Tasa de aprendizaje para el actor.
γ Factor de descuento.
λ Factor de decaimiento de las trazas de elegibilidad.
$\mathcal{N}$ Distribución de ruido muestreable para generar exploración.
f Extractor de características. $f : \mathcal{S} \to \mathbb{R}^m$
$\mathcal{S} \subseteq \mathbb{R}^S$ Espacio de estados.
$\mathcal{A} \subseteq \mathbb{R}^A$ Espacio de acciones.

Salida: (θ, φ).

Inicializar de manera arbitraria de los parámetros $\varphi \in \mathbb{R}^A \times \mathbb{R}^m$ y $\theta \in \mathbb{R}^m$
para $n = 1$ hasta N **hacer**
  Inicializar trazas de elegibilidad $e = 0_m$ ($e \in \mathbb{R}^m$)
  Inicializar s (condición inicial)
  $\phi \leftarrow f(s)$
  $a \leftarrow \varphi\phi + n$ ($n \sim \mathcal{N}$)
  mientras no *terminal* **hacer**
    Ejecutar a. Observar r, s', *terminal*
    $\phi' \leftarrow f(s')$
    $a' \leftarrow \varphi\phi' + n$ ($n \sim \mathcal{N}$)
    Cálculo de la predicción del valor de estado
    si terminal **entonces**
      $y \leftarrow r$
    si no
      $y \leftarrow r + \gamma\, \theta^T \phi'$
    fin si
    Actualización de las trazas de elegibilidad
    $e \leftarrow \gamma\lambda e + \phi$
    Actualización de parámetros
    $\theta \leftarrow \theta + \alpha_C e \left[y - \theta^T \phi\right]$
    $\varphi \leftarrow \varphi + \alpha_A e\, a \left[y - \theta^T \phi\right]$
    $\phi \leftarrow \phi'$
    $a \leftarrow a'$
  fin mientras
fin para
devolver (θ, φ).

uso de agentes independientes, permite diseñar la recompensa de cada agente, de tal manera de facilitar el proceso de aprendizaje. Por otro lado, la separación entre agentes permite utilizar herramientas de computación paralela para acelerar cálculos, y permite ahorrar importantes recursos computacionales, tanto en memoria como en procesamiento [39] [26]. Sin embargo, la formulación del Aprendizaje Reforzado Descentralizado rompe muchas de las hipótesis del Aprendizaje Reforzado tradicional, en especial a las realizadas sobre el proceso de *Markov* subyacente. Debido a que cada agente actúa de manera independiente, sin conocer las acciones de los demás agentes descentralizados, el proceso pierde la propiedad de *Markov*. Adicionalmente, debido a que los agentes modifican su comportamiento a lo largo del entrenamiento, el proceso también deja de ser estacionario.

Pese a las dificultades teóricas, en los estudios realizados por [39] y [26], se demuestra empíricamente la efectividad del Aprendizaje Reforzado Descentralizado, aplicado en problemas relacionados a la robótica. En particular, en [39] se entrena un agente directamente en un robot físico, permitiendo evaluar el Aprendizaje Reforzado Descentralizado en problemas reales, con un alto nivel de ruido, y en [26], se estudian además los beneficios computacionales del uso de esta metodología.

2.3. Algoritmos de Deep Reinforcement Learning

Los algoritmos y metodologías clásicas de Aprendizaje Reforzado permiten abordar correctamente problemas sencillos, o de relativa baja dimensionalidad. Sin embargo, las malas propiedades de escalabilidad presentados por las técnicas clásicas limitan fuertemente las aplicaciones reales, a la vez que reduce el interés sobre Aprendizaje Reforzado en la comunidad científica.

Es en este contexto que el Aprendizaje Reforzado Profundo o *Deep Reinforcement Learning* revoluciona el campo del Aprendizaje Reforzado mediante los resultados obtenidos en una gama muy amplia de problemas, tales como videojuegos [4] [5], juegos de mesa [10] y problemas de locomoción [14]. A nivel general, los beneficios producto del uso de *Deep Reinforcement Learning* son un conjunto de herramientas escalables tanto en complejidad computacional como en cantidad de datos con respecto a la dimensionalidad del problema, procesos de entrenamiento estables y estrategias prácticas de abordar problemas con observabilidad parcial.

Es importante mencionar que pese a los numerosos éxitos y desarrollos en el área de *Deep Reinforcement Learning*, pocos de ellos son exclusivos de esta metodología, siendo los principales ejemplos de este caso el uso de *Experience Replay* [4] y *Target Networks* [5] [6]. El mecanismo de *Experience Replay* aprovecha el concepto de algoritmos *off-policy* para utilizar experiencias previas en forma de *mini-batches*, mejorando notoriamente la eficiencia de datos, convergencia y estabilidad. Por otro lado el concepto de *Target Networks* permite otorgar consistencia y estabilidad al momento de realizar optimizaciones sobre el error de diferencia temporal. Fuera de estos dos principales desarrollos, la mayoría de los demás avances consisten en adaptaciones de herramientas desarrolladas en el ámbito de las redes neuronales, usualmente posteriores al nacimiento de *Deep learning*, la cual es una familia de técnicas dentro del área de redes neuronales que busca aprender representaciones directamente a partir de de datos. Este es el caso de optimizadores con mejores tasas de convergencia que el gradiente descendente, como es el caso de *RMSProp* y *Adam* [40], herramientas de normalización como *Batch Normalization* [41], y métodos para abordar el desvanecimiento del gradiente en redes neuronales como *ReLU* [42].

En la actualidad, sigue siendo incierto cual es el límite de esta nueva propuesta. En particular, luego de utilizar *Deep Reinforcement Learning* como uno de los componentes esenciales de *AlphaGO* [10], se ha planteado como nuevo desafío el popular videojuego *StarCraft II* [43], que está caracterizado, a diferencia del *GO* por una baja observabilidad, dinámicas complejas, espacios de estado y acción mucho mayores, y la presencia de múltiples agentes como elementos de juego. En el ámbito de la robótica son objetos de especial interés estudiar problemas con observabilidad parcial [44] [16] [11], procesos dinámicos y potencialmente adversariales, y finalmente cuales son las condiciones o herramientas que permiten realizar la transferencia de políticas de manera exitosa entre simuladores y ambientes reales [45] [46]. Un resumen completo acerca de los logros, implementaciones y desafíos futuros relacionados a *Deep Reinforcement Learning* puede encontrarse en [47].

Las subsecciones siguientes presentan algunos de los componentes más importantes o recurrentes dentro de *Deep Reinforcement Learning*. En particular, las Secciones 2.3.1 y 2.3.2

presentan los mecanismos de *Experience Replay* y *Target Networks*, componentes esenciales para la mayoría de las implementaciones conocidas. Posteriormente, en las Secciones 2.3.3 y 2.3.4 se presentan los algoritmos *Deep Q-Learning* y *Deep Deterministic Policy Gradients* respectivamente, que corresponden a los primeros y más conocidos algoritmos de *Deep Reinforcement Learning*. Finalmente las Secciones 2.3.5 y 2.3.6 abordan temas particulares relacionados a la convergencia de los algoritmos, y a los problemas con observabilidad parcial.

2.3.1. Experience Replay

Experience Replay consiste en un mecanismo diseñado para entrenar agentes de Aprendizaje Reforzado, utilizando experiencias previas en el tiempo. Inicialmente formulado en [48], y posteriormente popularizado por el trabajo realizado en [4] décadas después, corresponde a una de las principales técnicas que dieron origen al denominado *Deep Reinforcement Learning*.

Al utilizar aprendizaje por diferenciales temporales con aproximadores funcionales, el objetivo del aprendizaje es reducir el error temporal sobre la distribución de los pares de estado-acción observados por el agente, y derivados por la política, tal como se presenta en la ecuación 2.16. Pese a formularse como la optimización sobre la esperanza de una variable aleatoria, la mayoría de los algoritmos como *Q-Learning* o *SARSA* realizan el aprendizaje de manera *online*, es decir, actualizan los parámetros en dirección del gradiente, por cada paso en el tiempo, muestreando el error de la ecuación 2.16 a partir de una única experiencia (tupla (s, a, r, s')). Si bien esta metodología presenta buenos resultados con aproximadores lineales, genera numerosas dificultades al considerar aproximadores no lineales, como lo son las redes convolucionales, o los perceptrones multi-capa. En [4] se argumenta que la razón de este suceso se explica debido a la alta correlación temporal de la naturaleza de las actualizaciones, además de un muestreo ineficiente de la ecuación 2.16.

A diferencia de los métodos *online*, el mecanismo de *Experience Replay* consiste en el uso de una base de datos finita denominada *Replay Memory*. Al utilizar *Experience Replay*, las experiencias que observa de agente son almacenadas en el *Replay Memory*, que consiste básicamente en un arreglo circular. Posteriormente, se muestrea un *mini-batch* de tuplas (s, a, r, s), con las que se estima el error de diferencias temporales de la ecuación 2.16, y se actualizan los parámetros siguiendo la dirección del gradiente.

El mecanismo de *Experience Replay* tiene validación en la neurociencia [5], donde se cree que al dormir, el cerebro revisita experiencias pasadas para aprender. Finalmente se debe destacar que existe un compromiso entre la frecuencia de ejecución del mecanismo de *Experience Replay* y el tamaño del *mini-batch*, debido a que dado un presupuesto computacional fijo, una mayor cantidad de actualizaciones permite una mayor velocidad de convergencia, mientras que un mayor tamaño de *mini-batch* permite realizar mejores estimaciones del error de diferencia temporal.

2.3.2. Target Networks

Uno de los aspectos negativos del uso de aproximación funcional en el aprendizaje por diferencias temporales, es el presentado en la ecuación 2.19, en donde el valor de predicción se considera fijo durante la optimización. El concepto de *Target Networks* busca abordar este fenómeno, mediante el uso de dos conjuntos de parámetros θ y θ'. De esta manera el error de diferencia temporal queda dado por la ecuación 2.21, donde efectivamente el valor de la predicción es constante, mientras se actualiza θ. Finalmente para que el objetivo de la optimización tenga sentido, cada n actualizaciones de θ, se igualan los parámetros $\theta' \leftarrow \theta$.

$$TD = r + \gamma \underset{a' \in \mathcal{A}}{max}\, Q_{\theta'}(s', a') - Q_{\theta}(s, a) \tag{2.21}$$

Pese a que la técnica de *Target Networks* es sencilla tanto en su formulación como en su implementación, reporta importantes mejoras en el rendimiento del algoritmo *Deep Q-Learning* [5]. Desde su formulación, ha sido utilizado en posteriores implementaciones tales como *Double Deep Q-Learning* [49] y *Dueling Network Architectures* [50]. Adicionalmente, en [6] se extiende el mecanismo de actualización de las *Target Networks* mediante actualizaciones parciales (*soft-updates*) como se presenta en la ecuación 2.22, donde $\tau \ll 1$. En dicho trabajo además se comprueba su desempeño en arquitecturas de Actor-Crítico, donde el concepto de *Target Networks* también juega un rol fundamental en el proceso de aprendizaje.

$$\theta' \leftarrow \theta'(1 - \tau) + \theta\tau \tag{2.22}$$

2.3.3. Deep Q-Learning

Deep Q-Learning [4] consiste en el primer algoritmo de *Deep Reinforcement Learning*, y es el responsable del origen del término. Si bien no corresponde a la primer implementación de algoritmo de Aprendizaje Reforzado que utiliza redes neuronales complejas como aproximador funcional, si es el primer algoritmo que realiza de manera efectiva, y con buenos resultados, tanto la tarea de predicción como la de control.
En su núcleo, *Deep Q-Learning* corresponde simplemente a una versión de *Q-Learning* con aproximador funcional, la cual es entrenada mediante *Experience Replay* en vez de un aprendizaje en línea, utiliza el concepto de *Target Networks* para el cálculo de las diferencias temporales, y es entrenado mediante el algoritmo de optimización *RMSProp*. Un esquema completo de *Deep Q-Learning* puede observarse en el Algoritmo 6.

El principal reconocimiento que se realiza a *Deep Q-Learning* consiste en abordar mediante Aprendizaje Reforzado espacios de estados de dimensiones tan altas como el presente en una imagen. Adicionalmente, gracias a los resultados obtenidos en los videojuegos de la consola *Atari*, junto con la publicación realizada en la revista *Nature* [5], se ha popularizado el área, trayendo consigo importantes esfuerzos por parte de la comunidad científica en el desarrollo de esta metodología.

Algoritmo 6 *Deep Q-Learning* con *Experience Replay* y *Target Networks*

Entrada: N Número de episodios.
M Capacidad del *Replay Memory*.
K Tamaño del *mini-batch*.
C Frecuencia de actualización de la *Target Network*.
γ Factor de descuento.
π_{expl} Método de exploración (e.g $\epsilon - greedy$).
Q Aproximador funcional diferenciable (e.g red neuronal convolucional).

Salida: θ.

Inicializar de manera arbitraria de los parámetros del aproximador funcional θ y θ'
Inicializar *Replay Memory* D con capacidad M
para $n = 1$ hasta N **hacer**
  Inicializar s (condición inicial)
  mientras no *terminal* **hacer**
    $a \sim \pi_{epl}(Q_\theta(s, \cdot))$
    Ejecutar a. Observar r, s', *terminal*
    Guardar transición (s, a, r, s') en D
    Muestrear *mini-batch* de tamaño K (s_B, a_B, r_B, s'_B) desde D
    si terminal **entonces**
      $y_B \leftarrow r_B$
    si no
      $y_B \leftarrow r_B + \gamma \underset{a'_B}{max}\, Q_{\theta'}(s'_B, a'_B)$
    fin si
    Ejecutar un paso de gradiente descendente en la dirección del gradiente de $[y_B - Q_\theta(s_B, a_B)]^2$ con respecto a θ
    Cada C pasos $\theta' \leftarrow \theta$
    $s \leftarrow s'$
  fin mientras
fin para
devolver θ.

2.3.4. Deep Deterministic Policy Gradients (DDPG)

Deep Deterministic Policy Gradient (DDPG) [6] corresponde a un algoritmo que toma las ideas principales detrás de *Deep Q-Learning*, y las adapta a dominios continuos mediante el uso de *Deterministic Policy Gradients (DPG)* [51] como herramienta de aprendizaje.

A nivel general, *DDPG* consiste en un algoritmo Actor-Crítico de naturaleza *off-policy*. Utilizando la formulación de *Target Networks*, *DDPG* mantiene dos copias tanto para la estructura del actor, como para la del critico, las cuales se denominan de comportamiento o *behavioral* (π_B, Q_B) y objetivo o *target* (π_T, Q_T). Durante el entrenamiento, la labor de π_B es generar trayectorias en conjunto con alguna estrategia de exploración (en el trabajo original se utiliza ruido Gaussiano correlacionado temporalmente), mientras que la labor de Q_B es estimar de manera *off-policy* el rendimiento de la red objetivo. Adicionalmente π_T y Q_T se utilizan únicamente en la predicción de las diferencias temporales. De esta manera, Q_B aprende mediante el uso de diferencias temporales tradicional, mientras que π_B se modifica según la regla de *DPG*. Finalmente, las redes T se actualizan de manera suave con respecto a las redes B. Un esquema completo de *DDPG* se encuentra en el Algoritmo 7.

Los principales beneficios de *DDPG* son una manera efectiva de abordar problemas de acciones continuas mediante aproximaciones funcional no lineales de gran tamaño, una arquitectura de Actor-Crítico *off-policy* que le permite utilizar el concepto de *Experience Replay*, y el uso de *Deterministic Policy Gradients*, que ofrece una mayor eficiencia en cuando a datos, debido a que la estimación del gradiente solo debe integrar sobre el espacio de estados, obteniendo un estimador de menor varianza.

2.3.5. Delimitación en espacios de acciones continuos

Un problema importante de abordar al tratar con algoritmos de aprendizaje en espacios continuos, consiste en el manejo de los bordes del espacio de acciones. Normalmente los espacios de acciones son acotados, pues tienen significados físicos, como lo es un actuador en un sistema robótico. Una manera de lidiar con este fenómeno consiste en añadir una función del tipo *tanh* a la salida de la política, y un adecuado re-escalamiento, de tal manera que la política solo pueda generar valores válidos. Sin embargo, tal como se menciona en [52], la combinación de aproximadores no lineales de gran tamaño conlleva numerosos problemas. En particular, en algoritmos de Actor-Crítico, durante los primeros episodios, las estimaciones del crítico suelen tener valores aleatorios o con mucha incertidumbre, produciendo fácilmente saturación en la política del actor. Al considerar entonces las reglas de actualización de pesos del actor, se identifica rápidamente el fenómeno del gradiente desvaneciente, que dado que ocurre a la salida de la red, no permite una propagación adecuada del gradiente. Además, debido a que la política saturada condiciona los pares de estado-acción visitados, es común que el entrenamiento quede estancado en la política saturada. Si bien las técnicas de exploración tienen el potencial de arreglar estas situaciones, para problemas de alta dimensionalidad, es común que el algoritmo quede efectivamente estancado [52]. Otra opción habitualmente utilizada en la literatura con respecto al uso de *tanh* para restringir los valores de las acciones generadas [2] [37] consiste en utilizar directamente la salida lineal de la política, sin una función no lineal que acote la política generada. Es estos casos, para evitar generar acciones

Algoritmo 7 *Deep Deterministic Policy Gradients*

Entrada: N Número de episodios.
M Capacidad del *Replay Memory*.
K Tamaño del *mini-batch*.
γ Factor de descuento.
τ Factor actualización de las redes objetivo.
$\mathcal{N}$ Proceso estocástico (e.g ruido Gaussiano).
Q Aproximador funcional diferenciable para la función de valor del par estado-acción (e.g red neuronal convolucional).
π Aproximador funcional diferenciable para la política (e.g red neuronal convolucional).

Salida: θ.

Inicializar de manera arbitraria de los parámetros θ_{π_B}, θ_{Q_B}, θ_{π_T} y θ_{Q_T}
Inicializar *Replay Memory* D con capacidad M
para $n = 1$ hasta N **hacer**
  Inicializar s (condición inicial)
  mientras no *terminal* **hacer**
    $a \leftarrow \pi_{\theta_B}(s) + n$ $(n \sim \mathcal{N})$
    Ejecutar a. Observar r, s', *terminal*
    Guardar transición (s, a, r, s') en D
    Muestrear *mini-batch* de tamaño K (s_B, a_B, r_B, s'_B) desde D
    si terminal **entonces**
      $y_B \leftarrow r_B$
    si no
      $y_B \leftarrow r_B + \gamma Q_{\theta_{Q_T}}(s'_B, \pi_{\theta_T(s'_B)})$
    fin si
    Ejecutar un paso de gradiente descendente en la dirección del gradiente de $[y_B - Q_{\theta_B}(s_B, a_B)]^2$ con respecto a θ_{Q_B}
    Ejecutar un paso de gradiente descendente en la dirección $\nabla_{\theta_{\pi_B}} J \approx \frac{1}{K} \sum_{i=0}^{K-1} \nabla_a Q_{\theta_{Q_B}}(s, a)_{s=s_{B_i}, a=\pi_{\theta_{\pi_B}}(s_{B_i})} \nabla_{\theta_{\pi_B}} \pi_{\theta_{\pi_B}}(s)_{s=s_{B_i}}$ con respecto a θ_{π_B}
    $\theta_{\pi_T} \leftarrow \theta_{\pi_T}(1 - \tau) + \theta_{\pi_B}\tau$
    $\theta_{Q_T} \leftarrow \theta_{Q_T}(1 - \tau) + \theta_{Q_B}\tau$
    $s \leftarrow s'$
  fin mientras
fin para
devolver θ.

inválidas, se suele saturar las acciones de la política manualmente, lo cual sin embargo en aproximaciones no lineales puede producir que el crítico recomiende llevar las acciones fueras de los bordes permitidos, provocando incluso divergencia de los valores de la política.

Para solucionar el problema de la delimitaciones de acciones por parte de la política, en [52] se identifican 2 metodologías adicionales a las mencionadas anteriormente: *Zeroing Gradients* e *Inverting Gradients*. Ambas técnicas consisten en métodos sencillos que permiten abordar la problemática anterior, modificando los gradientes relacionados a las actualizaciones en la política, teniendo una implementación directa en algoritmos basados en *Deterministic Policy Gradients*, como lo es el caso de *Deep Deterministics Policy Gradients*, pues en dicho caso el entrenamiento está basado en $\nabla_a Q(s,a)$.

La primera de las técnicas desarrolladas en [52] corresponde a la de *Zeroing Gradients*. En esta implementación, la política no tiene ninguna delimitación en las acciones que puede generar, y las acciones son saturadas externamente al cálculo de la política. Sin embargo en el caso de que la política genere valores inválidos, se modifica el gradiente $\nabla_a Q(s,a)$ de tal manera que no pueda generar acciones con valores más extremos, solo permitiendo una propagación del gradiente hacia rangos válidos. La simpleza de este técnica queda expuesta en la ecuación 2.23, donde se puede identificar que la implementación de esta técnica puede ser realizada mediante una simple máscara binaria en el gradiente $\nabla_a Q(s,a)$.

$$\nabla_a Q(s,a) \leftarrow \nabla_a Q(s,a) \times \begin{cases} 0 & \text{si } a > a_{max} \wedge \nabla_a Q(s,a) > 0 \\ 0 & \text{si } a < a_{min} \wedge \nabla_a Q(s,a) < 0 \\ 1 & \text{si no} \end{cases} \tag{2.23}$$

La segunda de la técnicas mencionadas corresponde a *Inverting Gradients*. En ésta técnica se realiza una distorsión del gradiente, de tal manera que que permita al entrenamiento acercarse lentamente a los bordes del espacio de acciones, y que en caso de llegar a generar valores inválidos, se asegure que el gradiente lleve a la política a valores válidos. En los experimentos realizados en [52] se reportan mejores resultados con *Inverting Gradients* que con *Zeroing Gradients*. Adicionalmente, tal como se puede apreciar en la ecuación 2.24, la distorsión del gradiente además cumple un rol de regularización, pues favorece el flujo del gradiente en torno a acciones nulas. Finalmente, la implementación de la ecuación 2.24 también puede implementarse de manera sencilla como una máscara sobre el gradiente $\nabla_a Q(s,a)$.

$$\nabla_a Q(s,a) \leftarrow \nabla_a Q(s,a) \times \begin{cases} (a_{max} - a)/(a_{max} - a_{min}) & \text{si } \nabla_a Q(s,a) > 0 \\ (a - a_{min})/(a_{max} - a_{min}) & \text{si no} \end{cases} \tag{2.24}$$

2.3.6. Observabilidad Parcial y Redes Recurrentes

Los problemas con observabilidad parcial son los más recurrentes en robótica, siendo pocos lo casos donde la formulación escogida como estado realmente cumple la propiedad de *Markov*. Esto genera que en la práctica las predicciones y acciones de control sean realizadas

sobre un estado s, sino que sobre una observación o, y el proceso que rige el problema consiste en un proceso de *Markov* con observabilidad parcial (*POMDP*).

Al utilizar imágenes, como lo es el caso de [4] y [6], la observabilidad parcial se hace evidente, pues no se puede determinar la dinámica de un modelo físico o un videojuego a partir de una sola imagen. Para problemas sencillos, el enfoque seguido por estos estudios suele ser suficiente. Se utiliza el concepto de *action-repeats*, en donde la misma acción se ejecuta por K instantes de tiempo consecutivos, y posteriormente las K imágenes correspondientes a dichos instantes de tiempo son apiladas en un tensor tridimensional (alto, ancho, número de *actions repeats*). De esta manera se delega la extracción de la dinámica temporal del problema a redes convolucionales, que debido a que utilizan usualmente filtros tridimensionales, permiten extraer información temporal. Sin embargo al intentar aplicar esta metodología a problemas más complejos, que necesiten una alta frecuencia en la ejecución de acciones, o que se deba planificar en grandes horizontes de tiempo, esta metodología no es escalable computacionalmente.

Para abordar este tipo de problemas, en la literatura se han desarrollado modificaciones a algoritmos como *Deep Q-Learning* o *DDPG* para utilizar redes recurrentes. En [44] se plantea el uso de redes *LSTM* para abordar la observabilidad parcial, y demuestra de manera explícita y empírica, que la versión modificada de *Deep Q-Learning* con redes *LSTM* permite inferir velocidades, a partir de secuencias de imágenes correspondientes a un instante te tiempo. Posteriormente, en [16] se valida además la factibilidad de esta metodología para algoritmos continuos, donde además se evalúa el uso de redes recurrentes tradicionales además de las redes *LSTM*, donde se evidencia que el desempeño de las redes *LSTM* es superior.

Pese al potencial de las redes recurrentes para abordar problemas de observabilidad parcial, existen dificultades teóricas con respecto a su uso en algoritmos de *Deep Reinforcement Learning* debido a la manera en que se entrenan las redes recurrentes. La utilización de una estrategia de *backpropagation-through-time* permite entrenar episodios completos (en un esquema análogo a métodos de *Monte Carlo*), propagando de manera correcta el estado interno de la red, y permitiendo asignar los gradientes de forma adecuada. Sin embargo, el uso de episodios completos, perjudica la hipótesis del muestreo aleatorio, que además rompe la descorrelación temporal. Por supuesto, el entrenamiento de una red recurrente de una manera tradicional no permite que el entrenamiento propague consigo el estado interno de la red. Una propuesta intermedia, que ofrece una manera factible de entrenar redes recurrentes, sin perder de manera significativa la hipótesis del muestreo aleatorio, consiste en muestrear varias trazas o secuencias de un tamaño acotado dentro de la *Replay Memory* [11]. Para cada secuencia se reinicia el estado interno de la red, y se ejecuta la red adelante en el tiempo. Posteriormente, en el *backward pass* para actualizar los parámetros, solo se tiene en consideración los últimos elementos de la secuencia, pues solo en aquellos el estado interno de la red recurrente se asume como válido.

Capítulo 3

Diseño e Implementación

En este capítulo se desarrolla el trabajo realizado, con énfasis en los distintos aspectos de los procesos de diseño e implementación. Para poder dar orden a este capítulo, se divide en dos secciones: La Sección 3.1 se enfoca en el desarrollo de los algoritmos e implementaciones realizadas (herramientas de Aprendizaje Reforzado), mientras que la Sección 3.2 dedica su contenido al desarrollo del caso de estudio, identificación de problemas y su modelamiento.

La Sección 3.1 consiste en las herramientras de Aprendizaje Reforzado diseñadas e implementadas. En la Sección 3.1.1 se hace un breve resumen de las implementaciones y herramientas existentes en el contexto de Aprendizaje Reforzado, identificando sus usos, propósitos y limitaciones, justificando la necesidad de la implementación de una librería propia de Aprendizaje Reforzado. Posteriormente, la Sección 3.1.2 detalla las diferentes herramientas utilizadas, principalmente otras librerías de *software*, utilizadas como dependencias en la librería implementada, así como cual es el rol que cumplen dentro de los objetivos de la librería. Finalmente, el resto de la Sección 3.1 enumera y explicita las diferentes implementaciones realizadas con un énfasis en los componentes propios de la implementación, usualmente ignorados en las publicaciones científicas, pero que permiten obtener mejoras en los ámbitos de eficiencia de cáclulos computacionales, eficiencia de recursos computacionales, extensibilidad, e integración.

La segunda sección de este capítulo consiste en la descripción el caso de estudio, en este caso el fútbol robótico. Se detallan en esta sección aspectos como los objetivos, alcances y problemática general, con énfasis en las herramientas utilizadas (Sección 3.2.2). Posteriormente, en las Secciones 3.2.3, 3.2.4 y 3.2.5, se identifican algunos de los problemas que por su complejidad de modelamiento, presentan las cualidades de problemas que deben ser resueltos mediante técnicas como el Aprendizaje Reforzado. En estas secciones se hace énfasis en la dificultad de los problemas identificados, sus características recurrentes en otros problemas de la robótica, así como una descripción completa de su modelamiento, que incluye el diseño de espacios de estado, acción, recompensa, entre otros.

3.1. Librería de Aprendizaje Reforzado - UChileRL

En esta sección se presenta la librería de Aprendizaje Reforzado *UChileRL*[1], una librería de agentes de aprendizaje, especialmente reforzado, con una arquitectura que le permite abordar problemas de decisiones secuenciales de manera general. Los principales enfoques o características son proveer interfaces comunes a las distintas implementaciones, alto grado de eficiencia computacional, y énfasis en aplicaciones robóticas.

3.1.1. Motivación

En la actualidad, pese a que existen numerosas aplicaciones que utilizan Aprendizaje Reforzado, se presenta un bajo nivel de estandarización con respecto a las herramientas utilizadas en dichas implementaciones. El caso más común es encontrar implementaciones atómicas relacionadas al contenido de una publicación, con el contenido justo y necesario para poder ejecutar el método propuesto.

Algunas de las excepciones a esta regla general consisten en el *package* de *ROS* enfocado a Aprendizaje Reforzado [53] y *RLLib* [54]. El principal aporte del *package* de *ROS* consiste en una interfaz común y versátil para distintas implementaciones de Aprendizaje Reforzado, y un alto grado de integrabilidad al estar desarrollado en una plataforma popular de robótica [55]. Sin embargo, junto a la baja cantidad de algoritmos implementados, la arquitectura propuesta puede consistir en un cuello de botella, debido a la arquitectura interna de comunicaciones de *ROS*, la cual paga con rendimiento la sencillez del trabajo de sincronización. Por otro lado, *RLLib* [54] consiste en una librería diseñada y validada en problemas de robótica y dispositivos embebidos. Esta librería posee una alta eficiencia computacional, y una interfaz completa para varios métodos de Aprendizaje Reforzado. Sin embargo, la interfaz propuesta está sobre ajustada a métodos tradicionales, y no le permite escalar a algoritmos modernos como los planteados en *Deep Reinforcement Learning*, los cuales no implementa. Finalmente, es común encontrar implementaciones de algoritmos de *Deep Reinforcement Learning* escritas en lenguajes de programación de tipado dinámico, que facilitan el proceso de diseño como lo es el caso de *Python*. Pese a la facilidad en la implementación, estos casos suelen ser desarrollados con énfasis en aplicaciones científicas, y con bajo o nulo énfasis en su posterior despliegue en plataformas físicas (recursos computacionales, eficiencia de cómputos, etc).

Adicionalmente a las librerías de implementaciones de Aprendizaje Reforzado, se presenta en la comunidad científica un interés en la estandarización de problemas de toma de decisiones secuenciales. Tal es el caso de *Open AI Gym* [56], una librería escrita en *Python* que genera una interfaz común para problemas de decisiones secuenciales, e implementa distintos ejemplos de este tipo de problemas, tanto clásicos (*MountainCar*, *InvertedPendulum*, etc), como modernos (*Atari*, *MuJoCo* [57], *Gazebo* [58], etc). El objetivo entonces de *Open AI Gym* es generar una plataforma común para que distintas implementaciones puedan ser ser evaluadas de manera clara y reproducible.

Es en este contexto donde se identifica la necesidad de implementación de una nueva

[1] `https://github.com/knzo25/UChileRL`

librería de Aprendizaje Reforzado. El principal objetivo de esta librería es generar una interfaz que permite reutilizar herramientas y conceptos como los presentes en *RLLib*, pero que sea los suficientemente versátil como para integrar mecanismos no convencionales como *Tansfer Learning*, Aprendizaje Reforzado Descentralizado y *Deep Reinforcement Learning*. Adicionalmente esta librería debe estar optimizada en cuanto a recursos computacionales, por el dominio en el cual se desea aplicar.

3.1.2. Herramientas utilizadas

En vista de lo objetivos planteados, se otorga especial atención a las herramientas externas utilizadas, usualmente librerías de *Software*, y lenguajes de programación. En primer lugar, se elige en lenguaje de programación *C++* por su mayor potencial a nivel de integración. Si bien en el ámbito de la robótica, cada vez es más común utilizar lenguajes de programación como *Python* o incluso ambientes de desarrollo heterogéneos como *ROS*, dependiendo del contexto, dichas soluciones pueden no estar disponibles. Adicionalmente, *C++* permite un mayor control a nivel de datos y arquitectura del *hardware*, permitiendo realizar optimizaciones en el rendimiento, y abordar de manera directa los distintos cuellos de botella.

Para la implementación de algoritmos de manera eficiente se utiliza *Eigen* [59], que corresponde a una librería orientada al uso de algebra líneal y algoritmos relacionados. La elección de *Eigen* se basa en su reconocida eficiencia computacional, y activo desarrollo. Adicionalmente, debido a que *Eigen* corresponde a una librería compuesta únicamente de encabezados, permite una obtener una alta portabilidad, además de permitir al compilador realizar todas las optimizaciones disponibles en la arquitectura utilizada. En este trabajo se utiliza *Eigen 3.2*, debido a su capacidad de trabajar con tensores.

Los algoritmos tradicionales, tanto en su ejecución como entrenamiento, se componen principalmente de operaciones sencillas como multiplicación de matrices, por lo que su implementación mediante herramientas de algebra linea como *Eigen* resulta sencilla, incluso en casos de utilizar herramientas complejas como la aritmética de elementos *sparse*. Sin embargo, para implementaciones más complejas como las presentes en *Deep Reinforcement Learning*, las herramientas provistas por *Eigen* no son suficientes. Para el uso y entrenamiento de redes neuronales, y herramientas asociadas (e.g métodos de optimización), existen numerosas librerías que facilitan dicha labor, tales como *Tensorflow* [60], *Caffe* [61] y *Darknet* [62]. *Darknet* [62] consiste en una librería para redes neuronales escrita en *C*, lo que le otorga una gran ventaja en términos de portabilidad e integración. Sin embargo, posee una estructura rígida en cuanto a las arquitecturas de redes neuronales que implementa, lo que dificulta la implementación de algoritmos de *Deep Reinforcement Learning*. Adicionalmente, debido a que es desarrollada de manera independiente, no tiene mucho soporte, y no presenta un alto grado de optimización, en especial en los procesos de entrenamiento. Otra herramienta corresponde a *Caffe* [61], una herramienta enfocada en *Deep Learning*, desarrollada por académicos de *UC Berkeley*. Si bien es considerada como una de las implementaciones más rápidas en relación a cálculos de redes convolucionales, y posee una amplia gama de implementaciones, sus requisitos de uso a nivel de instalación dificultan fuertemente su uso en dispositivos embebidos. Finalmente, *Tensorflow* [60] consiste en una librería de grafos computacionales, aunque es mayoritariamente utilizada para el entrenamiento de redes neuronales. Al estar

desarrollada por *Google*, tiene un alto nivel de soporte, y es actualmente una de las herramientas favoritas de la comunidad científica en relación a redes neuronales. Adicionalmente, debido a su estructura basada en grafos, permite el diseño e implementación de las estructuras complejas presentes en *Deep Reinforcement Learning*, lo que ha generado que muchas de las implementaciones existentes de *Deep Reinforcement Learning* sean realizadas en esta plataforma.

En este trabajo se utiliza *Tensorflow* como herramienta para el uso de redes neuronales. El principal fundamento corresponde a la versatilidad de la herramienta, lo que permite una amplia gama de implementaciones y a la alta cantidad de algoritmos ya implementados, principalmente relacionados a redes neuronales y algoritmos de optimización. Sin embargo se debe tener en consideración que la mayor parte del desarrollo en *Tensorflow* se realiza en *Python*, el lenguaje predilecto en la comunidad de *Machine Learning*, pese a que *Tensorflow* está realmente implementado en *C++*. En este trabajo se utiliza un esquema mixto con respecto al uso de *Tensorflow*. Se utiliza la *API* de *Tensorflow* en *Python*, para el diseño e implementación de los grafos computacionales relacionados a los modelos utilizados en Aprendizaje Reforzado, pero se ejecutan dichos grafos utilizando la *API* de *Tensorflow* en *C++*, combinando de manera efectiva la flexibilidad en diseño de *Python*, y la eficiencia e integrabilidad de *C++*.

3.1.3. Arquiectura e Implementaciones

El diseño básico de *UChileRL* está basado en la formulación del Aprendizaje Reforzado presentada en la Figura 2.3. Este diseño básico de un proceso de Aprendizaje Reforzado puede observarse en la Figura 3.1.

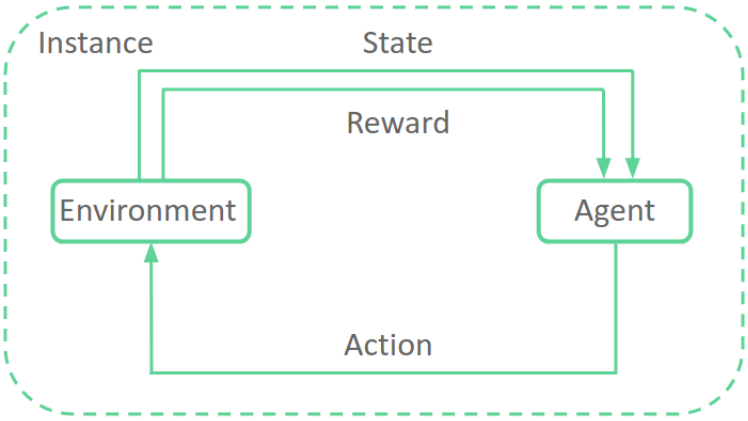


Figura 3.1: Esquema de los componentes presentes en *UChileRL* para un proceso básico de Aprendizaje Reforzado

A continuación se desarrollan los detalles específicos de los componentes de la Figura 3.1. En particular se destaca que los componentes instancia, ambiente y agente, corresponden a clases abstractas o virtuales de *C++*, permitiendo realizar la labor de interfaces para las

distintas implementaciones particulares.

- **Instancia:** Corresponde a la entidad encargada de manejar los distintos aspectos del proceso de aprendizaje, en particular las interacciones entre el agente de aprendizaje y el entorno. Se implementan distintos tipos de instancias, tales como entrenamiento, re-entrenamiento y prueba, en sus formulaciones episódicas y continuas. Adicionalmente, se considera también el caso de un proceso de aprendizaje con *Transfer Learning*, el cual puede encontrarse en la Figura 3.2.

- **Ambiente:** Corresponde al problema a resolver y su modelamiento (en términos de la Figura 2.3, implementa al intérprete). Su principal objetivo es realizar la formulación de variables de estado y proveer la señal de recompensa. Adicionalmente provee información acerca de los espacios de estado y acción (naturaleza, extremos, etc), provee la información acerca de estados terminales (condiciones de fin de episodio), y permite controlar la petición de acciones del proceso de aprendizaje, para poder modelar acciones sincronizadas en el tiempo (e.g *timesteps* fijos en el tiempo), o con algún otro evento externo, como el reinicio de las condiciones de un episodio.

- **Agente:** Corresponde a la implementación del algoritmo de aprendizaje. La interfaz básica del agente es minimalista, y solo establece la generación de acciones a partir de estados durante tiempo de prueba, y adicionalmente requiere de recompensa durante el entrenamiento.

- **Estado:** En *UChileRL* solo se modelan espacios de estados continuos, debido a la naturaleza de los problemas que busca resolver. Adicionalmente, se utiliza una formulación similar a la establecida en *Open AI Gym* [56] para la formulación de estados. En particular, en *UChileRL* se reconoce la formulación de estados como un vector, como un tensor, o como una combinación de las anteriores. Dicha formulación tiene la ventaja que permite abordar problemas en los que se utilizan datos de baja dimensionalidad, a la vez que datos de alta dimensionalidad, como lo es el caso de una imagen.

- **Acción:** La acción corresponde a una estructura sencilla. Establece la naturaleza de la acción (discreta o continua), su dimensionalidad y su valor.

- **Recompensa:** En Aprendizaje Reforzado la recompensa constituye esencialmente en un numero escalar. En *UChileRL* se considera a la recompensa como un vector, lo que permite modelar tanto Aprendizaje Reforzado tradicional (recompensa escalar), como Aprendizaje Reforzado Descentralizado, en donde la recompensa puede ser escalar, o tener una dimensionalidad igual a la del espacio de acciones.

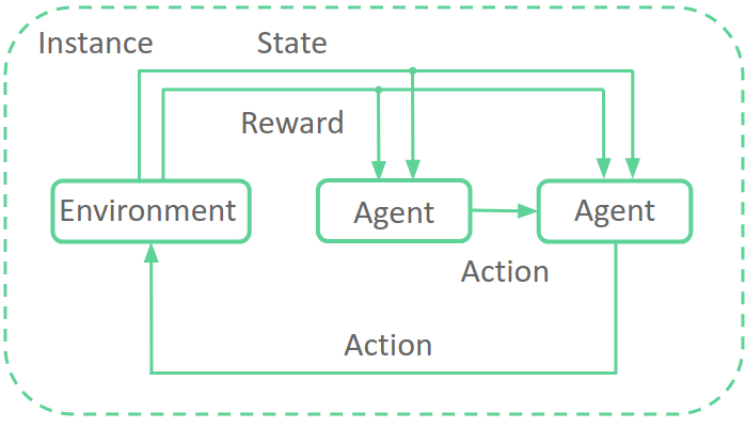


Figura 3.2: Esquema de los componentes presentes en *UChileRL* para un proceso de Aprendizaje Reforzado con *Transfer Learning*

En la Figura 3.3 se presentan los distintos agentes o algoritmos de aprendizaje implementados en *UChileRL*, los cuales corresponden principalmente a los algoritmos desarrollados en el Capítulo 2, en conjunto con *Double Q-Learning* y *Expected SARSA*, que corresponden a variantes de los algoritmos *Q-Learning* y *SARSA* respectivamente. En líneas punteadas se presentan interfaces para implementaciones posteriores, y en líneas sólidas se presentan implementaciones particulares. Esta formulación permite que los algoritmos puedan reutilizar su formulación común, implementando solo lo que los diferencia de otros algoritmos similares.

Los algoritmos bajo la categoría lineal corresponden a los distintos algoritmos presentados en la Sección 2.2, que están caracterizados por el uso de aproximación funcional lineal, y la minimización sencilla y en línea, del error cuadrático medio del error de diferencias temporales. Los algoritmos pertenecientes a la categoría lineal-discreto corresponden a algoritmos basados en la estimación de funciones de valor, en particular la función de valor del par estado-acción, y la manera en que estiman el error de diferencia temporal marca la diferencia entre las distintas implementaciones. Adicionalmente, todos los algoritmos de esta categoría pueden abordar problemas de múltiples dimensiones en el espacio de acciones mediante el uso del Aprendizaje Reforzado Descentralizado. Por otro lado, el algoritmo Actor-Crítico lineal es el único algoritmo lineal implementado que permite el uso de acciones continuas y potencialmente multi-dimensionales, sobre el cual también se puede utilizar la formulación del Aprendizaje Reforzado Descentralizado de manera opcional.

Adicionalmente a los algoritmos lineales clásicos, se implementa *Deep Q-Learning* y *Deep Deterministic Policy Gradient (DDPG)*. Estos algoritmos, debido a su complejidad tanto en su arquitectura, como en su entrenamiento, son implementados mediante *Tensorflow*. A nivel de implementación, las principales características de estos algoritmos, se centran en el entrenamiento por *mini-batches*, y el *Experience Replay*.

En robótica, usualmente el algoritmo de aprendizaje debe coexistir con numerosos otros

sistemas (e.g localización, actuación, entre otros). Si bien los algoritmos de *Deep Renforcement Learning* son complejos, sus tiempos de ejecución son usualmente del orden de un par de milisegundos. Sin embargo, el proceso de un *mini-batch* suele durar decenas de milisegundos, lo que ocasiona un retraso en el módulo completo, afectando a los demás sistemas que dependan de la acción realizada por el agente. Sin embargo, la ejecución del *minibatch* no tiene por que ser realizada de manera secuencial tras la selección de acción o ejecución de la política. En *UChileRL* se reconoce esta situación, y se permite, de manera opcional, realizar el proceso de entrenamiento por *minibatches* en un hilo de ejecución independiente. Este proceso permite abordar eficientemente uno de los cuellos de botella presentes en la implementación de *Deep Reinforcement Learning* en aplicaciones robóticas, donde el tiempo de ejecución afecta de manera directa el desempeño del sistema. Sin embargo, no se debe confundir esta solución con las presentes en [63], debido a que el paralelismo solo ocurre el momento de ejecutar un *minibatch*. Es decir, la siguiente ejecución de la política debe esperar en el caso que en hilo que ejecuta el *minibach* anterior aún no termina su propósito.

El segundo aspecto de implementación relevante consiste en el *Experience Replay*. En [4], [6] y trabajos posteriores, se utiliza *Experience Replay* con una capacidad de alrededor de un millón de experiencias. Para problemas de baja dimensionalidad (en este contexto menor a 1000), es factible representar el *Experience Replay* en memoria *RAM*. Sin embargo, para problemas de alta dimensionalidad, principalmente aquellos que utilizan imágenes, el tamaño del *Experience Replay* no permite utilizar directamente memoria *RAM* en la mayoría de los dispositivos actuales. Por otro lado utilizar directamente almacenamiento en disco conlleva latencias que repercuten considerablemente en el tiempo de ejecución total. En *UChileRL* se aborda este problema mediante un esquema mixto entre almacenamiento en memoria y uso de disco duro. A lo largo del entrenamiento, las experiencias son almacenadas en un disco duro, pero el algoritmo de entrenamiento, al momento de requerir un *minibatch*, lo obtiene directamente de memoria *RAM* sin tener que sufrir de la latencia del acceso a disco. Este fenómeno se implementa mediante conceptos de las ciencias de la computación, en particular de computación gráfica. Se utiliza un hilo de ejecución independiente que utiliza una estrategia de triple *buffer* para mantener en memoria siempre 3 *minibatches* disponibles para el hilo del algoritmo de aprendizaje. Esta formulación permite, bajo ciertos supuestos, que el algoritmo de entrenamiento nunca deba esperar a que esté disponible el siguiente *minibatch*.

Adicionalmente a los aspectos especiales mencionados con respecto a la implementación, se aplican las técnicas desarrolladas en [52] y [11], desarrolladas además en las Secciones 2.3.5 y 2.3.6, correspondiendo a técnicas para evitar la saturación de las redes neuronales involucradas, y poder abordar problemas de observabilidad parcial respectivamente.

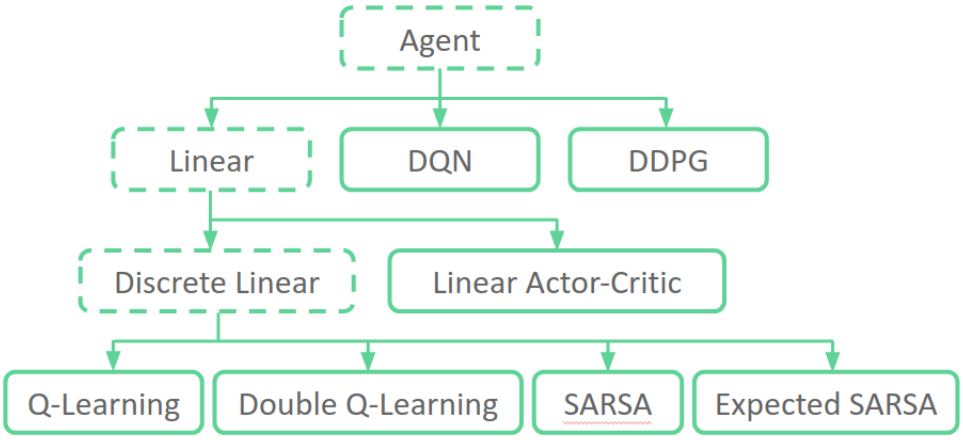


Figura 3.3: Agentes de Aprendizaje Reforzado implementados en *UChileRL*

En la Figura 3.4 se presentan los distintos ambientes implementados en *UChileRL*. Los ambientes *Dribbling*, *In-walk Kicking* y *Navigation* corresponden a los problemas estudiados en este trabajo. Adicionalmente, se integran en *UChileRL* los ambientes disponibles en *OpenAI Gym*, lo que permite evaluar los algoritmo implementados en *UChileRL* con una amplia gama de problemas, tanto clásicos como *MountainCar Inverted Pendulum*, problemas de simuladores físicos como *Box2D* y *MuJoco*, y problemas de alta dimensionalidad, como los juegos de la consola *Atari*.

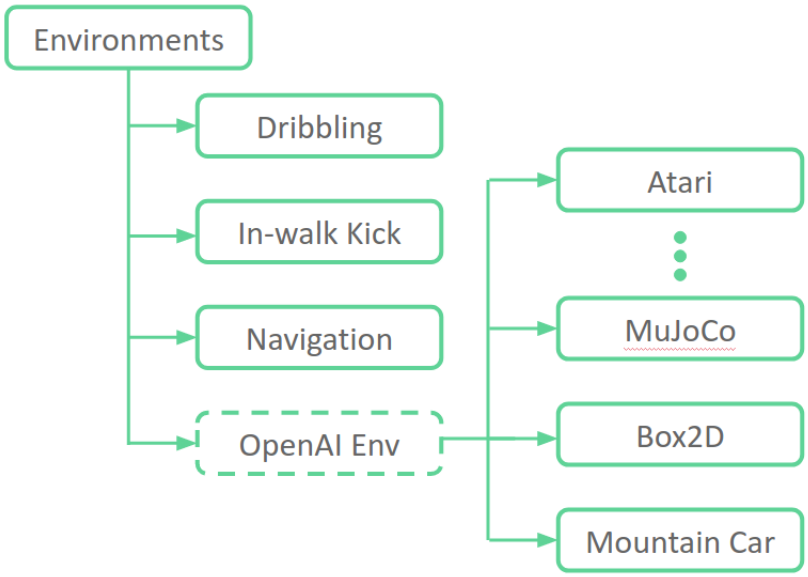


Figura 3.4: Ambientes de Aprendizaje Reforzado implementados en *UChileRL*

El último aspecto a desarrollar con respecto a la implementación de *UChileRL* consiste en los métodos de exploración, que son componentes claves en la totalidad de los algoritmos de Aprendizaje Reforzado, dentro de la temática de explotación vs. exploración.

En la Figura 3.5 se presenta un resumen de los métodos implementados. Las técnicas de exploración discretas realizan la exploración mediante la selección de acciones a partir

de los valores estimados de la función de valor del par estado-acción (valor Q). El principal requerimiento, es que permitan visitar con una probabilidad no nula las distintas acciones, y que degeneren en una elección avara (*greedy*). En particular, en *UChileRL* se implementas los mecanismos de exploración de acciones discretas más utilizados en la literatura, correspondientes a *greedy*, $\epsilon - greedy$ y *softmax*. Adicionalmente, se modulan los parámetros de exploración a lo largo del entrenamiento, para permitir realizar una fuerte exploración en episodios iniciales, y posteriormente favorecer la explotación del conocimiento adquirido, mediante un decremento en la exploración. Dicha modulación se realiza variando de manera lineal o exponencial los parámetros ϵ de $\epsilon - greedy$ y T de *softmax*.

Por otro lado, para acciones continuas, se implementa un sistema de exploración mediante ruido Gaussiano aditivo, y una variante que utiliza ruido correlacionado en el tiempo, lo que le otorga momento a la exploración (proceso de *Ornstein–Uhlenbeck*). El efecto de la exploración en algoritmos de Actor-Crítico, permite adicionalmente a mejorar las estimaciones del crítico, obtener mejores estimaciones del gradiente con el cual aprende el actor. La ventaja del ruido Gaussiano aditivo corresponde a una mayor rapidez en el proceso de aprendizaje, pues se realizan pequeñas perturbaciones en las acciones. Sin embargo el utilizar un proceso de *Ornstein–Uhlenbeck* como ruido aditivo permite desestancar el proceso de aprendizaje en situaciones en los que se obtiene una convergencia sub-óptima temprana, pues permite generar acciones con comportamientos más complejos, debido al momento en la exploración.

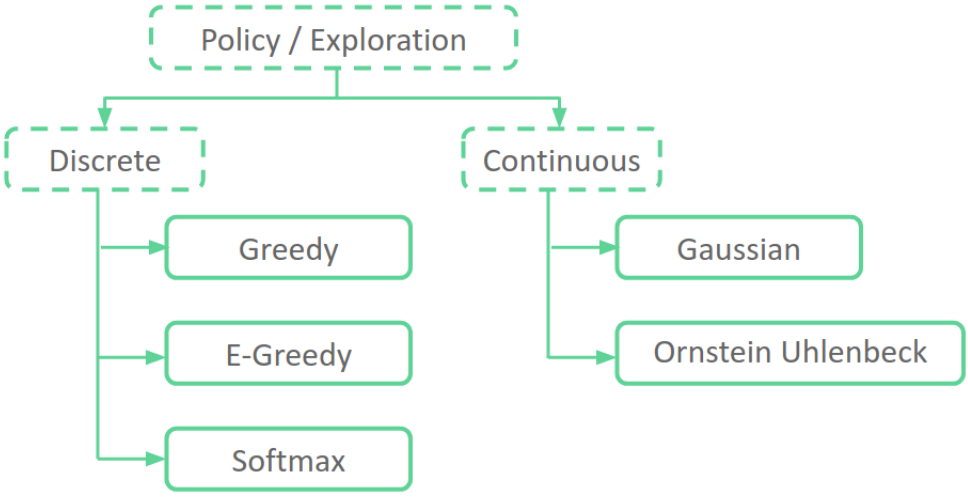
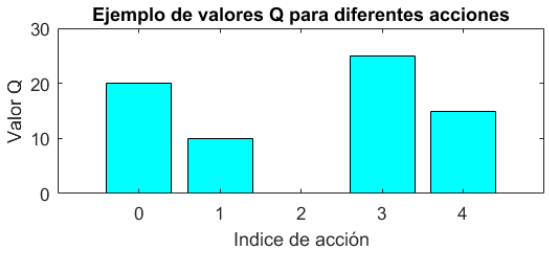
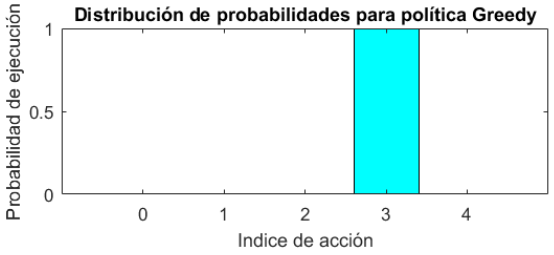


Figura 3.5: Métodos de exploración implementados en *UChileRL*

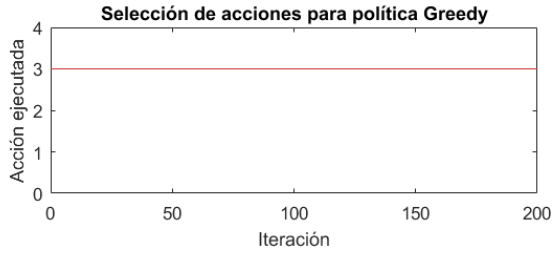
Para comprender la naturaleza de los métodos de exploración implementados, en las Figuras 3.6 y 3.7 se presentan ejemplos de los métodos anteriormente mencionados, para los casos de acciones discretas y continuas respectivamente.



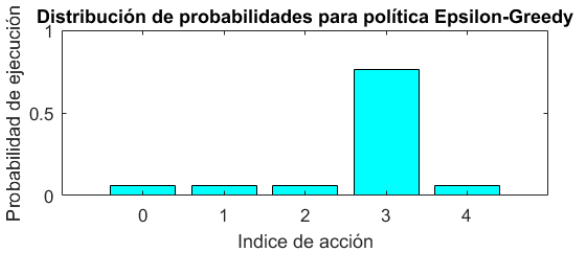
(a) Ejemplo de valores Q para 5 acciones



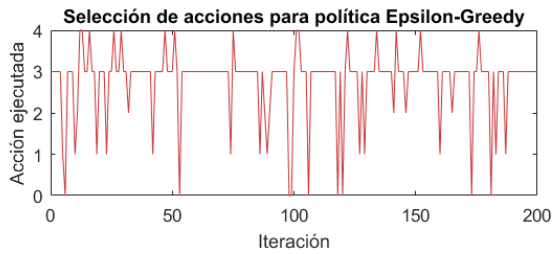
(b) Probabilidad de selección de acciones para política *greedy*



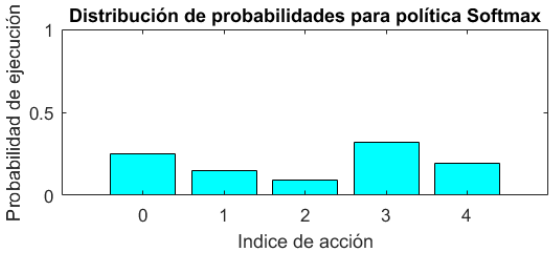
(c) Secuencia temporal de acciones para política *greedy*



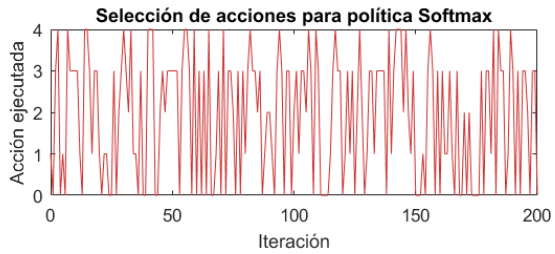
(d) Probabilidad de selección de acciones para política ϵ-*greedy*



(e) Secuencia temporal de acciones para política ϵ-*greedy*

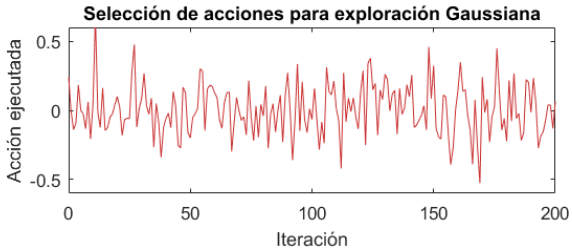


(f) Probabilidad de selección de acciones para política *Softmax*

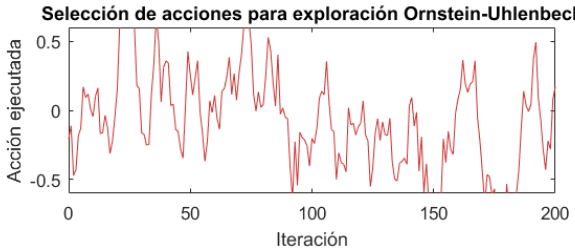


(g) Secuencia temporal de acciones para política *Softmax*

Figura 3.6: Ejemplos de probabilidad de selección de acciones y secuencias temporales de acciones para distintos mecanismos de exploración. Los parámetros asociados son $\epsilon = 0.3$ y $T = 20$



(a) Ruido aditivo Gaussiano



(b) Ruido aditivo de proceso de *Ornstein–Uhlenbeck*

Figura 3.7: Ejemplos de mecanismo de exploración para acciones continuas. Los parámetros asociados son $\sigma = 0.2$ y $\theta = 0.15$

3.2. Caso de estudio

En esta sección se introduce el caso de estudio, correspondiente al fútbol robótico. En la Sección 3.2.1 se presenta el contexto en el cual está situado, sus características, alcances y objetivos. Posteriormente, en la Sección 3.2.2 se detallan las herramientas relacionadas al caso de estudio con las que se realiza este trabajo. Finalmente, las Secciones 3.2.3, 3.2.4 y 3.2.5 desarrollan problemas particulares dentro del fútbol robótico que son modelados dentro de la formulación del Aprendizaje Reforzado.

3.2.1. Introducción y Objetivo

La *RoboCup (Robot Soccer World Cup)* [1] es una competencia internacional anual, que busca promover la investigación, desarrollo y divulgación de la robótica mediante un desafio atractivo y a la vez desafiante. El objetivo planteado por la *RoboCup* es codicioso y es un proyecto a largo plazo. Pese a no tener repercusiones económicas inmediatas, el cumplimiento de su meta posee profundos impactos tanto sociales como a nivel académico. El objetivo planteado es el siguiente:

> "Para mediados del siglo 21, un equipo completo de robots humanoides ganará un encuentro contra los vencedores del más reciente mundial, cumpliendo con las reglas oficiales de la *FIFA*"

El objetivo planteado es en sí, un desafío monumental que abarca múltiples disciplinas. Una vista global permite identificar este objetivo como un hito tanto a nivel social como científico, mientras que uno más practico lo permite ver como un problema común o estándar, que permite la evaluación de distintas metodologías, en pos de un objetivo compartido.

Si bien desde un punto de vista técnico los desafíos presentes en la *RoboCup* son numerosos, a continuación, en la Tabla 3.1 se presenta un cuadro comparativo entre las diferencias entre los desafios de problemas tradicionales en el campo de la inteligencia artificial como es el caso de Ajedrez, y aquellos presentes en el dominio de la *RoboCup*[2].

Tabla 3.1: Comparación entre las características del Ajedrez y la *RoboCup*

	Ajedrez	RoboCup
Ambiente	Estático	Dinámico
Dinámica de acciones	Turnos	Tiempo Real
Observabilidad	Completa	Parcial
Tipo de información	Simbólica	Abstracta
Control	Central	Distribuido

Finalmente, las características del dominio introducido por la *RoboCup*, por su complejidad e interdisciplinariedad, la convierten en el perfecto caso de estudio para diferentes áreas

[2] `http://www.robocup.org/objective/`

como fusión sensorial en tiempo real, toma de decisiones secuenciales, aprendizaje, planificación en tiempo real, sistemas multi-agente, visión computacional, locomoción, entre otros.

3.2.2. Herramientas Utilizadas

Dentro de las diferentes competencias que organiza la *RoboCup*, se destaca la *Standard Platform League (SPL)*, debido a que en dicha competencia, todos los equipos participantes deben utilizar la misma plataforma común, la cual actualmente es el robot *NAO* [64].

Debido a que todos los equipos deben utilizar la misma plataforma, el desarrollo en esta liga está enfocado en sistemas genéricos, sin poder utilizar *hardware* espécifico (*ASIC*, sensores extra, procesamiento externo, etc), y de manera natural se ha producido una filosofía de tipo *open source*, en la cual muchos de los equipos de la liga publican de manera *online* sus implementaciones, algoritmos y estrategias, con el objetivo de mejorar el desarrollo a nivel general de la liga, además de permitir la reproducibilidad de su investigación. En particular, los principales promotores de esta filosofía son los equipos *B-Human*[3] de la Universidad de Bremen, *Nao-Devils*[4] de la Universidad Técnica de *Dortmund*, *rUNSWift* de la Universidad de *New South Wales (UNSW Sydney)* y *HTWK*[5] de la Universidad de Ciencias Aplicadas de *Leipzig*, que publican gran parte de desarrollo, lo cual es de suma importancia para la liga, dado que son los principales exponentes de la misma.

En este trabajo se utiliza la base de *software* provista por *B-Human* [65], que ha sido fundamental en el desarrollo de la liga [66]. Este *software* consiste en un conjunto de herramientas que contiene una implementación completa para obtener altos desempeños en la *RoboCup*. En particular este *software* contiene todos los sistemas necesarios para el funcionamiento del robot (visión, localización, actuación, etc), provee un simulador robótico, herramientas de depuración, entre otras herramientas útiles para el desarrollo e investigación. Debido a la naturaleza de este trabajo, es importante mencionar las distintas características que posee el simulador robótico incorporado. El simulador utilizado por *B-Human* consiste en *SimRobot* [67], un simulador inicialmente desarrollado para la liga predecesora de la *SPL*, la *Four Legued League*. Este simulador, desarrollado inicialmente por el *German Team*, ha sido desde entonces desarrollado y mantenido por *B-Human* para su uso en la *SPL* (un ejemplo del ambiente provisto actualmente por este simulador puede observarse en la Figura 3.8). La simulación física es realizada mediante *ODE*[6], un motor de física tridimensional de sólidos rígidos. Adicionalmente, *SimRobot* modela los distintos sensores del robot *NAO*, que incluyen las cámaras del robot, las cuales obtienen su información directamente a partir del simulador, mediante un *viewport* de *OpenGL*[7]. A un alto nivel, *SimRobot* es suficientemente fidedigno con el robot real, permitiendo realizar pruebas de concepto en las áreas de toma de decisiones, modelamiento y localización. Sin embargo, debido a los limitados modelos físicos, y a una renderización simple del ambiente simulado mediante *OpenGL*, los sistemas de actuación y percepción son los que presentan una mayor discordancia con la realidad (*reality gap*), por

[3] `https://www.b-human.de`
[4] `http://naodevils.de/`
[5] `http://robocup.imn.htwk-leipzig.de/`
[6] `http://www.ode.org/`
[7] `https://www.opengl.org/`

lo que usualmente son desarrollados de manera más cercana al robot físico.

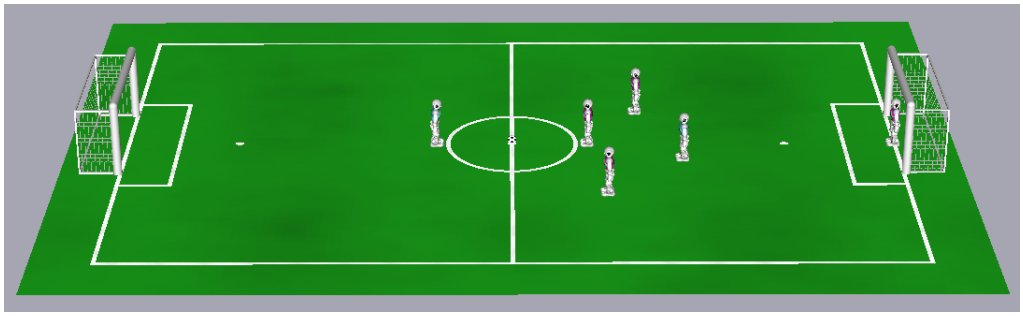


Figura 3.8: Toma de pantalla del simulador *SimRobot*

3.2.3. Ball Dribbling

En el contexto del fútbol robótico, *Ball Dribbling* corresponde a la habilidad o comportamiento en el cual un robot debe llevar una pelota de fútbol de manera controlada hacia un objetivo dado [24]. La importancia de esta habilidad en el contexto del fútbol robótico es clara, pues permite generar estrategias de juego complejas, mantener el dominio de la pelota y últimamente aumentar la capacidad de marcar goles. Sin embargo, debido a la naturaleza del objetivo del problema, que es llevar la pelota lo más rapido posible hacia el objetivo, a la vez de realizarlo de una manera controlada (pelota lo más cerca posible de los pies del robot), el problema resulta complejo de modelar, y difícil de resolver con buenos resultados utilizando estrategias de control tradicionales, debido a la falta de una señal de error tradicional a minimizar, y a la falta de una referencia.

Adicionalmente, son muchos los factores que dificultan el modelamiento de este problema, debido a la naturaleza del propio contexto. La información de la posición de la pelota con respecto al robot nunca es conocida con certeza, y debe ser resuelta mediante algoritmos de visión computacional, y potencialmente, utilizar un filtro para realizar una estimación robusta, teniendo además la posibilidad de obtener estimaciones en casos de falla en la detección. Por otro lado, los modelos cinemáticos que permiten proyectar objetos en la imagen de la cámara a coordenadas cartesianas con respecto a un punto fijo del robot introducen una componente de incertidumbre adicional. Esto se debe a que la estimación de la cadena cinemática se calcula con respecto a un punto fijo en el suelo (pie de soporte). Sin embargo, debido a la naturaleza bípeda del robot, el pie de soporte se alterna a lo largo de de un movimiento de caminata, lo que introduce errores en la estimación. Finalmente, debido a que usualmente el objetivo a donde se desea llevar la pelota corresponde a una información de alto nivel, esta suele estar formulada en coordenadas globales (cancha) y para poder ser utilizada se debe hacer uso de la estimación de la localización del robot, lo que añade un nivel más en la incertidumbre.

Debido a la dificultad de modelamiento de este problema, estrategias tradicionales no suelen ser atractivas como herramientas de resolución, y en la literatura se ha resuelto este problema mediante estrategias de Aprendizaje de Máquinas tales como Aprendizaje Refor-

zado [24] [25] y Aprendizaje Interactivo [68]. En [24] se formula por primera vez y de manera formal el problema de *Ball Dribbling*, y se utiliza una estrategia conjunta de Aprendizaje Reforzado y Controladores Difusos, para disminuir la complejidad del problema. Posteriormente en [25] se resuelve el problema en su totalidad mediante el uso de Aprendizaje Reforzado, y en [68] se resuelve este problema con excelentes resultados mediante el uso de Aprendizaje Interactivo.

En este trabajo se sigue la formulación realizada en [25], y se compara la metodología propuesta en dicho trabajo, con las herramientas introducidas en el Capítulo 2.

3.2.3.1. Modelamiento

Durante el diseño de problemas mediante Aprendizaje Reforzado es de suma importancia tener en mente las distintas limitaciones clásicas con respecto a la dimensionalidad del modelo. Debido a esto, se pone especial esfuerzo en un diseño minimalista, que permita expresar la mayor cantidad de aspectos del problema de tal manera que cumpla la propiedad de *Markov*, manteniendo la dimensionalidad del modelo lo más reducida posible.

En el caso de *Ball Dribbling*, existen numerosas alternativas para su modelamiento. En particular, una opción ingenua consiste en el uso de los distintos sensores del robot (acelerómetro, sensores de presión, cámara, etc) como variables de estado, y los grados de libertad actuados como espacio de acciones. Si bien dicho modelamiento permite describir el problema en un alto grado, su alta dimensionalidad no le permite ser representado mediante técnicas tradicionales, y su proceso de aprendizaje resulta peligroso, debido a que una exploración en dicho espacio de acciones, produce acciones inestables, y durante la mayor parte del proceso de aprendizaje, el robot deberá aprender a caminar, previo a realizar realmente la labor de *Ball Dribbling*. Una última dificultad de este modelamiento consiste en la nula robustez en la transferencia de la política entre el simulador y el robot físico, debido al *reality gap* producto de la simulación de los actuadores y los sensores.

En este trabajo, se utiliza la propuesta originada en [24], en la cual se integra al Agente de Aprendizaje en el *framework* de *B-Human* [65], tal como se presenta en la Figura 3.9. El *reality gap* producto de el modelamiento de los sensores se amortigua mediante el uso del sistema de visión y localización de *B-Human*. Adicionalmente, se delega la generación directa de valores de los actuadores al sistema de locomoción desarrollado en [69], de tal manera disminuir el *reality gap* producto de la simulación de los actuadores, debido a que el sistema de locomoción se efectúa en lazo cerrado, presentando rendimientos similares entre ambientes simulados y reales.

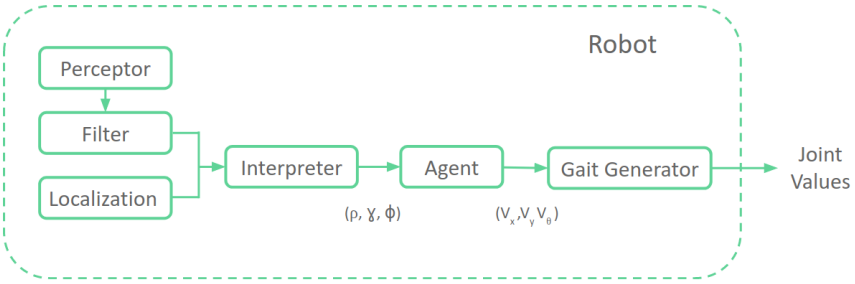


Figura 3.9: Diagrama de bloques del diseño propuesto para *Ball Dribbling*

El modelo propuesto para el espacio de estados consiste en una formulación geométrica del problema mediante el uso de 3 variables de estado: la distancia del robot a la pelota (ρ), el ángulo entre la orientación del robot y la pelota (γ), y el complemento del ángulo que forman el robot, la pelota y el objetivo (ϕ). Adicionalmente, las variables de estado no son formuladas desde el centro u origen del robot, sino que son calculadas desde uno de los dos pies, el cual es determinado mediante un sistema de selección basado en el pié más cercano a la pelota, el cual incorpora histéresis para evitar comportamientos oscilatorios. Un ejemplo de la formulación geométrica planteada puede observarse en la Figura 3.10.

El sistema de locomoción omnidireccional desarrollado en [69] permite controlar desde un alto nivel el comportamiento del sistema de locomoción. Esto permite que el agente de aprendizaje genere acciones que posteriormente son utilizadas como referencias de la caminata omnidireccional, que en este caso corresponden a comandos de velocidad (V_x, V_y, V_θ).

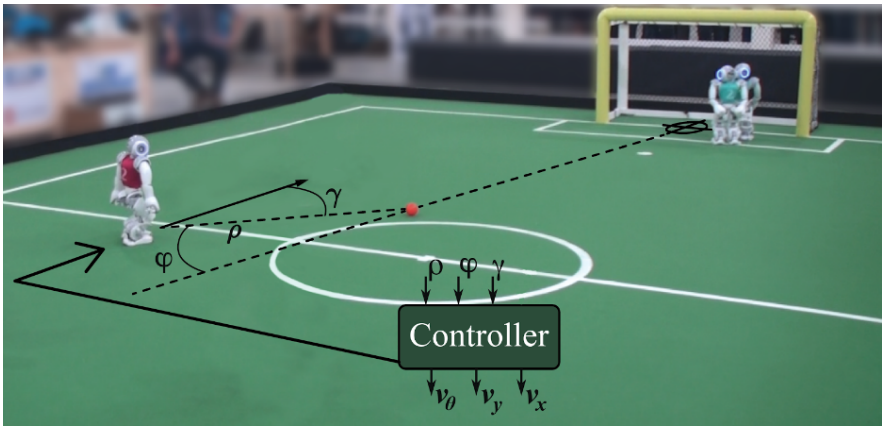


Figura 3.10: Ejemplo de la formulación para *Ball Dribbling*, en una sutuación real de juego

La formulación del problema realizada tiene numerosas consecuencias. La utilización de módulos externos al aprendizaje, permite facilitar la labor de transferencia de un simulador al robot *físico* suponiendo que dichos módulos se desempeñan de manera similar entre el simulador y el ambiente real. Por otro lado, la reducción de dimensionalidad en el espacio de estados mediante una formulación geométrica permite un modelamiento sencillo y de baja dimensionalidad, que permite lograr un aprendizaje en una cantidad baja de episodios. Sin embargo, la formulación realizada no considera en ningún caso obstáculos o robots del

equipo contrario, por lo que el rendimiento del sistema queda limitado en situaciones reales. Además, debido a la formulación de estados no incluye la posición del robot en la cancha, es posible que en algunos casos, el robot lleve la pelota fuera del campo de juego. Finalmente, el uso del sistema de locomoción presentado en [69] permite reducir considerablemente la dimensionalidad del espacio de acciones, y dar seguridad al proceso de aprendizaje, debido a que el sistema de locomoción se encarga de la estabilidad del robot. No obstante lo anterior, el uso del sistema de locomoción omnidireccional tal como se plantea en la Figura 3.9 trata a dicho sistema como una caja cerrada, y no modela ni considera ninguno de sus aspectos prácticos. En particular, no existe ninguna sincronización entre el proceso de Aprendizaje y la generación de movimientos. Una limitación final con respecto al uso del sistema de locomoción, es que limita los movimientos generados, lo cual si bien es teóricamente una limitación, en la práctica no llega tener una marcada relevancia, debido al rendimiento del sistema de locomoción.

3.2.3.2. Recompensa y Rendimiento

En Aprendizaje Reforzado, la recompensa define el tipo de comportamiento resultante del entrenamiento. Para poder evaluar de manera cuantitativa el producto del entrenamiento, es común utilizar los criterios de recompensa acumulada o promedio (durante un episodio). Sin embargo, dicha formulación no permite evaluar de manera comparativa diferentes formulaciones de recompensa, y además no permite en muchos casos evaluar el rendimiento real que se quiere lograr, debido a que la recompensa no siempre tiene un significado directo en términos de rendimiento. Para poder compararar el diseño de distintas recompensas, además de poder dar un significado claro a las políticas resultantes, en las ecuaciones 3.1 y 3.2 se presentan dos índices de rendimiento que permiten caracterizar el resultado del entrenamiento de manera clara.

$$V_{avg} = \frac{1}{TVx_{max}} \sum_{t=1}^{T} V_x(i) cos(\theta(i)) \tag{3.1}$$

$$faultRate = \frac{1}{T} \sum_{t=1}^{T} fault(i) \tag{3.2}$$

$$fault(i) = \begin{cases} 1 & \text{si } |\phi_i| > \phi_{fault} \vee |\gamma_i| > \gamma_{fault} \vee |\rho_i| > \rho_{fault} \\ 0 & \text{si no} \end{cases} \tag{3.3}$$

En la ecuación 3.1, θ representa la orientación del objetivo con respecto al robot, de al manera que V_{avg} representa en cada instante de tiempo la proyección de la velocidad frontal, en la dirección hacia el objetivo, teniendo un valor cercano a uno, mientras más rápido se acerque el robot hacia el objetivo. Por otro lado, para poder evaluar el control del robot sobre

la pelota, se introduce el índice de rendimiento de la ecuación 3.2, en donde el término $fault$ queda descrito en la ecuación 3.3, en donde se caracteriza el control de la pelota mediante simples umbrales en las variables de estado. En este caso, un índice de $faultRate$ cercano a 0 representa un control absoluto sobre la pelota, sin tener en consideración la velocidad a la cual se realiza el control.

En este trabajo, se utilizan dos formulaciones para la recompensa del problema de *Ball Dribbling*. El primer modelamiento corresponde al realizado en [25], en donde se utiliza el formulamiento del Aprendizaje Reforzado Descentralizado para aprender los distintos componentes del vector de velocidades (V_x, V_y, V_θ), y las recompensas utilizadas pueden observarse en las ecuaciones 3.4, 3.5 y 3.6. El uso de Aprendizaje Reforzado Descentralizado permite un modelamiento sencillo de la recompensa, pues permite identificar la labor de cada agente, para poder resolver el problema en su totalidad. En las ecuaciones 3.4, 3.5 y 3.6, ρ_{max}, ϕ_{max} y γ_{max} corresponden a los valores extremos del espacio de estados, mientras que Vx_{max} corresponde al máximo valor del espacio de acciones del agente relacionado a V_x, el cual se elige igual a la velocidad máxima del sistema de locomoción.

$$r_x = \begin{cases} -\frac{\rho}{\rho_{max}} - \frac{|\gamma|}{\gamma_{max}} - \frac{|\phi|}{\phi_{max}} - (1 - \frac{V_x}{Vx_{max}}) & \text{si } fault \vee V_x < Vx_{max} \\ 1 & \text{si no} \end{cases} \tag{3.4}$$

$$r_y = \begin{cases} -1 - \frac{|\phi|}{\phi_{max}} & \text{si } |\phi| > \phi_{fault} \\ 2 - \frac{|\phi|}{\phi_{max}} & \text{si no} \end{cases} \tag{3.5}$$

$$r_\theta = \begin{cases} -1 - \frac{|\phi|}{\phi_{max}} - \frac{|\gamma|}{\gamma_{max}} & \text{si } |\phi| > \phi_{fault} \vee |\gamma| > \gamma_{fault} \\ 2 - \frac{|\phi|}{\phi_{max}} - \frac{|\gamma|}{\gamma_{max}} & \text{si no} \end{cases} \tag{3.6}$$

El segundo modelo de recompensa corresponde a la recompensa escalar tradicional presentado en la ecuación 3.7. Esta formulación de recompensa utiliza únicamente recompensas negativas, por lo que el agente busca terminar el episodio recibiendo la menor cantidad de castigo posible. Se penaliza fuertemente el hecho de salir de la cancha, debido a que una recompensa únicamente negativa favorece las soluciones en las que el robot salga de la cancha lo más rápido posible. Adicionalmente, los primeros términos de la ecuación 3.7 representan el tiempo en el que una caminata diferencial demora en alinearse con la pelota, y el resto de los términos representan el compromiso entre estar cerca de la pelota, y acercarse al objetivo rápidamente.

$$r = \begin{cases} -100 & \text{si } ball\ or\ robot\ out\ of\ field \\ -\frac{\rho}{\rho_{max}} - \frac{|\gamma|+|\phi|}{V_{\theta_{max}}} - 5(1 - \frac{V_x}{V_{x_{max}}})(2 - cos(\phi))(2 - cos(\gamma)) & \text{si } fault \\ -5(1 - \frac{V_x}{V_{x_{max}}}) & \text{si no} \end{cases} \tag{3.7}$$

3.2.3.3. Condiciones de entrenamiento

Al igual que el diseño de la recompensa, las condiciones de entrenamiento (iniciales, de término, etc), influyen directamente en el comportamiento que surge del uso de Aprendizaje Reforzado. En la Figura 3.11 se presenta un ejemplo del ambiente bajo el cual se realiza el proceso de entrenamiento. La pelota siempre parte en la misma posición, y el objetivo es llevarla dentro de la circunferencia marcada en rojo. No se utiliza ningún obstáculo, debido a que la formulación de estados no los incluye.

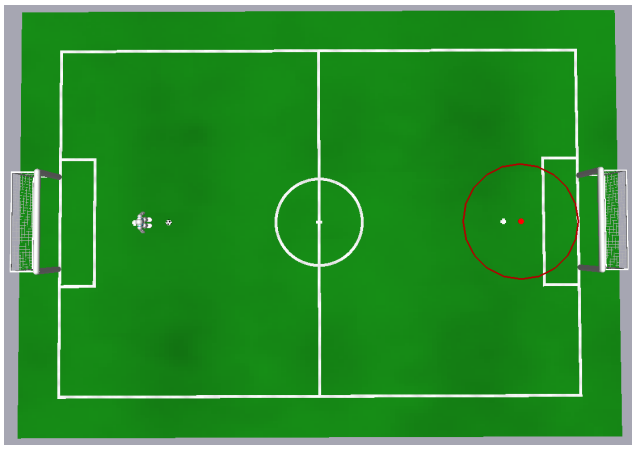
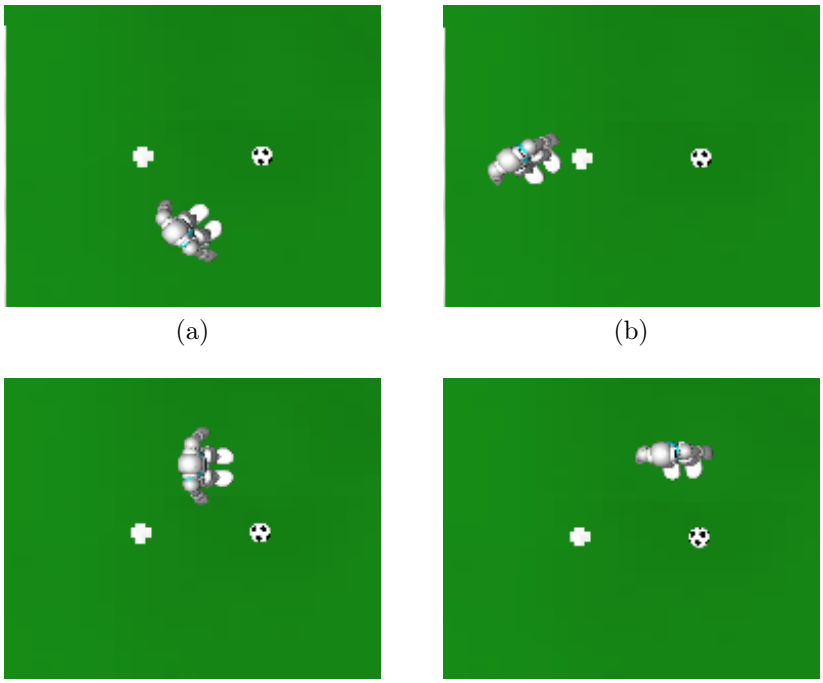


Figura 3.11: Condiciones de entrenmiento para *Ball Dribbling*. La circunferencia roja marca la condición de término del episodio al robot llevar la pelota dentro de su área. El círculo rojo representa el objetivo de *BallDribbling*, el cual es usado en el cálculo de ϕ

Las condiciones de término de un episodio son las siguientes: se supera el tiempo máximo de un episodio, el robot o la pelota salen de la cancha, o se llega al objetivo. Se debe fijar un término del episodio por criterios temporales para evitar comportamientos periódicos, en los cuales la política genere una solución estacionaria. Adicionalmente, asignar una cantidad máxima de iteraciones por episodio permite manejar el decaimiento de la exploración. Por otro lado, se debe terminar el episodio al salir de la cancha el robot o la pelota, debido a que en situaciones reales la cancha tienen dimensiones finitas, y se debe penalizar fuertemente violar dicha restricción. Finalmente, el término del episodio por cumplimiento de objetivo es natural, pero además evita una recompensa inmediatamente negativa tras sobrepasar el objetivo (ϕ se indetermina y posteriormente cambia de signo al pasar tras el objetivo).

Las condiciones iniciales permiten especificar cuales son las situaciones que el agente de aprendizaje debe resolver. A medida que el proceso de aprendizaje avanza, y la exploración disminuye, los estados visitados van siendo modificados, y el uso de una condición inicial determinística favorece el sobre ajuste a dicha situación inicial. Adicionalmente, para algoritmos de aprendizaje con aproximación funcional no lineal, el uso de condiciones iniciales aleatorias cobra mayor importancia, pues a diferencia de los algoritmos no lineales, donde cada experiencia modifica aquellos estados cercanos a ella, en los algoritmos no lineales, sobre todo aquellos basados de redes neuronales de varias capas, una experiencia puede modificar diversos puntos de operación. En esta implementación, se utiliza una inicialización aleatoria uniforme directamente sobre las variables de estado. Esto quiere decir que se determinan las

variables de estado muestreadas desde una distribución uniforme de rangos, tras los cuales se calcula la posición del robot y su orientación inicial, tal de cumplir dicha formulación de estado (la pelota siempre parte en la misma posición). Algunos ejemplos de inicialización aleatoria se presentan en la Figura 3.12.



(a) (b) (c) (d)

Figura 3.12: Ejemplos de condiciones iniciales aleatorias para *Ball Dribbling*

3.2.4. In-walk Kick

En el fútbol robótico, los movimientos de patada pueden ser considerados como los segundos más importantes luego de los movimientos de locomoción, debido a su impacto en el rendimiento del juego (capacidad directa de marcar goles). Debido a esto, diversos métodos han sido desarrollados para el diseño e implementación de movimientos de patada [70] [71] [72] [73], los cuales en su mayoría pueden ser clasificados dentro de 2 grupos: métodos que implementan movimientos independientes [70] [71] [72], y métodos integrados en el sistema de locomoción para generar patadas (*In-walk kick*) [73]. La generación de movimientos independientes para realizar patadas permite el uso de todas las capacidades disponibles en el *hardware*, en especial las capacidades de los actuadores. Adicionalmente, la implementación mediante movimientos independientes permite un análisis más sencillo, permite utilizar criterios de equilibrio estático, etc. Sin embargo, el uso de movimientos independientes sufre de tiempos de activación más extensos, debido a que se debe realizar una transición entre otros movimientos como el de locomoción, antes de poder iniciar el movimiento de patada. Por otro lado, las *In-walk kick*, originalmente diñadas e implementadas en [73], modifican las fases del sistema de locomoción para crear pasos más largos, lo que permite realizar movimientos de patada mientras que el robot camina, lo que lo acerca al movimiento de un jugador de fútbol humano. Esta formulación permite eliminar en gran medida los tiempos de activación

del movimiento, pero las características del movimiento de patada están fuertemente limitadas por las capacidades del sistema de locomoción, además de las limitaciones del *hardware*.

En este trabajo se propone una variante de la implementación de las *In-walk kick* desarrolladas en [73], tal como se presentan en [26]. Esta implementación utiliza las capacidades del sistema de locomoción de manera directa, sin la necesidad de modificar dicho sistema para crear fases distintas, y solo utiliza la inercia del movimiento del robot para efectuar una patada.

3.2.4.1. Modelamiento

Para el modelamiento de este problema se reutiliza la mayor parte del diseño realizado para el problema de *Ball Dribbling*. En particular, se utilizan el mismo espacio de estado, es decir se caracteriza el problema mediante el conjunto de variables geométricas (ρ, γ, ϕ), y se caracteriza el espacio de acciones por el vector de velocidades (V_x, V_y, V_θ) de una caminata omnidireccional, como la desarrollada en [69]. La reutilización de esta formulación permite heredar las mismas ventajas y desventajas de este modelo, tal como se presentan en la Sección 3.2.3, y lo que marca la diferencia entre *Ball Dribbling* e *In-Walk Kicks* consiste principalmente en el diseño de la recompensa y las condiciones de entrenamiento. Adicionalmente, el diseño propuesto posee numerosas ventajas con respecto a la implementación en [73]. En dicha implementación, las reglas de activación de las *In-walk kicks* son diseñadas manualmente paras las distintas zonas de operación. En cambio la formulación mediante Aprendizaje Reforzado, permite la creación automática de dichas reglas, al mismo tiempo que éstas poseen menores retrasos, y presentan potencialmente un mayor rendimiento, debido a que el proceso completo, desde el acercamiento a la pelota, hasta la patada en sí, son realizados mediante Aprendizaje Reforzado (aprendizaje *end-to-end*).

Pese a que distintas publicaciones validan el rendimiento del diseño anterior [24] [25] [68], dicho diseño no considera diversos aspectos del problema, tales como la naturaleza bípeda del robot, o la implementación del sistema de locomoción. En esta sección se realiza una extensión al modelo básico, debido a que en el problema de *In-walk Kick*, a diferencia del problema de *Ball Dribbling*, el rendimiento del problema es principalmente el resultado de una única interacción entre el robot y la pelota, por lo que es necesario un alto nivel de precisión en las acciones. En particular, las modificaciones propuestas son las siguientes:

- **Sincronización con el sistema de locomoción:** Previas implementaciones para *Ball Dribbling*, consideran el sistema de locomoción como una caja cerrada y generan referencias de velocidad para una caminata omnidireccional, señales que son entregadas de manera uniforme en instantes de tiempo discretos. Sin embargo, los sistemas de locomoción, como el planteado en [69], actualizan la referencia ejecutada solo una vez por cada paso del robot, de una manera no uniforme en el tiempo, pues cada paso se realiza mediante lazo cerrado en un sistema de control interno, de tal manera que no todos los pasos tienen igual duración en la práctica. El hecho de no modelar este fenómeno produce que las acciones realizadas por el sistema de aprendizaje, coincidan con fases aleatorias del sistema de locomoción, incluso permitiendo que algunas acciones

del agente no sean ejecutadas, lo que viola la propiedad de *Markov*. Para solucionar esta limitación, en este diseño se sincronizan las acciones del agente con el sistema de locomoción, de tal manera que el agente solo ejecute acciones en el instante de tiempo en el cual el sistema de locomoción modifica la referencia para el siguiente paso.

- **Tipo de fase:** Debido a la naturaleza del sistema de locomoción desarrollado en [69], donde cada paso culmina con un único pie de soporte, situaciones simétricas ocurren al sincronizar el sistema de locomoción con las acciones del agente. En particular, no se puede identificar cual pié corresponde al pié de soporte (pié fijo durante un paso). Para modelar esta situación, se añade una cuarta dimensión al espacio de estados, que permite representar de manera binaria el pie de soporte.
- **Selector de pié:** Pese a que el sistema de selector de pié propuesto en [24] presenta buenos resultados, puede limitar el rendimiento, debido a que la selección de pié sobre el cual se calculan las variables de estado es desconocida para el agente, lo que viola la propiedad de *Markov*. Adicionalmente, el uso del selector de pié no permite al sistema identificar la simetría del propio robot, y los movimientos que aprende, no pueden hacer un uso correcto de la inercia del robot. En este trabajo, se propone calcular las variables de estado desde el centro del robot (punto intermedio entre los pies del robot) en vez de a partir de sistemas de de referencia basados en los pies, para que el agente, en conjunto con la variable de estado consistente en el tipo de fase, pueda aprender su propio selector de pies.

Finalmente, el esquema de la Figura 3.13 presenta el modelo realizado para *In-walk kick*, donde se identifica la integración del módulo de aprendizaje con respecto a los demás sistemas presentados en [65].

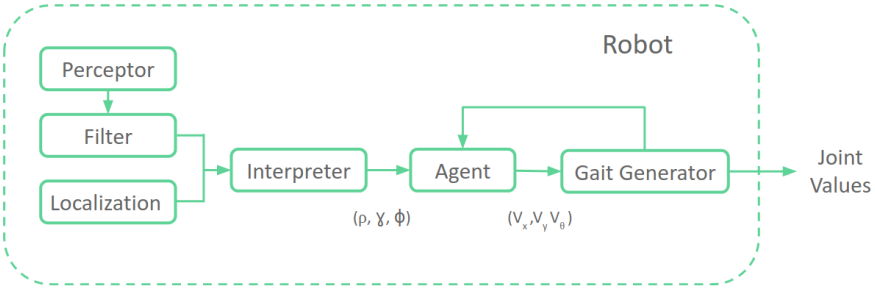


Figura 3.13: Diagrama de bloques del diseño propuesto para *In-walk Kick*

3.2.4.2. Recompensa y rendimiento

La recompensa para el problema de *In-walk Kick* debe permitir generar los movimientos complejos para realizar una patada. Para ello, la recompensa de la ecuación 3.8 incorpora un término de recompensa negativa, que guia al robot a golpear la pelota. Para fomentar patadas fuertes, una opción consiste en asignar una recompensa positiva cuando se marque un gol. Sin embargo, el refuerzo sobre un comportamiento tan específico, ocasiona que dicho refuerzo sea efectuado pocas veces durante el entrenamiento, debido a la dificultad de alcanzar

dicho comportamiento mediante estrategias tradicionales, ocasionando que la aleatoriedad del proceso dificulte fuertemente que la convergencia de la política a movimientos de patada. Para fomentar la ejecución de patadas fuertes, pero de una manera guiada, se utiliza el primer término de la ecuación 3.8, en donde para situaciones en las que se patea la pelota, se asigna una recompensa positiva según la fuerza de la patada y su precisión. El término K regula la importancia de lograr una patada efectiva, y mientras bajos valores dificultan la convergencia, valores excesivos pueden fomentar el surgimiento de comportamientos innecesarios dentro del objetivo de una patada. Por otro lado, los términos φ_0 y α_0 representan el compromiso entre patadas fuertes y precisas. Finalmente, los términos φ_{error} y α_{error} representan el error producto de la distancia final de la pelota al arco, y la desviación de la trayectoria de la pelota con respecto al centro del arco respectivamente, los cuales son además considerados como índices de rendimiento general al caracterizar el resultado del aprendizaje.

$$r = \begin{cases} K exp\left(-\frac{\varphi_{error}}{\varphi_0}\right) exp\left(-\frac{\alpha_{error}}{\alpha_0}\right) & \text{si } ball\, touched \\ -\left(\frac{\rho}{\rho_{max}} + \frac{|\phi|}{\phi_{max}} + \frac{|\gamma|}{\gamma_{max}}\right) & \text{si no} \end{cases} \tag{3.8}$$

- **Rendimiento de distancia**(ρ_{error}): Corresponde a la distancia entre la posición final de la pelota y la linea de gol, normalizada por la distancia inicial de la pelota al arco. Este índice mide la habilidad de generar movimientos bruscos de patada.
- **Rendimiento de ángulo**(φ_{error}): Corresponde a la desviación de la trayectoria de la pelota con respecto al centro del arco, normalizada por la apertura del arco con respecto a la posición inicial de la pelota. Permite medir la precisión de las patadas generadas y depende fuertemente del alineamiento previo.
- **Porcentaje de goles:** Mide el rendimiento real del objetivo de las patadas. Debido a su comportamiento binario para un único episodio, se calcula sobre una ventana móvil de 100 episodios.

3.2.4.3. Condiciones de entrenamiento

De igual manera al problema de *Ball Dribbling* las condiciones de entrenamiento juegan un rol esencial en la política que resulta del proceso de aprendizaje. De igual manera al problema anterior, el episodio termina si el robot o la pelota salen de la cancha, o si el episodio supera cierta cantidad de tiempo. Adicionalmente el episodio debe terminar luego de la primera interacción (colisión) entre el robot y la pelota, de tal manera que la recompensa positiva producto de dicha interacción, se entregue una sola vez por episodio y de manera terminal.

Adicionalmente, las condiciones iniciales de entrenamiento, determinan las zonas de operaciones bajo las cuales se espera que la política resultante presente buenos rendimientos. Las condiciones de entrenamiento, y en particular las iniciales, se pueden observar en la Figura 3.14, en donde se presenta con in círculo rojo el objetivo hacia donde se desea patear (centro del arco) y con un arco de circunferencia rojo las posibles posiciones iniciales del robot. Adicionalmente, la orientación del robot también es aleatoria.

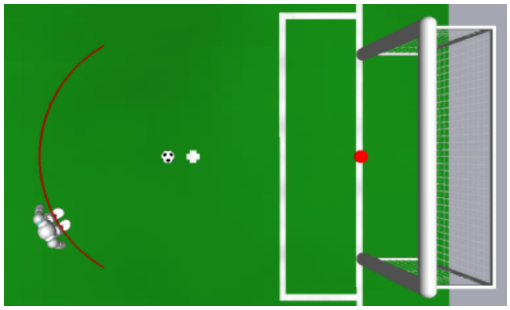


Figura 3.14: Condiciones de entrenamiento para *In-walk Kick*

3.2.5. Navegación Visual sin Mapas

La navegación es un problema recurrente en la robótica móvil en la cual el robot debe planificar y ejecutar su recorrido entre dos puntos. Este desafío tradicionalmente es resuelto de manera separada mediante un sistema de localización y mapeo y otro de planificación. El sistema de localización tiene como objetivo identificar la posición del robot dentro de un mapa. Adicionalmente, en los casos que el mapa sea desconocido, éste también debe ser generado. Por otro lado, el sistema de planificación (*path planning*) se ejecuta de manera paralela al sistema de localización y tiene como objetivo el diseño de una trayectoria desde el punto en donde se estima que está el robot, hacia una posición final, maximizando algún criterio relacionado con el tiempo que tarda en realizarse la trayectoria y la seguridad del camino.

En el contexto de la *RoboCup*, el problema de navegación presenta características importantes. Si bien la base del mapa es conocida (cancha, dimensiones, bordes, etc), los elementos u obstáculos son dinámicos. Es usual encontrar en la *RoboCup*, en particular en la *SPL* algoritmos de *path planning* tales como *Radpidly-exploring Random Tree (RRT)*, campos potenciales y grafos visuales. Dichas técnicas usualmente consideran los obstáculos como estáticos, y replanifican la trayectoria cada ciertos pasos para considerar los cambios en la posición de los demás jugadores. Adicionalmente, debido a que la información que observa cada jugador es limitada, es usual encontrar estrategias como la presentada en [74], en la cual los distintos jugadores de un mismo equipo, estiman de manera colaborativa sus mapas de obstáculos, de manera que el algoritmo de *path planning* pueda tomar decisiones más informadas. El principal limitante de la metodología tradicional consiste en que es incapaz de modelar la dinámica de los demás robots para poder realizar una planificación óptima, la cual es causada por la dificultad de estimar la traslación de robots entre imágenes de manera robusta, mientras el propio robot está en movimiento, además de las dificultades para calcular la orientación de otros robots.

En este trabajo se propone el uso de una navegación sin el uso de mapas, utilizando directamente información visual. Esta propuesta intenta resolver la problemática de los algoritmos tradicionales de *path planning*, en los que la incapacidad de modelar obstáculos móviles o muy cercanos causa considerables colisiones entre robots, tanto de equipos contrarios, como del mismo equipo. Este trabajo está inspirado en la propuestas de [20], [11] y [21] en las cuales se utiliza *Deep Reinforcement Learning* para labores de navegación o similares. En [20] se

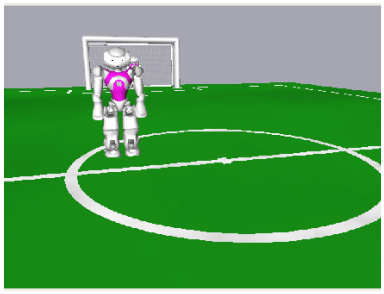
utiliza *Deep Reinforcement Learning* para una tarea de navegación con evasión de obstáculos, la cual sin embargo no utiliza información visual y utiliza directamente las mediciones de sensores de basa dimensionalidad (barrido de láser) como espacio de estados. Por otro lado, en [11] se resuelve una labor similar a navegación, consistente en el videojuego *Doom*, utilizando redes recurrentes para tratar el problema de la observabilidad parcial del entorno. Finalmente, este trabajo continúa el trabajo realizado en [21], en el cual se realiza una labor de navegación con información visual para un agente robótico. La principal diferencia de este trabajo con la propuesta de [21] consiste en que en este trabajo no se busca generar imágenes de profundidad para ser utilizadas como espacio de estados, y se propone el uso de redes recurrentes, como alternativa al uso de *action-repeats*.

Adicionalmente, la formulación de este problema tiene características que lo permiten relacionarse con muchas otras tareas en robótica, por lo cual su desarrollo tiene repercusiones en otras aplicaciones. En primer lugar, la observabilidad del entorno se encuentra fuertemente reducida, debido a que la principal fuente de información, es decir la cámara, tiene una apertura limitada, de modo que no se tiene una visión periférica del entorno. Adicionalmente, el agente debe combinar en sus acciones las labores de llegar a su objetivo, mientras adquiere información para poder planificar su trayectoria de manera segura. Esta necesidad, no explícita, surge debido a la baja observabilidad del entorno, en la cual de no adquirir información de manera adecuada, produce trayectorias con posibles colisiones. Adicionalmente, el proceso de aprendizaje incentiva al agente a aprender a extraer información que es difícil de extraer o modelar manualmente (ubicación y posición de otros robots). Finalmente, el agente de aprendizaje debe interactuar con otros agentes con características dinámicas, lo cual es representativo del contexto general de la robótica móvil.

3.2.5.1. Modelamiento

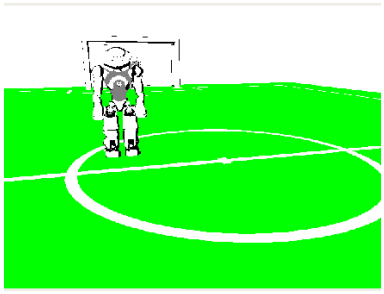
Debido a la naturaleza del problema, que presenta en una altísima dimensionalidad en el espacio de estados, producto del uso de imágenes, toma relevancia el diseño de la formulación de dicho espacio. En particular, este diseño es relevante al momento de considerar la transferencia del producto del aprendizaje al robot físico. Un ejemplo de la importancia de este diseño se presenta en el trabajo realizado en [45], donde se demuestra que un diseño ingenuo con respecto al diseño del problema resulta en un nulo rendimiento en el sistema físico, pese a alcanzar excelentes resultados en los ambientes de simulación.
Debido a la dificultad de simular correctamente la naturaleza de las imágenes de un partido de fútbol robótico real, en particular las condiciones lumínicas, en este trabajo se utiliza una representación alternativa a la tradicional representación *RGB* o escala de grises. En particular, se utiliza un sistema de segmentación automático independiente, basado en el trabajo desarrollado en [75], en el cual se desarrolla un sistema de segmentación, que transforma imágenes *RGB* en imágenes de clases (4 en el contexto del fútbol robótico: verde, blanco, negro y otros). El uso de esta representación permite que el sistema de aprendizaje se vea enfrentado a representaciones de naturaleza similar, tanto durante el aprendizaje en un simulador, como durante su ejecución en un robot real, tal como se muestra en las Figuras 3.15.c y 3.15.d. Adicionalmente, el uso de esta representación permite codificar en un solo canal la información de color, a diferencia de la representación *RGB* que utiliza 3 canales, por lo que las convoluciones de la primera capa presentan un importante ahorro en término

de cómputos, y permite el uso de redes convolucionales cuantizadas [76] sin mayor pérdida de información, debido a que esta representación puede ser codificada en 2 bits (4 clases de color). Finalmente, se opta por un submuestreo de la imagen original (640×480), a una resolución menor de (80×60), siguiendo las líneas de diseño de [4] y [6], en las cuales se utilizan dimensiones similares para resolver problemas directamente a partir de pixeles. El proceso completo a partir del cual se obtiene la representación elegida puede observarse en la Figura 3.15, en donde se hace énfasis en la robustez al *reality gap* de esta representación.



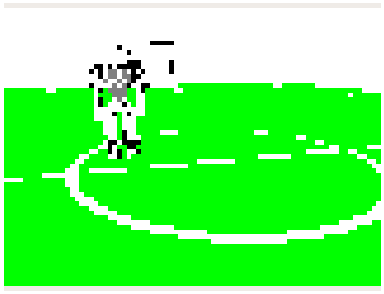
(a) Imagen simulada RGB (640x480)

(b) Imagen real RGB (640x480)

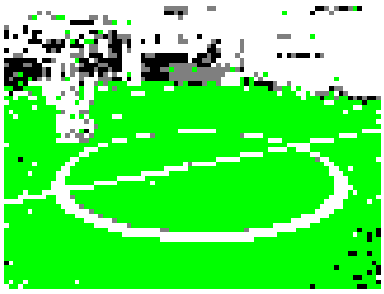


(c) Imagen simulada segmentada (640x480)

(d) Imagen real segmentada (640x480)



(e) Imagen simulada submuestreada (80x60)



(f) Imagen real submuestreada (80x60)

Figura 3.15: Ejemplos de imágenes utilizadas en Navegación visual

Adicionalmente al uso de una imagen segmentada como representación, se añaden dos variables de estado: distancia al objetivo, y orientación hacia el objetivo con respecto a la orientación del robot. Estas variables de estado se calculan con respecto a la posición a la que se desea llegar.

Debido a lo novedoso de este problema, se proponen tres diseños distintos para el espacio

de acciones. El primero de ellos consiste en un modelo simple, donde al igual que en los problemas anteriores, se actúa sobre el controlador omnidireccional proporcionado por el sistema de locomoción. En este caso sin embargo, para simplificar el problema, el agente solo decide el componente de velocidad V_θ, mientras que el resto de los componentes queda fijo $V_x = V_{max}$ y $V_y = 0$. Este modelo permite que el agente solo deba preocuparse de la dirección a la cual se desea llegar mientras esquiva obstáculos. La principales limitaciones de este modelo son que el robot debe tener la posición de la cabeza fija durante el entrenamiento, lo que disminuye la información que queda en su rango visual, y que en casos en que exista un obstáculo cercano, la sub utilización del sistema de locomoción omnidireccional, no le permite evadir correctamente obstáculos.

Un segundo modelo corresponde en añadir al espacio de acciones el movimiento lateral de la cabeza del robot (*pan angle*). Esta formulación resulta interesante, pues incluye de manera intrínseca el problema de la visión activa [77] [78], en el cual el robot debe elegir la posición de su cabeza con tal de maximizar la información adquirida, en el contexto de una tarea específica. A diferencia del trabajo desarrollado en [78], el sistema de visión activa propuesto es entrenado para un fin específico y no requiere el uso de observaciones de un sistema de visión externo. Adicionalmente, a diferencia de [77], la ejecución del sistema de visión activa no requiere el uso de complejos modelos probabilísticos iterativos, debido a que el sistema queda incluido en la política de Aprendizaje Reforzado (que en este caso corresponde a una red convolucional, al utilizar imágenes). Otra consideración importante con respecto a este diseño del espacio de acción, es que requiere una variable de estado adicional correspondiente al ángulo actual de la cabeza (*pan*), para que el agente tenga una referencia de hacia donde está observando. Finalmente, se debe considerar que si bien esta formulación presenta un potencial mucho más amplio, dificulta el proceso de aprendizaje, debido a los grados de libertad que introduce.

Un último modelo para el espacio de acciones consiste en la adición de los componentes de velocidad V_y y V_θ de la formulación del controlados omnidireccional del sistema de locomoción. Esta última formulación permite el uso completo de las capacidades del sistema locomoción disponible, pero nuevamente aumenta la complejidad del entrenamiento.

3.2.5.2. Recompensa y Rendimiento

Para fomentar el avance del robot hacia su objetivo, se formula la recompensa de la ecuación 3.9. Mientras el robot se acerca hacia su objetivo la recompensa es positiva, mientras que si se aleja, la recompensa es negativa. Adicionalmente, si el robot colisiona con otro obstáculo o sale de la cancha el episodio termina, lo que es considerado como un castigo, pues deja de recibir recompensas positivas (favoreciendo la evasión de obstáculos). Una característica de este modelo de recompensa, es que la recompensa total a la que puede acceder el robot durante un episodio, está acotada por la distancia inicial hacia el objetivo.

$$r = V_x cos(\theta) \tag{3.9}$$

Un aspecto negativo del diseño anterior, es que si bien es elegante en su simpleza, no

permite asignar un compromiso parametrizable entre las condiciones terminales y el avance hacia el objetivo. Es por esto que también se considera la recompensa de la ecuación 3.10, la cual utiliza en todo momento recompensas negativas, favoreciendo el acercamiento hacia el objetivo, debido a que termina el episodio, pero se penaliza fuertemente mediante el valor de K aquellas situaciones en las cual el episodio termina sin llegar al objetivo, para evitar que el agente de aprendizaje favorezca buscar el término temprano del episodio en vez de avanzar hacia el objetivo.

$$r = \begin{cases} -K & \text{si colisión} \\ -V_{max} + V_x cos(\theta) & \text{si no} \end{cases} \tag{3.10}$$

Debido a la complejidad del entrenamiento, es difícil considerar mediciones de rendimiento de la política a partir de un único episodio. Por esta razón, se define como índice de rendimiento, el cumplimiento de llegar al objetivo sin colisiones, promediado mediante una ventana móvil, por varios episodios, de tal manera de poder tener una noción del comportamiento de la política en diferentes puntos de operación determinados por las condiciones iniciales, que incluyen el objetivo al cual se desea llegar. Un índice adicional de rendimiento, tiene relación a la eficiencia de la trayectoria generada, por lo que también se utiliza una ventana móvil sobre el criterio V_{avg} definido para el problema de *ball dribbling*, que permite cuantificar el rendimiento de una trayectoria, mediante su proyección sobre una trayectoria lineal ideal (la cual usualmente es infactible debido a los obstáculos).

3.2.5.3. Condiciones de entrenamiento

Debido a las formulaciones de recompensa realizadas en la Sección 3.2.5.2, los criterios de término de un episodio son claros. El episodio debe terminar el existir una colisión entre el agente y un obstáculo, al llegar al objetivo y al salir de la cancha. Adicionalmente, el episodio debe terminar si se sobrepasa un tiempo límite.

Las condiciones iniciales es este caso no solo abarcan la posición inicial aleatoria del robot, sinó que también la posición inicial de los obstáculos de navegación (otros robots). Adicionalmente, para cada episodio se utiliza un objetivo de navegación distinto, el cual puede estar ocupado por otro robot.

Además de las condiciones iniciales, se debe considerar la naturaleza de los obstáculos. En este caso se identifican dos tipos de problema a evaluar. Un entrenamiento en los cuales los obstáculos son aleatorios, pero estáticos, y otro donde los obstáculos tienen sus propios destinos de navegación, en cuyo caso los robots que no forman parte del proceso de aprendizaje, utilizan herramientas de navegación tradicionales y sus objetivos son también aleatorios.

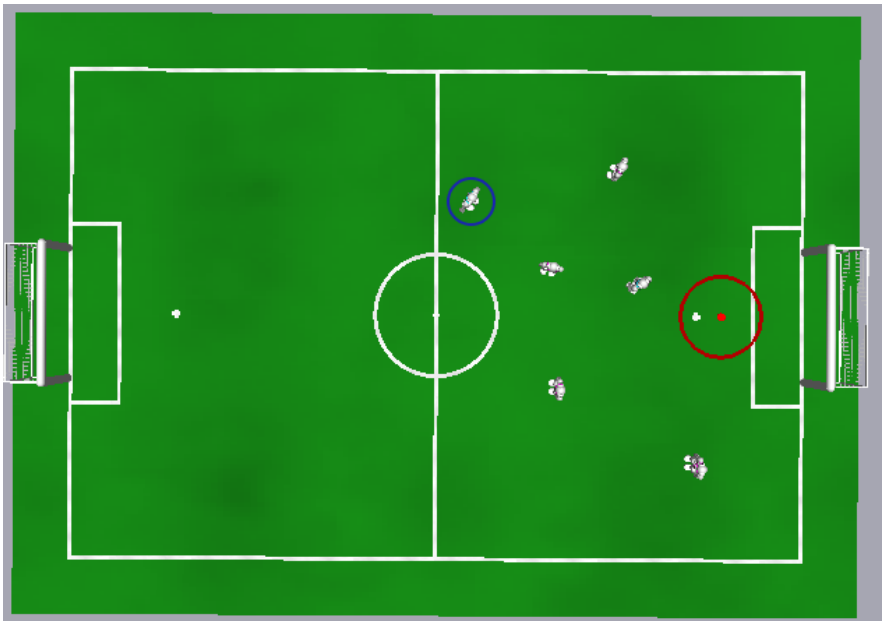


Figura 3.16: Condiciones de entrenamiento para Navegación Visual

En la Figura 3.16 se presenta un ejemplo de las condiciones de entrenamiento de un episodio. El robot encerrado por la circunferencia azul corresponde al agente, mientras que el resto de los robots corresponden a obstáculos. El objetivo de navegación queda marcado por el centro del círculo rojo, y su radio representa la tolerancia de convergencia, con el cual se considera que el robot alcanza el objetivo.

Capítulo 4

Análisis de Resultados

En este capítulo se presentan los resultados de los procesos de Aprendizaje Reforzado para los problemas identificados en las Secciones 3.2.3, 3.2.4 y 3.2.5. Para cada uno de los problemas identificados, se utilizan algunas de las herramientas introducidas en el Capítulo 2, según la implementación detallada en la Sección 3.1. El objetivo de los experimentos realizados es validar las hipótesis planteadas en el modelamiento de cada problema, validar el funcionamiento de los algoritmos presentes en la literatura para problemas relacionados al caso de estudio y comparar el desempeño de los mismos, con el objetivo de poder identificar de manera empírica su comportamiento en diversas situaciones, de tal forma de poder determinar los casos de uso de cada herramienta, además de permitir extraer información útil para otros desarrollos.

En la Sección 4.1 se presentan los experimentos realizados para el problema de *Ball Dribbling*. Debido a que *Ball Dribbling* corresponde a un problema previamente reportado en la literatura, el enfoque de los experimentos realizados se centra en la comparación de distintos algoritmos, utilizando como principales indicadores de comparación el rendimiento final, los tiempos de ejecución, y la naturaleza del proceso de entrenamiento. Debido a la alta cantidad de hiperparámetros, la mayoría de estos se elige de acuerdo a los valores sugeridos en la literatura, mientras se enfoca en el estudio de dichos parámetros que tienen mayor incidencia sobre los criterios planteados.

Por otro lado, en la Sección 4.2 se exponen las pruebas realizadas sobre el problema de *In-walk Kick*. A diferencia de la metodología de los experimentos realizados para *Ball Dribbling*, para *In-walk Kick* el enfoque se centra en la comparación de los modelos básico y extendido, buscando evaluar la diferencia entre diseños ingenuos, y aquellos que modelan las interacciones del agente con sistemas externos. Adicionalmente se exploran en esta sección el uso de aproximadores no convencionales, su rendimiento y beneficios computacionales.

Finalmente, en la Sección 4.3 se ofrecen los resultados para el problema de Navegación Visual. Debido a que corresponde al problema menos documentado en la literatura, y a que presenta una mayor dificultad, la naturaleza de los experimentos se enfoca tanto en el rendimiento alcanzado por las distintas herramientas, ejemplos de la política resultante, posteriores análisis del proceso de diseño, falencias encontradas y metodologías para abordarlas.

4.1. Ball Dribbling

Previo a la presentación de los distintos experimentos realizados, se detallan a continuación los distintos parámetros propios de las condiciones de entrenamiento y otros hiperparámetros producto del proceso de diseño. Dichos parámetros son comunes a todos los experimentos realizados a menos que se indique lo contrario.

En la Tabla 4.1 se presentan los parámetros relevantes a las condiciones de entrenamiento, cuya representación gráfica puede observarse en la Figura 3.11. Las dimensiones de la cancha corresponden a $9.000\,[mm]$ de largo y $6.000\,[mm]$ representando a las dimensiones de una cancha oficial *RoboCup*. En esta cancha el objetivo es llevar la pelota hacia $Objetivo_{dribbling}$ desde $Pelota_{inicial}$. El hecho de que tanto el objetivo como la posición inicial sean posiciones fijas en la cancha no sesga de manera importante el proceso de entrenamiento, debido que la formulación del espacio de estados no utiliza coordenadas globales de la cancha. El tiempo máximo asignado a cada episodio es T_{max}, donde su valor corresponde aproximadamente al tiempo que el robot se demora en dar una vuelta completa a la cancha. Por otro lado, $\triangle t$ corresponde al intervalo de tiempo entre acciones del agente. Esto quiere decir que cada $\triangle t$ milisegundos el agente toma una decisión, la cual es ejecutada constantemente por el sistema de locomoción hasta la siguiente acción del agente. El valor de $\triangle t$ se escoge a partir de [24] y corresponde aproximadamente al tiempo en el que el robot ejecuta un paso. Finalmente, los valores de ρ_{fault}, γ_{fault} y ϕ_{fault} representan los rango de valores que determinan el criterio de rendimiento *fault* y sus valores inciden directamente en las métricas de rendimiento, además de directa o indirectamente en la función de recompensa.

Tabla 4.1: Parámetros del ambiente de *Ball Dribbling*

Parámetro	Valor
$cancha_{largo}$	$9.000\,[mm]$
$cancha_{ancho}$	$6.000\,[mm]$
$Objetivo_{dribbling}$	$(2.000\,[mm], 0\,[mm])$
$Pelota_{inicial}$	$(-2.600\,[mm], 0\,[mm])$
$R_{convergencia}$	$500\,[mm]$
T_{max}	$200.000\,[ms]$
$\triangle t$	$200\,[ms]$
ρ_{fault}	$600\,[mm]$
γ_{fault}	$20°$
ϕ_{fault}	$20°$
episodios	500

En la Tabla 4.2 se presentan los valores asignados a la inicialización aleatoria. Se plantean las condiciones iniciales en su forma de variables de estado para poder controlar de manera sencilla las zonas de operación que se le exigen al agente que pueda resolver. La elección de valores se basa en aquellas condiciones de juego para las que resulta útil el uso de *Ball Dribbling*. Un ejemplo de este fenómeno es que para distancias altas el uso de *Ball Dribbling* no resulta conveniente, debido a que su modelamiento no incluye la evasión de obstáculos. Finalmente, la posición inicial del robot y su orientación se puede derivar a partir de la

posición del objetivo, de la pelota, y de las condiciones iniciales en su formulación de variables de estado.

Tabla 4.2: Rangos de inicialización aleatoria para *Ball Dribbling*

	Valor mínimo	Valor máximo
$\rho_{inicial}$	$200\,[mm]$	$600\,[mm]$
$\gamma_{inicial}$	$-90°$	$90°$
$\phi_{inicial}$	$-90°$	$90°$

La elección de valores para el espacio de estados se presenta en la Tabla 4.3. Pese a que algoritmos que utilizan redes neuronales *MLP* no requieren conocer los valores extremos del espacio de estados, otros algoritmos, principalmente aquellos que utilizan aproximación lineal, deben ubicar sus funciones base a lo largo del espacio de estados, por lo que conocer de antemano los valores extremos resulta crucial, principalmente para obtener una representación eficiente.

Tabla 4.3: Rangos del espacio de estados para *Ball Dribbling*

	Valor mínimo	Valor máximo
ρ	$0\,[mm]$	$800\,[mm]$
γ	$-120°$	$120°$
ϕ	$150°$	$150°$

Los valores asignados a los rangos del espacio de acciones se presentan en la Tabla 4.4, y su elección se basa en los parámetros asociados de la caminata omnidireccional presente en [65].

Tabla 4.4: Rangos del espacio de acciones para *Ball Dribbling*

	Valor mínimo	Valor máximo
V_x	$0\,[mm/s]$	$100\,[mm/s]$
V_y	$-30\,[mm/s]$	$30\,[mm/s]$
V_θ	$-0,4\,[rad/s]$	$0,4\,[rad/s]$

4.1.1. Actor-Crítico Lineal

Los parámetros del agente Actor-Crítico Lineal, desarrollado en la Sección 2.2.4 se presentan en la Tabla 4.5. El valor de γ es ampliamente utilizado en la literatura para problemas con un objetivo claro, El valor de la tasa de aprendizaje del crítico $\alpha_C = 0,1$ es común en algoritmos de crítico, y se escoge $\alpha_A < \alpha_C$, lo que corresponde a un requerimiento teórico en la convergencia de los algoritmos de *Actor-Crítico*, que de forma general significa que el crítico debe ser capaz de evaluar el desempeño del actor más rápido de lo que éste puede cambiar. Por otro lado, se utiliza tanto para el actor como el crítico el mismo factor de decaimiento de las trazas de elegibilidad, utilizando el valor más recurrente en la literatura, debido a que

menores valores no presentan mejoras significativas, y valores más altos presentan errores de convergencia. Finalmente el valor σ_{RBF} representa el ancho del *kernel Gaussiano*, y se presenta en una escala normalizada en donde $\triangle$ consiste en la separación de las bases de aproximación uniformemente distribuidas lo largo de una dimensión del espacio de estados.

Tabla 4.5: Parámetros del agente Actor-Crítico Lineal para *Ball Dribbling*

Parámetro	Valor
α_A	$0,01$
α_C	$0,1$
γ	$0,99$
λ_A	$0,9$
λ_C	$0,9$
Función base	RBF
σ_{RBF}	$0,5\triangle$

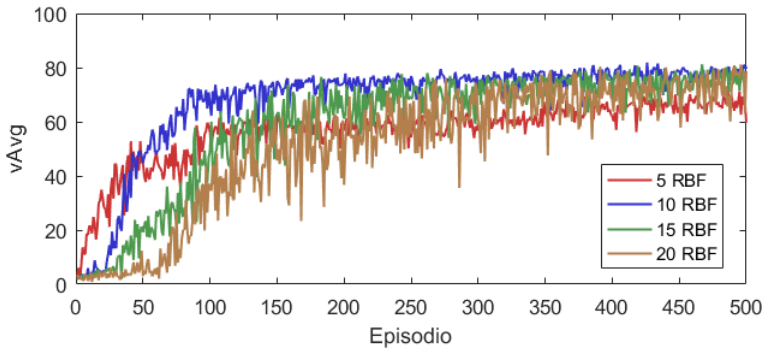
Adicionalmente, en todas las pruebas realizadas para *Ball Dribbling* con *Actor-Crítico Lineal*, se utiliza un método de exploración de ruido aditivo *Gaussiano*, cuyos parámetros quedan descritos en la Tabla 4.6. Los parámetros presentes en la Tabla 4.6 se encuentran normalizados, es decir, su acción en la política debe ser escalada mediante el rango de cada dimensión del espacio de acciones. σ_0 representa la exploración inicial, la cual decrece según una regla de decaimiento, en este caso exponencial, hasta el valor σ_{final} en $\sigma_{dec} \times episodios$. Adicionalmente, para que la política ejecutada no se vuelva determinista, el valor de exploración tiene como mínimo σ_{min}.

Tabla 4.6: Parámetros de exploración *Gaussiana* para *Ball Dribbling*

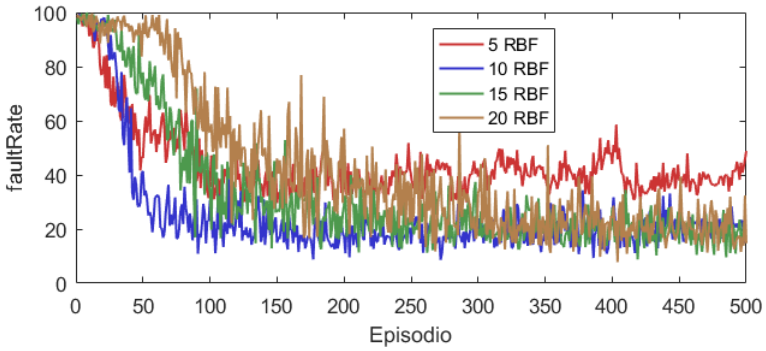
Parámetro	Valor
σ_0	1
σ_{dec}	$0,3$
σ_{final}	$0,15$
σ_{min}	$0,01$
Decaimiento	Exponencial

Además de los parámetros mencionados en las Tablas 4.5 y 4.6, se debe definir la cantidad de bases por dimensión en la aproximación funcional, decisión que impacta fuertemente en los tiempos de ejecución, expresividad de la representación, y proceso de convergencia. A continuación se presentan los resultados producto de las simulaciones de variar la cantidad de bases *RBF*.

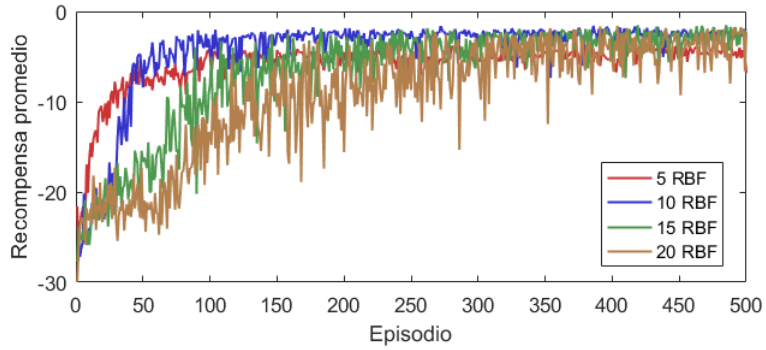
La Figura 4.1 presenta los distintos indicadores del proceso de aprendizaje para 5, 10, 15 y 20 funciones base por dimensión del espacio de estados. Los gráficos corresponden en cada caso al promedio de 10 simulaciones del proceso de aprendizaje completo, y la varianza de cada curva representa tanto el efecto de las condiciones iniciales aleatorias, así como la capacidad de generalización del aprendizaje. Adicionalmente, los índices de rendimiento, inicialmente introducidos con valores dentro de la unidad, se presentan en términos porcentuales.



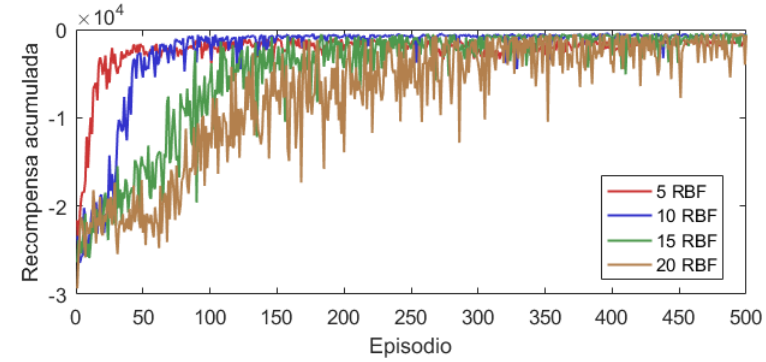
(a)



(b)



(c)



(d)

Figura 4.1: Variación del número de funciones base para el algoritmo Actor-Crítico Lineal

Tabla 4.7: Indices de rendimiento en *test* para el agente Actor-Crítico Lineal en el problema de *Ball Dribbling*

RBF	$V_{avg}\,[\,\%]$	*faultRate* $[\,\%]$	Número de parámetros	Tiempo de procesamiento$[ms]$
5	$77,75 \pm 28,19$	$46,45 \pm 28,94$	500	$0,04$
10	$91,17 \pm 9,18$	$19,5 \pm 15,76$	4.000	$0,23$
15	$87,64 \pm 15,09$	$17,91 \pm 16,91$	13.500	$0,64$
20	$81,42 \pm 28,23$	$22,99 \pm 27,97$	32.000	$1,36$

La Tabla 4.7 presenta las estadísticas de rendimiento de la política final de los casos anteriores. Los índices de rendimiento V_{avg} y *faultRate* corresponden a la media y a la desviación estandar de utilizar la política final por 100 episodios para cada uno de los 10 experimentos realizados para cada conjunto de parámetros. Adicionalmente, los tiempos de ejecución son calculados ejecutando la política 10^6 veces, utilizando como herramienta de medición, el reloj de pared interno del computador donde se realizan las simulaciones.

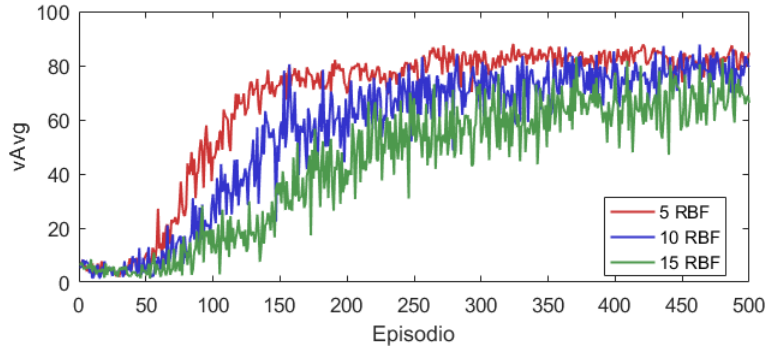
4.1.2. SARSA

De manera análoga al caso del agente Actor-Crítico Lineal, en la Tabla 4.8 se presentan los parámetros propios de *SARSA*. A diferencia del Actor-Crítico Lineal, *SARSA* corresponde a un método de crítico, por lo cual posee menos parámetros. En este caso se utilizan aquellos correspondientes al crítico del agente anterior, justificando la elección de valores de la misma manera.

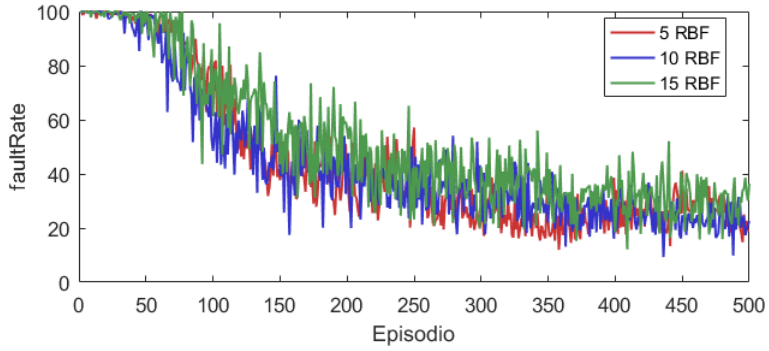
Tabla 4.8: Parámetros de *SARSA* para *Ball Dribbling*

Parámetro	Valor
α	$0,1$
γ	$0,99$
λ	$0,9$
Función base	RBF
σ_{RBF}	$0,5\triangle$

A diferencia del caso anterior, *SARSA* utiliza acciones discretas, por lo que se debe utilizar un método distinto de exploración. En esta ocasión se utiliza el método $\epsilon - greedy$, cuyos parámetros se encuentran en la Tabla 4.9. De manera análoga al caso de exploración *Gaussiana* se utiliza una estrategia de decaimiento en la exploración, para favorecer fuertes exploraciones en episodios iniciales, y explotación a medida que avanzan los episodios. En este caso el parámetro de modulación en la exploración consiste en el valor de ϵ, el cual se hace decrecer exponencialmente con un valor mínimo, tal que la política no se vuelva determinística.



(a)



(b)

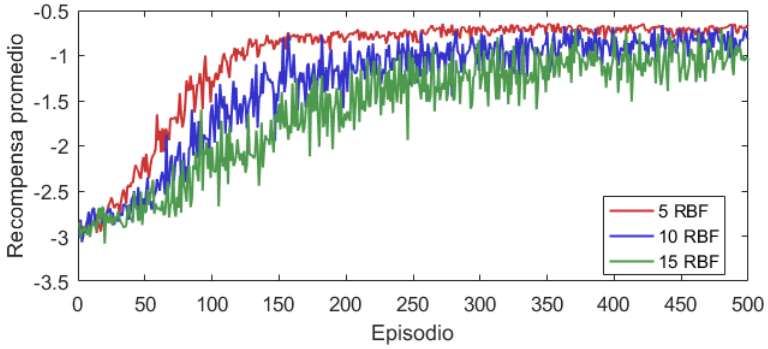
Figura 4.2: Índices de rendimiento para la variación del número de funciones base para el algoritmo *SARSA*, manteniendo un número de acciones por dimensiones igual a 11 para el problema *Ball Dribbling*

Tabla 4.9: Parámetros de exploración $\epsilon - greedy$ para el problema de *Ball Dribbling*

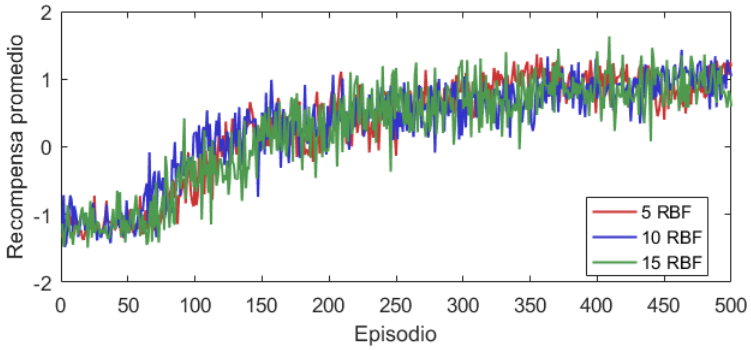
Parámetro	Valor
ϵ_0	1
ϵ_{dec}	$0,3$
ϵ_{final}	$0,05$
ϵ_{min}	$0,01$
Decaimiento	Exponencial

Adicionalmente a la cantidad de funciones base en el espacio de estados, en *SARSA*, se debe elegir la cantidad de acciones por cada dimensión del espacio de acciones. En las Figuras 4.2 y 4.3 se muestran los índices de rendimiento y recompensa respectivamente, del proceso de variar la cantidad de funciones base, manteniendo una cantidad de 11 acciones por dimensión. Adicionalmente, en las Figuras 4.4 y 4.5 se presentan los índices de rendimiento y recompensa con respecto a los experimento de modificar el número de acciones, manteniendo en este caso 10 funciones base por dimensión del espacio de estados.

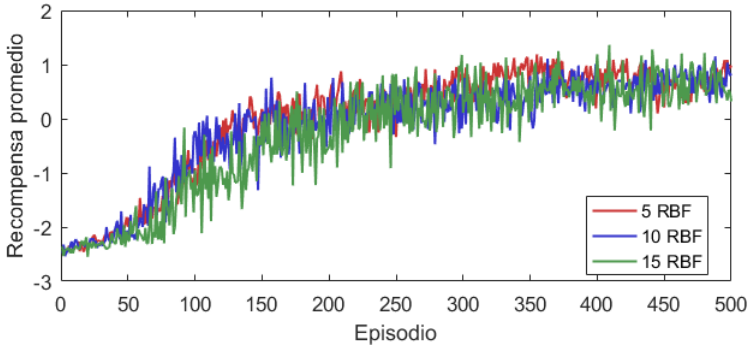
La Tabla 4.10 contiene las estadísticas de interés de los experimentos relacionados a *SARSA* y *Ball Dribbling*. Nuevamente los resultados son el producto de realizar 10 simulaciones completas para cada caso, y los tiempos de ejecución son calculados ejecutando la política



(a) Agente V_x

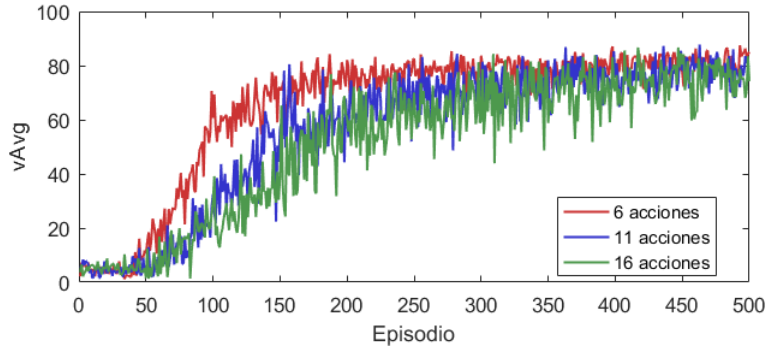


(b) Agente V_y

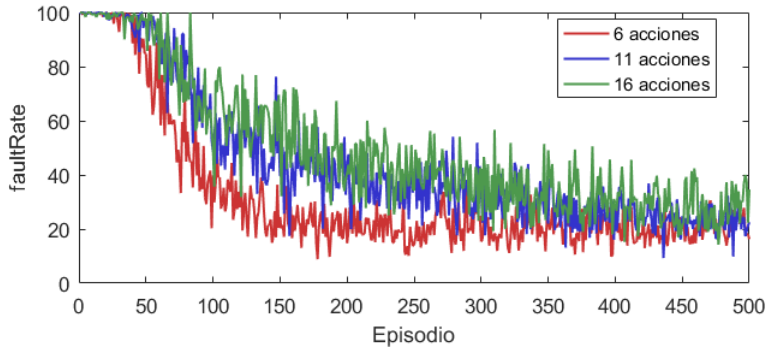


(c) Agente V_θ

Figura 4.3: Recompensas del agente para la variación del número de funciones base para el algoritmo *SARSA*, manteniendo un número de acciones por dimensiones igual a 11 para el problema *Ball Dribbling*

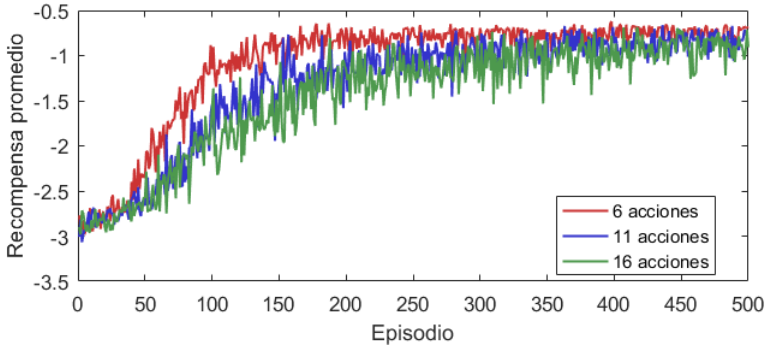


(a)

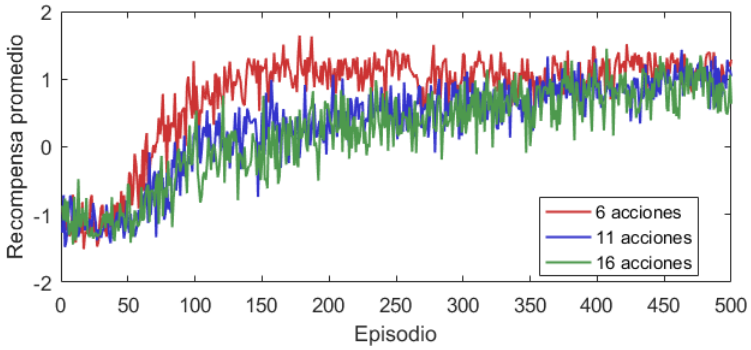


(b)

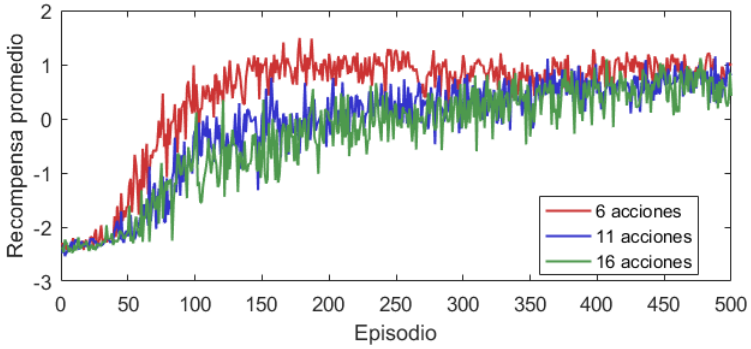
Figura 4.4: Índices de rendimiento para la variación del número de acciones por dimensión para el algoritmo *SARSA*, manteniendo el número de funciones base por dimensión en 10 para el problema *Ball Dribbling*



(a) Agente V_x



(b) Agente V_y



(c) Agente V_θ

Figura 4.5: Recompensas del agente para la variación del número de acciones por dimensión para el algoritmo *SARSA*, manteniendo el número de funciones base por dimensión en 10 para el problema *Ball Dribbling*

Tabla 4.10: Indices de rendimiento en *test* para el agente *SARSA* en el problema de *Ball Dribbling*

RBF	*Acciones*	V_{avg}	*faultRate*	Número de parámetros	Tiempo de procesamiento[ms]
5	11	$83,46 \pm 10,16$	$23,18 \pm 16,13$	4.125	$0,07$
10	6	$81,11 \pm 12,91$	$21,28 \pm 17,78$	18.000	$0,32$
10	11	$78,49 \pm 16,18$	$44,45 \pm 33,21$	33.000	$0,4$
10	16	$75,1 \pm 20,8$	$28,41 \pm 24,17$	48.000	$0,42$
15	11	$68,12 \pm 22,3$	$30 \pm 23,18$	111.375	$1,21$

una cantidad considerable de veces, con el objetivo de poder realizar una medición correcta.

4.1.3. DDPG

En el caso del algoritmo *DDPG*, debido a su uso de redes neuronales complejas en una estructura de Actor-Crítico, el proceso de diseño involucra la elección de numerosos parámetros.

La arquitectura utilizada se presenta en las Figuras 4.6 y 4.7, la cual está basada fuertemente el trabajo desarrollado en [6]. En ambas redes se utilizan funciones de activación *ReLU* en las capas interiores para facilitar la propagación del gradiente, mientras que en la capa final del crítico se utiliza una función de activación lineal, debido a se trata de un problema de regresión, donde el valor del par estado-acción no se encuentra acotado. Por otro lado, la función de activación de la capa de salida del actor corresponde a una tangente hiperbólica, con el objetivo de delimitar el rango de acciones posibles. En esta implementación, se normalizan tanto las entradas y las salidas de la red, de tal manera que las entradas y las salidas siempre están en el rango $[-1, 1]$, con el objetivo de disminuir los efectos de las escalas que utiliza cada señal. Finalmente, la red se inicializa con el método *xavier*, que busca inicializar la red de tal manera que el valor de cada neurona distribuya de manera normal con media cero y varianza unitaria.

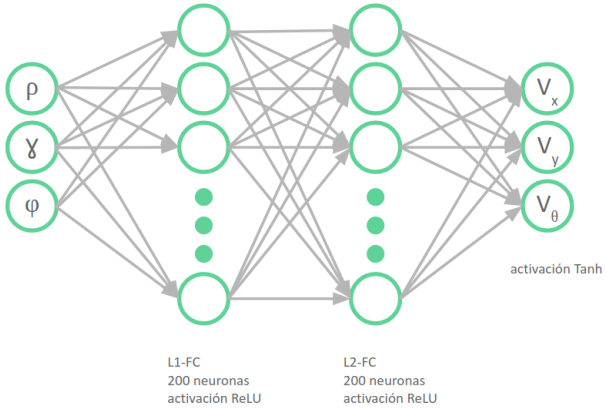


Figura 4.6: Estructura de la red neuronal del actor de *DDPG* en el problema *Ball Dribbling*

Tabla 4.11: Parámetros de *Adam* para el problema de *Ball Dribbling*

Parámetro	Valor
α_C	10^{-3}
α_A	10^{-4}
β_1	$0,9$
β_2	$0,999$

Tabla 4.12: Parámetros de *DDPG* para el problema de *Ball Dribbling*

Parámetro	Valor
γ	0.99
Frecuencia de *batch*	1
Tamaño de *batch*	32
Capacidad del *Replay Memory*	100000
τ	$0,001$

Figura 4.7: Estructura de la red neuronal del crítico de *DDPG* en el problema *Ball Dribbling*

Adicionalmente a la arquitectura y los parámetros propios de una red neuronal, el algoritmo de optimización utilizado corresponde a *Adam* [40], cuyos parámetros se presentan en la Tabla 4.11, donde β_1 y β_2 corresponden a las estimaciones del primer y segundo momento respectivamente, mientras que los parámetros α_C y α_A corresponden a las tasas de aprendizaje del crítico y el actor, donde nuevamente se escogen de tal manera que $\alpha_C > \alpha_A$.

Finalmente, en la Tabla 4.12 se presentan los parámetros propios de *DDPG*, donde una frecuencia de *batch* unitaria significa que en cada iteración se realiza un *minibatch* y el parámetro τ corresponde al factor de actualización de las *target network*.

Tabla 4.13: Índices de rendimiento en *test* para el agente *DDPG* en el problema de *Ball Dribbling*, variando la cantidad de neuronas

Neuronas	V_{avg}	*faultRate*	Número de parámetros	Tiempo de procesamiento[ms]
50	$68,21 \pm 36,79$	$56,06 \pm 36,25$	2.903	$0,07$
100	$89,97 \pm 10,10$	$22,37 \pm 19,57$	10.803	$0,32$
200	$92,24 \pm 6,13$	$14,39 \pm 9,15$	41.603	$0,4$
300	$91,81 \pm 5,82$	$12,08 \pm 7,07$	92,403	$0,42$

Tabla 4.14: Índices de rendimiento en *test* para el agente *DDPG* en el problema de *Ball Dribbling* variando el tamaño del *minibatch*

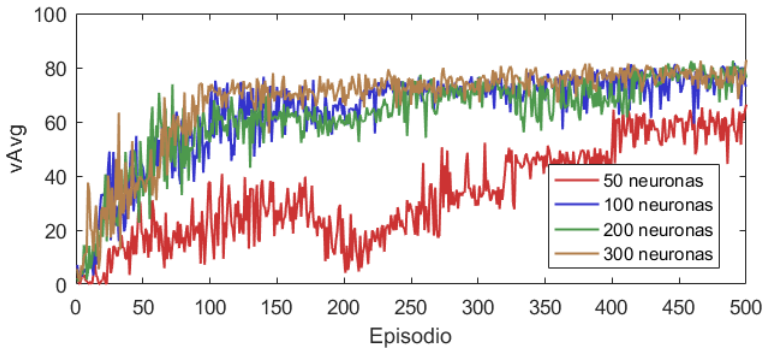
Tamaño de *minibatch*	V_{avg}	*faultRate*
1	$83,46 \pm 10,16$	$23,18 \pm 16,13$
8	$81,11 \pm 12,91$	$21,28 \pm 17,78$
32	$78,49 \pm 16,18$	$44,45 \pm 33,21$
64	$75,1 \pm 20,8$	$28,41 \pm 24,17$

Debido a la alta cantidad de parámetros presentes en *DDPG*, se realizan numerosas pruebas, variando en cada ocasión uno de los parámetros mencionados anteriormente. En la Figura 4.8 se presentan los resultados de variar el número de neuronas en cada capa, con el objetivo de analizar el efecto del número de parámetros, tanto en el nivel de rendimiento que permite alcanzar (expresividad), así como su velocidad de convergencia, estabilidad, etc. Luego, en la Figura 4.9 se presentan los efectos de modificar el tamaño de los *minibatch*, buscando estudiar la estabilidad del entrenamiento, a medida que se acercan a las condiciones presentadas en [35]. Finalmente, en la Figura 4.10 se observan los resultados de utilizar las distintas estrategias para delimitar el rango de las acciones de la política, es decir comparar el método base de utilizar una función tangente hiperbólica (*squashing gradients*), con *zeroing gradients* e *inverting gradients* [52].

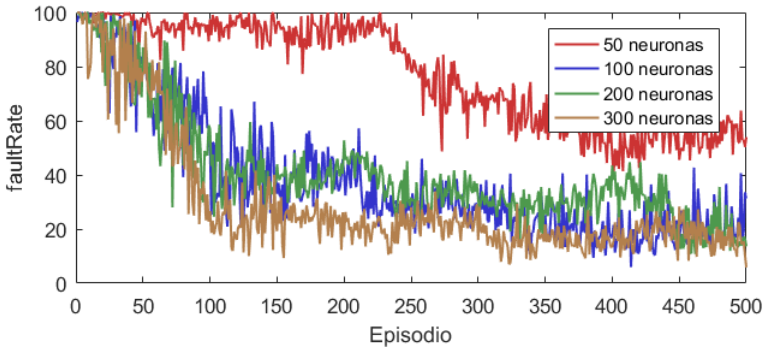
Adicionalmente a los resultados del proceso de entrenamiento, las Tablas 4.13, 4.14 y 4.15 presentan las contrapartes correspondientes a los índices de rendimiento en tiempo de *test*. En esta ocasión se hace la distinción que los tiempos de ejecución con calculados utilizando únicamente la CPU del computador en el cual se realizan las simulaciones, tanto por la ausencia de dispositivos como GPU en la mayoría de los sistemas robóticos, como por el hecho que para modelos de bajo tamaño como los utilizados, el uso de GPU ralentiza los tiempos de ejecución.

Tabla 4.15: Índices de rendimiento en *test* para el agente *DDPG* en el problema de *Ball Dribbling* utilizando diversas técnicas de delimitación de la política

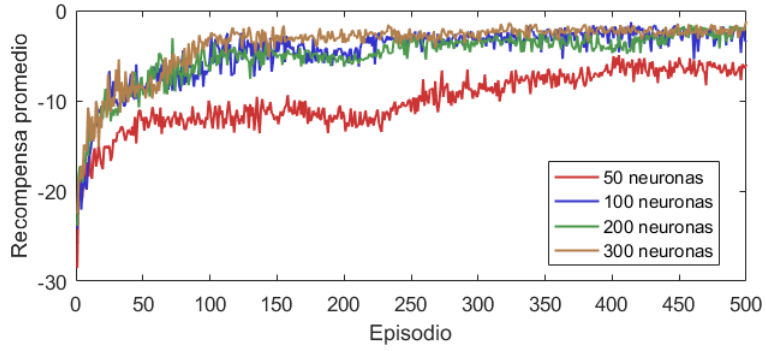
Delimitación	V_{avg}	*faultRate*
squashing	$83,46 \pm 10,16$	$23,18 \pm 16,13$
zeroing	$81,11 \pm 12,91$	$21,28 \pm 17,78$
inverting	$78,49 \pm 16,18$	$44,45 \pm 33,21$



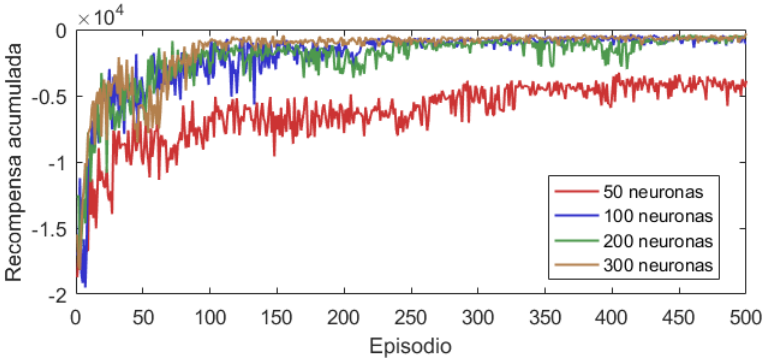
(a)



(b)

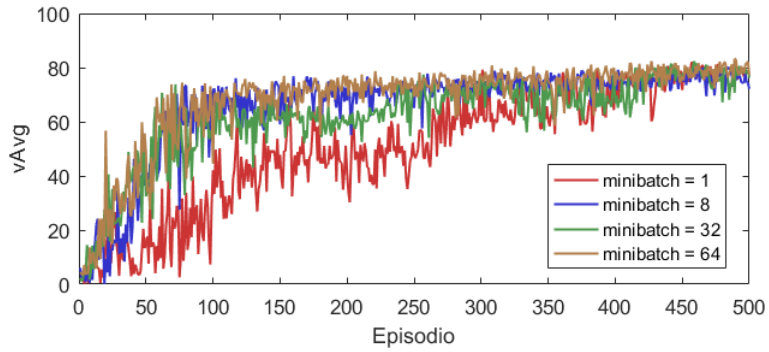


(c)

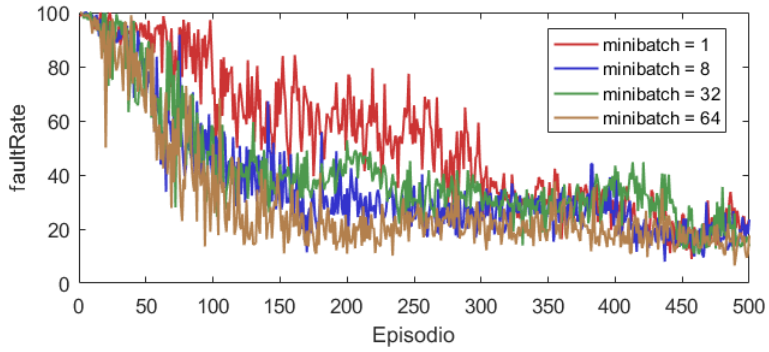


(d)

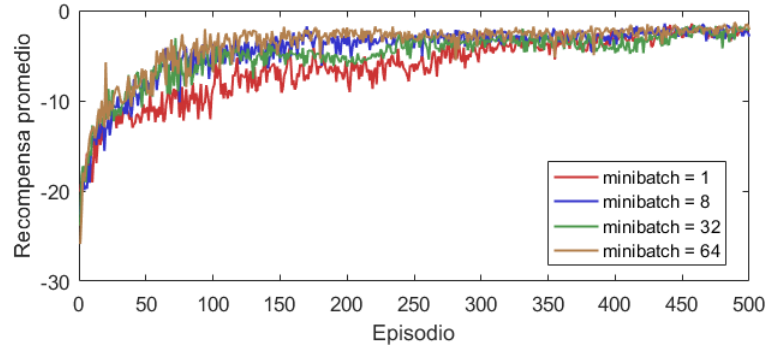
Figura 4.8: Variación del número de neuronas en cada capa (utilizando el mismo número de neuronas en cada capa oculta) en el algoritmo *DDPG* para el problema *Ball Dribbling*



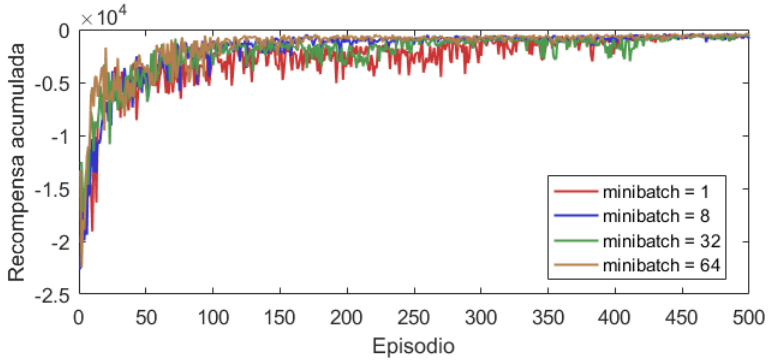
(a)



(b)

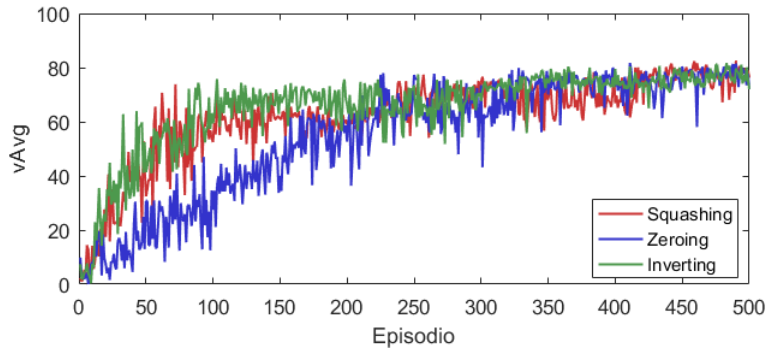


(c)

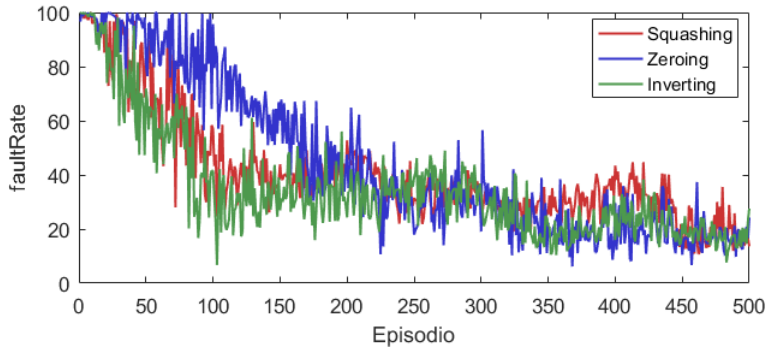


(d)

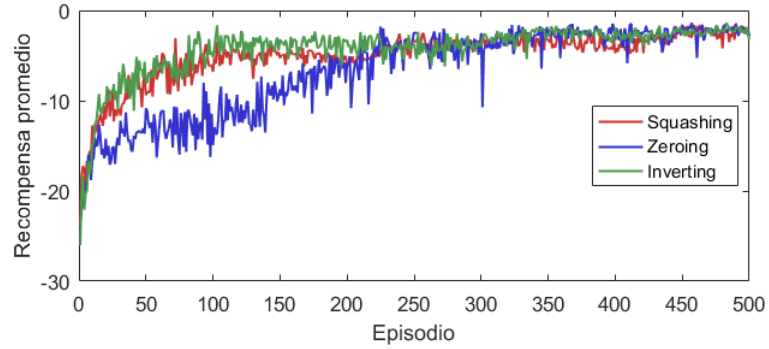
Figura 4.9: Variación del tamaño del *minibatch* en el algoritmo *DDPG* para el problema *Ball Dribbling*



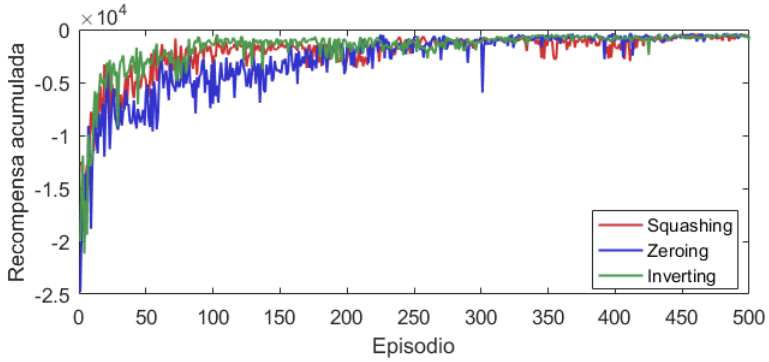
(a)



(b)

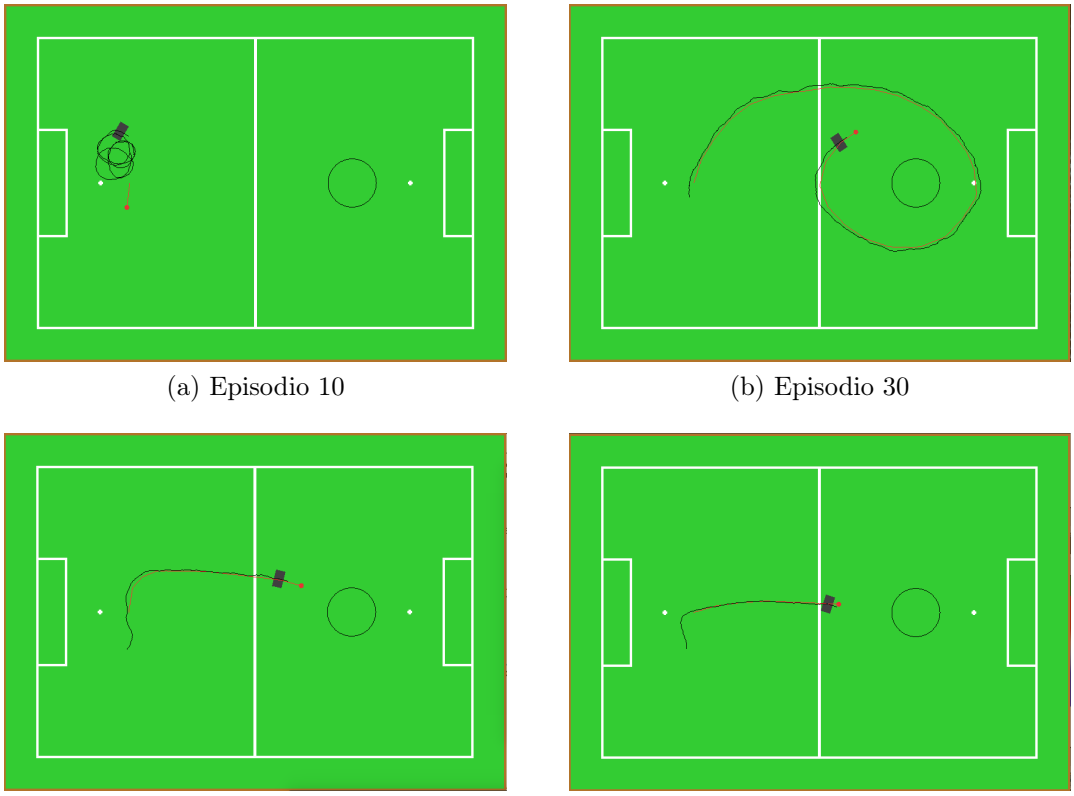


(c)



(d)

Figura 4.10: Distintas técnicas de delimitación de la política en el algoritmo *DDPG* para el problema *Ball Dribbling*



(a) Episodio 10 (b) Episodio 30

(c) Episodio 100 (d) Episodio 400

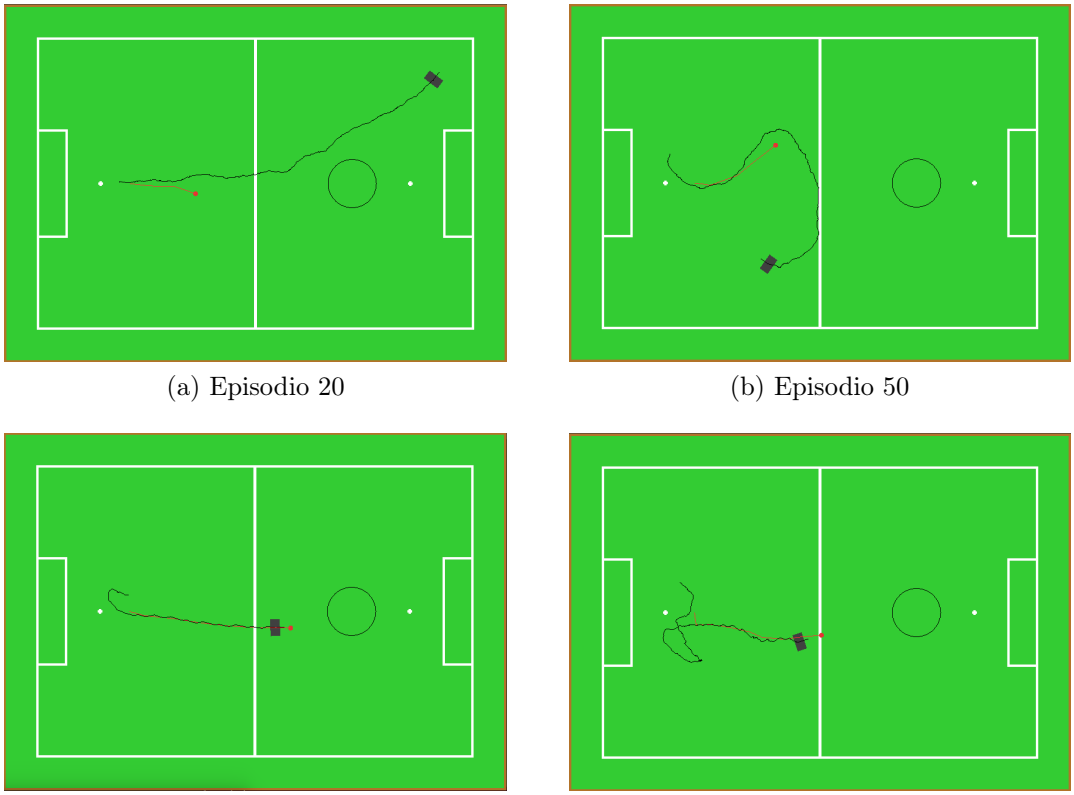
Figura 4.11: Ejemplos de episodios utilizando *DDPG* para el problema *Ball Dribbling*

4.1.4. Proceso de Aprendizaje

Adicionalmente a presentar los gráficos correspondientes al proceso de entrenamiento con los distintos índices de rendimiento, es ilustrador observar la evolución del sistema de aprendizaje a través de los episodios de manera visual. Esta inspección permite identificar la naturaleza del Aprendizaje Reforzado en general, así como también permite evidenciar las diferencias entre algoritmos. Adicionalmente, es mediante la observación continua del proceso de aprendizaje, que se pueden identificar ideas para futuros diseños o mejoras.

Para el problema de *Ball Dribbling*, una manera de observar el proceso de entrenamiento, es ubicar una cámara en el simulador, de tal manera de tener una vista superior, y observar el recorrido que traza tanto el robot como la pelota, permitiendo identificar de manera directa los efectos de la política generada por el proceso de aprendizaje.

Por motivos de comparación, se evalúan los procesos de entrenamiento de *DDPG*, un algoritmo que genera acciones continuas, con *SARSA*, que genera acciones discretas. En las Figuras 4.11 y 4.12 se presentan ejemplos de episodios a lo largo del proceso de entrenamiento, tanto para *DDPG* como para *SARSA*, utilizando trazas negras para indicar el recorrido del robot, y trazas rojas para indicar el recorrido de la pelota.



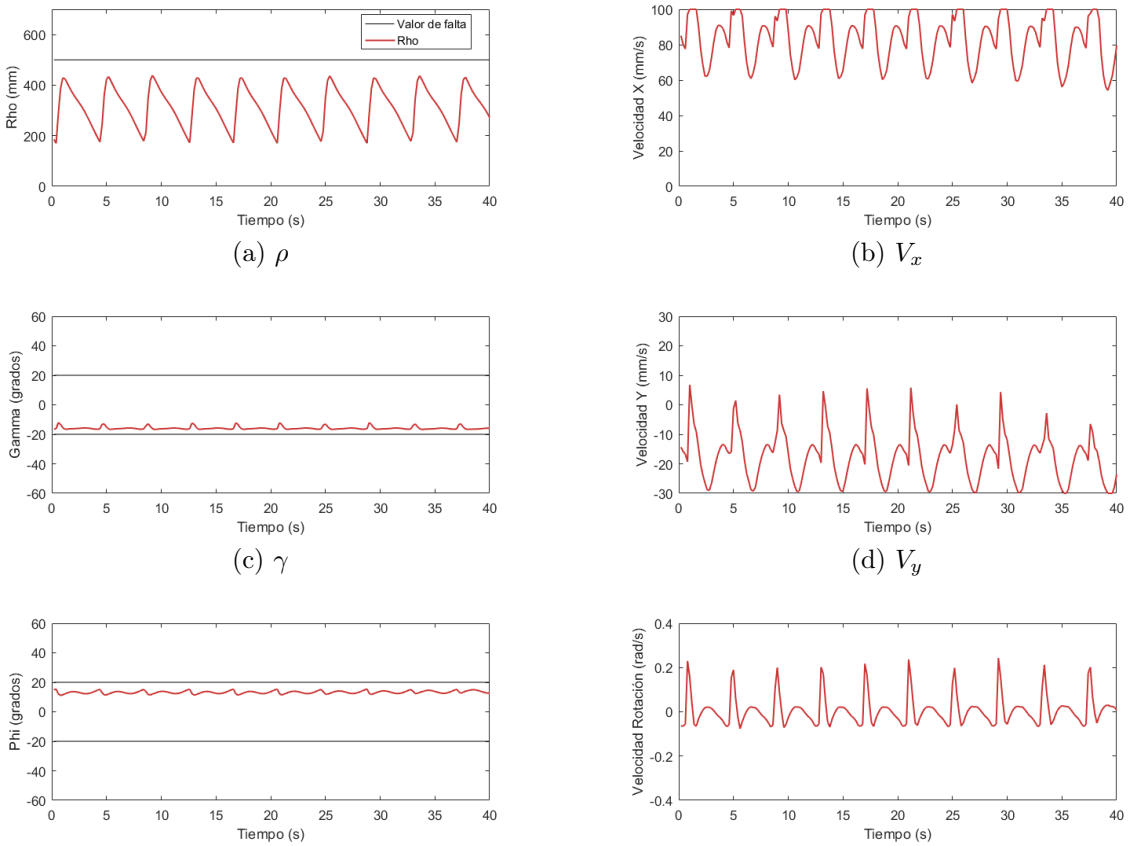
(a) Episodio 20

(b) Episodio 50

(c) Episodio 400

(d) Episodio 401

Figura 4.12: Ejemplos de episodios utilizando *SARSA* para el problema *Ball Dribbling*



(a) ρ (b) V_x

(c) γ (d) V_y

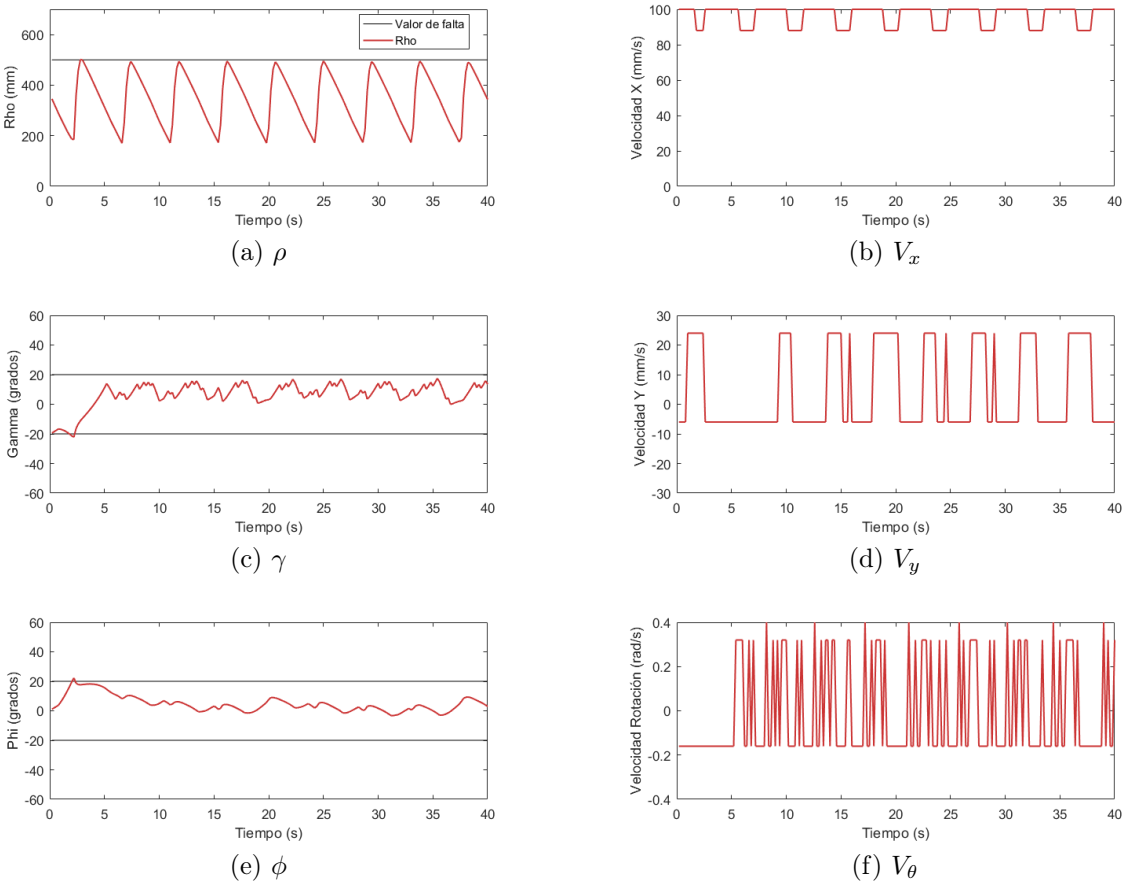
(e) ϕ (f) V_θ

Figura 4.13: Política resultante utilizando *DDPG* para el problema *Ball Dribbling*

4.1.5. Política Resultante

Otra manera de poder evaluar el proceso de aprendizaje, es visualizar de manera simultánea los valores de estado en conjunto con los valores de acciones (es decir el mapeo de la política). Este procedimiento permite evaluar de manera cuantitativa la política, y así identificar posibles fallas en el modelamiento, y otros aspectos.

De manera análoga al caso anterior, se presentan gráficos tanto para el agente *DDPG*(Figura 4.13) como para *SARSA*(Figura 4.14). Los datos de estas figuras corresponden al regimen permanente de la política, es decir ignorando las iteraciones iniciales, en las cuales el robot se alinea con respecto al objetivo y la pelota, y comienza la tarea de regulación. Un aspecto a considerar, es que para los casos presentados en esta sección, se utiliza $\rho_{fault} = 500\,[mm]$.



(a) ρ (b) V_x

(c) γ (d) V_y

(e) ϕ (f) V_θ

Figura 4.14: Política resultante utilizando *SARSA* para el problema *Ball Dribbling*

4.1.6. Análisis

Parámetros de diseño

El diseño de sistemas de aprendizaje involucra la definición de numerosos parámetros. En el caso del Aprendizaje Reforzado, estos consisten principalmente de aquellos relacionados a la optimización numérica, a aquellos que definen los modelos de aprendizaje y a aquellos correspondientes al problema de exploración. En este trabajo se utilizan en la mayoría de los casos, aquellos parámetros comúnmente recomendados en la literatura, modificando únicamente aquellos relacionados al diseño de los modelos de aprendizaje, con el objetivo de estudiar la capacidad de generalización de dichos valores, lo que permite validar su aplicación o transferencia a otros dominios.

Los experimentos realizados en las secciones anteriores para los agentes Actor-Crítico Lineal, *SARSA* y *DDPG* permiten validar la generabilidad del uso de los parámetros recomendados en la literatura. En particular, de las Figuras 4.1, 4.2 y 4.8 se puede observar que en términos de índices de rendimiento, los 3 agentes evaluados son capaces de resolver el problema, y con niveles similares de rendimiento.

Tal como se menciona anteriormente, el principal foco de estudio en los experimentos consiste en el análisis de los parámetros de los modelos de aprendizaje, estudiando el efecto de estos parámetros en su complejidad computacional y el proceso de convergencia (valor asintótico y velocidad de convergencia).

En el caso de Actor-Crítico lineal, a partir de la Figura 4.1 se puede observar que una baja cantidad de funciones base por dimensión del espacio de estados limita el rendimiento del sistema, lo que se se le puede adjudicar a la baja expresividad de la representación. Otro fenómeno producto de las cantidades de funciones base *RBF* consiste en la velocidad de convergencia. Si bien para una baja cantidad de funciones base la velocidad de convergencia es alta, a medida que se aumenta dicha cantidad, la velocidad de convergencia disminuye drásticamente. En este caso, la razón detrás de este fenómeno consiste en que debido a la naturaleza de la representación, donde cada parámetro está asociado a una zona del espacio de estados fija, una baja cantidad de funciones base favorece la generalización (a coste de expresividad), mientras que una alta cantidad de funciones base favorece la expresividad del modelo, sacrificando generalización. Este compromiso entre generalización y expresividad se manifiesta además en la alta variabilidad en las curvas de recompensa y rendimiento, donde para un alto número de funciones base, el agente presenta excelentes desempeños para ciertas zonas de operación, pero pobres desempeños para otras.

El análisis correspondiente al caso de *SARSA* es similar al caso de Actor-Crítico lineal en cuando comparten la misma representación del espacio de estados mediante funciones *RBF*. Al variar el número de funciones base, proceso observado en las Figuras 4.2 y 4.3, se observa nuevamente que el uso de un número alto de funciones base retrasa el aprendizaje, a la vez que produce una mayor varianza en los desempeños. No obstante a lo anterior, a diferencia del caso de Actor-Crítico Lineal, el uso de 5 funciones base, que en algoritmo anterior presenta pobres desempeños, en el caso de *SARSA* el uso de 5 funciones base presenta igual comportamiento asintótico que el resto de los experimentos. En este caso, una de las razones

de este fenómeno consiste en que a diferencia del agente Actor-Crítico, donde para cada zona de operación (centro de una *RBF*) existe un parámetro, en el caso de *SARSA* se asignan n_a parámetros, correspondientes a la cantidad de acciones discretas posibles, en este caso igual a 11. Un análisis propio de *SARSA* al corresponder al único algoritmo de acciones discretas utilizado, corresponde al efecto de discretizar el espacio de acciones, fenómeno estudiado en los experimentos de las Figuras 4.4 y 4.5. En este caso se puede apreciar que en el rendimiento final del sistema, no existen mayores diferencias entre utilizar 6, 11 o 16 acciones por dimensión del espacio de acciones. En cambio, si existen notorios efectos en la tasa de convergencia, los cuales son principalmente atribuidos al fenómeno de la exploración, donde un número alto de acciones posibles dificulta visitar todas las acciones frecuentemente. De estos resultados se puede extraer que pese a que *Ball Dribbling* consiste naturalmente en un problema de acciones continuas, es posible obtener buenos rendimientos utilizando una discretización del problema, lo que identifica que no es necesario un control realmente fino en este problema. Otro aspecto de interés es considerar los beneficios de la metodología descentralizada, pues se puede observar un retraso en el proceso de convergencia al aumentar de 6 a 16 acciones, efecto que se acentúa en caso de no utilizar la metodología descentralizada, en cuyo caso el número de acciones correspondería a $16 \times 16 \times 16 = 4.096$ para poder representar 16 acciones por dimensión.

El último de lo agentes a analizar corresponde a *DDPG*, donde se presentan resultados de variar el número de neuronas de las capas ocultas (número de parámetros), tipo de delimitación del espacio de acciones, y tamaño del *minibatch*. Los resultados de la variación de neuronas en las capas ocultas puede observarse en la Figura 4.8 donde se observan dos fenómenos de importancia.

En primer lugar, un uso bajo de neuronas en las capas ocultas no permite resolver de manera correcta el problema, pese a poseer una cantidad de parámetros comparable o superior a otros agentes clásicos que utilizan representaciones *RBF*. En este caso una posible razón de este fenómeno consiste en la naturaleza de los parámetros. En el caso los parámetros de la representación *RBF*, estos son ponderaciones de características no lineales con respecto al estado, lo que le asigna un poder de representación base. En cambio, en el caso de las redes neuronales *feedforward*, se suele decir que las características de interés deben ser aprendidas por el sistema a partir de las entradas brutas (en este contexto las variables de estado), proceso que utiliza los mismos parámetros de aprendizaje, por lo que un número bajo de neuronas no permite obtener una buena representación. En resumen, la utilización de representaciones *RBF* involucra un conocimiento previo y estructuras preestablecidas que facilita el aprendizaje, mientras que las redes neuronales multi-capa deben aprender una estructura que permita resolver el problema.

El segundo fenómeno de interés con respecto al número de neuronas corresponde a la posterior invarianza en el proceso de aprendizaje mientras se aumenta el número de neuronas, a diferencia de los algoritmos anteriores, donde aumentar el número de parámetros involucra una disminución en la velocidad de aprendizaje. En este caso, se le puede atribuir este fenómeno a la estructura de la red neuronal, donde el gradiente tiene el potencial de distribuirse sobre una mayor cantidad de parámetros, a diferencia de la representación *RBF*, donde la mayor parte del gradiente se enfoca sobre una cantidad ínfima de parámetros.

Un caso de especial atención corresponde a los experimentos realizados con respecto al

tamaño del *minibatch* de *DDPG* en la Figura 4.9. Se puede observar como el disminuir el tamaño del *minibatch* en un amplio rango de valores, no disminuye la velocidad de convergencia, ni su valor asintótico. Las causas de este fenómeno principalmente pueden ser atribuidas a que en presencia de una baja dimensionalidad, las estimaciones del gradiente pueden ser realizadas con pocos datos sin perder precisión, y que *Adam*, que es precisamente un método de optimización estocástico, es robusto a estimaciones del gradiente con una alta varianza. Finalmente, un hecho a considerar con respecto al tamaño del *minibatch* corresponde al caso límite, es decir cuando el *minibatch* es de tamaño unitario. En este caso se observa un aprendizaje con una menor velocidad de convergencia, el cual sin embargo tiene el mismo valor asintótico que el presentado para otros tamaños de *minibatch*. Se hace énfasis en que pese a corresponder a estimaciones de gradientes con una única experiencia, no se trata de un aprendizaje en línea como el presente en algoritmos tradicionales como *SARSA* y Actor-Crítico Lineal, sino que dicha experiencia es obtenida aleatoriamente desde el *Replay Memory*, rompiendo las correlaciones temporales, que es el principal fundamento de la técnica *Experience Replay*. Una posible consecuencia de la robustez del algoritmo a tamaños bajos de *minibatch*, es la posibilidad de usar largas cadenas temporales, para presupuestos computacionales limitados (tamaño total del *minibatch*), lo que facilita el entrenamiento de agentes utilizando redes recurrentes [11].

Otro fenómeno estudiado para el caso de *DDPG* corresponde al uso de mecanismos de delimitación de los valores de la política presentados en [52]. Los resultados de la utilización de los 3 mecanismos planteados puede observarse en la Figura 4.10, donde se puede identificar que solo el método *zeroing gradients* obtiene un desempeño diferente, en este caso inferior. Sin embargo, de las simulaciones realizadas, se observa que en algunos casos, el método tradicional, en este contexto denominado *squashing gradients* alcanza una política saturada, tal como se presenta en la Figura 4.11.a, donde la saturación del actor, condiciona las futuras experiencias, y en algunos casos estanca de manera permanente el proceso de aprendizaje y la política.

Aprendizaje de la Política

Los rendimientos alcanzados por un sistema de aprendizaje no son los únicos factores de interés al momento de elegir una metodología determinada. En el caso del Aprendizaje Reforzado cobra especial importancia el proceso de evolución de la política del agente, debido a que determina en gran medida su aplicabilidad en sistemas físicos, debido al peligro inherente de la exploración y por los posibles riesgos de desplegar un sistema que presente rendimientos fuertemente desiguales en distintas zonas de operación.

En la Figura 4.11 se presentan algunos ejemplos de episodios del proceso de aprendizaje realizado por el algoritmo *DDPG*, representados mediante estelas que corresponden a los caminos seguidos por el robot y la pelota. En el caso del algoritmo *DDPG*, su carácter de Actor-Crítico que utiliza acciones continuas, determina en gran medida la naturaleza del aprendizaje. En primer lugar, debido a que se tiene una representación independiente de la política, los efectos del aprendizaje de la política repercuten de manera suave sobre las estimaciones del crítico. Adicionalmente, el uso de acciones continuas garantiza en gran medida la correlación temporal de las acciones (en ausencia de exploración), asumiendo la misma

propiedad en el espacio de estados. En el caso particular de *DDPG* los episodios iniciales tienen como principal naturaleza la correspondiente a la política inicial o nula, superpuesta por la naturaleza de la exploración. Sin embargo, debido al uso de redes neuronales como aproximadores, y a unas estimaciones iniciales deficientes del crítico, es común encontrar situaciones como la presentada en la Figura 4.11.a, en la cual se puede apreciar una saturación de la política. Estas situaciones suelen recuperarse mediante la apropiada exploración, pero en caso de utilizar *squashing gradients* es posible que la política no se recupere, y se mantenga en saturación por el resto del entrenamiento. En la Figura 4.11.b se puede apreciar que el agente adquiere un control de la pelota, el cual sin embargo no es lo suficientemente fino como para poder llevar la pelota hacia su destino. En episodios posteriores, como el presentado en la Figura 4.11.c se puede observar como tras un período inicial de control de la pelota, el agente logra mantener un control adecuado y lleva la pelota sin problemas hacia el objetivo. Finalmente, en episodios terminales, como el presentado en la Figura 4.11.c se puede apreciar como se realizan comportamientos altamente no lineales, que le permiten al agente rápidamente entrar en control de la pelota, minimizando el criterio de *faultRate* en iteraciones iniciales del episodio, y llevando eficazmente la pelota a su objetivo.

En segundo lugar, en la Figura 4.12 se presentan episodios representativos del proceso de aprendizaje de *SARSA*, propio de los algoritmos de crítico y acciones discretas. En este caso, debido a que la política se deriva de las estimaciones del crítico, presenta un carácter inestable e inconsistente durante episodios tempranos. Adicionalmente, debido a la naturaleza de la exploración, que en este caso no es aditiva, se puede explorar de mejor manera el espacio de acciones, pero al mismo tiempo la labor del crítico es más compleja. Finalmente, la propia naturaleza de las acciones discretas permite generar comportamientos complejos y con poca correlación temporal entre las acciones. En el caso del uso de *SARSA* en el problema de *Ball Dribbling* se puede observar a partir de la Figura 4.12.a, que en episodios tempranos el agente aprende a interactuar con la pelota, pero no logra adquirir un control de la misma. A medida que transcurren episodios, el agente permite controlar la pelota por períodos más largos de tiempo, como se presenta en la Figura 4.12.b, donde sin embargo, al perder el control de la pelota, no se puede recuperar. Finalmente, en episodios más avanzados, como el presente en la Figura 4.12.c, se puede observar como el agente logra obtener un control de la pelota, y matenerlo hasta llegar al objetivo. Sin embargo, incluso en episodios finales, tal como el que se representa en la Figura 4.12.d, existen zonas de operación donde no se presenta una política consolidada, la cual sin embargo al llegar a un punto de operación aprendido, si vuelve a adquirir el control de la pelota.

Otro aspecto a considerar con respecto a la política, además de observar sus efectos en el sistema global, corresponde al análisis presentado en la Sección 4.1.5, donde se plantea la importancia de observar la señales temporales de la formulación de estado y acción que representan de manera conjunta la política.

En el caso de *DDPG*, una secuencia temporal de las señales anteriormente mencionadas puede encontrarse en la Figura 4.13, donde se escoge dicha ventana de tiempo una vez que el agente logra controlar inicialmente la pelota (régimen estacionario). Un aspecto que se destaca de esta representación de la política consiste en el control que el agente realiza sobre la distancia de la pelota, presentado en la Figura 4.13.a, mediante la secuencia temporal de ρ. En este caso el agente mantiene un control de la pelota por debajo del nivel correspondiente al valor de *fault*, para lo cual regula la velocidad frontal V_x, en instantes precisos de tiempo,

tal como puede presentarse en la Figura 4.13.b. Otro fenómeno de interés corresponde al control sobre los valores de estado γ y ϕ, los cuales si bien permanecen fuera de los rangos de *fault*, permanecen cercanos al borde. Esta situación se debe a la formulación de la recompensa establecida en la ecuación 3.7, en la cual no se penalizan los valores de γ y ϕ, a menos que se encuentre en situación de *fault*. Finalmente, destacan las secuencias de acciones del agente, las cuales si bien pueden llegar a saturar, como en las Figuras 4.13.b y 4.13.d, dicho fenómeno solo sucede temporalmente, lo cual en este caso es debido al uso de *inverting gradients*.

Por otro lado, la Figura 4.14 presenta el mapeo de la política para el caso del agente *SARSA*. En este caso se destaca la relación entre las Figuras 4.14.a y 4.14.b, en las cuales las acciones de control de V_x generan una situación óptima en terminos de índices de rendimiento, pues mantiene $\rho \leq \rho_{fault}$ en condición crítica, tal que $\rho_{max} = \rho_{fault}$, mientras utiliza la velocidad más alta posible. Adicionalmente a este fenómeno se destaca que debido a la formulación de la recompensa descentralizada de las ecuaciones 3.5 y 3.5, si existe un control efectivo sobre las variables de estado angulares incluso en la zona fuera de *fault*. Finalmente es importante destacar la naturaleza de las acciones de control, especialmente en los agentes relacionados a V_y y V_θ, las cuales son fuertemente discontinuas, y si bien no presentan valores saturados, si consisten en acciones de control *bang-bang*, las cuales en sistemas físicos como robots pueden generar daños a los actuadores, además de presentar inconsistencias entre el rendimiento simulado y el físico.

Comparación entre Agentes

Los análisis realizados en las secciones anteriores buscan destacar las principales ventajas y desventajas de cada método con el objetivo de presentar guias generales al momento de la elección de la herramienta a utilizar en problemas de naturaleza similar.

A partir de los resultados presentados en las Secciones 4.1.1, 4.1.2 y 4.1.3, no se puede obtener conclusiones claras con respecto a que agente se desempeña mejor para este problema en términos de rendimiento final. Por otro lado, con respecto a la eficiencia de datos (en este contexto la velocidad de convergencia), el algoritmo Actor-Crítico lineal presenta un mejor desempeño, lo cual sin embargo no permite concluir su superioridad respecto a *DDPG*, debido a que en dicho algoritmo se utiliza una tasa de aprendizaje fuertemente inferior a la utilizada en Actor-Crítico lineal. Finalmente, la existencia de una alta cantidad de hiper parámetros no estudiados impide obtener una conclusión clara o definitoria en términos globales de aprendizaje, y la elección de la herramienta debe guiarse por otros criterios relacionados con la naturaleza de cada algoritmo.

Otro punto de comparación corresponde al despliegue de la política en sistemas físicos o limitados. En este sentido, para este tipo de problemas de baja dimensionalidad, los 3 agentes logran buenos desempeños utilizando tiempos de ejecución cercanos a $0,5\,[ms]$, lo que les permite desempeñarse en la mayoría de las plataformas y aplicaciones. En este sentido, pese a que *DDPG* utiliza métodos complejos y considerables recursos computacionales durante su entrenamiento, su ejecución en *test* comprende solo la multiplicación y adición de matrices, lo que le permite obtener tiempos de ejecución acotados. Adicionalmente, cabe destacar que en el caso de algoritmos de naturaleza Actor-Crítico, en tiempo de ejecución no se utiliza la estructura del crítico y solo se ejecuta la política (actor), evitando el uso de cálculos

innecesarios. Finalmente, cabe destacar que debido a que la representación *RBF* presenta fuertemente la maldición de la dimensionalidad, escalando exponencialmente su complejidad con la dimensionalidad del espacio de estados, 3 dimensiones puede ser considerado como un punto de inflexión a partir del cual es conveniente el uso de otras representaciones como por ejemplo las redes neuronales *feed-forward*.

En términos de modelamiento del problema, el uso del Aprendizaje Reforzado Descentralizado permite facilitar el proceso de diseño de la recompensa, lo que se puede apreciar al comparar la ecuación 3.7 con las ecuaciones 3.4, 3.5 y 3.6, donde resulta sencillo identificar la labor de cada agente, lo que además produce rendimientos similares a diseños centralizados de la recompensa.

El último punto a comprar, y el que presenta un carácter más concluyente, corresponde a la naturaleza de la política resultante. Tal como se puede apreciar de las Figuras 4.13 y 4.14, el uso de acciones discretas genera acciones de control del estilo *bang-bang* las cuales suele ser dañinas para sistemas físicos, mientras que el uso de acciones continuas genera acciones con una menor tasa de cambio en situaciones comparables. Finalmente, el uso de representación *RBF*, debido a la localidad de sus bases, limita la generalización del aprendizaje, presentando diferencias abrumadoras en el rendimiento para disintas zonas de operación. En éste último punto, el uso de redes neuronales *feed-forward* mediante técnicas de *Experience Replay* permiten un aprendizaje más general, obteniendo en la práctica rendimientos comparables en las distintas zonas de operación.

4.2. In-walk Kick

Tal como se discute en la Sección 3.2.4, el modelamiento del problema *In-walk kick* presenta fuertes similitudes con el problema *Ball Dribbling*, siendo la principal diferencia que *In-walk Kick* posee una naturaleza fuertemente episódica, en la cual el desempeño general depende de una única interacción entre el robot y la pelota.

Debido a esta gran similitud, y al objetivo del problema, en esta sección se omite el análisis comparativo entre las diferentes implementaciones de Aprendizaje Reforzado realizado para el problema de *Ball Dribbling*, y se enfoca el estudio en el modelo extendido presentado en la Sección 3.2.4, cuyo objetivo es mejorar el rendimiento del agente, mediante una diseño consciente del resto de los módulos presentes en el sistema (en este caso la locomoción del robot). Adicionalmente, en esta sección se valida el uso de las funciones de soporte finito (*FSBF*), una alternativa a la representación lineal *RBF* que permite reducir fuertemente los costos computacionales.

Para los experimentos relacionados a *In-walk Kick* se utiliza el agente *SARSA* mediante la formulación del Aprendizaje Reforzado Descentralizado como herramienta para poder abordar múltiples dimensiones en el estado de acciones. En este caso, los rangos de valores del espacio de estado y acción se presentan en las Tablas 4.16 y 4.17, donde los valores de los rangos del espacio de estado son más acotados, buscando enfocar la solución en un conjunto más reducidos de zonas de operación, y con rangos más altos en el espacio de acciones, dado que se buscan movimientos bruscos de locomoción, con el objetivo de crear fuertes movimientos de patada.

Tabla 4.16: Rangos del espacio de estados para el problema *In-walk Kick*

	Valor mínimo	Valor máximo
ρ_{center}	$0\,[mm]$	$800\,[mm]$
γ_{center}	$-70°$	$70°$
ϕ_{center}	$-90°$	$90°$
fase	-1	1

Tabla 4.17: Rangos del espacio de acciones para el problema *In-walk Kick*

	Valor mínimo	Valor máximo
V_x	$0\,[mm/s]$	$120\,[mm/s]$
V_y	$-70\,[mm/s]$	$70\,[mm/s]$
V_θ	$-0,52\,[rad/s]$	$0,52\,[rad/s]$

Adicionalmente, en la Tabla 4.18 se presentan los parámetros del agente en su formulación descentralizada, además de los parámetros de la aproximación funcional. Por otro lado, en la Tabla 4.19 se presentan los parámetros de exploración para una política $\epsilon - greedy$, y finalmente en la Tabla 4.20 se presentan las condiciones de entrenamiento y los parámetros de la función de recompensa, donde cabe destacar que se omiten las dimensiones físicas del

entorno de aprendizaje, debido a que se utiliza el mismo ambiente que en el caso de *Ball Dribbling*.

Tabla 4.18: Parámetros de *SARSA* para el problema *In-walk Kick*

Parámetro	Valor
α	$0,1$
γ	$0,99$
λ	$0,9$
Función base	RBF
RBF_ρ	15
RBF_γ	11
RBF_ϕ	13
σ_{RBF}	$0,5\triangle$
$acciones_x$	15
$acciones_y$	11
$acciones_\theta$	13

Tabla 4.19: Parámetros de exploración $\epsilon - greedy$ para el problema de *In-walk kick*

Parámetro	Valor
ϵ_0	1
ϵ_{dec}	$0,2$
ϵ_{final}	$0,05$
ϵ_{min}	0
Decaimiento	Exponencial

Tabla 4.20: Parámetros de entrenamiento generales para el problema *In-walk Kick*

Parámetro	Valor
Episodios	1.500
T_{max}	200 $[s]$
γ_{init}	0
$\phi_{init-min}$	$-45°$
$\phi_{init-max}$	$45°$
ρ_{init}	1.000 $[mm]$
$dist_{pelota-arco}$	1.500 $[mm]$
K	50
φ_0	300 $[mm]$
$alpha_0$	$14°$

4.2.1. Comparación de modelos

En los gráficos de la Figura 4.15 se presentan los índices de rendimiento introducidos en la Sección 3.2.4.2 para el problema *In-walk Kick*. Los gráficos corresponde al resultado de 15

Tabla 4.21: Tiempos de ejecución para el problema *In-walk Kick*

	Tiempo de ejecución $[ms]$
RBF	4.42
FSBF	0.17

simulaciones para cada caso, para las cuales se presenta en línea sólida la media y las barras de error representan el error estandar. Se utiliza el error estandar en vez de la desviación estandar, pues el objetivo de este experimento no consiste en evaluar la variabilidad del proceso de aprendizaje y las condiciones iniciales, sonó poder determinar cual de los dos modelos propuestos permite alcanzar mejores resultados a nivel de rendimiento.

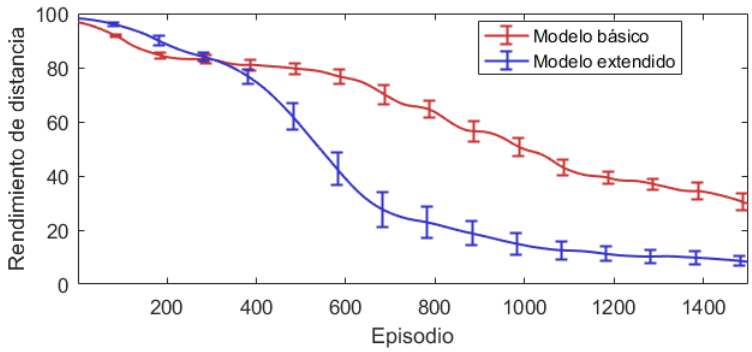
4.2.2. Comparación de funciones base

Además de los experimentos realizados en la Sección 4.2.1 correspondientes a la comparación entre distintos modelamientos del problema *In-walk Kick*, en esta sección se evalúa el uso de funciones de soporte finito como alternativas a la representación *RBF* como aproximador lineal, identificando sus efectos tanto en el rendimiento del agente, como en su efecto en los tiempos de ejecución. En la Figura 4.16 se presentan los distintos índices de rendimiento, utilizando las distintas funciones de soporte finito presentadas en la Figura 4.17, utilizando en todos los casos el modelo extendido para el problema *In-walk Kick*.

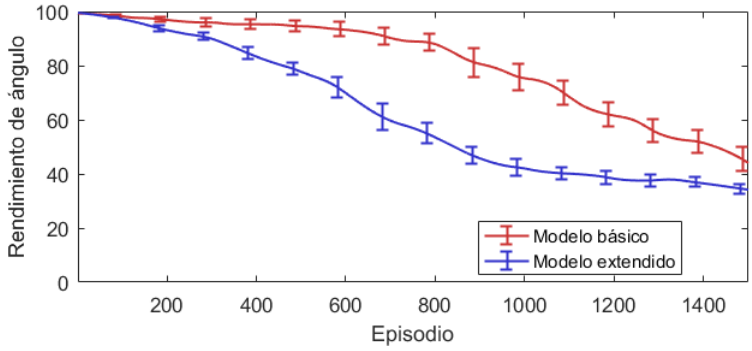
De manera complementaria a los índices de rendimiento del agente, en la Tabla 4.21 se presentan los tiempos de ejecución tanto para la aproximación mediante *RBF* como para la aproximación mediante funciones base de soporte finito (*FSBF*). Para realizar una evaluación entre representaciones comparables, se utiliza en ambos casos una función *Gaussiana* con el mismo valor de σ, pero donde la versión *FSBF* corta la función en 3σ, obteniendo efectivamente una función con soporte finito. Los tiempos de ejecución son medidos directamente en el robot *NAO* que posee limitados recursos computacionales, y de manera integrada con el resto de los sistemas del robot (comparte tiempo de procesamiento del único núcleo). Los tiempos corresponden al tiempo de reloj, y se destaca que para distintas funciones base de soporte finito, pero con un mismo ancho de soporte, sus tiempos de ejecución son idénticos.

4.2.3. Análisis

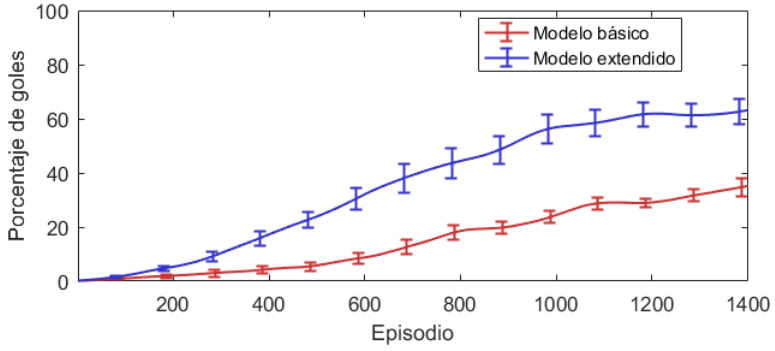
A nivel general, los resultados de las Secciones 4.2.1 y 4.2.2 permiten validar el proceso de diseño y formulación del problema *In-Walk Kick*, al alcanzar altos porcentajes de gol, bajo condiciones adversas. Destaca en este sentido, que el sistema de aprendizaje logra resolver un problema complejo, en el cual se desea reforzar un comportamiento específico, difícil de modelar, y en el cual el agente debe aprender sin mayores guías. En este mismo sentido, cobra importancia la labor del aprendizaje por diferencias temporales, que a partir de patadas exitosas, permite de manera correcta, reforzar las acciones pasadas que le permitieron generar dicho comportamiento final, especialmente debido al uso de $TD(\lambda)$. Finalmente cabe recalcar la importancia de la exitosa labor de la exploración en este problema, pues debido a la



(a)

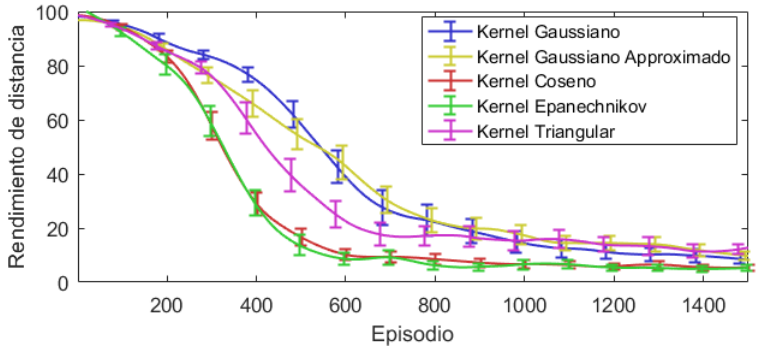


(b)

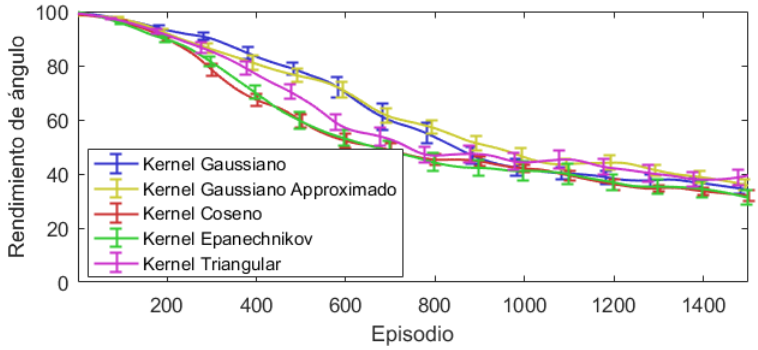


(c)

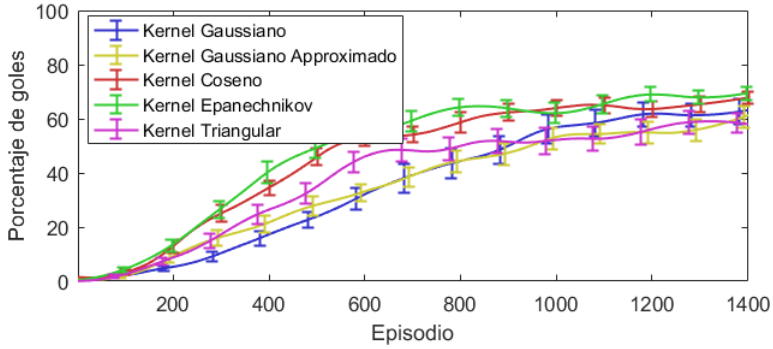
Figura 4.15: Rendimiento del agente para los dos modelos planteados para el problema *In-walk Kick*



(a)



(b)



(c)

Figura 4.16: Rendimiento del agente utilizando distintas funciones base para el problema *In-walk Kick*

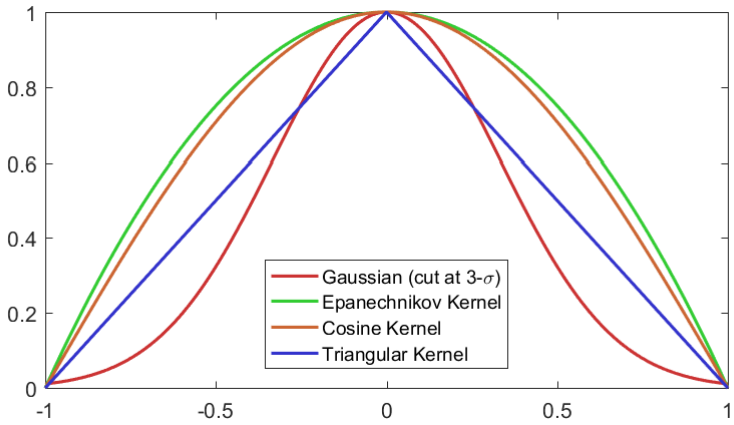


Figura 4.17: *Kernels* utilizados como funciones base de soporte finito para aproximación funcional

ausencia de una mayor guia de la recompensa durante iteraciones previas a la patada, es compleja la realización de las secuencias de acciones que permiten generar una patada, por lo cual la mayoría de los episodios iniciales terminan sin acceso a una recompensa positiva.

Los resultados de la comparación de los modelos básico y extendido presentados en la Sección 4.2.1 permiten validar el funcionamiento del modelo extendido. En particular, en la Figura 4.15 se puede observar que un mejor diseño consciente de la existencia de otros sistemas permite alcanzar aproximadamente un 50 % más de goles con respecto a un diseño ingenuo, lo cual se debe matemáticamente a que el proceso de decisiones se acerca de mejor manera a la propiedad de *Markov* (mejora en la observabilidad del proceso). Este resultado permite identificar la necesidad de enfocar fuertemente el proceso de diseño de la solución en el modelamiento del problema, adicionalmente al diseño del agente. En particular, la identificación de un conjunto de variables que permitan aproximar de manera correcta una formulación de estados tiene fuertes efectos en el rendimientos del sistema.

El último de los experimentos realizados consiste en la evaluación de distintas bases para la aproximación funcional lineal presentados en la Sección 4.2.2. De estos experimentos se destaca el similar desempeño de las bases *Gaussiana* y *Gaussiana* aproximada, que justifica la hipótesis de las *FSBF*, que establece que debido a la ínfima ponderación de la mayoría de las bases al resultado de una aproximación linea, es factible utilizar funciones de soporte finito, sin perder rendimiento. Otro fenómeno de interés es que las *FSBF* no solo presentan un rendimiento similar a las *RBF Gaussianas*, sino que en en algunos caso como las funciones base *Epapnechnikov* y coseno, su rendimiento es marcadamente mayor. Finalmente cabe destacar que el uso de las *FSBF* tiene mayor relevancia en aquellas aplicaciones donde se desee utilizas control a altas frecuencias, por ejemplo sobre los 500 $[Hz]$, debido al *speedup* producido por esta representación en comparación con la representación tradicional, tal como se puede evidenciar en la Tabla 4.21.

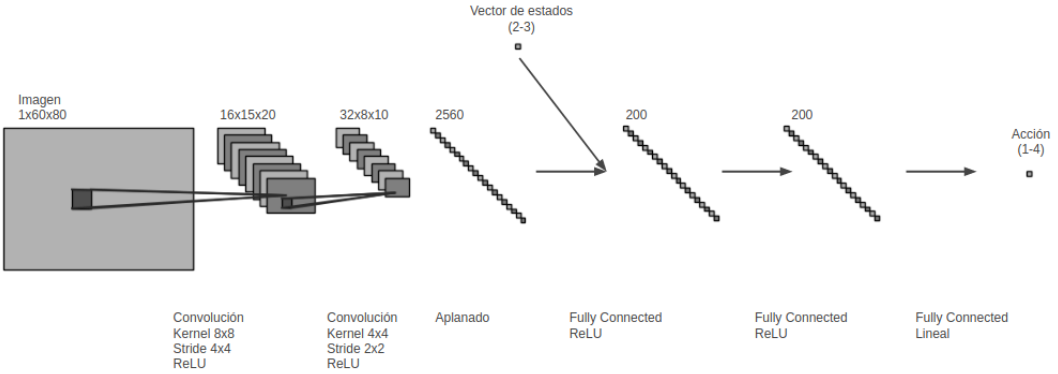
4.3. Navegación visual

Tal como se menciona al comienzo del Capítulo 4, debido a que el problema de navegación visual corresponde al problema menos estudiado en la literatura, el enfoque de los experimentos realizados en esta sección se centra en el modelamiento del problema, y el grado de rendimiento que permite obtener la resolución mediante Aprendizaje Reforzado. Adicionalmente, debido a la naturaleza de alta dimensionalidad del problema, no es factible utilizar directamente métodos tradicionales, y en esta ocasión se consideran solo algoritmos de *Deep Reinforcement Learning*. Finalmente, de las herramientas presentadas en la Sección 2.3 solo se considera el uso del algoritmo *DDPG*, pues a diferencia de *Deep Q-Learning*, permite abordar problemas de acciones contínuas y multidimensionales, recordando que los modelos planteados en la Sección 3.2.5 utilizan entre 1 y 4 dimensiones para el espacio de acciones.

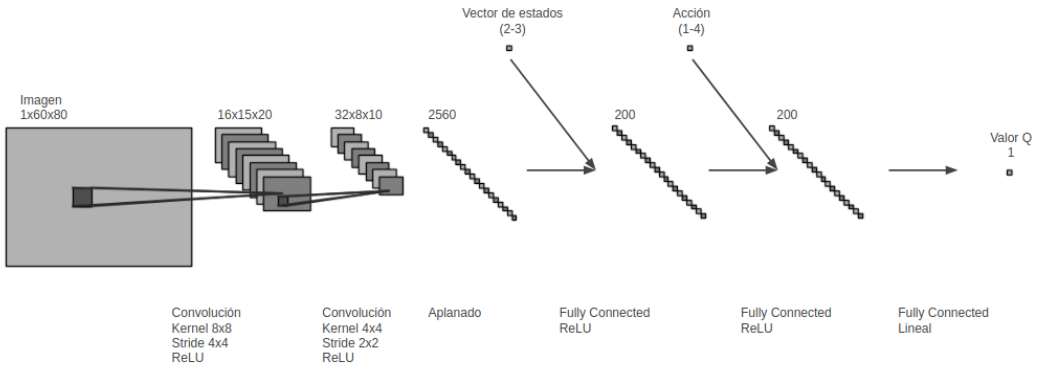
Las condiciones y el ambiente del entrenamiento son similares a las utilizadas para el caso de *Ball Dribbling*, en particular se asigna un tiempo máximo de $200\,[s]$ para cumplir la tarea de navegación, y las acciones del agente son realizadas cada $200\,[ms]$. La principal diferencia con respecto a las condiciones de entrenamiento con respecto a los casos anteriores, corresponde a que el episodio termina de manera temprana en el caso de la colisión del agente con otro robot, debido al objetivo de la navegación visual.

De manera análoga a lo desarrollado en la Sección 4.1.3, la utilización del algoritmo *DDPG* involucra la elección de numerosos parámetros de diseño. En primer lugar, las arquitecturas de las redes neuronales utilizadas tanto para el actor como para el crítico son presentadas en la Figura 4.18. Debido al uso de imagenes como pare de la formulación de estados, las primeras capas corresponden a capas convolucionales, que utilizan *kernels* de tamaño 8×8 y 4×4, además de reemplazar el uso de operadores *maxpool* en pos de *strides* mayores a 1, siguiendo las guías de diseño presentadas en los trabajos iniciales de [6] y [4]. Por otro lado, la elección del tamaño de la imagen se escoge de dimensiones similares a as presentados en trabajos anteriores, además de escoger dimensiones correspondientes a un submúltiplo de la resolución original (640×480), favoreciendo un submuestreo rápido y una alineación en memoria de los datos. Finalmente, como la formulación de estados considera el objetivo de navegación adicionalmente a la imagen obtenida de la cámara del robot, se integran estos valores como entradas adicionales a la primera capa *fully-connected* de la red neuronal utilizada como aproximador funcional, tal como se presenta en las Figuras 4.18.a y 4.18.b.

Para el problema de navegación, a diferencia del problema de *Ball Dribbling*, se busca generar una política con una planificación en un horizonte más largo en el tiempo. Por esta razón se utiliza el método de exploración mediante ruido *Gaussiano* con momento, que simula un proceso de *Ornstein-Uhlenbeck* y que permite realizar una exploración más compleja del espacio de estados. Los parámetro de exploración utilizados para este problema se presentan en la Tabla 4.22.



(a) Actor



(b) Crítico

Figura 4.18: Arquitectura de las redes neuronales utilizadas por *DDPG* para el problema de Navegación Visual. (a) corresponde a la estructura del actor y representa la política $\pi(s)$, mientras que (b) representa al crítico, que estima la función de valor $Q(s, a)$

Tabla 4.22: Parámetros de exploración *Gaussiana* con momento para Navegación visual

Parámetro	Valor
σ_0	$0,4$
σ_{dec}	$0,3$
σ_{final}	$0,1$
σ_{min}	$0,1$
θ	$0,15$
Decaimiento	Exponencial

En la Tabla 4.23 se presentan los parámetros relacionados a *DDPG*, donde el único cambio con respecto a los utilizados previamente para *Ball Dribbling*, corresponde al tamaño utilizado para el *Replay Memory*, el cual se escoge con un valor más acotado, debido a que en este caso las experiencias constituyen imágenes, las cuales consumen considerables recursos de memoria. Adicionalmente, se utiliza nuevamente el optimizador *Adam*, utilizando los mismos parámetros presentados anteriormente en la Tabla 4.11 para el problema *Ball Dribbling*.

Tabla 4.23: Parámetros de *DDPG* para el problema de Navegación Visual

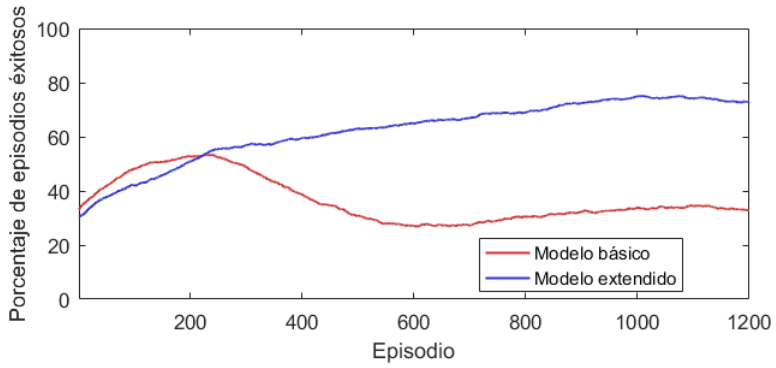
Parámetro	Valor
γ	$0,99$
Frecuencia de *batch*	1
Tamaño de *batch*	64
Capacidad del *Replay Memory*	20.000
τ	$0,001$

4.3.1. Comparación de modelos

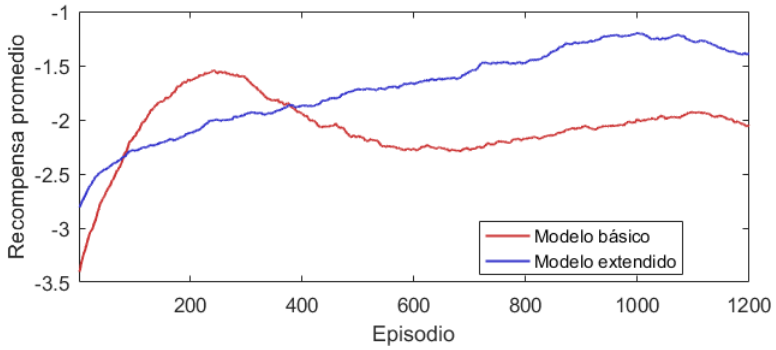
En esta sección se presentan los resultados del proceso de entrenamiento para el problema de Navegación visual, evaluando 2 de los modelos planteados en la Sección 3.2.5. El primero de ellos, en este caso denominado modelo básico, corresponde a la utilización únicamente de la componente de rotación de la caminata omnidireccional como espacio de acciones. Por otro lado, el modelo extendido considera el uso de las capacidades completas de la caminata omnidireccional del robot (3 ejes de velocidad), además de permitir un control del ángulo *pan* de la cabeza del robot (movimiento horizontal). Adicionalmente, en el modelo extendido, se hace uso de redes recurrentes *LSTM* para realizar la labor de integración de temporal, abordando el problema de la observabilidad parcial, lo cual se realiza añadiendo una capa de celdas *LSTM* de manera posterior a la primera capa *fully connected*.

En el caso del modelo extendido, el proceso de entrenamiento y muestreo de experiencias desde la base de datos *Experience Replay* se modifica siguiendo el proceso diseñado en [11] para el correcto entrenamiento de redes recurrentes en Aprendizaje Reforzado. En los experimentos realizados se utilizan trazas de experiencia de largo 16, actualizando las redes utilizando únicamente los últimos 4 elementos de cada traza.

En las Figuras 4.19.a y 4.19.b se presentan los resultados obtenidos de las simulaciones



(a)

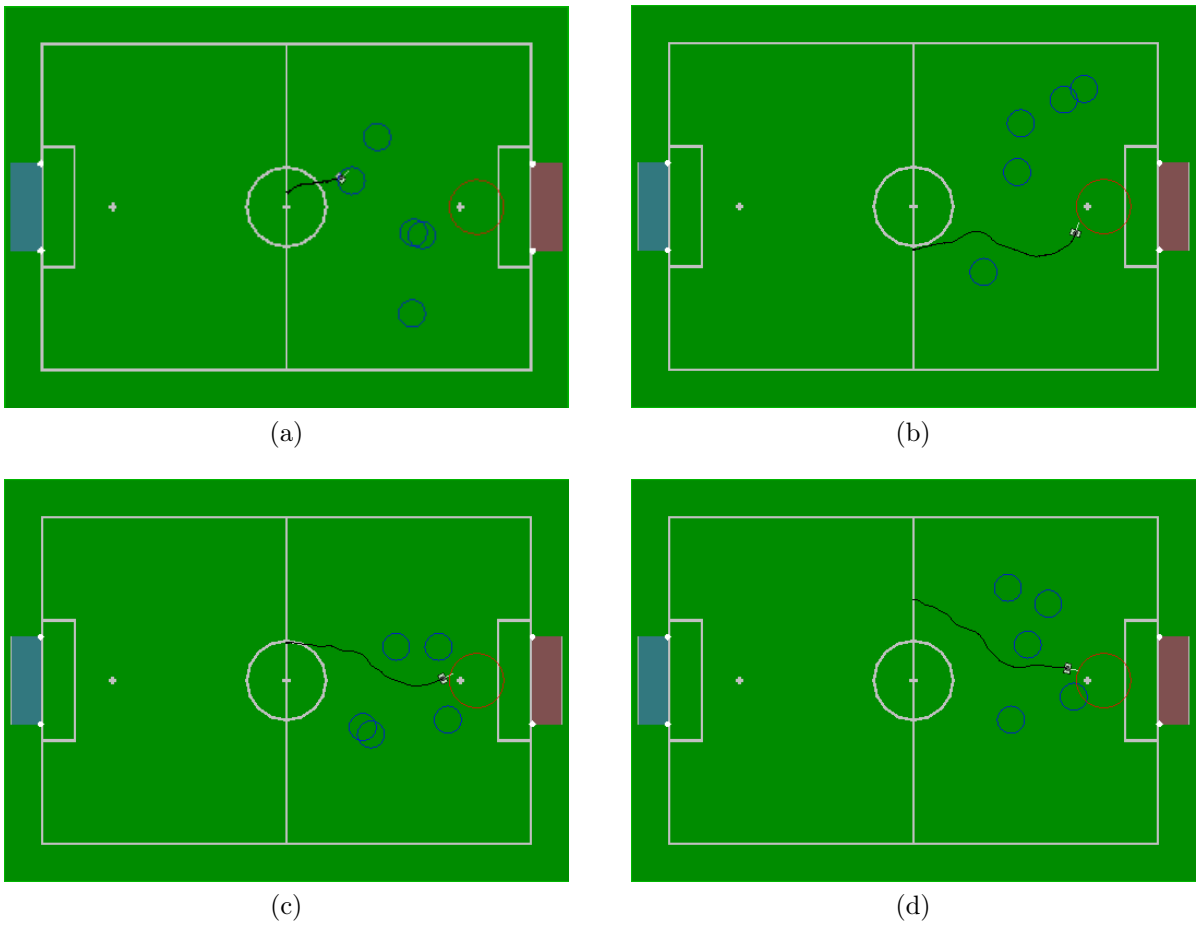


(b)

Figura 4.19: Comparación de los resultados del entrenamiento de los modelos básicos y extendidos para el problema de Navegación visual

realizadas correspondientes al porcentaje de episodios exitosos y a la recompensa promedio por episodio respectivamente. En este caso se considera exitoso un episodio en el cual se llega a la posición objetivo sin haber incurrido en ninguna colisión con alguno de los obstáculos presentes, los cuales son ubicados aleatoriamente al comienzo de cada episodio. Finalmente, los resultados de la Figura 4.19 corresponden al promedio de los resultados de 5 simulaciones independientes de 1.500 episodios, los cuales además son suavizados mediante un promedio móvil con un ancho de 300 episodios, con el objetivo de poder observar la evolución del aprendizaje, el cual resulta complejo de evaluar directamente debido a la alta diferencia de dificultad entre las condiciones de cada episodio.

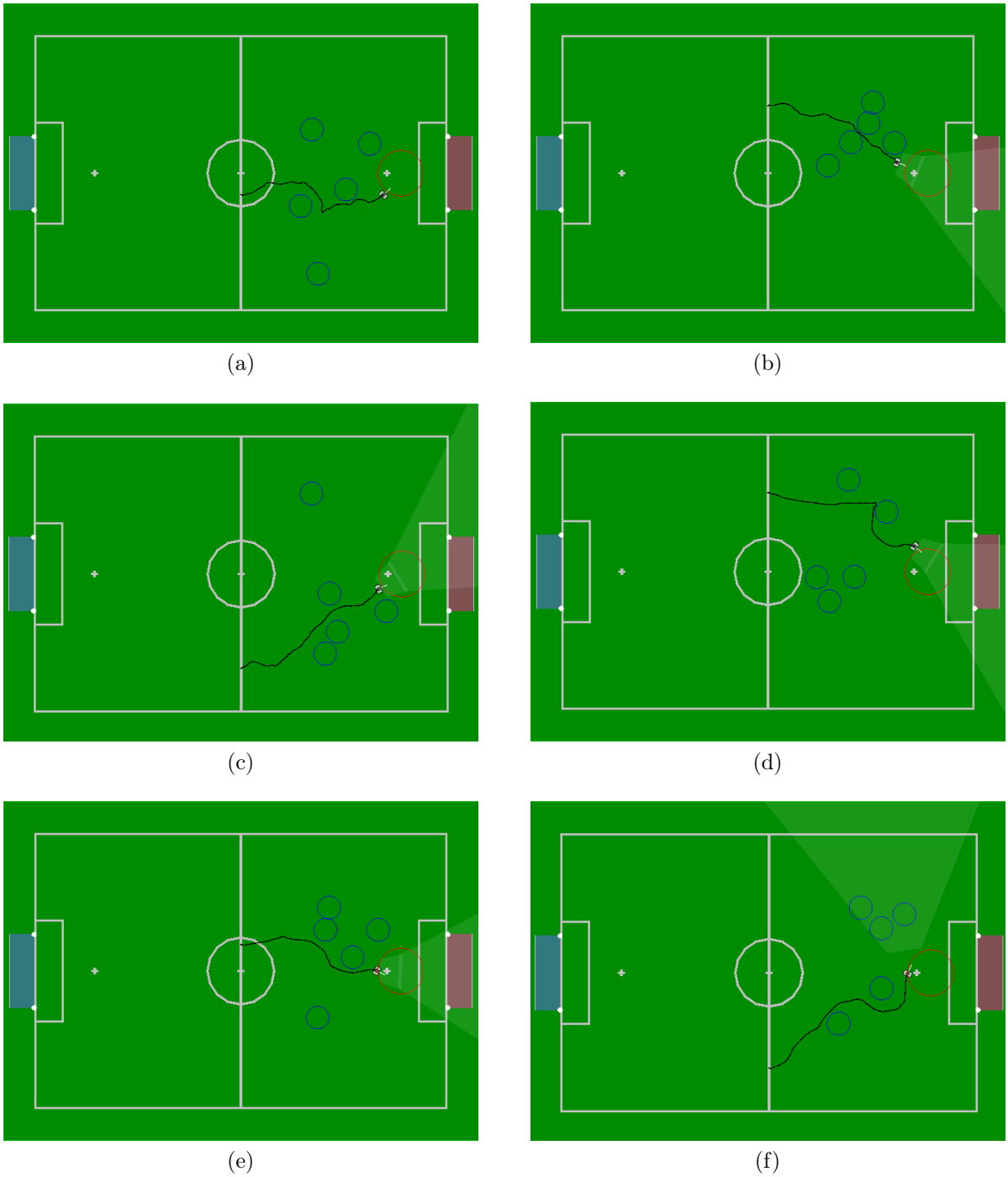
Finalmente, en las Figuras 4.20 y 4.21 se presentan ejemplos de episodios para los modelos básico y extendido. En cada figura, el episodio se representa mediante el camino seguido por el robot (curva negra), a la vez que se marca con circunferencias azules la posición de los obstáculos y con una circunferencias rojo la posición objetivo. En el caso del modelo extendido, se presenta además mediante un cono originado en la posición del robot, el campo de visión de las cámaras del robot, lo que resulta relevante debido a la capacidad del robot, a mover su cabeza con el objetivo de adquirir mayor información del ambiente. Es importante destacar que se presentan para cada modelo tanto casos de episodios exitosos, como de episodios que terminan de manera temprana debido a colisiones.



(a) (b)

(c) (d)

Figura 4.20: Ejemplos de episodios para el modelo básico de Navegación visual



(a) (b) (c) (d) (e) (f)

Figura 4.21: Ejemplos de episodios para el modelo completo de Navegación visual utilizando redes recurrentes

4.3.2. Análisis

Los resultados de la Sección 4.3.1 permiten validar el uso de *Deeo Reinforcement Learning* en aplicaciones robótica, haciendo uso directamente de información visual. Adicionalmente, es importante destacar la capacidad de derivar políticas capaces de realizar labores de navegación con buenos desempeños directamente a partir de imágenes, sin acceso o estimación intermedia de sensores de profundidad, tal como se realiza en el trabajo de [20] y [21].

La evolución del rendimiento de lo largo del entrenamiento para los dos modelos considerados presentados en las Figuras 4.19.a y 4.19.b permite identificar que el modelo básico presenta una mayor velocidad de convergencia en episodios iniciales con respecto al modelo extendido, lo que se le puede atribuir al hecho de que la exploración de los espacios de estado y acción resultan más sencillos, al solo poder actuar sobre una sola componente o dimensión del controlador de velocidad de la caminata del robot. Sin embargo, pese a presentar un mejor desempeño inicial, el modelo básico se ve prontamente estancado, e incluso disminuye su rendimiento, lo cual se le puede atribuir a la dificultad de realizar la labor de asignación de crédito a las distintas experiencias, al existir fuertes problemas de observabilidad en este modelo. Por otro lado, el modelo extendido presenta una evolución estable y una mejora monótona a lo largo de los episodios, sobrepasando prontamente el rendimiento del modelo básico y llegando a obtener aproximadamente el doble de episodios exitosos que dicho modelo, lo que se puede justificar debido al hecho de poder ejecutar acciones mucho más complejas debido al uso de los 3 ejes del controlador de velocidad, y a la capacidad de integrar observaciones temporalmente mediante el uso de redes recurrentes.

Adicionalmente al rendimiento numérico de los agentes, los resultados presentados en las Figuras 4.20 y 4.21 permiten analizar cualitativamente las políticas obtenidas por esta metodología. En la Figura 4.20 se puede apreciar como los caminos seguidos por el robot son de naturaleza sencilla, y el agente es capaz de planificar sus acciones al observar obstáculos a distancia. Sin embargo, debido a que solo se puede controlar la velocidad de rotación, y no existe una integración temporal de observaciones, se presentan numerosas situaciones donde el agente no puede evadir obstáculos, tanto por las limitaciones producto del espacio de acciones, como por la baja observabilidad del proceso. Por otro lado, la Figura 4.21 permite identificar varias de las características de la políticas resultantes del uso del modelo extendido. En primer lugar, tal como se puede apreciar en las Figuras 4.21.a y 4.21.d, el uso de la caminata omidireccional permite la generación de trayectorias complejas, lo que le permite adaptarse situaciones díficiles durante el entrenamiento. Adicionalmente, como se puede observar principalmente en las Figuras 4.21.a, 4.21.d y 4.21.f, el uso de redes recurrentes, permite en muchos casos actuar de manera correcta, incluso sin observar los elementos de interés. Ejemplos de este fenómeno consisten en la capacidad de recuperarse a situaciones inesperadas (Figura 4.21.d), y planificar la evasión de obstáculos, rodeándolos (Figuras 4.21.a y 4.21.f). Finalmente se destaca el enfoque que realiza el agente sobre los obstáculos, favoreciendo el seguimiento de la posición de los mismos (Figura 4.21.f), lo que permite identificar el correcto surgimiento de comportamiento asociados al concepto de visión activa.

Finalmente, es importante destacar que para este diseño, la ejecución de la política incurre en tiempos de cómputo considerables, que en el caso de sistemas embebidos como el robot *NAO* pueden ser prohibitivos. En el caso que los tiempos de ejecución en el robot sean

inviables, una opción consiste en seguir las guias de diseño planteadas en [79], donde se logra ejecutar redes convolucionales en menos de un milisegundo dentro del robot *NAO*, mediante una reducción sistemática de los parámetros de las redes. Si no se logra encontrar una arquitectura que presente buenos resultados en el entrenamiento o que no pueda ser desplegada en tiempo real, otra alternativa consiste en utilizar herramientas de *hardware* como la desarrollada en [80] que consiste esencialmente en una arquitectura de procesamiento externo mediante un dispositivo *Odroid*, o incluso *Nvidia Jetson*, que tiene la ventaja de poseer de manera integrada una cantidad limitada de núcleos *CUDA*.

Capítulo 5

Conclusiones

En este trabajo se realizó un estudio de la literatura relacionada al Aprendizaje Reforzado, tomando como eje transversal sus aplicaciones en robótica. En ese sentido se presentaron orígenes, fundamentos, objetivos y limitaciones tanto para las metodologías clásicas y como para las del estado del arte, lo que permitió identificar aquellos conceptos claves al momento de desplegar dichas técnicas en problemas asociados a la robótica.

Posteriormente, se diseño e implementó la librería de aprendizaje reforzado *UChileRL*, cuyo diseño permitió la implementación de algoritmos tales como *SARSA*, Actor-Crítico Lineal, *DDPG*, entre otros, de una manera sencilla, y utilizando una interfaz común, independiente del problema particular a resolver. Adicionalmente, se validó el uso de la librería en distintos problemas, tanto relacionados al caso de estudio, como con problemas clásicos de la literatura. Finalmente, se validó su integración en sistemas embebidos como el robot *NAO*, reportando tiempos de ejecución asequibles para su uso en tiempo real.

Luego, se examinó el caso de estudio, con énfasis en la determinación de problemas no resueltos, o con resultados parciales, que debido a su naturaleza fuesen candidatos a ser resueltos mediante técnicas avanzadas como el Aprendizaje Reforzado. En cada uno de los problemas identificados se realizó un detallado proceso de diseño, haciendo incapíe en aquellos conceptos clave, debido a su recurrencia en aplicaciones robóticas o por su carácter crítico tanto en términos de rendimiento, factibilidad y transferencia a sistemas físicos. En este sentido se destacan 3 metodologías claves para mejorar rendimientos y disminuir el *reality gap* entre simulaciones y robots físicos.

La primera metodología identificada corresponde a abordar el *reality gap* producto del modelamiento y simulación de actuadores, mediante el no uso del Aprendizaje Reforzado como herramienta de control motor directo, utilizando en cambio un sistema intermedio de control independiente para dicha labor, de manera que el agente actúa sobre dicho sistema. La ventaja de esta propuesta consiste en delegar el manejo del *reality gap* a sistemas *ad-hoc* en caso de existir, y enfocar los esfuerzos de diseño únicamente en el agente de Aprendizaje Reforzado. En este trabajo como ejemplo de esta metodología se delega la generación de movimientos directos a un sistema de caminata omnidireccional, la cual es controlada a un alto nivel por el agente de Aprendizaje Reforzado.

La segunda de las metodologías identificadas consiste en otorgar un fuerte enfoque en la integración del agente de Aprendizaje Reforzado con los demás sistemas presentes en la aplicación robótica. Si bien esta idea resulta natural en el ámbito de la Robótica, donde la integración siempre juega un rol fundamental, es común encontrar aplicaciones de robótica provenientes del campo de la Inteligencia Artificial, en las cuales no se toma en cuenta este proceso. En este trabajo se evalúan los beneficios de realizar este proceso de manera correcta, al considerar las características de la implementación del sistema de locomoción del robot al momento de realizar el diseño del agente de Aprendizaje Reforzado.

En último lugar, se desarrolla un sistema sencillo para lidiar con el *reality gap* producto del uso de imágenes simuladas, cuyo efecto, de no ser considerado, puede generar un nulo rendimiento en sistemas físicos. El mecanismo propuesto consiste en utilizar un sistema de segmentación entre la imagen bruta y la representación utilizada por el agente. Esta propuesta permite mantener un uso de las propiedades espaciales del contenido de la imagen, facilita el aprendizaje debido a la separación en clases del contenido de la imagen y permite el uso sin pérdida de precisión de redes convolucionales cuantizadas. Sin embargo se debe tener en consideración, que si bien en términos de *reality gap* el uso de esta metodología permite abordar dicho problema completamente, en la práctica su rendimiento está dado por la calidad y disponibilidad del sistema de segmentación.

Para los problemas de baja dimensionalidad estudiados (*Ball Dribbling* e *In-walk Kicks*) se evidenció que los distintos métodos estudiados, tanto clásicos como del estado del arte, permiten resolver de manera correcta los problemas planteados, con rendimientos similares y en todos los casos, tiempos de ejecución factibles para su despliegue en aplicaciones reales. Finalmente cabe destacar que debido a la alta cantidad de hiper-parámetros existentes, no se pudo obtener una conclusión clara con respecto a la superioridad de alguna técnica en particular. En vista de lo anterior, para problemas de baja dimensionalidad se plantea una elección de la herramienta a utilizar mediante otro tipo de criterios.

En primer lugar, se debe tener en cuenta la diferencia entre los instantes de entrenamiento y ejecución (*test*). Esta distinción es importante, pues si bien métodos de *Deep Reinforcement Learning* involucran complejas operaciones matemáticas durante su entrenamiento, posteriormente solo utilizan operaciones sencillas como aritmética de matrices. Un ejemplo concreto de la importancia de este criterio corresponde al entrenamiento del sistema de aprendizaje directamente en sistemas embebidos, con bajos recursos computacionales. En este caso no se puede directamente entrenar el agente en el dispositivo embebido mediante técnicas como *DDPG*, pero si es factible el entrenamiento mediante técnicas como *SARSA*. En cambio, si el entrenamiento es realizado de manera externa en un simulador, es posible posteriormente desplegar políticas de *DDPG* y *SARSA* en el sistema físico. Finalmente es importante destacar que existen metodologías que aprovechan el aprendizaje *off-line* de algoritmos como *DDPG* y realizan el proceso de aprendizaje (actualización de los modelos) en dispositivos externos, mientras que las experiencias se obtienen de manera asíncrona en el sistema físico, permitiendo el entrenamiento de algoritmos complejos de *Deep Reinforcement Learning* en dispositivos limitados.

Un segundo criterio a considerar al momento de elegir un algoritmo consiste en la homogeneidad del rendimiento de la política, debido a que en muchas aplicaciones es fundamental poder asegurar el funcionamiento de sistemas en amplios rangos de operación. En este senti-

do, se destaca que debido a la localidad de la representación *RBF*, es posible obtener políticas finales con desempeños fuertemente disparejos, incluso en zonas de operación similares, mientras que el uso de redes neuronales *feed-forward*, obtiene políticas con un rendimientos más uniforme en las distintas zonas de operación.

En último lugar, también es crucial la naturaleza de la política resultante, incluso en casos donde ambas políticas presenten rendimientos similares. El caso emblemático de esta situación corresponde a los algoritmos que utilizan acciones discretas, pues pese a que pueden resolver problemas continuos mediante una adecuada discretización del espacio de acciones, la naturaleza de la política en estos casos, debido a que no existen restricciones en las acciones, consiste en controles de tipo *bang-bang*, los cuales suelen ser dañinos para sistemas físicos en cuanto involucran movimientos bruscos con fuertes torques o fuerzas asociadas sobre el sistema físico. Otro hecho de especial importancia en este sentido corresponde al *reality gap*, el cual se presenta con mayor fuerza para el caso de acciones discretas debido a la diferencia entre el funcionamiento de actuadores reales y simulados.

Finalmente, mediante la resolución del problema de Navegación visual, se validó empíricamente la capacidad del uso de *Deep Reinforcement Learning* como herramienta para abordar problemas de alta dimensionalidad mediante el uso de información visual. Un punto importante relacionado a este problema, es que pese a utilizar cantidades acotadas de episodios, comportamientos complejos surgen mediante esta metodología, relacionados a una planificación a largo plazo del problema, y aversión a riesgos, incluso generando estrategias para abordar la baja observabilidad del proceso. Finalmente es importante destacar que mediante la validación de este problema se abre la posibilidad para nuevos desarrollos debido a la alta cantidad de problemas complejos para los cuales es necesario el uso de representaciones de alta dimensionalidad.

5.1. Trabajo Futuro

Las herramientas y técnicas desarrolladas en el contexto de este trabajo permiten abordar de manera sencilla distintos nuevos problemas mediante Aprendizaje Reforzado, con un enfoque orientado al diseño del problema a resolver. Un caso especial de este fenómeno corresponde a la problemática presente en el caso de estudio, donde debido debido a la existencia de problemas resueltos anteriormente, tales como *Ball Dribbling*, el proceso resulta particularmente sencillo.

Adicionalmente, debido al éxito obtenido en problemas como navegación visual, se abre la posibilidad de abordar diversos problemas relacionados con información visual. Para el caso de estudio se identifican 2 problemas que se originan en situaciones estudiadas en este trabajo.

En primer lugar se puede plantear de manera formal la labor de visión activa mediante estrategias de *Deep Reinforcement Learning*. Una manera de realizar este trabajo consiste en plantear esta tarea de manera independiente, en la cual se pueden utilizar criterios de teoría de la información, con el objetivo de maximizar la adquisición de información, mientras se realiza alguna actividad. Otra formulación consiste en el planteamiento de la visión activa,

como un problema a resolver de manera concurrente a otra problemática. En esta situación se desea resolver una tarea específica, sobre la cual el problema de visión activa se plantea como un problema descentralizado con respecto al original, lo que permite un diseño independiente y *ad-hoc*, mientras que el aprendizaje se mantiene orientado al cumplimiento del objetivo original.

Finalmente, otro de los problemas identificados corresponde a una extensión de los problemas *Ball Dribbling* y Navegación visual. En este problema se plantea de manera análoga a lo desarrollado en [81] la separación del problema de *Ball Dribbling* en 2 subproblemas: la capacidad de llevar la pelota entre un punto inicial y otro final (problema estudiado en este trabajo), y a la realización de dicha acción modelando ambientes dinámicos como los presentes en un encuentro de fútbol robótico. Se plantea entonces un esquema de *Layered Learning* [82], para lo cual en primer lugar se resuelve el primer problema mediante un modelo de baja dimensionalida como el planteado en este trabajo, lo que le otorga una fuerte robustez al *reality gap*, y posteriormente se cumple el segundo objetivo mediante técnicas de *Deep Reinforcement learning*, de una manera similar a lo realizado para el problema de navegación visual, modificando en tiempo real el objetivo de *Ball Dribbling* del problema anterior, permitiendo integrar ambos problemas según el esquema de *Layered Learning*.

Bibliografía

[1] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: the Robot World Cup Initiative. *Proceedings of the International Conference on Autonomous Agents*, 1998.

[2] Richard S Sutton and Andrew G Barto. *Introduction to Reinforcement Learning*, volume 4. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[3] J. Andrew Bagnell. Reinforcement Learning in Robotics: A Survey. *Springer Tracts in Advanced Robotics*, 97:9–67, 2014.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A Riedmiller. Playing Atari with Deep Reinforcement Learning. *CoRR*, 2013.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

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

[7] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[8] W. Schultz, P. Dayan, and P. R. Montague. A Neural Substrate of Prediction and Reward. *Science*, 275(5306):1593–1599, mar 1997.

[9] Gerald Tesauro. Temporal Difference Learning and TD-Gammon. *Commun. ACM*, 38(3):58–68, 1995.

[10] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[11] Guillaume Lample and Devendra Singh Chaplot. Playing FPS Games with Deep Reinforcement Learning. sep 2016.

[12] N Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 3, pages 2619 – 2624 Vol.3, 2004.

[13] Bernhard Hengst. Reinforcement learning of bipedal lateral behaviour and stability control with ankle-roll activation. pages 411–418, 2013.

[14] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. Emergence of Locomotion Behaviours in Rich Environments. 2017.

[15] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3389–3396, 2017.

[16] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *CoRR*, 2015.

[17] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of Model-Based Reinforcement Learning: Applications on Robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, may 2017.

[18] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A Survey on Policy Search for Robotics. *Found. Trends Robot*, 2(1–2):1–142, 2013.

[19] Ivo Grondman, Lucian Buoniu, Gabriel A D Lopes, and Robert Babuška. A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. pages 1–17, 2012.

[20] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation. pages 1–6, mar 2017.

[21] Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning. 2017.

[22] X. Jia. *Deep Learning for Actor-Critic Reinforcement Learning*. Master thesis, TU Delft, 2015.

[23] Ercan Elibol, Juan Calderon, Martin Llofriu, Carlos Quintero, Wilfrido Moreno, and Alfredo Weitzenfeld. Power Usage Reduction of Humanoid Standing Process Using Q-Learning. In *RoboCup 2015: Robot World Cup XIX*, volume 9513, pages 251–263, 2015.

[24] Leonardo Leottau, Carlos Celemin, and Javier Ruiz-del Solar. Ball Dribbling for Humanoid Biped Robots: A Reinforcement Learning and Fuzzy Control Approach. In Reinaldo A C Bianchi, H Levent Akin, Subramanian Ramamoorthy, and Komei Sugiura, editors, *RoboCup 2014: Robot World Cup XVIII*, volume 8992, pages 549–561. Springer International Publishing, 2015.

[25] David Leonardo Leottau and Javier Ruiz-del Solar. An Accelerated Approach to Decentralized Reinforcement Learning of the Ball-Dribbling Behavior. In *AAAI Workshop: Knowledge, Skill, and Behavior Transfer in Autonomous Robots*, 2015.

[26] Kenzo Lobos-Tsunekawa, David L. Leottau, and Javier Ruiz-del Solar. Toward Real-Time Decentralized Reinforcement Learning using Finite Support Basis Functions. *RoboCup 2017 Symposium*, 2017.

[27] Christopher J.C.H. Watkins and Peter Dayan. Technical Note: Q-Learning. *Machine Learning*, 8(3):279–292, 1992.

[28] G a Rummery and M Niranjan. On-line Q-learning using connectionist systems. (September), 1994.

[29] Geoffrey J. Gordon. Reinforcement Learning with Function Approximation Converges to a Region. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS)*, pages 1040–1046, Denver, CO, USA, 2000.

[30] J Park and I W Sandberg. Universal Approximation Using Radial-Basis-Function Networks. *Neural Computation*, 3(2):246–257, jun 1991.

[31] Lucian Busnoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. Reinforcement learning and dynamic programming using function approximators. page 260, 2010.

[32] Alborz Geramifard, Thomas J Walsh, Stefanie Tellex, Girish Chowdhary, Nicholas Roy, and Jonathan P How. A Tutorial on Linear Function Approximators for Dynamic Programming and Reinforcement Learning. *Foundations and Trends in Machine Learning*, 6(4):375–451, 2013.

[33] G D Konidaris, S Osentoski, and P S Thomas. Value Function Approximation in Reinforcement Learning using the Fourier Basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pages 380–385, 2011.

[34] André da Motta Salles Barreto and Charles W Anderson. Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. *Artificial Intelligence*, 172(4):454–482, 2008.

[35] Martin Riedmiller. Neural fitted Q iteration - First experiences with a data efficient neural Reinforcement Learning method. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3720 LNAI:317–328, 2005.

[36] Javier García and Fernando Fernández. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*, 16:1437–1480, 2015.

[37] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2010.

[38] Lucian Busoniu, Bart De Schutter, and Robert Babuska. Decentralized Reinforcement Learning Control of a Robotic Manipulator. In *Ninth International Conference on Control, Automation, Robotics and Vision, {ICARCV} 2006, Singapore, 5-8 December 2006, Proceedings*, pages 1–6, 2006.

[39] David L. Leottau, Aashish Vatsyayan, Javier Ruiz-del Solar, and Robert Babuška. Decentralized Reinforcement Learning Applied to Mobile Robots. In *20th Annual RoboCup International Symposium*, number July, pages 368–379. 2017.

[40] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[41] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, abs/1502.0, 2015.

[42] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[43] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. StarCraft II: A New Challenge for Reinforcement Learning. *CoRR*, abs/1708.0, 2017.

[44] Matthew J Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. *CoRR*, abs/1507.0, 2015.

[45] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter I Corke. Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control. *CoRR*, abs/1511.0, 2015.

[46] Andrei A Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-Real Robot Learning from Pixels with Progressive Nets. *CoRR*, abs/1610.0, 2016.

[47] K Arulkumaran, M P Deisenroth, M Brundage, and A A Bharath. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine*, 34(6):26–38, nov 2017.

[48] Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Pittsburgh, PA, USA, 1992.

[49] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. *CoRR*, abs/1509.0, 2015.

[50] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling Network Architectures for Deep Reinforcement Learning. *CoRR*, abs/1511.0, 2015.

[51] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin A Riedmiller. Deterministic Policy Gradient Algorithms. In *ICML*, volume 32 of *{JMLR} Workshop and Conference Proceedings*, pages 387–395. JMLR.org, 2014.

[52] Matthew J Hausknecht and Peter Stone. Deep Reinforcement Learning in Parameterized Action Space. *CoRR*, abs/1511.0, 2015.

[53] Todd Hester. RL-TEXPLORE-ROS-pkg. https://github.com/toddhester/rl-texplore-ros-pkg, 2011.

[54] Saminda Abeyruwan and Ubbo Visser. RLLib: C++ Library to Predict, Control, and Represent Learnable Knowledge Using On/Off Policy Reinforcement Learning. In *RoboCup 2015: Robot World Cup XIX. Lecture Notes in Computer Science*, volume 9513, pages 356–364. Springer, 2015.

[55] M Quigley, B P. Gerkey, K Conley, J Faust, T Foote, J Leibs, E Berger, R Wheeler, and A Y. Ng. ROS: An open-source Robot Operating System. *ICRA Workshop on Open Source Software*, 3:1–6, 2009.

[56] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *CoRR*, abs/1606.0, 2016.

[57] E Todorov, T Erez, and Y Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, oct 2012.

[58] Iker Zamora, Nestor Gonzalez Lopez, Victor Mayoral Vilches, and Alejandro Hernandez Cordero. Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo. *CoRR*, abs/1608.0, 2016.

[59] Gaël Guennebaud, Benoît Jacob, and Others. Eigen v3. http://eigen.tuxfamily.org, 2010.

[60] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *CoRR*, abs/1603.0, 2016.

[61] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[62] Joseph Redmon. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/, 2013.

[63] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *CoRR*, abs/1602.0, 2016.

[64] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jérôme Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. Mechatronic design of NAO humanoid. In *IEEE International Conference on Robotics and Automation*, pages 769–774, Kobe, Japan, 2009. IEEE.

[65] Thomas Rofer, Tim Laue, Jonas Kuball, Andre Lubken, Florian Maas, Judith Muller, Lukas Post, Jesse Richter-Klug, Peter Schulz, Andreas Stolpmann, Alexander Stowing, and Felix Thielke. B-Human Team Report and Code Release 2016, 2016.

[66] Thomas Röfer and Tim Laue. *On B-Human's Code Releases in the Standard Platform League - Software Architecture and Impact*, pages 648–655. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[67] Tim Laue, Kai Spiess, and Thomas Röfer. SimRobot - A General Physical Robot Simulator and Its Application in RoboCup. *RoboCup 2005: Robot Soccer World Cup IX*, pages 173–183, 2006.

[68] Carlos Celemin and Javier Ruiz-del Solar. Interactive Learning of Continuous Actions from Corrective Advice Communicated by Humans, 2015.

[69] Colin Graf and Thomas Röfer. A Center of Mass Observing 3D-LIPM Gait for the RoboCup Standard Platform League Humanoid. In *RoboCup 2011: Robot Soccer World Cup XV*, pages 102–113. Springer, Berlin, Heidelberg, 2011.

[70] Felix Wenk and Thomas Röfer. *Online Generated Kick Motions for the NAO Balanced Using Inverse Dynamics*, pages 25–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[71] Judith Müller, Tim Laue, and Thomas Röfer. Kicking a Ball - Modeling Complex Dynamic Motions for Humanoid Robots. In Javier Ruiz-del Solar, Eric Chown, and Paul G Ploeger, editors, *RoboCup 2010: Robot Soccer World Cup XIV*, volume 6556 of *Lecture Notes in Artificial Intelligence*, pages 109–120. Springer; Heidelberg; http://www.springer.de/, 2011.

[72] Arne Böckmann and Tim Laue. Kick Motions for the NAO Robot using Dynamic Movement Primitives. In *RoboCup 2016: Robot World Cup XIX*, Lecture Notes in Artificial Intelligence. Springer, 2017.

[73] Thomas Röfer, Tim Laue, Judith Müller, Alexander Fabisch, Fynn Feldpausch, Katharina Gillmann, Colin Graf, Thijs Jeffry de Haas, Alexander Härtl, Arne Humann, Daniel Honsel, Philipp Kastner, Tobias Kastner, Carsten Könemann, Benjamin Markowsky, Ole Jan Lars Riemann, and Felix Wenk. B-Human Team Report and Code Release 2011, 2011.

[74] Pablo Cano and Javier Ruiz-del Solar. Robust Tracking of Multiple Soccer Robots Using Random Finite Sets. In Sven Behnke, Raymond Sheh, Sanem Sariel, and Daniel D Lee, editors, *RoboCup 2016: Robot World Cup XX*, volume 9776 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2016.

[75] Thomas Reinhardt. Kalibrierungsfreie Bildverarbeitungsalgorithmen zur echtzeitfähigen Objekterkennung im Roboterfußball. Master's thesis, HTWK Leipzig, 2011.

[76] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *CoRR*, abs/1609.0, 2016.

[77] Pablo Guerrero, Javier Ruiz-del Solar, and Miguel Romero. *Explicitly Task Oriented Probabilistic Active Vision for a Mobile Robot*, pages 85–96. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[78] Matías Mattamala, Constanza Villegas, José Miguel Yáñez, Pablo Cano, and Javier Ruiz-del Solar. A Dynamic and Efficient Active Vision System for Humanoid Soccer Robots. In *RoboCup 2015: Robot World Cup XIX. Lecture Notes in Computer Science*. Springer, 2015.

[79] Nicolás Cruz, Kenzo Lobos-Tsunekawa, and Javier Ruiz-del solar. Using Convolutional Neural Networks in Robots with Limited Computational Resources : Detecting NAO Robots while Playing Soccer. *RoboCup 2017 Symposium*, 2017.

[80] Matías Mattamala, Gonzalo Olave, Clayder González, Nicolás Hasbún, and Javier Ruiz-del Solar. The NAO Backpack: An Open-hardware Add-on for Fast Software Development with the NAO Robot. *CoRR*, abs/1706.0, 2017.

[81] Carlos Celemin, David Leonardo Leottau, Rodrigo Perez, Javier Ruiz-del Solar, and Manuela Veloso. Interactive Machine Learning Applied to Dribble a Ball in Soccer with Biped Robots. 2017.

[82] Peter Stone and Manuela Veloso. Layered Learning. In Ramon López de Mántaras and Enric Plaza, editors, *Machine Learning: ECML 2000 (Proceedings of the Eleventh European Conference on Machine Learning)*, pages 369–381. Springer Verlag, Barcelona, Spain, 2000.