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

ESTUDIO E IMPLEMENTACIÓN DE ALGORITMO *DREAMER* PARA RESOLVER PROBLEMA DE NAVEGACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA

ELISA MARÍA PARGA MONTT

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

PROFESOR CO-GUÍA:
FRANCISCO LEIVA CASTRO

COMISIÓN:
ANDRÉS CABA RUTTE

Este proyecto ha sido parcialmente financiado por Proyecto FONDECYT 1201170 y Proyecto ANID-PIA AFB220002

SANTIAGO DE CHILE
2024

RESUMEN DE LA TESIS PARA OPTAR
AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA
POR: ELISA MARÍA PARGA MONTT
FECHA: 2024
PROF. GUÍA: JAVIER RUIZ DEL SOLAR SAN MARTÍN

ESTUDIO E IMPLEMENTACIÓN DE ALGORITMO *DREAMER* PARA RESOLVER PROBLEMA DE NAVEGACIÓN

La navegación autónoma en robótica móvil representa un desafío significativo, abordado a lo largo de décadas de investigación. Este trabajo se centra en la aplicación del algoritmo de aprendizaje reforzado *Dreamer* para proporcionar una solución robusta y eficiente al problema. Mediante "imaginación latente" y el uso de *World Models*, características del algoritmo, se busca obtener políticas de navegación que sean no solo eficientes en cantidad de muestras, sino que también capaces de generalizar bien.

Para alcanzar este propósito, se lleva a cabo una validación del correcto funcionamiento del algoritmo en entornos de ejecución clásicos. Posteriormente, se realizan pruebas en dos mundos de navegación que presentan distintos niveles de dificultad. En dichas pruebas, se exploran dos tipos de observaciones para evaluar el comportamiento del algoritmo en diversas condiciones.

Los resultados obtenidos se presentan al final del informe, destacando tanto la eficiencia del método propuesto en cuanto a interacciones agente-ambiente, como la sensibilidad que el algoritmo presenta ante parametrizaciones diferentes. Además, se discuten posibles direcciones para futuras investigaciones, identificando áreas que podrían beneficiarse de una mayor exploración y refinamiento. Este trabajo contribuye al avance en el campo de la navegación autónoma, proporcionando valiosas perspectivas y estableciendo una base para investigaciones futuras en esta área de estudio.

A mis padres, Andrés y Carolina.

Agradecimientos

Es difícil cerrar mi carrera de ingeniería civil eléctrica agradeciéndole, por medio de este breve texto, a solo un par de personas, pues han sido muchas las que me han acompañado a lo largo de esta aventura.

En primer lugar me gustaría agradecerle a mi familia por todo su apoyo y compañía a lo largo de mis estudios. Gracias a mis hermanos, Maida, Jose y Pedro, por siempre creer en mí, aguantarme y apoyarme hasta último minuto. A mis padres, por haber estado siempre a la escucha durante estos 7 largos años y haberme entregado todas las herramientas que tenían para fomentar mi aprendizaje y mis proyectos.

Estoy muy agradecida también, de todos los amigos que crucé en mi pasada por la universidad. Gracias Pedro, Werner, Pascal, Coni, Cata, Javi, Cata, Fer, Isi, Brook, Sofi, Maniega, Max, Cami, Pauli, Pablo, Luca por haber estado siempre ahí, cuando a pesar de que el estudio se hacía difícil y tedioso, las risas no faltaban. Quiero agradecerle también a Joaquín, quien fue la persona que más me apoyó en todos mis proyectos universitarios y sobretodo en esta tesis. Gracias por haberme sacado sonrisas en todo momento y haberme motivado en mis peores momentos.

Creo que mi carrera universitaria no hubiese sido lo mismo sin mi experiencia en Francia. Gracias Laura, Marga, Afnan, Theo, Renan, Carmen, Luca, Anna, Matis por haber sido mis compañeros y cómplices en 3 años inolvidables. Gracias por haber sido mi familia y mis amigos cuando me encontraba lejos y por todas las cosas que me enseñaron.

No podría no mencionar también a mis amigas del colegio, las *Cumpas SGC* y las de *Fiesta en la playa* que a pesar de no entender absolutamente nada de lo que estudiaba me motivaron y me apoyaron siempre.

Finalmente, quiero agradecer a Francisco Leiva por todo el apoyo brindado a lo largo de esta tesis, por sus conocimientos y disponibilidad que fueron cruciales para sacar este trabajo adelante. También al profesor Javier Ruiz del Solar por haberme dado la oportunidad de trabajar en su área de investigación.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto y motivación

La navegación autónoma es un problema relevante en robótica móvil [3]. A grandes rasgos, este se refiere a la capacidad de los robots para moverse de manera autónoma en entornos que podrían ser desconocidos y/o dinámicos, sin colisionar con obstáculos. La navegación autónoma, tiene múltiples aplicaciones en diversos sectores como en logística, minería, exploración, agricultura, entre otros [4]. En este contexto, por medio de la automatización de procesos industriales, se ha logrado mayor eficiencia, productividad y seguridad, lo que incentiva la investigación en el rubro y motiva aún más la implementación de sistemas autónomos en estas áreas. Dada la relevancia del problema de navegación, se han presentado diversas propuestas que buscan solucionarlo utilizando diferentes métodos [5, 6, 7, 8].

Ciertos métodos clásicos [9, 10] logran resolver el problema de navegación con éxito en escenarios realistas, por ejemplo, logrando encontrar trayectorias libres de colisiones a partir de señales ruidosas y parcialmente observables. No obstante, estos sistemas requieren considerables esfuerzos ingenieriles y computacionales, y pueden fallar en ambientes más desafiantes o con una mayor cantidad de restricciones. En general, estos métodos requieren modelos precisos y conocimiento detallado del sistema. En entornos cambiantes o poco estructurados, estos enfoques pueden no ser capaces de adaptarse de manera efectiva. A pesar de poder descomponer el problema de navegación en diferentes tareas menos complejas, como la evasión de colisiones o alcance de la región objetivo, los métodos clásicos no logran dar soluciones robustas en estos entornos.

En las últimas décadas, han habido grandes avances en la investigación de métodos basados en aprendizaje, como por ejemplo, *machine learning* (ML), *deep learning* (DL) o *reinforcement learning* (RL). Estos métodos han permitido automatizar tareas previamente realizadas por seres humanos o resolver complejos problemas de manera robusta. Los avances en métodos basados en aprendizaje han revolucionado la tecnología y brindado soluciones a diversas problemáticas en diferentes industrias.

En particular, el aprendizaje reforzado o *reinforcement learning* (RL), ha logrado un gran progreso en múltiples aspectos [4]. Los métodos basados en este paradigma tienen la capacidad de resolver una diversa gama de problemas desafiantes, fundamentalmente de sistemas autónomos

que deben tomar decisiones secuenciales. El objetivo del RL es lograr que agentes autónomos artificiales, mediante interacciones con su entorno, sean capaces de tomar acciones para realizar una tarea determinada. Mediante ensayo y error, los agentes buscan maximizar las recompensas que obtienen tras ejecutar acciónes.

Los algoritmos desarrollados bajo la perspectiva del aprendizaje reforzado, han entregado soluciones más robustas a diferentes tareas de control, automatizando completamente agentes robóticos [11, 12, 13]. El *reinforcement learning*, presenta un gran potencial para resolver el problema de navegación, pues reduce los esfuerzos ingenieriles que presenta el desarrollo e implementación de métodos clásicos. Además, los ambientes del mundo real, a menudo están sujetos a incertidumbre, como ruido sensorial, cambios imprevistos y variabilidad en las condiciones. Los métodos clásicos presentan dificultades para manejar esta incertidumbre, mientras que los enfoques basados en RL pueden ser diseñados para ser robustos frente a condiciones variables.

Sin embargo, a pesar de que el aprendizaje reforzado presenta ventajas prometedoras, cuando se trata del problema de navegación en casos más realistas, las dificultades persisten. Realizar una exploración eficiente de entornos desconocidos o grandes hace que el problema sea más complejo, la percepción sensorial puede traer ruido o errores que confundan al agente o para el caso de aplicaciones más reales, garantizar seguridad de navegación deja de ser trivial. Por estas razones, al momento de resolver problemas utilizando aprendizaje reforzado, se requiere de una gran cantidad de interacciones agente-ambiente. Considerando los riesgos que presenta la ejecución de acciones exploratorias, tanto para el robot como para su entorno, la puesta en práctica de un entrenamiento en el mundo real se hace poco factible. Por este motivo, los simuladores se han transformado en una herramienta fundamental para el desarrollo de algoritmos de aprendizaje reforzado.

El uso de simuladores, permite recrear ambientes para que los agentes aprendan a resolver sus tareas determinadas y generar experiencias para el aprendizaje. No obstante, existe una brecha entre los entornos simulados y el mundo real, pues la dinámica de las interacciones o las observaciones pueden ser diferentes, esto se conoce como "*reality gap*". Cuando esta disparidad es especialmente notable, la ejecución de un agente que ha sido entrenado en simulaciones se ve perjudicada al ser desplegado en el entorno real. Esto puede resultar en un comportamiento completamente disímil al observado durante el entrenamiento en simulador. Además de presentar el desafío del "*reality gap*", los simuladores pueden requerir extensos períodos de entrenamiento. Cada interacción con el entorno conlleva una respuesta del simulador que toma un tiempo determinado. A pesar de que dicha respuesta sea rápida, cuando se trata de tareas complejas, la gran cantidad de experiencias necesarias se traduce en largos tiempos de entrenamiento.

Con el propósito de mitigar los desafíos mencionados previamente, el trabajo propuesto busca adaptar el algoritmo *Dreamer* [2] para resolver el problema de navegación. Este algoritmo se caracteriza por ser un método que utiliza una menor cantidad de interacciones agente-ambiente en comparación con otros algoritmos de aprendizaje reforzado (*e.g.* [14, 11, 15]).

Dreamer está diseñado para problemas desarrollados en entornos clásicos de aprendizaje reforzado, como lo son [16] o [17]. A pesar de esto, el algoritmo ha sido ampliamente utilizado en los últimos años debido a su gran capacidad de aprendizaje, dando soluciones robustas a variadas tareas de control y al mismo tiempo reduciendo considerablemente tanto costos computacionales como tiempos de entrenamiento. Este trabajo, busca extender dicho algoritmo de modo que pueda resolver de manera eficiente el problema de navegación autónoma.

Al aplicar esta adaptación del algoritmo *Dreamer*, se espera mejorar significativamente la eficiencia y el rendimiento de la navegación autónoma en entornos complejos. Esto abrirá nuevas oportunidades en diversas aplicaciones prácticas, por ejemplo, donde múltiples robots puedan colaborar de manera efectiva para lograr objetivos comunes.

1.2. Objetivos

1.2.1. Objetivo general

El objetivo de la presente memoria de título es desarrollar e implementar una adaptación del algoritmo Dreamer a un entorno donde se resuelva el problema de navegación. Este se entiende como lograr que un agente autónomo logre llegar a una región objetivo previamente definida, sin colisionar con obstáculos que encuentre en un entorno inicialmente desconocido.

1.2.2. Objetivos específicos

Los objetivos específicos que se esperan cumplir con este trabajo son los siguientes:

1. Modelar el problema de navegación como un *Partially Observable Markov Decision Process* (POMDP).
2. Implementar y validar el algoritmo *Dreamer* en entornos clásicos de aprendizaje reforzado para luego adaptarlo a entornos de navegación.
3. Definir métricas de desempeño para evaluar el algoritmo.
4. Comparar resultados del algoritmo en robots con diferentes señales sensoriales.

1.3. Estructura del documento

Este documento, presenta la siguiente estructura:

- **Capítulo 2:** se exponen los fundamentos teóricos necesarios para comprender el trabajo realizado. Se definen los procesos de decisión de Markov, se hace una revisión general del aprendizaje reforzado y finalmente se presentan los algoritmos en los que se sustenta esta memoria.
- **Capítulo 3:** en este capítulo se hace un estudio del estado del arte con respecto a la navegación autónoma. En particular, se presenta el problema, ciertos métodos clásicos para resolverlo y las soluciones propuestas por el aprendizaje reforzado profundo.
- **Capítulo 4:** en este capítulo, se expone la metodología seguida para resolver el problema de navegación. En particular, se presentan los entornos clásicos en los que se valida el correcto funcionamiento del algoritmo, la formulación matemática del problema de navegación como un POMDP, la propuesta para resolverlo y una breve explicación de la implementación realizada.
- **Capítulo 5**: se presenta la configuración de los entrenamientos realizados, los resultados obtenidos, su interpretación y se plantea una discusión en torno a estos. Se comparan diferentes configuraciones de entrenamiento y se propone la mejor de estas para un entorno de navegación simple y complejo.

- **Capítulo 6:** finalmente, se exponen las conclusiones obtenidas a partir del trabajo realizado y se sugieren las posibles direcciones que podría tomar la investigación a futuro.

Capítulo 2

Marco Teórico

2.1. Procesos de Decisión de Markov y Aprendizaje Reforzado

2.1.1. Procesos de Decisión de Markov

Los Procesos de Decisión de Markov (*Markov Decision Processes* [MDPs]) son una formulación matemática empleada para modelar problemas de toma de decisiones secuenciales. Estos permiten definir problemas de aprendizaje automático donde un 'agente' (quien toma las decisiones) aprende una tarea en particular en un 'ambiente' (el entorno que en el cuál el agente se encuentra inmerso) inicialmente desconocido. Los MDPs se definen matemáticamente por la tupla $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$ donde:

- $\mathcal{S}$ es un conjunto de estados.
- $\mathcal{A}$ es un conjunto de posibles acciones a ejecutar por el agente.
- $\mathcal{T}(s, a, s') = p(s'|s, a)$ es una función de transición estocástica que entrega la probabilidad de pasar al estado s' a partir de un estado s y una acción a.
- $r(s, a)$ es la función de recompensa esperada para cada transición de estado.
- γ es un factor de descuento, representado por un número real entre $[0, 1]$.

Los Procesos de Decisión de Markov Parcialmente Observables (*Partially Observable Markov Decision Processes* [POMDPs]) son una generalización de los MDPs donde un agente no puede observar completamente el estado de su entorno. Formalmente, los POMDPs quedan definidos por una tupla $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \mathcal{O}, \Omega, \gamma)$ donde:

- $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$ definen un MDP.
- $\mathcal{O}(s', a, o) = p(o_{t+1} = o|s_{t+1} = s', a_t = a)$ es la función de observación.
- Ω es el conjunto de observaciones.

En cada instante de tiempo $t = 1, ..., T$ un agente observa o_t, ejecuta una acción a_t, recibe una recompensa escalar r_t y transiciona a un nuevo estado s_{t+1} según la función $\mathcal{T}$. Con esto, el par estado-acción (s_t, a_t) determina la probabilidad de llegar a un estado s_{t+1} y de obtener una recompensa r_t. Esto se conoce como la "propiedad de Markov".

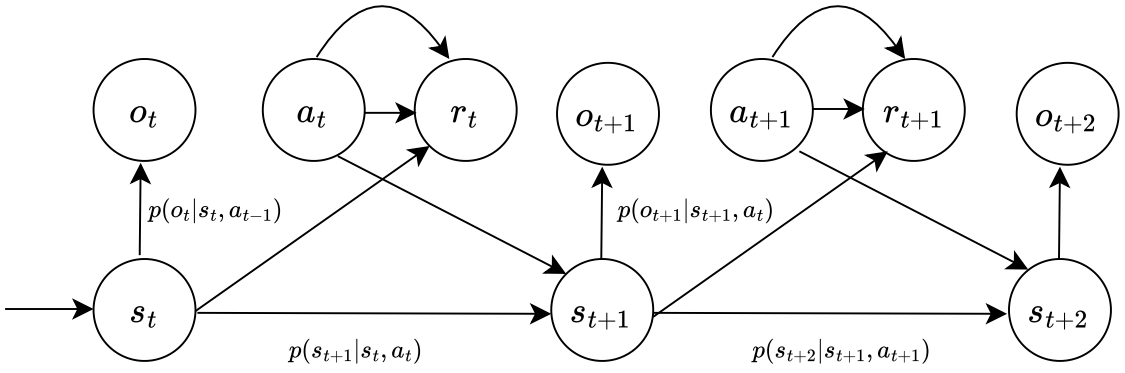


Figura 2.1: Flujo de observaciones, estados, acciones y recompensas de un POMDP.

2.1.2. Aprendizaje Reforzado

El aprendizaje reforzado es un área de la inteligencia artificial que resuelve problemas de toma de decisiones secuenciales. En su formulación, un agente activo y autónomo interactúa con un entorno inicialmente desconocido. Mediante prueba y error (interacción con un ambiente modelado como un MDP o una variante de este), el agente logra aprender una política $\pi(a|s)$ que le permite cumplir una tarea determinada.

Así, el agente percibe un estado s_t, ejecuta una acción a_t según la política encontrada $\pi(a_t|s_t)$. Mediante la ejecución de la acción, el agente transiciona a un estado s_{t+1} y se obtiene una recompensa r_t. Dicha interacción con el ambiente da lugar a una distribución de probabilidades sobre las secuencias de pares estado-acción llamadas "trayectorias". Entonces, si el estado inicial del agente esta dado por una distribución de probabilidad $p(s)$ y el intervalo de tiempo en el que el agente interactúa con el entorno está acotado por T, la distribución de las trayectorias tomadas por el agente queda dada por la Ecuación 2.1:

$$p_\pi(s_1, a_1, ..., s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{2.1}$$

El aprendizaje reforzado tiene como fin encontrar una política que maximice el retorno que recibe el agente con respecto a las trayectorias que visita en su interacción con el ambiente. Dicha interacción con el entorno puede finalizar en un estado terminal, en cuyo caso la secuencia de estados recorridos se denomina como episodio. El estado inicial y terminal de un episodio determinan el inicio y el fin de este y no son necesariamente distintos de los otros estados en los que se pueda encontrar un agente. Los problemas episódicos están acotados por un horizonte temporal $T < \infty$, por lo que el valor del retorno que recibe un agente queda completamente definido por la suma de las recompensas obtenidas por el agente a partir de un cierto pase de tiempo:

$$R_t = \sum_{k=t}^{T} r_k \tag{2.2}$$

Existen ambientes donde no se tienen estados terminales, por lo que las trayectorias que siguen los agentes en estos no están acotadas temporalmente. En estos casos, si se mantiene la formulación de retornos según la Ecuación 2.2, el problema podría no tener solución, pues el valor esperado del retorno podría ser infinito. Es por esto que se utiliza un factor de descuento $\gamma \in [0, 1]$ que pondera las recompensas obtenidas por el agente en cada instante de tiempo. La introducción de este factor de descuento permite definir un retorno acotado cuando $T = \infty$ suponiendo que $\gamma < 1$ y logra establecer un orden de importancia a las recompensas instantáneas. Un valor de γ más cercano a 1 le otorga mayor relevancia a recompensas futuras mientras que un valor más cercano a 0 promueve el aprendizaje de comportamientos que maximizan recompensas inmediatas. Así, el retorno que recibe un agente en su interacción con el ambiente queda definido por la Ecuación 2.3:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2.3}$$

En base a la distribución de trayectorias que sigue el agente (Ecuación 2.1) y a la definición del retorno que recibe en su interacción con el ambiente (Ecuaciones 2.2, 2.3), se define que el objetivo del aprendizaje reforzado es encontrar una política que maximice el retorno obtenido por el agente en la secuencia de estados recorridos. Formalmente:

$$J_{\text{RL}}(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t) \right] \tag{2.4}$$

Con el objetivo de encontrar dicha política, el agente debe considerar los retornos futuros que recibirá a partir de las acciones que tomará en cada instante de tiempo. Esto lo logra mediante la función de valor. La función de valor, $V^\pi(s)$, nos entrega el valor esperado del retorno obtenido tras ejecutar una trayectoria de estados según una política π. Partiendo de un cierto estado s, la función de valor se define entonces por la Ecuación 2.5:

$$V^\pi(s) = \mathbb{E}_{\tau_t \sim \pi(\tau_t)} \left[\sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k) | s_t = s \right] \tag{2.5}$$

Similarmente, se puede definir la función de valor del par estado-acción $Q^\pi(s, a)$. Esta entrega el valor esperado de los retornos dado un estado inicial s, la elección de una acción a y el seguimiento de trayectorias según la política π:

$$Q^\pi(s, a) = \mathbb{E}_{\tau_t \sim \pi(\tau_t)} \left[\sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k) | s_t = s, a_t = a \right] \tag{2.6}$$

Notando que el aprendizaje reforzado busca maximizar la función de valor, se puede decir que su objetivo es en consecuencia, encontrar una política óptima π^* tal que se maximice $V^\pi(s)$ y $Q^\pi(s, a)$. Formalmente, se busca encontrar:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.7}$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{2.8}$$

2.1.3. Aprendizaje Reforzado Profundo

A medida que aumenta la dimensión de estados y de acciones, la formulación clásica del aprendizaje reforzado se ve limitada. Es por esto que recientemente se ha empezado a combinar el aprendizaje profundo (*deep learning*) con el aprendizaje reforzado. El *Deep Reinforcement Learning* (DRL) se caracteriza por el uso de redes neuronales artificiales como aproximadores funcionales que permiten manejar de manera eficaz grandes volúmenes de información de alta dimensionalidad. Con esto, se lograron resolver complejas tareas de toma de decisiones para las que una propuesta empleando aprendizaje reforzado clásico no había tenido buen desempeño. El primer trabajo que popularizó el uso de redes neuronales en aprendizaje reforzado fue Deep Q-Network [18].

Dentro de las posibles arquitecturas de redes neuronales, se encuentran las redes neuronales recurrentes. Estas son utilizadas generalmente para encontrar patrones en secuencias de datos y se caracterizan por poseer conexiones recurrentes o de retroalimentación [19]. Una de las estructuras de redes recurrentes más utilizada es la conocida como *Gated Recurrent Unit* (GRU) [20]. Esta se caracteriza por poseer dos celdas principales (*update gate* y *reset gate*) que deciden qué información es relevante para pasar hacia las capas de salida. Lo especial de ellas es que pueden entrenarse para conservar información "histórica", por lo que son muy útiles cuando se desean realizar largas predicciones hacia el futuro.

Las redes neuronales convolucionales (CNN, por sus siglas en inglés) [21] se caracterizan principalmente por su capacidad de extraer información a partir de imágenes. Esta arquitectura es utilizada en el aprendizaje reforzado profundo cuando las observaciones que recibe un agente son imágenes. La redes convolucionales extraen las características relevantes en una imagen realizando convoluciones con filtros (*kernels*). De esta forma, se logra reducir la dimensionalidad de las observaciones considerablemente y el número de parámetros a entrenar, optimizando el aprendizaje.

Una red neuronal interesante y con múltiples aplicaciones en algoritmos de aprendizaje reforzado para interpretar observaciones es PointNet [22]. Esta red está diseñada para realizar clasificación o segmentación semántica de nubes de puntos 3D definidos de la forma $\{p_j\}_{j=1}^{m}$, donde cada elemento p_j tiene coordenadas $(x_j, y_j, z_j) \in \mathbb{R}^3$. Dicho conjunto de puntos presenta tres propiedades fundamentales: es invariante a permutaciones, sus puntos pertenecen a un espacio que tiene una métrica de distancia asociada (permitiendo capturar estructuras locales) y es invariante a transformaciones. Dicho esto, PointNet es una red de gran utilidad para obtener información de nubes de puntos.

A modo de ejemplo, una de las tareas difíciles a controlar en el campo de la robótica que fue resuelta empleando DRL es la manipulación de objetos. A grandes rasgos, esta consiste en coordinar un brazo robótico para realizar tareas precisas y complejas, como coger y colocar objetos, ensamblar componentes e incluso cocinar. Dentro de los primeros trabajos que utilizó aprendizaje reforzado profundo para este tipo de tareas, fue el expuesto en [23], donde por medio de redes

convolucionales se logró coordinar un agente cuyas observaciones eras imágenes monoculares para que fuera capaz de agarrar objetos con precisión.

Otro trabajo que promovió el gran desempeño de los algoritmos de aprendizaje reforzado profundo fue [24]. En dicha investigación, se propuso un algoritmo con el cual un agente aprendió una política exitosa directamente a partir de entradas sensoriales de alta dimensión. Dicho algoritmo se puso a prueba en el desafiante dominio de los clásicos juegos de Atari 2600 [25]. El agente fue capaz de superar el rendimiento de todos los algoritmos anteriores y alcanzar un nivel comparable al de un jugador humano profesional en un conjunto de 49 juegos. Este trabajo tiende un puente entre las entradas sensoriales de alta dimensión y las acciones, lo que da como resultado el primer agente artificial capaz de aprender a sobresalir en una amplia gama de tareas desafiantes.

Así, dependiendo del algoritmo a utilizar, las redes neuronales permiten representar funciones de valor, parametrizar políticas o modelos del ambiente. Por ejemplo, en algoritmos como el *Asynchronous Advantage Actor-Critic* (A3C) se hace uso de una red neuronal para parametrizar una distribución que define una política y para estimar una función de valor [15]. Este método permitió paralelizar su entrenamiento de manera eficiente y lograr mejores resultados que otros algoritmos.

2.2. Algoritmos Model Free y Model Based

Dentro de los algoritmos de aprendizaje reforzado encontramos dos categorías: *Model Free* y *Model Based*. Estos se diferencian en que los primeros no aprenden una representación de la dinámica del MDP en el cuál se encuentra el agente, mientras que los que pertenecen a la segunda categoría sí.

Los algoritmos de tipo *Model Free* buscan, a partir de la interacción entre el agente y el ambiente, aprender una política, sin tener en cuenta la dinámica específica de dicha interacción. En general, estos requieren una gran cantidad de interacciones agente ambiente.

Los algoritmos *Model Based* se caracterizan por el aprendizaje del modelo del ambiente. Después de repetidas interacciones entre un agente y su entorno, las transiciones son registradas en un *dataset* de experiencias $D = \{(s_t, a_t, a_{t+1})\}$ que luego son utilizadas para obtener una función que caracteriza la dinámica del entorno.

Estudios recientes han mostrado que el uso de algoritmos *Model Based* entrega mejores resultados que los *Model Free* en ciertas tareas de control automático [8]. Esto se debe a que son más eficientes al momento de muestrear experiencias [26], entregan una convergencia más asintótica y dan soluciones más robustas [27].

En el contexto de los algoritmos *Model Based*, se encuentran los llamados *World Models* [28]. Estos son representaciones del ambiente, aprendidas por los agentes en base a un conjunto de experiencias previamente recolectadas. En estos modelos, los agentes tienen sensores que comprimen las observaciones en representaciones de la información obtenida. Por medio de una componente de memoria, pueden hacer predicciones sobre estados futuros basados en información histórica. Además, los agentes tienen una componente de toma de decisiones que decide las acciones a ejecutar basándose solamente en representaciones creadas por sus observaciones y su memoria.

Mediante los *World Models*, los robots pueden imaginar trayectorias que corresponden a espacios latentes donde se predicen estados, acciones y recompensas. Por medio de esta "imaginación", los agentes logran aprender una política para una tarea determinada.

2.3. Aprendizaje de *World Models*

Deep Planning Network (PlaNet) [1] es un algoritmo *Model Based* que propone un agente que aprende la dinámica del entorno en el cual está inmerso mediante imágenes y elige acciones a través de una planificación en espacios latentes. El modelo del mundo (*World Model* permite predecir con precisión las recompensas en una secuencia de tiempo. PlaNet ha demostrado muy buenos resultados en múltiples tareas de control. A continuación se presentan sus principales contribuciones.

2.3.1. Planificación en espacios latentes

Para modelar la dinámica del entorno de ejecución, PlaNet se basa en experiencias previas del agente. Con esto, se define un POMDP donde en cada instante de tiempo t, estados ocultos s_t, observaciones de imágenes o_t. vectores de acciones continuas a_t y recompensas escalares r_t siguen una dinámica estocástica:

$$\begin{aligned} \textit{Transition function: } & s_t \sim p(s_t|s_{t-1}, a_{t-1}) \\ \textit{Observation function: } & o_t \sim p(o_t|s_t) \\ \textit{Reward function: } & r_t \sim p(r_t|s_t) \\ \textit{Policy: } & a_t \sim p(a_t|o_{\leq t}, a_{<t}) \end{aligned}$$

PlaNet aprende un modelo de transición, de observación y de recompensas a partir de la experiencia recolectada por el agente. Además, con el objetivo de inferir aproximaciones de observaciones en estados ocultos, mediante el algoritmo, el agente aprende un codificador (*encoder*) $q(s_t|o_{\leq t}, a_{<t})$.

Como el agente no visita inicialmente todo el ambiente, se recolecta iterativamente nueva experiencia en el algoritmo para refinar el modelo del mundo. Inicialmente, el robot recolecta S episodios actuando con una política aleatoria y cada C actualizaciones de parámetros, se agrega un episodio al *dataset* de experiencias. Al momento de recolectar nuevos episodios, se agrega un pequeño ruido Gaussiano a la selección de acciones con el fin de añadir cierta exploración. Para reducir el horizonte de planificación y entregar una señal de aprendizaje más limpia al modelo, cada acción se repite R veces. El Algoritmo 1 detalla con mayor precisión el bucle de entrenamiento.

Algorithm 1 Deep Planning Network (PlaNet)

Input:
C: número de actualizaciones de parámetros.
T: cantidad de interacciones agente ambiente.
R: cantidad de ejecuciones de la acción seleccionada con el entorno.
Inicializar *dataset* D con S episodios aleatorios.
Inicializar parámetros θ del modelo aleatoriamente.
while not converged **do**
`// Ajuste del modelo`
for $s = 1, ..., C$ **do**
Muestrear secuencias $\{(a_t, o_t, r_t)_{t=k}^{k+L}\}_{i=1}^{B} \sim \mathcal{D}$ aleatoriamente del *dataset* de experiencias.
Calcular función de pérdida $\mathcal{L}(\theta)$ a partir de la Ecuación 2.9.
Ajustar parámetros del modelo $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$
end for
`// Recolección de experiencias`
$o_1 \leftarrow$ `env.reset()`
for $t = 1, ..., [\frac{T}{R}]$ **do**
Inferir estado oculto a partir del estado actual $q(s_t|o_{\leq t}, a_{<t})$ de los registros.
$a_t \leftarrow$ `planner`$(q(s_t|o_{\leq t}, a_{<t}), p)$
Agregar ruido de exploración $\epsilon \sim p(\epsilon)$ a las acciones.
for $k = 1, ..., R$ **do**
$r_t^k, o_{t+1}^k \leftarrow$ `env.step`(a_t)
end for
$r_t, o_{t+1} \leftarrow \sum_{k=1}^{R} r_t^k, o_t^k$
end for
$\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^{T}\}$
end while

2.3.2. Recurrent State Space Model

Con el objetivo de poder evaluar múltiples secuencias de acciones en cada intervalo de tiempo, se usa un modelo de estado-espacio recurrente (RSSM, por sus siglas en inglés). Este permite predecir secuencias de acciones a partir de estados latentes s_t. Así, para lograr una planificación exitosa en una imaginación latente, el RSSM requiere tanto de una componente estocástica como de una determinista en su modelo de transición.

Para el modelo, se consideran secuencias $\{o_t, a_t, r_t\}_{t=1}^{T}$ en instantes de tiempo discretos t, observaciones de imágenes o_t, vectores de acciones continuas a_t y recompensas escalares r_t. Un modelo típico de espacio-estado se muestra en la Figura 2.2b y se asemeja a un proceso de Markov parcialmente observable. Este define procesos generativos de imágenes y recompensas usando una secuencia de estados ocultos $\{s_t\}_{t=1}^{T}$:

$$\textit{Transition model: } s_t \sim p(s_t|s_{t-1}, a_{t-1})$$
$$\textit{Observation model: } o_t \sim p(o_t|s_t)$$
$$\textit{Reward model: } r_t \sim p(r_t|s_t)$$

El modelo de transición es una distribución Gaussiana con media y varianza parametrizada por una red neuronal, el modelo de observación también queda definido por una distribución Gaussiana con media parametrizada por una red deconvolucional y una matriz identidad como covarianza y el modelo de recompensa, al igual que en los casos anteriores, sigue también una distribución normal escalar, con media parametrizada por una red neuronal y varianza unitaria.

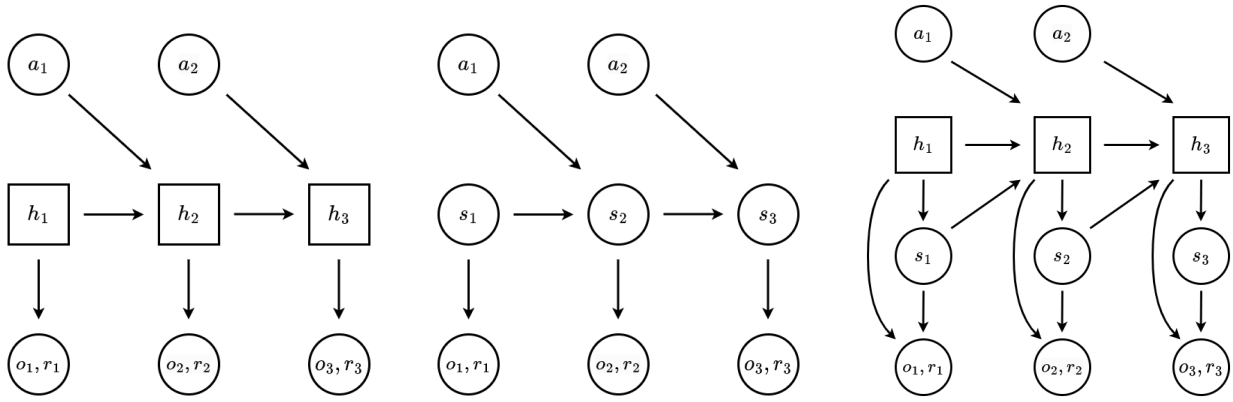
Debido a que el modelo no es lineal, no se pueden obtener directamente los estados posteriores necesarios para la optimización de parámetros. Es por esto, que se utiliza un codificador $q(s_{1:T}|o_{1:T}, a_{1:T}) = \prod_{t=1}^{T} q(s_t|s_{t-1}, a_{t-1}, o_t)$ que infiere aproximaciones de estados posteriores a partir de observaciones y acciones pasadas. Donde $q(s_t|s_{t-1}, a_{t-1}, o_t)$ sigue una distribución normal con media y varianza parametrizadas por una red convolucional. Se usa la distribución posterior condicionada por observaciones pasadas debido a que se desea utilizar el modelo para planificar trayectorias.

Por medio del codificador, se construye una cota variacional para el logaritmo de la función de verosimilitud de los datos, lo cual permite definir la función de pérdida del modelo de transición. Esta cota variacional se obtiene a partir de la desigualdad de Jensen (ver Ecuación 2.9):

$$\begin{aligned}\ln p(o_{1:T}|a_{1:T}) &\triangleq \ln \int \prod_t p(s_t|s_{t-1}, a_{t-1}) p(o_t|s_t) ds_{1:T} \\ &\geq \sum_{t=1}^{T} \Bigg(\mathbb{E}_{q(s_t|o_{\leq t}, a_{<t})} \left[\ln p(o_t|s_t) \right] - \\ &\hookrightarrow \mathbb{E}_{q(s_{t-1}|o_{\leq t-1}, a_{<t-1})} \left[\mathrm{KL}[q(s_t|o_{\leq t}, a_{<t}) || p(s_t|s_{t-1}, a_{t-1})] \right] \Bigg)\end{aligned} \tag{2.9}$$

La componente determinista del RSSM corresponde a una secuencia de vectores de activación $\{h_t\}_{t=1}^{T}$, que permiten al modelo acceder no solo al último estado latente, sino que también a todos los estados previos de manera determinista. Con esto, el *Recurrent State Space Model* queda definido de la siguiente forma:

$$\textit{Deterministic state model: } h_t = f(h_{t-1}, s_{t-1}, a_{t-1}) \tag{2.10}$$
$$\textit{Stochastic state model: } s_t \sim p(s_t|h_t) \tag{2.11}$$
$$\textit{Observation model: } o_t \sim p(o_t|h_t, s_t) \tag{2.12}$$
$$\textit{Reward model: } r_t \sim p(r_t|s_t) \tag{2.13}$$



(a) Deterministic State Model **(b)** Stochastic State Model **(c)** Recurrent Space-State Model

Figura 2.2: Modelos de la Dinámica Latente (adaptados de [1]).

Donde $f(h_{t-1}, s_{t-1}, a_{t-1})$ se implementa como una red neuronal recurrente (RNN, por sus siglas en inglés). La Figura 2.2 esquematiza dichos modelos.

2.4. Dreamer

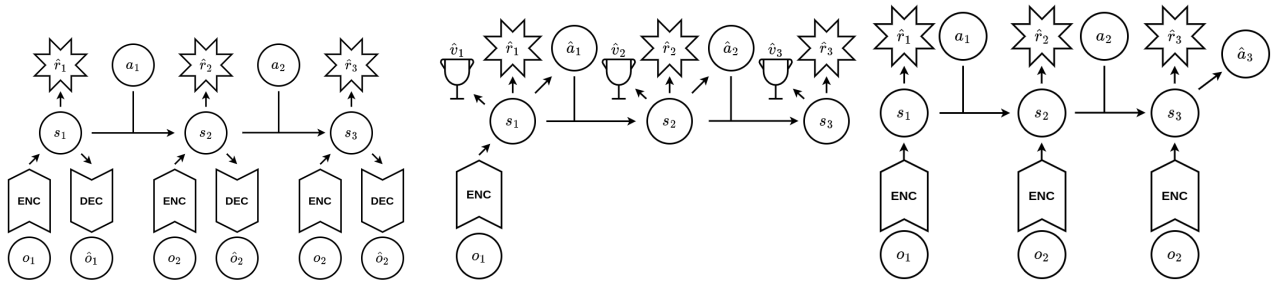
El algoritmo *Dream to Control* [2], es un método que utiliza los *World Models* como herramienta de aprendizaje para que un agente pueda lograr tareas complejas. Este algoritmo ha demostrado excelentes resultados en tareas de control clásico, pues permite un aprendizaje eficiente en cuanto al uso de experiencias, reduce el tiempo computacional de convergencia y logra aprendizaje incluso en línea [29]. A continuación se explican sus principales fundamentos.

2.4.1. Control con *World Models*

Dreamer es un algoritmo de aprendizaje reforzado que aprende comportamientos en un largo horizonte a partir de trayectorias hipotéticas recreadas por espacios latentes compactos de su *world model*. Para lograr su objetivo, *dreamer* sigue tres procesos principales que se pueden ejecutar en paralelo y repetir hasta que el agente logre completar su tarea:

1. *Dynamic Learning*: aprendizaje de la dinámica latente del entorno en la cual el agente se encuentra (*world model*) a partir de un *dataset* de experiencias recolectadas previamente. Estos espacios latentes permiten predecir futuras recompensas a partir de acciones y observaciones anteriores.
2. *Behavior Learning*: aprendizaje de un comportamiento mediante un algoritmo de actor crítico [1]basado en predicciones generadas por el *world model*.
3. *Environment Interaction*: ejecución de dichos comportamientos en el entorno para obtener nuevas experiencias.

[1]Los algoritmos actor-crítico son aquellos que aproximan conjuntamente $V(s)$ o $Q(s, a)$ y una política $\pi(a|s)$ [15].



(a) Deterministic State Model **(b)** Stochastic State Model **(c)** Recurrent Space-State Model

Figura 2.3: Procesos que sigue el algoritmo *Dream to Control* (adaptados de [2]).

La Figura 2.3 esquematiza estos tres procesos. A la izquierda un agente aprende a codificar observaciones y acciones en espacios latentes, y predice recompensas (Figura 2.3a). Al centro, *Dreamer* predice valores y acciones que maximizan futuras predicciones de la función de valor en trayectorias imaginadas (Figura 2.3b). A la derecha, el agente recibe observaciones, codifica estados y predice acciones para ejecutarlas en el entorno real (Figura 2.3c).

El modelo de la dinámica latente que el agente usa consiste en tres componentes principales, modelo de representación, modelo de transición y modelo de recompensas. El *representation model* se obtiene a partir de transformaciones de las observaciones y acciones mediante *encoders* que se compactan en estados latentes s_t que siguen transiciones Markovianas. El *transition model* predice futuros estados del modelo sin acceder a sus observaciones que los producirían. El *reward model* predice las recompensas a partir de los estados del modelo. Formalmente, estos modelos se definen según la Ecuación 2.14:

$$\begin{aligned} &\textit{Representation model: } p(s_t|s_{t-1}, a_{t-1}, o_t) \\ &\textit{Transition model: } q(s_t|s_{t-1}, a_{t-1}) \\ &\textit{Reward model: } q(r_t|s_t) \end{aligned} \tag{2.14}$$

En las ecuaciones anteriores p hace referencia a las distribuciones que generan estados latentes a partir de muestras del entorno de ejecución real, mientras que q alude a aproximaciones que permiten la imaginación latente, pues el modelo de transición permite predecir trayectorias en un horizonte definido sin tener que imaginar u observar las imágenes que las generarían. Así, no se requiere mucha memoria y se pueden hacer predicciones de manera eficaz imaginando múltiples trayectorias en paralelo.

2.4.2. Aprendizaje de política mediante imaginación latente

La dinámica latente aprendida por *Dreamer* define un MDP completamente observable, pues los estados latentes s_t siguen transiciones "Markovianas". Se denotará con índice de tiempo τ los elementos imaginados. Las trayectorias imaginadas inician en verdaderos estados del modelo s_t, pues provienen de una observación del *dataset* de experiencias. Estas trayectorias siguen las predicciones que entrega el modelo de transición $s_\tau \sim q(s_\tau|s_{\tau-1}, a_{\tau-1})$, el modelo de recompensas

$r_\tau \sim q(r_\tau|s_\tau)$ y una política $a_\tau \sim q(a_\tau|s_\tau)$. El objetivo es maximizar la suma descontada de recompensas imaginadas con respecto a la política: $\mathbb{E}_q[\sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau]$.

Considerando las trayectorias imaginadas en un horizonte finito H, *Dreamer* usa un modelo actor crítico para aprender la política. De esta forma, se obtiene un modelo de acciones y de funciones de valor en el espacio latente del *world model*. El modelo de acciones permite hacer predicciones para resolver la tarea designada en el entorno imaginado, mientras que el modelo de la función de valor estima las recompensas imaginadas obtenidas según lo indicado por el modelo de acciones. Formalmente:

$$\begin{aligned} &\textit{Action model: } a_\tau \sim q_\phi(a_\tau|s_\tau) \\ &\textit{Value model: } v_\psi(s_\tau) \approx \mathbb{E}_{q(\cdot|s_\tau)} \left[\sum_{\tau=t}^{t+H} \gamma^{\tau-t} r_\tau \right] \end{aligned} \tag{2.15}$$

Ambos modelos son entrenados de manera complementaria como en un algoritmo de *policy iteration*, donde el *action model* busca maximizar una estimación del valor, mientras que el *value model* busca hacer coincidir la estimación del valor que cambia según lo que indica el modelo de acciones.

Se utilizan redes neuronales tanto para el modelo de acciones, como para el modelo de la función de valor con parámetros ϕ y ψ respectivamente. El actor entrega como salida un vector obtenido a partir de una muestra obtenida de una distribución Gaussiana cuyos parámetros son generados por la red neuronal a la cual se le aplica una transformación tangente hiperbólica:

$$a_\tau = \tanh(\mu_\phi(s_\tau) + \sigma_\phi(s_\tau)\epsilon), \tag{2.16}$$

donde $\epsilon \sim \text{Normal}(0, \mathbb{I})$.

Las estimaciones del valor que toman los estados de las trayectorias imaginadas $\{s_\tau, a_\tau, r_\tau\}_{\tau=t}^{t+H}$ se obtienen mediante la siguiente expresión:

$$V_\lambda(s_\tau) = (1-\lambda) \sum_{n=1}^{H-1} \lambda^{n-1} V_N^n(s_\tau) + \lambda^{H-1} V_N^H(s_\tau) \tag{2.17}$$

Donde $V_N^k(s_\tau) = \mathbb{E}_{q_\theta, q_\phi}[\sum_{n=\tau}^{h-1} \gamma^{n-\tau} r_n + \gamma^{h-\tau} v_\psi(s_h)]$, con $h = \text{mín}(\tau + k, t + H)$. Este término, estima las recompensas k pasos en el futuro con el modelo de la función de valor aprendido. *Dreamer* utiliza V_λ, que es un promedio de estimaciones de diferentes k ponderado exponencialmente para regularizar el *bias* y la varianza.

Para actualizar los modelos de acciones y función de valor, en primer lugar se obtienen las estimaciones de valor mediante $V_\lambda(s_\tau)$ para todos los estados s_τ de la trayectoria imaginada. El objetivo del *action model* $q_\phi(a_\tau|s_\tau)$ es predecir acciones que entregan estados con un alto valor

según las estimaciones. El objetivo del *value model* $v_\psi(s_\tau)$, por su parte, es actualizarse según los valores estimados. Formalmente:

$$\begin{aligned} a_\tau \sim q_\phi(a_\tau|s_\tau) &\longleftarrow \max_\phi \mathbb{E}_{q_\theta,q_\phi}\left[\sum_{\tau=t}^{t+H} V_\lambda(s_\tau)\right] \\ v_\psi(s_\tau) &\longleftarrow \min_\psi \mathbb{E}_{q_\theta,q_\phi}\left[\sum_{\tau=t}^{t+H} \frac{1}{2}\|v_\psi(s_\tau) - V_\lambda(s_\tau)\|^2\right] \end{aligned} \tag{2.18}$$

2.4.3. Aprendizaje de la dinámica latente

Para poder aprender políticas mediante imaginación se requiere de un modelo del entorno que tenga la capacidad de generalizar correctamente. El algoritmo se centra en *World Models* capaces de predecir largas secuencias de espacios latentes y que permitan al agente imaginar muchas trayectorias en paralelo. Para poder obtener esta representación del entorno se proponen tres métodos: *reward prediction*, *image reconstruction* y *contrastive estimation*.

- *Reward prediction*: como se presentó en la Sección 2.4.1, la imaginación latente requiere de un modelo de representación $p(s_t|s_{t-1}, a_{t-1}, o_t)$, de un modelo de recompensas $q(r_t|s_t)$ y de transición $q(s_t|s_{t-1}, a_{t-1})$. Esto se puede lograr mediante la predicción de futuras recompensas dadas acciones y observaciones.
- *Image reconstruction*: este método se basa en PlaNet [1], donde el *World Model* consiste en las siguientes componentes:

$$\begin{gathered} \textit{Representation model: } p_\theta(s_t|s_{t-1}, a_{t-1}, o_t) \\ \textit{Observation model: } q_\theta(o_t|s_t) \\ \textit{Reward model: } q_\theta(r_t|s_t) \\ \textit{Transition model: } q_\theta(s_t|s_{t-1}, a_{t-1}) \end{gathered} \tag{2.19}$$

  Estas componentes son optimizadas en conjunto para maximizar la *"variational information bottleneck"*. Esta cota incluye términos de la reconstrucción de las observaciones, recompensas y un regularizador KL. La esperanza es tomada en el *dataset* y modelo de representación:

$$\mathcal{J}_{REC} = \mathbb{E}_p\left[\sum_t (\mathcal{J}_O^t + \mathcal{J}_R^t + \mathcal{J}_D^t)\right] + const$$

  donde $\mathcal{J}_O^t = \ln q(o_t|s_t)$, $\mathcal{J}_R^t = \ln q(r_t|s_t)$ y $\mathcal{J}_D^t = -\beta \mathrm{KL}(p(s_t|s_{t-1}, a_{t-1}, o_t)||q(s_t|s_{t-1}, a_{t-1}))$.
- *Contraestive estimation:* para fomentar la información mutua entre los estados del modelo y las observaciones, se predicen estados a partir de imágenes. De esta forma se reemplaza el *observation model* por un *state model*:

$$\textit{State model: } q_\theta(s_t|o_t) \tag{2.20}$$

  El estado marginal que se utiliza en la reconstrucción del objetivo (para el caso de las observaciones se asumía constante) puede ser estimado usando *noise contrastive estimation* (NCE).

Este método promedia el estado sobre todas las observaciones o' del batch que contiene la secuencia. De esta forma:

$$\mathcal{J}_{NCE} = \mathbb{E}\left[\sum_t (\mathcal{J}_S^t + \mathcal{J}_R^t + \mathcal{J}_D^t)\right] \tag{2.21}$$

donde $\mathcal{J}_S^t = \ln q(s_t|o_t) - \ln(\sum_{o'} s(s_t|o'))$.

2.4.4. Algoritmo

El Algoritmo 2 expone el pseudo-código de *Dreamer*. En este, se encuentran sus principales componentes: el modelo de representación $p_\theta(s_t, |s_{t-1}, a_{t-1}, o_t)$, de transición $q_\theta(s_t|s_{t-1}, a_{t-1})$, de recompensa $q_\theta(r_t|s_t)$, de acción $q_\phi(a_t|s_t)$ y de valor $v_\psi(s_t)$.

Los hiperparámetros que se definen en *Dreamer* son la cantidad de *seed episodes* S, la cantidad de *update steps* C, el *batch size* B, el largo de las secuencias L, el horizonte de imaginación H y la tasa de aprendizaje α.

Algorithm 2 Dreamer

Inicializar *dataset* $\mathcal{D}$ con S episodios aleatorios.
Inicializar parámetros θ, ϕ, ψ de redes neuronales aleatoriamente.
while *not converged* **do**
  for *update step* $c = 1, ..., C$ **do**
    `// Aprendizaje dinámica del mundo`
    Muestrear B secuencias de datos $\{(a_t, o_t, r_t)\}_{t=k}^{k+L} \sim \mathcal{D}$.
    Calcular estados $s_t \sim p_\theta(s_t|s_{t-1}, a_{t-1}, o_t)$
    Actualizar θ usando *representation learning*.
    `// Aprendizaje de la política`
    Imaginar trayectorias $\{(s_\tau, a_\tau\}_{\tau=t}^{t+H}$ de cada s_t.
    Predecir recompensas $E(q_\theta(r_\tau|s_\tau))$ y valores $v_\psi(s_\tau)$.
    Calcular estimaciones de valor $V_\lambda(s_\tau)$.
    Actualizar $\phi \leftarrow \phi + \alpha\nabla_\phi \sum_{\tau=t}^{t+H} V_\lambda(s_\tau)$.
    Actualizar $\psi \leftarrow \psi - \alpha\nabla_\psi \frac{1}{2}||v_\psi(s_\tau) - V_\lambda(s_\tau)||^2$.
  end for
  `// Interacción con el entorno`
  $o_1 \leftarrow$ `env.reset()`
  for time step $t = 1, ..., T$ **do**
    Calcular $s_t \sim p_\theta(s_\theta|s_{t-1}, a_{t-1}, o_t)$ del historial.
    Calcular $a_t \sim q_\phi(a_t|s_t)$ con el modelo de actor.
    Agregar ruido de exploración a la acción.
    $r_t, o_{t+1} \leftarrow$ `env.step`(a_t)
  end for
  Agregar experiencia al *dataset* $\mathcal{D} \leftarrow \mathcal{D} \cup \{(a_t, o_t, r_t)\}_{t=1}^{T}$
end while

2.5. Entornos de ejecución DeepMind

Con los grandes avances de la investigación en aprendizaje reforzado, han surgido diferentes conjuntos de entornos de ejecución con variadas tareas a controlar. Estos han permitido comparar y establecer puntos de referencia sobre el desempeño de los algoritmos de aprendizaje reforzado. A modo de ejemplo, el *Arcade Learning Environment* (ALE) [25], provee una interfaz con cientos de entornos de juegos Atari 2600, cada uno diseñado para desafiar jugadores humanos. En este contexto, *DeepMind Control Suite* [17] propone un conjunto de problemas de control continuo que permiten establecer puntos de referencia para los algoritmos de *reinforcement learning*.

Esta agrupación de entornos de ejecución se focaliza exclusivamente en tareas de control continuo. La estructura que posee la función de recompensa, las acciones y observaciones de estos ambientes ofrece curvas de aprendizaje fáciles de interpretar, lo que permite establecer métricas para comparar el desempeño de diferentes algoritmos.

El proceso de decisión de Markov que modela matemáticamente estos entornos queda definido por un conjunto de estados $\mathcal{S}$, de acciones $\mathcal{A}$, una función de transición **f(s, a)**, una función de observación **o(a, a)** y una función de recompensa escalar **r(s, a)**. En particular:

- **Estados:** el estado **s** es un vector de número reales $\mathcal{S} = \mathbb{R}^{\dim(\mathcal{S})}$.
- **Acciones:** un vector **a** $\in \mathcal{A} = [-1, 1]^{\dim(\mathcal{A})}$.
- **Observaciones:** la función $o(s, a)$ describe las observaciones posibles del agente. Todas las tareas del conjunto son "fuertemente" observables, es decir, el estado puede ser recreado a partir de una sola observación.
- **Recompensas:** el rango de recompensas de estos entornos de ejecución se mueve en el intervalo unitario **r(s, a)** $\in [0, 1]$.

Todas las tareas a controlar funcionan con el motor de física MuJoCo [30]. En *DeepMind Control Suite*, un **dominio** hace referencia a un modelo físico, mientras que una **tarea** indica una instancia del modelo con un MDP en particular. En este trabajo se utilizan los dominios *Cart-Pole* y *Cheetah* con las tareas *balance* y *run* respectivamente.

Capítulo 3

Estado del Arte

3.1. Navegación autónoma

El problema de navegación autónoma es fundamental en robótica móvil, pues este abarca un gran espectro de aplicaciones que necesitan soluciones robóticas robustas como vehículos autónomos, drones con pilotaje automático o incluso buques. El objetivo general de la navegación autónoma es identificar un camino óptimo desde un punto de partida a una región objetivo (*target*), evadiendo obstáculos, ya sean estos estáticos o dinámicos.

Una propuesta popular para resolver este tipo de problemas consiste en la combinación de diferentes algoritmos. La manera tradicional de afrontar el problema de navegación hace uso de *Simultaneous Localization and Mapping* (SLAM) para construir un mapa del entorno desconocido y luego utilizar algoritmos de localización para determinar la posición actual del agente y moverse hacia su *target* usando *path planning*.

Métodos clásicos de SLAM que usan herramientas visuales presentan dos desafíos principales: (1) lograr extraer características de imágenes de manera efectiva para representar la información que se entrega y (2) la posibilidad de equivocación del algoritmo en caso de movimiento de objetos [31].

Otros métodos de SLAM que hacen uso de láser para adquirir información, presentan nuevos desafíos como el tiempo de procesamiento que consume la actualización del mapa de obstáculos y la necesidad de un láser denso que permita entregar información con precisión.

En el campo de la navegación autónoma los métodos de *path planning* son claves para el desarrollo de algoritmos. Según la cantidad de información del ambiente que obtienen los agentes, los problemas pueden se clasificados en planificación global o planificación local.

La planificación global ocurre cuando el entorno estático es completamente conocido, es decir, el robot logra establecer caminos sin colisionar inmerso en un ambiente estático según un algoritmo específico. La planificación local considera principalmente la habilidad de evasión de obstáculos en un ambiente dinámico. El robot solo conoce una parte del entorno y actualiza la información que posee de este mediante la obtención de señales a través de sensores [32].

Algoritmos tradicionales de *path planning* presentan desafíos importante. En primer lugar, la dificultad de mantener en memoria representaciones de posiciones basados en una grilla. En segundo lugar, el poder computacional necesario para calcular intensamente nuevos caminos en un entorno dinámico en tiempo real, lo que limita la reactividad del agente.

A pesar de que se utilicen técnicas de planificación de movimiento combinadas con mejoras del aprendizaje reforzado u otros métodos clásicos para resolver tareas de navegación, siguen habiendo costos computacionales e ingenieriles muy elevados para obtener soluciones más realistas. Esto se debe a que existe una brecha de realidad entre los simuladores y un entorno de ejecución físico, es difícil atribuir recompensas correctamente, son poco eficientes al momento de muestrear experiencias, entre otros [33]. Es por esto que el uso de aprendizaje reforzado profundo es de gran interés y será presentado en la próxima sección.

3.2. Navegación autónoma utilizando Aprendizaje Reforzado Profundo

Con la gran capacidad de representación que tiene el *deep learning*, nuevas ideas han sido introducidas para abordar tareas de navegación autónoma con complejas señales de sensores como entrada. En particular, estos métodos describen el problema de navegación como un proceso de decisión de Markov parcialmente observable, cuyo objetivo es maximizar el retorno esperado dadas interacciones agente-ambiente, y cuyas observaciones corresponden a la información entregada por un sensor.

Mediante interacción con el entorno, los métodos de aprendizaje reforzado profundo encuentran su política óptima guiando al agente a su región objetivo. Los algoritmos que usan redes neuronales como herramienta de aprendizaje para tareas de navegación autónoma, tienen la ventaja de no necesitar la creación de un mapa y de tener una gran capacidad de aprendizaje sin depender de la precisión de los sensores que posea el robot, logrando un gran desempeño.

No obstante el gran potencial que presentan estas herramientas de aprendizaje supervisado, no ha sido particularmente explotadas para abordar el problema de navegación. En [34] se presentan múltiples algoritmos de aprendizaje reforzado profundo para resolver la navegación visual (una tecnología fundamental para agentes artificiales). Sin embargo, dicha investigación concluye con que aún hay dos grandes desafíos pendientes.

En primer lugar, la dimensionalidad de las señales de entrada de los agentes de navegación introduce la necesidad de acumular una cantidad sustancial de experiencias con el entorno para lograr un aprendizaje efectivo. Este proceso, que se vuelve especialmente prolongado en entornos complejos, plantea interrogantes sobre la eficiencia del entrenamiento y la escalabilidad de los algoritmos. Numerosos estudios han abordado este desafío, proponiendo técnicas como la selección de características relevantes y la reducción de la dimensionalidad mediante técnicas avanzadas de procesamiento de datos [35, 36].

En segundo lugar, la generalización del aprendizaje entre diferentes entornos y, más críticamente, de simuladores a la realidad, sigue siendo un obstáculo importante. Las investigaciones actuales sugieren que los métodos DRL para navegación visual a menudo luchan por adaptarse efectiva-

mente a nuevas dinámicas y complejidades ambientales [34]. Esta falta de generalización plantea preguntas sobre la aplicabilidad de estos enfoques en escenarios del mundo real y destaca la necesidad de estrategias que permitan una transferencia de conocimiento más efectiva.

La capacidad de generalización en el contexto de la navegación autónoma se ha convertido en un tema central de discusión. Pasar de un entorno de entrenamiento a otro y, más crucialmente, de simuladores a la realidad, presenta desafíos sustanciales. Este fenómeno se evidencia en la investigación presentada en [37], que resalta las dificultades persistentes para lograr una transición suave entre diferentes contextos, lo que subraya la necesidad de abordar esta limitación crítica en los enfoques de DRL actuales.

En este contexto, el algoritmo Dreamer [2] emerge como una solución innovadora que aborda de manera efectiva los desafíos mencionados. Dreamer se distingue por su enfoque eficiente en el muestreo de experiencias y su capacidad para generar trayectorias imaginadas, lo que facilita la generalización a través de diferentes contextos. Estas características lo convierten en una herramienta valiosa para superar las limitaciones actuales en términos de escalabilidad y transferencia de conocimiento.

Estos precedentes subrayan la necesidad de investigar y desarrollar enfoques que mejoren la eficiencia del aprendizaje reforzado profundo para la navegación autónoma. En particular, mediante la investigación en el rubro, se puede explotar el gran potencial de aprendizaje y generalización que tienen las redes neuronales para desarrollar métodos que tengan mejor desempeño y se adapten mejor al mundo real. La implementación y evaluación de estas estrategias pueden abrir nuevas perspectivas para superar las limitaciones actuales (que poseen otros algoritmos) y avanzar hacia soluciones más robustas.

Capítulo 4

Navegación monoagente utilizando *Dreamer*

En este capítulo se presenta la metodología adoptada para resolver el problema de navegación monoagente utilizando el algoritmo *Dreamer*. Se expone el proceso de validación del algoritmo en dos entornos clásicos de aprendizaje reforzado, se presenta la formulación matemática del problema de navegación como un *Partially Observable Markov Decision Process* (POMDP) y la implementación realizada adaptando *Dreamer*.

En primer lugar, para confirmar el correcto funcionamiento del algoritmo, se utilizan los entornos *cartpole* y *cheetah* de la librería `dm_control` [17]. En segundo lugar, se adapta el algoritmo *Dreamer* para lograr un aprendizaje en un entorno de navegación simulado en ROS [38].

Mediante el trabajo propuesto, se obtiene una política parametrizada a través de una red neuronal que entrega como acciones los comandos de velocidades continuas para un robot ideal simulado. Dichas acciones se derivan del procesamiento de estados latentes que corresponden a codificaciones de las observaciones del agente. Estas provienen de sensores LiDAR, de la posición de su región objetivo y de estimaciones de velocidad basadas en odometría. Con la política encontrada, se busca que el agente logre llegar a su región objetivo evitando colisionar con el entorno en el que se encuentra.

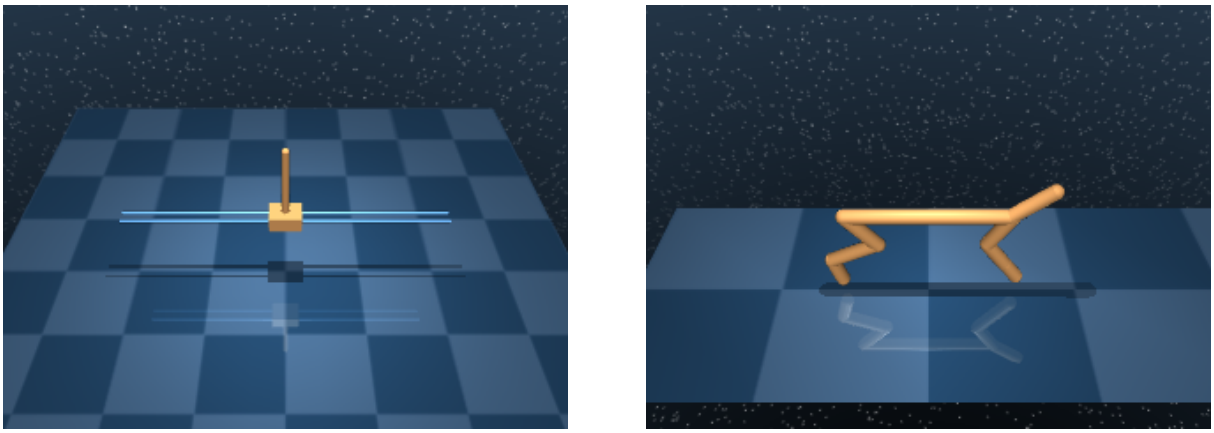
4.1. Dreamer en MuJoCo

Con el objetivo de confirmar el correcto funcionamiento del algoritmo *Dreamer*, se utilizan los entornos de *DeepMind Control Suite*. Esta librería posee un conjunto de ambientes donde se simulan tareas de control continuo de una forma estandarizada. Estas tareas están programadas en Python y utilizan la física de MuJoCo [30]. Los experimentos se diseñan con el fin de comparar el desempeño del algoritmo con los resultados reportados originalmente en [2]. Con esto, se evalúa la capacidad del agente para resolver tareas en un largo horizonte temporal y con acciones continuas.

La configuración de los entornos utilizada al momento de implementar el algoritmo *Dreamer* presenta ciertas diferencias con aquella expuesta en el artículo [2]. En el contexto de este estudio, las observaciones que posee el agente de su entorno son vectores de señales sensoriales que contienen unidades físicas significativas para la tarea a controlar, mientras que las utilizadas en la

implementación original corresponden a imágenes. El desarrollo realizado fue llevado a cabo de esta manera debido a que en el entorno de navegación, las señales que obtiene el robot a partir de sus sensores en cada instante de tiempo, son procesadas para entregar una observación vectorial.

Para el presente trabajo, se utilizaron dos entornos de ejecución, *cartpole* y *cheetah* con las tareas *swingup* y *run* respectivamente. La Figura 4.1 muestra gráficamente dichos entornos. Para ambas configuraciones, las tareas son episódicas, es decir, finalizan luego de transcurrir una cierta cantidad de transiciones de estados (*steps*). En estos ambientes, los episodios comienzan en estados aleatorios y finalizan luego de que el agente haya realizado 1000 *steps*. A continuación se señalan algunos detalles de cada uno.



(a) Cartpole **(b)** Cheetah

Figura 4.1: Entornos de ejecución MuJoCo.

4.1.1. *Cheetah*

En este entorno, el agente es un robot bidimensional que posee 9 eslabones y 8 articulaciones que los conectan. La idea de este entorno de ejecución es encontrar una política capaz de coordinar el torque aplicado a cada una de las articulaciones para lograr que el *cheetah* logre correr hacia adelante lo más rápido posible (Figura 4.1b).

La recompensa que obtiene el agente es linealmente proporcional a su velocidad hasta un valor máximo de 10 metros por segundo, es decir $r(v) = \text{máx}(0, \text{mín}((v/10), 1)$, donde v es la velocidad del *cheetah*. Las observaciones de este robot corresponden a los valores de las posiciones de las distintas partes de su cuerpo, seguido de las velocidades de dichas partes. Las acciones que toma el agente en este entorno, corresponden al torque aplicado a las 6 articulaciones que posee el *cheetah* en sus piernas. La Tabla 4.1 presenta las dimensiones de las componentes del entorno de ejecución *cheetah* con la tarea *run*.

Tabla 4.1: Dimensiones y rango de valores entorno *Cheetah*.

Componente	Dimensión	Rango de valores
Observaciones	17	[-inf, inf]
Acciones	6	[-1, 1]
Recompensa	1	[0, 1]

4.1.2. Cartpole

En esta configuración de ambiente de MuJoCo, el agente tiene como objetivo, controlar las fuerzas que se le aplican a su carro horizontalmente para que el *pole* que se encuentra inicialmente apuntando hacia abajo se disponga hacia arriba y logre mantenerse en posición vertical (Figura 4.1a).

Las observaciones que recibe el agente corresponden a información sobre la posición y la velocidad del carro, y del ángulo que existe entre el *pole* y el carro. La acción que toma el agente corresponde a la velocidad que se aplica horizontalmente. La Tabla 4.2 resume las dimensiones de las componentes del entorno.

Tabla 4.2: Dimensiones y rango de valores entorno *Cartpole*.

Componente	Dimension	Rango de valores
Observaciones	5	[-inf, inf]
Acciones	1	[-1, 1]
Recompensa	1	[0, 1]

4.1.3. Configuración experimental

La implementación del algoritmo *Dreamer* fue realizada en Python 3.8.10 y se utilizó Pytorch 2.0.1 como *framework* para su desarrollo [39]. Cada modelo del algoritmo fue implementado como una red neuronal, seis de estas forman parte del *World Model* 2.10 y las otras dos corresponden al actor y al crítico. A continuación se describen a grandes rasgos dichos modelos.

- ***Encoder***: esta red se encarga de codificar las observaciones del agente en *embeddings* (representaciones vectoriales de información).
- ***Decoder***: representa el modelo de observación del agente. Esta red neuronal parametriza la media de una distribución normal con una desviación estándar unitaria.
- ***Reward Model***: corresponde al modelo de recompensas que utiliza el agente para aprender la dinámica del mundo. Esta red neuronal parametriza la media de una distribución normal con desviación estándar unitaria.
- ***Recurrent Model***: es el modelo que se encarga de aprender a generar estados dererministas, cuyo objetivo es mantener información sobre las transiciones realizadas desde el inicio del episodio hasta el estado actual del agente.
- ***Representation Model***: esta red neuronal toma como entrada el estado determinista y el *embedding* entregado por el codificador y entrega como salida un estado latente que intenta modelar el estado actual del agente por medio de una distribución *posterior*.
- ***Transition Model***: se encarga de realizar las transiciones puramente en estados latentes, es decir, sin acceso directo a la codificación de las observaciones. Esto lo realiza a través de una distribución *prior*.
- ***Actor***: este modelo parametriza la media y desviación estándar de una distribución normal a partir de la cual se muestrean las acciones que toma el agente.
- ***Critic***: esta red busca entregar una estimación del valor esperado del retorno que recibe el agente dado un estado latente. Para ello, la red parametriza la media de una distribución Gaussiana con desviación estándar unitaria.

La Figura 4.2 muestra el flujo de información e interacción que realizan las redes neuronales para aprender el modelo del mundo. Los recuadros en violeta corresponden a las tres componentes de la función de pérdida. En el esquema, MLP hace referencia a *multi-layer perceptron*, GRU a *gated recurrent unit* [20] y ELU a la función de activación utilizada [40]. Al igual que como se señala en el algoritmo originalmente, para aprender la política y función de valor el agente usa un algoritmo de actor-crítico [15] en estados latentes.

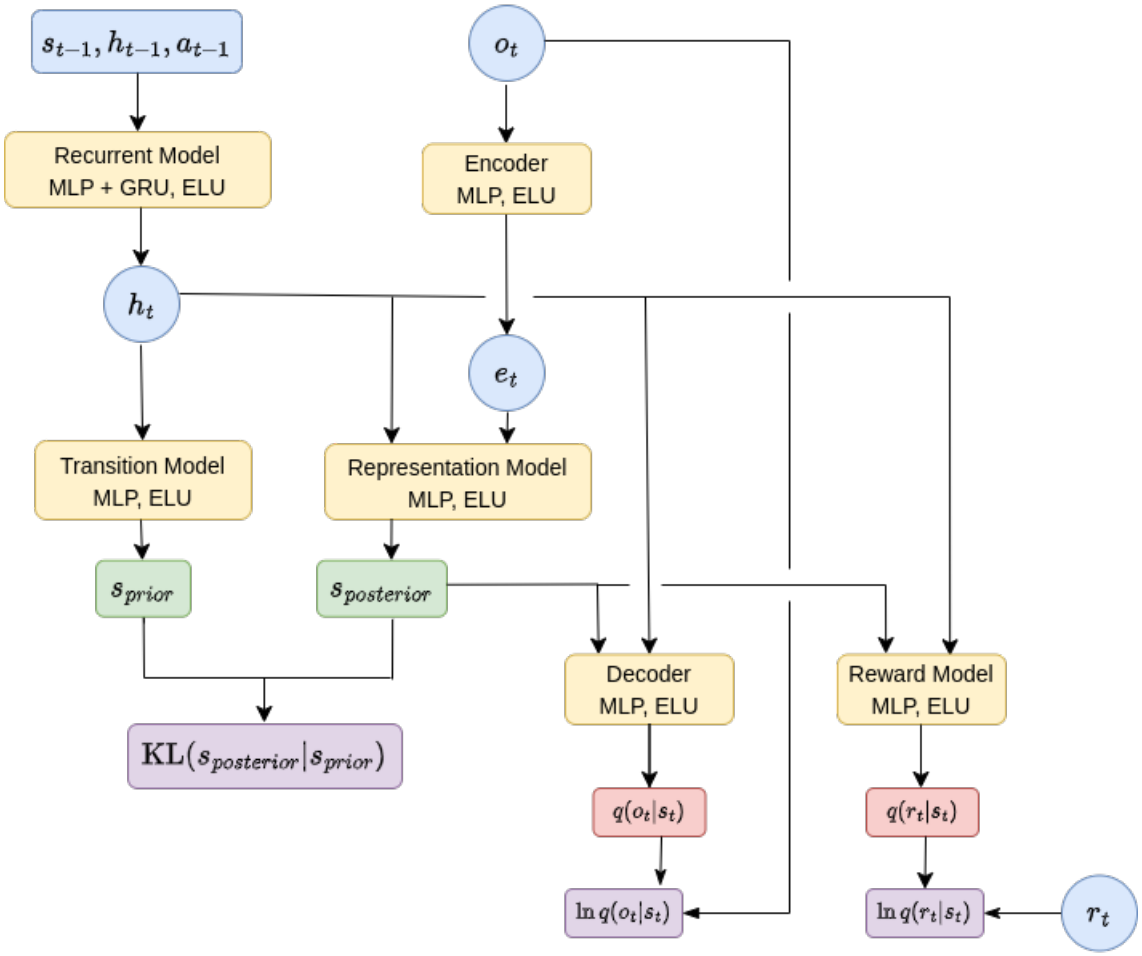


Figura 4.2: Esquema de aprendizaje del mundo en implementación de *Dreamer*.

Todas las redes tienen 400 neuronas en sus capas ocultas, excepto por los modelos de transición, de representación y recurrente, cuyas capas ocultas poseen 200 neuronas. Se utilizan 1000 episodios de entrenamiento y 5 episodios de recolección de experiencias aleatorias para inicializar el *experience replay buffer* [18]. Cada transición de estados es registrada en el *replay buffer* y se utiliza un optimizador *Adam* [41] para realizar 100 actualizaciones de parámetros una vez finaliza un episodio. Por cada episodio de entrenamiento, es decir, por cada 100 actualizaciones de parámetros, se realizan 3 episodios de evaluación. La Tabla 4.3 resume los hiperparámetros utilizados en estos entornos.

Tabla 4.3: Hiperparámetros utilizados para entrenamiento del algoritmo *Dreamer* en entornos de MuJoCo.

Parámetro	Valor
Tamaño máximo de *Experience Replay*	$1 \cdot 10^6$
Tasa de aprendizaje actor	$8 \cdot 10^{-5}$
Tasa de aprendizaje crítico	$8 \cdot 10^{-5}$
Tasa de aprendizaje modelo del mundo	$6 \cdot 10^{-4}$
Factor de descuento (γ)	0.99
Factor λ	0.95
Tamaño de *batch*	50
Dimensión *embedding*	1024
Largo de secuencias	50
Horizonte de planificación	15

4.2. Dreamer para resolver el problema de navegación

La capacidad de navegación es un problema fundamental para robots móviles y por lo tanto, ampliamente estudiada. Su objetivo principal es identificar trayectorias óptimas o sub-óptimas desde un punto inicial hasta una región objetivo (*target*) evitando colisionar con posibles obstáculos del entorno. A medida que la tecnología avanza, entregar soluciones robustas a este problema abre un campo de investigación de gran interés, pues estas pueden ser aplicadas en diversas industrias, como la logística, la minería, la conducción autónoma, entre otros.

El problema de navegación implica dos sub-tareas fundamentales que el agente debe abordar simultáneamente: la primera es alcanzar una región objetivo previamente definida, mientras que la segunda implica evitar colisiones con obstáculos del entorno donde se encuentra inmerso. Más aún, se busca que las trayectorias que unen los puntos de partida del robot y sus respectivas regiones objetivo, sean óptimas.

Dentro de las posibles soluciones del problema, se pueden separar en formulaciones clásicas y soluciones utilizando aprendizaje de máquinas. Siendo estas últimas de gran interés debido a que no dependen del ajuste o calibración de diversos parámetros. Por esta razón, el trabajo realizado hace uso de aprendizaje reforzado para dar una solución robusta al problema de navegación.

En las siguientes secciones, se presenta el caso de estudio en particular y cómo este será abordado e implementado utilizando aprendizaje reforzado. En particular, la Sección 4.2.1 expone el algoritmo a utilizar, los detalles del entorno de ejecución y del robot, la Sección 4.2.2 propone una formulación matemática del problema como un proceso de decisión de Markov parcialmente observable. Finalmente, en la Sección 4.2.3 se explica a grandes rasgos la implementación realizada en Python.

4.2.1. Propuesta

Para resolver el problema de navegación, se propone extender el algoritmo *Dreamer* [2] a un entorno donde se simule un mundo con obstáculos y un robot (ver Algoritmo 2). Mediante la ejecución de los tres procesos principales del algoritmo que permiten el aprendizaje, se espera

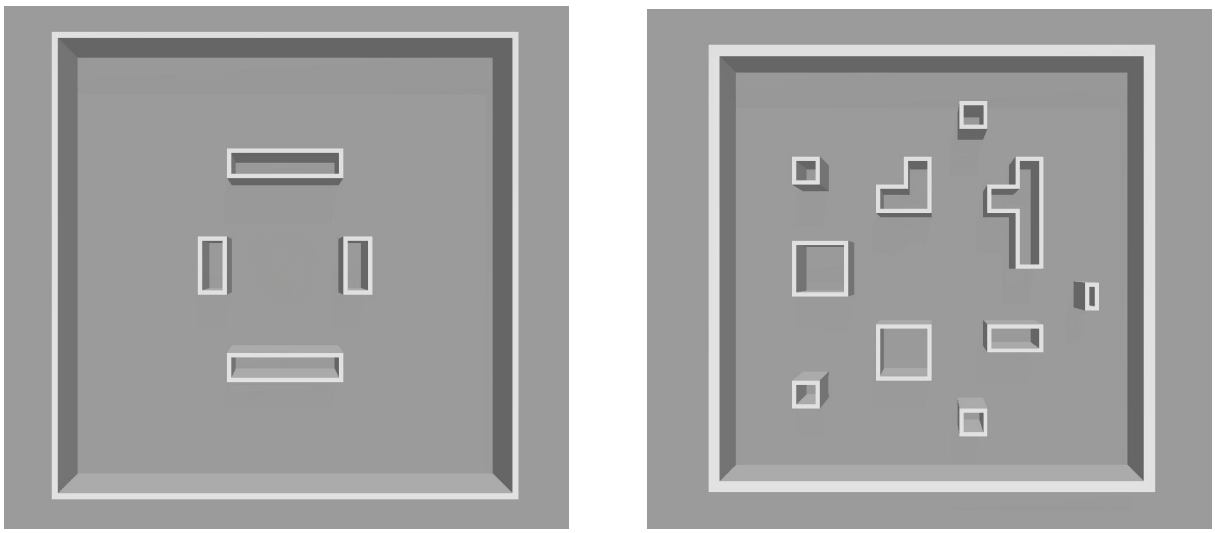
lograr que un agente se desplace hacia una región objetivo (*target*) previamente definida y sin colisionar con obstáculos del mundo,s de manera autónoma.

Los procesos de entrenamiento son realizados en simulaciones usando Gazebo [42] y ROS Noetic (*Robotic Operating System*) [38]. Con el objetivo de reducir las diferencias entre las propiedades de un robot simulado y uno real, se utiliza un robot ideal que intenta imitar la trayectoria y el desplazamiento que un agente autónomo *skid-steered* posee. Con esto, la respuesta que tiene el robot simulado ante comandos de velocidades posee una geometría de colisión que abarca más espacio que la de un robot real Husky A200^{TM} (Figura 4.3). De este modo, se consideran los posibles derrapes que pueden ocurrir empíricamente, y así, la política aprendida considera este escenario.

Figura 4.3: Robot Husky A200 simulado en Gazebo.

Se prueba el desempeño del agente en el mundo utilizando dos tipos de sensores. Primero, se equipa al agente con un único sensor LiDAR 2D que le entrega información sobre el entorno en un campo de visión de 360 grados. Posteriormente, se entrena al agente con dos sensores LiDAR que le proporcionan un campo de visión de 270 grados cada uno, para que luego mediante el procesamiento de estas mediciones de rango, el robot pueda generar un mapa local de 360° de campo de visión. En ambos casos, la información es tratada y entregada como observaciones al agente (ver Sección 4.2.2 para mayor detalle).

Con el propósito de lograr la convergencia del algoritmo, se propone en primer lugar entrenar al agente en un entorno simple, con obstáculos uniformes y simétricos, para luego pasar a un entorno más complejo, donde el agente se enfrenta a variadas formas de obstáculos distribuidos dentro del ambiente (Figura 4.4). Ambos entornos poseen 16 metros de ancho y de alto. El objetivo de seguir dicha metodología es encontrar los hiperparámetros correctos del algoritmo y función de recompensa experimentalmente, entrenando en el mundo simple para luego entrenar en el mundo complejo y evaluar correctamente el desempeño de *Dreamer*. Se procede de esta forma debido a que en el ambiente simple el agente requiere de una menor cantidad de episodios para que el robot aprenda una política, lo que se traduce en un menor tiempo de entrenamiento, así se evalúan diferentes parámetros de manera más eficaz. Ambos mundos se basan en lo expuesto en [43].



(a) Entorno Simple **(b)** Entorno complejo

Figura 4.4: Entornos de ejecución ambiente de navegación.

La Figura 4.5 esquematiza la configuración espacial del robot y de su región objetivo. En la representación, v_x indica la velocidad lineal del agente, v_θ la velocidad angular, ρ_{target} la distancia hacia el *target* y θ_{target} el ángulo generado entre la orientación del robot y la posición de su región objetivo.

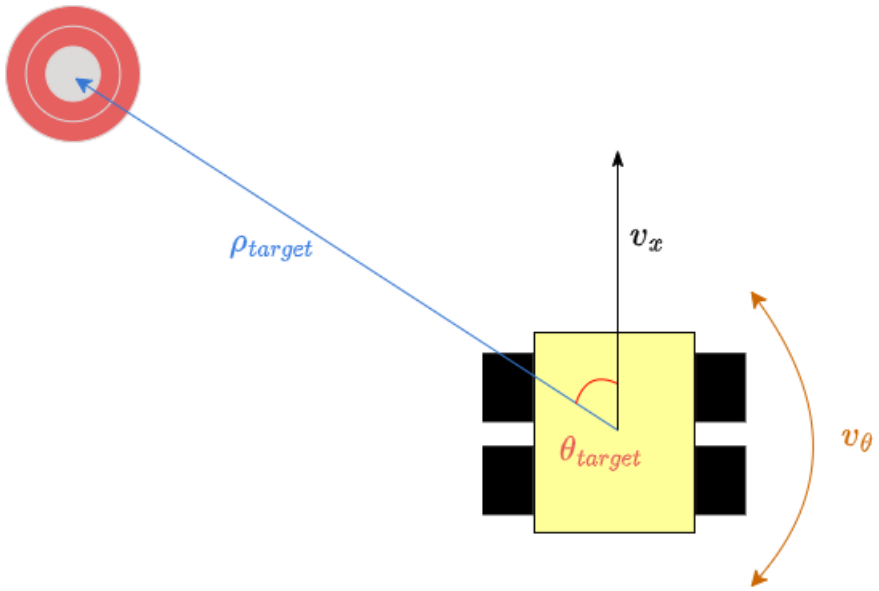


Figura 4.5: Configuración espacial robot simulado.

En la sección que sigue, se presenta la formulación del problema como un proceso de decisión de Markov parcialmente observable. En particular se describen las observaciones, acciones y función de recompensa que permiten el aprendizaje de la política.

4.2.2. Formulación del POMDP

La interacción entre el agente y el entorno de ejecución es modelada como un *Partially Observable Markov Decision Proccess* (POMDP). Como se expone en el Capítulo 2 de este trabajo, un POMDP se define por un conjunto de estados $\mathcal{S}$, un conjunto de acciones $\mathcal{A}$, una función de recompensa $r(s, a)$, una función de transición $\mathcal{T}(s, a, s') = p(s'|s, a)$, una función de observación $\mathcal{O}(o|s', a)$, un conjunto de observaciones Ω y un factor de descuento $\gamma \in [0, 1]$. Considerando esta formulación, tanto $\mathcal{S}$ como $\mathcal{T}(a, a, s')$ quedan determinados por la dinámica del entorno de ejecución. Por otro lado, las acciones, observaciones y función de recompensa deben ser diseñadas para abordar el problema de navegación. Esta formulación se inspira en lo propuesto en [43] y [44], con ciertas adaptaciones para este problema en particular.

Función de Recompensa

Esta se construye con el fin de promover la llegada a la región objetivo penalizado colisiones y movimientos arriesgados. La Ecuación 4.1 indica cómo se define $r(s, a)$, donde ρ_{thresh} corresponde a un umbral de distancia usada para definir si el agente ha llegado a su *target* o no y ρ^t_{target} indica la distancia euclideana entre el agente y su región objetivo en el instante de tiempo t:

$$r(s,a) = \begin{cases} r^t_{navigation} & \text{si } \rho^t_{target} \geq \rho_{thresh} \\ r^t_{success} & \text{si } \rho^t_{target} < \rho_{thresh} \\ r^t_{collision} & \text{si el agente colisiona} \end{cases} \tag{4.1}$$

Así, si el agente llega a su región objetivo (verificando que la distancia euclidiana entre su posición y la del *target* sea menor al umbral establecido), una recompensa positiva es asignada ($r^t_{success}$). No obstante, si el agente colisiona con un obstáculo del entorno, una recompensa negativa es asignada ($r^t_{collision}$).

El término $r^t_{navigation}$ busca incentivar al agente a moverse hacia su *target*, penalizando comportamientos peligrosos. Se define como $r^t_{navigation} = \alpha_{rew} \cdot (r^t_{speed} + r^t_{target} + r^t_{fov} + r^t_{danger})$.

El término r^t_{fov} busca lograr que el agente aprenda a mantener la región objetivo dentro de un campo de visión de $2\theta_{fov}$ radianes proyectados desde el punto de referencia local del agente. Este se define como:

$$r^t_{fov} = (F \cdot \cos(\theta^t_{target}) - F') \cdot \mathbb{1}_{\{\theta^t_{target}:|\theta^t_{target}|>\theta_{fov}\}}(\theta^t_{target}), \tag{4.2}$$

donde F y F' son constantes.

Los términos r^t_{speed} y r^t_{target} buscan incentivar al agente a moverse hacia su región objetivo. En particular, r^t_{speed} penaliza altas velocidades y r^t_{target} premia al agente cuando se acerca al *target*. Estos quedan definidos de la siguiente forma:

$$r^t_{speed} = S \cdot \tilde{v}^t_{odom-x} \cos(\theta^t_{target}) \tag{4.3}$$

$$r^t_{target} = T \cdot \mathbb{1}_{\{\rho^t_{target}:\rho^t_{target}<\rho^{t-1}_{target}\}}(\rho^t_{target}) - T' \tag{4.4}$$

En las expresiones anteriores tanto S, como T y T' son constantes.

La cuarta componente de la recompensa de navegación, r^t_{danger} representa una penalización hacia al robot por aproximarse a un obstáculo. Dicha penalización es entregada al agente si alguna de las mediciones de rango, proveniente de sus sensores, es menor a la distancia de seguridad previamente definida, $\rho_{danger-thresh}$. Esta recompensa queda definida como:

$$r^t_{danger} = D \cdot \mathbb{1}_{\{\rho^t_{sensor}:\rho^t_{sensor}<\rho_{danger-thresh}\}}(\rho^t_{sensor}), \tag{4.5}$$

donde ρ^t_{sensor} corresponde a la medición de menor distancia radial del sensor, es decir al punto más cercano de los obstáculos que rodean al agente. El valor de $\rho_{danger-thresh}$ se determina considerando las dimensiones y *footprint* (huella) del robot a utilizar.

El factor que escala la recompensa de navegación (α_{rew}), busca acotar el rango de valores entre los cuales se mueve $r_{navigation}$. Esto se debe a que las recompensas de los entornos de ejecución de *Deep Mind Control Suite* se encuentran acotadas en el intervalo [0, 1] y el algoritmo *Dreamer* fue configurado y diseñado originalmente para lograr la convergencia en dichos ambientes. Más aún, la configuración paramétrica con la que se valida el correcto funcionamiento del algoritmo, se realiza en dichos entornos. Mediante el escalador de la recompensa, se busca ajustar los valores que esta toma para lograr la convergencia de *Dreamer* en entornos de navegación. Esta componente no forma parte de lo propuesto en [43].

Con el fin de promover el desplazamiento hacia el *target*, todos los términos que definen $r^t_{navigation}$ toman el valor de 0 como máximo valor. Cuando el agente toma decisiones arriesgadas o indeseables, fuertes penalizaciones son entregadas. Las constantes establecidas en las definiciones de las componentes de $r^t_{navigation}$ están ajustadas para que las penalizaciones tengan valores mínimos comparables: -6 para r^t_{target}, -8 para r^t_{fov} y -1 para r^t_{speed}. La Tabla 4.4 muestra los valores de las constantes que se utilizaron para definir dichas componentes de la recompensa de navegación.

Tabla 4.4: Valores de constantes que definen parcialmente la recompensa de navegación.

Constante	Valor
T	5
T'	6
F	3
F'	5
S	1

La última componente de $r^t_{navigation}$, que es la recompensa de peligro r^t_{danger}, al igual que α_{rew}, $r^t_{success}$ y $r^t_{collision}$ se determinan experimentalmente, pues los valores que pueden tomar tienen un gran impacto en la convergencia del algoritmo y por ende en el desempeño del agente. Estas componentes serán analizadas en mayor detalle en el próximo capítulo, donde se plasman los resultados obtenidos ante variaciones de estos parámetros.

Observaciones

Se propone formular las observaciones del agente según el siguiente vector:

$$o_t = (o_{sensor}, o_{odom}, o_{target}) \tag{4.6}$$

Donde la primera componente, o_{sensor}, corresponde al procesamiento de la información proveniente del sensor del robot. Cuando el agente posee solo un LiDAR con 360 grados de campo de visión, la señal emitida es procesada para entregar un vector que representa las distancias a los obstáculos más cercanos que rodean al agente. Dicho vector se obtiene acotando los valores que provienen del sensor $o^t_{range} = (\rho^t_1, ..., \rho^t_n)$ a un rango de $[\rho_{min}, \rho_{max}]$ (Ecuación 4.7), reduciendo el número de mediciones y normalizando sus valores. La primera transformación que se hace es la siguiente, siendo ρ^t_i para todo $i \in \{1, ..., n\}$ una medición del sensor perteneciente a o_{range}:

$$\rho^t_i = \begin{cases} \rho^t_i & \text{si } \rho_{min} < \rho^t_i < \rho_{max} \\ \rho_{min} & \text{si } \rho^t_i \leq \rho_{min} \\ \rho_{max} & \text{si } \rho^t_i \geq \rho_{max} \end{cases} \tag{4.7}$$

Con el objetivo de reducir la cantidad de valores del vector, se selecciona la información más relevante para el problema en particular. Esto se efectúa tomando segmentos del vector ya acotado al rango mencionado anteriormente y seleccionando un solo valor representante (algo similar a lo que se conoce como *pooling* en redes neuronales convolucionales [45]). Para el caso de estudio, dicho valor corresponde al mínimo de la sección elegida. De esta forma, el agente tiene noción de la distancia más próxima hacia los obstáculos del mundo. Una vez seleccionados los m valores que representa al vector de observación, este es normalizado dividiendo todos sus valores por ρ_{max}, resultado en el siguiente vector, donde $\bar{\rho}^t_i$ para todo $i \in \{1, ..., m\}$, representa el valor mínimo de cada segmento seleccionado:

$$o^t_{sensor} = \left(\frac{\bar{\rho}^t_1}{\rho_{max}}, ..., \frac{\bar{\rho}^t_m}{\rho_{max}} \right) \tag{4.8}$$

Como la dimensión del mundo es de 16x16 metros, se elige -18 y 18 como valores de ρ_{min} y ρ_{max} respectivamente. Por otro lado, la totalidad de mediciones del sensor que se seleccionan son $m = 30$.

Las señales emitidas por los sensores LiDARs, entregan distancias $\{\rho_i\}^n_{i=1} \in (\rho_{min}, \rho_{max}) \cup \{\underline{\rho}, \bar{\rho}\}$, que poseen un ángulos asociados $\{\theta_i\}^n_{i=1}$ por cada punto i del entorno. ρ_{min} y ρ_{max} definen el rango de valores de los puntos y $\underline{\rho}$ y $\bar{\rho}$ son codificaciones para los valores que se encuentran bajo ρ_{min} y sobre ρ_{max} respectivamente, como en [43]. Con esto, se obtiene un vector de coordenadas polares del ambiente:

$$o^t_{points} = \left\{ \left(\rho^t_i \cos(\theta^t_i), \rho^t_i \sin(\theta^t_i) \right) \right\}^{k \leq n}_{i=1}, \tag{4.9}$$

donde k corresponde a la cantidad de mediciones que se encuentran dentro del rango del LiDAR para el instante de tiempo t.

Los puntos de la Ecuación 4.9, son luego normalizados y procesados con una red neuronal que utiliza extractores de características inspirados en PointNet [22] para obtener una nube de puntos 2D que represente el entorno del agente.

Como se indica en [43], la componente $o_{odom} = (\hat{v_x}, \hat{v_\theta})$ se define como el vector que contiene las estimaciones normalizadas de velocidad lineal y angular del robot basadas en odometría:

$$o^t_{odom} = \left(\frac{v^t_x}{v^{max}_x}, \frac{v^t_\theta}{v^{max}_\theta} \right), \tag{4.10}$$

donde v^t_x y v^t_θ corresponden a las estimaciones de velocidad lineal y angular no normalizadas respectivamente, y v^{max}_x y v^{max}_θ a sus valores máximos.

Finalmente, la componente $o_{target} = (\rho_{target}, \theta_{target})$ queda definida como un vector que contiene las coordenadas polares normalizadas de la región objetivo (*target*):

$$o^t_{target} = \left(\frac{\text{mín}\{\rho^t_{target}, \rho^{max}_{target}\}}{\rho^{max}_{target}}, \frac{\theta^t_{target}}{\pi} \right) = (\hat{\rho}^t_{target}, \hat{\theta}^t_{target}), \tag{4.11}$$

donde ρ^{max}_{target} es la distancia máxima a la que se puede encontrar la región objetivo de la posición inicial del agente.

Las componentes del vector de observaciones se agrupan para procesarse por separado pero en un mismo codificador. Con esto, la red neuronal puede obtener la información relevante de cada una de las componentes del vector de observación por separado.

Acciones

Dado que el robot simulado es ideal diferencial (imita un comportamiento *skid-steered*) y responde a comandos de velocidad continuos, las acciones quedan definidas por un vector de dos dimensiones: $a_t = (v^t_x, v^t_\theta)$, donde $v^t_x \in [0, v^{max}_x]$ corresponde a la velocidad lineal del agente y $v^t_\theta \in [v^{min}_\theta, v^{max}_\theta]$ corresponde a su velocidad angular.

4.2.3. Configuración experimental

En cuanto a la configuración episódica del ambiente en entrenamiento, para el entorno simple se definen como estados terminales: llegar a la región objetivo y superar un límite de interacciones agente-ambiente, en este caso, 400 *steps*. Esta definición del entorno no es muy clásica, pues en los simuladores que abordan el problema de navegación, generalmente se considera la colisión como un estado terminal. El eliminar la colisión como estado terminal se toma como medida exploratoria, debido a que el agente requiere una gran cantidad de colisiones para aprender un modelo del mundo robusto a partir del cual logre obtener una política.

Para el caso del mundo complejo, la configuración episódica de entrenamiento queda definida según lo establecido clásicamente por la literatura para entornos de navegación. Es decir, existen tres escenarios posibles que pueden finalizar un episodio: alcanzar objetivo de navegación, colisionar con un obstáculo o superar los *steps* de tiempo límite.

La implementación del algoritmo *Dreamer* para el entorno de navegación en ROS se realiza adaptando el desarrollo realizado para los entornos de MuJoCo (ver Sección 4.1.3). Tanto el decodificador (*Observation Model*), como el modelo de recompensa, el *Recurrent State Space Model*, el actor y el crítico poseen la misma estructura que la utilizada para los entornos *cheetah* y *cartpole*, se le cambian las dimensiones de entrada y salida para adaptarlo al problema de navegación. Por otro lado, el codificador de observaciones se adapta para que procesen las observaciones que posee el robot en el entorno de navegación.

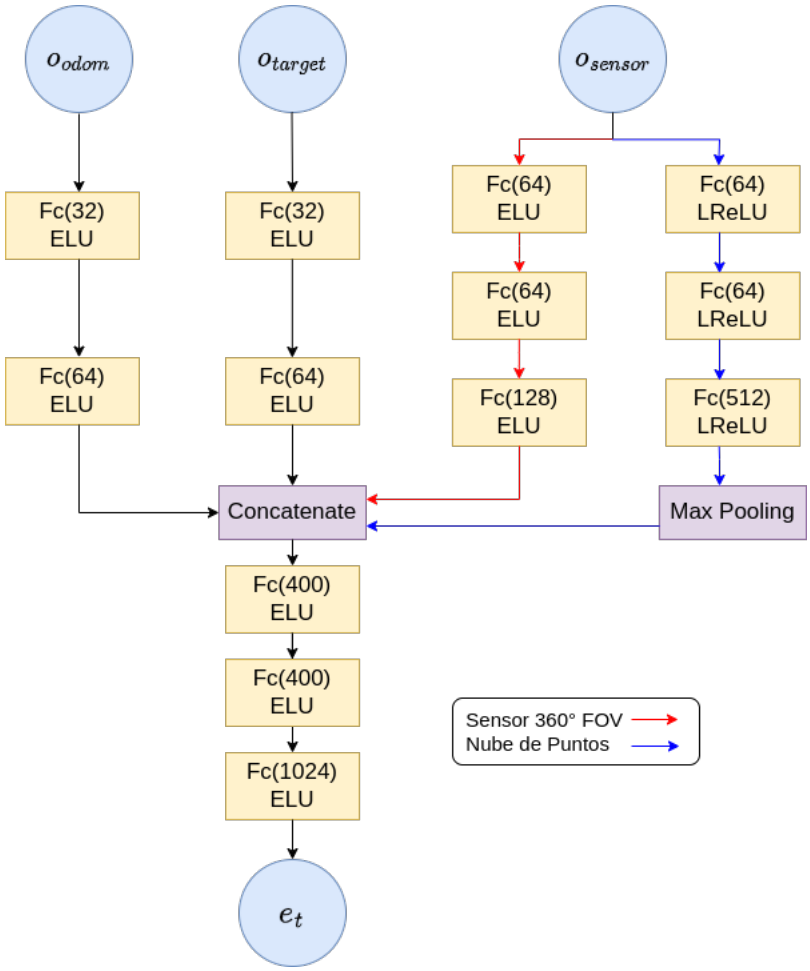


Figura 4.6: Parametrización multimodal del codificador para el entorno de navegación. En el caso en que las observaciones generan una nube de puntos para ser procesadas (flecha azul), se implementa un bloque de *Max Pooling*, pues este forma parte de los extractores de características de la red PointNet.

En particular, debido a que las observaciones del agente de navegación provienen de fuentes de información diferentes y son representadas a través de estructuras distintas, se procesan bajo una perspectiva multimodal similar a lo presentado en [46] para el caso en que las observaciones son vectoriales. Como se expuso en la Sección 4.2.2, estas se definen como $o_t = (o_{sensor}, o_{odom}, o_{target})$,

donde o_{sensor} corresponde a las observaciones que provienen del sensor, o_{odom} a las estimaciones de velocidad instantánea basadas en odometría y o_{target} a las coordenadas polares de la región objetivo. Así, con el objetivo de obtener información relevante, se procesa cada componente de o_t independientemente mediante extractores de características y sus resultados son concatenados (ver Figura 4.6).

En la Figura 4.6, la estructura de las redes neuronales sigue la notación "Tipo (parámetros) Función de activación", o simplemente "Tipo", en caso de capas no paramétricas. "Fc" significa *fully connected* y su parámetro representa la cantidad de neuronas asociadas a su capa. El vector e_t representa el *embedding* que se obtiene a través de la codificación de las observaciones.

Los extractores de características que se utilizan para procesar tanto o_{odom} como o_{target} son redes neuronales *fully connected*. El procesamiento de o_{sensor} depende del sensor a utilizar. En el caso en que el agente se encuentra equipado por un láser LiDAR con 360 grados de campo de visión, el extractor de características, al igual que en el caso anterior, es una red *fully connected*, pues se requiere procesar únicamente un vector. Por otro lado, en el caso en que el robot utilice dos sensores LiDAR que entregan una nube de puntos, se hace uso de parte de los extractores de características de la red PointNet [22] para procesar las señales emitidas del sensor [46].

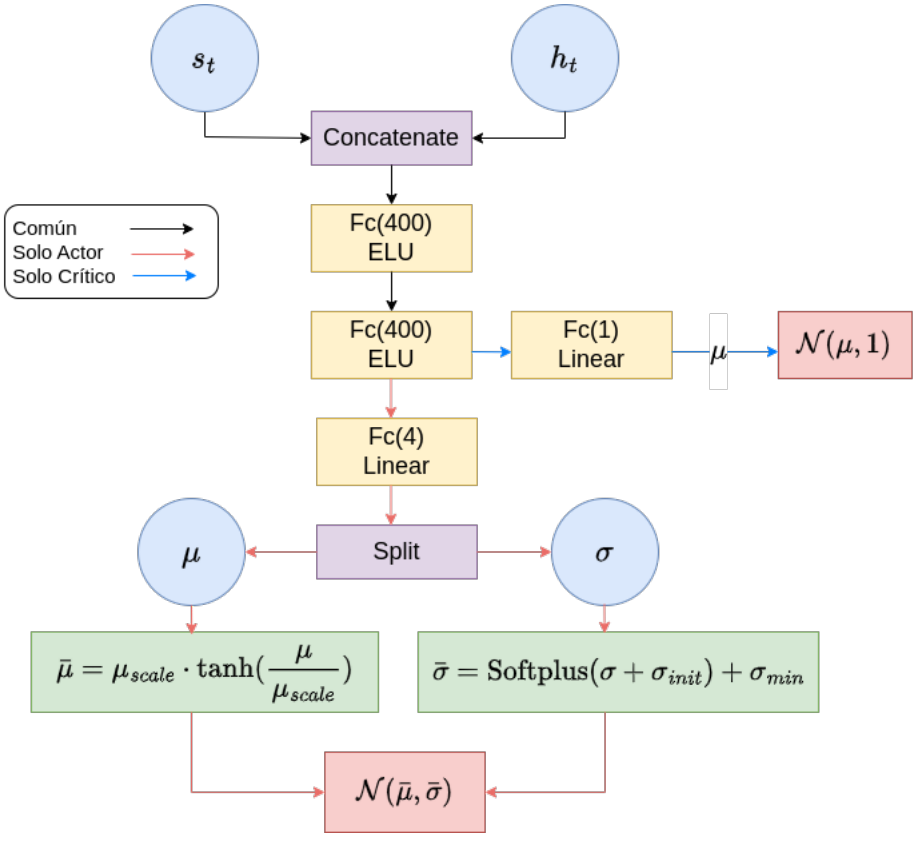


Figura 4.7: Aprendizaje de política a partir de estados latentes.

Al igual que en el caso de los entornos de ejecución de MuJoCo, se utiliza el flujo de información de la Figura 4.2 para aprender la dinámica del mundo. La Figura 4.7 esquematiza cómo el agente utiliza los estados latentes para aprender una política, donde s_t corresponde a la componente

estocástica de los estados y h_t a la determinista. Todas las capas ocultas de las redes neuronales poseen 400 neuronas y usan ELU [40] como función de activación.

Para llevar a cabo los entrenamientos, se hace un llenado inicial del *replay buffer* [18] con $50 \cdot 10^3$ *steps* del entorno realizados mediante acciones aleatorias. A continuación, se itera en un bucle de $500 \cdot 10^3$ *steps* (hasta observar convergencia), en los cuales, luego de cada interacción con el entorno, se realiza una cierta cantidad (dependiendo del entorno) de actualizaciones a los parámetros de las redes utilizando optimizadores *Adam* [41]. Para la selección de las acciones durante el entrenamiento se aplica un ruido de Ornstein-Uhlenbeck (OU), modulado por un factor que decae de 1.0 a 0.05 en $80 \cdot 10^3$ *steps* en el mundo simple y en $160 \cdot 10^3$ *steps* en el mundo complejo. Finalmente, los hiperparámetros del algoritmo *Dreamer* se resumen en la Tabla 4.5.

Tabla 4.5: Hiperparámetros utilizados para entrenamiento del algoritmo *Dreamer* en entorno de navegación.

Parámetro	**Valor**
Tamaño máximo de *Experience Replay*	$1 \cdot 10^6$
Tasa de aprendizaje actor	$1 \cdot 10^{-4}$
Tasa de aprendizaje crítico	$1 \cdot 10^{-4}$
Tasa de aprendizaje modelo del mundo	$1 \cdot 10^{-4}$
Factor de descuento (γ)	0.95
Factor λ	0.95
Tamaño de *batch*	32
Dimensión *embedding*	1024
Largo de secuencias	32
Horizonte de planificación	15

Capítulo 5

Entrenamientos, resultados y discusión

5.1. Entrenamientos en entornos MuJoCo

A continuación se exponen los resultados obtenidos tras realizar los entrenamientos en los entornos de *Deep Mind Control Suite*, *cheetah* y *cartpole*, con las tareas *run* y *swingup*, respectivamente.

Como se indicó en el capítulo precedente, el entrenamiento de los agentes se realiza a lo largo de 1000 episodios. Cada uno de estos consiste en 1000 *steps* en los cuales el agente toma acciones buscando maximizar su recompensa. Luego de cada episodio de entrenamiento, se realizan 100 actualizaciones de los parámetros (*updates* en inglés) de todas las redes neuronales que componen el modelo del mundo y la política. Una vez finaliza cada episodio de entrenamiento, se ejecutan 3 episodios de evaluación en los cuales el agente toma acciones de manera determinista, es decir, sin interferencia de un ruido de exploración. Los parámetros de entrenamiento que se utilizaron para ambos entornos de ejecución se encuentran en la Tabla 4.3.

En particular, para ambos ambientes se expone la recompensa episódica promedio de evaluación a lo largo de los *steps* de entrenamiento y la evolución de las funciones de pérdida del actor y del crítico a través de las actualizaciones de parámetros. Las funciones de pérdida de las redes neuronales que componen el modelo del mundo se encuentran en el anexo A 2 y A 3.

5.1.1. Resultados Entorno *Cheetah*

La Figura 5.1 muestra la evolución de la recompensa episódica a lo largo del entrenamiento. En particular, el trazado azul representa el promedio de la recompensa episódica obtenida en 3 trayectorias de evaluación ejecutadas luego de cada episodio de entrenamiento. La sombra celeste que sigue el trazado corresponde a la desviación estándar de dicho retorno medio a lo largo de los tres episodios de evaluación.

Para obtener una mejor visualización de los resultados, se eligen de manera representativa 10 valores de la recompensa episódica promedio de los 3 episodios de evaluación a lo largo del entrenamiento. Por lo que se observan los valores que corresponden al retorno promedio cada $100 \cdot 10^3$ *steps* de entrenamiento (100 episodios).

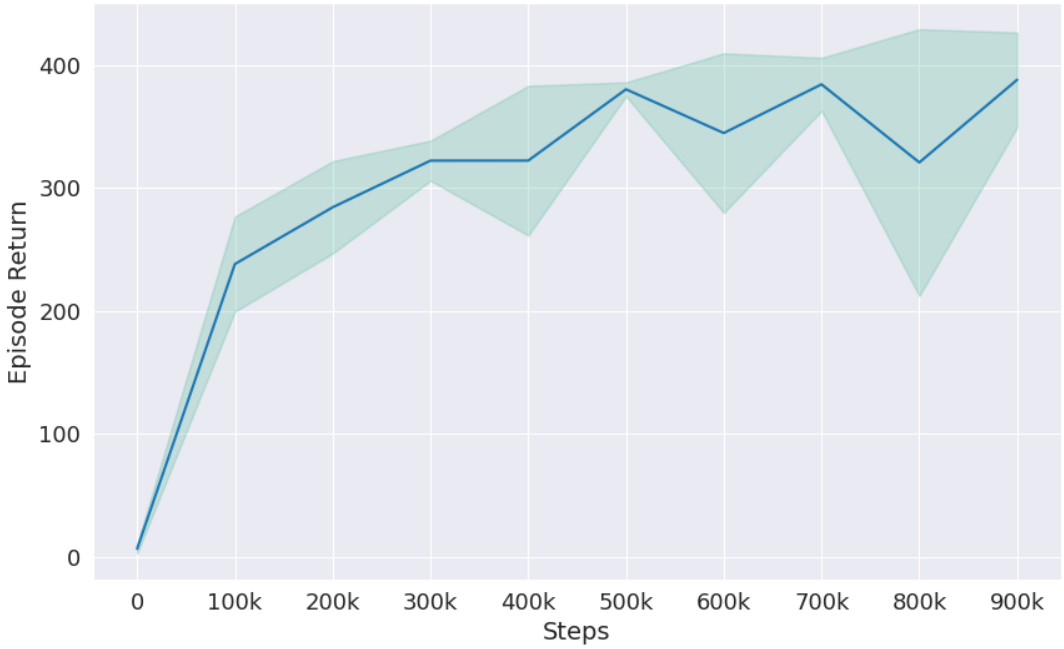
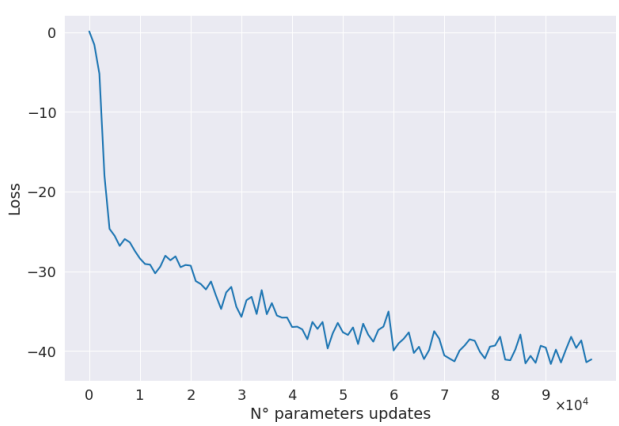
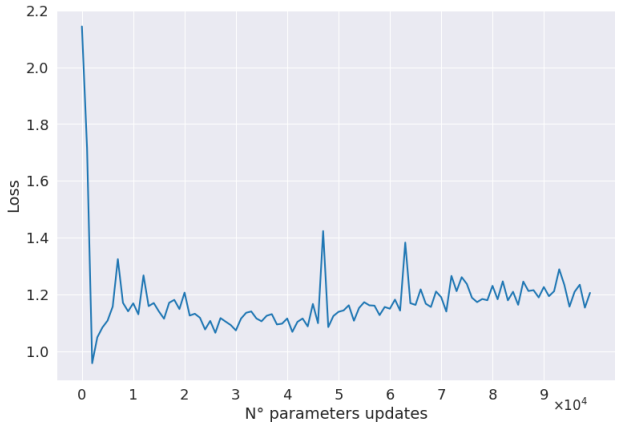


Figura 5.1: Recompensa episódica promedio a lo largo de los 10^6 *steps* de entrenamiento.

En el gráfico anterior se observa cómo el agente logra aprender una política que maximiza su recompensa a lo largo de los *steps* de entrenamiento. En efecto, se observa un gran aumento de la recompensa episódica en los primeros 300 episodios ($300 \cdot 10^3$ interacciones con el ambiente), donde esta pasa de 0 a 320 aproximadamente y posteriormente se estabiliza presentando pequeñas oscilaciones en torno a los 350. Se observa también, cómo a partir del momento en que la recompensa episódica alcanza valores más elevados (aproximadamente desde los $500 \cdot 10^3$ *steps* en adelante), el retorno episódico presenta una mayor varianza que podría deberse a las diferencias que presentan los episodios de evaluación.



(a) Función de pérdida actor



(b) Función de pérdida crítico

Figura 5.2: *Loss* de entrenamiento a lo largo de actualizaciones de parámetros.

La Figura 5.2 expone las funciones de pérdida (*loss* en inglés) del actor (5.2a) y del crítico (5.2b). Se aprecia que la *loss* del actor decae de 0 a -50 aproximadamente a lo largo de las actualizaciones de parámetros, mientras que la del crítico parece oscilar constantemente entre 1.0 y 1.4. Dichas fluctuaciones, son comunes cuando se toma un enfoque de actor-crítico para el aprendizaje de la política, como es el caso del algoritmo *Dreamer*.

El comportamiento que se observa en dichas funciones de pérdida revela patrones intrigantes en el entrenamiento del agente, sin embargo, para comprender completamente su impacto y significado, se debe analizar la naturaleza del algoritmo y del entorno utilizado, esto será discutido en la Sección 5.1.3.

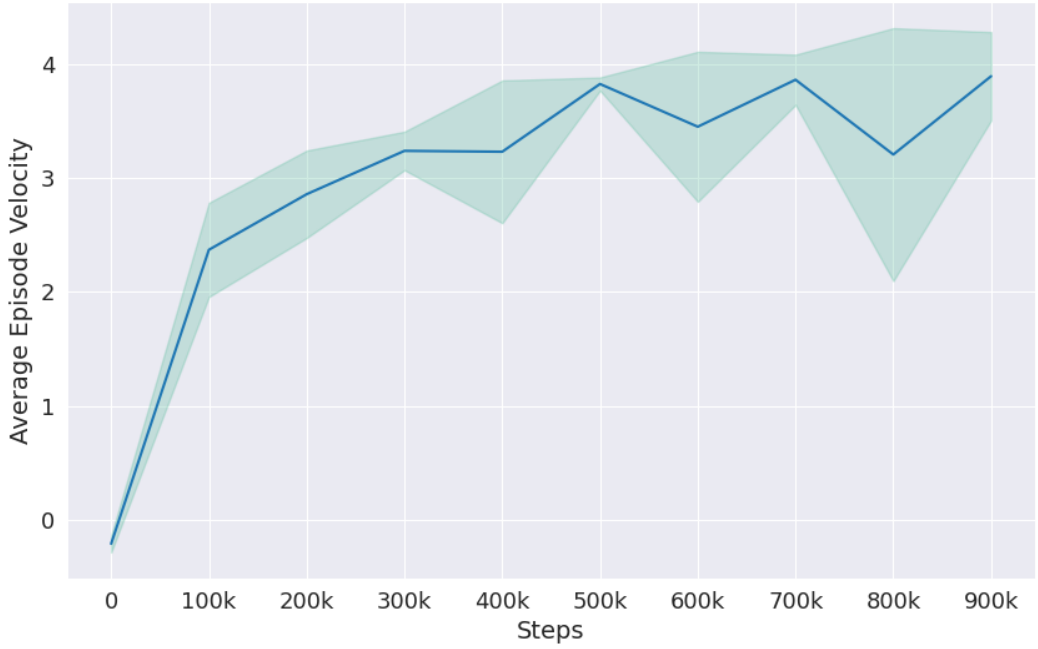


Figura 5.3: Velocidad episódica promedio del agente *cheetah* a lo largo de los 10^6 *steps* de entrenamiento.

El gráfico de la Figura 5.3 muestra la evolución de la velocidad episódica promedio del *cheetah* a lo largo del entrenamiento. Se observa cómo el agente en los primeros $100 \cdot 10^3$ steps de entrenamiento logra aumentar considerablemente su velocidad, pues esta pasa de ser negativa (lo cual significa que el agente retrocede en lugar de avanzar), a tomar valores cercanos a los 2.5 m/s. Dicho comportamiento, revela que el agente logra aprender una política que le entrega un mayor retorno, pues la función de recompensa que define el ambiente, depende de la velocidad del *cheetah*.

Al comparar los resultados obtenidos con los presentados en [17], se observa que el algoritmo *Dreamer* es más eficiente que los algoritmos D4PG [47] y A3C [15] en cuanto a interacciones con el ambiente, pues para alcanzar valores similares de recompensa episódica, tarda considerablemente menos *steps* de entrenamiento. Más aún, en su punto de convergencia, *Dreamer* supera a A3C en términos de recompensa episódica promedio, alcanzando valores de 400, en comparación con los 200 de A3C.

Más aún, al comparar los resultados obtenidos por *Dreamer* para el entorno *cheetah*, con los presentados en [1], donde se utiliza el algoritmo *model based* PlaNet, se observan comportamientos similares. En ambos casos la recompensa episódica promedio alcanza valores cercanos a 400 al rededor de los $500 \cdot 10^3$ *steps* de entrenamiento, lo cual revela la convergencia y correcta implementación del algoritmo *Dreamer*.

5.1.2. Resultados entorno *Cartpole*

El gráfico que se presenta a continuación (Figura 5.4) exhibe la variación de la recompensa episódica a lo largo del proceso de entrenamiento del *cartpole*. Al igual que en el caso anterior, la línea azul representa el promedio de la recompensa episódica obtenida en tres trayectorias de evaluación realizadas después de cada episodio de entrenamiento. La sombra celeste que acompaña la línea refleja la desviación estándar de dicho promedio de retorno a lo largo de los tres episodios de evaluación.

Tras observar el comportamiento de la recompensa episódica en la Figura 5.4, se aprecia cómo esta comienza a crecer a medida que el agente interactúa con el entorno. Más aún, se recalca que el aprendizaje se concentra en los primeros $300 \cdot 10^3$ *steps*, donde la recompensa episódica promedio alcanza valores que superan los 800 y posteriormente permanece estable en torno a dicho valor, presentando ciertas variaciones menores.

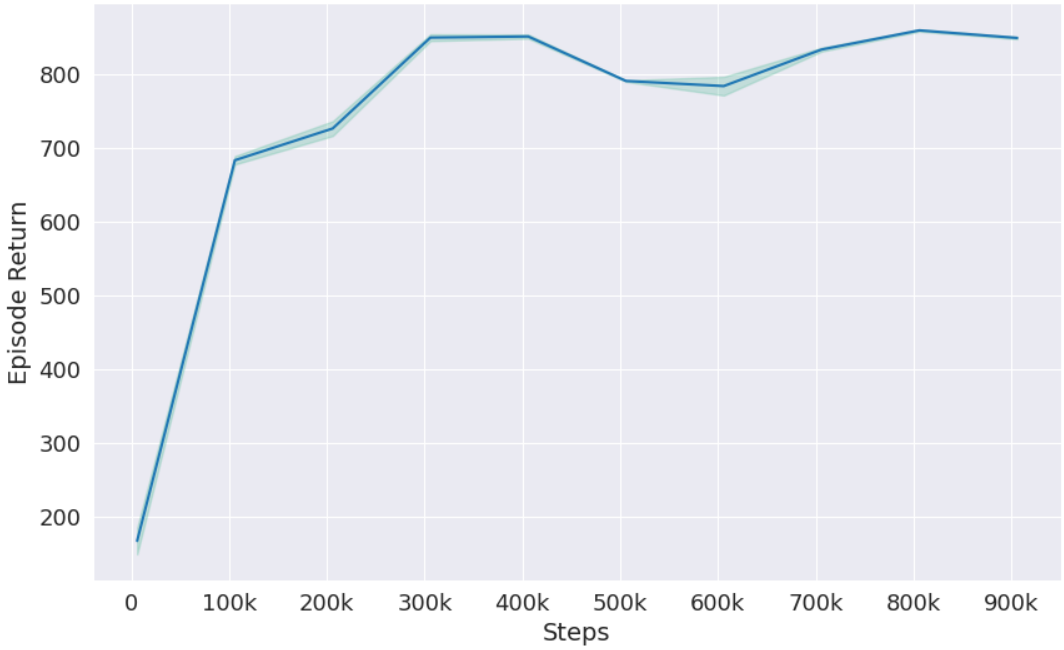
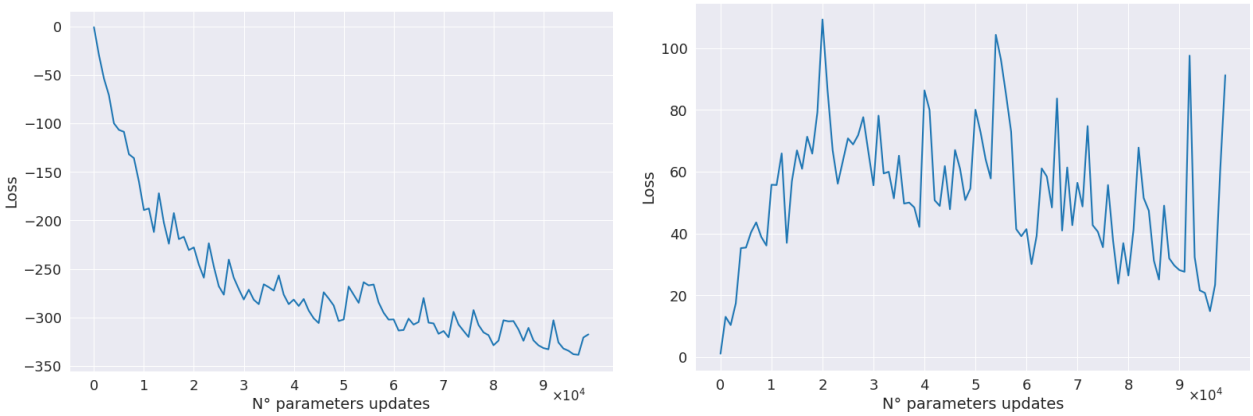


Figura 5.4: Recompensa episódica promedio a lo largo de los 10^6 *steps* de entrenamiento.

La Figura 5.5 muestra las funciones de pérdida del actor (Figura 5.5a) y del crítico (Figura 5.2b) a lo largo del entrenamiento. Se evidencia un fuerte decrecimiento de la *loss* en las primeras $3 \cdot 10^4$ actualizaciones de parámetros, donde esta cae de 0 a -300 aproximadamente. Este fenómeno se

relaciona con la tendencia que sigue la Figura 5.4 mencionada anteriormente, donde se ve el mayor crecimiento de la recompensa episódica en los primeros $300 \cdot 10^3$ *steps*.

El comportamiento que sigue la función de pérdida del crítico en este entorno (Figura 5.5b), no se caracteriza por decrecer a lo largo de las actualizaciones de parámetros de la red neuronal que lo representa. Más aún, a grandes rasgos se observa un crecimiento en los primeros $4 \cdot 10^4$ *updates* y un posterior decrecimiento, en los cuales los valores que toma la *loss* es muy oscilante.



(a) Función de pérdida actor **(b)** Función de pérdida crítico

Figura 5.5: *Loss* de entrenamiento a lo largo de actualizaciones de parámetros.

La Figura 5.6 muestra la cantidad de *steps* promedio en que el *pole* se encuentra en posición vertical por episodio de evaluación a lo largo de los 10^6 *steps* de entrenamiento. El criterio utilizado para determinar si el *pole* se encuentra en posición vertical, se fundamenta en las observaciones recibidas por el agente en cada *step* de evaluación. Si el ángulo que forma el *pole* con respecto al eje vertical es menor a un umbral establecido, entonces se dice que este se encuentra posicionado correctamente hacia arriba del *cart*. Para el caso se define como umbral un ángulo de 22.5°.

Se observa un comportamiento muy similar al que presenta la recompensa episódica promedio (figura 5.4), donde el mayor aprendizaje del agente se concentra en los primeros $300 \cdot 10^3$ *steps* de entrenamiento. En este contexto, el agente pasa de lograr mantener el *pole* en posición vertical durante 150 *steps* a 850 *steps* promedio en un episodio de evaluación.

Los resultados que revelan las Figuras 5.4 y 5.6 mejoran lo obtenido por los algoritmo SLAC [48], D4PG [47] y PlaNet [1] en cuanto a la recompensa episódica promedio alcanzada y la cantidad de *steps* de entrenamiento necesarios para que el agente la obtenga. Más aún, se observa que la implementación del algoritmo *Dreamer* realizada en el presente trabajo alcanza resultados similares a los expuestos en su publicación original [2].

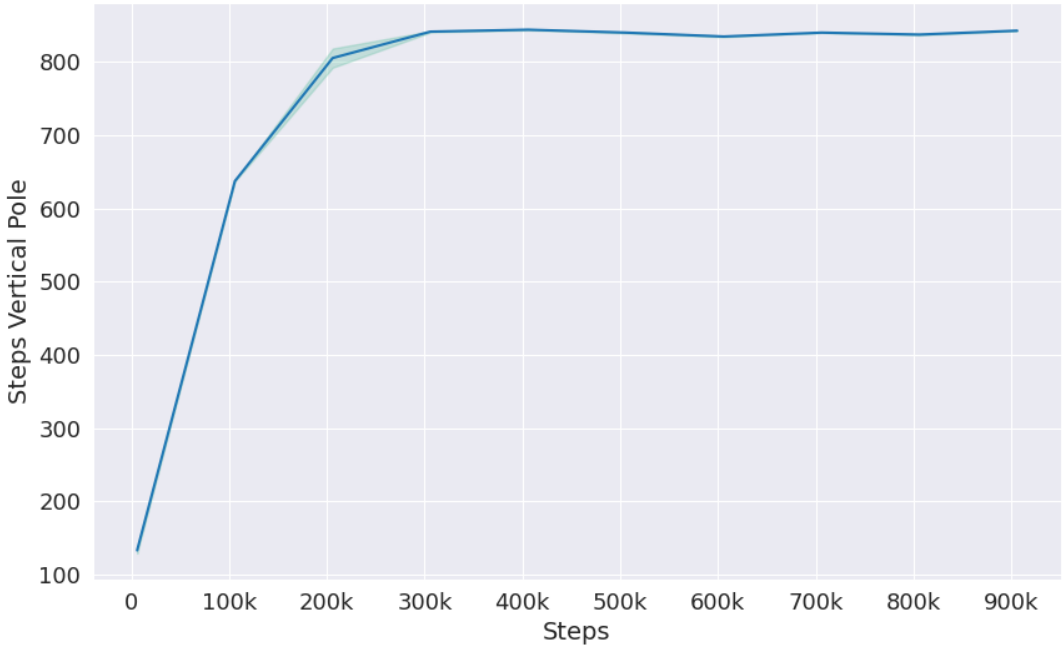


Figura 5.6: Cantidad de *steps* en los que el *pole* se encuentra vertical por episodio promedio a lo largo de los 10^6 *steps* de entrenamiento.

5.1.3. Discusión

Con el propósito de analizar la convergencia del algoritmo *Dreamer*, en esta sección se discuten los resultados presentados anteriormente. En particular, se revela un aumento de la recompensa episódica promedio a lo largo de las interacciones del agente con el entorno durante el entrenamiento tanto para *cheetah* como para *cartpole*. Además, se recalca que para ambos ambientes la función de pérdida del actor decrece mientras que la del crítico no sigue una tendencia en particular a medida que aumentan las actualizaciones de los parámetros que representan dichas redes.

En los dos ambientes donde se evaluó el desempeño del algoritmo, se aprecia que el mayor crecimiento de la recompensa episódica promedio ocurre en el inicio del entrenamiento al igual que el mayor decrecimiento de la función de pérdida del actor. Específicamente se observa este fenómeno en los primeros 500 episodios de entrenamiento para el entorno *cheetah* y 300 para el entorno *cartpole*. Esto revela que los agentes lograron encontrar una política que maximiza el valor esperado de la recompensa a partir de los estados latentes generados por el aprendizaje del *Recurrent State Space Model* que propone *Dreamer*, y por ende, la convergencia del algoritmo.

A pesar de que en ambos ambientes se logró aumentar el retorno episódico a lo largo de los *steps* de entrenamiento, se aprecia que *cartpole* alcanza valores más elevados que *cheetah*; mientras que el primero supera una recompensa promedio de 800, el segundo alcanza los 400. Esto se debe a la naturaleza de las funciones de recompensa de cada entorno de ejecución y al valor que obtiene cada agente según el estado en que se encuentra. Además, en el entorno de ejecución *cheetah* se busca controlar seis valores que componen el vector de acciones simultáneamente mientras que en *cartpole*, solamente uno, lo que facilita la tarea a controlar y por ende, el aprendizaje de la política.

Por otro lado, al comparar las Figuras 5.4 y 5.1, se observa que el retorno de los episodios de evaluación para *cartpole* presentan mucha menos variabilidad que para el agente *cheetah*. Esto se debe, en parte, a que en el ambiente *cheetah* hay más escenarios posibles que en el entorno *caratpole* haciendo de este último, un ambiente más predecible.

Como se mencionó en la sección precedente, el comportamiento de la función de pérdida del crítico no se caracteriza por sugerir un decrecimiento evidente en los entornos *cartpole* y *cheetah*. Esto no se traduce directamente en la no convergencia del algoritmo, pues, como se apuntó anteriormente, el agente logra aprender una política que maximiza su recompensa en ambos entornos. El fenómeno que las Figuras 5.5b y 5.2b sugieren, es frecuente en los algoritmos que utilizan métodos de actor-crítico [15] para el aprendizaje de la política, como es el caso de *Dreamer*.

Más aún, dada la naturaleza del algoritmo utilizado (aprendizaje sobre imaginación latente), la optimización del crítico puede presentar más variabilidad. Al ser este, una red neuronal que busca aproximar la función de valor utilizando un modelo de recompensas y estados latentes aprendidos por el agente, las actualizaciones y cambios de dichos modelos se traducen en cambios para el crítico, haciendo que este no logre minimizar perfectamente su función de pérdida. Además, el hecho de que el fin de un episodio quede determinado por una cantidad de interacciones con el ambiente específica (para el caso 1000), hace que la estimación de los retornos para cada estado sea más difícil.

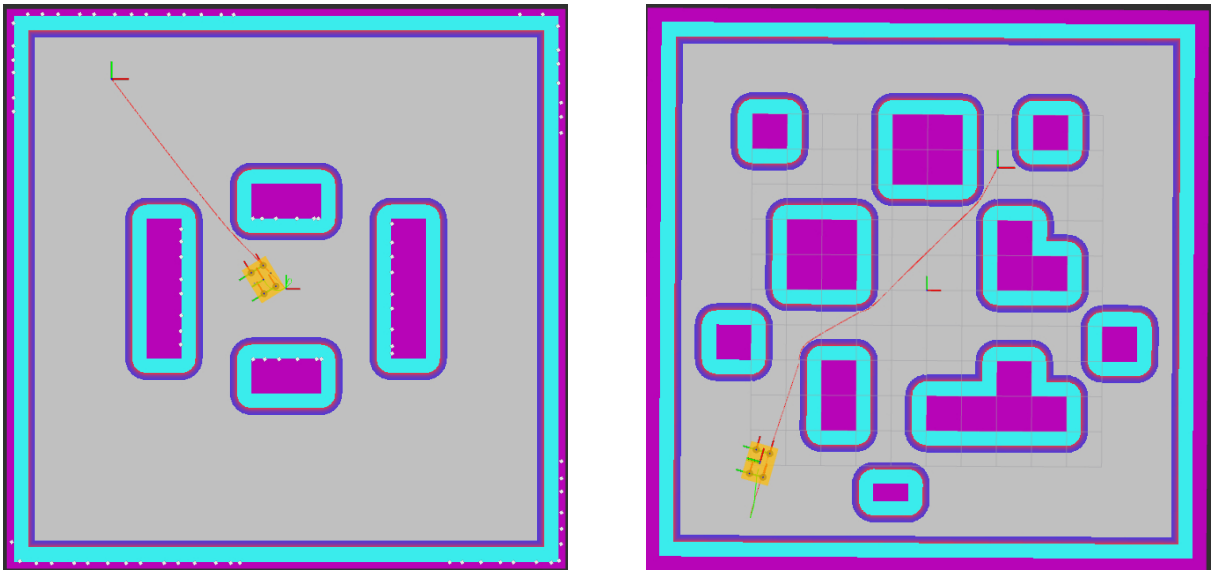
En síntesis, la presente discusión revela la correcta implementación y convergencia del algoritmo *Dreamer*. Estos hallazgos son un punto de referencia que permiten una comprensión más integral del método utilizado. Con esto, se establece una base que permite la adaptación del algoritmo a otros entornos, como por ejemplo el de navegación.

5.2. Entrenamiento en entorno de navegación

Con el objetivo de lograr que un agente autónomo aprenda una política de navegación utilizando el algoritmo *Dreamer*, se implementan dos mundos con obstáculos en el simulador Gazebo (Figura 4.4). Como se mencionó en el Capítulo 4 de este trabajo, el algoritmo es desarrollado en Python y se utiliza ROS como interfaz entre el algoritmo de aprendizaje y las simulaciones.

El procesamiento computacional requerido durante el desarrollo de los entrenamientos y validaciones de las políticas en simulaciones, es realizado en dos servidores equipados cada uno con una unidad de procesamiento gráfico: Nvidia GeForce GTX 1080 y Nvidia GeForce RTX 3060, respectivamente.

A fin de visualizar el comportamiento del agente durante el proceso de entrenamiento y validación, se utilizan mapas del entorno de navegación (Figura 5.7). Las regiones color violeta representan los obstáculos y muros del entorno, cuando el robot se encuentra cercano a dichas zonas, este recibe una recompensa de peligro (ver Ecuación 4.5). En ambos mapas del entorno de navegación, el pequeño rectángulo amarillo corresponde al robot, las pequeñas coordenadas (verde para el eje y y rojo para el eje x) corresponden a la posición de la región objetivo y el trazado rojo que las une con el agente corresponde a la trayectoria óptima que debería seguir el robot. Finalmente, el trazado verde detrás del agente (que se alcanza a ver en mayor detalle en la Figura 5.7b), corresponde a la trayectoria que ha seguido el agente desde su pose inicial.



(a) Mundo simple **(b)** Mundo complejo

Figura 5.7: Mapas de entrenamiento en simulador de navegación.

Como se mencionó en el capítulo precedente, la configuración episódica de los entornos de entrenamiento cambia según el mundo a utilizar. En el entorno simple de la Figura 5.7a, con el objetivo de incentivar la exploración del agente se definen únicamente la llegada a la región objetivo y el ejecutar más de 400 *steps* (*time-out*) como estados terminales. En el mundo complejo de la Figura 5.7b, a esta configuración, se le agrega la colisión como estado terminal. Esto se debe a que experimentalmente, al realizar este cambio en las condiciones de término para el mundo complejo, se obtienen mejores resultados.

La región objetivo (*target*) a la que debe llegar el agente sin colisionar se localiza aleatoriamente en cualquier espacio libre del mundo (donde no hay obstáculos), con una distancia mínima a la posición inicial del robot de 3 metros. Se considera que el agente alcanza su objetivo de navegación cuando la distancia euclideana medida desde su centro a las coordenadas de la región objetivo es menor o igual a 30 centímetros y además que el ángulo entre la posición del *target* y la orientación del robot sea menor o igual a 20 grados.

Para evaluar el desempeño del algoritmo se utilizan las siguientes métricas:

- **Tasa de éxito promedio** (*success rate*, SR): corresponde a la razón entre el número de episodios en los que el agente logra alcanzar su objetivo de navegación sin colisionar con los obstáculos del ambiente y el total de episodios de evaluación.
- **Tasa de colisión** (*collision rate*, CR): indica la proporción de episodios en los cuales el agente colisionó del total de episodios de evaluación.
- **Tasa de tiempo límite** (*time-out rate*, TR): se refiere a la proporción de episodios finalizados por *time-out* de la totalidad de episodios de evaluación.
- **Recompensa episódica promedio** (*average episode return*, AER): corresponde al promedio del retorno que recibe el agente en cada episodio de evaluación.
- **Éxito ponderado por largo de la trayectoria** (*success weighted by path length*, SPL [49]):

entrega una métrica sobre la optimalidad de las trayectorias seguidas por el agente. Esta se define como:

$$\text{SPL} = \frac{1}{M_{eval}} \sum_{i=1}^{M_{eval}} \mathbb{1}_{S_i} \cdot \frac{l_i^{\text{shortest}}}{\max l_i^{\text{shortest}}, l_i^{\text{agent}}} \tag{5.1}$$

Donde M_{eval} corresponde a la cantidad de episodios de evaluación. Para cada episodio i, S_i indica si el el agente tuvo éxito o no, l_i^{shortest} es el largo la trayectoria más corta hacia el *target* y l_i^{agent} el largo del camino seguido por el agente.

Se consideran $500 \cdot 10^3$ *steps* de entrenamiento hasta observar convergencia del algoritmo, donde cada 2000 *steps* son guardados los pesos de los modelos con el fin de evaluar el proceso de aprendizaje.

5.2.1. Resultados del entrenamiento en el mundo simple

Al comienzo de cada episodio de entrenamiento se localiza el robot al centro del mundo con una orientación aleatoria con el objetivo de dificultar la tarea y hacer más robusto el aprendizaje. Dicha configuración se lleva a cabo de esta manera debido a que, de lo contrario, pueden haber muchos casos en los que la trayectoria que deba seguir el robot para llegar a su *target*, no se vea enfrentada a un obstáculo.

Según lo señalado en la Sección 4.2.2 del presente texto, los valores de r^t_{danger}, $r^t_{success}$, $r^t_{collision}$ y α_{rew} que terminan la definición de la función de recompensa son determinados experimentalmente. Dichos valores varían según el sensor con el que se equipa al agente, y por ende, según las observaciones que este recibe. Los valores encontrados están resumidos en la Tabla 5.1 tanto para observaciones con un único sensor LiDAR de 360° de campo de visión (FOV, por sus siglas en inglés) como para los dos sensores a partir de los cuales se construye la nube de puntos (PCL, por sus siglas en inglés).

Tabla 5.1: Parámetros de la función de recompensa para entrenamiento de algoritmo *Dreamer* en entorno de navegación simple.

Observación	r^t_{danger}	$r^t_{success}$	$r^t_{collision}$	α_{rew}
360° FOV	-17	15	-20	0.32
PCL	-17	15	-20	0.5

Con respecto a el entrenamiento de las redes neuronales, para ambas observaciones se realizan 2 actualizaciones de parámetros (*updates*) luego de cada *step* de entrenamiento utilizando optimizadores Adam [41], con las tasas de aprendizaje señaladas en la Tabla 4.5.

Dados los valores que definen la función de recompensa y la configuración de las actualizaciones de parámetros, la Tabla 5.2 resume los resultados de la evaluación a la mejor política obtenida mediante los entrenamientos realizados. Dicha política se encuentra tras evaluar los modelos del mundo y del actor cada 2000 *steps* de tres entrenamientos ejecutados bajo la misma configuración, promediando las métricas de desempeño y seleccionando la iteración para la cual se obtuvieron mejores métricas en promedio.

A pesar de haber realizado los entrenamientos con una configuración episódica donde la colisión no es un estado terminal, las evaluaciones se realizan sobre un ambiente que sigue las consideraciones clásicas de un entorno de navegación, es decir, donde tanto la colisión, como la llegada al *target* y el *time-out* son estados terminales. Las métricas expuestas se encuentran definidas en la subsección anterior. Los resultados de la tabla corresponden tanto a la configuración del agente con observaciones provenientes del sensor con 360° FOV, como del agente que utiliza una nube de puntos para codificar la información del entorno.

Tabla 5.2: Promedio y desviación estándar de las métricas de desempeño obtenidas tras ejecutar 100 episodios de evaluación en 3 entrenamientos independientes del agente de navegación en mundo simple.

Obs.	***Steps***	**CR**	**TR**	**SR**	**SPL**	**AER**
360° FOV	$78 \cdot 10^3$	0.4 ± 0.165	0.0 ± 0.0	0.6 ± 0.179	0.478 ± 0.148	-84.01 ± 49.17
PCL	$106 \cdot 10^3$	0.1 ± 0.037	0.0 ± 0.0	0.9 ± 0.037	0.649 ± 0.031	-72.87 ± 20.95

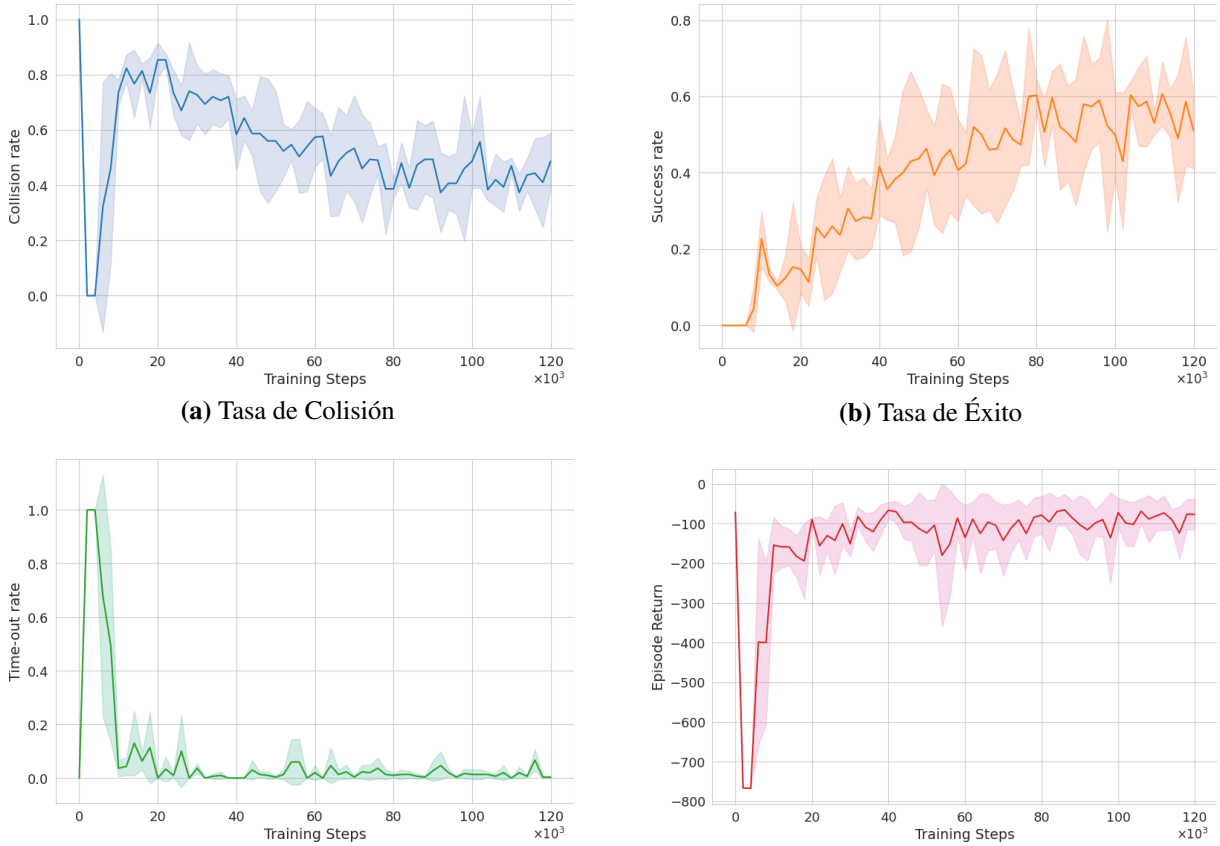
Los valores expuestos en la tabla precedente revelan que la suma de las tres tasas de evaluación (*success rate, collision rate* y *time-out rate*) es igual a 1.0, pues abarcan los tres escenarios posibles que pueden acabar con un episodio de evaluación: llegar a la región objetivo, colisionar con un obstáculo del mundo o alcanzar un máximo de 400 interacciones agente-ambiente. Para el caso en que el agente esta equipado por un sensor que abarca 360° de campo de visión, el mejor rendimiento se alcanza a los $78 \cdot 10^3$ *steps* de entrenamiento y en el caso del agente PCL, la mejor política se obtiene a los $106 \cdot 10^3$ *steps*.

Los resultados obtenidos muestran que el agente que utiliza nube de puntos como método para codificar observaciones del mundo presenta un mejor desempeño que el agente que utiliza un único sensor con 360 grados de campo de visión, con una tasa de éxito del 90 %. El coeficiente SPL, revela que las trayectorias que sigue el agente que utiliza PCL son mejores que las que sigue el agente con el otro tipo de observaciones, pues posee un valor más cercano a 1.0.

En cuanto a los valores que toma la recompensa episódica promedio en ambos casos, no se puede hacer una comparación justa, pues el escalador de la recompensa de navegación para el caso en que se usa PCL es mayor y por ende, la recompensa más negativa.

Las Figuras 5.8 y 5.9 muestran las curvas de la evolución de la tasa de colisión, tasa de éxito, tasa de tiempo límite y recompensa episódica en los entrenamientos del agente con observaciones del sensor 360° FOV y nube de puntos respectivamente. En ambas figuras se aprecia que, a medida que avanzan los pasos de entrenamiento, la tasa de tiempo límite (véase Figuras 5.8c y 5.9c) y la recompensa episódica (véase Figuras 5.8d y 5.9d) exhiben un comportamiento asintótico. Se evidencia un marcado descenso en la *time-out rate* y un aumento evidente en la *episode return*.

Los gráficos muestran además, el momento al inicio del entrenamiento en que la red neuronal del actor se satura y entrega valores extremos. En estas situaciones, la velocidad lineal del robot es igual a 0 y su velocidad angular es máxima, de modo que gira en torno a su propio eje sin desplazarse. Es por esto que durante estos *steps*, la tasa de tiempo límite es 1.0 y la recompensa alcanza valores muy negativos. Este fenómeno también es observado en [43], por lo que es posible que provenga de la función de recompensa.



(a) Tasa de Colisión

(b) Tasa de Éxito

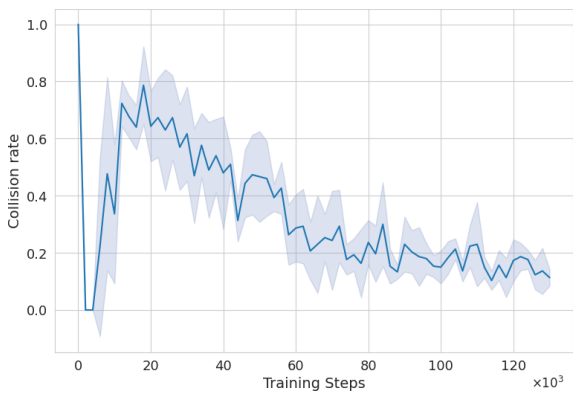
(c) Tasa de Tiempo Límite

(d) Recompensa Episódica

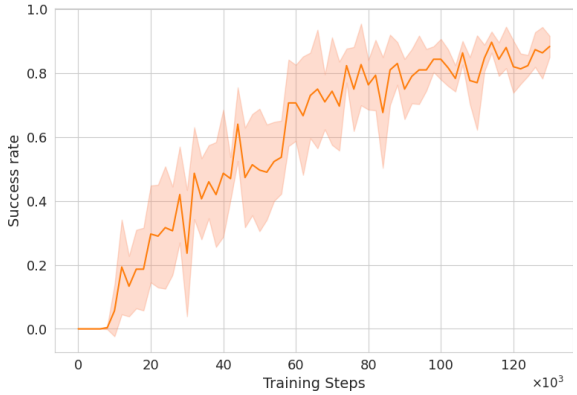
Figura 5.8: Evolución de tasa de colisión, tasa de éxito, tasa de tiempo límite y retorno episódico promedio a lo largo de los *steps* de entrenamiento para agente con observaciones de 360° de campo de visión. Cada punto de las curvas representa el promedio y la desviación estándar de cada una de las métricas obtenidas a partir de 100 episodios de evaluación para 3 entrenamientos independientes.

Para ambas configuraciones (PCL y 360° FOV), se observa que la tasa de colisión y de éxito presentan un comportamiento oscilante a lo largo de los *steps* de entrenamiento. No obstante, se observa que cuando el agente se encuentra equipado con dos sensores de profundidad a partir de los cuales construye una nube de puntos del entorno (Figuras 5.9a y 5.9b), la tasa de colisión comienza a decrecer mientras que la de éxito crece a lo largo que aumentan los pasos de entrenamiento. De manera general, se observa que a partir de los $60 \cdot 10^3$ *steps* la CR no supera el 30 % mientras que la SR no baja de 70 % aproximadamente. Esto revela que el agente logra aprender a evadir las colisiones y alcanzar su objetivo.

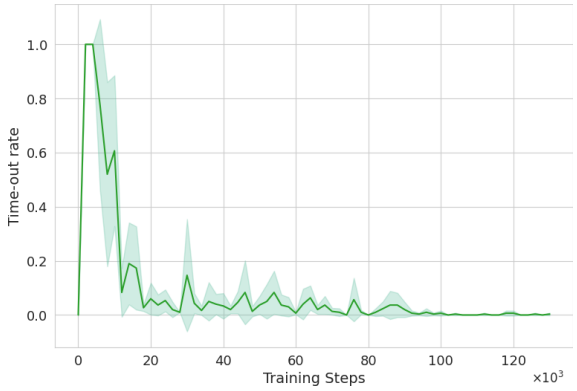
En cuanto a los gráficos que muestran estas métricas a lo largo del entrenamiento utilizando las mediciones de 360° de campo de visión (Figuras 5.8a y 5.8b), se aprecia que a grandes rasgos, la tasa de colisión disminuye y la de éxito aumenta, pero sus valores son oscilantes y no revelan una tendencia tan clara como en el caso anterior. Al observar dichas figuras con detención, se observa el *peak* reportado en la tabla 5.2 a los $78 \cdot 10^3$ *steps* de entrenamiento, donde la tasa de éxito presenta su valor más elevado y la tasa de colisión su valor más bajo (sin considerar los casos en que el *time-out rate* es igual a 1.0 al inicio del entrenamiento).



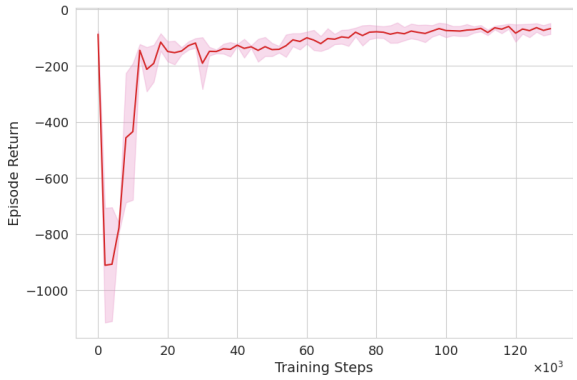
(a) Tasa de Colisión



(b) Tasa de Éxito



(c) Tasa de Tiempo Límite



(d) Recompensa Episódica

Figura 5.9: Evolución de tasa de colisión, tasa de éxito, tasa de tiempo límite y retorno episódico promedio a lo largo de los *steps* de entrenamiento para agente con observaciones de nube de puntos. Cada punto de las curvas representa el promedio y la desviación estándar de cada una de las métricas obtenidas a partir de 100 episodios de evaluación para 3 entrenamientos independientes.

5.2.2. Resultados del entrenamiento en el mundo complejo

El entorno complejo se compone de una región cuadrada que contiene múltiples obstáculos de diferentes formas repartidos por el entorno, generando pasadizos de diferente ancho (todos permiten el paso del robot). Esto hace que para el agente sea más difícil encontrar una política óptima. Al comienzo de cada episodio de entrenamiento se localiza el robot en una posición y orientación aleatorias (Figura 5.7b).

Al igual que en el caso del mundo simple, los valores de r^t_{danger}, $r^t_{success}$, $r^t_{collision}$ y α_{rew} que terminan la definición de la función de recompensa son determinados experimentalmente. Los valores encontrados están resumidos en la Tabla 5.3, tanto para observaciones con un único sensor LiDAR de 360° de campo de visión como para los dos sensores, a partir de los cuales se construye la nube de puntos.

Tabla 5.3: Parámetros de la función de recompensa para entrenamiento de algoritmo *Dreamer* en entorno de navegación complejo.

Observación	r^t_{danger}	$r^t_{success}$	$r^t_{collision}$	α_{rew}
360° FOV	-10	100	-100	0.8
PCL	-10	100	-100	1.0

En cuanto a la optimización de los pesos de las redes neuronales que componen el modelo del mundo y la política, se realiza una actualización de parámetros luego de cada *step* de entrenamiento para el caso en que el agente utiliza observaciones 360° FOV y dos actualizaciones en el caso en que el robot utiliza PCL.

Utilizando los parámetros que caracterizan la función de recompensa en el entorno complejo y las especificaciones sobre los ajustes de parámetros, se presentan en la Tabla 5.4 los resultados de la evaluación de la política óptima obtenida en los entrenamientos realizados. Al igual que para el caso del mundo simple, la política encontrada se obtiene tras evaluar el modelo del mundo y el actor cada 4000 *steps* de tres entrenamientos ejecutados bajo la misma configuración, promediando las métricas de desempeño y seleccionando la iteración para la cual se presentan las mejores métricas. Los resultados proporcionados abarcan tanto la configuración del agente que emplea observaciones provenientes de un sensor con un campo de visión (FOV) de 360 grados, como aquella del agente que utiliza una nube de puntos para codificar la información ambiental.

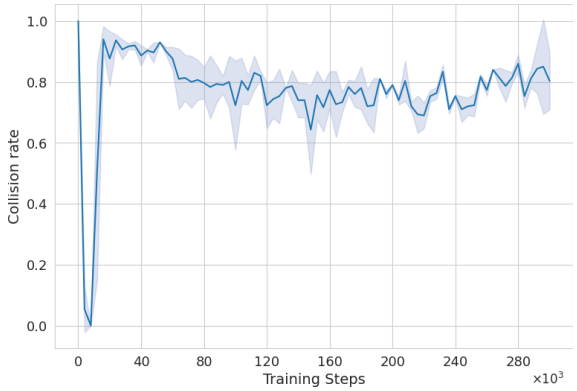
Tabla 5.4: Promedio y desviación estándar de las métricas de desempeño obtenidas tras ejecutar 100 episodios de evaluación en 3 entrenamientos independientes del agente de navegación en mundo complejo.

Obs.	***Steps***	**CR**	**TR**	**SR**	**SPL**	**AER**
360° FOV	$148 \cdot 10^3$	0.6 ± 0.146	0.0 ± 0.0	0.4 ± 0.146	0.313 ± 0.139	-130.13 ± 19.97
PCL	$160 \cdot 10^3$	0.5 ± 0.071	0.0 ± 0.0	0.5 ± 0.075	0.347 ± 0.695	-274.89 ± 65.07

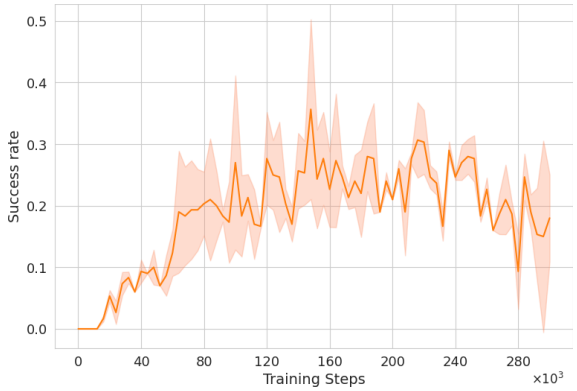
Los datos presentados en la tabla muestran que, al igual que en el entorno simple, el agente PCL obtiene un mejor desempeño que el agente 360° FOV. Este resultado se respalda por una tasa de éxito más alta, alcanzando un valor de 50 % de éxito, en comparación con el agente 360° FOV que logra un 40 % de *success rate*. Además, el agente PCL presenta una tasa de colisión inferior, registrando un valor de 50 %, mientras que el agente 360° FOV presenta una tasa de colisión más alta, llegando a 60 %.

Otra métrica que revela una mejor política por parte del agente PCL, es el coeficiente *success weighted by path length* (SPL). Este evidencia que las trayectorias seguidas por el agente PCL son ligeramente mejores que las trayectorias del agente 360° FOV. En efecto, el robot que posee observaciones de nubes de puntos presenta un factor SPL de 0.35, mientras que en el otro caso, el agente obtiene un valor de 0.31.

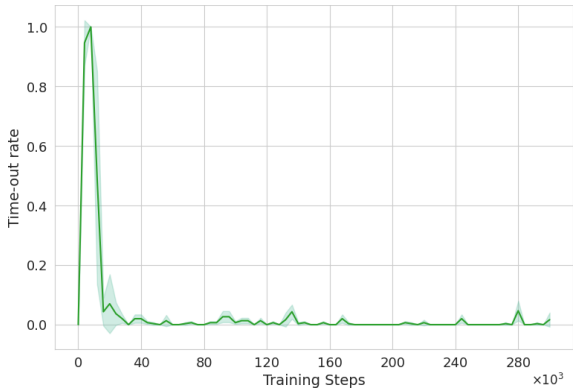
Al igual que en el caso anterior, no se puede hacer una comparación equitativa en cuanto al retorno episódico promedio entre las dos configuraciones, pues el escalador de la recompensa es menor en la configuración 360° FOV, por lo que la recompensa de navegación se mueve en un rango de valores menor.



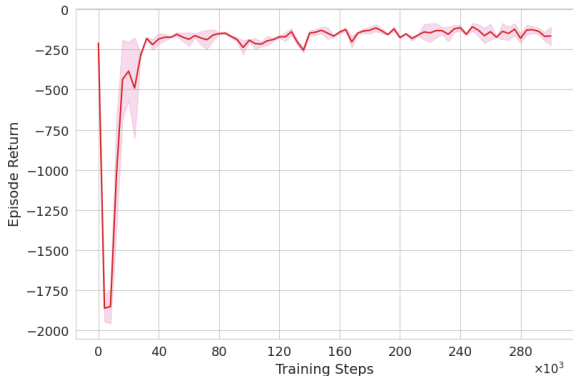
(a) Tasa de Colisión



(b) Tasa de Éxito



(c) Tasa de Tiempo Límite



(d) Recompensa Episódica

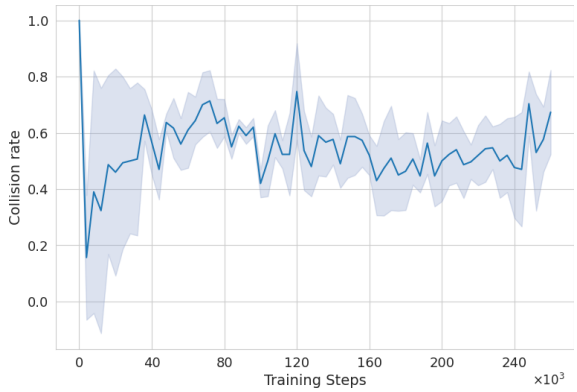
Figura 5.10: Evolución de tasa de colisión, tasa de éxito, tasa de tiempo límite y retorno episódico promedio a lo largo de los *steps* de entrenamiento para agente con observaciones de 360° de campo de visión. Cada punto de las curvas representa el promedio y la desviación estándar de cada una de las métricas obtenidas a partir de 100 episodios de evaluación para 3 entrenamientos independientes.

En las Figuras 5.10 y 5.11 se aprecia el progreso de las tasas de colisión, éxito, tiempo límite y recompensa episódica promedio a lo largo de los *steps* de entrenamiento. A pesar de que las métricas señaladas en la Tabla 5.4 sugieren que el desempeño de la configuración PCL es mejor, es interesante observar el fenómeno que ocurre en los gráficos de la *time-out rate* (Figuras 5.10c y 5.11c) y del *average episode return* (Figuras 5.10d y 5.11d). Se aprecia cómo, para el agente 360° FOV las dos métricas presentan un claro comportamiento asintótico en los primeros $40 \cdot 10^3$ pasos de entrenamiento, mientras que para la configuración PCL, tanto la tasa de tiempo límite como la recompensa episódica promedio toman valores más oscilantes sin presentar un comportamiento asintótico tan evidente.

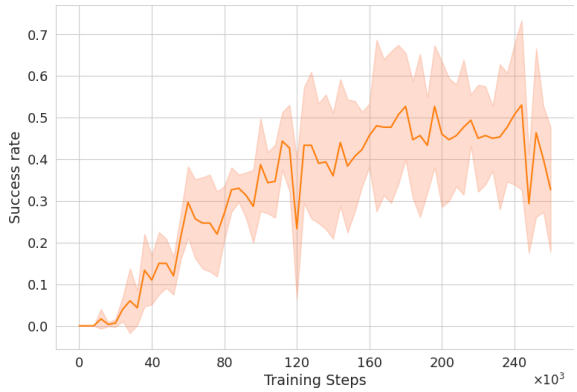
De manera general, se observa que en ambos casos el comportamiento que sigue la tasa de éxito a lo largo del entrenamiento es opuesto al que toma la tasa de colisión. Dicho patrón es lógico, pues la tasa de tiempo límite se mantiene relativamente baja a partir de los 40 mil *steps*. Es interesante recalcar también, el estancamiento y las oscilaciones de los valores que toman las tasas de éxito y de colisión a lo largo del entrenamiento, tanto para el agente 360° FOV, como para PCL. El fenómeno observado revela que los agentes no logran encontrar una política robusta y dejan de aprender a partir de aproximadamente los $160 \cdot 10^3$ *steps* de entrenamiento. Las posibles explicaciones para

este comportamiento son abordadas en la Sección 5.2.3.

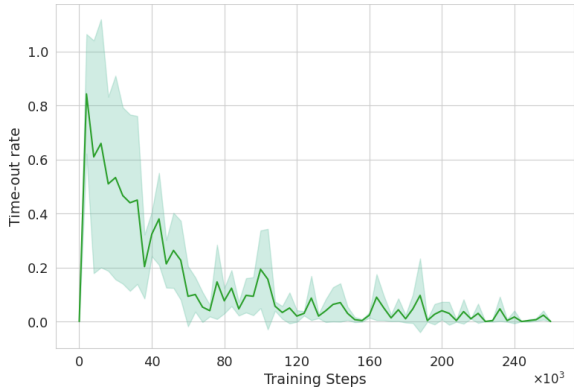
No obstante, las figuras 5.10a, 5.10b y 5.11a, 5.11b muestran el crecimiento de la *success rate* y por consiguiente, el decrecimiento de la *collision rate* en los primeros $150 \cdot 10^3$ *steps* de entrenamiento, tanto para el agente 360° FOV, como para PCL. Se observa que en el caso del agente que usa nube de puntos para codificar sus observaciones, dicho aumento es más pronunciado. Estas curvas evidencian los puntos reportados en la Tabla 5.4, donde se alcanza un valor más elevado de tasa de éxito para la configuración PCL.



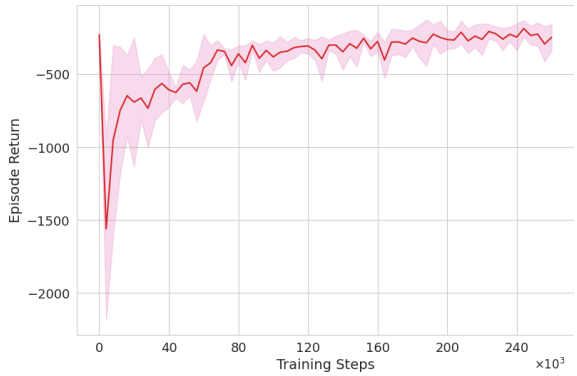
(a) Tasa de Colisión



(b) Tasa de Éxito



(c) Tasa de Tiempo Límite



(d) Recompensa Episódica

Figura 5.11: Evolución de tasa de colisión, tasa de éxito, tasa de tiempo límite y retorno episódico promedio a lo largo de los *steps* de entrenamiento para agente con observaciones de nube de puntos. Cada punto de las curvas representa el promedio y la desviación estándar de cada una de las métricas obtenidas a partir de 100 episodios de evaluación para 3 entrenamientos independientes.

A pesar de que el agente PCL presenta un mejor desempeño que el agente 360°, este no presenta una convergencia en términos de *success rate*. Se logra aumentar la recompensa episódica considerablemente y el SR, pero se observa que el actor del agente no logra obtener una política óptima, pues el número de colisiones que presenta, es considerable. Sin embargo, la cantidad de interacciones agente ambiente que se realizan para alcanzar el mejor conjunto de métricas del entrenamiento es pequeña para la complejidad del entorno.

5.2.3. Discusión

A continuación, se examina la convergencia del algoritmo *Dreamer* en los entornos de navegación según los resultados presentados en las secciones anteriores. Concretamente, se destaca que tanto en el mundo simple como en el mundo complejo, el agente dotado de observaciones provenientes de una nube de puntos obtiene un mejor rendimiento que el agente que utiliza las mediciones del sensor con 360 grados de campo de visión.

Si bien en las cuatro configuraciones agente-ambiente analizadas se logra aumentar la recompensa episódica promedio y la tasa de éxito, solamente el agente PCL en el mundo simple logra obtener una política autosuficiente y autónoma. Dicho escenario es el único que logra superar un 90 % de tasa de éxito. No obstante, la política encontrada con el agente PCL en el mundo simple, no logra generalizar bien, pues al momento de evaluarla en el entorno complejo, muestra un desempeño deficiente (ver Anexos A 1).

Una explicación plausible para este resultado radica en las numerosas regiones navegables del entorno complejo que otorgan al agente recompensas de peligro, a diferencia del entorno simple donde el robot no enfrenta desafíos similares. Además, dado que el algoritmo utiliza representaciones "ìmaginadas" para derivar una política, se vuelve más susceptible a cambios en la recompensa. Esto se evidencia en los valores que parametrizan la función que define el retorno, los cuales varían significativamente de un entorno a otro. En este contexto, la falta de generalización al evaluar en el mundo complejo puede atribuirse a las marcadas diferencias en las parametrizaciones de la función de recompensa entre ambos entornos.

Por otro lado, una interpretación que podría explicar el peor rendimiento del agente 360° FOV en ambos entornos es que sus observaciones no le proveen información clara sobre los obstáculos que lo rodean. Al codificar mediciones de rango que se encuentran en todas las direcciones, el agente no logra discernir correctamente entre los obstáculos que tiene frentea él o en otras direcciones, haciendo más confusa su percepción del entorno. Más aún, el escalador de la recompensa con el que se obtiene la mejor política cuando se equipa al agente con el sensor 360° FOV, al ser menor que en el caso PCL, revela que justamente se enfrenta a complicaciones mientras navega, pues necesita atribuirle valores más pequeños a la recompensa obtenida en dichos estados para obtener una política más óptima.

Durante la experimentación del presente trabajo, se realiza también la prueba de entrenar al agente en el mundo simple y luego utilizar los pesos de las redes obtenidos en el punto de convergencia, para inicializar los parámetros de dichas redes neuronales en un entrenamiento del mundo complejo, algo muy similar a lo que se conoce como *"fine-tunning"* [50]. Al no obtener buenos resultados, no se prosigue con dicho experimento. Una posible explicación para el fracaso de esta estrategia, es la diferencia en la parametrización de la función de recompensa que poseen los entornos para que el agente logre aprender una política.

Una posible explicación para las oscilaciones que presentan las curvas de *success rate* y *collision rate* en el mundo complejo, es el sobre ajuste de parámetros de las redes neuronales que conforman el modelo del mundo. Cuando ocurre este fenómeno, conocido como *overfitting*, las redes neuronales no logran representar de manera precisa las observaciones recibidas por el agente. En consecuencia, cuando el agente se enfrenta a información que no ha procesado previamente (por ejemplo, al encontrarse en estados que no ha experimentado), no logra codificarla adecuadamente

y pierde la capacidad de generalizar su toma de decisiones. Dado que el mundo complejo presenta una gran cantidad de obstáculos de diversas formas y el agente tiene múltiples posiciones iniciales con orientaciones distintas, no puede obtener una representación robusta que le permita imaginar trayectorias correctamente.

Las fluctuaciones en la tasa de éxito a lo largo de los *steps* de entrenamiento pueden ser atribuidas también, a los hiperparámetros que configuran el *replay buffer* y las actualizaciones de parámetros. Es posible que la cantidad de muestras necesarias para actualizar los parámetros de las redes que representan la dinámica del mundo y la política no sea la que se establecida en la experimentación presentada en este trabajo. A pesar de hacer modificaciones en la cantidad de experiencias iniciales recolectadas por el agente, el *batch size* o el largo de las secuencias muestreadas, no se encuentra un ajuste óptimo.

Las curvas de evolución de las métricas presentadas en las secciones anteriores y el proceso de experimentación realizado para encontrar los resultados presentados, revelan también, que el algoritmo es sumamente sensible ante variaciones en las observaciones, función de recompensa, ruido de exploración y cantidad de actualizaciones de parámetros que se realizan a las redes neuronales que componen *Dreamer*. Los hallazgos presentados, demuestran la importancia del diseño de las observaciones y parametrizaciones asociadas para que un agente autónomo logre resolver el problema de navegación empleando este algoritmo de aprendizaje reforzado.

Capítulo 6

Conclusiones

6.1. Conclusiones del trabajo realizado

El trabajo expuesto ha abordado el desarrollo e implementación del algoritmo *Dreamer* para resolver el problema de navegación. En concreto, se ha validado el correcto funcionamiento del método en entornos clásicos de aprendizaje reforzado, para luego adaptarlo a entornos donde se simula el problema de navegación.

En una primera instancia, los entornos utilizados del conjunto de *Deep Mind Control Suite* permitieron validar la correcta ejecución y aprendizaje de los agentes. En particular, se observó que tanto para el entorno *cartpole* como para *cheetah*, el agente logró encontrar una política que le permite maximizar su recompensa y controlar su tarea en particular. Más aún, mediante el entrenamiento de las políticas en estos entornos, se pudo demostrar que el aprendizaje del agente era bastante rápido, pues el mayor crecimiento del retorno episódico promedio y decrecimiento de la función de pérdida del actor se concentra en los primeros *steps* de entrenamiento.

En segundo lugar, se adaptó el desarrollo realizado de *Dreamer* a entornos de navegación. Concretamente, se implementaron dos ambientes de navegación con niveles de dificultad distintos. En cada uno de estos entornos, se llevaron a cabo pruebas con dos agentes que se diferencian en el tipo de observación que utilizan y en consecuencia, en el codificador empleado para obtener representaciones de dichas observaciones. Inicialmente, se dotó al agente con un sensor de distancia que abarca un campo de visión de 360 grados. Luego, se implementaron dos sensores que, a través de sus mediciones de rango, proporcionan nubes de puntos 2D como observaciones del entorno.

Los resultados obtenidos permiten concluir que el algoritmo *Dreamer* posee un gran potencial para entregar soluciones eficientes a tareas complejas (en cuanto a interacciones agente-ambiente). Tanto en el entorno simple, como en el entorno complejo de navegación, el agente requiere de pocas interacciones para lograr aumentar su tasa de éxito y disminuir sus colisiones.

Se concluye también, que el agente dotado de observaciones provenientes de nubes de puntos obtiene un mejor desempeño que el robot que utiliza las mediciones de rango con 360° FOV. En ambos entornos el agente PCL aprende una mejor política en cuanto a *success rate* y *collision rate*. En particular, para la configuración del ambiente complejo, dicha política es alcanzada en una

cantidad de interacciones agente-ambiente significativamente menor.

Si bien no se logró encontrar una política suficientemente robusta para un agente de navegación autónomo en el mundo complejo, se demuestra que el algoritmo posee un gran potencial para resolver tareas de planificación con desafíos importantes. La sensibilidad del desempeño de *Dreamer* ante cambios en los parámetros de su entrenamiento, hizo que el proceso de experimentación y ajuste de parámetros fuera bastante complicado. Esto motiva seguir probando nuevas configuraciones que permitan exprimir todo el potencial que promete el algoritmo.

6.2. Trabajo a futuro

A medida que la robótica avanza hacia un futuro cada vez más autónomo, la navegación de robots se ha convertido en un área crítica de investigación. En el transcurso de esta tesis se ha abordado el desafío de la navegación robótica, empleando técnicas avanzadas de aprendizaje reforzado. La exploración y resolución de este problema han proporcionado valiosas perspectivas sobre la mejora de la capacidad de los robots para interactuar eficientemente con su entorno.

Sin embargo, como cualquier campo en constante evolución, la navegación robótica presenta oportunidades y desafíos continuos que merecen atención en futuras investigaciones. Esta sección se dedica a esbozar dos direcciones potenciales para el trabajo futuro en este campo.

6.2.1. Imágenes como observaciones

La presente investigación ha centrado su atención en la utilización exclusiva de observaciones simbólicas del entorno, es decir, datos vectoriales que contienen información relevante sobre el ambiente. No obstante, es importante señalar que el algoritmo *Dreamer* fue concebido originalmente para procesar imágenes como observaciones del agente, haciendo uso de redes neuronales convolucionales.

Considerando esta perspectiva, una vía prometedora para futuras investigaciones consistiría en abordar el mismo problema de navegación, pero dotando al robot con cámaras o proporcionándole imágenes del sistema como observaciones. Esta extensión del enfoque actual podría ofrecer mejoras sustanciales en la eficacia del algoritmo, pues mediante observaciones visuales las redes neuronales que componen el modelo del mundo podrían capturar de manera más directa y completa la información del entorno. Con esto, el agente lograría aprender una política más robusta y, por ende, mejorar su rendimiento.

El tener más información sobre el entorno en el que se desenvuelve el robot y en particular, sobre los obstáculos a los que se enfrenta, sería crucial para reducir el número de colisiones. Los resultados presentados en este trabajo revelan que el agente siempre logra llegar a su destino en caso de que se le permita continuar luego de colisionar, en otras palabras, con las políticas encontradas, no hay casos en los que un episodio acabe por *time-out*. Por lo tanto, sería interesante comparar las bases presentadas con un desarrollo basado en aprendizaje a partir de pixeles.

Más aún, la introducción de imágenes como observaciones podría facilitar la obtención de políticas de navegación de manera más eficiente, posiblemente reduciendo la cantidad de interacciones requeridas entre el agente y el entorno para alcanzar un rendimiento óptimo o encontrando una

política que entregue un mayor retorno episódico. Esta expansión del alcance del estudio abre nuevas posibilidades para el perfeccionamiento del algoritmo *Dreamer* en el contexto de la navegación basada en información visual, presentando un terreno fértil para investigaciones futuras en la mejora de la eficiencia y adaptabilidad de los agentes en entornos dinámicos.

En este contexto, estudios previos han respaldado que el uso de imágenes RGB del entorno puede conducir a mejoras significativas en el rendimiento de los algoritmos de navegación. Por ejemplo, el trabajo de [51] demostró que la inclusión de información visual a través de imágenes RGB en el proceso de toma de decisiones del agente resultó en una mejora considerable en la capacidad de navegación y la eficiencia del aprendizaje en un entorno dinámico. Este hallazgo refuerza la hipótesis de que la transición a observaciones visuales en el marco de *Dreamer* podría ser un camino valioso para optimizar la navegación del agente y reducir la cantidad de interacciones agente-ambiente.

6.2.2. Navegación Multiagente

El presente trabajo, aborda únicamente entornos de navegación estáticos. Hasta el presente, han habido grandes avances en escenarios de este tipo, pero aún queda un gran espacio de investigación para resolver problemas de evasión de colisiones en entornos dinámicos [52].

En el contexto de entornos de ejecución dinámicos, una ruta de desarrollo prometedora se encuentra en la expansión de los enfoques actuales hacia el problema de la navegación multiagente, cuyo foco es el diseño y control de múltiples agentes autónomos que deben moverse de manera eficiente y segura en un entorno compartido que podría ser desconocido para alcanzar objetivos de navegación.

La transición hacia la navegación multiagente representa un paso natural para abordar escenarios más realistas, pues plantea desafíos adicionales en comparación con la navegación de un solo agente. El mismo problema con múltiples robots podría implicar por un lado, la colaboración, comunicación y toma de decisiones conjuntas entre agentes o por otro, generar un ambiente de competencia donde podrían surgir nuevos comportamiento.

En el aprendizaje reforzado multiagente, los Procesos de Decisión de Markov son generalizados como un juego estocástico o *Markov Game* (MG) [53]. El aprendizaje en entornos multiagente es más complejo que en ambientes con un solo agente, pues estos no solo interactúan con el entorno, sino que también entre ellos [54]. Existen dos métodos clásicos para abordar problemas que involucren a múltiples robots: aprendizaje centralizado y descentralizado.

Los algoritmos que siguen una propuesta centralizada consideran a todos los agentes como iguales y se basan en compartir todas las experiencias de modo que se aprenda una única política. Dicha política indica a cada robot qué acción tomar dado su estado actual y les permite desenvolverse correctamente para resolver una tarea específica. Por su parte, los métodos descentralizados utilizan directamente algoritmos de agente único de modo que los robots aprendan de manera independiente (cada agente aprende su propia política, considerando los otros agentes como parte del entorno).

Existe también otra una forma de abordar los problemas que involucran múltiples agentes conocida como *Centralized Training with Descentralized Excecution*. Este método busca obtener las ventajas de los dos paradigmas mencionados anteriormente. Durante el proceso de entrenamiento,

cada agente extrae información global del ambiente a partir de una variedad de métodos centralizados, como *action-state value*, trayectorias históricas de un *experience buffer* global y otros procedimientos basados en la información explícita de sensores [33]. A partir de estas experiencias, se construye una política colectiva que busca maximizar las recompensas de de todos los agentes. Para recaudar nueva información, la ejecución de dicha política se hace de manera independiente y cada agente actúa en base a sus propias observaciones locales.

Utilizar el trabajo presentado en esta memoria para una futura implementación del problema de navegación multiagente (también conocido como *multiagent pathfinding and collision avoidance problem*) es casi directa usando un entrenamiento centralizado con ejecución descentralizada. En particular, se podría realizar tanto un aprendizaje del mundo como de la política de manera centralizada y una recolección de nuevas experiencias de forma independiente. Así, la convergencia sería simultánea para todos los agentes mientras que cada agente explora el mundo y recauda información.

Con el fin de utilizar las bases presentadas en esta tesis, para que el aprendizaje sea continuo a lo largo del entrenamiento y los agentes logren obtener una política robusta, se podría realizar en dos fases diferentes: entorno con obstáculos estáticos (agentes aislados dentro del entorno de ejecución con objetivos en su propia región) y entorno con obstáculos dinámicos (otros agentes). Así, en una primera instancia, los agentes aprenden un modelo del mundo y una política suficientemente robustas siguiendo una metodología similar a la propuesta en el presente escrito. En una segunda instancia, se permitirían las interacciones entre los múltiples robots con el objetivo de que aprendan a no colisionar entre ellos y a llegar a sus respectivos *targets*.

En resumen, el presente trabajo ha explorado con éxito la aplicación de técnicas avanzadas de aprendizaje reforzado en el campo de la navegación robótica, centrándose en entornos estáticos. Sin embargo, para seguir avanzando hacia escenarios más realistas y dinámicos, se propone una dirección de investigación futura centrada en la navegación multiagente. Esta expansión hacia entornos dinámicos plantea desafíos adicionales, como la colaboración y la competencia entre robots autónomos. Se han discutido diferentes enfoques, desde métodos centralizados hasta descentralizados, destacando la prometedora estrategia de *Centralized Training with Descentralized Execution*. La resolución de este problema tiene un gran potencial para diversas aplicaciones ingenieriles como exploración marina, agricultura inteligente, rescate ante desastres, entre otras [55].

La implementación de la metodología sugerida plantea diversos retos, desde la necesidad de establecer un aprendizaje continuo entre las fases de entrenamiento mencionadas hasta la búsqueda de hiperparámetros óptimos para lograr la convergencia del algoritmo o encontrar una política que logre generalizar bien, pues no es trivial que un robot se desenvuelva de manera correcta en un escenario diferente al que fue entrenado. Además, se presenta el desafío de gestionar la incertidumbre en los estados y desarrollar un simulador multiagente capaz de facilitar las transiciones de todos los robots. Este conjunto de desafíos subraya la complejidad inherente al proceso, destacando la importancia de abordar cada aspecto de manera cuidadosa y estratégica.

Bibliografía

[1] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[2] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[3] Jose Ricardo Sanchez-Ibanez, Carlos J Perez-del Pulgar, and Alfonso García-Cerezo. Path planning for autonomous mobile robots: A review. *Sensors*, 21(23):7898, 2021.

[4] Xuesu Xiao, Bo Liu, Garrett Warnell, and Peter Stone. Motion planning and control for mobile robot navigation using machine learning: a survey. *Autonomous Robots*, 46(5):569–597, 2022.

[5] Shuhuan Wen, Yanfang Zhao, Xiao Yuan, Zongtao Wang, Dan Zhang, and Luigi Manfredi. Path planning for active slam based on deep reinforcement learning under unknown environments. *Intelligent Service Robotics*, 13:263–272, 2020.

[6] Manh Luong and Cuong Pham. Incremental learning for autonomous navigation of mobile robots based on deep reinforcement learning. *Journal of Intelligent & Robotic Systems*, 101(1):1, 2021.

[7] Pengyu Yue, Jing Xin, Huan Zhao, Ding Liu, Mao Shan, and Jian Zhang. Experimental research on deep reinforcement learning in autonomous navigation of mobile robot. In *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 1612–1616. IEEE, 2019.

[8] Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

[9] Sean Quinlan and Oussama Khatib. Elastic bands: Connecting path planning and control. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807. IEEE, 1993.

[10] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[11] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval

Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[12] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[13] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[14] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[15] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[17] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite, 2018.

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

[19] Robin M Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*, 2019.

[20] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[21] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[22] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[23] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research*, 37(4-5):421–436, 2018.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G

Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[25] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[26] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[27] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[28] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[29] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In *Conference on Robot Learning*, pages 2226–2240. PMLR, 2023.

[30] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[31] Kai Zhu and Tao Zhang. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology*, 26(5):674–691, 2021.

[32] ZiXuan Liu, Qingchuan Wang, Bingsong Yang, et al. Reinforcement learning-based path planning algorithm for mobile robots. *Wireless Communications and Mobile Computing*, 2022, 2022.

[33] Lu Dong, Zichen He, Chunwei Song, and Changyin Sun. A review of mobile robot motion planning methods: from classical motion planning workflows to reinforcement learning-based architectures. *Journal of Systems Engineering and Electronics*, 34(2):439–459, 2023.

[34] Fanyu Zeng, Chen Wang, and Shuzhi Sam Ge. A survey on visual navigation for artificial agents with deep reinforcement learning. *IEEE Access*, 8:135426–135442, 2020.

[35] Marvin Chancán and Michael Milford. Citylearn: Diverse real-world environments for sample-efficient navigation policy learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1697–1704. IEEE, 2020.

[36] Shaohua Lv, Yanjie Li, Qi Liu, Jianqi Gao, Xizheng Pang, and Meiling Chen. A deep safe reinforcement learning approach for mapless navigation. In *2021 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1520–1525. IEEE, 2021.

[37] Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement lear-

ning. *IEEE Access*, 9:153171–153187, 2021.

[38] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[40] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). arxiv 2015. *arXiv preprint arXiv:1511.07289*, 2, 2016.

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

[42] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[43] Francisco Leiva and Javier Ruiz-del Solar. Robust rl-based map-less local planning: Using 2d point clouds as observations. *IEEE Robotics and Automation Letters*, 5(4):5787–5794, 2020.

[44] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, 2017.

[45] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[46] Francisco Leiva, Kenzo Lobos-Tsunekawa, and Javier Ruiz-del Solar. Collision avoidance for indoor service robots through multimodal deep reinforcement learning. In *RoboCup 2019: Robot World Cup XXIII 23*, pages 140–153. Springer, 2019.

[47] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

[48] Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *Advances in Neural Information Processing Systems*, 33:741–752, 2020.

[49] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[50] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[51] Wei Zhu and Mitsuhiro Hayashibe. Autonomous navigation system in pedestrian scenarios using a dreamer-based motion planner. *IEEE Robotics and Automation Letters*, 2023.

[52] Wenhao Ding, Shuaijun Li, Huihuan Qian, and Yongquan Chen. Hierarchical reinforcement learning framework towards multi-agent navigation. In *2018 IEEE international conference on robotics and biomimetics (ROBIO)*, pages 237–242. IEEE, 2018.

[53] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 50(9):3826–3839, 2020.

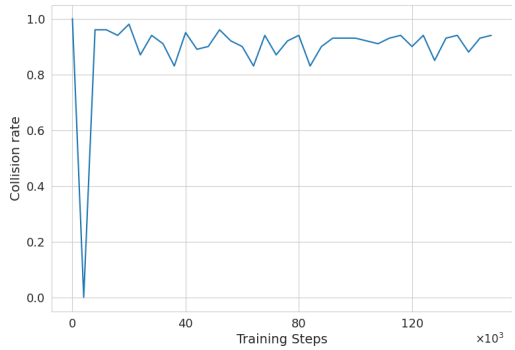
[54] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

[55] Zifan Xu, Bo Liu, Xuesu Xiao, Anirudh Nair, and Peter Stone. Benchmarking reinforcement learning techniques for autonomous navigation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9224–9230. IEEE, 2023.

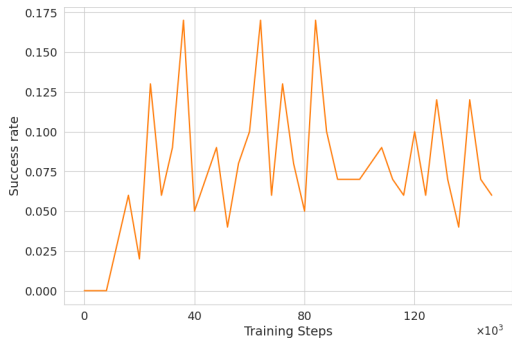
Anexo

A 1. Entorno de Navegación

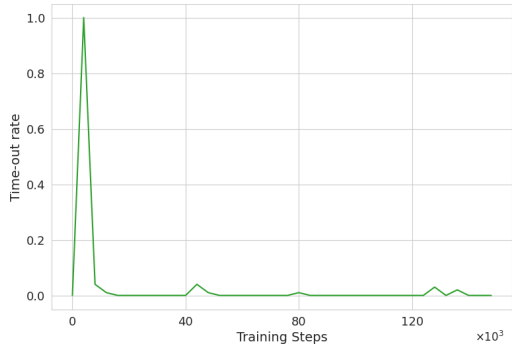
La Figura 6.1 muestra los resultados de la evaluación del entrenamiento con mejores resultados del mundo simple, en el mundo complejo. Se observa que a pesar de aumentar el retorno episódico, no se logran evadir las colisiones y llegar a la región objetivo con éxito.



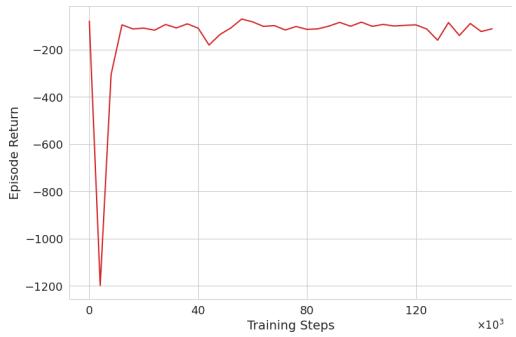
(a) Tasa de Colisión



(b) Tasa de Éxito



(c) Tasa de Tiempo Límite



(d) Recompensa Episódica

Figura 6.1: Evolución de tasa de colisión, tasa de éxito, tasa de tiempo límite y retorno episódico a lo largo de los *steps* de entrenamiento de la evaluación del mejor entrenamiento del entorno simple en el entorno complejo.

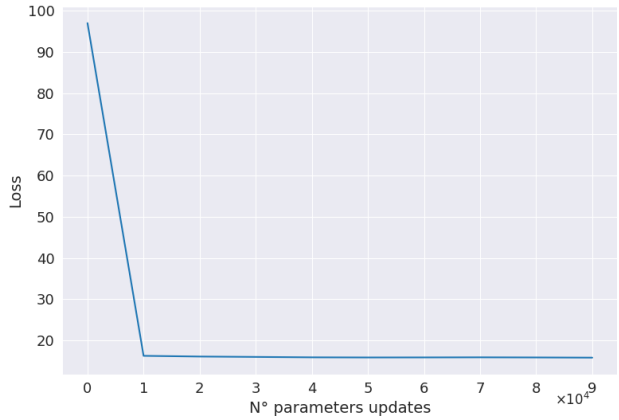
La Tabla 6.1 muestran el resultado de las mejores métricas obtenidas de la evaluación del mejor entrenamiento del mundo simple, en el mundo complejo. Se aprecia que la política aprendida en el mundo simple, no funciona en el mundo complejo con la misma configuración del entorno.

Tabla 6.1: Métricas de desempeño agente con mejor política del mundo simple en el mundo complejo.

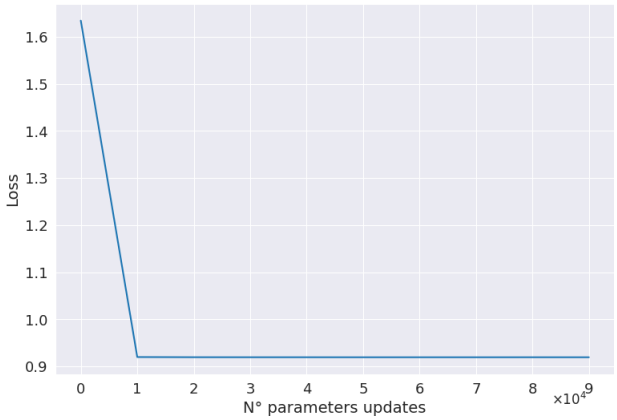
Observación	*Steps* de entrenamiento	CR	TR	SR	SPL	AER
PCL	$36 \cdot 10^3$	0.83	0.0	0.17	0.14	-91.38

A 2. Cheetah

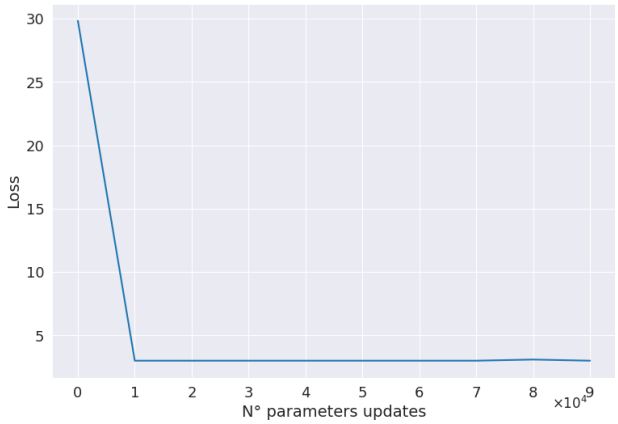
A continuación se exponen los gráficos de las funciones de pérdida (*loss*) que permiten el aprendizaje de la dinámica del mundo a lo largo del entrenamiento para el entorno *cheetah*.



(a) Modelo de Observación



(b) Modelo de Recompensa

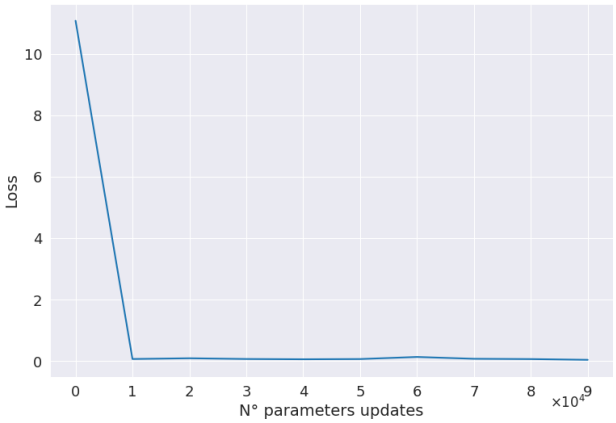


(c) Recurrent Space-State Model

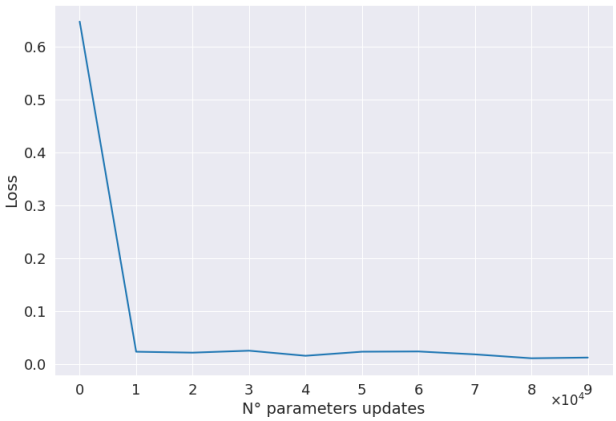
Figura 6.2: Funciones de pérdida modelos a través de actualizaciones de parámetros en entorno *cheetah*.

A 3. Cartpole

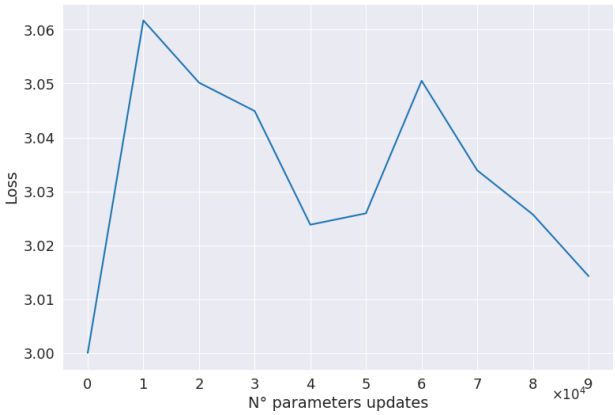
La Figura 6.3 muestra la evolución de las funciones de pérdida (*loss*) que permiten el aprendizaje de la dinámica del mundo a lo largo del entrenamiento para el entorno *cartpole*.



(a) Modelo de Observación



(b) Modelo de Recompensa



(c) Recurrent Space-State Model

Figura 6.3: Funciones de pérdida modelos a través de actualizaciones de parámetros en entorno *cartpole*.