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

ACELERANDO EL APRENDIZAJE REFORZADO DE POLÍTICAS DE UN AGENTE MÓVIL MEDIANTE RETROALIMENTACIÓN CORRECTIVA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA

VALENTINA PAZ BRAIN DE LA BARRA

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR

PROFESOR CO-GUÍA:
FRANCISCO LEIVA CASTRO

MIEMBROS DE LA COMISIÓN:
ISAO PARRA TSUNEKAWA
FRANCISCO RIVERA SERRANO

Este trabajo ha sido parcialmente financiado por proyecto FONDECYT 1201170

SANTIAGO DE CHILE
2022

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA
POR: VALENTINA PAZ BRAIN DE LA BARRA
FECHA: 2022
PROF. GUÍA: JAVIER RUIZ DEL SOLAR

ACELERANDO EL APRENDIZAJE REFORZADO DE POLÍTICAS DE UN AGENTE MÓVIL MEDIANTE RETROALIMENTACIÓN CORRECTIVA

Uno de los grandes desafíos que posee la aplicación del aprendizaje reforzado profundo, es la falta de eficiencia de muestras durante el entrenamiento de un agente, en otras palabras, el agente requiere de muchas iteraciones para converger a un comportamiento deseado.

Un enfoque utilizado para lidiar con este problema, es el de acelerar el aprendizaje de un agente mediante el uso de *feedback* correctivo junto con el uso de recompensas.

Dada esta situación, se propone en este trabajo el incorporar una señal de retroalimentación correctiva para mejorar la eficiencia del aprendizaje de la política o comportamiento de un agente, en particular, la eficiencia de muestras durante su entrenamiento.

Para llevar esto a cabo, se busca implementar un algoritmo híbrido, compuesto por un algoritmo de aprendizaje reforzado profundo (DDPG) y un algoritmo de aprendizaje de máquinas interactivo (D-COACH). Este será empleado sobre un agente robótico móvil, con el fin de llevar a cabo la tarea de navegación autónoma, más en especifico, la de planificación local en 2D, en un ambiente de simulación.

El algoritmo híbrido propuesto, se contrasta con los algoritmos utilizados para su construcción, DDPG y D-COACH, además de un algoritmo secuencial compuesto por ambos. Se compara la eficiencia de muestras entre estos algoritmos y el desempeño que logra cada uno, mediante una evaluación sobre el mismo ambiente de entrenamiento y validación en un ambiente nunca antes visto por el agente.

A todas las hormigas de nuestra sociedad,
que levantan más peso de lo que pueden cargar.
Gracias por su inspiración.

Agradecimientos

Quiero dar gracias a mi familia por apoyarme en todas mis decisiones académicas desde que tengo memoria, con su cariño, paciencia y los recursos que han podido brindarme para lograr llegar hasta donde he llegado, con mención especial a mi mamá: Ivonne De la Barra, que siempre lo ha dado todo para ayudarme a lograr mis objetivos. También agradecer a todas las personas maravillosas que he podido conocer a lo largo de mi vida, que me han brindado su amistad y cariño. A las personas que fueron importantes y ya no están, a las que siguen.. Gracias por existir y haber existido.

Agradecer a todas las personas que me dieron su soporte durante este proceso; la gente del laboratorio de robótica de la universidad y del AMTC, que siempre tuvieron una muy buena disposición para ayudarme tanto con conocimientos como materiales y equipos, que fueron cruciales para el desarrollo de esta memoria; mis amigos y compañeros de la carrera: Pollito Matamoros, Juanjo Gutiérrez, Cris Benavides, Gianluca D'Agostino y la Carlita Kotthoff, que siempre estuvieron ahí para mí, ya sea para carretear, conversar o estudiar; los amigos que hice en la facultad y que siempre buscaron sacar lo mejor de mí: Mati Rojas hippie, Gabo Moya, Ednar y Óscar Pimentel; a Sebonsio López de Maturana, por su apoyo incondicional, siempre buscando un panorama o conversación para distraerme del exceso de estudio y trabajo, por cocinarme y darme ánimo cuando no podía sacar la cabeza del computador; a todos los compañeros y compañeras que tuve en los proyectos y tareas, que hicieron más divertida la vida universitaria.

Finalmente, agradecer a Francisco Leiva por darme todo su apoyo durante la implementación de mi trabajo de título, logré aprender muchas cosas de robótica y aprendizaje reforzado desde cero, que hubiesen sido muy difíciles sin tu co-guía. También al profesor Javier por su constante preocupación y total apoyo al trabajo realizado, gracias.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación y antecedentes

Uno de los grandes paradigmas de aprendizaje de máquinas es el aprendizaje reforzado, el cual busca determinar qué acciones debe escoger un agente al resolver una tarea, a medida que este interactúa con su entorno. La política o comportamiento que el agente aprenda dependerá de las recompensas recibidas por las acciones ejecutadas [1, 2].

El aprendizaje reforzado al ser combinado con el uso de técnicas de aprendizaje profundo, es denominado aprendizaje reforzado profundo. Este campo de investigación ha sido capaz de resolver múltiples tareas de toma de decisiones que poseen un gran grado de complejidad, a la hora de poder representar los estados en los que se encuentra inmerso el agente al interactuar en ambientes con observaciones de datos no estructurados, como imágenes, señales de audio, luz, entre otras. El uso de aprendizaje reforzado profundo, abre una cantidad inmensa de nuevas aplicaciones en los distintos campos del conocimiento tales como la robótica, finanzas, salud entre otros [2]; así el agente puede tener patrones de datos que le permiten tomar mejores decisiones al verse dentro de ambientes de estado continuo no estructurados, es decir donde el agente puede tomar acciones en un espacio continuo mediante el uso de información no estructurada.

Estos agentes artificiales al encontrarse en ambientes desconocidos, deben adaptarse de rápidamente, mediante su interacción con el ambiente, lo cuál no es fácil de conseguir debido a las múltiples iteraciones que debe realizar el agente al aprender de manera autónoma. La idea de introducir el aprendizaje que reciben los humanos en agentes artificiales nace de los inicios de la inteligencia artificial con Alan Turing [3]. Este aprendizaje puede ser obtenido de distintas fuentes, como en el caso de los humanos; estas pueden ser de la experiencia, a partir de sus compañeros (otros agentes que interactúan en un mismo ambiente), profesores (como humanos y modelos que enseñan de distintas formas como realizar la tarea deseada); pruebas de ensayo-error (método que en el agente desarrolla la tarea sin saber si estará correcto o no el resultado, y de haberla desarrollado exitosamente, es aceptada como una solución válida), etc [4]. Dado el contexto de aprendizaje descrito, es natural plantear un método de aprendizaje que involucre al humano como profesor, es ahí donde surge el Aprendizaje de Máquinas Interactivo (IML), el cual incorpora al humano dentro del proceso de aprendizaje del agente, mediante una señal de consejo que puede ser descrita de múltiples formas [5]. En particular, la integración del consejo humano dentro del proceso de aprendizaje, es un método que ha

sido ampliamente utilizado a lo largo de la literatura de aprendizaje reforzado, como un método capaz de acelerar el proceso de aprendizaje mediante la incorporación de recompensas intermedias [5, 6, 7].

Dentro de este trabajo se desea incorporar como señal de consejo la retroalimentación correctiva [8, 9, 10], también llamada retroalimentación instructiva, la cuál implica implícitamente que la acción realizada es incorrecta, donde más allá de la crítica a la acción, se entrega al agente la información sobre la acción correcta. Lo anterior, es empleado con el fin de lograr una optimización y mejora en la calidad de las muestras utilizadas por el agente para su aprendizaje, logrando obtener en otras palabras, una mejora en la eficiencia de muestras, para así acelerar el aprendizaje de un agente autónomo en la realización de distintas tareas. Lo descrito, ha sido aplicado en múltiples contextos y trabajos [11, 12, 13] , incluyendo al humano dentro del *loop* de aprendizaje reforzado profundo de distintas formas, sobre distintos contextos.

Una de las tareas de interés, es la de planificación local 2D [14, 15], la cual se desarrolla dentro del paradigma de navegación robótica [16], y consiste principalmente en el desafío de que un agente robótico móvil, sea capaz de desplazarse en un ambiente 2D a partir de las observaciones realizadas a partir de su interacción con el ambiente instantáneamente, a lo largo de una serie de poses de navegación locales, con el fin de alcanzar el objetivo de navegación determinado por alguna de sus planificaciones globales, esto mientras el agente logra evitar obstáculos en el camino.

1.2. Descripción del problema

Uno de los desafíos que tiene la aplicación del aprendizaje profundo, es la falta de eficiencia en el entrenamiento (interacción del agente con el mundo mediante exploración), es decir, que el agente requiere de muchas iteraciones para poder converger a un comportamiento deseado.

Se plantea como solución a este problema la incorporación de retroalimentación correctiva para mejorar la eficiencia del aprendizaje de la política o comportamiento de un agente, en particular, la eficiencia de muestras durante su entrenamiento.

Se tiene como tarea a resolver por el agente robótico, la de planificación local en navegación robótica, la cual se traduce en el lograr que el agente móvil logre desplazarse hacia su objetivo local, sin chocar con paredes u obstáculos dentro del ambiente en el que se encuentra, con el fin de probar su funcionamiento tanto en el ambiente en el que entrena, como en alguno que no haya visto antes, y así evaluar su desempeño.

Esta tarea se llevará acabo mediante el aprendizaje de la política del agente con el uso de aprendizaje reforzado profundo y aprendizaje de máquinas interactivo, sobre configuraciones experimentales diseñadas en ambientes de simulación y potencialmente en el mundo real, para validar su funcionamiento y buen desempeño de la tarea a realizar.

1.3. Objetivos

1.3.1. Objetivos generales

El objetivo general de este trabajo, consiste en acelerar el entrenamiento de un agente móvil mediante el uso combinado de aprendizaje reforzado profundo y retroalimentación correctiva. En otras palabras, que el agente sea capaz de realizar la tarea de planificación local 2D, de forma de que su aprendizaje sea más rápido al incluir al humano como profesor. Este algoritmo híbrido, debe ser capaz de funcionar en ambientes de simulación, sobre mapas que poseen obstáculos fijos en cada episodio de entrenamiento.

Como agente móvil se utilizará el robot *Clearpath Husky A200* [17], junto con el uso de dos sensores láser *Hokuyo UTM-30LX-EW* [18] dispuestos de modo que se puede tener una visión del ambiente en 360° en torno al agente. Se hará uso de la plataforma de simulación **Gazebo 11** [19] y el *framework* de **ROS-Noetic** [20], para desarrollar una aplicación que permita llevar a cabo el entrenamiento de la política del agente.

1.3.2. Objetivos específicos

- Proponer e implementar un algoritmo que permita incluir retroalimentación correctiva en el aprendizaje de la política de un agente robótico móvil para resolver la tarea de navegación robótica, en un ambiente 2D.
- Validar el algoritmo propuesto a través de una configuración experimental en una plataforma de simulación, utilizando un agente robótico móvil.

Capítulo 2

Marco teórico y estado del arte

2.1. Procesos de decisión de Markov

Los Procesos de Decisión de Markov (MDP)[1] son la formalización clásica del problema de toma de decisión secuencial, donde el efecto de una acción en un estado o situación determinada, influye sobre la obtención de las recompensas inmediatas y los estados subsecuentes junto con las recompensas futuras asociadas a estos. En particular, esta formulación matemática es utilizada para describir, teóricamente, el problema de aprendizaje reforzado, el cual se traduce en el aprendizaje del comportamiento de un agente en un ambiente desconocido, mediante su interacción con éste.

Un Proceso de Decisión de Markov μ, se define como una tupla $\mu = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, donde:

- $\mathcal{S}$ es un conjunto de estados markovianos [1].
- $\mathcal{A}$ un conjunto de acciones.
- $\mathcal{P}(s, a, s')$ es la función de transición de estados que modela la dinámica del MDP. Esta función indica la probabilidad de transición de un estado $s \in \mathcal{S}$ a un estado siguiente $s' \in \mathcal{S}$, dada una acción $a \in \mathcal{A}$, lo cuál se representa mediante la siguiente expresión, $\mathcal{P}(s, a, s') = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$.
- $\mathcal{R}(s, a)$ es la función de recompensas, que indica la recompensa asociada a cada transición de estado, esta se define como $\mathcal{R}(s, a) = \mathbb{E}[r_t | s_t = s, a_t = a]$.

Lo anterior, puede ser descrito como la interacción secuencial entre el agente y el ambiente en cada paso de tiempo discreto $t \in \{1, 2, 3, ..., T\}$. Donde en cada paso de tiempo t, el agente obtiene el estado s_t del ambiente, sobre el cual decide ejecutar una acción a_t. Un paso de tiempo después $(t + 1)$, como consecuencia de la acción ejecutada, el agente obtiene una recompensa numérica r_t dada por $\mathcal{R}(s, a)$, que transiciona al estado siguiente s_{t+1}, dado por $\mathcal{P}(s, a, s')$. En la Figura 2.1, se presenta la estructura de una MDP anteriormente descrita.

[1] Los estados Markovianos [1] son la información disponible que posee el agente en un instante de tiempo $t = 0, 1, .., n$ sobre el ambiente, y que cumplan la propiedad de Markov, o de independencia entre estados. Más en específico, que hay independencia entre el estado futuro (s_{t+1}) de los estados pasados (s_{t-1}), por lo que la información relevante se encuentra en el estado presente o actual (s_t). La propiedad de Markov se presenta mediante la siguiente expresión: $\mathbb{P}(s_{t+1}|s_t) = \mathbb{P}(s_{t+1}|s_t, s_{t-1}, ...s_0)$.

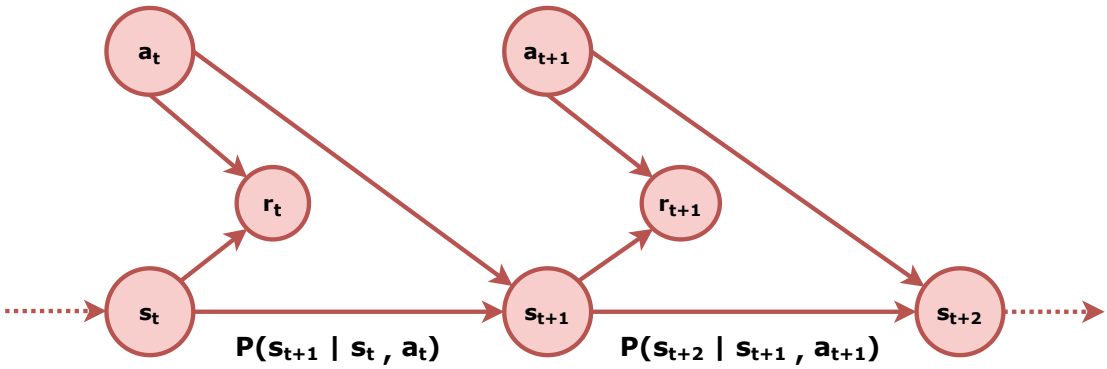


Figura 2.1: Estructura de un Proceso de Decisión de Markov.

2.2. Aprendizaje reforzado

El Aprendizaje Reforzado hace alusión al aprendizaje que adquiere un agente, tras realizar un mapeo del estado actual a las acciones que este puede realizar sobre el ambiente a lo largo del tiempo, para lograr maximizar una señal numérica de recompensa [1]. Es importante notar que el agente no es informado de las acciones que debe tomar, ya que este debe descubrir las acciones que le permiten maximizar las recompensas utilizando el método de ensayo-error, con la finalidad de lograr el objetivo o meta de la tarea que desea resolver.

La representación de la interacción entre el agente y su entorno, se visualiza en la Figura 2.2.

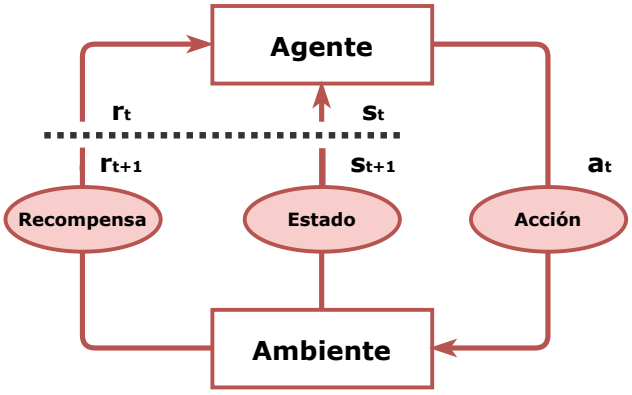


Figura 2.2: Interacciones entre Agente-Ambiente.

Cualquier método que busque resolver problemas de este tipo, se considera como un método de aprendizaje reforzado.

2.2.1. Recompensas, retorno y política

En el Aprendizaje Reforzado, el objetivo del agente es el de maximizar la suma de las recompensas r_t dada $\mathcal{R}(s_t, a_t)$, las cuales se describen mediante la función de retorno finito descrita en la ecuación (2.1).

Estas recompensas son definidas hasta un paso temporal final T, el cual de ser finito, genera un término de la secuencia temporal actual. Lo anterior es llamado episodio. Un episodio posee un estado especial de término y de comienzo, los cuales definen el inicio y fin de un episodio y no necesariamente son distintos de los estados que pueda obtener un agente. Las tareas que poseen episodios, son llamadas tareas episódicas.

$$G_t = \sum_{k=t}^{T} r_k \tag{2.1}$$

En el caso donde no se tienen estados terminales ($T = \infty$) se tiene que la tarea no esta acotada temporalmente. Es por lo anterior, que es relevante el uso del factor de descuento $\gamma \in [0, 1]$, ya que permite definir dentro de un rango de tiempo acotado el problema, considerando que las recompensas también poseen valores acotados, y que la función de retorno esperado no obtendrá un valor infinito, donde en este caso particular $\gamma < 1$. Además, permite definir un horizonte de relevancia de las recompensas, en donde se le asigna mayor importancia a las recompensas más inmediatas cuando $\gamma \to 0$, y en el caso en que $\gamma \to 1$ se le asigna una mayor relevancia a las recompensas futuras. Se introduce así la función de retorno esperado con descuento en la Ecuación (2.2).

$$G_t = \sum_{k=t}^{T} \gamma^k r_{t+k} \tag{2.2}$$

El comportamiento del agente se define a partir de una función llamada política $\pi(a_t|s_t)$. Una política es una función que permite mapear desde un estado, las probabilidades de seleccionar las acciones posibles. A partir de la definición presentada en la Sección 2.1, entregado un estado s_t al agente, este ejecuta una acción a_t dada por la función de política $\pi(a_t|s_t)$, con la que se genera un estado siguiente s_{t+1} en su ambiente y obtiene una recompensa r_t asociada al estado y acción actual.

De la interacción agente-ambiente, se genera una distribución de probabilidad $\mathbb{P}_\pi(\cdot)$ sobre las trayectorias $\tau = \{s_0, a_0, ...s_T, a_T\}$ que sigue el agente, donde el valor de T puede ser acotado o infinito, tal como se menciona en al comienzo de la Sección 2.2. La distribución de probabilidad sobre las trayectorias τ, se define a partir de la distribución sobre el estado inicial $\mathbb{P}(s_0)$, la probabilidad de transición de estados $\mathbb{P}(s_{t+1}|s_t, a_t)$ y la política del agente $\pi(a_t|s_t)$. La expresión es definida en la Ecuación (2.3).

$$\mathbb{P}_\pi(\tau) = \prod_{t=1}^{T} \mathbb{P}(s_1) \cdot \pi(a_t|s_t) \cdot \mathbb{P}(s_{t+1}|s_t, a_t) \tag{2.3}$$

Formalmente, el objetivo del aprendizaje reforzado es el de maximizar la función de retorno esperado evaluada sobre las trayectorias seguidas por el agente al interactuar con el ambiente. Esto se presenta en la Ecuación (2.4).

$$J(\pi) = \mathbb{E}_{\tau \sim \mathbb{P}_\pi(\tau)} \left[\sum_{t=1}^{T} \gamma^{t-1} \mathcal{R}(s_t, a_t) \right] \tag{2.4}$$

2.2.2. Funciones de valor

Para lograr el objetivo descrito anteriormente, de maximizar la función presentada en la Ecuación (2.4), se introducen dos funciones frecuentemente utilizadas para resolver el problema del Aprendizaje Reforzado, denominadas Funciones de Valor. Estas permiten registrar en cada paso de tiempo t a lo largo de la interacción agente-ambiente, el retorno esperado dependiendo de la política del agente. Estas se presentan a continuación:

- **Función de Valor-Estado:** Es el retorno esperado que recibe un agente siguiendo una política π a partir de un estado s, sobre una trayectoria $\tau_t = \{s_t, a_t, ...s_T, a_T\}$ recorrida por el agente a partir del estado s_t. Esta se denota por $V^\pi(s)$, y se define en la Ecuación (2.5).

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

- **Función de Valor-Acción:** Es el retorno esperado que recibe un agente siguiendo una política π a partir de un estado s tras haber ejecutado una acción a, sobre la trayectoria recorrida τ_t, a partir del estado s_t. Esta se denota por $Q^\pi(s)$, y se define en la Ecuación (2.6).

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

Las funciones de valor $V^\pi(s)$ y $Q^\pi(s, a)$, cumplen además la cualidad de tener una relación de recurrencia entre ellas: Ambas funciones pueden ser descritas en función de la otra, en la Figura 2.3 se presenta un diagrama de las relaciones entre funciones con un proceso de decisión de Markov tomando 2 estados para tiempos t y $t + 1$.

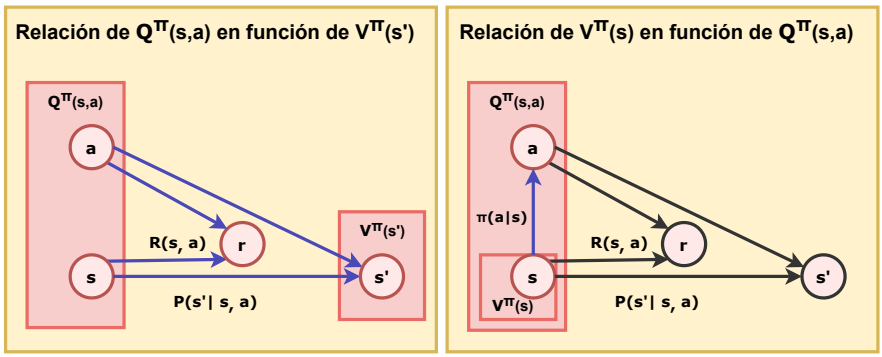


Figura 2.3: Diagrama de relación entre funciones $V^\pi(s)$ y $Q^\pi(s, a)$.

La función $V^\pi(s)$ se escribe en términos de $Q^\pi(s, a)$, debido a que el valor de un estado depende del valor de todas las acciones posibles para este estado, y de cómo cada acción puede ser tomada bajo la política utilizada por el agente $\pi(a|s)$. Esto se describe en la Ecuación (2.7).

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(a|s)} \left[Q^\pi(s, a) \right] \tag{2.7}$$

En el caso de la función $Q^{\pi}(s,a)$, esta se puede escribir en términos de $V^{\pi}(s)$, ya que la primera depende de la recompensa obtenida tras la ejecución de una acción sobre el estado actual, donde el estado siguiente s' sigue la probabilidad de transición descrita por $\mathbb{P}(s'|s,a)$. Lo anterior se presenta en la Ecuación (2.8).

$$Q^{\pi}(s,a) = \mathcal{R}(s,a) + \gamma \mathbb{E}_{s' \sim \mathbb{P}(s'|s,a)} \left[V^{\pi}(s') \right] \tag{2.8}$$

Para cualquier MDP finito se puede definir una política óptima π_* que es mejor o igual al resto de las políticas $\pi_* \geq \pi \ , \forall \pi$. Lo anterior se debe a que las funciones de valor determinan un orden parcial sobre las políticas; esto se traduce en que para una política π mejor o igual a una política π' ($\pi \geq \pi'$) si y sólo si el retorno esperado de π es mayor o igual al de π' ($V^{\pi}(s) \geq V^{\pi'}(s) \ , \forall s \in \mathcal{S}$).

Las políticas óptimas π_* se obtienen a partir de la maximización del retorno esperado $J(\pi)$, de modo que $\pi_* = \underset{\pi}{\operatorname{argmax}}\, J(\pi)$. Puede haber una o más políticas óptimas, todas denotadas por π_*. Estas políticas óptimas comparten la misma función de Valor-Estado, presentada en la Ecuación (2.9). La función $V^*(s)$ se denomina función Valor-Estado óptima.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \ \ , \forall s \in \mathcal{S} \tag{2.9}$$

Las políticas óptimas comparten también la función de Valor-Acción óptima $Q^*(s,a)$ descrita en la Ecuación (2.10).

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \ \ , \forall s \in \mathcal{S} \ \wedge \ \forall a \in \mathcal{A} \tag{2.10}$$

2.2.3. Algoritmos de aprendizaje reforzado

De todas las formas de aprendizaje de máquinas, el aprendizaje reforzado es el más cercano al aprendizaje realizado por humanos y animales, y muchos de estos algoritmos core fueron inspirados por sistemas biológicos [1]. El objetivo de estos algoritmos consiste en encontrar la política óptima π_*, a través de distintos enfoques. La política o comportamiento del agente, tradicionalmente se plantea como una función que mapea desde el espacio de los estados al de las acciones, donde esta puede ser una función determinista o estocástica, que se optimiza a medida de que el agente interactúa con el ambiente y encuentra la maximización de su función de recompensas a partir de este mapeo. En los casos donde el problema se torna más complejo debido a la gran cantidad de estados que puede tener $\mathcal{S}$ y no sea directo el poder utilizar una tabla que relacione estados con acciones por ejemplo, es necesario representar la política como un modelo parametrizado $\pi_\theta(a|s) = f(a|s;\theta)$, donde θ son los parámetros que se desean optimizar para obtener la política óptima. Lo anterior es utilizado en la práctica, de forma tal que el problema a resolver se convierte en encontrar los parámetros θ^* que optimizan la política del agente, lo cual se presenta en la Ecuación (2.11).

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, J(\pi_\theta) \tag{2.11}$$

Existen múltiples métodos para resolver este problema, pero la estructura general de funcionamiento de los algoritmos de aprendizaje reforzado se presenta en la Figura 2.4, la cual consiste en:

- **Generación de muestras:** Se obtienen una cantidad finita de muestras o experiencia, mediante la colección de tuplas (s_t, a_t, r_t, s_{t+1}).
- **Estimación del retorno o ajuste del modelo:** A partir de las muestras recolectadas se estima el retorno esperado, donde se busca maximizarlo para obtener una política óptima.
- **Optimización de la política:** Tras realizar la estimación anterior, se optimiza la política actual mediante la obtención de una nueva política mejorada, por ejemplo, utilizando la expresión $\nabla_\theta J(\pi_\theta)$ para actualizar los parámetros θ de la política. Otra forma de hacerlo es mediante el uso de una función de valor.

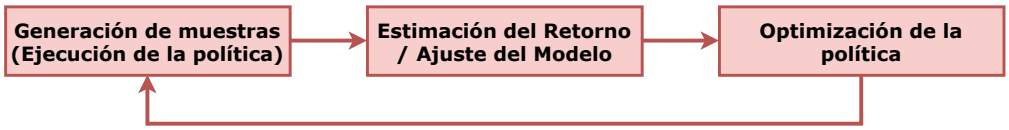


Figura 2.4: Anatomía de los algoritmos de aprendizaje reforzado.

En cuanto a la clasificación de estos algoritmos (Ver Figura 2.6), existen dos grandes ramas en base a las cual se puede dividir el aprendizaje reforzado, la primera es el aprendizaje directo o libre de modelo (*Model-Free*), esto implica que no se posee una representación del ambiente y se busca aprender la política a partir de la interacción del agente con el ambiente mediante el paradigma de prueba-error. La segunda consiste en el aprendizaje mediante el uso de un modelo del ambiente o función de predicción de las transiciones de estados y recompensas (*Model-Based*), lo cual puede ser utilizado para planificación de las acciones futuras del agente, entre otras utilidades como la generación de muestras sintéticas. Lo anterior se puede plasmar en el esquema que se presenta en la Figura 2.5.

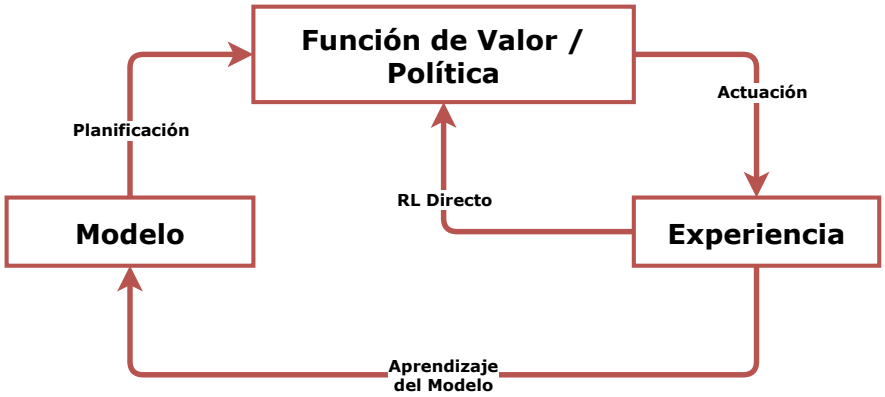


Figura 2.5: Relación entre aprendizaje, planificación y actuación.

Dentro de los tipos de modelos **Model-Free**, existen tres grandes categorías de aprendizaje:

- **Basado en Funciones de Valor:** Esta clase de algoritmos aprende la política de forma implícita, mediante el uso de funciones parametrizadas $V_\theta^\pi(s)$ o $Q_\theta^\pi(s,a)$, para encontrar los parámetros θ que logran la aproximación de las funciones mencionadas a sus valores óptimos correspondientes a cada función de valor $V^*(s)$ o $Q^*(s)$.

- **Aprendizaje de la Política:** Este tipo de método busca aprender de manera directa los parámetros θ de la política $\pi_\theta(a|s)$, mediante la optimización de la función $J(\pi_\theta)$. Se optimiza en dirección de $\nabla_\theta J(\pi_\theta)$, donde se realiza una optimización de gradiente ascendente sobre los parámetros de la política.

- **Actor-Crítico:** En este caso, se utilizan ambos métodos mencionados anteriormente, donde la política representa al actor y la función de valor al crítico. El actor indica la acción a ejecutar y el crítico entrega una estimación de su desempeño.

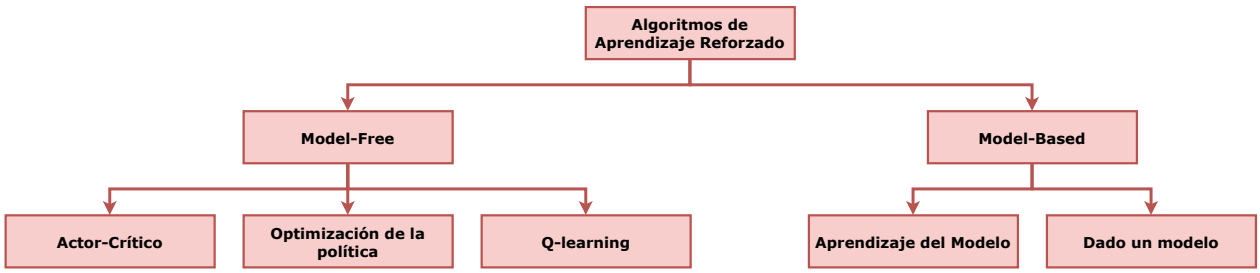


Figura 2.6: Tipos de algoritmos de aprendizaje reforzado.

Una última característica relevante en el funcionamiento de los algoritmos, es si funcionan como algoritmos *On-Policy* u *Off-Policy*. Los algoritmos *On-Policy* se caracterizan por evaluar y actualizar la política π, a partir de experiencia evaluada en la misma política, donde esta experiencia se renueva cada vez que la política es actualizada. En el caso de los algoritmos *Off-Policy*, estos actualizan la política π a partir de experiencia sampleada de interacciones previas a lo largo de todo el entrenamiento que ha tenido el agente con el ambiente, obtenida mediante una distribución arbitraria μ, distinta de la política π en entrenamiento. En general se considera que los algoritmos *Off-Policy* logran optimizar la eficiencia de muestras debido a la reutilización de experiencias previas de otros episodios pasados durante el entrenamiento de la política.

2.3. Aprendizaje reforzado profundo

Si bien el aprendizaje reforzado ha sido bastante popular en los últimos años debido a su éxito en el diseño (del problema y de las características a utilizar) y resolución de problemas de toma de decisión secuencial [2], tales como desarrollar un agente capaz de jugar *Atari* [21], lograr jugar contra equipos de principiantes de *Dota* [22] o incluso derrotar al campeón del juego *AlphaGo* [23]; muchos de estos logros se deben a la integración del aprendizaje profundo o *deep learning* [24], que permite abordar problemas con un grado de complejidad mayor al momento de representar los estados complejos o con información no estructurada como imágenes o señales de sonido, y las acciones del sistema mediante el uso de redes neuronales profundas.

La combinación de ambos, es llamada aprendizaje reforzado profundo, el cual resulta ser de gran utilidad en problemas que poseen un espacio de estados de alta dimensionalidad. El gran éxito del aprendizaje reforzado profundo, se debe a que se puede utilizar la información o datos obtenida/os por el agente en su interacción con el ambiente, y ser procesada para obtener características de alto, medio y bajo nivel de la información sin procesamiento, lo cual permite aprender distintos niveles de abstracción de los datos sin la necesidad de un experto para diseñar las características de manera manual, pues estas son aprendidas por la red neuronal empleada [2].

2.3.1. Funciones de aproximación y redes neuronales profundas

La aproximación de función es una familia de técnicas matemáticas y estadísticas utilizadas para representar una función de interés cuando esta es difícil de representar de manera exacta o explícita.

En particular, la aproximación de función es utilizado para representar políticas, funciones de valor, y modelos de avance o *forward models* de forma paramétrica o no. Las redes neuronales profundas, modelos lineales, entre otros son utilizados como funciones de aproximación para representar las distintas funciones de interés en los distintas tareas de aprendizaje reforzado [25]. Para efectos de esta memoria, se desea enfocar la atención en redes neuronales profundas *feed forward*.

Redes neuronales artificiales (ANN)

Las redes neuronales artificiales son modelos matemáticos utilizados para modelar tareas de distintos tipos, como de predicción, clasificación, etc; con el fin de relacionar los valores de salida con los datos de entrada del sistema, mediante el ajuste de los parámetros del modelo. A continuación se describen los modelos neuronales a utilizar, incluyendo un diseño de red particular a utilizar.

1. **Redes neuronales prealimentadas o *feedforward*:** Son una serie de modelos de regresión logística apiladas de manera que la salida de cada modelo es la entrada del otro hasta la capa final, que posee un modelo de regresión logística o lineal, dependiendo de si el problema es de clasificación o regresión [26]. Estas se definen mediante las ecuaciones

presentadas a continuación:

$$p(y|x,\theta) = N(y|w^T z(x), \sigma^2) \tag{2.12}$$

$$z(x) = g(V \cdot x) = [g(v_1^T \cdot x), ..., g(v_H^T \cdot x)] \tag{2.13}$$

Donde $g(\cdot)$ es una función de activación no lineal (2.13), o función de transferencia (como la función logística), $z(x)$ es una capa oculta, con H neuronas o unidades ocultas, V es la matriz de pesos que une las entradas con los nodos ocultos, y w es el vector de pesos que une la capa oculta con la salida de la red. Es importante la no linealidad de la función $g(\cdot)$, ya que en caso contrario, colapsa en un gran modelo de regresión lineal [26].

En la Figura 2.7, se presenta una visualización de una red feed forward, donde los círculos simbolizan las neuronas del sistema y las flechas las conexiones entre estas.

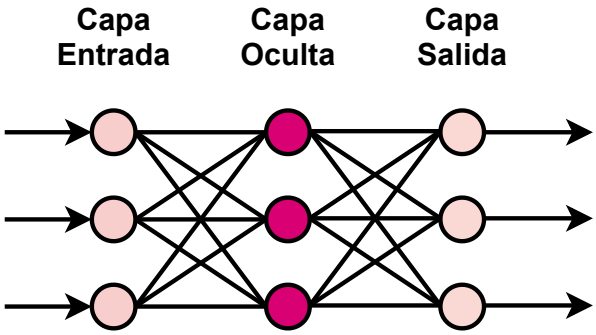


Figura 2.7: Diagrama de red neuronal *feed forward*.

2.3.2. DQN: Deep Q-Networks

El algoritmo DQN, se introduce por primera vez en el trabajo *"Playing Atari with Deep Reinforcement Learning"* [21, 27], donde se utilizan redes neuronales profundas como aproximadores de funciones de valor-acción, con el fin de aprender políticas a partir de observaciones de alta dimensionalidad, utilizando aprendizaje reforzado. En el primer trabajo en particular, se utilizan los *frames* de imágenes de una serie de 7 videojuegos de **Atari 2600**, donde se obtienen resultados satisfactorios en el área de estudio del aprendizaje reforzado, y logra superar el conocimiento experto humano en algunos de estos juegos.

Al ser combinado con el método *Experience Replay* [28], se almacenan muestras para ser utilizadas en el aprendizaje de tipo *off-policy*, de forma que se obtienen muestras sampleadas en *minibatches* de forma aleatoria para ser utilizados en el aprendizaje de la política del agente. Esto logra evitar la correlación entre las observaciones que percibe el agente durante su aprendizaje.

DQN logra introducir una metodología para estabilizar, controlar y modelar políticas construidas en base al uso de redes neuronales profundas, donde estos últimas son modelos que pueden variar en diseño y complejidad [29], además del contexto donde estas pueden ser utilizadas, tales como en el procesamiento de imágenes, audio, señales, entre otros. Lo anterior, vuelve este trabajo uno de los más importantes como la base del aprendizaje reforzado

profundo que hoy se conoce.

La metodología introducida se presenta en Algoritmo 1, que al utilizar redes neuronales como aproximadores de la función de valor-acción, y permite construir un modelo de la política que se adapta a una entrada con datos no estructurados, como lo son los píxeles en este caso. Además, se utiliza una red neuronal objetivo o *target Q-Network* [27], la cual es una copia de la red que aproxima a la función de valor-acción, y que actualiza cada C iteraciones sus parámetros, haciendo copia de los parámetros del aproximador de función de la función valor-acción. Lo expuesto, permite que se reduzca la posibilidad de oscilación o divergencia de los parámetros θ, ante algún retraso en la actualización de la red $Q_\theta(s, a)$. Esto permite calcular la variable y_j sin afectar la optimización de la función de valor-acción.

La limitante que posee DQN, se relaciona con el espacio de acciones de control utilizado, ya que estas son de naturaleza discreta, lo cual acota las aplicaciones de este algoritmo en tareas de control que necesiten un espacio de acción continuo.

Algoritmo 1: Deep Q-Network (DQN)

Inicializar *replay buffer* D.
Inicializar función de valor $Q_\theta(s, a)$ de parámetros θ
Inicializar función *target* $Q_{\bar{\theta}}(s, a)$ de parámetros $\bar{\theta} \leftarrow \theta$
for *episodio = 1, .., M* **do**
  Se obtiene s_1
  for t *= 1, .., T* **do**
    Con probabilidad ε, se elige a_t aleatoriamente, si no $a_t = \arg\max_{a \in A} Q_\theta(s_t, a)$
    Ejecutar acción a_t en el ambiente, observar r_t, s_{t+1}.
    Guardar transición $(s_t, a_t, r_t, s_{t+1},)$ en D.
    Samplear un minibatch aleatorio de transiciones (s_j, a_j, r_j, s_{t+1}) de D.

$$\text{Calcular } y_j = \begin{cases} r_j & \text{si } s_{j+1} \text{ es un estado terminal} \\ r_j + \gamma \cdot \max_{a'} Q_{\bar{\theta}}(s_{j+1}, a') & \text{caso contrario} \end{cases}$$

    Actualizar $Q_\theta(s, a)$ minimizando $L(\theta) = \frac{1}{N} \sum_{j=1}^{N} (y_j - Q_\theta(s_j, a_j))^2$
    Actualizar la red *target* $Q_{\bar{\theta}}(s, a)$ cada C iteraciones: $\bar{\theta} \leftarrow \theta$
  end
end

2.3.3. DDPG: Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG)[30] es un algoritmo *model free* de tipo actor-crítico (off-policy), basado en *Deterministic Policy Gradient* (DPG)[31] para operar sobre espacios de acción continuos.

Este algoritmo al modelar la política del agente, mediante una red neuronal profunda, logra ampliar las aplicaciones de DQN, en contextos donde se necesite un espacio de acciones de tipo continuo.

DDPG utiliza los descubrimientos de DQN [21], mediante el uso de aproximadores de funciones neuronales para las funciones actor-crítico, donde sus parámetros son capaces de ajustarse de forma estable y robusta, ya que:

1. Hacen uso de muestras almacenadas en un *Replay Buffer*, durante el entrenamiento *off-line*, con el fin de reducir la correlación entre muestras. Esto debido a que se pueden samplear de manera aleatoria muestras de este *Replay Buffer*. Este método se conoce como *Experience Replay* [28].

2. Se entrenan a partir de una red neuronal objetivo o *target Q-Network* [27], con el fin de entregar *targets* consistentes.

Se presenta el algoritmo de aprendizaje a continuación:

Algoritmo 2: Deep deterministic policy gradient (DDPG)

Inicializar las redes de actor $\mu_\phi(s)$ y crítico $Q_\theta(s,a)$ con pesos ϕ y θ.
Inicializar las redes *target*, actor $\mu_{\bar{\phi}}(s)$ y crítico $Q_{\bar{\theta}}(s,a)$ con pesos $\bar{\phi}$ y $\bar{\theta}$.
Inicializar *Replay Buffer R*.
for *Episodio = 1,2,..,M* **do**
  Inicializar un proceso aleatorio V para exploración.
  Se obtiene observación inicial s_1.
  for *t= 1,2,..,T* **do**
    Seleccionar acción $a_t = \mu_\phi(s) + V_t$.
    Ejecutar acción a_t sobre el ambiente.
    Observar la recompensa r_t y el nuevo estado s_{t+1}.
    Almacenar la transición (a_t, s_t, r_t, s_{t+1}) en R.
    Samplear un minibatch de muestras aleatorias de tamaño N (a_i, s_i, r_i, s_{i+1}).
    Se define la etiqueta $y_i = r_i + \gamma Q_{\bar{\theta}}(s_{i+1}, \mu_{\bar{\phi}}(s_{i+1}))$
    Actualizar la red Q minimizando la función $L(\theta) = \frac{1}{N}\sum_{i=0}^{N}\left(y_i - Q_\theta(s_i, a_i)\right)^2$
    Actualizar la red μ maximizando la función $J(\phi) = \frac{1}{N}\sum_{i}^{N} Q_\theta(s_i, \mu_\phi(s_i))$
    Actualizar las redes *target*: $\bar{\theta} \leftarrow \lambda\theta + (1-\lambda)\bar{\theta}$ y $\bar{\phi} \leftarrow \lambda\phi + (1-\lambda)\bar{\phi}$
  end
end

2.4. Navegación robótica autónoma

Los robots móviles autónomos son robots que pueden realizar las tareas deseadas, como lo son la navegación y exploración autónomas, en entornos estructurados o no estructurados sin una guía humana continua [16]. Los grandes problemas que enfrenta la navegación, se pueden representar según las siguientes etapas [16]:

- **Percepción del mundo:** Percibir el mundo mediante sensores para obtener características del mundo como imágenes, mediciones láser, mediciones de ondas sonoras, entre otras.
- **Planificación del camino:** Utilizar las características para crear una secuencia ordenada de "puntos" objetivos que el robot debe alcanzar.
- **Generación de camino:** Generar un camino a través de la secuencia de puntos objetivos obtenida.
- **Seguimiento del camino:** Controlar que el robot móvil pueda seguir la ruta prevista, mediante algún diseño de control.

Dentro de la navegación, existen tres categorías globales a partir de las cuales se pueden definir distintos tipos de navegación robótica [16]:

- **Navegación basada en mapas (*Map-Based*):** Este tipo de sistemas dependen de modelos geométricos o mapas topológicos del ambiente. Los robots poseen *a priori* la información del ambiente, tal que su sistema de navegación integra la información del mapa del ambiente entregado.
- **Navegación basada en mapas construidos (*Map-Building-Based*):** Sistemas que utilizan sensores para construir su propio modelo geométrico o topológico del ambiente y utilizar estos para navegar. Este tipo de navegación se diferencia del primero, debido a que este debe pasar por una etapa de exploración, tal que pueda obtener una representación interna del espacio mediante sus sensores, para luego poder realizar la tarea de navegación.
- **Navegación sin mapa (*Mapless*):** Estos son sistemas que no utilizan representaciones explícitas sobre el espacio donde se llevará a cabo la navegación, si no que recurre al reconocimiento de objetos encontrados en el ambiente o el seguimiento de estos mediante la generación de movimientos basados en observaciones obtenidas a partir de sus sensores.

En este trabajo, se desean abarcar sistemas de navegación sin presencia de un mapa, debido a que en términos prácticos, no siempre se posee un mapa del lugar donde se desea navegar [32].

Dado lo anterior, se enfoca la atención sobre la tarea de planificación local, que permite al robot tener una percepción local de su ambiente y mejorar las decisiones que se deseen tomar al momento de realizar una tarea, más en específico de navegación, a medida de que este se encuentra en interacción con el ambiente.

2.4.1. Navegación robótica basada en aprendizaje reforzado profundo

En la actualidad se han realizado múltiples trabajos relacionados con la navegación robótica incorporando el uso del aprendizaje reforzado profundo [15, 32, 33, 34], con el interés de encontrar la política óptima a través de la guía del robot a su posición objetivo mediante su interacción con el ambiente. Esta tarea sigue siendo desafiante al momento de modelar la política o comportamiento del robot, para lograr alcanzar el objetivo de navegación, debido a que aún no se tiene un modelamiento o parametrización estandarizado.

Una de las ventajas de incorporar el aprendizaje reforzado profundo, es que facilita la navegación sin el uso de mapas y reduce la dependencia en la precisión de las mediciones de los sensores [33], gracias al uso de modelos neuronales que permiten relacionar estas mediciones con las acciones deseadas.

Muchos algoritmos de DRL como DQN, DDPG, PPO [35] entre otros, han sido utilizados dentro del contexto de navegación robótica basada en DRL. Estos métodos permiten describir el proceso de navegación mediante un proceso de decisión de Markov, que utiliza la información observada por el robot, mediante sus sensores, con la meta de máximizar el retorno esperado [33].

En el trabajo realizado en [15], se realiza un estudio exhaustivo de distintos tipos de configuraciones tanto en el vector de observaciones como en las parametrizaciones utilizadas a partir del uso de sensores de LiDAR. Esto se realiza con el fin de encontrar una aproximación robusta para el diseño de planificadores locales basados en aprendizaje reforzado. A partir de este trabajo, se encuentran múltiples herramientas e ideaciones útiles para incorporar en este trabajo, dentro de las cuales se buscan utilizar las parametrizaciones, ambientes de entrenamiento y la función de recompensa, adaptándolas en el contexto de esta memoria.

2.5. Aprendizaje de máquinas interactivo

Para efectos de esta memoria, el aprendizaje de máquinas interactivo o *interactive machine learning (IML)*, se conoce como un área de investigación que reúne una serie de técnicas en las que los humanos se hacen parte del aprendizaje de los agentes, mediante señales de aprendizaje que se pueden conceptualizar como instrucciones, demostraciones o retroalimentación, entre otras [5]. Se tiene un diagrama de proceso genérico de aprendizaje reforzado con guía humana, adaptable para las distintas formas en las que este puede ser integrado dentro de un agente artificial, para comprender el funcionamiento general de esta área de estudio. Este diagrama de procesos se presenta en la Figura 2.8.

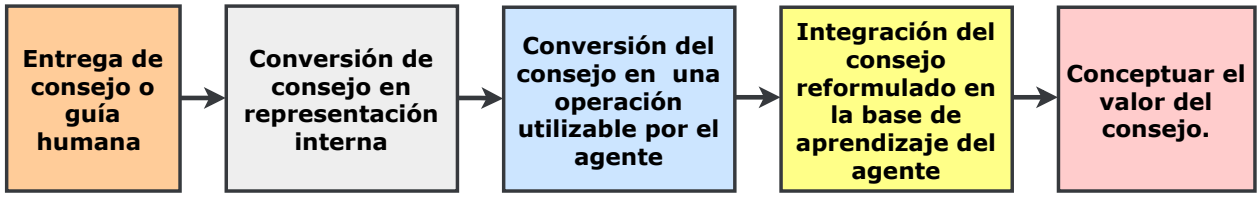


Figura 2.8: Proceso del aprendizaje reforzado con guía humana genérico [5].

A partir de lo anterior, se describe el proceso de aprendizaje reforzado con guía humana de forma genérica, de manera que se comienza por realizar una descripción de cómo la guía humana es proveída al sistema del agente, lo cuál se puede diseñar de múltiples formas. Luego, se continua con la codificación de la información percibida del consejo para ser transformada en una representación interna del agente. Finalmente se busca integrar esta señal de consejo dentro del algoritmo de aprendizaje del agente, de forma de que pueda ser utilizada como una función interna que sea entendida por este, para influir dentro de las acciones y estados futuros.

Existen muchas formas de materializar la guía humana dentro del aprendizaje del agente [5, 36], estas en general se pueden clasifican en 5 grandes categorías [5]:

- Limitaciones Generales
- Instrucciones Generales
- Guía Humana
- Retroalimentación Correctiva
- Retroalimentación Evaluativa

En particular se desea profundizar dentro del marco de trabajo que involucra la Retroalimentación Correctiva.

2.5.1. Retroalimentación correctiva humana

El esquema de interacción de retroalimentación correctiva que se tiene (Figura 2.9), consiste en que el humano a partir de la obtención del par acción-estado, es capaz de indicarle al agente mediante una señal H_t de retroalimentación, cómo mejorar el rendimiento de la acción del agente para dicho estado.

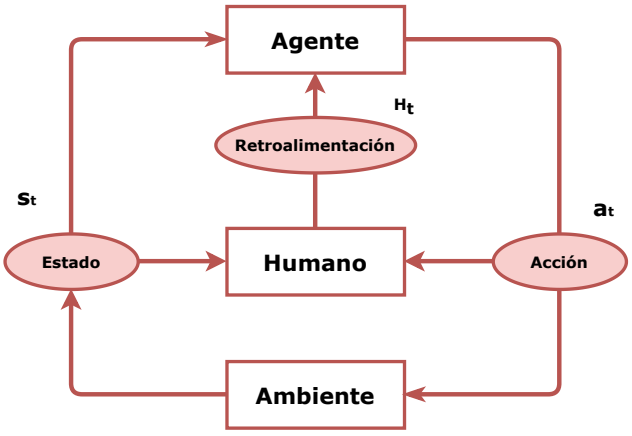


Figura 2.9: Esquema de aprendizaje reforzado utilizando retroalimentación o *feedback*.

Existen múltiples razones por las que es de interés de estudio el incorporar la retroalimentación humana en el aprendizaje de máquinas, comenzando por la facilidad y variedad

en la forma en la que esta puede ser incorporada. La razón por la cuál se desea estudiar en particular, se encuentra asociada a la eficiencia de muestras, por lo que se desea encontrar la forma de caracterizar esta señal de tal forma de que el agente logre un buen desempeño con la menor cantidad de ejemplos posible. Si bien ha sido un área de estudio con muchos exponentes, sigue siendo de interés el cómo integrar esta señal de forma de que el agente obtenga una buena interpretación de esta [36].

En este trabajo, el enfoque expuesto en la señal de retroalimentación correctiva se plantea en COACH [8, 9, 10], la cuál implica implícitamente que la acción realizada es incorrecta, donde más allá de la crítica a la acción, se entrega al agente la información sobre la acción correcta (información como: se debe ir más despacio, falta moverse más a la derecha, etc), lo cual ha sido visto como una buena forma de incorporación de retroalimentación, debido a que requiere menor exploración que la retroalimentación evaluativa (retroalimentación que se entrega una vez que el agente ya realizó una acción) y se le permite saber al agente sobre las acciones que este realiza, sin esperar a que este escoja la mejor [5]. Se estudia en especifico, la variación de este algoritmo, que contempla el uso de redes neuronales profundas. Este algoritmo se denomina D-COACH.

D-COACH: Deep COrrective Advice Communicated by Humans

En COACH [8], la retroalimentación correctiva h_t sobre las acciones a_t del agente son descritas a partir de tres valores: $h_t := \{-1, 0, 1\}$, donde el valor negativo indica si la acción realizada debe ser disminuida en magnitud (si se debe reducir la velocidad, giro del agente, etc.), el valor 1 indica si esta debe ser incrementada y el valor 0 indica que el agente no necesita corrección de la acción ejecutada. Esta señal en conjunto con una magnitud de error e permiten generar una señal de error, que permite actualizar los parámetros de la política $\pi_\theta(\cdot)$ del agente.

$$\text{error} = h_t \cdot e \tag{2.14}$$

Este *framework* utiliza dos funciones de aproximación, una asociada a la política del agente $\pi_\theta(\cdot)$ y otra al modelo de retroalimentación humana $H_\psi(\cdot)$, donde cabe notar que no se utilizan funciones de valor que utilizan recompensas/costos para modelar el proceso de aprendizaje.

COACH introduce como funciones de aproximación, modelos lineales de funciones bases. Con lo anterior se construye la política del agente $\pi_\theta(s) = \phi(s) \cdot \theta$ y el modelo de predicción de retroalimentación humana $\pi_\psi(s) = \phi(s) \cdot \psi$ que busca modelar la magnitud del error del agente, para actualizar los parámetros θ de la política $\pi_\theta(\cdot)$. Ambos modelos, la política del agente $\pi_\theta(\cdot)$ y el modelo de retroalimentación humano $H_\psi(\cdot)$, actualizan sus parámetros a partir del uso de SGD una vez recibida la señal de retroalimentación h_t.

Lo anterior, explica la estructura básica de la retroalimentación correctiva utilizada en COACH [8]. Sobre este trabajo, se han desarrollado otros que incorporan el uso de redes neuronales profundas como D-COACH [9] o combinar el uso de la retroalimentación correctiva con algoritmos de aprendizaje reforzado clásico [10]. En el Algoritmo 3, se presenta la estructura de D-COACH [9], que aprovecha el uso de las redes neuronales profundas como aproximadores de función de la política del agente.

Algoritmo 3: D-COACH

Inicializar magnitud de error e.
Inicializar parámetros máximos y mínimos (m y M) para el *replay buffer* .
Inicializar intervalo b de actualización del *replay buffer*.
Inicializar *replay buffer* $B = []$.
for t = *1,2,..,N* **do**
  Observar el estado s_t
  Ejecutar $a_t = \pi(s_t)$
  Obtener retroalimentación humana h_t
  if $h_t! = 0$ **then**
    Se define el error $error_t = h_t \cdot e$
    Se define la etiqueta $y_{label} = a_t + error_t$
    Almacenar la transición (a_t, s_t) en B.
    Actualizar red $\pi(\cdot)$, utilizando SGD con par (s_t, y_{label})
    if *largo(B)* $\geq m$ **then**
      Actualizar red $\pi(\cdot)$, utilizando SGD con mini-batch sampleado del *buffer* B.
    end
    if *largo(B)*$>M$ **then**
      $B = B[2 : M + 1]$
    end
  end
  if $mod(t, b) = 0$ *y largo(B)* $\geq m$ **then**
    Actualizar red $\pi(\cdot)$, utilizando SGD con mini-batch sampleado del *buffer* B.
  end
end

Capítulo 3

Formalización del problema

3.1. Descripción general

Como ha sido mencionado anteriormente, el problema a resolver consiste en lograr que un agente móvil representado por un robot *Clearpath Husky A200* [17] (Figura 3.1), logre llevar a cabo la tarea de planificación local en 2D. En este trabajo se busca a través de la combinación de un algoritmo de aprendizaje profundo con un de aprendizaje de máquinas interactivo, más en específico de los algoritmos DDPG y D-COACH.

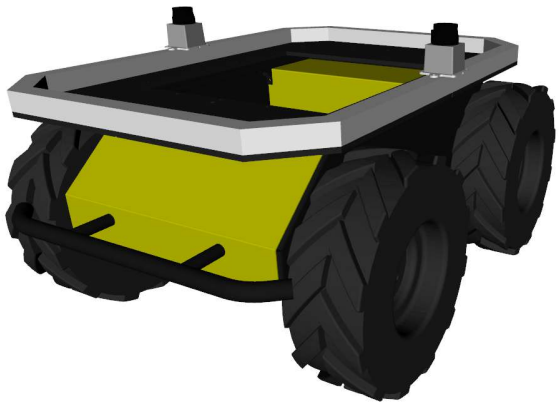


Figura 3.1: Simulación del robot **Husky** en la plataforma **Gazebo**.

Se busca parametrizar la política del agente, utilizando un modelo neuronal diseñado específicamente para el contexto de navegación local bidimensional, utilizando una nube de puntos y odometría del robot, según a lo propuesto en [15]. La idea es lograr obtener una política óptima o mediante el entrenamiento de esta utilizando los algoritmos descritos, junto con las observaciones capturadas del ambiente de navegación, a partir de los sensores incorporados en el robot.

En este contexto, se incorporarán parametrizaciones, funciones de recompensas y ambientes de simulación del trabajo realizado en [15], para modelar el sistema de navegación a utilizar. Se utilizará el *framework* de D-COACH [9, 10], para implementar un algoritmo de tipo híbrido que logre incorporar los beneficios del aprendizaje reforzado profundo y el

aprendizaje de máquinas interactivo, con el fin de mejorar la eficiencia de muestras en el entrenamiento de la política del agente, y comparar esto con los métodos de manera individual, además de compararlo con un método que utiliza a D-COACH y DDPG de manera secuencial.

Para modelar la señal de retroalimentación correctiva humana h_t, se utilizará el plan global del agente, que se computa a partir de la librería *global_planner* [37], con el fin de automatizar y simplificar la obtención de esta señal. El plan global utilizado, brindará como información las poses óptimas a las cuales debería llegar el robot, con el fin de llegar a la posición objetivo.

Para lograr que el agente logre su objetivo, se realizará la acción de control sobre la velocidad lineal y angular del robot, se utilizarán dos sensores láser *Hokuyo UTM-30LX-EW* [18] y también se usarán sensores de colisiones, colocados estratégicamente en el robot. Todo lo anterior esta pensado para ser simulado en la plataforma de simulación de **Gazebo**.

En las subsecciones siguientes, se presentarán el espacio de acciones, observaciones, uso de la retroalimentación humana correctiva, el algoritmo híbrido propuesto a usar, parametrización de la política y función de valor, el ambiente de entrenamiento, la señal de retroalimentación humana y la función de recompensa, que permiten modelar el problema a tratar.

3.2. Espacio de observaciones

Las observaciones a utilizar se dividen en tres categorías:

- **Odometría ($O_{\mathbf{odom}}$):** Estimaciones de velocidad lineal y angular del agente a partir del uso de los *encoders* del robot.

$$O_{\text{odom}} = (\hat{v}^x, \hat{v}^\theta) \tag{3.1}$$

- **Posición Objetivo ($O_{\mathbf{objetivo}}$):** Posición relativa del punto objetivo con respecto al eje principal del robot.

$$O_{\text{objetivo}} = (x^{\text{objetivo}},\ y^{\text{objetivo}}) \tag{3.2}$$

- **Mediciones de Rango ($O_{\mathbf{pcl}}$):** Nube de puntos 2D de tamaño variable [15], capturadas a partir de los sensores LiDAR que se encuentran dispuestos en los costados laterales, sobre la plataforma del robot. Estos sensores entregan en coordenadas polares una distancia ρ_i y un ángulo θ_i por cada punto i, y se genera el vector de puntos $(x_1, x_2, .., x_k)$, a partir de la Ecuación (3.3). Para construir la nube de puntos, se utiliza la librería *ira_laser_tools* [38], que permite integrar la información de las mediciones de rango de ambos sensores.

$$O_{pcl} = \begin{cases} \{(\rho_i \cos(\theta_i)), (\rho_i \sin(\theta_i))\}_{i=1}^{k \leq n} & \text{si} \quad \text{k} \geq 0 \\ 0 & \text{si} \quad \text{k} = 0 \end{cases} \tag{3.3}$$

En la Figura 3.2, se presenta el diagrama de acciones $(v_t^x\ , v_t^\theta)$ en colores fucsia y verde

respectivamente, junto con las observaciones del robot: Posición objetivo O_{objetivo} en color rojo y cyan, las mediciones de rango de los sensores láser O_{pcl} en color gris y la observación de odometría del robot O_{odom} es simplemente una estimación de las acciones del robot, por lo que siguen la misma referencia que estas.

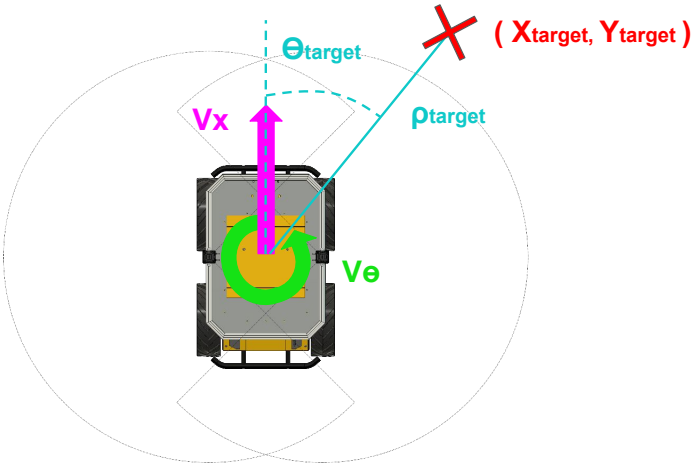


Figura 3.2: Diagrama de acciones de velocidad en la base móvil, posición del objetivo de navegación con respecto al origen de la base del robot y campo de visión de sensores Láser.

3.3. Retroalimentación humana correctiva

En la sección 2.5.1, se habla del aprendizaje correctivo y se explica de manera general el principio de funcionamiento y modelamiento del algoritmo D-COACH, el cual utiliza la señal de retroalimentación o *feedback* humano correctivo, entregado por un profesor humano o artificial. Para efectos de esta memoria se genera una señal humana artificial, mediante la corrección de las acciones que realizará el agente, incorporando una señal de error que será generada a partir de la posición $(\rho_{t+1}, \theta_{t+1})$ próxima del agente generada por su política, la cual se comparará con la posición entregada por el planificador global (o *global planner*) del robot, el cual entrega la posición más cercana ρ^{gp}_{t+1}, θ^{gp}_{t+1} a la cual debería llegar en un instante de tiempo siguiente $t+1$. A partir de la diferencia de estas posiciones, definidas en (3.4) y (3.5), se generará un vector de error que será empleado para crear la señal correctiva humana h_t.

$$e^t_\rho = \rho^{\text{gp}}_{\text{t+1}} - \rho_{t+1} \tag{3.4}$$
$$e^t_\theta = \theta^{\text{gp}}_{\text{t+1}} - \theta_{t+1} \tag{3.5}$$

Una vez obtenida esta señal de error, se genera el vector $h_t = (h^t_\rho, h^t_\theta)$, el cual indica si la acción debe ser reducida (signo negativo) o aumentar en magnitud (signo positivo), a partir del cumplimiento de un vector umbral mínimo $e_{umbral} = (e^{umbral}_\rho, e^{umbral}_\theta)$ para cada

componente de error $e_t = (e_\rho^t,\ e_\theta^t)$, tal que si e_t supera el umbral se toma el signo del error, en caso contrario se toma como valor 0. Lo anterior se muestra en (3.6).

$$h_i^t = \begin{cases} \text{sign}(e_i^t) & \text{si} \quad e_i^t \geq e_i^{umbral} \\ 0 & \text{si} \quad e_i^t < e_i^{umbral} \end{cases} \tag{3.6}$$

Donde i $\in\ \{\rho, \theta\}$.

Finalmente, se incorpora un vector de valores estáticos $e_{\text{fixed}} = (e_\rho,\ e_\theta)$, con el fin de incorporar una magnitud de error fija para modelar la señal de error a utilizar para generar las acciones con retroalimentación correctiva, de manera discreta. Esto se muestra en (3.7).

$$\text{error} = \left(h_t^\rho \cdot e_\rho\ ,\quad h_t^\theta \cdot e_\theta\right) \tag{3.7}$$

3.4. Espacio de acciones

Se tendrá control sobre la velocidad lineal y angular del agente, considerando que el agente tiene capacidad de observación de 360°, donde las velocidades se encuentran dentro del intervalo $[v_{min},\ v_{max}]$, que se compone de las velocidades mínimas y máximas alcanzadas por el robot en $[rad/s]$ de forma angular y $[m/s]$ en lineal, notando que este no tendrá restricciones con tomar velocidades negativas, es decir que podrá retroceder y girar en ambos sentidos. Las acciones a generar a partir de la política se presentan en (3.8).

$$a_t = \left(v_t^x\ ,\quad v_t^\theta\right) \tag{3.8}$$

3.5. Parametrización de la política

Para llevar a cabo el entrenamiento de la política, se utilizan un algoritmo de aprendizaje reforzado *off-policy* de tipo actor-crítico (DDPG), de aprendizaje de máquinas interactivo (D-COACH), y uno que fusione ambos paradigmas de aprendizaje (modelo híbrido: D-COACH y DDPG). Se usan así, dos arquitecturas de redes neuronales profundas, para parametrizar tanto la política como la función de valor, donde la última no es empleada para el algoritmo D-COACH.

Se presenta en la Figura 3.3, la arquitectura **multi-modal PCL** propuesta, que se obtiene del trabajo realizado en [15]. Esta permite integrar los distintos tipos de observaciones obtenidos por el agente, e involucrarlos dentro del proceso de aprendizaje de la política del agente. Esta posee una pequeña adaptación, donde se cambia la función de activación de la salida de la velocidad lineal, de una sigmoide a una tangente hiperbólica, debido al intervalo de operación a usar por esta misma.

Para el caso del actor o la política, este posee como entradas de la red las observaciones y como salida las acciones, que son en este caso la velocidad lineal y angular del robot. Estas últimas se presentan en color amarillo.

Dentro de la Figura, la expresión `Fc(n)`, hace referencia a una capa de una red neuronal *feedforward fully-connected*, que posee `n` neuronas. Las funciones de activación `LReLU`, `Tanh` y `Linear`, son las funciones *Leaky ReLU*, tangente hiperbólica y lineal respectivamente.

En el caso de la función de valor $Q(s, a)$, se reciben como entradas de la red las observaciones y acciones, donde estas últimas se representan en línea punteada celeste. Esta red, a diferencia de la anterior, posee sólo una salida asociada a la estimación de la función de valor misma, también referenciada en celeste.

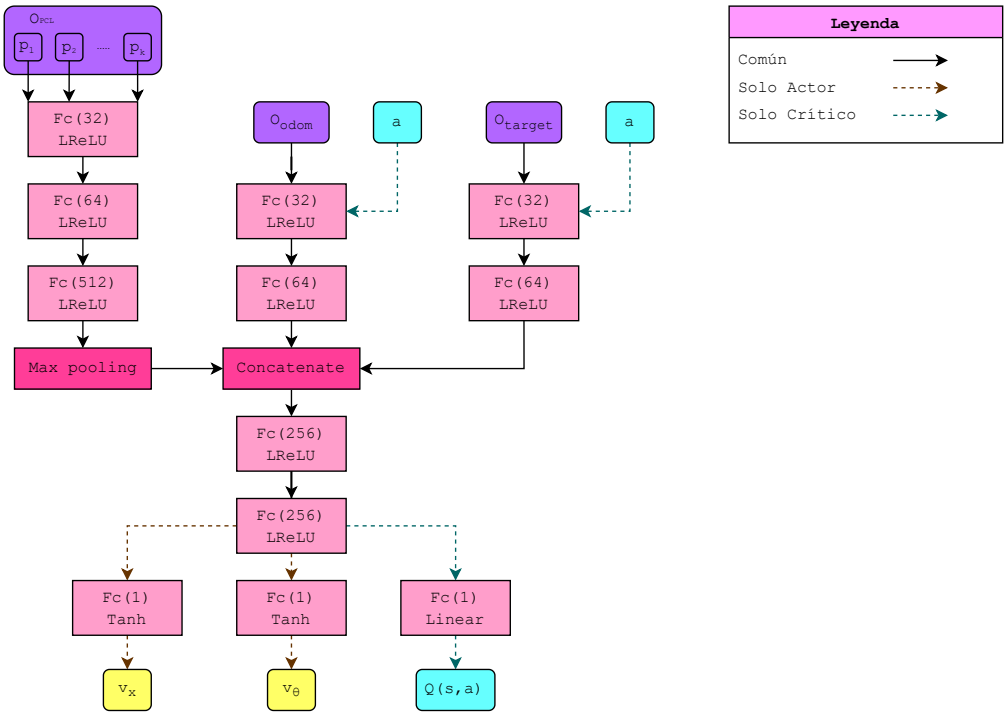


Figura 3.3: Diagrama de red neuronal empleada para parametrizar la política y función de valor. Esta arquitectura se extrae del trabajo realizado en [15].

3.6. Algoritmo híbrido propuesto: D-COACH y DDPG.

En esta sección se introduce un algoritmo de carácter híbrido. El fin de juntar un algoritmo de aprendizaje reforzado (DDPG) con uno de aprendizaje de máquinas interactivo (D-COACH), es de combinar las mejores características de estos algoritmos en la resolución de alguna tarea.

D-COACH, logra agilizar el aprendizaje de la política mediante el uso de la señal de retroalimentación correctiva, la cual indica como mejorar cada acción emitida por el agente en cada instante de tiempo, donde si bien se logra encontrar una buena solución a la tarea, esta no necesariamente es una política óptima. DDPG por su lado, permite estabilizar y refinar el comportamiento de la política del agente, pero con la desventaja de tardar una gran cantidad de iteraciones para lograr esto.

Combinados, se espera lograr que el agente pueda obtener una política óptima en una cantidad de iteraciones de entrenamiento reducida. Cabe destacar que a diferencia de DDPG, el algoritmo D-COACH al ser un algoritmo de aprendizaje interactivo no utiliza una función de recompensa para el entrenamiento de su política.

Este algoritmo utiliza cuatro redes neuronales profundas, para modelar el aproximador de función de la política $\mu_\phi(s)$ y la función de valor $Q_\theta(s, a)$, junto con la copia de cada una de estas para construir las redes *target*. Estos modelos se definen en la sección 3.5, donde el modelo de la política $\mu_\phi(s)$, se actualiza tanto por el paradigma de aprendizaje de D-COACH como el de DDPG. Estas redes se inicializan con pesos distribuidos de manera aleatoria.

Se hace uso de dos *replay buffers*, el primero es utilizado por D-COACH, es un *replay buffer* (B_{COACH}) que toma ventanas móviles de M muestras, que luego son sampleadas de forma aleatoria en *batches* de tamaño b, con los cuales se entrena el modelo neuronal de la política. Luego el segundo *replay buffer* (B_{DDPG}), utilizado por el tipo de actualización utilizada en DDPG, es un *replay buffer* que posee tamaño M_d, donde este tamaño es lo suficientemente grande como para almacenar información de forma histórica, considerando muestras desde el primer instante de entrenamiento, hasta el final. Este también samplea muestras en batches de tamaño b_d, de manera aleatoria, con el fin de actualizar la política del agente.

Para realizar la actualización de los parámetros de la política y función de valor, se utilizan 3 tipos de enfoques:

- ***Feedback*:** Este tipo de actualización se realiza sobre los parámetros de la política del agente, y ocurre cada vez que el agente se desvíe del plan global a partir de un error umbral, asociado a cada una de las acciones del agente. En este caso, para la velocidad lineal y angular. Una vez detectado este error, se emplea la señal $error_t$ descrita en la sección 3.3, sobre la cual se define la etiqueta con la acción corregida $y_t = a_t + error_t$, y se utiliza como valor target para la función de perdidas utilizada por D-COACH, que busca asemejar lo más posible las acciones emitidas por la política a las acciones corregidas y_t.

  Esta actualización de parámetros, se realiza en dos oportunidades dentro de este enfoque: Una en cada paso de tiempo con la muestra actual, y otra con el *replay buffer* B_{COACH} una vez que posee un mínimo de m muestras, tal como se realiza en [9].

- **Factor de regularización de entrenamiento α:** Se introduce este factor que busca regular la forma en la que aprende la política el agente. Este factor se encuentra acotado entre un valor máximo y mínimo, y posee una tasa de decaimiento lineal. Lo anterior implica que α decae a partir de una iteración que se determina de manera empírica, y afecta en la importancia de las funciones de costos utilizadas para actualizar al actor, además de si se debe considerar la actualización realizada por D-COACH o no. Esto se diseña de la siguiente manera:

  - **Regularización de las funciones de costos:** Se regula el impacto de las fun-

ciones de de costo, mediante el uso del valor del factor α, como factor que escala las funciones $J(\phi)$ y $F(\phi)$, que son las funciones de perdidas utilizadas en la actualización de la política de DDPG y D-COACH, respectivamente. Esto impacta en cuanto pondera el error de cada una sobre la actualización de los pesos de la red neuronal del actor. Esto esta implementado de forma que α al decaer en cada paso de tiempo, le da más importancia a la actualización realizada por DDPG, debido a que el factor pondera como $1 - \alpha$ sobre la función $J(\phi)$ utilizada originalmente en DDPG (Algoritmo 2). En contraposición, este factor le quita relevancia a lo largo del entrenamiento a la función $F(\phi)$, utilizada por D-COACH.

Este parámetro se fija en este caso, en el valor $\alpha = 0.5$ como factor inicial, con el fin de que ambas funciones posean la misma importancia al comienzo del entrenamiento, pero que esto mute una vez que este vaya transcurriendo. Así se utilizan el uso de la retroalimentación para acelerar la fase inicial del entrenamiento de manera correctiva sobre las acciones iniciales, acompañado del uso de la función de recompensas que logra modelar el comportamiento del agente a partir de las observaciones.

- **Umbral de relevancia:** Al momento de decaer a lo largo del tiempo de entrenamiento, α cuando toma valores que comienzan a ser pequeños, le entrega una muy baja relevancia al error calculado por la función $F(\phi)$. Por lo anterior, una vez que α obtiene valores iguales o inferiores al umbral k, se decide no actualizar más los pesos de las redes mediante el uso de D-COACH, y se continua solamente con la actualización de DDPG.

- **Temporal:** Este tipo de actualización se realiza cada x cantidad de iteraciones de entrenamiento, sobre los parámetros de D-COACH. Esta puede realizarse, una vez que se cumpla la cantidad mínima de m muestras en el *replay buffer* B_{COACH} y que se cumpla también que el factor de regularización haya superado el umbral k.

  Además, en cada iteración se realiza la actualización de las redes actor-crítico de DDPG, junto con sus respectivas redes *target*, utilizando la muestras del *replay buffer* B_{DDPG}. La actualización de parámetros de las redes *target*, se lleva a cabo utilizando los parámetros de las redes actor-crítico, sumado a un factor de suavizado de estos parámetros.

A lo largo del documento, se referirá a este algoritmo como híbrido.

Algoritmo 4: D-COACH y DDPG (híbrido)

Inicializar las redes de actor $\mu_\phi(s)$ y crítico $Q_\theta(s,a)$ con pesos ϕ y θ.
Inicializar las redes *target*, actor $\mu_{\bar{\phi}}(s)$ y crítico $Q_{\bar{\theta}}(s,a)$ con pesos $\bar{\phi}$ y $\bar{\theta}$.
Inicializar magnitud de error e y umbral de error para indicar el uso de *feedback* e_{umbral}.
Inicializar *Replay Buffers*: $B_{\text{D-COACH}} = []$ y B_{DDPG}.
Inicializar parámetros máximos y mínimos (m y M) para el *Replay Buffer* $B_{D\text{-}COACH}$.
Inicializar intervalo b de actualización del *Replay Buffer* $B_{D\text{-}COACH}$.
for t = *1,2,..,N* **do**
  Observar el estado s_t
  Ejecutar $a_t = \mu_\phi(s_t)$ sobre el ambiente.
  Observar la recompensa r_t y el nuevo estado s_{t+1}.
  Almacenar la transición (a_t, s_t, r_t, s_{t+1}) en B_{DDPG}.
  Obtener retroalimentación humana h_t y error de posición e_t
  while $\alpha > k$ **do**
    if $e_t > e_{umbral}$ **then**
      Se define el error $error_t = h_t \cdot e_{\text{fixed}}$
      Se define la etiqueta $y_{label} = a_t + error_t$
      Almacenar la transición (a_t, s_t) en $B_{\text{D-COACH}}$.
      Actualizar la red μ minimizando la función $F(\phi) = \alpha \cdot \frac{1}{N} \sum_{i=0}^{N} \left(y_{label} - \mu_\phi(s_t)\right)^2$, con par (s_t, y_{label})
      if $largo(B_{D\text{-}COACH}) \geq m$ **then**
        Actualizar la red μ minimizando la función $F(\phi)$, con minibatch sampleado del *Replay Buffer* $B_{\text{D-COACH}}$.
      end
      if $largo(B_{D\text{-}COACH}) > M$ **then**
        $B_{\text{D-COACH}} = B_{\text{D-COACH}}[2 : M+1]$
      end
    end
    if $mod(t,x) = 0$ ***and*** $largo(B_{D\text{-}COACH}) \geq m$ **then**
      Actualizar la red μ minimizando la función $F(\phi)$, con minibatch sampleado del *Replay Buffer* $B_{\text{D-COACH}}$.
    end
  end
  Samplear minibatch de N muestras aleatorias (a_i, s_i, r_i, s_{i+1}) de B_{DDPG}.
  Se define la etiqueta $y_i = r_i + \gamma Q_{\bar{\theta}}(s_{i+1}, \mu_{\bar{\phi}}(s_{i+1}))$
  Actualizar la red Q minimizando la función $L(\theta) = \frac{1}{N} \sum_{i=0}^{N} \left(y_i - Q_\theta(s_i, a_i)\right)^2$
  Actualizar la red μ maximizando la función $J(\phi) = (1-\alpha) \cdot \frac{1}{N} \sum_{i}^{N} Q_\theta(s_i, \mu_\phi(s_i))$
  Actualizar las redes *target*: $\bar{\theta} \leftarrow \lambda\theta + (1-\lambda)\bar{\theta}$ y $\bar{\phi} \leftarrow \lambda\phi + (1-\lambda)\bar{\phi}$
end

3.7. Función de recompensa

Para modelar la función de recompensa, se pretende considerar los aspectos planteados en [15]. Se utiliza la distancia euclidiana ρ_t^{objetivo} definida en (3.9), para determinar la distancia entre la pose actual del robot en el instante de tiempo t con respecto a la posición objetivo

a la cual se desea llegar.

$$\rho_t^{\text{objetivo}} = \sqrt{(x^{\text{objetivo}} - x_t)^2 + (y^{\text{objetivo}} - y_t)^2} \tag{3.9}$$

A partir de esta distancia, se generan tres alternativas para modelar la recompensa del agente (Ecuación 3.10). La primera consiste en la colisión del agente, a partir de la cual se le otorga una penalización grande. Luego, se generan dos intervalos posibles, uno que entrega una recompensa de éxito, una vez que el agente se encuentra dentro de un rango cercano a la posición objetivo a la cual se quiere llegar. La última es la recompensa de navegación, que se otorga en caso de que el robot no alcance el umbral mínimo de cercanía a la meta.

$$r_t = \begin{cases} r_t^{navegar} & \text{si} \quad \rho_t^{\text{objetivo}} > \rho_t^{\text{umbral}}, \\ r_t^{exito} & \text{si} \quad \rho_t^{\text{objetivo}} \leq \rho_t^{\text{umbral}}, \\ r_t^{colision} & \text{si} \quad \text{el agente colisiona.} \end{cases} \tag{3.10}$$

La recompensa de navegación, a su vez, se descompone en 4 recompensas:

- **Seguimiento del objetivo:** Consiste en una recompensa que se modela con el fin de que el agente mantenga su dirección a la posición objetivo. Esta se define a partir de la siguiente expresión:

$$r_t^{\text{objetivo}} = \frac{v_{max}}{\hat{v}_t^x} \cdot \cos\left(\theta_t^{\text{objetivo}}\right) + \text{A} \cdot \mathbb{1}_{\{\rho_t^{\text{objetivo}} \leq \rho_{t-1}^{\text{objetivo}}\}} - \text{B} \tag{3.11}$$

- **Peligro:** Penalización en caso de estar cerca de algún obstáculo. Se define una región de peligro definida en el intervalo $[d^{min}, d^{max}]$, tal que si la observación menor de O_{pcl} es mayor a d^{max} no hay peligro para el agente, en caso de estar dentro del intervalo, hay un peligro de colisión no grave y en caso de ser menor a d^{min} hay un peligro mayor. Dado el tipo de peligro que corre el agente, cambia la penalización entregada. Esta penalización, se introduce en este trabajo.

$$r_t = \begin{cases} \text{C}_1 & \text{si} \quad {O_{pcl}^{min}}_t > d^{max} \\ \text{C}_2 & \text{si} \quad {O_{pcl}^{min}}_t \in [d^{min}, d^{max}] \\ \text{C}_3 & \text{si} \quad {O_{pcl}^{min}}_t < d^{min} \end{cases} \tag{3.12}$$

- **Penalización a velocidad angular:** Penalización al agente cuando este navega con grandes velocidades angulares. También penaliza las grandes variaciones de velocidad angular entre cada paso de tiempo consecutivo.

$$r_t^{v_\theta} = -2K_t^{v_\theta} \cdot \mathbb{1}_{\{K_t^{v_\theta} > \text{D}\}} \tag{3.13}$$

$$K_t^{v_\theta} = \frac{1}{2v_{\max}^{\theta}} \max\{2|\hat{v}_t^{\theta}|, |\hat{v}_t^{\theta} - \hat{v}_{t-1}^{\theta}|\} \tag{3.14}$$

- **FOV:** Penalización a distancias angulares superiores a 240º del agente con respecto a la posición objetivo. Esto con el fin de orientar al agente de frente en su llegada a la

posición de destino.

$$r_{fov} = (3 \cdot \cos\left(\theta_t^{\text{objetivo}}\right) - 5) \cdot \mathbb{1}_{\{|\theta_t^{\text{objetivo}}| > 120\check{\text{r}}\}} \tag{3.15}$$

Así, finalmente, la función de recompensa de navegación, se describe como la suma ponderada de las recompensas mencionadas, donde el ponderador se fija con el fin de dar más/menos relevancia a cada componente. Esta se presenta en la Ecuación siguiente:

$$r_t^{navegar} = 2 \cdot r_t^{\text{objetivo}} + 0.75 \cdot r_t^{\text{peligro}} + 2 \cdot r_t^{v_\theta} + 0.75 \cdot r_{fov} \tag{3.16}$$

Los parámetros de las funciones utilizados se presentan en la Tabla 3.1.

Tabla 3.1: Parámetros utilizados en las penalizaciones presentes en la función de recompensas.

Parámetro	A	B	C_1	C_2	C_3	d^{min} (m)	d^{max} (m)	D
Valor	5	6	0	-2	-4	(0.75, 0.4)	(1.27, 0.97)	0.5

Capítulo 4

Entrenamiento y evaluación del agente en simulaciones

4.1. Ambiente de entrenamiento

Para entrenar a un agente bajo el paradigma del aprendizaje reforzado, se opta por el uso de una plataforma de simulación debido a que esta permite mayor flexibilidad en cuanto a diseño e implementación, además de control de parámetros del sistema de control y ambiente. Otra razón de peso, se relaciona con la fase de exploración necesaria en caso de tener que entrenar a una gente de RL en el mundo real, ya que esto puede implicar la ejecución de acciones que pueden ser potencialmente peligrosas tanto para el robot como para su entorno.

Se hace uso del simulador **Gazebo 11** [19], junto con el *framework* de **ROS-Noetic** [20], para caracterizar y desarrollar aplicaciones tanto para el ambiente de entrenamiento como para el agente robótico. El entrenamiento se lleva a cabo utilizando el lenguaje de programación **Python**, donde se emplea la librería **Pytorch 1.10.0** [39]. Se utiliza el sistema operativo **Ubuntu 20.04** para llevar a cabo este trabajo.

4.1.1. Mapa y posiciones válidas

Se utiliza el mapa del ambiente de simulación presentado en la Figura 4.1.b, para generar el plan global a utilizar como señal de retroalimentación automatizada, junto con la generación de poses iniciales y finales para cada episodio.

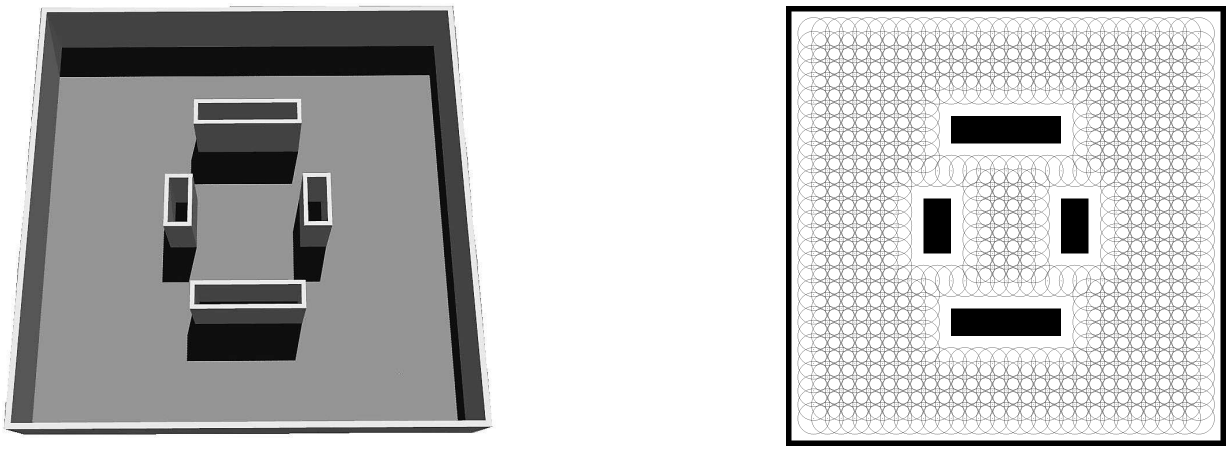
Para la generación del plan global, se hace uso de la librería *navfn* [40], que permite generar un camino desde la pose inicial del robot a la posición objetivo, evitando los obstáculos y las áreas cercanas a estos, con el fin de evitar rutas óptimas infactibles. Este camino generado, indica cada una de las poses que el agente debería tomar para lograr llegar de forma óptima a su destino. La granularidad de este plan depende de la resolución del mapa utilizado, y en este caso se posee un gran grado de concentración de poses para los planes generados; por lo que se realiza un submuestreo cada 15 muestras, con el fin de adaptar esta cantidad al agente utilizado. La muestra t+15 será equivalente a la muestra t+1 utilizada para el cálculo de la señal de error explicada en la Sección 3.3.

Dentro de este mapa se indican también las poses válidas que puede tomar el agente.

Estas son clave para determinar las posiciones iniciales y finales de cada episodio. Se calculan a partir de una estimación realizada, donde se obtiene el radio máximo del robot desde el centro de la base, sumado a un error de estimación de 25 cm, y se generan circunferencias en las que el robot puede aparecer en cualquier rotación realizada a partir del centro de estas. Se utiliza el radio máximo, en lugar del *footprint* por simplicidad de cálculo.

Una vez encontradas todas las poses posibles, se realiza un submuestreo de estas, y se seleccionan las poses de inicio y termino de cada episodio a partir de estas.

Los mapas utilizados para entrenamiento y validación del sistema, se introducen en [41] y han sido utilizados en otros trabajos como en [15, 42], donde el ambiente y mapa utilizados en entrenamiento (Figura 4.1), serán llamados ambiente y mapa simple. Para validación (Capítulo 5), se llamarán mapa y ambiente complejo. Ambos mapas poseen una dimensión de 16 mts^2.



(a) Ambiente Simple

(b) Mapa Simple

Figura 4.1: Ambiente y mapa de simulación utilizando en el entrenamiento y evaluación de las políticas del agente. **(a)** Ambiente simple simulado en **Gazebo**. **(b)** Mapa del ambiente simple, con las posiciones de inicio y término válidas para el agente en este ambiente de simulación.

4.1.2. Condiciones episódicas del ambiente

Se fijan las siguientes condiciones episódicas:

- **Distancia mínima entre pose inicial del agente y posición objetivo:** Se define una distancia mínima de 5 metros, entre la pose inicial del agente y la posición objetivo a la cual se desea llegar. Esto con el fin de eliminar los casos en donde la ruta sea muy sencilla debido a lo corto de la trayectoria.
- **Posiciones episódicas aleatorias:** En cada episodio se tiene una disposición distinta para la pose inicial y la posición final. Esta configuración, se determina de forma aleatoria al comienzo de cada episodio, considerando la restricción de distancia y las poses válidas generadas. Así el agente enfrentará desafíos distintos y variables en cada episodio.

- **Condiciones de término:** Se termina un episodio, cuando ocurre cualquiera de los tres siguientes eventos:
  - **Colisión del agente:** Este evento ocurre cuando la caja de colisión del agente tiene contacto con alguna de las murallas del mundo utilizado. Esto junto con la recompensa de valor negativo, busca evitar que el agente repita este comportamiento.
  - **Tiempo límite (*Time-Out*):** Esto ocurre cuando el agente no es capaz de llegar a la posición objetivo, en una cantidad determinada de pasos. Se fija una cantidad máxima de pasos por episodio, con el fin de acotar la cantidad de pasos que el agente emplea para llegar a un objetivo, y acotar la extensión de los episodios.
  - **Cumplimiento en el objetivo del agente:** Este caso ocurre cuando el agente logra llegar a la posición de destino, donde se considera un umbral mínimo de error entre la pose final del agente y la posición final deseada. Se asocia a una recompensa positiva, con el fin de modelar el comportamiento deseado para el agente.

4.2. Experimentos y Resultados

Se utilizarán los algoritmos DDPG, D-COACH, DDPG y COACH secuencial y el propuesto híbrido (Algoritmo híbrido de DDPG y COACH), para entrenar la política presentada en la Sección 3.5, una cantidad de cinco veces por algoritmo.

Los experimentos se realizan en la plataforma de simulación, siguiendo las condiciones episódicas mencionadas en la Sección anterior. Para cada episodio, el agente tendrá como máximo 200 interacciones con el ambiente, para cualquiera de los algoritmos utilizados. Se emplea un controlador que utiliza una tasa de frecuencia fija, velocidades lineales y angulares acotadas, con valores que se presentan en la Tabla 4.1.

El entrenamiento, se realiza con la finalidad de que el agente logre obtener una política con la cual pueda desempeñar bien la tarea de planificación local 2D, a lo largo de las iteraciones de entrenamiento.

Para evaluar el buen desempeño de las políticas del robot, sobre el mismo ambiente en donde se realiza el entrenamiento, se evalúan 200 veces las políticas obtenidas en entrenamiento cada x iteraciones de entrenamiento. Lo anterior, se hace con el fin de obtener distintas métricas de rendimiento, las cuales permiten verificar que tan bien funciona la política del agente.

Las métricas a utilizar en este trabajo se presentan a continuación:

- ***Avg. Episode Reward* o Retorno Promedio por Episodio:** Es el promedio de los retornos no descontados que obtiene el agente, a medida de que se evalúa su desempeño a lo largo de cada episodio de evaluación.
- ***Avg. Success Rate (SR)* o Tasa de Éxito Promedio:** Esta tasa, es la razón entre el número de episodios en los que el agente logra llegar a la posición objetivo, sobre el número de episodios en los que el agente es evaluado. Se utiliza un umbral de 0.3 metros, donde si el agente logra acercarse al objetivo de navegación a una distancia menor o igual a la distancia umbral, se considera que logró llegar al objetivo.

- ***Avg. Collision Rate (CR)* o Tasa de Colisiones Promedio:** Razón entre los episodios donde el agente colisionó sobre la cantidad de episodios totales de evaluación.
- ***Avg. Time-out Rate (TR)* o Tasa de Tiempo Límite Promedio:** Es la razón entre la cantidad de episodios de evaluación donde el agente supera el máximo de interacciones permitidas por episodio para lograr la tarea, sobre el número total de episodios.
- ***Avg. Episode Steps* Pasos Promedio por Episodio:** Es el promedio de las interacciones agente-ambiente obtenidas o pasos de tiempo, antes de llegar a un estado terminal.

Tabla 4.1: Parámetros del controlador utilizado.

	Parámetro	**Valor**
Control	Frecuencia de control (Hz)	5.0
	$[v^x_{min}, v^x_{max}]$ (m/s)	[-1.0, 1.0]
	$[v^\theta_{min}, v^\theta_{max}]$ (rad/s)	[-1.0, 1.0]

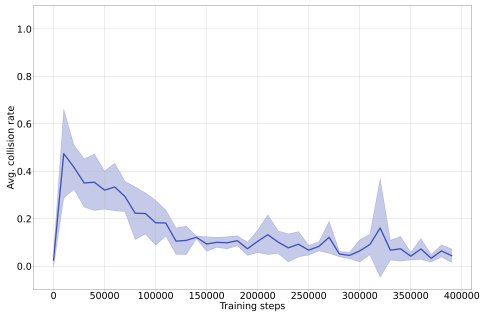
DDPG

Para llevar a cabo el entrenamiento utilizando DDPG en el ambiente de simulación, se realiza un entrenamiento de 400 mil pasos, donde cada uno de los episodios acepta una cantidad máxima de 200 interacciones agente-ambiente. A lo largo de cada paso de tiempo, se utiliza un ruido de exploración generado por un proceso estocástico *Ornstein–Uhlenbeck* [43], que es sumado a las acciones obtenidas por la política del agente, con el fin de introducir un proceso de exploración inicial que vaya decayendo a lo largo del entrenamiento, para lograr visitar una mayor cantidad de estados sin desviarse totalmente en la ejecución de la tarea. Este ruido se diseña a partir de los parámetro presentados en la Tabla 4.2, junto con los parámetros utilizados en el entrenamiento de la política por el algoritmo DDPG. En particular, no se hace uso del factor de decaimiento en los experimentos utilizados, conservando el valor del ruido inicial, ya que se encuentra una mejor performance con esta configuración.

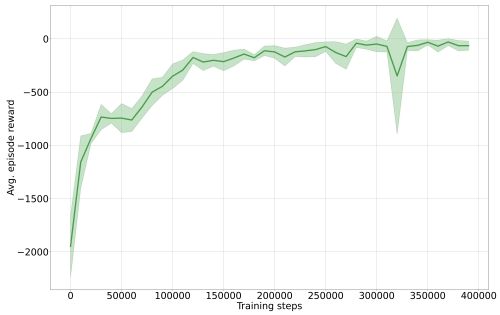
Tabla 4.2: Parámetros de entrenamiento utilizados por el algoritmo DDPG.

	Parámetro	**Valor**
DDPG	Tasa de aprendizaje del actor	0.0001
	Tasa de aprendizaje del crítico	0.001
	Factor de suavizado λ	0.001
	Factor de descuento γ	0.99
	Tamaño máximo del *Experience Replay*	400000
	Tamaño del *minibatch*	256
Ruido de Exploración O-U	Factor Inicial	1.0
	Frecuencia de Sampleo (Hz)	2.5
	μ	0.0
	σ	0.3
	θ	0.15

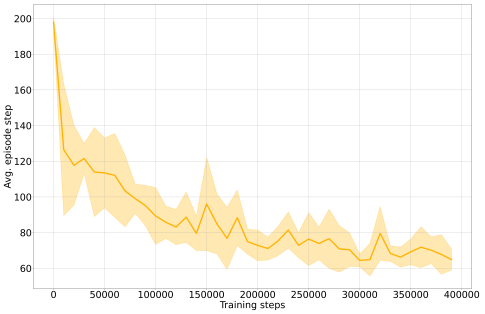
Los resultados relacionados con la evolución de la política a lo largo del entrenamiento, se presentan en la Figura 4.2. A partir de la Figura 4.2.d, donde se presenta la tasa de éxito, se observa que la curva posee un comportamiento convergente sobre el 90 % de éxito, encontrando su valor más alto en la iteración número 370000. Esto indica que la tarea logra ser ejecutada con un buen desempeño al ser evaluada sobre el mismo ambiente donde esta fue entrenada. Así mismo, al interpretar la curva asociada a la tasa de colisiones (Figura 4.2.a), se tiene que esta no supera el 10 % a partir de los 350000 pasos de tiempo y que al comenzar no superaba el 50 % de las veces en las que ocurría, por lo que es un factor que se logra mejorar y minimizar rápidamente durante los primeros 150 mil pasos de entrenamiento. Así mismo se logra ver una optimización en la cantidad de pasos (interacciones agente-ambiente) ejecutados a lo largo del entrenamiento (Figura 4.2.c) y una curva de recompensas que logra obtener valores de penalización cercanos a cero finalizando el entrenamiento (Figura 4.2.b), notando también que los valores más negativos del comienzo del entrenamiento son asignados principalmente por la mayor tasa de colisiones y de *time-out* del agente.



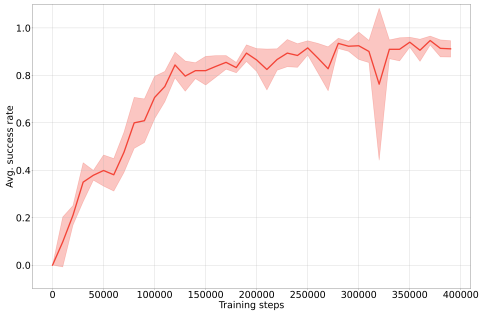
(a) Collision Rate



(b) Avg. Reward



(c) Avg. Steps



(d) Success Rate

Figura 4.2: Resultados de DDPG: Evaluación de política en ambiente simple.

D-COACH

Para entrenar el agente utilizando D-COACH, se emplean 80 mil pasos de entrenamiento debido a que mantiene un comportamiento de carácter estacionario al extender más el entrenamiento, sin encontrar una solución que mejore de forma considerable.

Se utilizan los parámetros presentados en la Tabla 4.3, donde se presentan los valores empleados para el cálculo de la señal de *feedback* automatizado empleado en el entrenamiento.

Tabla 4.3: Parámetros de entrenamiento de D-COACH.

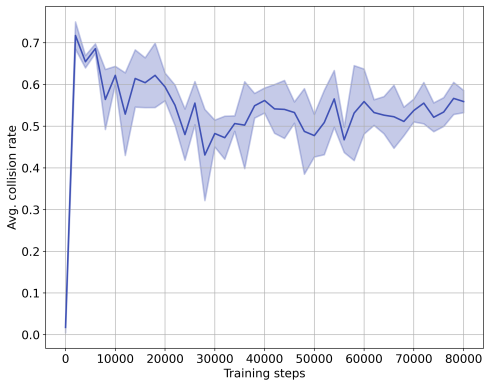
	Parámetro	**Valor**
D-COACH	Tasa de aprendizaje de la política	0.00025
	Error fijo e_{fixed} (m)	[1.0, 0.25]
	Error umbral e_{umbral} (m)	[0.01, 0.02]
	Tamaño máximo del Experience Replay	1000
	Tamaño mínimo del Experience Replay	16
	Tamaño del mini batch	256
	Intervalo de actualización b	1

En el caso de D-COACH, los resultados obtenidos se presentan en la Figura 4.3. A lo largo del entrenamiento, se tiene una cantidad de pasos promedio cercana a los 75 pasos por episodio (Figura 4.3.c).

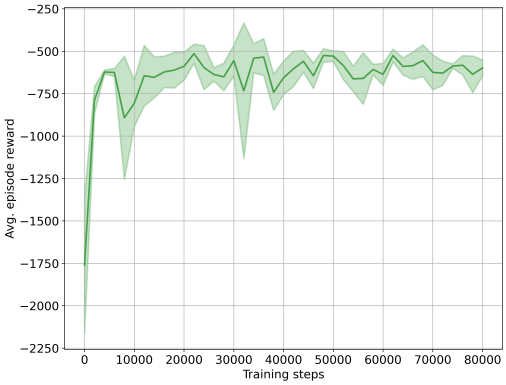
La mejor política se encuentra casi a la mitad del entrenamiento en los 50 mil pasos. Si bien D-COACH no logra llegar más allá de un 55 % en la ejecución correcta de la tarea de navegación local con el set de parámetros encontrados (Figura 4.3.d), se logra llegar a este valor antes de las 60 mil iteraciones, donde DDPG alcanza sólo un 40 % y logra alcanzar el 60 % recién a los 100 mil pasos de tiempo.

Uno de los problemas de D-COACH, es que no identifica cuando está logrando de forma exitosa la tarea y cuando esta cometiendo una penalización grave, lo cual generalmente se introduce mediante la función de recompensas en el caso del aprendizaje reforzado. Debido a lo anterior, es difícil lograr una mejor *performance*, ya que solo aprende de la señal de retroalimentación correctiva. Pese a lo anterior, logra llegar a una solución buena en una cantidad inferior de pasos de tiempo.

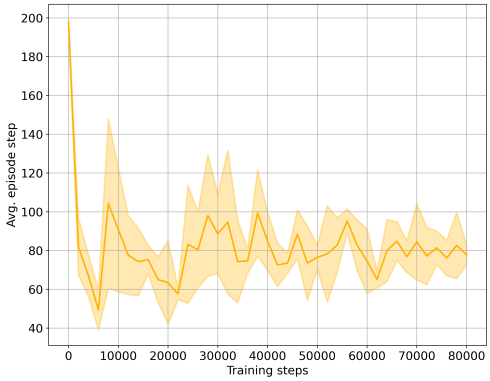
La cantidad de colisiones a lo largo del entrenamiento no logra bajar del 40 % de las veces (Figura 4.3.a), reflejado en la función de recompensas promedio (Figura 4.3.b) que se estabiliza en las penalizaciones de valores negativos cercanos a -500. Lo anterior indica que, a pesar de ser entrenado en un ambiente simple con una baja cantidad de obstáculos, tiene dificultad para evadirlos en la realización de la tarea.



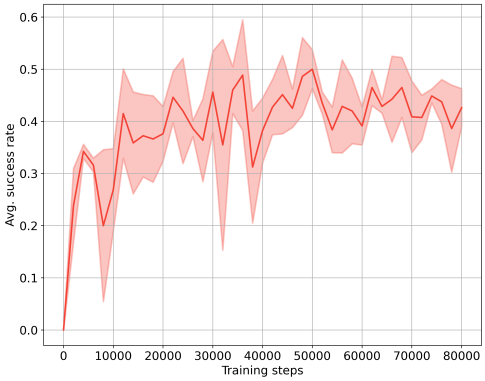
(a) Collision Rate



(b) Avg. Reward



(c) Avg. Steps

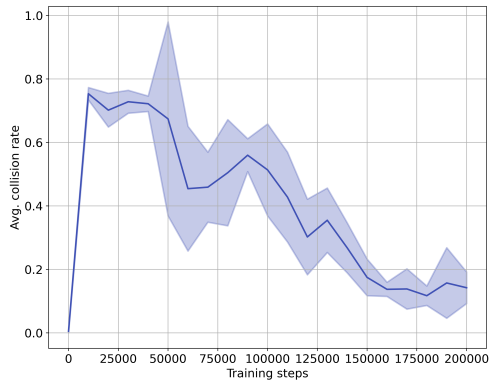


(d) Success Rate

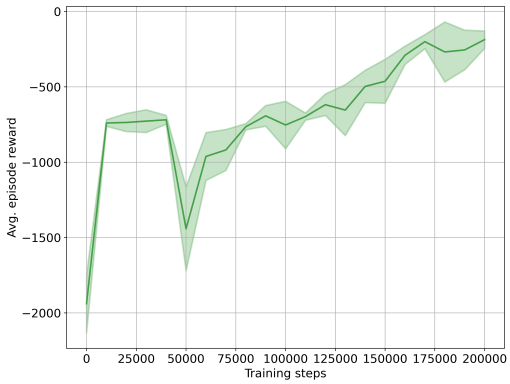
Figura 4.3: Resultados de D-COACH: Evaluación de política en ambiente simple.

DDPG y D-COACH: Secuencial

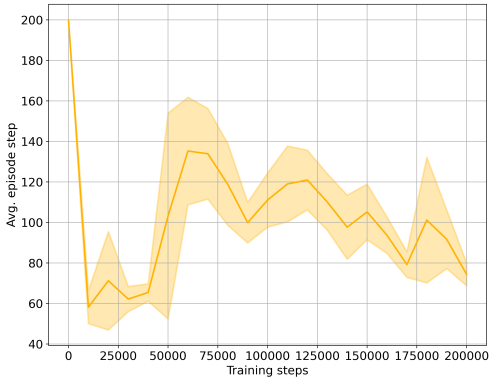
Este algoritmo, emplea los mismos parámetros utilizados en D-COACH y DDPG, mencionados en las Tablas 4.3 y 4.2. Se entrena en un total de 200 mil iteraciones, tal que se entrena primero la política utilizando el algoritmo D-COACH a lo largo de 50 mil pasos, paso de tiempo donde logra el mejor desempeño este algoritmo por si sólo; y luego se continua con el entrenamiento, utilizando el algoritmo DDPG hasta los 200 mil pasos.



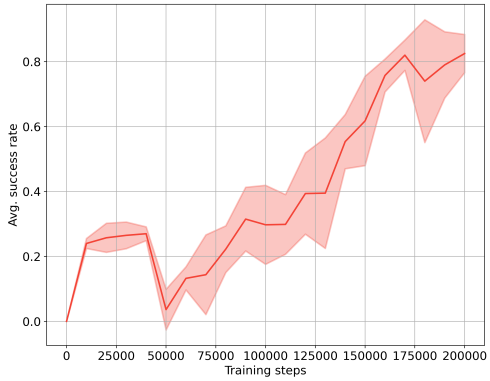
(a) Collision Rate



(b) Avg. Reward



(c) Avg. Steps



(d) Success Rate

Figura 4.4: Resultados de DDPG y COACH (Secuencial): Evaluación de política en ambiente simple.

Los resultados obtenidos con el algoritmo secuencial, revelan que al usar DDPG posterior a encontrar una política buena definida por D-COACH, si bien permite llegar a un 90 % de tasa de éxito a las 180 mil iteraciones como máximo, en promedio sólo logra alcanzar un 74 % mientras que DDPG posee una tasa de un 83.3 % (Figura 4.4.d). Lo anterior, indica que este algoritmo posee un buen desempeño en una baja cantidad de iteraciones, pero no es capaz de mejorar la eficiencia de muestras en este contexto.

En las Figuras 4.4.c, 4.4.a y 4.4.b, se puede observar que tiene un comportamiento favorable en la disminución de colisiones a lo largo del entrenamiento (bajo un 20 %), junto con un aumento en las recompensas obtenidas, logrando también una cantidad promedio de pasos por episodio cercana a los 75 pasos, lo cual indica que también descienden las veces en las que el agente no logra llegar al objetivo del episodio por un *time-out*.

DDPG y D-COACH: Híbrido

Este algoritmo es entrenado a lo largo de 170 mil pasos, y emplea los parámetros presentados en la Tabla 4.4. Para el caso de las acciones utilizadas por este algoritmo, se utiliza el mismo ruido de exploración, pero que comienza con un factor de 0.3 y decae desde los 82 mil pasos. En la tabla son especificados los parámetros utilizados por este algoritmo.

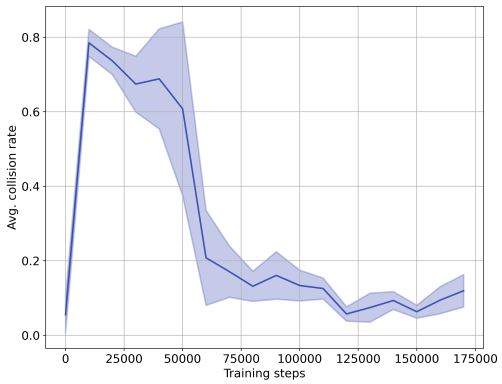
Tabla 4.4: Parámetros de entrenamiento de Algoritmo híbrido (COACH Y DDPG).

	Actualiza	**Parámetro**	**Valor**
D-COACH & DDPG (Híbrido)	D-COACH	Tasa de aprendizaje del actor	0.0002
		Error fijo e_{fixed} (m)	[1.0, 0.15]
		Error umbral e_{umbral} (m)	[0.01, 0.028]
		Tamaño máximo del Experience Replay	256
		Tamaño mínimo del Experience Replay	16
		Tamaño del mini batch	128
		Intervalo de actualización b	8
	DDPG	Tasa de aprendizaje del actor	0.0001
		Tasa de aprendizaje del crítico	0.001
		Tamaño del mini batch	128
		Tamaño máximo del Experience Replay	400000
	Híbrida	Factor inicial de regularización α_0	0.5
		Factor final de regularización α_F	0.05

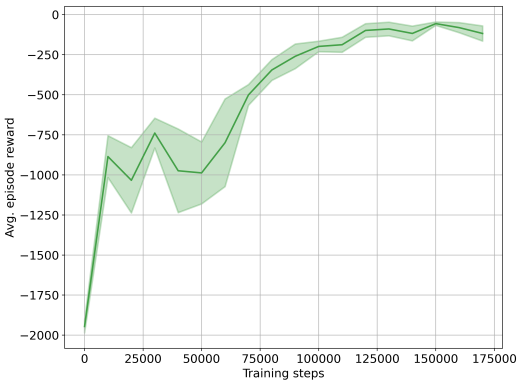
Los resultados asociados al algoritmo híbrido (Figura 4.5), son bastante alentadores ya que al fusionar ambas técnicas, se logra llegar a un 93 % de tasa de éxito en la iteración 150 mil, lo que reduce en un 58 % la cantidad de iteraciones utilizadas para encontrar un buen ajuste de parámetros sobre el aproximador de función utilizado en comparación con DDPG (Figura 4.5.d). Si bien el valor es levemente más bajo que el obtenido con DDPG, logra superar el 90 % lo cual es excelente.

Al comparar el algoritmo híbrido con el algoritmo secuencial, donde ambos utilizan DDPG y COACH en su composición, se puede observar que ambos cumplen con el objetivo de eficiencia de muestras, con respecto a un algoritmo de DRL como DDPG. El algoritmo propuesto logra tener un mejor desempeño tanto en eficiencia de muestras como en tasa de éxito obtenida, logrando llegar a una política con una tasa máxima de 93 % a las 150 mil iteraciones, y una tasa promedio de 91.79 % que supera tanto a DDPG que sólo alcanza una tasa de éxito de un 82 % como al algoritmo secuencial que llega a una tasa de 61.75 %.

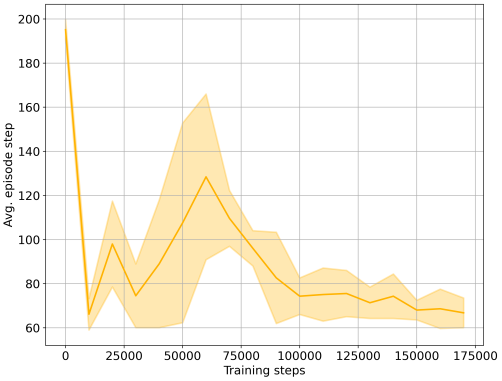
En la Figura 4.5.a, se observa que la tasa de colisiones es más alta al comienzo en comparación con los algoritmos anterior alcanzando valores cercanos al 80 %, pero logra decaer de forma abrupta cerca de los 60 mil pasos de entrenamiento, rodeando valores cercanos al 10 %. Las recompensas episódicas promedio (Figura 4.5.d), logran converger a penalizaciones pequeñas cercanas a 0, presentando las mayores penalizaciones a los episodios iniciales donde hay una mayor tasa de colisiones y de *time-out*. Aquí se llega a una cantidad de pasos promedio de 70 por episodio (Figura 4.5.c).



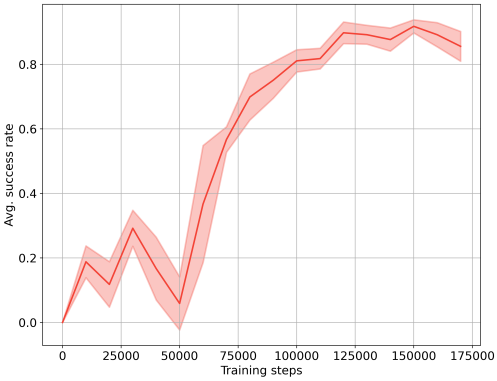
(a) Collision Rate



(b) Avg. Reward



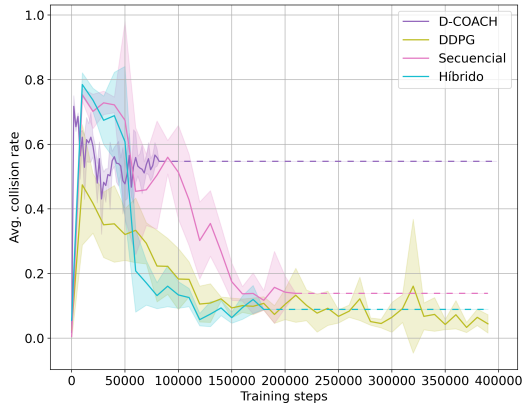
(c) Avg. Steps



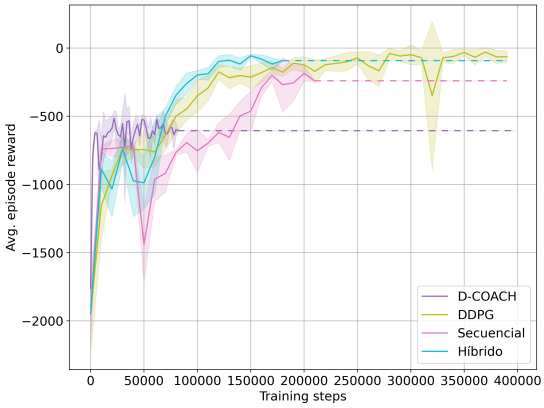
(d) Success Rate

Figura 4.5: Resultados de DDPG y COACH (Híbrido): Evaluación de política en ambiente simple.

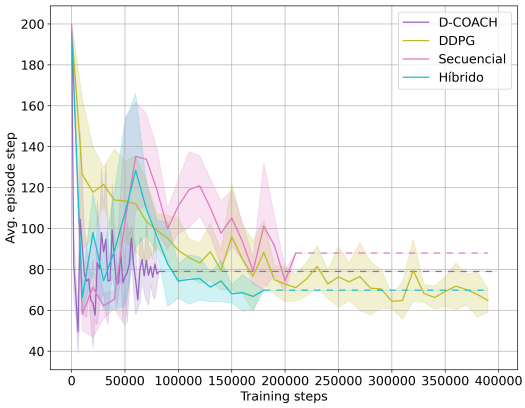
En las Figuras 4.6.a, 4.6.b, 4.6.c y 4.6.d, se puede visualizar la evolución del entrenamiento de todos los algoritmos en conjunto, donde las líneas punteadas representan el promedio de los últimos 5 puntos de entrenamiento con el fin de mostrar la tendencia del entrenamiento hasta ese punto. Se evidencia con mayor claridad el excelente desempeño que posee el algoritmo híbrido con respecto al resto de los algoritmos evaluados.



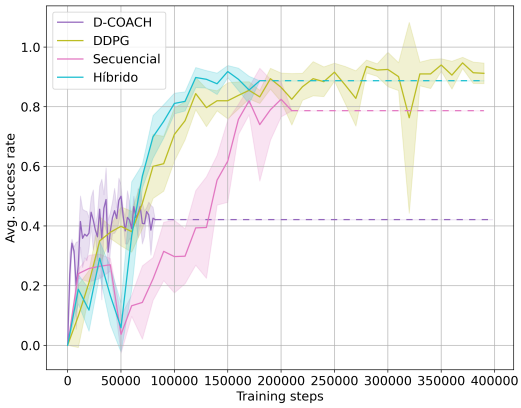
(a) Collision Rate



(b) Avg. Reward



(c) Avg. Steps



(d) Success Rate

Figura 4.6: Resultados de todos los algoritmos: Evaluación de política en ambiente simple.

Finalmente se puede observar en las Tablas 4.5 y 4.6, el resumen de la evaluación realizada sobre los 4 algoritmos implementados, sobre la iteración con mejor desempeño.

Tabla 4.5: Resultados de Evaluación: Promedio de las métricas obtenidas a partir de la mejor política obtenida en entrenamiento para cada experimento.

Algoritmo	**SR**	**TR**	**CR**	**Avg. Steps**
DDPG	0.947 ± 0.019	0.034 ±0.03	0.033 ± 0.016	70.135 ± 7.478
D-COACH	0.5 ± 0.037	0.021 ±0.014	0.477 ± 0.051	76.501 ± 6.297
Híbrido	0.918 ± 0.02	0.019 ± 0.011	0.063 ± 0.017	68.05 ± 4.457
Secuencial	0.82 ± 0.025	0.034 ± 0.011	0.138 ± 0.064	79.221 ± 6.345

Tabla 4.6: Resultados de Evaluación: Mejor política para cada algoritmo, obtenida del total de experimentos realizados.

Algoritmo	**SR**	**TR**	**CR**	**Avg. Steps**
DDPG	0.975	0.005	0.02	61.81
D-COACH	0.605	0.01	0.385	82.93
Híbrido	0.945	0.015	0.04	76.09
Secuencial	0.9	0.03	0.069	79.32

En la tabla 4.5, se presentan los valores promediados de la iteración donde se tiene la mejor política de entrenamiento total para cada algoritmo, sobre los 5 experimentos realizados, junto con la desviación estándar asociada a cada uno de estos. Se observa que el algoritmo híbrido y DDPG, poseen una muy buena tasa de éxito superior a un 90 %, y con una baja desviación estándar en los resultados. El algoritmo secuencial, les sigue con una tasa de éxito promedio de un 82 %, y una baja variabilidad en la cantidad de pasos promedio por episodio de entrenamiento. D-COACH presenta un desempeño menor, con una tasa de éxito de un 50 %, pero en una cantidad de pasos mucho menor que cualquiera de los otros algoritmos, lo que indica el gran potencial que posee este algoritmo, al momento de encontrar una buena política en pocas iteraciones.

Por otro lado, en la Tabla 4.6, se tienen los resultados obtenidos para las mejores políticas de cada algoritmo. En esta última, se observa que algoritmo híbrido si bien no logra alcanzar el mismo desempeño obtenido por DDPG, logra llegar a una tasa de éxito de la tarea lo suficientemente alta (94.5 %) a los 150 mil iteraciones de entrenamiento a diferencia de DDPG que logra en 370 mil alcanzar un máximo de 97.5 %, lo cual optimiza notablemente la rapidez del aprendizaje del agente, gracias al complemento de D-COACH.

Ambos algorítmos que usan DDPG y D-COACH en su composición, logran obtener políticas con un buen desempeño, pero sólo el algoritmo híbrido es capaz de mejorar la eficiencia de muestras en el entrenamiento de manera exitosa.

Capítulo 5

Validación de las políticas del agente en simulaciones

Este capitulo, presenta los resultados obtenidos a partir de la validación de la política entrenada para cada algoritmo en simulación. Esto se realiza en un ambiente distinto nunca antes visto por el agente, con el fin de validar que el agente tenga la capacidad de poder resolver la tarea de navegación local de manera generalizada, junto con observar su comportamiento y evaluar su desempeño. Se validarán las mejores políticas obtenidas para cada algoritmo, tomando en cuenta la iteración donde se tuvo mejor desempeño en promedio de los 5 experimentos realizados.

(a) Ambiente complejo

(b) Mapa complejo

Figura 5.1: Ambiente y mapa de simulación utilizando en la validación de las políticas del agente. **(a)** Ambiente complejo simulado en **Gazebo**. **(b)** Mapa del ambiente complejo, con las poses de inicio y término válidas para el agente en este ambiente de simulación.

Para validar cada una de las mejores políticas obtenidas por cada algoritmo, se consideran las mismas condiciones episódicas utilizadas en el entrenamiento, donde estas son evaluadas 500 veces en el ambiente de validación presentado en la Figura 5.1. En este ambiente se presenta una dificultad de obstáculos mayor, pero se mantiene la forma de las paredes exteriores y tamaño del ambiente de 16mts^2, con un alto y ancho de 16 metros.

Los resultados de validación se presentan en las Tablas 5.1 y 5.2, donde se observa que sólo DDPG logra obtener un rendimiento superior al 90 % en la tasa de éxito de la tarea, tanto en promedio de la evaluación de la mejor métrica en el ambiente complejo, como la mejor métrica obtenida a partir de dicha política. En el caso de D-COACH, el agente falla de manera rotunda al cambiar de ambiente, no logrando alcanzar ni un 20 % en la tasa de éxito, y tomando valores cercanos al 90 % en la tasa de colisiones. Lo anterior indica, que DDPG es capaz de entrenar una política que puede generalizar su aprendizaje de la tarea en distintos espacios sin el uso de un mapa, mientras que D-COACH sólo es capaz de entregar políticas que si bien no funcionen de manera perfecta por sí solas, logran converger rápidamente en una tasa de éxito superior al 50 %, que sólo son validas para el mismo ambiente de navegación utilizado en entrenamiento. El algoritmo secuencial por su lado, obtiene valores similares al algoritmo híbrido propuesto, logrando un mejor desempeño que D-COACH, pero no supera al algoritmo híbrido en cuanto a eficiencia de muestras ni en la tasa de éxito.

Tabla 5.1: Resultados de Validación: Promedio de las métricas obtenidas a partir de la mejor política obtenida en entrenamiento para cada experimento.

Algoritmo	**SR**	**TR**	**CR**	**Avg. Steps**
DDPG	0.91 ± 0.064	0.02 ± 0.007	0.072 ± 0.056	70.197 ± 6.332
D-COACH	0.104 ± 0.038	0.0175 ± 0.025	0.878 ± 0.061	49.92 ± 18.58
Híbrido	0.52 ± 0.074	0.0267 ± 0.013	0.453 ± 0.086	58.19 ± 9.475
Secuencial	0.47 ± 0.006	0.058 ± 0.054	0.47 ± 0.005	71.41 ± 10.1

Tanto el algoritmo híbrido como el secuencial, heredan la falencia de D-COACH, ya que no logran generalizar bien su aprendizaje en este ambiente nuevo, donde el algoritmo híbrido posee una mejor tasa de éxito máxima (cercana al 58 %), pero que no es suficiente para ser utilizada como una política autosuficiente y autónoma.

Esto expone el *trade-off* entre la eficiencia de muestras y la generalización de aprendizaje de la tarea. Al introducir la aceleración del aprendizaje por parte del algoritmo D-COACH sobre el algoritmo DDPG, se pierde generalización por parte de la política del agente de la tarea a realizar.

Tabla 5.2: Resultados de Validación: Mejor política para cada algoritmo, obtenida del total de experimentos realizados.

Algoritmo	**SR**	**TR**	**CR**	**Avg. Steps**
DDPG	0.956	0.014	0.03	61.47
D-COACH	0.156	0.062	0.78	80.05
Híbrido	0.582	0.042	0.376	65.5
Secuencial	0.48	0.025	0.495	75.43

Capítulo 6

Conclusión y trabajo futuro

La presente memoria busca introducir, explicar teóricamente, revisar el estado del arte relacionado, formalizar y presentar los resultados obtenidos, a partir del estudio de la implementación de un sistema híbrido que logre integrar el aprendizaje reforzado profundo con el aprendizaje de maquinas interactivo, más en específico con una señal de retroalimentación correctiva. Lo anterior, con el fin de acelerar el aprendizaje de un agente robótico móvil, en el contexto de navegación robótica local.

A lo largo del trabajo, se presentan los principios teóricos y conceptos clave, que buscan dar una base de conocimiento al lector, con el propósito de que pueda asimilar y entender la teoría detrás del aprendizaje de máquinas interactivo, aprendizaje reforzado profundo y la navegación robótica local. Se presentan luego, algoritmos y trabajos relacionados que permiten dar un piso sobre el cual se puede plantear el trabajo actual.

Finalmente, se propone una formalización al problema de estudio y se presentan los resultados asociados a esta propuesta. Este problema se formaliza mediante la propuesta, diseño e implementación de un algoritmo híbrido de aprendizaje reforzado profundo con el uso de una señal de retroalimentación correctiva, entregada a partir de un planificador global, que da a conocer las poses que debería tomar el agente a lo largo de la ejecución de la tarea a resolver. Esta señal será utilizada sobre las acciones del agente móvil, que en este caso son las velocidades lineal y angular.

Esto se implementa en simulaciones utilizando el *framework* de **ROS-Noetic**, en el contexto de navegación robótica local, donde se utiliza el robot *Clearpath Husky A200* junto con el uso de sensores láser *Hokuyo UTM-30LX-EW*.

El algoritmo propuesto, se construye a partir de los algoritmos DDPG (DRL) y D-COACH (IML). Este algoritmo híbrido, es comparado junto con los algoritmos que fueron utilizados para su construcción, en otras palabras, los algoritmos DDPG y D-COACH a secas, junto con un algoritmo secuencial construido a partir de estos dos. Estos 4 algoritmos, se emplean para entrenar una política, la cuál es evaluada y validada en simulaciones.

El objetivo general consiste en realizar la tarea de planificación local 2D, donde esta además de ser cumplida, debe realizarse tal que su aprendizaje sea más rápido o acelerado, mediante el uso de este algoritmo híbrido.

Este algoritmo es diseñado de tal forma, que se busca compatibilizar y potenciar los beneficios de cada algoritmo por separado, con la intención de llegar a una política deseada en la menor cantidad de interacciones posibles, durante la etapa de entrenamiento. Posteriormente, se realiza una evaluación de este en el mismo ambiente de entrenamiento, seguido de una validación en un ambiente nuevo para el agente.

En la evaluación de los cuatro algoritmos, se logran resultados satisfactorios, ya que tanto el algoritmo híbrido como el secuencial logran una excelente solución sobre el 90 % de tasa de éxito, para la mejor política, pero solo el algoritmo híbrido permiten obtener una eficiencia de muestras en el entrenamiento del agente, en un 58 % menos de pasos de entrenamiento.

Al momento de validar las mejores políticas obtenidas por cada algoritmo en un ambiente de simulación distinto, DDPG logra obtener políticas con un desempeño superior al 90 % en su tasa de éxito, mientras que el algoritmo híbrido alcanza como mejor tasa de éxito un 58 %. D-COACH, por su parte obtiene una tasa de éxito inferior al 20 %, lo que demuestra lo difícil que es para este algoritmo generalizar el aprendizaje obtenido.

Dado lo anterior, se concluye que si bien el uso de los algoritmo híbrido y secuencial dentro de un ambiente específico logra acelerar de manera exitosa el aprendizaje de la tarea de planificación local 2D, no son buenos al momento de generalizar la tarea y emplear la política en ambientes distintos, efecto que pudo haber sido inducido por el uso de D-COACH en ambos algoritmos, ya que este le dice al algoritmo que hacer, pero nunca que no debe hacer. Además del uso del plan global, que condiciona el aprendizaje del agente a la morfología del ambiente de entrenamiento, por la falta de exploración.

El algoritmo híbrido logra un mejor desempeño que el algoritmo secuencial, tanto en la tasa de éxito de la política como en la cantidad de iteraciones en las que llega al mejor resultado. Lo anterior, invita a profundizar en el estudio de la implementación realizada, sus propiedades y el potencial que este pueda tener.

Como trabajo futuro, se propone hacer un análisis en los factores que puedan afectar en la generalización del aprendizaje en el algoritmo híbrido. Un factor a considerar, puede ser el de poner énfasis en la etapa de exploración del agente, mediante el estudio de la magnitud del ruido de exploración utilizado o el uso de una señal de retroalimentación correctiva con probabilidad aleatoria. Otro factor interesante de estudiar, puede ser la exploración de distintos tipos de algoritmos de IML, con el fin de mejorar la eficiencia de muestras de manera generalizada. El factor relacionado con la naturaleza de la señal de retroalimentación correctiva, en el sentido de ser entregada por un humano o de manera artificial, es otra línea de investigación interesante de estudiar, ya que bajo un ambiente con seguridad tanto para el robot como el ambiente, se puede hacer entrenamiento del agente robótico sin el uso de simuladores al utilizar una señal de *feedback* humano.

Bibliografía

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

[2] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[3] A. M. TURING, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, pp. 433–460, 10 1950.

[4] P. V. N. Korupolu, "Integrating human instructions and reinforcement learners : An srl approach," 2012.

[5] A. Najar and M. Chetouani, "Reinforcement learning with human advice. A survey," *CoRR*, vol. abs/2005.11016, 2020.

[6] V. Gullapalli and A. Barto, "Shaping as a method for accelerating reinforcement learning," in *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pp. 554–559, 1992.

[7] W. B. Knox and P. Stone, "Interactively shaping agents via human reinforcement: The tamer framework," in *Proceedings of the Fifth International Conference on Knowledge Capture*, K-CAP '09, (New York, NY, USA), p. 9–16, Association for Computing Machinery, 2009.

[8] C. Celemin and J. Ruiz-del Solar, "Coach: Learning continuous actions from corrective advice communicated by humans," in *2015 International Conference on Advanced Robotics (ICAR)*, pp. 581–586, 2015.

[9] R. Pérez-Dattari, C. Celemin, J. Ruiz-del-Solar, and J. Kober, "Interactive learning with corrective feedback for policies based on deep neural networks," *CoRR*, vol. abs/1810.00466, 2018.

[10] C. Celemin, J. Ruiz-del Solar, and J. Kober, "A fast hybrid reinforcement learning framework with human corrective feedback," *Autonomous Robots*, vol. 43, pp. 1173–1186, Jun 2019.

[11] X. Chen and C. Hickey, "Parallelized interactive machine learning on autonomous vehicles," in *NAECON 2018 - IEEE National Aerospace and Electronics Conference*, pp. 286–291, 2018.

[12] I. Moreira, J. Rivas, F. Cruz, R. Dazeley, A. Ayala, and B. Fernandes, "Deep reinforcement learning with interactive feedback in a human–robot environment," *Applied Sciences*, vol. 10, no. 16, 2020.

[13] G. Warnell, N. Waytowich, V. Lawhern, and P. Stone, "Deep tamer: Interactive agent shaping in high-dimensional state spaces," 2018.

[14] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[15] F. Leiva and J. Ruiz-del Solar, "Robust rl-based map-less local planning: Using 2d point clouds as observations," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5787–5794, 2020.

[16] N. N. S. a. Amitava Chatterjee, Anjan Rakshit, *Vision Based Autonomous Robot Navigation: Algorithms and Implementations*. Studies in Computational Intelligence 455, Springer-Verlag Berlin Heidelberg, 1 ed., 2013.

[17] C. Robotics, "Husky ugv - outdoor field research robot by clearpath." https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/.

[18] Hokuyo, "Utm-30lx-ew - hokuyo usa." https://hokuyo-usa.com/products/lidar-obstacle-detection/utm-30lx-ew.

[19] N. Koenig and A. 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)*, vol. 3, pp. 2149–2154 vol.3, 2004.

[20] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system."

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[22] OpenAI, "Openai five." https://blog.openai.com/openai-five/, 2018.

[23] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan 2016.

[24] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[25] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[26] K. P. Murphy, *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013.

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

[28] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293–321, May 1992.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning.," in *ICLR* (Y. Bengio and Y. LeCun, eds.), 2016.

[31] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 387–395, PMLR, 22–24 Jun 2014.

[32] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4423–4430, 2018.

[33] K. Zhu and T. Zhang, "Deep reinforcement learning based mobile robot navigation: A review," *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021.

[34] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, p. 1527–1533, IEEE, 2017.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[36] R. Zhang, F. Torabi, G. Warnell, and P. Stone, "Recent advances in leveraging human guidance for sequential decision-making tasks," *Autonomous Agents and Multi-Agent Systems*, vol. 35, Jun 2021.

[37] "global-planner." http://wiki.ros.org/global_planner.

[38] "ira-laser-tools." http://wiki.ros.org/ira_laser_tools.

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[40] "navfn." http://wiki.ros.org/navfn.

[41] L. Xie, S. Wang, S. Rosa, A. Markham, and N. Trigoni, "Learning with training wheels: Speeding up training with a simple controller for deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6276–6283, May 2018.

[42] N. G. Marticorena Vidal, "Manipulación móvil mediante aprendizaje reforzado profundo," 2021.

[43] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Phys. Rev.*, vol. 36, pp. 823–841, Sep 1930.