

UNIVERSIDAD MAYOR DE SAN ANDRES
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA ELECTRONICA



PROYECTO DE GRADO

**“CONTROL DE UN PÉNDULO INVERTIDO BASADO EN
APRENDIZAJE REFORZADO PROFUNDO”**

POSTULANTE: RANIERO HUMBERTO CALDERÓN ANTEZANA

TUTOR: ING. OSCAR MAURICIO AMÉSTEGUI MORENO

LA PAZ 2022



**UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE INGENIERIA**



LA FACULTAD DE INGENIERIA DE LA UNIVERSIDAD MAYOR DE SAN ANDRÉS AUTORIZA EL USO DE LA INFORMACIÓN CONTENIDA EN ESTE DOCUMENTO SI LOS PROPÓSITOS SON ESTRICAMENTE ACADÉMICOS.

LICENCIA DE USO

El usuario está autorizado a:

- a) Visualizar el documento mediante el uso de un ordenador o dispositivo móvil.
- b) Copiar, almacenar o imprimir si ha de ser de uso exclusivamente personal y privado.
- c) Copiar textualmente parte(s) de su contenido mencionando la fuente y/o haciendo la cita o referencia correspondiente en apego a las normas de redacción e investigación.

El usuario no puede publicar, distribuir o realizar emisión o exhibición alguna de este material, sin la autorización correspondiente.

TODOS LOS DERECHOS RESERVADOS. EL USO NO AUTORIZADO DE LOS CONTENIDOS PUBLICADOS EN ESTE SITIO DERIVARA EN EL INICIO DE ACCIONES LEGALES CONTEMPLADAS EN LA LEY DE DERECHOS DE AUTOR.

*Dedicado a mi familia,
fuentes de amor incondicional y de apoyo,
ejemplos fundamentales de vida.*

Agradecimientos

Agradezco a mis familia Humberto y Roxana, por su incansable apoyo, paciencia e infinito amor, a los cuales atribuyo este y cada uno de los logros alcanzados a lo largo de mi vida.

A mi tutor Oscar Mauricio Améstegui Moreno, por sus valiosas enseñanzas a lo largo de mi crecimiento dentro la carrera, y por su invaluable guía en el desarrollo del presente proyecto.

A mi novia Sarah, por su apoyo a lo largo del desarrollo de este proyecto y por su constante e incondicional entrega de amor.

A los profesores de la carrera, por haberme transmitido de manera desinteresada su invaluable conocimiento.

Resumen

Los sistemas inteligentes capaces de aprender y desarrollar comportamientos complejos a partir de experiencias, se han desarrollado de manera significativa durante la última década dentro un campo emergente denominado aprendizaje reforzado profundo, el cual posee interés por parte de la academia y la industria a nivel mundial. El presente proyecto se enfoca en el desarrollo de un sistema de aprendizaje reforzado profundo aplicado tareas de control en el mundo real. Se desarrolla un agente capaz de aprender a controlar el péndulo invertido; una planta usada como banco de pruebas para el desarrollo de controladores. Mediante el desarrollo de dicho agente, se busca probar que existe una alternativa al modelado matemático de plantas y sintonización de controladores.

Índice general

Agradecimientos.....	III
Resumen.....	IV
Índice general	V
Índice de Ilustraciones.....	IX
Índice de Tablas	XI
Glosario de Términos	XII
Capítulo 1 Introducción	1
1.1. Antecedentes	1
1.1.1. Sistemas de Control Convencionales.....	2
1.2. Descripción de la problemática.....	5
1.3. Justificación del proyecto	6
1.3.1. Justificación Académica	6
1.3.2. Justificación Técnica.....	6
1.4. Objetivos	6
1.4.1. Objetivo Principal	6
1.4.2. Objetivos Específicos.....	6
1.5. Alcances.....	7
1.6. Limites	7
Capítulo 2 Marco Teórico	8
2.1. Sistemas de Aprendizaje Profundo	8
2.1.1. Neuronas Artificiales	9
2.1.2. Redes Neuronales Artificiales Feedforward	10
2.1.3. Funciones de Activación.....	13
2.1.4. Entrenamiento de una Red Neuronal	16
2.2. Sistemas de Aprendizaje Reforzado Profundo	21
2.2.1. Componentes de un sistema de aprendizaje reforzado profundo.....	22
2.2.2. Interacción agente-entorno.....	23
2.2.3. Procesos de Decisión de Markov	24
2.2.4. Retorno.....	26
2.2.5. Política	27
2.2.6. Funciones de Valor	27

2.2.7. Ecuaciones de Bellman	28
2.2.8. Función Objetivo	28
2.2.9. Funciones de Valor Óptimas.....	29
2.2.10. Ecuaciones de Bellman Óptimas	29
2.2.11. Dilema Exploración-Explotación.....	30
2.2.12. Clasificación de los Algoritmos de Aprendizaje Reforzado Profundo.....	30
2.2.13. Gradientes Profundos de Política Determinística	31
2.2.14. Gradientes Profundos Gemelos y Atrasados de Política Determinística.....	35
2.3. Dinámica de un Péndulo Invertido	38
2.4. Aspectos de Software para la Implementación de Sistemas de Aprendizaje Reforzado Profundo.....	42
2.4.1. Requerimientos de Tiempo de ejecución de Redes Neuronales en un Lazo de Control	42
2.4.2. Análisis del tiempo de Ejecución de una Red Neuronal Feedforward	42
2.4.3. Aceleración de la Ejecución de las Redes Neuronales Mediante el Uso de GPUs....	43
Capítulo 3 Ingeniería del proyecto	45
3.1. Propuesta de la Arquitectura Para el Control Mediante Aprendizaje Reforzado Profundo del Péndulo Invertido.....	45
3.1.1. Visión General	45
3.1.2. Esquema del sistema	45
3.1.3. Agente.....	46
3.1.4. Clase agente	48
3.1.5. Entorno.....	49
3.1.6. Módulo de Exploración.....	51
3.1.7. Módulo de Métrica.....	51
3.1.8. Módulo Principal	52
3.1.9. Módulo de Demostración.....	53
3.1.10. Arquitectura Hardware del Sistema	53
3.2. Requerimientos de Hardware del Proyecto.....	54
3.2.1. Planta.....	54
3.2.2. Placa de Microcontrolador.....	55
3.2.3. Estación de Trabajo.....	55
3.3. Requerimientos de Software del Proyecto	56

3.3.1. Lenguaje de Programación	56
3.3.2. Librería de Aprendizaje Profundo	56
3.4. Elección de las herramientas hardware del proyecto.....	57
3.4.1. RoboBalance Store Linear Inverted Pendulum.....	57
3.4.2. Núcleo STM32F767ZI.....	60
3.4.3. Estación de Trabajo.....	63
3.5. Elección de las herramientas software del proyecto	63
3.5.1. Python	63
3.5.2. Pytorch	64
3.5.3. Numpy.....	67
3.5.4. PySerial	68
3.5.5. STM32CubeIDE	69
3.6. Implementación del sistema de control.....	70
3.6.1. Proceso de decisión Markov del Péndulo Invertido	70
3.6.2. Implementación del Entorno.....	73
3.6.3. Implementación del Agente	80
3.6.4. Implementación del Módulo de Exploración.....	88
3.6.5. Implementación del Módulo de Métrica.....	89
3.6.6. Implementación del Módulo Principal.....	90
3.6.7. Implementación del Módulo de Demostración.....	92
3.7. Pruebas del sistema.....	93
3.7.1. Entrenamiento del Agente.....	93
3.7.2. Pruebas de rendimiento.....	96
Capítulo 4 Análisis y discusión de los resultados	97
4.1. Experimentación y resultados	97
4.2. Análisis de desempeño.....	100
4.2.1. Columpiado Hacia Arriba (Swing Up)	100
4.2.2. Estabilización en la Posición Vertical.....	102
4.3. Análisis de Perturbaciones de Carga.....	104
4.4. Análisis de Perturbaciones Externas	106
Capítulo 5 Conclusiones y recomendaciones.....	109
5.1. Conclusiones	109

5.2. Recomendaciones	113
Bibliografía	114
Apéndice.....	120
A.1. Código del Subsistema de Medición y Control.....	120
A.2. Código del Módulo de Interfaz Agente-Entorno.....	125
A.3. Código del Módulo de Memoria de Experiencias.....	127
A.4. Código del Módulo de Redes Neuronales.....	128
A.5. Código de la Clase Agente	129
A.6. Código del Módulo de Exploración	132
A.7. Código del Módulo Principal	132
A.8. Código del Módulo de Demostración	134

Índice de Ilustraciones

Ilustración 1 Sistema de control realimentado	2
Ilustración 2 Juego de balanceo.....	3
Ilustración 3 Esquema general péndulo invertido en carro	3
Ilustración 4 Humanoide simulado controlado por agente.....	5
Ilustración 5 Paradigmas de inteligencia artificial	8
Ilustración 6 Funcionamiento de una neurona artificial	10
Ilustración 7 Ejemplo de arquitectura de red neuronal feedforward	11
Ilustración 8 Grafo computacional para la red neuronal ejemplo	13
Ilustración 9 Función de activación lineal.....	14
Ilustración 10 Función de activación sigmoide	14
Ilustración 11 Función de activación tangente hiperbólica	15
Ilustración 12 Función de activación ReLU.....	15
Ilustración 13 Efecto de la tasa de aprendizaje	18
Ilustración 14 Retropropagación representada en el grafo computacional	19
Ilustración 15 Interacción agente-entorno en la generación de una trayectoria	23
Ilustración 16 Esquema simplificado de un péndulo invertido	38
Ilustración 17 Análisis del centro de masa del péndulo invertido.....	39
Ilustración 18 Tiempo de ejecución de la red neuronal vs número de neuronas.....	43
Ilustración 19 Tiempo de ejecución de la red neuronal en GPU.....	44
Ilustración 20 Esquema de interacción modificado	46
Ilustración 21 Arquitectura del agente	46
Ilustración 22 Modulo de redes neuronales e instancias	48
Ilustración 23 Arquitectura del entorno.....	49
Ilustración 24 Modulo de interfaz agente-entorno	50
Ilustración 25 Subsistema de medición y control.....	51
Ilustración 26 Modulo principal	52
Ilustración 27 Modulo de demostración	53
Ilustración 28 Arquitectura hardware del sistema	54
Ilustración 29 Péndulo invertido elegido.....	58
Ilustración 30 Driver DMT-542T	59
Ilustración 31 Funcionamiento de un encoder incremental.....	59
Ilustración 32 Placa de desarrollo Nucleo STM32F767ZI.....	60
Ilustración 33 Funcionamiento simplificado de la interfaz serial con acceso directo a memoria	62
Ilustración 34 Funcionamiento de la interfaz de encoder.....	62
Ilustración 35 Funcionamiento de la interfaz de encoder en modos x2 y x4	63
Ilustración 36 STM32CUBEIDE con la herramienta STM32CUBEMX	70
Ilustración 37 Esquema del péndulo invertido adquirido.....	71
Ilustración 38 Formato de trama enviado por el subsistema de medición y control	76
Ilustración 39 Análisis del valor de cuenta_encoder con relación al ángulo del péndulo.....	79
Ilustración 40 Arquitectura de RedActor	84

Ilustración 41 Arquitectura de RedCritico	85
Ilustración 42 Comportamiento de la función de recompensa para los parámetros elegidos	94
Ilustración 43 Evolución de la recompensa a lo largo del proceso de entrenamiento.....	97
Ilustración 44 Evolución de la recompensa durante los primeros 32 episodios.....	98
Ilustración 45 Evolución de la recompensa durante los primeros 75 episodios.....	99
Ilustración 46 Evolución de la recompensa durante los últimos 200 episodios.....	99
Ilustración 47 Columpiado hacia arriba de la prueba 8.....	101
Ilustración 48 Evolución de las variables de estado a lo largo de la prueba 14	103
Ilustración 49 Trayectoria 13 con barra de perturbación de carga	106
Ilustración 50 Respuesta del sistema a perturbaciones externas	107

Índice de Tablas

Tabla 1 Plantas de péndulo invertido consideradas	57
Tabla 2 Placas de microcontrolador consideradas	60
Tabla 3 Placas de microcontrolador consideradas, especificaciones	61
Tabla 4 Estación de trabajo, especificaciones.....	63
Tabla 5 Fuente: Librerías de aprendizaje profundo, comparación.....	65
Tabla 6 Entornos de desarrollo para NUCLEO-F767ZI, comparación	69
Tabla 7 Periféricos del microcontrolador utilizados	74
Tabla 8 Comandos numéricos del subsistema de medición y control	74
Tabla 9 Parámetros de instanciación de la clase Entorno	77
Tabla 10 Variables inicializadas al instanciar la clase Entorno	78
Tabla 11 Método especiales de Entorno	80
Tabla 12 Parámetros de instanciación de la clase Memoria	81
Tabla 13 Variables y objetos inicializados al instanciar la clase Memoria	81
Tabla 14 Argumentos de entrada del método almacenar_paso.....	81
Tabla 15 Argumento de entrada del método tomar_muestras	82
Tabla 16 Métodos utilitarios de la clase Memoria	83
Tabla 17 Parámetros de instanciación de la clase RedActor	84
Tabla 18 Arquitectura de RedActor	84
Tabla 19 Arquitectura de RedCritico	85
Tabla 20 Métodos de guardado y cargado de las redes neuronales	85
Tabla 21 Parámetros de instanciación de la clase Agente	86
Tabla 22 Variables y objetos inicializados al instanciar la clase Agente.....	87
Tabla 23 Métodos de la clase Agente	88
Tabla 24 Variables y objetos inicializados al instanciar la clase Exploracion.....	89
Tabla 25 Métodos de la clase Exploracion	89
Tabla 26 Lista inicializada al instanciar la clase Metrica	90
Tabla 27 Métodos de la clase Metrica	90
Tabla 28 Parámetros del módulo principal	91
Tabla 29 Parámetros de la función de recompensa usados para el entrenamiento	93
Tabla 30 Parámetros para las arquitecturas de RedActor y RedCritico.....	95
Tabla 31 Fuente: Parámetros de instanciación del agente usados para el entrenamiento.....	95
Tabla 32 Parámetros del módulo principal usados para el entrenamiento.....	96
Tabla 33 Tiempo mínimo de balanceo hacia arriba.....	100
Tabla 34 Comparación de las dos barras usadas.....	104
Tabla 35 Tiempo mínimo de balanceo hacia arriba con barra para pruebas de perturbación....	105

Glosario de Términos

Agente

Componente de todo sistema de aprendizaje reforzado profundo, es un ente computacional que busca lograr un determinado objetivo dentro un entorno, eligiendo acciones para interactuar con el entorno y mejorar su propio rendimiento progresivamente en base a datos obtenidos de dichas interacciones.

Algoritmo de estimación adaptiva de momento (Adaptive Moment Estimation, ADAM)

Algoritmo propuesto como una mejora del algoritmo de descenso de gradiente estocástico, el cual usa estimaciones de momentos de los gradientes para acelerar la convergencia en el entrenamiento de redes neuronales.

Aprendizaje Automático (Machine Learning, ML)

Campo de la inteligencia artificial, dentro el cual se desarrollan sistemas que sean capaces de aprender las reglas y funciones que mapean determinadas entradas a sus correspondientes salidas mediante el uso de datos, sin que sean programadas de manera explícita.

Aprendizaje Profundo (Deep Learning, DL)

Campo del aprendizaje automático que usa redes neuronales artificiales profundas para el aprendizaje de las reglas y funciones a partir de los datos.

Aprendizaje Reforzado (Reinforcement Learning, RL)

Campo del aprendizaje automático, dentro el cual se desarrollan agentes, los cuales aprenden a tomar decisiones en base a de datos de interacciones.

Aprendizaje Reforzado Profundo (Deep Reinforcement Learning, RL)

Campo del aprendizaje automático, el cual aprovecha las redes neuronales y técnicas del aprendizaje profundo para el desarrollo de agentes.

Descenso de Gradiente

Algoritmo de optimización básico usado para el entrenamiento de redes neuronales.

Descenso de Gradiente Estocástico

Algoritmo propuesto como una mejora del algoritmo de descenso de gradiente. Usa lotes de datos tomados de manera aleatoria para acelerar la convergencia en el entrenamiento de redes neuronales.

Entorno

Componente de todo sistema de aprendizaje reforzado profundo. Es un ente simulado o físico, dentro el cual el agente decide cuales acciones ejecutar, buscando influenciarlo de manera que se pueda lograr un determinado objetivo.

Gradientes Profundos de Política Determinística (Deep Deterministic Policy Gradients, DDPG)

Algoritmo de aprendizaje reforzado profundo usado para entrenar agentes mediante la aproximación de una política optima y una función de valor de estado-acción de manera simultánea.

Gradientes Profundos Gemelos y Atrasados de Política Determinística (Twin Delayed Deep Deterministic Policy Gradients, TD3)

Algoritmo de aprendizaje reforzado profundo propuesto como una mejora del algoritmo de Gradientes Profundos de Política Determinística. Emplea técnicas que evitan la sobreestimación de la función de valor de estado-acción para mejorar la convergencia hacia una política óptima.

Política

Componente principal del agente, es un ente computacional encargado de tomar decisiones acerca de cuáles acciones ejecutar dentro el entorno para lograr un determinado objetivo.

Retropropagación

Algoritmo usado para calcular gradientes en el entrenamiento de redes neuronales.

Neurona Artificial

Ente computacional inspirado por las neuronas en la naturaleza, implementado mediante una función no lineal de una o más entradas. Es el componente más básico de una red neuronal.

Red Neuronal Artificial (Red Neuronal)

Arreglo por capas de neuronas artificiales, el cual posee la propiedad de ser un aproximador no-lineal universal de funciones. Posee una capa de entrada, una o más capas escondidas y una capa de salida.

Red Neuronal Artificial Profunda (Red Neuronal Profunda)

Red Neuronal que posee dos o más capas escondidas.

Proceso de Decisión de Markov (Markov Decision Process, MDP)

Proceso que enmarca y define de manera formal la base matemática de las interacciones entre el agente y el entorno dentro todo sistema de aprendizaje reforzado o aprendizaje reforzado profundo.

Capítulo 1

Introducción

1.1. Antecedentes

En el marco de la línea de investigación en aprendizaje reforzado profundo impulsada por la reciente adopción masiva e investigación en sistemas de aprendizaje profundo, tanto la academia como la industria buscan el desarrollo de sistemas que, con un mayor grado de autonomía, sean capaces de decidir y ejecutar la mejor acción posible dentro de un determinado entorno, aprendiendo a tomar dicha decisión a partir de experiencias [1].

Actualmente se ha desarrollado y explorado tanto en la academia como en la industria, variadas y diferentes técnicas de inteligencia artificial, que aprovechan el desarrollo acelerado de las redes neuronales durante la última década, dentro el campo denominado aprendizaje profundo, el cual fue impulsado por una mayor disponibilidad de sistemas para la obtención de datos, así como las mayores capacidades computacionales, tanto en cuanto a velocidad de procesamiento como en cuanto a mayores capacidades de almacenamiento de datos. Esta unión sentó una base sobre la cual desarrolló el campo denominado aprendizaje reforzado profundo, el cual busca crear sistemas de inteligencia artificial capaces de aprender y modificar su comportamiento a partir de experiencias propias, con el fin de lograr un determinado propósito dentro de un entorno [2].

La idea a partir de la cual se desarrolló el presente proyecto de grado, es de aprovechar dichas capacidades computacionales y el desarrollo del aprendizaje reforzado profundo, como una alternativa a los sistemas de control tradicionales, los cuales están basados en la obtención de modelos que representan al sistema a controlar y la sintonización de controladores tanto en el área analógica como digital. Mediante el presente trabajo, se demuestra que es posible aprovechar las técnicas de inteligencia artificial desarrolladas dentro el aprendizaje reforzado profundo, para lograr el control de un sistema real utilizado como prueba de referencia de sistemas de control, sin la necesidad de adherirse a las líneas de diseño de controladores establecidas actualmente.

Los primeros sistemas de control realimentado ideados por el hombre se remontan a mediados del siglo 3 antes de cristo [3]. Sin embargo, la base teórica de los sistemas de control actuales no se desarrolló hasta los años 1900, con ejemplos claros como el trabajo de Minorsky en 1922 acerca de la estabilidad de sistemas a partir de sus ecuaciones diferenciales, el trabajo de Nyquist durante la década de los 1930 que introdujo el análisis de la estabilidad de sistemas a lazo cerrado y amplificadores realimentados [4] y el trabajo de Bode para el análisis en el dominio de la frecuencia [5].

Sobre esta base matemática, la llegada de la segunda guerra mundial produjo un interés más amplio en el desarrollo de los sistemas de control, impulsado por la necesidad bélica de sistemas de posicionamiento de antenas de radar y de control de naves entre otros. Se desarrollaron sistemas de control realimentado para cumplir las tareas anteriormente mencionadas utilizando principalmente técnicas de diseño en el dominio de la frecuencia [6].

A partir del interés por los sistemas de control generado durante la segunda guerra mundial, se dio un interés por la investigación y desarrollo de técnicas de control más avanzadas que sigue hasta el día de hoy.

1.1.1. Sistemas de Control Convencionales

Convencionalmente, se define ampliamente como sistema de control, a todo sistema que busca lograr una respuesta deseada de un determinado proceso [7].

1.1.1.1. Sistemas de control realimentado

Los sistemas de control realimentados son el conjunto de sistemas de control que se encuentra en más amplio uso actualmente, siendo objeto de estudio y desarrollo constante. Esto se debe a que, dado un diseño correcto, los sistemas de control realimentados poseen generalmente la capacidad de seguir una referencia y estabilizar sistemas inestables [8]. Todo esquema de control realimentado posee los siguientes elementos:

- Un proceso determinado (denominado planta) del cual se desea controlar una o más salidas.
- Uno o más elementos de medición que generan señales proporcionales a las salidas del proceso.
- Un elemento de comparación, que genera una señal de error entre la salida del proceso y una referencia de salida deseada.
- Un controlador que genera una señal de control, utilizada como entrada a la planta.

En adición, es posible que el sistema pueda ser afectado por perturbaciones en diferentes ubicaciones a lo largo del lazo de control realimentado. El esquema de sistema de control realimentado descrito se muestra en la ilustración 1.

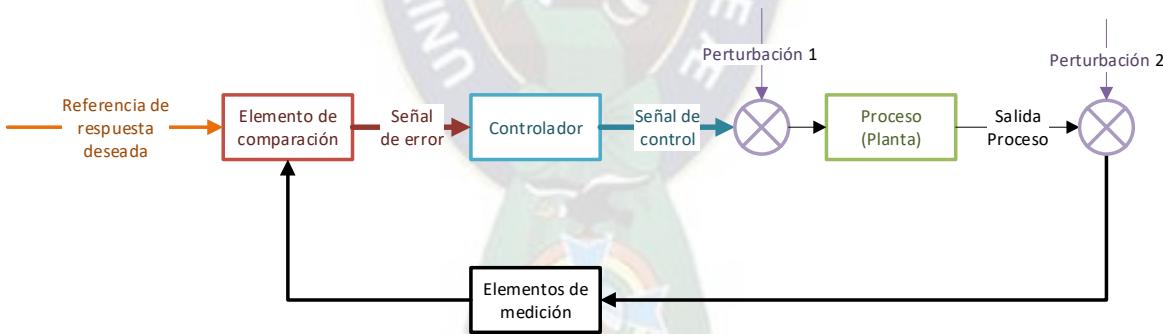


Ilustración 1 Sistema de control realimentado. Fuente: Elaboración propia.

1.1.1.2. Flujo de trabajo general para el diseño de un sistema de control realimentado

Sin considerar el tipo de controlador que se desee utilizar sobre un determinado proceso (por ejemplo, PID, de realimentación de estado o cualquier otro), existe un flujo de trabajo que implica un número básico de pasos a seguir para diseñar el controlador [9]. Dicho flujo se resume a continuación.

- **Modelado de la planta:** Previamente a realizar el diseño del controlador, el diseñador debe obtener un modelo matemático de la planta, el cual debe ser lo suficientemente exacto con relación a la planta real. Esto se debe a que sobre este modelo se basa el resto del proceso de diseño del controlador.

- **Elección del tipo de controlador y diseño:** De acuerdo con las restricciones de desempeño requeridas y las características de la planta a controlarse, el diseñador realiza la elección de un tipo de controlador. Seguidamente utiliza las herramientas matemáticas propias del controlador elegido para llevar su diseño a cabo.
- **Simulación del controlador:** Se realiza una simulación del desempeño del controlador diseñado sobre el modelo matemático de la planta; de no obtenerse el desempeño deseado, se debe regresar al paso anterior.
- **Implementación del controlador:** Se implementa el controlador físicamente, de manera que pueda interactuar con la planta real.
- **Pruebas del sistema de control:** Se pone a prueba el funcionamiento del controlador sobre la planta real, midiendo su desempeño; de no encontrarse dentro los parámetros de rendimiento deseados, se debe regresar al paso de elección del tipo de controlador o llevar a cabo el diseño nuevamente.

1.1.1.3. Péndulo invertido en un carro

Se define como péndulo invertido, todo sistema en el cual el centro de masa de una barra rígida se encuentra por encima de su punto de apoyo [10]. Un ejemplo simple de un péndulo invertido es el clásico juego infantil de balanceo de una escoba en una mano, el cual se muestra en la ilustración 2.

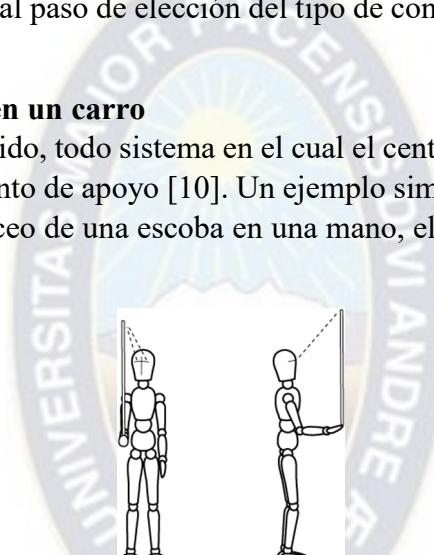


Ilustración 2 Juego de balanceo. Fuente: [11].

El péndulo invertido en un carro por lo general, consiste en una barra conectada a un sensor de posición angular montado en un carro, dicho carro es accionado por un motor. Un esquema básico del péndulo invertido en un carro se observa en la ilustración 3.

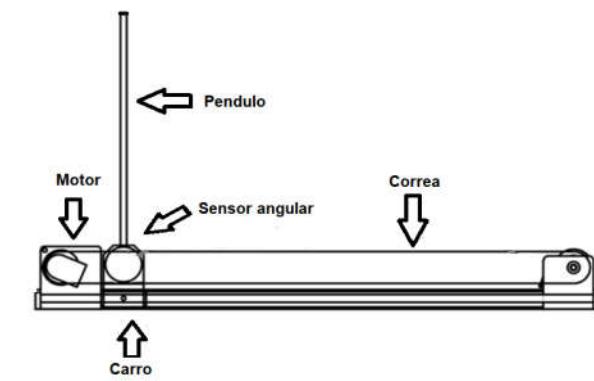


Ilustración 3 Esquema general péndulo invertido en carro. Fuente: Elaboración propia.

El péndulo invertido en un carro se ha utilizado durante las últimas 5 décadas como una prueba de referencia sobre el rendimiento de los diferentes tipos de controladores, ya que se trata de un sistema sub-actuado, no lineal e inestable, por lo cual propone un desafío para los diferentes controladores que puedan ser usados sobre él, pero a su vez, es un sistema relativamente sencillo de implementar [12].

1.1.1.4. Sistemas de Aprendizaje Reforzado Profundo

Los sistemas de aprendizaje reforzado profundo, son sistemas de inteligencia artificial que buscan aprender computacionalmente a partir de interacciones con un entorno de manera que, dado un estado del entorno en un instante de tiempo, el sistema de aprendizaje reforzado profundo elija una acción correcta a realizarse dentro el entorno, para influenciarlo de una manera que se pueda cumplir un objetivo determinado dentro de él [13].

Los sistemas de aprendizaje reforzado se idearon desde la década de los años 1960 [14], sin embargo, estos obtuvieron resultados favorables dentro una lista de tareas limitada. No fue hasta la década del 2010 que se renovó interés en este campo. Gracias al acelerado desarrollo de técnicas de aprendizaje profundo, fue posible aplicarlas en conjunto con el conocimiento de aprendizaje reforzado desarrollado en décadas anteriores, dando paso a un nuevo campo que toma conocimiento y técnicas de los dos anteriores, denominado aprendizaje reforzado profundo, el cual es un extenso y activo campo de investigación y desarrollo hoy en día.

Dado el reciente interés en el aprendizaje reforzado profundo, se ha obtenido logros notables a partir del desarrollo y aplicación de sus numerosas técnicas, mayormente dentro entornos simulados y de juegos. Ejemplos destacados de dichos logros se resumen a continuación:

- Se desarrolló agentes basados en aprendizaje reforzado profundo, los cuales fueron capaces de aprender a ganar en juegos clásicos como Pong y Space Invaders, usando imágenes del juego como entrada [15].
- Se realizaron agentes basados en aprendizaje reforzado profundo capaces de aprender estrategias óptimas a partir de experiencias para ganar en el juego de estrategia Starcraft II, superando a jugadores humanos campeones del juego a nivel mundial [16].
- Un agente de aprendizaje reforzado profundo fue entrenado exitosamente sobre un simulador de automóviles, obteniendo un sistema capaz de realizar la tarea de conducción autónoma de un automóvil dentro del simulador [17].
- Técnicas de aprendizaje reforzado profundo fueron utilizadas exitosamente para el aprendizaje del control de un brazo robot manipulador de 6 grados de libertad, con el fin realizar tareas de posicionamiento de este y seguimiento de un objeto [18].
- Un agente basado en aprendizaje profundo reforzado fue capaz de aprender a partir de experiencias, a mantener un humanoide de pie y moverlo para avanzar adelante dentro un entorno simulado con obstáculos, aplicando valores correctos de torque sobre más de una decena de articulaciones al mismo tiempo, como se muestra en la ilustración 4 [19].

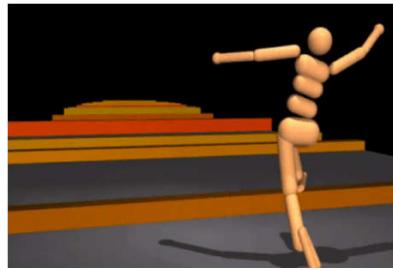


Ilustración 4 Humanoide simulado controlado por agente. Fuente: [20].

1.2. Descripción de la problemática

Actualmente, se deben seguir los pasos del flujo de diseño de un sistema de control realimentado descritos en la sección 1.1.1.2, muchas veces realizando varias iteraciones de dicho flujo hasta obtener un controlador con rendimiento satisfactorio, lo cual se denomina realizar la sintonización del controlador.

Además, dentro dicho flujo de diseño de un sistema de control, especialmente para plantas no lineales, por lo general existe un compromiso entre complejidad de diseño del controlador y rendimiento del mismo. Por un lado, si se desea utilizar controladores más simples para los cuales existen herramientas de diseño rápido, entonces se debe linealizar el modelo del sistema, lo cual implica una simplificación que introduce un error en el modelo matemático; esto tiene el potencial de afectar el desempeño del controlador.

Por otro lado, si el sistema requiere un rendimiento mucho más alto o los controladores basados en modelos linealizados fallan, se debe utilizar controladores más complejos que usan campos de conocimiento más específicos de la teoría de control, los cuales requieren de herramientas matemáticas más complejas.

En resumen, para el diseño de un controlador para un proceso en específico se requiere de un proceso iterativo, el cual parte del conocimiento para derivar un modelo matemático de la planta con tanta complejidad como sea necesaria. Seguidamente, se requiere del diseño de un controlador con el uso de conocimientos matemáticos específicos y a menudo complejos.

Alternativamente, se puede aprovechar el aprendizaje reforzado profundo para realizar un proceso iterativo pero automático, el cual busca eliminar la necesidad de un modelado matemático del sistema; este busca optimizar directamente las acciones tomadas por el controlador sobre la planta, de modo que, dada una entrada de medición del estado del sistema, se genere un comando de control adecuado. Esto proporciona la posibilidad de evitar los procedimientos matemáticos de diseño mencionados anteriormente, proveyendo una alternativa que puede encontrarse al alcance de aquellos con un conocimiento más intuitivo y menos específico en cuanto a técnicas de control.

En adición, se observa que actualmente los sistemas de aprendizaje reforzado profundo han sido exitosos en la realización de variadas tareas de inteligencia artificial, dentro entornos mayormente simulados. De los resultados obtenidos dentro dichos entornos, es posible considerar que el campo posee una base teórica y un conjunto de técnicas que pueden ser aplicadas dentro de entornos reales.

Sin embargo, teniendo en cuenta que el campo es relativamente joven, en adición al hecho de que sus técnicas fueron orientadas inicialmente a la toma de decisiones dentro el campo de la inteligencia artificial, ejemplos del uso de sistemas de aprendizaje reforzado profundo en interacción con sistemas en el mundo real no son comunes, en especial en cuanto a su aplicación para el control dentro entornos reales.

1.3. Justificación del proyecto

1.3.1. Justificación Académica

El proyecto de grado se justifica desde el punto de vista académico, puesto que aplica técnicas y procedimientos de aprendizaje reforzado profundo; un campo emergente, en constante desarrollo y a la vanguardia para la implementación de un controlador aplicable al péndulo invertido; un sistema altamente no lineal e inestable, el cual es utilizado ampliamente como punto de referencia para la prueba de estrategias de control desde hace décadas.

Por medio del proyecto de grado, se buscó demostrar que es posible aplicar técnicas de aprendizaje reforzado profundo al control de sistemas reales. Mediante la aplicación de dichas técnicas, se muestra la existencia de una alternativa al flujo de diseño de sistemas de control establecido por la academia descrito en la sección 1.1.1.2, generando un controlador que no requiere el modelado del sistema, ni de diseño de un controlador mediante procedimientos del control clásico o moderno.

1.3.2. Justificación Técnica

El proyecto de grado se justifica desde el punto de vista técnico, ya que a través de este estudio se explora la utilización de técnicas de aprendizaje reforzado profundo disponibles en la literatura científica, las cuales hacen posible explorar una alternativa diferente al enfoque tradicional de diseño e implementación de sistemas de control, realizando un sistema de control que pretende simplificar el proceso de implementación de un controlador para un determinado proceso a través de técnicas modernas desarrolladas dentro del campo de la inteligencia artificial.

Dicho sistema además puede además constituirse en una plataforma base sobre la cual se puede desarrollar futuros experimentos para la investigación de sistemas de control basados en aprendizaje reforzado profundo.

1.4. Objetivos

1.4.1. Objetivo Principal

El objetivo principal del proyecto de grado se resume en:

“Controlar un péndulo invertido utilizando aprendizaje reforzado profundo, que permita iterativamente mejorar el desempeño del sistema de control en base a sus propias experiencias”

1.4.2. Objetivos Específicos

Los objetivos específicos desarrollados para el cumplimiento del objetivo principal, se detallan a continuación.

- Proponer una arquitectura de software básica para el aprendizaje reforzado profundo aplicado al control del péndulo invertido.
- Realizar una elección justificada de las herramientas hardware y software a utilizarse en el proyecto.
- Proponer las escalas de medición del estado del sistema, de manera que dicho estado pueda ser usado para expresar el objetivo de control del agente.
- Proponer un criterio de desempeño (denominado función de recompensa) que permita al agente evaluar y mejorar su propio rendimiento.
- Proponer un agente basado en la literatura de aprendizaje reforzado profundo que, a partir de las mediciones del estado, genere la acción adecuada para posicionamiento del péndulo en su posición vertical.
- Mostrar y analizar los resultados del sistema de aprendizaje reforzado profundo aplicado al control del péndulo invertido.

1.5. Alcances

Los alcances sobre los cuales se desarrolló el proyecto de grado son listados a continuación.

- Se realizará el control de la posición angular de un péndulo invertido adquirido en el mercado mediante aprendizaje reforzado profundo.
- Se elegirá y utilizará una de las diversas técnicas proveniente de la literatura del aprendizaje reforzado profundo, para llevar a cabo el control del péndulo invertido y lograr de forma automática su posición vertical.
- Se realizará la implementación del software del proyecto mediante el uso de la amplia selección de librerías software probadas en la academia para aprendizaje profundo.

1.6. Límites

Finalmente, los límites fijados para el desarrollo del proyecto se muestran a continuación.

- El proyecto se limitará a utilizar un péndulo invertido disponible en el mercado, omitiendo el diseño e implementación desde cero de un péndulo invertido.
- El proyecto solamente se limitará a estabilizar el péndulo invertido en la posición vertical independientemente de la posición del carro.
- El proyecto solamente se limitará a utilizar una técnica ya desarrollada entre las técnicas existentes del aprendizaje profundo reforzado.
- El proyecto se limitará a utilizar las librerías disponibles para aprendizaje profundo, omitiendo realizar implementaciones propias desde cero de algoritmos y componentes de aprendizaje profundo.

Capítulo 2

Marco Teórico

Dentro este capítulo, se explica la teoría y técnicas del aprendizaje profundo aplicables en el marco del desarrollo de un controlador basado en aprendizaje reforzado profundo para el péndulo invertido. Seguidamente, se desarrolla la teoría de aprendizaje reforzado profundo y las técnicas usadas para el desarrollo del proyecto de grado. Luego, se justifica mediante una base teórica las razones por las cuales el péndulo invertido sobre un carro es una planta que propone un desafío para todo sistema de control. Finalmente, se analiza algunos aspectos de software que pueden influenciar la implementación de un controlador basado en aprendizaje reforzado profundo.

2.1. Sistemas de Aprendizaje Profundo

Existen diversas definiciones que explican qué son los sistemas de inteligencia artificial, una que se considerada apropiada al presente trabajo menciona: “Todo sistema de inteligencia artificial busca automatizar tareas que normalmente serían llevadas a cabo por un ser inteligente” [21].

Para lograr automatizar una tarea definida, los sistemas de inteligencia artificial “clásicos” desarrollados hasta finales de la década de 1980 requieren de reglas que deben ser diseñadas y programadas cuidadosamente de una manera explícita por una persona, estas reglas deben mapear los datos de entrada del sistema a salidas correctas. Este tipo de inteligencia artificial se denomina inteligencia artificial simbólica.

Los sistemas de aprendizaje automático (ML; Machine Learning por su sigla en inglés) desarrollados a partir de la década de 1990 cambian ese paradigma; se utiliza una gran cantidad de ejemplos de datos de entrada, junto a sus correspondientes salidas correctas, para aprender las reglas que transforman las entradas a las salidas correctas. El aprendizaje de dichas reglas se denomina entrenamiento del sistema de aprendizaje automático [22]. El funcionamiento de los dos paradigmas mencionados anteriormente se representa en la ilustración 5.



Ilustración 5 Paradigmas de inteligencia artificial. Fuente: Elaboración propia.

Luego que el sistema de ML ha sido entrenado, se lo puede utilizar para recuperar aproximaciones a las salidas correctas a partir de entradas; esto se denomina realizar inferencia usando el sistema de aprendizaje automático previamente entrenado [23].

El proceso de entrenamiento usando datos de ejemplo y las técnicas relacionadas para llevarlo a cabo, se encuentran dentro un campo que se denomina aprendizaje supervisado. El nombre

proviene del hecho que los datos ejemplo usados para entrenar al sistema están etiquetados con salidas correctas, las cuales proveen una manera de supervisar automáticamente el proceso de entrenamiento [24]. El aprendizaje supervisado es uno de los tres principales campos dentro el aprendizaje automático¹.

Los sistemas de aprendizaje profundo (DL, Deep Learning por su sigla en inglés), son un subconjunto de técnicas y algoritmos que pertenecen a ML, los cuales usan redes neuronales artificiales profundas para aprender una aproximación de las funciones que transforman los datos de entrada en salidas correctas.

Los sistemas de DL poseen un cuerpo de teoría y técnicas extenso que se encuentra en constante desarrollo, dentro esta sección se abarca una parte del cuerpo de conocimiento de DL, la cual permitió realizar un sistema de aprendizaje reforzado profundo para el control del péndulo invertido.

2.1.1. Neuronas Artificiales

Las redes neuronales artificiales son sistemas computacionales inspirados en la manera en la que las neuronas de un sistema nervioso, trabajan en conjunto para realizar tareas complejas. Tomando inspiración en la naturaleza humana/animal, la unidad básica de procesamiento es la neurona, en este caso denominada neurona artificial, la cual se encuentra encargada de realizar los cálculos en los sistemas de aprendizaje profundo.

Una neurona artificial con un número n de entradas lleva a cabo un funcionamiento descrito mediante las siguientes operaciones:

- Primeramente, multiplica cada entrada x_1, x_2, \dots, x_n por su respectivo parámetro de peso w_1, w_2, \dots, w_n , realizando una suma ponderada de sus entradas. A continuación, añade a la suma ponderada un escalar denominado de polarización b ; es decir realiza lo que se conoce como una operación lineal sobre la entrada, como se representa en la ecuación (2.1.1).

$$z(x_1, x_2, \dots, x_n) = x_1w_1 + x_2w_2 + \dots + x_nw_n + b \quad (2.1.1)$$

Es útil expresar la operación lineal que se lleva a cabo dentro una neurona como una multiplicación entre dos vectores; el vector de entradas $X = [x_1, x_2, \dots, x_n]$ y el vector de pesos $W = [w_1, w_2, \dots, w_n]^T$, como se muestra en la siguiente ecuación.

$$z(X) = X * W + b \quad (2.1.2)$$

- Seguidamente, el resultado de $z(X)$ es usado como argumento de una función no lineal $g(z)$, la cual se denomina función de activación, obteniéndose la salida o activación a de la neurona, según la ecuación (2.1.3).

$$a = g(z) \quad (2.1.3)$$

La ilustración 6 muestra de una manera gráfica el funcionamiento de una neurona artificial, dentro la cual se realizan las operaciones descritas en el listado anterior.

¹ Los otros dos grandes campos dentro el aprendizaje automático son el aprendizaje no supervisado y el aprendizaje reforzado.

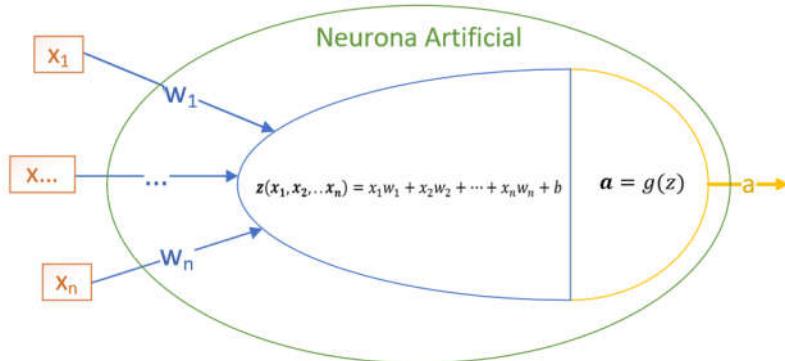


Ilustración 6 Funcionamiento de una neurona artificial. Fuente: Elaboración propia.

2.1.2. Redes Neuronales Artificiales Feedforward

Una red neuronal por lo general se encuentra conformada por varias neuronas artificiales, las cuales se encuentran organizadas por capas. Si dentro dicha organización, las salidas de las neuronas de una capa anterior se encuentran dirigidas solamente hacia una capa posterior sin ningún tipo de retroalimentación, entonces la red neuronal se denomina red neuronal feedforward [25].

Además, si dentro una red neuronal la salida de cada neurona de una capa se encuentra conectada a las entradas de todas las neuronas de la siguiente capa, entonces la conexión entre capas ocasiona que las capas se denominen capas densamente conectadas o capas completamente conectadas [26].

Las redes neuronales feedforward con capas densamente conectadas son usadas como aproximadores de funciones no lineales y constituyen la base fundamental de los sistemas de aprendizaje profundo, ya que son capaces de aproximar cualquier función, siempre que la función de activación en sus neuronas sea una función no-polinomial [27].

Toda red neuronal feedforward debe poseer cada uno de los tres siguientes tipos de capas.

- **Capa de Entrada:** La capa de entrada colecta y organiza las entradas de la red neuronal. Por lo general, no realiza ninguna operación sobre dichas entradas y por lo tanto no posee neuronas. El tamaño de la capa de entrada denominado $n^{[l]}$, es a menudo igual al número de componentes que posee el vector de datos de entrada a la red neuronal.
- **Capa(s) escondidas:** Las capas escondidas realizan operaciones sobre sus entradas para transformarlas utilizando neuronas artificiales. El número de capas escondidas de una red neuronal, denominado L , es variable y a criterio de su diseñador; en diversas aplicaciones dicho número puede variar desde una capa escondida hasta cientos o incluso miles. De igual manera, el tamaño de cada capa, denominado $n^{[k]}$, representa el número de neuronas que la conforman y es otra variable bajo control del diseñador de la red neuronal.
- **Capa de salida:** La capa de salida realiza una última transformación que finaliza el mapeo de datos de entrada a la salida, concentrándolo a las dimensiones correctas y escalándolo mediante

el uso de neuronas artificiales. Su tamaño, denominado $n^{[o]}$, es por lo general igual al número de componentes que posee el vector de datos de salida.

El número total de capas de una red neuronal, denominado K , considera solamente el número de capas escondidas y la capa de salida. Por lo tanto, se cumple que $K = L + 1$.

El tamaño de la capa de entrada $n^{[i]}$, el número total de capas K y los tamaños de cada una de las capas $n^{[k]}$, junto a la elección de la función de activación utilizada en las neuronas de cada capa, son parámetros de diseño de una red neuronal, que el diseñador debe considerar cuidadosamente; dichos parámetros definen la arquitectura de la red neuronal [28].

La ilustración 7 y su consecuente explicación muestran una arquitectura de red neuronal feedforward con capas densamente conectadas. Provee un ejemplo de los conceptos descritos anteriormente.

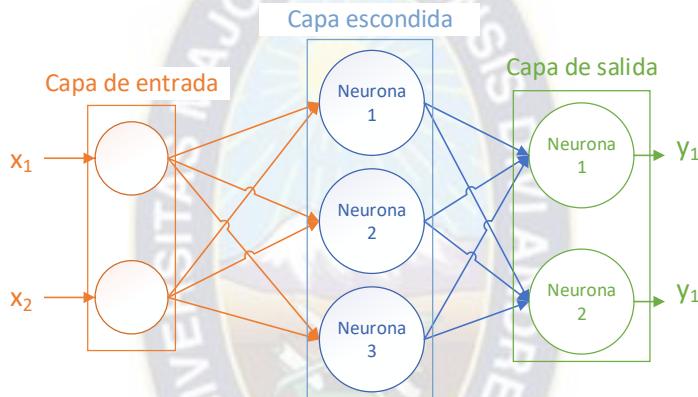


Ilustración 7 Ejemplo de arquitectura de red neuronal feedforward. Fuente: Elaboración propia.

- La red neuronal mostrada posee una capa de entrada de tamaño $n^{[i]} = 2$, esto implica que los datos de entrada son vectores de dimensión 2 de la forma $\mathbf{x} = [x_1 \ x_2]$.
- El número de capas escondidas de la red neuronal es $L = 1$; la única capa escondida es de tamaño $n^{[1]} = 3$, por lo tanto, posee 3 neuronas.
- La red neuronal posee una capa de salida es de tamaño $n^{[o]} = n^{[2]} = 2$. Por lo tanto, posee 2 neuronas. Esto implica que los datos de salida son vectores de dimensión 2 de la forma $\mathbf{y} = [y_1 \ y_2]$.
- La red neuronal posee un número total de capas $K = L + 1 = 2$.

La arquitectura de red neuronal descrita en el ejemplo, es un diseño capaz de aproximar una función \mathbf{h} que mapea vectores de entrada de dimensión 2 a vectores de salida de dimensión 2, es decir una función $\mathbf{h}: \mathbb{R}^2 \rightarrow \mathbb{R}^2$. Sin embargo, modificando la arquitectura de la red neuronal de acuerdo con las dimensiones de entrada y salida que sean necesarias, es posible aproximar funciones capaces de mapear vectores de un número arbitrario de dimensiones \mathbf{u} a vectores de otro número arbitrario de dimensiones \mathbf{v} , approximando una función $\mathbf{h}: \mathbb{R}^u \rightarrow \mathbb{R}^v$.

Una red neuronal con una arquitectura que posee un número de capas escondidas mayor o igual a dos se considera una red neuronal profunda. El uso extenso de redes neuronales profundas y el

impacto que tuvo en los resultados obtenidos usándolas, hizo que el campo se denomine aprendizaje profundo [29].

Cada neurona en cada capa de una red neuronal feedforward realiza las operaciones descritas en la sección 2.1.1. Por lo tanto, posee sus propios parámetros de peso y polarización. Es más sencillo representar las operaciones realizadas dentro cada capa como operaciones matriciales en vez de tomar en cuenta cada neurona en específico. A continuación, se introduce y explica la notación usada para representar dichas operaciones.

- Los datos de entrada a la capa numero k de la red neuronal son iguales a los datos de salida de la capa $k - 1$ y se encuentran representados por la matriz denominada $A^{[k-1]}$.
- El parámetro denominado $w_{ij}^{[k]}$, es el peso que afecta la conexión entrante número j a la neurona número i que pertenece capa numero k de la red neuronal. Este peso se encuentra dentro el vector columna de pesos de su correspondiente neurona, denominado $W_i^{[k]}$. Se representa todos los parámetros de peso de las neuronas de una capa dentro una matriz denominada $W^{[k]}$, la cual es un arreglo por columnas de los vectores de pesos $W_i^{[k]}$ de cada neurona en la capa, como se muestra en la ecuación (2.1.4).

$$W^{[k]} = \begin{bmatrix} W_1^{[k]} & W_2^{[k]} & \dots & W_{n^{[k]}}^{[k]} \end{bmatrix} \quad (2.1.4)$$

El conjunto de todas las matrices $W^{[k]}$ de todas las capas pertenecientes a una red neuronal se denomina simplemente como W .

- El parámetro de polarización denominado $b_i^{[k]}$, completa la operación lineal que se lleva a cabo dentro la neurona número i que pertenece capa número k de la red neuronal. Es también útil representar los parámetros de polarización de todas las neuronas dentro la capa k como un vector, el cual es denominado $B^{[k]}$ y se muestra a continuación.

$$B^{[k]} = \begin{bmatrix} b_1^{[k]} & b_2^{[k]} & \dots & b_{n^{[k]}}^{[k]} \end{bmatrix}^T \quad (2.1.5)$$

El conjunto de todos los vectores $B^{[k]}$ de todas las capas pertenecientes a una red neuronal se refiere simplemente como B .

- La tupla denominada φ de todos los parámetros de peso y polarización de todas las capas de la red neuronal es de la forma $\varphi = (W, B)$, se la conoce simplemente como los parámetros de la red neuronal.
- La matriz denominada $Z^{[k]}$ contiene los resultados de la operación lineal realizada por cada una de las neuronas de la capa numero k . Se obtiene mediante la multiplicación de matrices mostrada en la ecuación (2.1.6).

$$Z^{[k]} = A^{[k-1]}W^{[k]} + B^{[k]} \quad (2.1.6)$$

- La función de activación denominada $g^{[k]}$, se encuentra definida para todas las neuronas de la capa k y es aplicada elemento por elemento a la matriz $Z^{[k]}$ para obtener la salida de la capa, la cual es denominada $A^{[k]}$ y se muestra a continuación.

$$A^{[k]} = g^{[k]}(Z^{[k]}) \quad (2.1.7)$$

- La salida final de la red neuronal denominada \hat{y} , es representada por $\hat{y} = A^{[K]}$, se la denomina la predicción de la red neuronal y es obtenida directamente de la capa de salida.

Mientras que la ilustración 7 es útil para explicar la estructura de una red neuronal feedforward básica, una manera más apropiada de ilustrar las operaciones que se llevan a cabo dentro una red neuronal es mediante un grafo computacional. La ilustración 8 muestra el grafo computacional equivalente a la red neuronal de la ilustración 7.

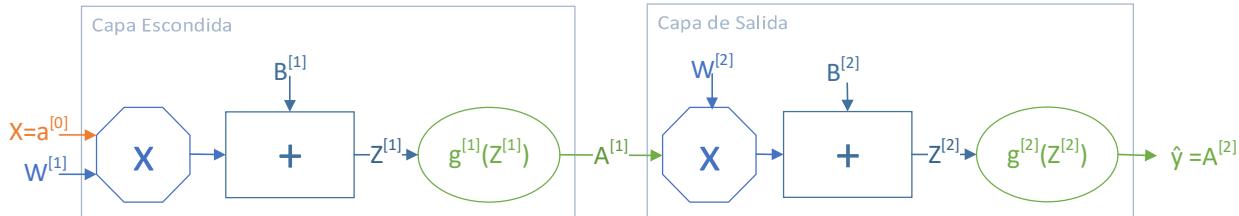


Ilustración 8 Grafo computacional para la red neuronal ejemplo. Fuente: Elaboración propia.

2.1.3. Funciones de Activación

Uno de los aspectos más importantes a la hora de diseñar una arquitectura de red neuronal, es la elección de las funciones de activación que serán usadas en sus capas.

Es posible utilizar cualquier función matemática como función de activación para las neuronas de una capa. Sin embargo, como se mencionó anteriormente, las funciones de activación deben ser no-polinomiales, como requerimiento para que la red neuronal sea capaz de aproximar cualquier función.

Añadido a esto, la disponibilidad de poder computacional limitado para implementar redes neuronales, junto a la tendencia dentro el campo de usar números cada vez mayores de capas escondidas, dieron lugar a arquitecturas que se vuelven computacionalmente prohibitivas, si en ellas se pretende utilizar funciones de activación arbitrarias y complejas.

Sobre las restricciones mencionadas, la experiencia obtenida por los practicantes de DL a través de los años ha llevado a adoptar un número pequeño de funciones relativamente sencillas, las cuales son usadas como funciones de activación dentro redes neuronales feedforward.

A continuación, se describe las funciones de activación esenciales y más usadas en arquitecturas de redes neuronales feedforward hoy en día.

- **Función lineal:** También denominada función de activación identidad. Es un mapeo directo su entrada z a la salida. Su ecuación se muestra a continuación, mientras que su comportamiento se encuentra graficado en la ilustración 9.

$$g(z) = z \quad (2.1.8)$$

La función lineal no posee límites en sus valores de salida. Por lo tanto, es usada mayormente como función de activación en la capa de salida de redes neuronales, si deben aproximar una función de la cual se debe obtener valores de salida no acotados.

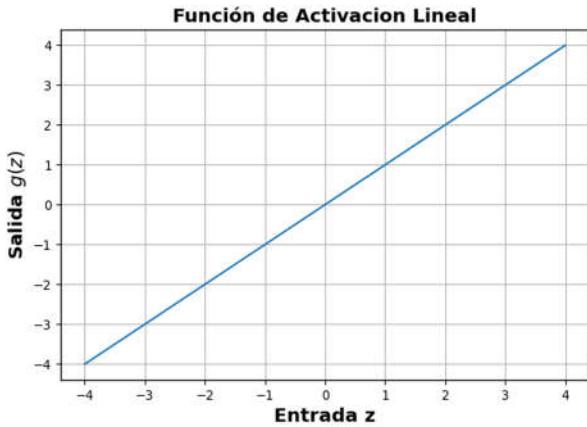


Ilustración 9 Función de activación lineal. Fuente: Elaboración propia.

- **Función sigmoide:** También denominada función logística. Es una función que mapea su entrada z a una salida no lineal entre los valores 0 y 1. Su ecuación y una gráfica de su comportamiento se muestran a continuación.

$$g(z) = \frac{1}{1 + e^{-z}} \quad (2.1.9)$$

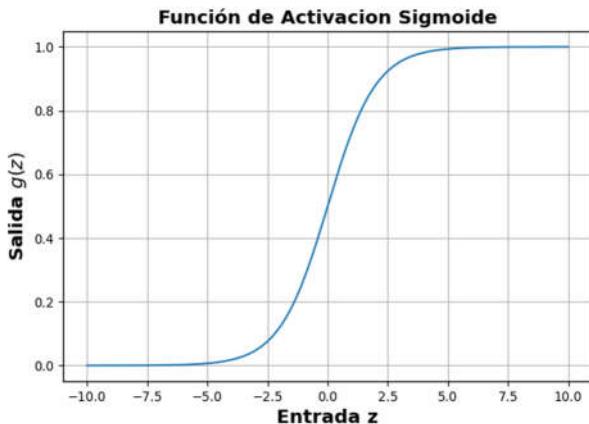


Ilustración 10 Función de activación sigmoidal. Fuente: Elaboración propia.

La función sigmoide es utilizada como función de activación en capas escondidas para introducir no-linealidad, pero también es utilizada como función de activación en las neuronas de la capa de salida para aproximar funciones que se encuentran acotadas entre 0 y 1, tales como funciones de probabilidad.

- **Función Tangente Hiperbólica (Tanh):** Es una función que mapea su entrada z a una salida no-lineal entre los valores -1 y 1. Su ecuación y una gráfica de su comportamiento se muestran a continuación.

$$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (2.1.10)$$

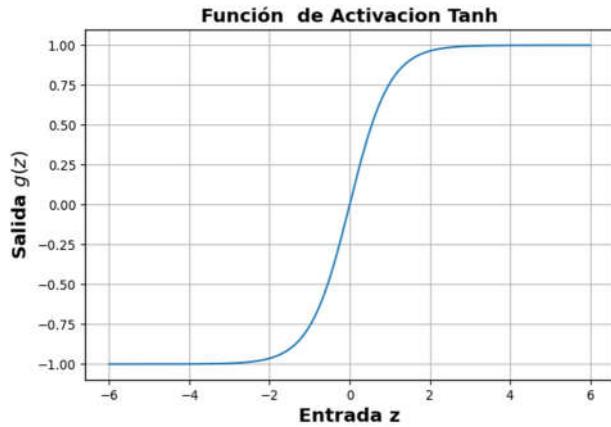


Ilustración 11 Función de activación tangente hiperbólica. Fuente: Elaboración propia.

La función TanH es utilizada mayormente como función de activación en capas escondidas para introducir no-linealidad, pero también es utilizada como función de activación en las neuronas de la capa de salida, para aproximar funciones que se encuentran acotadas entre un valor máximo positivo y negativo simétricos con respecto a cero.

- **Función Unidad Lineal Rectificada (ReLU):** Es una función que mapea su entrada z a una salida 0 si es que z es un valor negativo. Si z es un valor positivo, entonces es mapeado directamente a la salida de la neurona. Su ecuación y una gráfica de su comportamiento se muestran a continuación.

$$g(z) = \begin{cases} 0; z < 0 \\ z; z \geq 0 \end{cases} \quad (2.1.11)$$

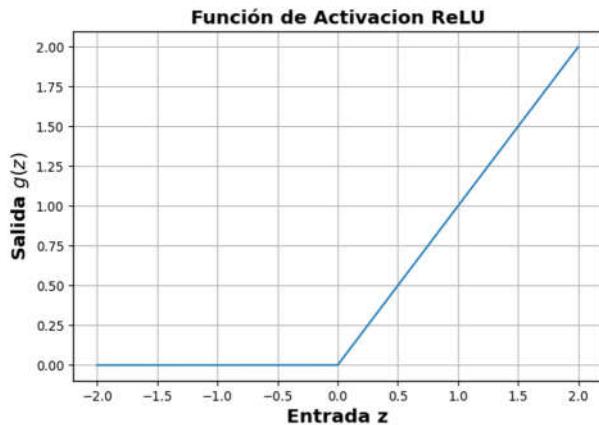


Ilustración 12 Función de activación ReLU. Fuente: Elaboración propia.

La función ReLU es utilizada en una gran mayoría como función de activación en neuronas de capas escondidas para introducir no-linealidad. Ya que la función ReLU requiere solamente de una comparación para retornar un valor que se encuentra previamente calculado, es computacionalmente eficiente comparada con otras funciones de activación, razón por la cual es la elección más común en la actualidad [30].

2.1.4. Entrenamiento de una Red Neuronal

Una vez que se ha diseñado la arquitectura de una red neuronal, previamente a que esta pueda ser útil para una determinada aplicación, se debe llevar a cabo el proceso de entrenamiento de la misma.

2.1.4.1. Inicialización de Una Red Neuronal.

Los parámetros de una red neuronal contenidos en \mathbf{W} y \mathbf{B} deben ser inicializados con valores determinados para que pueda ser entrenada. En adición, se sabe que la elección de dichos valores puede afectar la evolución del proceso de entrenamiento.

Una regla para generar los valores de inicialización de la red neuronal feedforward, considerada como una buena práctica, es inicializar los parámetros de peso $\mathbf{W}^{[k]}$ de cada capa k de la red neuronal es usando una distribución de probabilidad normal con media $\mu = \mathbf{0}$ y variancia $\sigma^2 = \frac{1}{n^{[k-1]}}$, donde $n^{[k-1]}$ es el número de neuronas en la capa anterior, mientras que se sugiere que el vector de polarización de dicha capa $\mathbf{B}^{[k]}$ sea inicializado con ceros [31].

2.1.4.2. Set de Datos

Se debe utilizar un set de datos para llevar a cabo el proceso de entrenamiento de una red neuronal. Un set de datos, es un conjunto de M ejemplos de pares entrada-salida, que representan de manera significativa la función que se desea aproximar utilizando la red neuronal. Cada ejemplo se denomina muestra y consta de un vector de datos de entrada $\mathbf{x}^{(m)} = [x_1^{(m)} \ x_2^{(m)} \ \dots \ x_n^{(m)}]$ y su salida correspondiente $\mathbf{y}^{(m)}$.

Todo set de datos se encuentra conformado por la tupla (\mathbf{X}, \mathbf{Y}) tal que \mathbf{X} es una matriz que posee todas las muestras de entrada y \mathbf{Y} es una matriz que posee las correspondientes muestras de salida, como se muestra a continuación.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \vdots \\ \mathbf{x}^{(M)} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} \mathbf{y}^{(1)} \\ \mathbf{y}^{(2)} \\ \vdots \\ \mathbf{y}^{(M)} \end{bmatrix} \quad (2.1.12)$$

2.1.4.3. Función de Pérdida

El proceso de entrenamiento busca modificar los parámetros de la red neuronal dentro \mathbf{W} y \mathbf{B} de manera que, dado un set de datos determinado con entradas contenidas en \mathbf{X} , las predicciones de la red neuronal contenidas en una matriz denominada $\widehat{\mathbf{Y}}$, aproximen los ejemplos correctos del set de datos, lo cual se expresa mediante $\widehat{\mathbf{Y}} \approx \mathbf{Y}$.

Para este fin, se debe definir primeramente una métrica mediante la cual se pueda cuantificar el error que existe entre las predicciones de la red neuronal $\widehat{\mathbf{Y}}$ y los valores de salida reales \mathbf{Y} . Dicha métrica se denomina función de pérdida y se la representa mediante \mathcal{L} .

Un tipo de función de pérdida que es utilizada a ampliamente en aprendizaje profundo para el entrenamiento de redes neuronales feedforward es el error cuadrático medio, el cual se calcula mediante la ecuación (2.1.13).

$$\mathcal{L}(\hat{Y}, Y) = \frac{1}{M} \sum_{m=1}^M (\hat{y}^{(m)} - y^{(m)})^2 \quad (2.1.13)$$

2.1.4.4. Algoritmo de Descenso de Gradiente

Definida una función de perdida, el proceso de entrenamiento se puede expresar en términos de minimizar dicha función de pérdida, ya que cuando esta se reduce a un mínimo, las predicciones de la red neuronal se acercan a los valores ejemplo dentro Y ; por consecuente se considera que la red neuronal aproxima una función que mapea X a las salidas Y .

El algoritmo de descenso de gradiente, es un algoritmo de optimización que es usado como una base para el entrenamiento de redes neuronales en aprendizaje profundo; provee una manera de mejorar los parámetros de peso W y los parámetros de polarización B de la red neuronal gradualmente a lo largo de varias iteraciones, buscando hallar un mínimo en la función de perdida.

Ya que la función de perdida es función de las predicciones \hat{Y} de la red neuronal, pero estas a su vez son funciones de X y de los parámetros de la red neuronal W y B , entonces es posible expresar a la función de perdida como $\mathcal{L}(\hat{Y}(X, W, B), Y)$.

El algoritmo se basa en el hecho que el gradiente de una función diferenciable indica la dirección de mayor crecimiento de dicha función. De manera conversa, el negativo de dicho gradiente indica la dirección de mayor decrecimiento. Entonces, hallando los gradientes de la función de perdida $\mathcal{L}(\hat{Y}(X, W, B), Y)$; indicados por ∇ , con respecto a los parámetros W y B , se la puede minimizar siguiendo la dirección mostrada por esos gradientes. El algoritmo 1 detalla la manera en la que se lleva a cabo el descenso de gradiente.

Algoritmo 1 Descenso de gradiente

Entradas: Red Neuronal Inicializada, Set de datos (X, Y) , tasa de aprendizaje α

- 1: **repetir**
 - 2: Calcular gradientes $\nabla_W L(\hat{Y}, Y)$, $\nabla_B L(\hat{Y}, Y)$
 - 3: $W \leftarrow W - \alpha * \nabla_W L(\hat{Y}, Y)$
 - 4: $B \leftarrow B - \alpha * \nabla_B L(\hat{Y}, Y)$
 - 5: **hasta** convergencia
-

Se denomina una época a cada iteración del algoritmo de descenso de gradiente, en la cual se ha utilizado todos los datos que se encuentran disponibles en el set de datos para optimizar la red neuronal.

El parámetro $\alpha \in [0, 1]$ se denomina tasa de aprendizaje; es responsable de escalar la medida en la cual cambian los parámetros de la red neuronal durante cada época y debe ser elegido cuidadosamente. Una tasa de aprendizaje alta genera saltos grandes dentro la función de perdida \mathcal{L} . Esto genera una convergencia rápida a un rango de valores que podría potencialmente estar lejos del mínimo.

Una tasa de aprendizaje demasiado baja, lleva a convergencia demasiado lenta dentro de \mathcal{L} , con la posibilidad que no se alcance un mínimo dentro un número de épocas de cómputo razonable. Finalmente, una elección correcta de α lleva a una convergencia gradual al mínimo de \mathcal{L} .

La ilustración 13 muestra un ejemplo simplificado, en el cual se minimiza la función de pérdida \mathcal{L} con relación a un solo parámetro w mediante descenso de gradiente. En ella, es posible observar el comportamiento con respecto a α en los tres casos descritos anteriormente a lo largo de cinco épocas de entrenamiento, demostrando la importancia de elegir una tasa de aprendizaje adecuada. El concepto ilustrado en este ejemplo simple, se extiende a una función de perdida en cualquier número de dimensiones, lo cual equivale a decir que es válido para una red neuronal con cualquier número de parámetros.

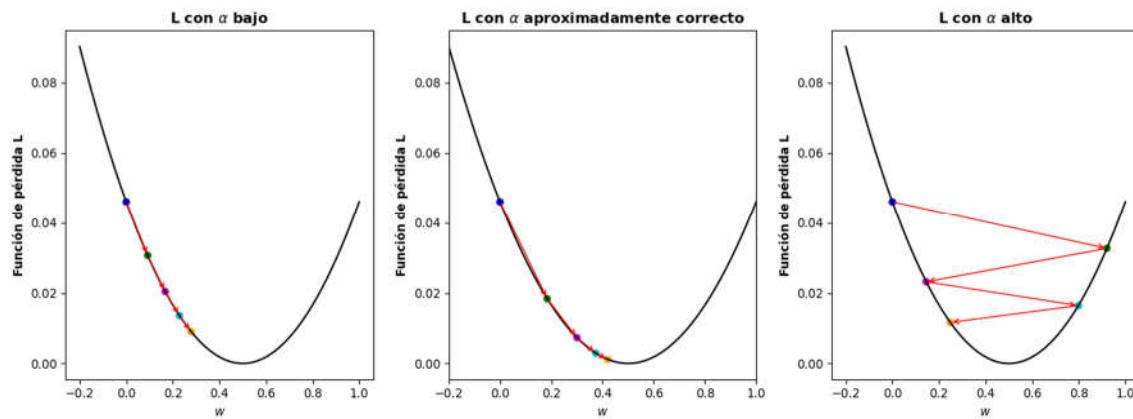


Ilustración 13 Efecto de la tasa de aprendizaje. Fuente: Elaboración propia.

2.1.4.5. Retropropagación

Se observó en la anterior sección que el algoritmo de descenso de gradiente requiere calcular los gradientes $\nabla_W \mathcal{L}$ y $\nabla_B \mathcal{L}$. El algoritmo de retropropagación, es un algoritmo ampliamente utilizado para este propósito en el entrenamiento de redes neuronales [32].

Se basa en la regla de la cadena, la cual expresa la derivada de una función compuesta $f(g(x))$ como un producto, como se muestra en la ecuación (2. 1. 14).

$$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x) \quad (2. 1. 14)$$

Se utiliza la regla de la cadena para calcular los gradientes de la función de perdida con respecto a los parámetros $W^{[k]}$ y $B^{[k]}$ de cada capa de la red neuronal como derivadas parciales, como se expresa en las dos ecuaciones siguientes.

$$\nabla_{W^{[k]}} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial W^{[k]}} \quad (2. 1. 15)$$

$$\nabla_{B^{[k]}} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial B^{[k]}} \quad (2. 1. 16)$$

Dichos gradientes se calculan de manera secuencial desde la capa de salida hacia la primera capa de la red neuronal, expresando $\frac{\partial \mathcal{L}}{\partial W^{[k]}}$ y $\frac{\partial \mathcal{L}}{\partial B^{[k]}}$ mediante la regla de la cadena como un producto, el cual contiene las derivadas parciales de todas las operaciones que se encuentran delante de la operación lineal que se lleva a cabo en la capa k , como muestran las ecuaciones (2.1.17) y (2.1.18).

$$\frac{\partial \mathcal{L}}{\partial W^{[k]}} = \frac{\partial \mathcal{L}}{\partial Z^{[k]}} \frac{\partial Z^{[k]}}{\partial W^{[k]}} \quad (2.1.17)$$

$$\frac{\partial \mathcal{L}}{\partial B^{[k]}} = \frac{\partial \mathcal{L}}{\partial Z^{[k]}} \frac{\partial Z^{[k]}}{\partial B^{[k]}} \quad (2.1.18)$$

A su vez para calcular el término $\frac{\partial \mathcal{L}}{\partial Z^{[k]}}$ en ambas ecuaciones, se usa la regla de la cadena nuevamente; se observa que es igual a un producto de la derivada parcial de la función de activación de su propia capa y de todas las operaciones llevadas a cabo en la capa $k + 1$:

$$\frac{\partial \mathcal{L}}{\partial Z^{[k]}} = \frac{\partial \mathcal{L}}{\partial A^{[k+1]}} \frac{\partial A^{[k+1]}}{\partial Z^{[k]}} \quad (2.1.19)$$

En base a las ecuaciones (2.1.17), (2.1.18) y (2.1.19), se hace posible calcular el gradiente para los parámetros de cada capa k de manera eficiente, empezando al final del grafo computacional de la red neuronal, almacenando las derivadas parciales que se calculan regresando hacia la capa de entrada.

La ilustración 14 muestra cómo se calcula los gradientes para la red neuronal representada por el grafo computacional de la ilustración 13; se observa cómo se retorna desde la función de perdida, almacenando las derivadas parciales de la función de perdida para la capa número k . Dichas derivadas después son utilizadas para calcular tanto el gradiente en la capa número $k - 1$ como para mejorar los parámetros de la red neuronal en el algoritmo de descenso de gradiente.

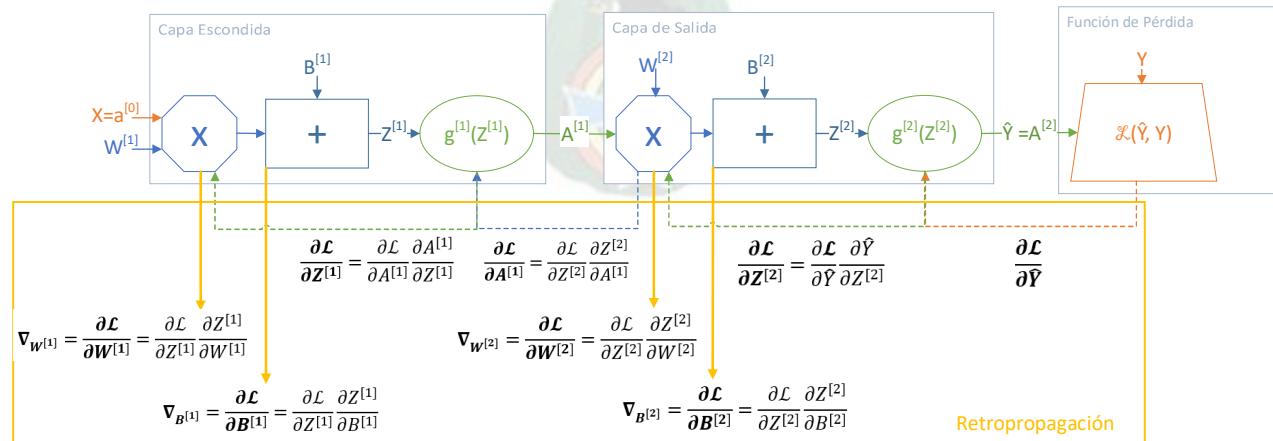


Ilustración 14 Retropropagación representada en el grafo computacional. Fuente: Elaboración propia.

2.1.4.6. Descenso de Gradiente Estocástico

El algoritmo de descenso de gradiente estocástico fue ideado como una mejora al algoritmo de descenso de gradiente [33], es detallado en el algoritmo 2.

Algoritmo 2 Descenso de gradiente estocástico

Entradas: Red Neuronal Inicializada, Set de datos (X, Y) , Tasa de aprendizaje α , Tamaño de lote D

- 1: $t \leftarrow 0$
 - 2: **repetir**
 - 3: **repetir**
 - 4: Tomar un lote (X_D, Y_D) de muestras aleatorias
 - 5: Calcular $\nabla_W L(\hat{Y}_D, Y_D)$, $\nabla_B L(\hat{Y}_D, Y_D)$
 - 6: $W_t \leftarrow W_{t-1} - \alpha * \nabla_W L(\hat{Y}_D, Y_D)$
 - 7: $B_t \leftarrow B_{t-1} - \alpha * \nabla_B L(\hat{Y}_D, Y_D)$
 - 8: $t \leftarrow t + 1$
 - 9: **hasta** hasta haber usado cada muestra de (X, Y)
 - 10: **hasta** convergencia
-

En el algoritmo de descenso de gradiente estocástico, se reemplaza los gradientes calculados usando el set de datos entero (X, Y) por gradientes calculados usando un subconjunto representado por (X_D, Y_D) llamado lote, el cual contiene D muestras tomadas al azar del set de datos, dichos gradientes son utilizados para mejorar los parámetros de la red neuronal. El tamaño D de lote que se utiliza, es una variable a discreción de la persona implementando el entrenamiento de la red neuronal.

Cada vez que un lote es utilizado para mejorar la red neuronal, se considera que ha transcurrido un paso de entrenamiento, el cual es indicado por la variable t , mientras que se considera que una época ha transcurrido después que cada muestra del set de datos se ha usado por lo menos una vez para calcular los gradientes.

2.1.4.7. ADAM

El algoritmo de estimación adaptiva de momento (ADAM; Adaptive Moment Estimation por su sigla en inglés), es un algoritmo de optimización basado en descenso de gradiente estocástico que fue ideado para mejorar su rendimiento [34]. Se lo detalla en el algoritmo 3.

ADAM utiliza estimaciones de los primeros y segundos momentos de los gradientes para escalar la tasa de aprendizaje de cada parámetro de cada neurona individualmente. Los primeros y segundos momentos estimados para el paso de entrenamiento t se encuentran dentro las matrices denominadas m_t y v_t respectivamente. Se estiman mediante una regla de decaimiento exponencial utilizando los parámetros $\beta_1, \beta_2 \in [0, 1]$, como se observa en las líneas 7 y 8 del algoritmo.

Ya que las matrices de ambos momentos son inicializadas con $m_0 = [0]$ y $v_0 = [0]$, las estimaciones de los momentos se encontrarán sesgadas hacia cero durante los primeros pasos de entrenamiento. Para contrarrestar dicho sesgo, se corrige ambas estimaciones para obtener las matrices \hat{m}_t y \hat{v}_t , como se muestra en las líneas 9 y 10 del algoritmo.

Finalmente, se mejora los parámetros de la red neuronal usando una ecuación modificada para incluir a \hat{m}_t y \hat{v}_t en la línea 11 del algoritmo. Este paso incluye un número pequeño ϵ ; su propósito es evitar la división entre cero.

Algoritmo 3 ADAM

Entradas: Red Neuronal Inicializada, Set de datos (X, Y) , Tamaño de lote D

Tasa de aprendizaje α , Tasas de decaimiento exponencial para los estimados de los momentos β_1 y β_2

- 1: $m \leftarrow [0], v \leftarrow [0]$
 - 2: $t \leftarrow 0$
 - 3: **repetir**
 - 4: **repetir**
 - 5: Tomar un lote (X_D, Y_D) de muestras aleatorias
 - 6: Calcular $\nabla_\varphi(\hat{Y}_D, Y_D)$
 - 7: $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\varphi(\hat{Y}_D, Y_D)$
 - 8: $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) \nabla_\varphi^2(\hat{Y}_D, Y_D)$
 - 9: $\hat{m}_t \leftarrow (m_t)/(1 - \beta_1^t)$
 - 10: $\hat{v}_t \leftarrow (v_t)/(1 - \beta_2^t)$
 - 11: $\varphi_t \leftarrow \varphi_{t-1} - \alpha * (\hat{m}_t)/(\sqrt{\hat{v}_t} + \epsilon)$
 - 12: $t \leftarrow t + 1$
 - 13: **hasta** hasta haber usado cada muestra de (X, Y)
 - 14: **hasta** convergencia
-

2.1.4.8. Hiperparámetros de un Sistema de Aprendizaje Profundo

Se denomina hiperparámetros de un sistema de aprendizaje profundo a toda cantidad o elección de diseño que se puede realizar para afectar el proceso de entrenamiento.

A continuación, se muestra un resumen de algunos hiperparámetros descritos a lo largo de esta sección:

- **Arquitectura de la red neuronal:** La cual comprende su número de capas, el tamaño de cada capa, y la elección de las funciones de activación para cada capa.
- **Elección del algoritmo de entrenamiento:** El cual puede afectar la rapidez con la cual la red neuronal converge a un mínimo en la función de perdida.
- **Tasa de aprendizaje:** Que como se observó anteriormente, puede afectar la convergencia de la red neuronal durante el proceso de entrenamiento.

2.2. Sistemas de Aprendizaje Reforzado Profundo

El aprendizaje reforzado (no profundo), es un campo dentro del aprendizaje automático en el cual se busca desarrollar sistemas capaces aprender comportamientos complejos, a partir de interacciones con el entorno para lograr un determinado objetivo. La base teórica y técnicas relacionadas fueron desarrolladas inicialmente durante la década de los años 1960, buscando mejorar la teoría de control óptimo [35].

El Aprendizaje Reforzado Profundo (DRL; Deep Reinforcement Learning por su sigla en inglés) es un campo emergente que toma técnicas de DL y las utiliza en conjunción con la teoría

desarrollada dentro el aprendizaje reforzado. Esta unión obtuvo logros notables y un desarrollo acelerado a partir de la década de los años 2010 [36], particularmente dentro entornos simulados y de video juegos. Actualmente el campo se encuentra en constante desarrollo y crecimiento.

2.2.1. Componentes de un sistema de aprendizaje reforzado profundo

Todo sistema de aprendizaje reforzado profundo se encuentra compuesto por dos entes mutuamente exclusivos que se encuentran en estrecha y constante interacción [37]:

2.2.1.1. Agente

El agente es un ente computacional encargado de decidir cual acción ejecutar dentro el entorno. La decisión de dicha acción, representada mediante a , es llevada a cabo por el agente usando una red neuronal, en base a una observación del estado del entorno denominada s . Mediante la acción, el agente busca influenciar al entorno de manera que alcance lograr un determinado objetivo.

El agente incrementa su rendimiento a partir de un proceso de interacción con el entorno, evaluando los resultados de las acciones elegidas anteriormente y mejorando la toma de decisiones hasta alcanzar el objetivo; esto implica que el agente aprende a tomar decisiones mejores a partir de sus interacciones con el entorno. El proceso de interacción, evaluación y aprendizaje es denominado el entrenamiento del agente.

2.2.1.2. Entorno

El entorno es un ente externo al agente, con el cual el agente interactúa y dentro el cual busca lograr un objetivo. El entorno posee sus propias reglas y funcionamiento interno, los cuales son desconocidos y no pueden ser afectados por el agente; las acciones que el agente ejecuta dentro el entorno son la única manera que el posee para influenciar al entorno hasta lograr el objetivo.

El entorno por lo general posee un estado interno que no es completamente observable por el agente, es decir que el estado del entorno es parcialmente observable. Para aplicar aprendizaje reforzado profundo, se debe garantizar previamente que el agente pueda observar un estado con suficientes variables independientes para caracterizar el estado del entorno completamente. Dado esto, en aprendizaje reforzado se usa los términos estado y observación intercambiablemente [38].

Dentro el ejemplo del humanoide simulado de DeepMind [19] se identifica los componentes principales de la siguiente manera:

- El objetivo es mantener al humanoide de pie y avanzar hacia adelante sin alejarse del centro del camino.
- El agente se encuentra compuesto por la red neuronal que decide los torques que se aplican a cada una de las articulaciones del humanoide simulado, junto a algoritmos que entrena la red neuronal para poder lograr el objetivo.
- El entorno se encuentra compuesto por el humanoide, el suelo, los obstáculos y demás factores externos que se encuentran en su proximidad y son capaces de influenciar de manera significativa el alcance del objetivo.
- El estado s se encuentra compuesto por las posiciones y velocidades del humanoide en cada eje del espacio, junto a los ángulos y velocidades angulares de cada una de sus articulaciones.

2.2.2. Interacción agente-entorno

El agente y el entorno se encuentran en estrecha interacción; se lleva cabo un intercambio de señales entre ambos a lo largo de una sucesión de instantes discretos de tiempo, la cual se describe a continuación.

2.2.2.1. Experiencias

Una experiencia, denominada también paso de tiempo, es la interacción más básica posible entre el entorno y el agente dentro un sistema de aprendizaje reforzado profundo [39].

En un instante de tiempo t , el agente observa que el entorno se encuentra en el estado s_t , entonces elige ejecutar una acción a_t dentro de él. Esto ocasiona que el entorno realice una transición a un estado s_{t+1} , que es nuevamente observado por el agente en el instante de tiempo $t + 1$ junto a una señal r_{t+1} , denominada señal de recompensa.

Una experiencia se expresa mediante la siguiente tupla:

$$(s_t, a_t, r_{t+1}, s_{t+1}) \quad (2.2.1)$$

La señal de recompensa es un escalar que informa al agente en qué medida haber escogido a_t en el estado s_t es beneficioso o perjudicial con respecto al alcance del objetivo.

2.2.2.2. Trayectorias

Una trayectoria es una serie de experiencias generadas por la interacción e intercambio de señales entre el agente y el entorno a lo largo de instantes de tiempo sucesivos.

Empezando de un instante de tiempo inicial $t_0 = 0$, en el cual el entorno se encuentra en un estado inicial s_0 , el proceso de generación de una experiencia es repetido en sucesión. Así, el agente genera información acerca de la evolución del estado y de la recompensa en relación con las acciones que elige tomar, generando trayectorias de experiencias denominadas τ , las cuales poseen la siguiente forma:

$$\tau = (s_0, a_0, r_1, s_1), (s_1, a_1, r_2, s_2), (s_2, a_2, r_3, s_3) \dots \quad (2.2.2)$$

La generación de una trayectoria τ esta transcurre en instantes de tiempo incrementales, mientras el agente toma solamente el estado s_t recibido en el instante de tiempo t como dato suficiente y actual para elegir una acción a_t . La ilustración 15, muestra la interacción agente-entorno en la generación de una trayectoria, junto al intercambio de señales descrito.

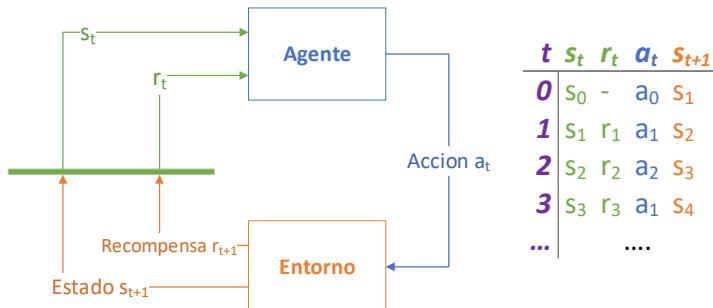


Ilustración 15 Interacción agente-entorno en la generación de una trayectoria. Fuente: Elaboración propia.

2.2.2.3. Tareas Episódicas y Tareas Continuas

Si en un determinado entorno, la interacción entre el agente y el entorno termina después de un número determinado de instantes de tiempo T , alcanzando un estado terminal denominado s_T , o si el entorno posee uno o más estados terminales s_T , los cuales al ser alcanzados terminan el ciclo de interacción forzosamente, entonces se dice que el sistema lleva a cabo una tarea episódica; la trayectoria generada hasta alcanzar el estado s_T se denomina un episodio, el cual posee la forma mostrada en la ecuación (2.2.3).

$$\tau_{\text{Episodio}} = (s_t, a_t, r_{t+1}, s_{t+1}), (s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}), \dots (s_{T-1}, a_{T-1}, r_T, s_T) \quad (2.2.3)$$

Después de la finalización de un episodio, por lo general el estado del sistema es devuelto a uno o más estados iniciales denominados s_0 , los cuales son tomados de una distribución de probabilidad que se considera estacionaria [40].

Si la interacción entre el agente y el entorno no posee una condición de terminación, entonces el instante de tiempo T se encuentra en el infinito y se dice que el sistema lleva a cabo una tarea continua.

Dentro el ejemplo de DeepMind [19] se lleva a cabo una tarea episódica, la cual termina cuando el androide simulado cae al suelo. Es decir, un episodio termina cuando el humanoide alcanza un estado en el cual su posición sobre el eje z se encuentra a la altura del suelo.

2.2.3. Procesos de Decisión de Markov

Se asume que, en todo sistema de aprendizaje reforzado profundo, los ciclos de interacción entre el agente y el entorno se llevan a cabo sobre la base de un Proceso de Decisión de Markov (MDP, Markov Decision Process por su sigla en inglés) [41].

Un MDP se encuentra definido por una tupla $(S, \mathcal{A}, \mathcal{P}, \mathcal{R})$, la cual consta de los elementos que se describen seguidamente.

2.2.3.1. Espacio de Estados S

El espacio de estados, representado mediante S , es el conjunto de todos los estados posibles que el entorno puede adoptar tal que cualquier estado s que se observe pertenecerá siempre a S :

$$s \in S \quad (2.2.4)$$

Cada estado s puede ser tanto un escalar, como un vector como con un número finito de componentes de la forma $s_t = [a, b, c, \dots, z]$. Sin embargo, el espacio de estados S puede ser un conjunto no finito, ya que uno o más elementos de s_t pueden pertenecer a conjuntos no finitos, como el conjunto de los números reales [42].

2.2.3.2. Espacio de Acciones

El espacio de acciones, representado mediante \mathcal{A} es el conjunto de todas las acciones posibles que el agente puede seleccionar tal que cualquier acción a será tomada siempre de \mathcal{A} :

$$a \in \mathcal{A} \quad (2.2.5)$$

De igual manera, cada acción debe tener un número finito de componentes, mientras que \mathcal{A} mismo puede ser un conjunto no finito.

2.2.3.3. Función de Transición de Estado

La función de transición de estado del entorno, representada mediante \mathcal{P} , es una distribución de probabilidad, la cual expresa la probabilidad que, dado que en un instante de tiempo t el entorno se encuentra en el estado s_t , al ejecutar la acción a_t , el estado del entorno cambiará a un determinado estado s_{t+1} . De dicha distribución se toma el siguiente estado observado por el agente, según lo que se expresa en la ecuación (2.2.6).

$$s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t) \quad (2.2.6)$$

La función de transición de estado posee una propiedad, mediante la cual se especifica que la suma de las probabilidades de transición hacia todos los estados siguientes siempre será igual a uno, como se muestra en la ecuación (2.2.7).

$$\sum_{s_{t+1} \in S} \mathcal{P}(s_{t+1}|s_t, a_t) = 1 \quad (2.2.7)$$

De manera importante, se asume que la función de transición de estado cumple la propiedad de Markov, la cual especifica que la probabilidad de cada estado que puede ser tomado en el instante de tiempo $t + 1$, es independiente de la historia de interacción a lo largo de la trayectoria [43], como expresa la ecuación (2.2.8).

$$\mathcal{P}(s_{t+1}|s_t, a_t) = \mathcal{P}(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2} \dots) \quad (2.2.8)$$

Esta propiedad reafirma que dentro de un sistema enmarcado como un proceso de decisión de Markov, solo se necesita la información del estado actual s_t para poder realizar una transición del entorno al estado siguiente s_{t+1} ; de esta propiedad se asume que el agente requiere solamente de s_t para realizar la elección de a_t , lo cual simplifica de manera significativa la manera en la cual este puede ser implementado.

2.2.3.4. Función de Recompensa

La función de recompensa \mathcal{R} es una función que asigna un valor escalar r_t a toda tupla de transición de estado de la forma (s_{t-1}, a_{t-1}, s_t) :

$$r_t = \mathcal{R}(s_{t-1}, a_{t-1}, s_t) \quad (2.2.9)$$

El escalar r_t es recibido cada vez que el entorno realiza una transición a un nuevo estado como consecuencia de una acción ejecutada y es utilizado por el agente como una medida de su propio rendimiento.

Por lo tanto, la recompensa debe ser un valor que cuantifica e informa si la acción ejecutada por el agente es una acción favorable o no con relación a alcanzar el objetivo; entonces la función \mathcal{R} debe ser diseñada de manera que codifique el objetivo que se busca alcanzar e informe al agente que tan lejos o cerca se encuentra de alcanzar dicho objetivo en cada instante de tiempo [44].

El ejemplo del humanoide simulado de DeepMind [19] muestra cómo se puede codificar un objetivo complejo como mantener un humanoide de pie y avanzar en una función recompensa. La ecuación de la función de recompensa para dicho ejemplo es la siguiente:

$$\begin{aligned} \mathbf{r}_t = \min(v_x, v_{max}) - 0.005(v_x^2 + v_y^2) - 0.05y^2 - 0.02\|u\|^2 + 0.02 & \quad (2.2.10) \\ (1) & \quad (2) \quad (3) \quad (4) \quad (5) \end{aligned}$$

La ecuación (2.2.10) codifica el objetivo de la siguiente manera: El término (1) asigna una recompensa positiva proporcional a la velocidad del humanoide sobre el eje x hasta un límite v_{max} . (2) asigna una recompensa negativa (penaliza) al agente proporcionalmente al cuadrado de las velocidades sobre los ejes x y y , esto informa al agente que no debe mover el humanoide en dirección del eje y , pero tampoco moverse demasiado rápido sobre el eje x . (3) penaliza la posición del humanoide sobre el eje y , esto informa que es preferible no alejarse del centro del camino. (4) penaliza proporcionalmente al cuadrado del torque aplicado en las articulaciones, esto informa que es preferible no aplicar demasiado torque, lo cual evita que se realice movimientos demasiado toscos. Finalmente, (5) asigna una recompensa positiva constante por cada instante de tiempo que pasa durante el episodio, esto informa que se debe evitar caer y que el episodio termine para seguir recolectando recompensas.

2.2.4. Retorno

Se define como retorno $\mathbf{G}_t(\tau)$ a la suma de todas las recompensas futuras \mathbf{r} obtenidas a partir de un instante de tiempo t a lo largo de una trayectoria τ , como se expresa en la ecuación (2.2.11).

$$\mathbf{G}_t(\tau) = r_{t+1} + r_{t+2} + \dots + r_T = \sum_{k=0}^T r_{t+k+1} \quad (2.2.11)$$

2.2.4.1. Retorno Descontado

La ecuación (2.2.11) puede resultar tanto en valores altos para episodios largos, como en valores infinitos en el caso de una tarea continua con $T = \infty$. Para evitar esto, se introduce un escalar $\gamma \in [0, 1]$ denominado tasa de descuento con el fin de acotar \mathbf{G}_t , obteniendo la expresión del retorno descontado que se muestra a continuación.

$$\mathbf{G}_t(\tau) = r_{t+1} + \gamma^1 r_{t+2} + \gamma^2 r_{t+3} \dots + \gamma^{T-1} r_T = \sum_{k=0}^T \gamma^k r_{t+k+1} \quad (2.2.12)$$

Además de acotar \mathbf{G}_t , la tasa de descuento permite asignar un peso a la influencia de las recompensas futuras [45]; mientras más pequeño γ , menos importancia tienen las recompensas provenientes de instantes de tiempo más alejados en el futuro. La influencia de γ sobre la importancia de las recompensas futuras en la ecuación (2.2.12), se hace evidente en los casos extremos descritos a continuación:

- Si $\gamma \approx 0$, entonces los términos de la ecuación que incluyen la recompensa r_{t+2} y sus subsecuentes se aproximan a cero dentro el retorno descontado. Por lo tanto, tienen una menor influencia sobre el valor de \mathbf{G}_t , el cual se approxima a un valor cercano a r_{t+1} .

- Si $\gamma \approx 1$, los términos de la ecuación que incluyen la recompensa r_{t+2} y sus subsecuentes tienen aproximadamente la misma influencia que r_{t+1} sobre el valor de G_t , el cual se approxima al retorno sin descuento.

Es posible reformular las ecuaciones del retorno de manera recursiva, como se muestra en la siguiente ecuación.

$$G_t(\tau) = r_{t+1} + \gamma G_{t+1}(\tau) \quad (2.2.13)$$

2.2.5. Política

La política es el componente del agente que se encuentra directamente encargado de decidir cual acción tomar a partir de una observación de estado. Por lo tanto, es el ente que controla el comportamiento del agente [46].

Existen políticas estocásticas y determinísticas. En el caso estocástico, la política π se define como un mapeo de un estado s_t a una distribución de probabilidades de acción, de la cual se toma la acción a_t :

$$a_t \sim \pi(a_t | s_t) \quad (2.2.14)$$

En el caso determinístico, una política determinística es un mapeo directo de un estado s_t a una acción a_t :

$$a_t = \pi(s_t) \quad (2.2.15)$$

En el aprendizaje reforzado profundo, las políticas son implementadas mediante redes neuronales. Si una red neuronal posee un conjunto de parámetros φ y es usada para implementar una política denominada π^φ , es posible utilizar todas las técnicas y algoritmos de DL para entrenar dicha política.

Dada una red neuronal con arquitectura definida π^1 que posee parámetros contenidos dentro φ_1 y la misma red neuronal en otro instante con un conjunto de parámetros φ_2 distintos $\varphi_1 \neq \varphi_2$, se dice que la red neuronal con φ_2 representa una política diferente.

2.2.6. Funciones de Valor

Las funciones de valor de una política son funciones que estiman el valor esperado del retorno (descontado o no descontado), su propósito es ayudar al agente a evaluar el rendimiento de su política.

2.2.6.1. Función de Valor de Estado

La función de valor de estado correspondiente a una política π , denotada por $V^\pi(s)$, representa el valor esperado del retorno si el sistema se encuentra en un estado s_t y elige todas las acciones consecuentes usando π [47], como expresa (2.2.16).

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi}[G(\tau) | s = s_t] \quad (2.2.16)$$

2.2.6.2. Función de Valor de Estado-Acción

La función de valor de estado-acción correspondiente a una política π , denotada por $Q^\pi(s, a)$, representa el valor esperado del retorno si el sistema se encuentra en un estado s_t , elige tomar la acción a_t y luego elige todas las consecuentes acciones usando π [48], como expresa la siguiente ecuación.

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi}[G(\tau) | s = s_t, a = a_t] \quad (2.2.17)$$

Existe una relación entre ambas funciones de valor; la función de valor de estado es obtenida calculando el valor esperado de la función de valor de estado-acción sobre todas las acciones tomadas de π , como expresa la siguiente ecuación.

$$V^\pi(s) = \mathbb{E}_{a \sim \pi}[Q^\pi(s, a)] \quad (2.2.18)$$

Ambos tipos de funciones de valor son aproximados mediante redes neuronales dentro DRL.

2.2.7. Ecuaciones de Bellman

Dada la forma recursiva del retorno expresada en la ecuación (2.2.13), tomando en cuenta que las funciones de valor representan estimaciones del retorno, es posible expresar cada función de valor en forma recursiva, obteniendo las denominadas ecuaciones de Bellman.

Reemplazando la ecuación (2.2.13) en la ecuación (2.2.16) y usando $V^\pi(s_{t+1})$ como estimado del retorno para el instante de tiempo $t + 1$, se obtiene la ecuación de Bellman para el valor de estado, la cual se muestra a continuación.

$$V^\pi(s_t) = \mathbb{E}_{a \sim \pi, s_{t+1} \sim \mathcal{P}}[r(s, a) + \gamma V^\pi(s_{t+1})] \quad (2.2.19)$$

Para obtener la ecuación de Bellman de valor de estado-acción (2.2.20), se reemplaza la ecuación (2.2.18) en (2.2.19), tomando en cuenta nuevamente que en la ecuación se usan datos del instante de tiempo $t + 1$.

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathcal{P}}[r(s, a) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]] \quad (2.2.20)$$

La principal ventaja de expresar las funciones de valor como ecuaciones de Bellman, es que con ello se hace posible calcular estimados de manera recursiva.

2.2.8. Función Objetivo

Dentro un proceso de decisión de Markov en el cual toma parte un agente que posee una política π^φ , la cual genera trayectorias τ , de las cuales se calcula el retorno $G(\tau)$ (descontado o no), la función objetivo $J(\pi^\varphi)$, expresada en la ecuación (2.2.21), se define como el valor esperado del retorno sobre las trayectorias generadas usando dicha política [49].

$$J(\pi^\varphi) = \mathbb{E}_{\tau \sim \pi^\varphi}[G(\tau)] \quad (2.2.21)$$

Si se logra maximizar la función objetivo $J(\pi^\varphi)$, entonces se puede decir que las recompensas a lo largo de toda trayectoria generada mediante π^φ se encontrarán en un valor máximo. De esto se puede intuir que, al maximizar la función objetivo, se alcanza el objetivo codificado implícitamente dentro la función de recompensa.

Por lo tanto, como se muestra en la ecuación (2.2.22), todo agente de aprendizaje reforzado profundo busca hallar una política optima denominada π^* tal que, cualquier trayectoria generada usando π^* maximice el valor de la función objetivo.

$$\pi^* = \operatorname{argmax}_{\pi} J(\pi^{\varphi}) \quad (2.2.22)$$

2.2.9. Funciones de Valor Óptimas

A partir la anterior definición de política optima, es posible definir expresiones para funciones de valor óptimas.

La función de valor de estado optima denominada $V^*(s)$, estima el valor esperado del retorno si el sistema se encuentra en el estado s_t y se elige todas las acciones consecuentes usando la política optima π^* :

$$V^*(s) = \max_{\pi} \mathbb{E}_{\tau \sim \pi}[G(\tau) | s = s_t] \quad (2.2.23)$$

La función de valor de estado-acción optima denominada $Q^*(s, a)$, estima el valor esperado del retorno si el sistema se encuentra en el estado s_t , se elige la acción a_t y luego se elige todas las acciones consecuentes usando la política optima π^* :

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\tau \sim \pi}[G(\tau) | s = s_t, a = a_t] \quad (2.2.24)$$

De la ecuación (2.2.24), es posible concluir que la acción optima $a_t^*(s_t)$ escogida por la política optima π^* para el estado s_t , será siempre la acción con el máximo retorno estimado para dicho estado usando $Q^*(s, a)$, como expresa la ecuación (2.2.25).

$$\pi^*(s_t) = a_t^*(s_t) = \operatorname{argmax}_a (Q^*(s_t, a)) \quad (2.2.25)$$

De la anterior ecuación, se observa que es posible aproximar $Q^*(s, a)$ y utilizarla para hallar las acciones que corresponden a la política óptima.

Es posible también expresar la función de valor de estado optima en términos de la función de valor de estado-acción optima; seleccionando siempre la acción a_t que provee el máximo retorno para el estado s_t , como se representa en la ecuación (2.2.26).

$$V^*(s_t) = \max_{a_t} Q^*(s_t, a_t) \quad (2.2.26)$$

2.2.10. Ecuaciones de Bellman Óptimas

También existen representaciones óptimas para las ecuaciones de Bellman, mostradas en las siguientes ecuaciones. Éstas permiten expresar las funciones de valor optimas de una manera recursiva.

$$V^*(s_t) = \max_{a_t} \mathbb{E}_{a \sim \pi, s_{t+1} \sim \mathcal{P}} [r(s, a) + \gamma V^*(s_{t+1})] \quad (2.2.27)$$

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathcal{P}} [r(s, a) + \gamma \max_{a_t} [Q^*(s_{t+1}, a_{t+1})]] \quad (2.2.28)$$

Las dos anteriores ecuaciones son útiles para llevar a cabo el proceso de entrenamiento de un agente de manera eficiente, dado que permiten calcular estimados de las funciones de valor óptimas de manera recursiva.

2.2.11. Dilema Exploración-Explotación

Tanto la política como las funciones de valor deben ser aprendidas a lo largo de un proceso de interacción con el entorno; a medida que se genera un mayor número de trayectorias usando la política, el valor esperado del retorno de las acciones escogidas va mejorando.

Sin embargo, dado que el agente desconoce la función de transición del entorno y la función de recompensa, no existe manera de saber si las acciones generadas por la política son realmente aproximaciones a las acciones óptimas o si existen otras acciones que llevan a un mayor valor esperado del retorno.

Esto se puede presentar especialmente en entornos donde el espacio de estados y el espacio de acciones son grandes, ya que pueden existir pares estado-acción sin descubrir o que se observan muy pocas veces, pero que llevan a valores de retorno substancialmente más altos que aquellos aprendidos por el agente.

Generar dichos pares de estado-acción nuevos, incluso si no se tiene conocimiento de sus consecuencias se denomina explorar el entorno, mientras que utilizar lo ya aprendido por la política para obtener valores de retorno conocidos se denomina explotar el conocimiento adquirido; se denomina el dilema exploración-explotación a la búsqueda de un balance entre explorar y explotar [50]:

- Demasiada exploración, no permite al agente converger hacia una política óptima, puesto que no permite al agente usar su conocimiento para generar experiencias que le permitan generalizar a partir de lo ya aprendido.
- Demasiada explotación, no permite al agente probar nuevas acciones en diferentes estados, probablemente estancándolo en una política que aparenta ser optima, pero que no lo es, por falta de descubrimiento de nuevos o diferentes pares estado-acción.

2.2.12. Clasificación de los Algoritmos de Aprendizaje Reforzado Profundo

Existen diferentes tipos de algoritmos usados para entrenar a un agente dentro el aprendizaje reforzado profundo. Estos pueden ser clasificados principalmente de acuerdo a dos criterios, los cuales se describen en esta subsección.

2.2.12.1. De Acuerdo a la Función Aprendida

La clasificación de acuerdo a la función aprendida divide los algoritmos de entrenamiento de los agentes en tres grupos principales:

- **Algoritmos basados en política:** Los algoritmos de agentes basados en política se dedican exclusivamente al aprendizaje directo de un aproximador a una política óptima $\pi^*(s)$.
- **Algoritmos basados en valor:** Los algoritmos de agentes basados en valor se dedican a aprender aproximaciones a una o ambas funciones de valor optimas $V^*(s)$ $Q^*(s, a)$; estas son utilizadas posteriormente por el agente para elegir las acciones y generar la política.
- **Algoritmos actor-crítico:** Los algoritmos de agentes basados en métodos actor-crítico son algoritmos con un enfoque híbrido, que se dedican a aprender aproximaciones tanto a funciones de valor optimas como a la política óptima. Su nombre se debe a que mientras que la política realiza elecciones y ejecuta acciones sobre el entorno (es decir actúa sobre este), las

funciones de valor informan al agente acerca de su rendimiento y las consecuencias a futuro de dichas acciones (es decir, proveen una crítica de las acciones tomadas por el actor).

2.2.12.2. De Acuerdo a la Manera en la que se Generan los Datos

Es posible también clasificar los algoritmos de entrenamiento de los agentes de acuerdo a la manera en la cual se deben generar los datos para entrenar las redes neuronales, dividiéndolos en dos grupos principales:

- **Algoritmos en-política:** Dada una política π^m , si el algoritmo de entrenamiento permite utilizar solamente datos de trayectorias generadas utilizando π^m para entrenarla, entonces el algoritmo se clasifica como un algoritmo en-política.
- **Algoritmos fuera-política:** Dada una política π^m , si el algoritmo permite utilizar datos de trayectorias generadas usando cualquier otra política o incluso un set de otras políticas $\pi^1, \pi^2, \dots, \pi^n$ para entrenar la política π^m , entonces se clasifica como un algoritmo fuera-política.

2.2.13. Gradientes Profundos de Política Determinística

Gradientes Profundos de Políticas Determinística (DDPG Deep Deterministic Policy Gradients por su sigla en inglés) es un algoritmo para entrenar agentes de tipo actor-criticó de manera fuera-política. Propone una manera de optimizar una política determinística aprendiendo la función de valor de estado-acción simultáneamente. El algoritmo DDPG está orientado a entornos donde el espacio de acción es continuo [51].

DDPG utiliza esencialmente dos tipos de redes neuronales para aproximar dos tipos de funciones, como se describe a continuación.

- **Red Actor:** Es una red neuronal representada por $\mu^\phi(s)$ con parámetros ϕ que busca aproximar una política óptima.
- **Red Crítico:** Es una red neuronal representada por $Q^\beta(s, a)$ con parámetros β que busca aproximar una función de valor de estado-acción óptima.

A continuación, se describen los componentes principales que permiten a DDPG llevar a cabo el entrenamiento de ambas redes.

2.2.13.1. Búfer de Reproducción

Dado que DDPG es un método fuera-política, este puede utilizar experiencias obtenidas a lo largo de varios episodios bajo distintas políticas, lo cual hace posible almacenar y recuperar la información de dichas experiencias en cualquier momento para entrenar la política, aumentando la cantidad de datos disponibles e incrementando la eficiencia del algoritmo en relación a los datos que requiere.

Para el propósito de almacenar y recuperar las experiencias se utiliza un búfer de reproducción \mathcal{D} , el cual almacena tuplas de la forma $(s_t, a_t, r_{t+1}, s_{t+1}, f)$. Dichas tuplas representan experiencias individuales, con una variable extra denominada f , encargada de indicar si dentro cada experiencia se alcanzó el final de un episodio.

Para entrenar al agente, se toma del búfer de reproducción un lote \mathcal{B} , el cual contiene m experiencias tomadas de manera aleatoria. El propósito del muestreo aleatorio es proveer al agente un lote de experiencias que no se encuentran correlacionadas temporalmente entre sí, de manera que el agente pueda generalizar la función de valor de estado-acción y la política sin tomar en cuenta las relaciones temporales entre experiencias. El entero m se considera un hiperparámetro de DDPG.

2.2.13.2. Aprendizaje de la Función de Valor de Estado-Acción

Para el aprendizaje de una aproximación a la función de valor de estado-acción, se usa una red neuronal denominada $Q^\beta(s, a)$ con parámetros β . Se asume que dicha red neuronal es diferenciable con respecto a la acción, por lo cual es posible utilizar todas las herramientas del aprendizaje profundo para aprender la aproximación.

Al tratarse de un algoritmo en el cual el espacio de acciones es continuo, la búsqueda del valor $\max_{a_t}[Q^*(s_{t+1}, a_{t+1})]$ en la ecuación (2.2.28) no es computacionalmente eficiente, en especial en espacios de estado de dimensiones altas. En su lugar, los autores de DPPG proponen la ecuación (2.2.29), mediante la cual se usa la política para aproximar dicho valor [51].

$$\max_{a_t}[Q^*(s_{t+1}, a_{t+1})] \approx Q^*(s_{t+1}, \pi(s_{t+1})) \quad (2.2.29)$$

Reemplazando la ecuación (2.2.29) en (2.2.28), se obtiene una nueva expresión que permite aproximar recursivamente la función de valor de estado-acción dentro el proceso de aprendizaje, como se muestra a continuación.

$$Q^\beta(s_t, a_t) \approx r(s, a) + \gamma Q^\beta(s_{t+1}, \pi(s_{t+1})) \quad (2.2.30)$$

Para entrenar $Q^\beta(s_t, a_t)$, se utiliza como función de perdida el error cuadrático medio entre el valor esperado del retorno estimado para s_t y a_t obtenido directamente de la red neuronal, y la aproximación obtenida mediante la ecuación (2.2.30). La función de perdida es calculada usando un lote de experiencias tomado del búfer de reproducción como se muestra en la ecuación (2.2.31).

$$\mathcal{L} = \mathbb{E}_{(s_t, a_t, r_{t+1}, s_{t+1}) \sim \mathcal{B}}[Q^\beta(s_t, a_t) - (r(s, a) + \gamma Q^\beta(s_{t+1}, \pi(s_{t+1})))]^2 \quad (2.2.31)$$

Considerando que cada vez que se alcanza un final de episodio, el valor de (2.2.30) es exactamente igual a la recompensa, ya que en todo estado final no existe recompensa futura ni retorno, la ecuación (2.2.32) es una expresión que toma en cuenta la variable f de las tuplas de experiencia para calcular la función de perdida.

$$\mathcal{L} = \begin{cases} \mathbb{E}_{(s_t, a_t, r_{t+1}, s_{t+1}, f) \sim \mathcal{B}}[Q^\beta(s_t, a_t) - (r(s, a) + \gamma Q^\beta(s_{t+1}, \pi(s_{t+1})))]^2; f = 0 \\ \mathbb{E}_{(s_t, a_t, r_{t+1}, s_{t+1}, f) \sim \mathcal{B}}[Q^\beta(s_t, a_t) - (r(s, a))]^2; f = 1 \end{cases} \quad (2.2.32)$$

Dicha ecuación expresa una función de perdida que al ser minimizada usando métodos de DL, ocasiona que el agente aprenda una aproximación a la función de valor estado-acción, usando las experiencias obtenidas a lo largo de varios episodios que se encuentran almacenados dentro del búfer de reproducción.

Tomando la media sobre todos los datos presentes en el lote \mathcal{B} como una aproximación a los valores esperados, entonces se obtiene ecuación (2.2.33), la cual es una expresión computable para la función de perdida de la red de valor de estado-acción.

$$\mathcal{L} = \begin{cases} \frac{1}{|\mathcal{B}|} * \sum_{(s_t, a_t, r_{t+1}, s_{t+1}, f) \sim \mathcal{B}} [Q^\beta(s_t, a_t) - (r(s, a) + \gamma Q^\beta(s_{t+1}, \pi(s_{t+1})))]^2; f = 0 \\ \frac{1}{|\mathcal{B}|} * \sum_{(s_t, a_t, r_{t+1}, s_{t+1}, f) \sim \mathcal{B}} [Q^\beta(s_t, a_t) - (r(s, a))]^2; f = 1 \end{cases} \quad (2.2.33)$$

2.2.13.3. Redes Objetivo

Se observa en la función de perdida (2.2.33), que se utiliza los mismos parámetros β de la red neuronal que se desea entrenar para generar el término $Q^\beta(s_{t+1}, \pi(s_{t+1}))$; esto puede llevar a que el proceso de entrenamiento sea inestable.

Como solución a esto, los autores de DDPG idearon que para generar dicho término se utilice una segunda red neuronal denominada red objetivo $Q^{\beta obj}$, la cual posee parámetros βobj . Dichos parámetros son cercanos, pero no iguales a β [51].

Los parámetros de la red objetivo $Q^{\beta obj}$ son actualizados con valores cercanos a los de la red neuronal Q^β después de un paso de entrenamiento. Para esto se utiliza la media de Polyak según la siguiente expresión.

$$\beta_{obj} \leftarrow \rho \beta_{obj} + (1 - \rho) \beta \quad (2.2.34)$$

De igual manera, se usa una red neuronal objetivo para la red política, introduciendo una segunda red neuronal $\mu^{\varphi obj}(s)$, la cual posee parámetros φobj , que son actualizados mediante la media de Polyak como se expresa a continuación.

$$\varphi obj \leftarrow \rho \varphi obj + (1 - \rho) \varphi \quad (2.2.35)$$

Se usa el hiperparámetro $\rho \in [0, 1]$ en la actualización de las redes neuronales mediante la media de Polyak.

2.2.13.4. Aprendizaje de la Política

La política utilizada en un agente DDPG es una política determinística, es decir que esta es un mapeo directo de la entrada estado s a una acción a como se muestra en la siguiente ecuación:

$$a = \mu^\varphi(s) \quad (2.2.36)$$

Dado que la red neuronal Q^β es una aproximación del retorno esperado, los autores de DDPG proponen la ecuación (2.2.37) como una modificación a la función objetivo expresada en la ecuación (2.2.21). De esta manera, se expresa la función objetivo en términos del estimado de la función de valor de estado-acción [51].

$$J(\varphi, \mathcal{B}) = \mathbb{E}_{(s_t) \sim \mathcal{B}} [Q^\beta(s_t, \mu^\varphi(s))] \quad (2.2.37)$$

En la ecuación (2.2.37), se toma la media sobre todos los valores obtenidos a partir del lote \mathcal{B} como una aproximación al valor esperado, entonces se obtiene la ecuación (2.2.38) como una expresión computable para la función objetivo de la política.

$$J(\varphi, \mathcal{B}) = \frac{1}{|\mathcal{B}|} * \sum_{(s_t) \sim \mathcal{B}} Q^\beta(s_t, \mu^\varphi(s)) \quad (2.2.38)$$

La función objetivo expresada en la ecuación (2.2.38) es maximizada usando algoritmos y técnicas de DL durante el proceso de entrenamiento.

2.2.13.5. Mitigación del Dilema Exploración-Explotación

Dado que se usa una política determinística, el dilema de exploración-explotación se hace particularmente aparente en el caso de DDPG, ya que no existe manera de generar nuevas acciones usando la red neuronal $\mu^\varphi(s)$ de manera directa.

Para añadir exploración a las acciones generadas por el agente, los autores de DDPG propusieron añadir ruido tomado de una distribución normal estándar con media 0 y desviación estándar 1 \mathcal{N} a las acciones generadas por la red neuronal μ^φ . Hacer esto permite generar una nueva acción que se desvía levemente de aquella elegida por la política, ayudando al agente a explorar el espacio de estados y acciones, según la ecuación (2.2.39) [51].

$$a_{entorno} = \mu^\varphi(s) + e_{ruido}\epsilon; \epsilon \sim \mathcal{N} \quad (2.2.39)$$

La variable e_{ruido} escala el valor del ruido tomado de la distribución normal estándar, de manera que el valor que la acción generada $a_{entorno}$ no sea muy alejado al valor obtenido de la política.

2.2.13.6. Algoritmo de DDPG

Tomando en cuenta los componentes anteriores, a continuación, se procede a detallar el algoritmo de DDPG a continuación.

Algoritmo 4 DDPG

Entradas: Red Neuronal Actor μ inicializada, Red neuronal Critico Q inicializada, Búfer de Reproducción D , Tasa de Descuento γ , Parámetro ρ Tamaño de Lote m , Numero de Episodios $n_{episodios}$, Numero de pasos de entrenamiento por episodio $n_{pasosentrenamiento}$

- 1: Inicializar redes neuronales objetivo $\beta_{obj} \leftarrow \beta$, $\varphi_{obj} \leftarrow \varphi$
- 2: **repetir**
- 3: Obtener el estado s_t del entorno
- 4: Seleccionar una acción usando $a = \mu^\varphi + e_{ruido}\mathcal{N}$
- 5: Obtener el estado siguiente del entorno s_{t+1} , la recompensa r_{t+1} y la señal que indica si se ha alcanzado un estado terminal f
- 6: Almacenar la tupla $(s_t, a_t, r_{t+1}, s_{t+1}, f)$ en el búfer de reproducción D
- 7: Si se alcanzo un estado terminal, reiniciar el entorno
- 8: **repetir**
- 9: Tomar un lote B de m experiencias aleatorias del búfer de reproducción D
- 10: Calcular $Q_{obj} = Q^{\beta_{obj}}(s_{t+1}, \mu^{\varphi_{obj}}(s_{t+1}))$

```

11:   si  $f = 1$  en una tupla  $(s_t, a_t, r_{t+1}, s_{t+1}, f)$  entonces
12:      $Q_{obj} = 0$  para esa tupla
13:   fin de si
14:   Calcular la función de perdida de la red de valor de estado-acción:
     $L = \frac{1}{|B|} \sum_{(s_t, a_t, r_{t+1}, s_{t+1}, f) \sim B} [Q^\beta(s_t, a_t) - (r_{t+1} + \gamma Q_{obj})]^2$ 
15:   Calcular el gradiente  $\nabla L$  y realizar un paso de entrenamiento sobre la
    red de valor de estado acción  $Q^\beta$ 
16:   Calcular la función objetivo de la red política:
     $J = \frac{1}{|B|} \sum_{s_t \sim B} Q^\beta(s_t, \mu^\varphi(s_t))$ 
17:   Calcular el gradiente  $\nabla J$  y realizar un paso de entrenamiento sobre la
    red política  $\mu^\varphi$ 
18:   Actualizar los parámetros a las redes objetivo usando la media de Polyak:
     $\beta_{obj} \leftarrow \rho\beta_{obj} + (1 - \rho)\beta, \varphi_{obj} \leftarrow \rho\varphi_{obj} + (1 - \rho)\varphi$ 
19:   hasta haber completado un numero de pasos de entrenamiento
     $n_{pasosentrenamiento}$ 
20: hasta haber completado un numero de episodios  $n_{episodios}$ 

```

2.2.14. Gradientes Profundos Gemelos y Atrasados de Política Determinística

Gradientes Profundos Gemelos y Atrasados de Política Determinística (TD3, Twin Delayed Deep Deterministic Policy Gradients por su sigla en inglés) es un algoritmo propuesto como una mejora a DDPG.

En la práctica, se encontró que DDPG es un algoritmo excesivamente sensible a variaciones de los hiperparámetros, esto implica que cambios pequeños en parámetros como la tasa de descuento γ o el parámetro ρ utilizado para actualizar las redes objetivo puede llevar a cambios drásticos en el rendimiento del agente [52].

Los autores de TD3 identificaron el problema principal que ocasiona dicho comportamiento al usar DDPG; existe una tendencia de sobreestimar los valores de estado-acción usando la red Q^β a lo largo del proceso de entrenamiento. Dado que la función objetivo de la política es directamente proporcional al valor de Q^β , el problema de sobreestimación provoca valores demasiado grandes en los gradientes de la política, los cuales pueden desencadenar un proceso de entrenamiento inestable que incluso puede provocar divergencia de una política adecuada.

Los autores de TD3 implementaron una serie de modificaciones sobre la base de DDPG que se demostraron efectivas en mejorar el rendimiento y la convergencia, dichas modificaciones se describen a continuación.

2.2.14.1. Suavizado de Acción Objetivo

Como se mencionó previamente, es posible que en DDPG la red de valor de estado-acción Q^β sobreestime sus valores; se hallaron casos en los cuales se generan picos exageradamente altos alrededor de una o más determinadas acciones en la función de valor de estado-acción estimada.

Esto ocasiona que el gradiente de la política tenga picos aún más altos alrededor de dichas acciones; como consecuencia, se observó que la política tiende a desarrollar comportamientos

incorrectos, los cuales enfatizan la elección de esas acciones, dado que durante el proceso de entrenamiento se estima que el retorno obtenido de elegirlas es incorrectamente alto.

Los autores de TD3 idearon el suavizado de acción objetivo como un método para prevenir este comportamiento durante el entrenamiento. El suavizado de acción objetivo implica añadir ruido para reducir la posibilidad que se desarrollen picos ya mencionados sobre acciones específicas, obligando al actor a generalizar sobre acciones similares [53].

Las acciones generadas utilizando la red objetivo $\mu^{\phi obj}$ son perturbadas con un ruido representado por ϵ , tomado de una distribución normal, limitado entre los valores máximo y mínimo c_{max} y c_{min} . Seguidamente, se limita las acciones objetivo con ruido añadido entre sus valores a_{max} y a_{min} si estos existiesen, ya que al añadir ruido existe la posibilidad que dichos límites sean sobrepasados. La ecuación (2.2.40) es usada para generar las acciones objetivo en TD3 y se muestra a continuación.

$$a_{t+1}^{obj} = \text{clip}(\mu^{\phi obj}(s_{t+1}) + \text{clip}(\epsilon, c_{max}, c_{min}), a_{max}, a_{min}); \epsilon \sim \mathcal{N} \quad (2.2.40)$$

2.2.14.2. Aprendizaje de Q Doble Limitado

La segunda estrategia propuesta por los autores de TD3 para mitigar la sobreestimación de la función de valor de estado-acción, es la utilización de dos redes neuronales para aproximar dos funciones de estado de valor-acción simultáneas, las cuales son denominadas $Q^{\beta 1}(s, a)$ y $Q^{\beta 2}(s, a)$. Cada una posee una red objetivo $Q_1^{\beta obj}$ y $Q_2^{\beta obj}$ [53].

Se obtiene estimados de valor de estado-acción de ambas redes objetivo, se compara los resultados obtenidos y se selecciona el mínimo estimado $\min_Q(s_{t+1})$ según la ecuación que se muestra a continuación.

$$\min_Q(s_{t+1}) = \min_{i=1,2} Q_i^{\beta obj}(s_{t+1}, a_{t+1}^{obj}) \quad (2.2.41)$$

El estimado mínimo del valor de estado-acción es usado en dos funciones de pérdida usando el error mínimo cuadrático; cada una corresponde a cada red Q, como se muestra en la ecuación (2.2.42).

$$\mathcal{L}_i = \begin{cases} \frac{1}{|\mathcal{B}|} * \sum_{(s_t, a_t, r_{t+1}, s_{t+1}, f) \sim \mathcal{B}} \left[Q_i^\beta(s_t, a_t) - (r(s, a) + \gamma \min_Q(s_{t+1})) \right]^2; f = 0 \\ \frac{1}{|\mathcal{B}|} * \sum_{(s_t, a_t, r_{t+1}, s_{t+1}, f) \sim \mathcal{B}} \left[Q_i^\beta(s_t, a_t) - r \right]^2; f = 1 \end{cases} \quad (2.2.42)$$

i = 1, 2 para cada una de las redes Q.

Seguidamente, se calcula una pérdida global para entrenar ambas redes de valor de estado-acción, como se expresa en la ecuación (2.2.43).

$$\mathcal{L}_{criticos} = \mathcal{L}_1 + \mathcal{L}_2 \quad (2.2.43)$$

Se usa dicha función de perdida global para calcular los gradientes para cada red de valor de estado-acción y poder entrenarlas.

La política es entrenada usando la función objetivo mostrada en la ecuación (2.2.44), la cual es similar a aquella usada en DDPG, pero usando la red de valor de estado-acción número 1 por elección de los autores de TD3 [53].

$$J(\varphi, \mathcal{B}) = \frac{1}{|\mathcal{B}|} * \sum_{(s_t) \sim \mathcal{B}} Q_1^\beta(s_t, \mu^\varphi(s)) \quad (2.2.44)$$

2.2.14.3. Actualización de la Política con Retraso

Finalmente, con el propósito de evitar que los cambios en la política sean demasiado bruscos, los autores de TD3 idearon que se realice los pasos de entrenamiento de la política y la actualización de las redes objetivo usando un retraso con respecto a los entrenamientos de las redes de valor de estado-acción.

Para generar el retraso, se utiliza un hiperparámetro propio de TD3 p_{retard} , el cual especifica cuantos pasos de entrenamiento de las redes de valor se deben realizar, antes de poder realizar un paso de entrenamiento de la red política y la actualización de las redes objetivo.

Por ejemplo, se puede realizar un paso de entrenamiento de la red actor y actualización de las redes objetivo cada 3 pasos de entrenamiento de las redes de valor de estado-acción, caso para el cual $p_{retard} = 3$.

2.2.14.4. Algoritmo de TD3

Tomando en cuenta las modificaciones anteriores, se procede a detallar el algoritmo de TD3 en el algoritmo 5.

Algoritmo 5 TD3

Entradas: Red Neuronal Actor μ^φ inicializada, Redes neuronales Crítico Q_1^β, Q_2^β inicializadas, Búfer de Reproducción D , Tasa de Descuento γ , Parámetro ρ , Pasos de retraso del entrenamiento de la red política p_{retard} , Tamaño de Lote m , Numero de Episodios $n_{episodios}$, Numero de pasos de entrenamiento por episodio $n_{pasosentrenamiento}$

- 1: Inicializar redes neuronales objetivo $\beta obj1 \leftarrow \beta 1, \beta obj2 \leftarrow \beta 2, \varphi obj \leftarrow \varphi$
- 2: **repetir**
- 3: Obtener el estado s_t del entorno
- 4: Seleccionar una acción usando $a = \mu^\varphi + e_{ruido}N$
- 5: Obtener el estado siguiente del entorno s_{t+1} , la recompensa r_{t+1} y la señal que indica si se ha alcanzado un estado terminal f
- 6: Almacenar la tupla $(s_t, a_t, r_{t+1}, s_{t+1}, f)$ en el búfer de reproducción D
- 7: Si se alcanzo un estado terminal, reiniciar el entorno
- 8: **repetir**
- 9: Inicializar contador de pasos de entrenamiento $p_{ent} = 0$
- 10: Tomar un lote B de m experiencias aleatorias del búfer de entrenamiento D
- 11: Calcular las acciones objetivo:

$$a_{t+1}^{obj} = clip(\mu^{\varphi obj}(s_{t+1}) + clip(\epsilon, c_{max, c_{min}}), a_{max}, a_{min}); \epsilon \sim N$$

- 12: Calcular el mínimo entre ambas redes neuronales objetivo:
 $\min_Q(s_{t+1}) = \min_{i=1,2} Q_i^{\beta obj}(s_{t+1}, a_{t+1}^{obj})$
- 13: si $f = 1$ en una tupla $(s_t, a_t, r_{t+1}, s_{t+1}, f)$ entonces
 $\min_Q(s_{t+1}) = 0$ para esa tupla
- 15: fin de si
- 16: Calcular la función de perdida para cada red de valor de estado-acción:
 $L_{1,2} = \frac{1}{|B|} \sum_{(s_t, a_t, r_{t+1}, s_{t+1}, f) \in B} [Q_{1,2}^\beta(s_t, a_t) - (r_{t+1} + \gamma \min_Q(s_{t+1}))]^2$
- 17: Calcular la función de perdida global de las redes de valor de estado-acción: $L_{criticos} = L_1 + L_2$
- 18: Calcular el gradient $\nabla L_{criticos}$ y realizar un paso de entrenamiento sobre cada red de valor de estado acción $Q_{1,2}^\beta$
- 19: si Si $p_{ent} \bmod p_{retard} = 0$ entonces
 Calcular la función objetivo de la red política:
 $J = \frac{1}{|B|} \sum_{s_t \sim B} Q_1^\beta(s_t, \mu^\varphi(s_t))$
- 21: Calcular el gradiente ∇J y realizar un paso de entrenamiento sobre la red política μ^φ
- 22: Actualizar los parámetros a las redes objetivo usando la media de Polyak:
 $\beta obj1, 2 \leftarrow \rho \beta obj1, 2 + (1 - \rho) \beta1, 2, \varphi obj \leftarrow \rho \varphi obj + (1 - \rho) \varphi$
- 23: fin de si
- 24: hasta haber completado un numero de pasos de entrenamiento $n_{pasosentrenamiento}$
- 25: hasta haber completado un numero de episodios $n_{episodios}$
-

2.3. Dinámica de un Péndulo Invertido

En la presente sección se realiza una justificación teórica de las propiedades que caracterizan al péndulo invertido como un sistema que propone un desafío para el desarrollo de controladores. Para dicho propósito, se desarrolla un modelo matemático a partir de un esquema simplificado del péndulo invertido, seguidamente se usa la teoría de control para poder caracterizar sus propiedades.

Un esquema simplificado de la forma general de un péndulo invertido montado sobre un carro se muestra en la ilustración 16.

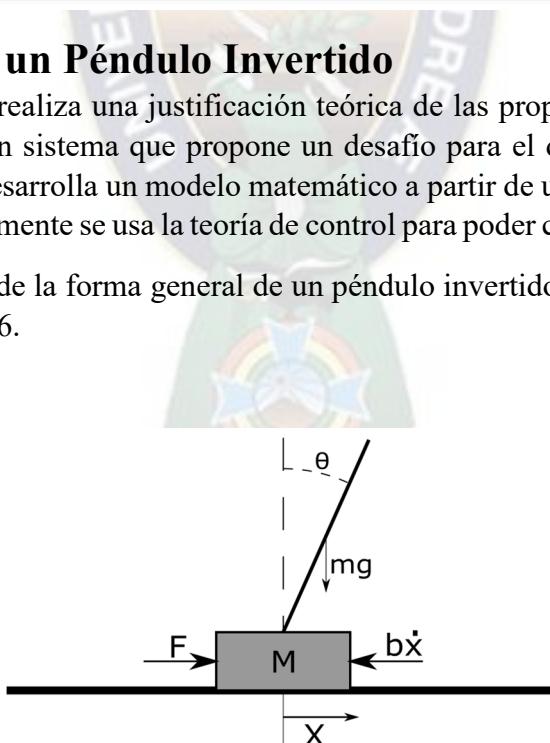


Ilustración 16 Esquema simplificado de un péndulo invertido. Fuente: Elaboración propia.

Para realizar un modelo matemático del péndulo invertido sobre un carro, se toma en cuenta sus parámetros físicos, listados a continuación.

M: Masa del carro [kg].

m: Masa del péndulo [kg].

l: Longitud de la base del péndulo a su centro de masa [m].

b: Coeficiente de fricción entre el carro y la pista.

g: Fuerza de gravedad $\left[\frac{m}{s^2}\right]$.

También, se usa las variables listadas a continuación para describir el estado completo del péndulo invertido sobre un carro.

x: Distancia del centro de la pista a la posición del carro.

\dot{x} : Velocidad del carro.

θ : Ángulo del péndulo con respecto a la vertical.

$\dot{\theta}$: Velocidad angular del péndulo.

Se deriva su modelo matemático tomando la posición del carro x y el ángulo del péndulo θ como grados de libertad. Se inicia hallando el LaGrangiano L del péndulo invertido.

$$L = T - V \quad (2.3.1)$$

$$T = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}mv^2 \text{ Energía cinética del sistema} \quad (2.3.2)$$

Para hallar la velocidad v del centro de masa del péndulo y la energía potencial del mismo, se realiza el análisis mostrado en la ilustración 17, consiguiendo las tres ecuaciones mostradas a continuación.

$$v^2 = \dot{x}_m^2 + \dot{y}_m^2 \quad (2.3.3)$$

$$x_m = x + l \sin \theta \rightarrow \dot{x}_m = \dot{x} + l\dot{\theta} \cos \theta \quad (2.3.4)$$

$$y_m = l \cos \theta \rightarrow \dot{y}_m = -l\dot{\theta} \sin \theta \quad (2.3.5)$$

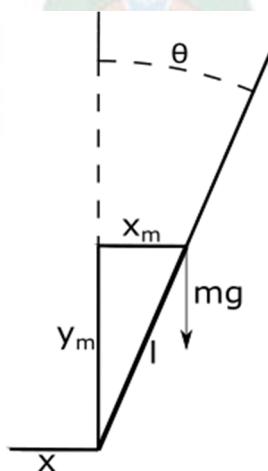


Ilustración 17 Análisis del centro de masa del péndulo invertido. Fuente: Elaboración propia.

Reemplazando (2.3.4) y (2.3.5) en (2.3.3), luego simplificando se obtiene ecuaciones (2.3.6) y (2.3.7).

$$\begin{aligned} v^2 &= \frac{1}{2}m\dot{x}^2 + \frac{1}{2}ml^2\dot{\theta}^2 + ml\dot{x}\dot{\theta} \cos \theta \rightarrow \\ T &= \frac{1}{2}(M+m)\dot{x}^2 + \frac{1}{2}ml^2\dot{\theta}^2 + ml\dot{x}\dot{\theta} \cos \theta \quad (2.3.6) \\ V &= mgy_m = mgl \cos \theta \quad (2.3.7) \end{aligned}$$

Para obtener la expresión completa del LaGrangiano, se reemplaza (2.3.6) y (2.3.7) en (2.3.1), obteniendo la ecuación (2.3.8).

$$L = \frac{1}{2}(M+m)\dot{x}^2 + \frac{1}{2}ml^2\dot{\theta}^2 + ml\dot{x}\dot{\theta} \cos \theta - mgl \cos \theta \quad (2.3.8)$$

Para hallar el sistema de ecuaciones que modelan el péndulo invertido, se aplica las ecuaciones de Euler-Lagrange (Ecuaciones (2.3.9) a (2.3.12)) a los dos grados de libertad del sistema.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = Q_x \quad (2.3.9)$$

$$Q_x = F - b\dot{x} \quad (2.3.10)$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = Q_\theta \quad (2.3.11)$$

$$Q_\theta = 0 \quad (2.3.12)$$

Reemplazando (2.3.8) y (2.3.10) en (2.3.9), (2.3.8) y (2.3.12) en (2.3.11), realizando las operaciones de diferenciación correspondientes y luego simplificando, se obtiene el sistema de ecuaciones diferenciales que modelan al péndulo invertido, mostrado en (2.3.13).

$$\begin{cases} (M+m)\ddot{x} + ml\ddot{\theta} \cos \theta - ml\dot{\theta}^2 \sin \theta + b\dot{x} = F \\ l\ddot{\theta} + \ddot{x} \cos \theta - g \sin \theta = 0 \end{cases} \quad (2.3.13)$$

A partir del modelo derivado, se observa que existen términos en el sistema de ecuaciones diferenciales (2.3.13) conformados por funciones no lineales del estado, tales como $ml\dot{\theta}^2 \sin \theta$ o $g \sin \theta$. De esa observación, sobre la base del modelo matemático derivado, se afirma que el péndulo invertido sobre un carro es un sistema no-lineal.

Prosiguiendo, la teoría de Lyapunov menciona que:

“Las propiedades de estabilidad de un sistema no-lineal en la cercanía de sus puntos de equilibrio, son esencialmente las mismas que de su aproximación linealizada. [54]”

El péndulo invertido posee dos puntos de equilibrio; el primero describe la posición en la cual el péndulo se encuentra en la posición vertical hacia abajo, para la cual se tiene un ángulo $\theta = \pi$, mientras que el segundo describe la posición vertical hacia arriba hacia arriba con un ángulo $\theta = 0$ [55].

El segundo punto de equilibrio $\boldsymbol{\theta} = \mathbf{0}$ es de particular interés, ya que las tareas de control del péndulo invertido buscan llevarlo y mantenerlo en cercanía de dicho punto. Por lo tanto, para hallar las propiedades de estabilidad del péndulo invertido en proximidad ese punto, primeramente, se linealiza el modelo previamente desarrollado, obteniendo el sistema de ecuaciones (2.3.14).

$$\begin{cases} (\mathbf{M} + \mathbf{m})\ddot{x} + \mathbf{ml}\ddot{\theta} + \mathbf{b}\dot{x} = F \\ \mathbf{l}\ddot{\theta} + \dot{x} - \mathbf{g}\theta = 0 \end{cases} \quad (2.3.14)$$

Realizando operaciones algebraicas entre las ecuaciones del sistema, se las puede expresar en términos de las segundas derivadas de x y de θ , obteniendo el sistema (2.3.15).

$$\begin{cases} \ddot{x} = -\frac{b}{M}\dot{x} - \frac{mg}{M}\theta + \frac{F}{M} \\ \ddot{\theta} = \frac{b}{Ml}\dot{x} + \frac{(M+m)g}{Ml}\theta - \frac{F}{Ml} \end{cases} \quad (2.3.15)$$

Del anterior sistema, se obtiene el modelo lineal en el espacio de estados (2.3.16), mostrado a continuación.

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{b}{M} & -\frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{b}{Ml} & \frac{(M+m)g}{Ml} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ -\frac{1}{Ml} \end{bmatrix} F \quad (2.3.16)$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} F$$

La estabilidad de todo sistema en el espacio de estados lineal puede definirse en términos de los autovalores de la matriz de transición de estados. Si todos los autovalores de dicha matriz poseen parte real negativa, entonces el sistema es estable. De manera conversa, para que el sistema sea considerado inestable, basta que uno de los autovalores posea parte real no negativa [56].

El polinomio característico de la matriz de transición de estado del modelo lineal en el espacio de estados (2.3.16) es:

$$\lambda^4 + \lambda^3 \frac{\mathbf{b}}{\mathbf{M}} - \lambda^2 \frac{(\mathbf{M} + \mathbf{m})\mathbf{g}}{\mathbf{Ml}} - \lambda \frac{(\mathbf{M} + \mathbf{m})\mathbf{bg}}{\mathbf{M}^2 \mathbf{l}} + \lambda \frac{\mathbf{mbg}}{\mathbf{M}^2 \mathbf{l}} = 0 \quad (2.3.17)$$

Factorizando la ecuación (2.3.17) es posible hallar de manera rápida el primer autovalor, el cual es $\lambda_1 = \mathbf{0}$.

Dado que λ_1 es un valor no negativo, se concluye que el péndulo invertido es un sistema inestable en la posición vertical hacia arriba.

A partir de lo desarrollado en la presente sección se afirma sobre la base teórica de control lo mencionado en el capítulo 1; el péndulo invertido sobre un carro es un sistema no lineal e inestable, por lo cual es un banco de pruebas desafiante para los sistemas de control desarrollados a lo largo de los años.

2.4. Aspectos de Software para la Implementación de Sistemas de Aprendizaje Reforzado Profundo

En la presente sección, se presenta aspectos relacionados a la implementación de sistemas de aprendizaje reforzado profundo para su uso en el control de plantas reales.

2.4.1. Requerimientos de Tiempo de ejecución de Redes Neuronales en un Lazo de Control

Como se observó en la sección 2.2, las redes neuronales son un bloque esencial en todo sistema de aprendizaje reforzado profundo, ya que son usadas para aproximar tanto una política óptima como las funciones de valor de estado.

Dado que las redes neuronales estas son usadas para generar las acciones que se ejecutan en el entorno, es importante tomar en cuenta su implementación en software, en especial cuando son usadas para interactuar con entorno compuesto por sistemas reales tales como el péndulo invertido usado en el presente proyecto.

Por lo general, las observaciones del estado del entorno son tomadas con una tasa de muestreo predeterminada, denominada T_s . Por lo tanto, la acción debe ser generada usando la red neuronal en un tiempo razonablemente pequeño con relación a dicha tasa de muestreo, esto debe cumplirse de manera que sea posible ejecutar la acción y observar la respuesta del entorno a dicha acción antes que se lleve a cabo la siguiente toma de muestra del estado del entorno.

2.4.2. Análisis del tiempo de Ejecución de una Red Neuronal Feedforward

A continuación, se procede a analizar el tiempo de ejecución de una red neuronal Feedforward con una capa de entrada de dimensión $n^{[i]}$, en términos del número de capas total de la red neuronal K y el número de neuronas $n^{[k]}$ en cada capa k .

Considerando una capa número k , con $n^{[k-1]}$ entradas de una capa anterior y que posee $n^{[k]}$ neuronas, cada neurona realiza la operación de combinación lineal mencionada en la sección 2.1.1. Por lo tanto, cada neurona realiza una multiplicación de las $n^{[k-1]}$ entradas por los $n^{[k-1]}$ parámetros de peso presentes en cada neurona, de esto se afirma que se realizan $n^{[k-1]} * n^{[k]}$ multiplicaciones en la capa k .

De lo anterior, se generaliza que para una red neuronal feedforward con K capas, en la cual cada capa k posee un numero $n^{[k]}$ de neuronas, se realiza el número de multiplicaciones total expresado en la ecuación (2.4.1).

$$n^{[i]} * n^{[1]} + n^{[1]} * n^{[2]} + \dots n^{[K-1]} * n^{[K]} = n^{[i]} * n^{[1]} + \sum_{k=1}^{K-1} n^{[k]} * n^{[k+1]} \quad (2.4.1)$$

Obviando el tiempo empleado en realizar la suma de los parámetros de polarización en cada neurona y obviando también el tiempo que se toma en aplicar las funciones de activación, es posible representar una aproximación al tiempo de ejecución de la red neuronal denominado $t_{neuronal}$, si se toma como una constante denominada t_{mul} el tiempo de ejecución de una multiplicación en el procesador. La ecuación obtenida para representar $t_{neuronal}$ se muestra a continuación.

$$t_{neuronal} = \left(n^{[i]} * n^{[1]} + \sum_{k=1}^{K-1} n^{[k]} * n^{[k+1]} \right) t_{mul} \quad (2.4.2)$$

Se omite de la expresión el tiempo utilizado para llevar a cabo la aplicación de las funciones de activación de cada neurona, ya que como se mencionó en la sección 2.1.3, estas fueron elegidas para ser calculadas rápidamente, además que son aplicadas una sola vez por cada neurona.

Para ilustrar la manera en la que la ecuación (2.4.2) se comporta, se tomó como ejemplo un microprocesador de computador de escritorio de gama media Intel Core i5 6600K, el cual es capaz de realizar hasta $5.29 * 10^9$ operaciones de punto flotante por segundo (GFLOPS) [57], por lo tanto, $t_{mul} = 1.8903592 * 10^{-10}$.

Se generó una gráfica tomando en cuenta una arquitectura de red neuronal que tiene 10 entradas, dos capas escondidas y 10 neuronas de salida, variando el número de neuronas en cada capa desde 128 hasta 1152. El tiempo de ejecución en relación al número de neuronas por capa usando el microprocesador mencionado se muestra en la ilustración 18.

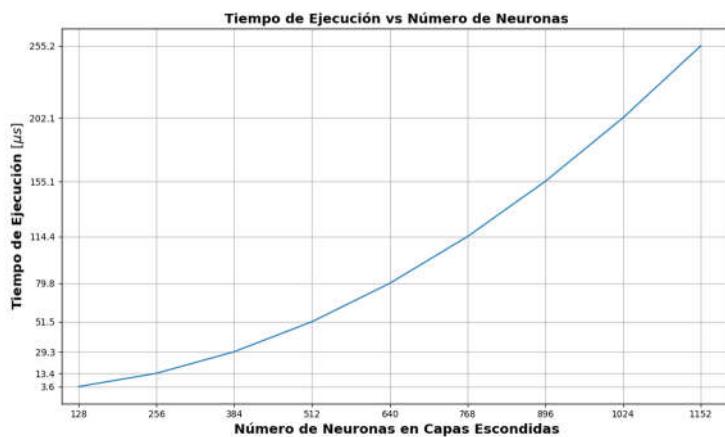


Ilustración 18 Tiempo de ejecución de la red neuronal vs número de neuronas. Fuente: Elaboración propia.

2.4.3. Aceleración de la Ejecución de las Redes Neuronales Mediante el Uso de GPUs.

Como se explicó en la sección anterior, las operaciones de multiplicación entre matrices que se llevan a cabo dentro una red neuronal para obtener sus resultados son aquellas que definen mayormente el tiempo de ejecución de una red neuronal.

Se mostró mediante la ilustración 18, que el tiempo de ejecución se hace substancialmente más largo en función del número de neuronas por capa; dicho tiempo puede incrementar de manera significativa si se eleva el número de capas de la red neuronal y aún más si se eleva el número de neuronas por capa.

Una manera de reducir el tiempo de ejecución es mediante computación paralela, ya que permite realizar operaciones entre matrices tales como multiplicación con un tiempo de ejecución significativamente menor, esto se logra dividiendo el problema en instancias de subproblemas más pequeños, que se pueden realizar de manera independiente y al mismo tiempo [58].

Las unidades de procesamiento de gráficos (GPUs), originalmente orientadas exclusivamente a la generación y procesamiento de gráficos en 3D en tiempo real, poseen cientos o miles de unidades capaces de realizar un número relativamente limitado de operaciones matemáticas, tales como la multiplicación.

Aprovechando estas capacidades presentes en ese hardware, la industria de la inteligencia artificial ha adoptado los GPU como una plataforma para acelerar el procesamiento de redes neuronales, ya que se puede aprovechar las unidades de procesamiento presentes dentro de un GPU para realizar las operaciones de multiplicación de matrices necesarias para redes neuronales en paralelo.

Para mostrar la aceleración que puede ser obtenida mediante el uso de GPUs, la ilustración 19 grafica el tiempo de ejecución de la red neuronal especificada en la sección 2.4.2 corriendo sobre un GPU de gama media de Nvidia, modelo GTX 1050, el cual es capaz de realizar hasta $1.84 * 10^{12}$ operaciones de punto flotante por segundo (GFLOPS) [59].

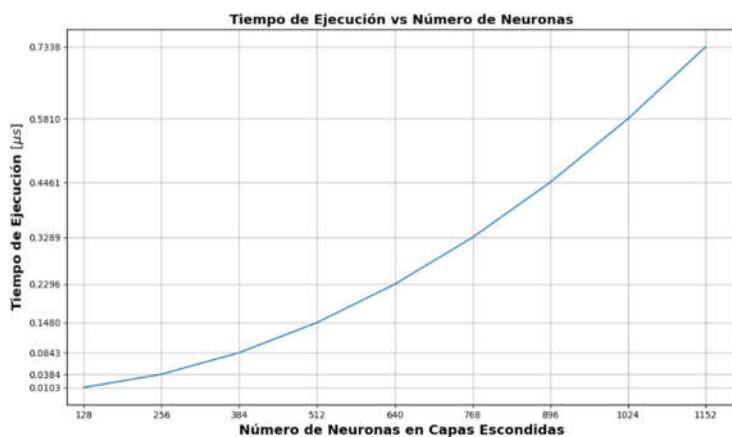


Ilustración 19 Tiempo de ejecución de la red neuronal en GPU. Fuente: Elaboración propia.

De la gráfica anterior, se observa que a partir de los cálculos realizados se obtiene en teoría un tiempo de ejecución 347 veces menor usando un GPU, razón por la cual es favorable el uso de GPUs en la implementación de sistemas de aprendizaje reforzado profundo, particularmente cuando estos están destinados a interactuar con sistemas reales y existen requerimientos de tiempo de ejecución.

Capítulo 3

Ingeniería del proyecto

3.1. Propuesta de la Arquitectura Para el Control Mediante Aprendizaje Reforzado Profundo del Péndulo Invertido

En la presente sección, se procede a detallar la arquitectura propuesta para controlar el péndulo invertido mediante aprendizaje reforzado profundo.

Primeramente, se propone una leve modificación al esquema básico de interacción entre el agente y el entorno, extendiéndolo con componentes destinados a facilitar la experimentación y obtención de datos para el análisis del rendimiento del agente a lo largo del proceso de entrenamiento.

3.1.1. Visión General

Como fue establecido en el capítulo 1, el objetivo del presente proyecto es el de realizar el control de un péndulo invertido sobre un carro; un sistema utilizado como banco de pruebas para la implementación de esquemas de control, mediante aprendizaje reforzado profundo. De esta manera, se busca demostrar que es posible aplicarlo al control de plantas reales, proponiendo una alternativa al flujo de diseño para sistemas de control mencionado en el mismo capítulo.

Se propuso y desarrolló una arquitectura modular que utiliza la programación orientada a objetos para separar de una manera clara y concisa los varios componentes del sistema de aprendizaje reforzado profundo, mediante la definición de diferentes clases que pueden ser instanciadas como objetos.

Dicha arquitectura fue ideada con la experimentación en mente; usando la programación orientada a objetos se pudo abstraer los diferentes módulos que componen al sistema, de manera que ellos se puedan cambiar y modificar como componentes individuales, los cuales pueden además ser instanciados con diferentes parámetros.

De lo anterior, puede destacar que la arquitectura resultante es modular y permite ser reutilizada para fines de experimentación futura usando tanto diferentes parámetros, como otro tipo de agentes, o incluso puede ser modificada para ser usada con entornos diferentes.

3.1.2. Esquema del sistema

La ilustración 15 mostrada en el capítulo anterior resume el esquema de interacción entre el agente y el entorno que se lleva a cabo en todo sistema de aprendizaje reforzado profundo. En el presente trabajo, se propuso una leve modificación a dicho esquema para lograr el modularidad ideada, añadiendo dos componentes extra como se muestra en la ilustración 20.

Dentro las siguientes secciones, se procede a describir a detalle cada uno de los componentes que componen la arquitectura ilustrada.

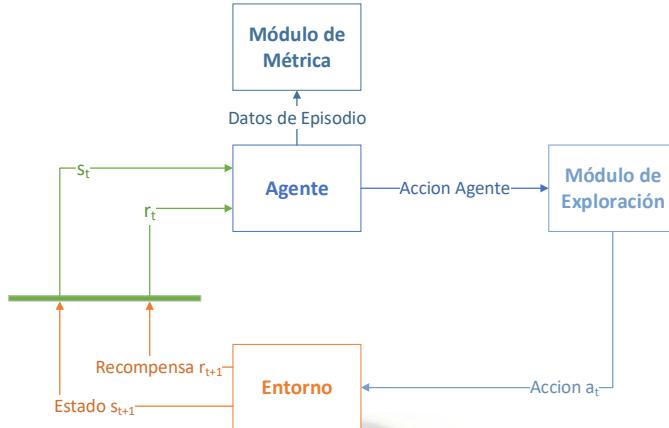


Ilustración 20 Esquema de interacción modificado. Fuente: Elaboración propia.

3.1.3. Agente

Se mencionó en el capítulo 2 que el agente es un ente puramente computacional. Dentro la arquitectura propuesta, es representado mediante una clase denominada Agente; dicha clase implementa una política determinística mediante una red neuronal denominada $\mu^\varphi(s)$. La política es entrenada mediante Gradientes Profundos Gemelos y Atrasados de Política Determinística (TD3) para alcanzar el objetivo de controlar al péndulo invertido en la posición vertical.

A su vez, el agente posee su propia arquitectura, la cual se encuentra compuesta por todos los componentes necesarios para implementar TD3, tales como el búfer de reproducción y las redes neuronales necesarias. Los componentes fueron separados en módulos individuales, como se muestra en la ilustración 21.

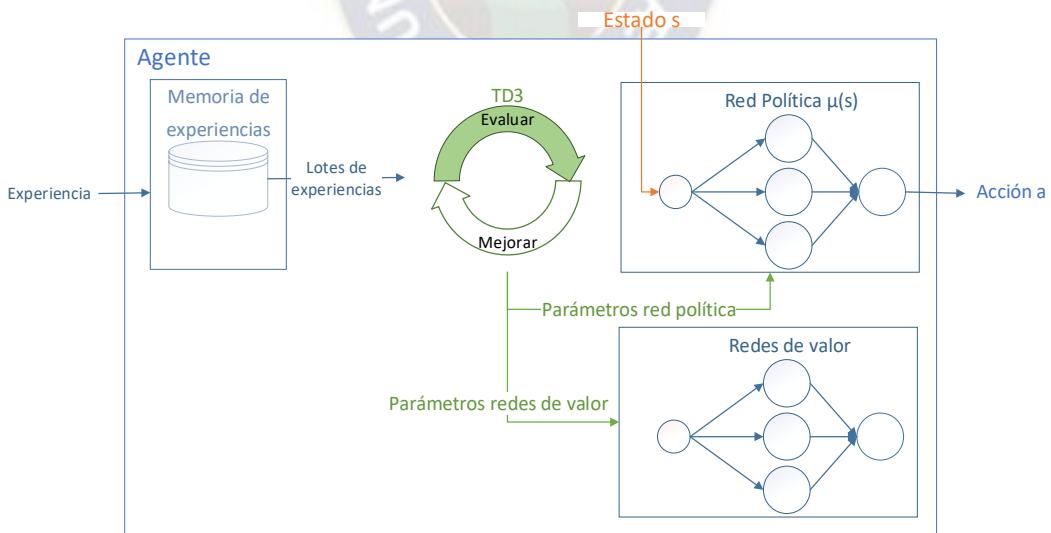


Ilustración 21 Arquitectura del agente. Fuente: Elaboración propia.

A continuación, se procede a explicar el funcionamiento de cada uno de los módulos que conforman la arquitectura del agente.

3.1.3.1. Módulo de Memoria de experiencias

El módulo de memoria de experiencias, fue diseñado como una representación del búfer de reproducción \mathcal{D} , el cual es necesario para el funcionamiento de todo agente basado en TD3. Por lo tanto, él debe cumplir dos funciones fundamentales:

- Almacenar tuplas de experiencias de la forma $(s_t, a_t, r_{t+1}, s_{t+1}, f)$ después de cada instante de tiempo de interacción entre el agente y el entorno.
- Generar lotes \mathcal{B} de un número arbitrario m de muestras de experiencias, las cuales son tomadas aleatoriamente de todas las experiencias almacenadas previamente.

Dado que se trata de un sistema que debe ser implementado computacionalmente, la clase denominada `Memoria`, la cual fue diseñada representar el módulo de memoria de experiencias, posee también las siguientes características:

- Al instanciarse como un objeto, el búfer de reproducción contenido en ella posee un tamaño fijo t_{mem} , tal que se pueda almacenar un número finito de experiencias dentro la memoria del computador que corre el agente.
- Cuando se haya llenado el búfer de reproducción, habiendo alcanzado un número de experiencias almacenadas igual a t_{mem} , empieza a sobrescribir las experiencias ya almacenadas a partir de la primera; esto permite al agente dejar de usar las experiencias más tempranas para entrenar la política y funciones de valor.
- Posee un método para almacenar su búfer de experiencias al disco duro.
- Posee un método para cargar su búfer de experiencias desde disco duro.

3.1.3.2. Módulo de Redes Neuronales.

Para diseñar el módulo de redes neuronales, primeramente, se observó que todo agente basado en TD3 utiliza un número total de seis redes neuronales:

- Dos redes neuronales Q_1^β y Q_2^β de tipo crítico, que aproximan la función de valor de estado-acción.
- Dos redes neuronales $Q_1^{\beta obj}$ y $Q_2^{\beta obj}$ de tipo critico, que se usan como redes objetivo para las dos anteriores en el proceso de entrenamiento.
- Una red μ^φ de tipo actor, que implementa la política determinística.
- Una red $\mu^{\varphi obj}$ de tipo actor, que se usa como red objetivo para la anterior en el proceso de entrenamiento.

Dado esto se observó que, aunque se utilizan seis redes neuronales en el proceso de entrenamiento

del agente, 4 de dichas redes (Q_1^β , Q_2^β , $Q_1^{\beta obj}$ y $Q_2^{\beta obj}$) aproximan el mismo tipo de función y deben tener la misma arquitectura de red neuronal. De igual manera, las dos redes neuronales restantes (μ^φ y $\mu^{\varphi obj}$) deben poseer la misma arquitectura de red neuronal.

Por lo tanto, para proveer las redes neuronales requeridas por TD3 de una manera acorde a la modularidad ideada, la arquitectura del agente propuso el módulo de redes neuronales, el cual posee las características que se describen a continuación.

- Implementa una clase denominada `RedActor`, la cual describe una arquitectura de red neuronal profunda que toma a su entrada al estado y lo mapea a una acción determinística.
- Implementa una clase denominada `RedCritica`, la cual describe una arquitectura de red neuronal profunda que toma a su entrada al estado junto a la acción y los mapea a un estimado del valor de estado-acción.
- Cada clase a su vez define su propio optimizador de red neuronal que utiliza el algoritmo ADAM.
- Cada clase posee un método para almacenar los parámetros de la red neuronal junto al estado de su optimizador a disco duro.
- Cada clase posee un método para cargar los parámetros de la red neuronal junto al estado de su optimizador desde disco duro.

De esta manera, se propuso una manera que permite instanciar las seis redes neuronales necesarias como objetos, definiendo solamente dos clases de redes neuronales. La ilustración 22 a continuación muestra la funcionalidad descrita.

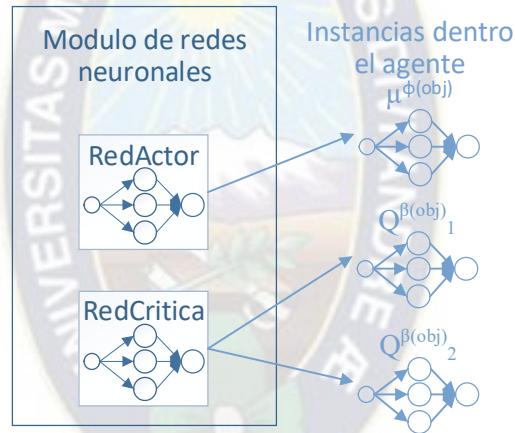


Ilustración 22 Modulo de redes neuronales e instancias. Fuente: Elaboración propia.

3.1.4. Clase agente

Finalmente, se definió al agente mismo, como una clase denominada `Agente`. Dicha clase aprovecha los dos módulos detallados anteriormente para llevar a cabo el funcionamiento de un agente basado en TD3. Por lo tanto, cumple con la funcionalidad descrita a continuación.

- Crea una instancia la clase `Memoria` con el propósito de almacenar y usar experiencias para el proceso de entrenamiento.
- Crea instancias la clase `RedActor` dos veces, generando las redes μ^φ y μ^{obj} .
- Crea instancias de la clase `RedCritica` 4 veces, generando las redes Q_1^β , Q_2^β , $Q_1^{\beta obj}$ y $Q_2^{\beta obj}$.
- Posee un método elegir acciones a en respuesta a estados s recibidos del entorno usando la red política μ^φ .
- Posee un método para almacenar experiencias de la forma $(s_t, a_t, r_{t+1}, s_{t+1}, f)$ en su instancia del módulo de memoria.

- Posee un método que le permite ejecutar su propia implementación del algoritmo de TD3 para llevar a cabo el entrenamiento.
- Posee un método para almacenar el estado de cada uno de los objetos instanciados dentro suyo a disco duro.
- Posee un método para cargar el estado de cada uno de los objetos instanciados dentro suyo desde disco duro.

3.1.5. Entorno

Dado que el entorno es definido según la teoría como todo aquello con lo que el agente interactúa, una manera errónea de enmarcarlo en el presente proyecto es simplemente considerar el péndulo invertido como una definición válida de entorno.

Sin embargo, se debe tomar en cuenta que el agente no interactúa de manera directa con el entorno, como es común en los sistemas simulados usados en la academia para el desarrollo de algoritmos de aprendizaje reforzado; en el presente proyecto la interacción es llevada a cabo con un sistema físico en el mundo real.

Por esa razón, además de diseñar el agente, fue necesario representar al entorno como una arquitectura que consta de módulos y subsistemas, los cuales son capaces de obtener información de los sensores del péndulo invertido y representarla de manera adecuada para el agente, así como proveer una manera de ejecutar las acciones elegidas por el agente en el mismo.

Por lo tanto, se definió el agente como una arquitectura compuesta tanto por el péndulo invertido real, como por componentes de software y hardware que se encuentran en comunicación con el fin de calcular el estado del péndulo invertido, calcular la recompensa y ejecutar las acciones que el agente elige.

La ilustración 23 muestra un esquema general de la arquitectura del entorno propuesta.

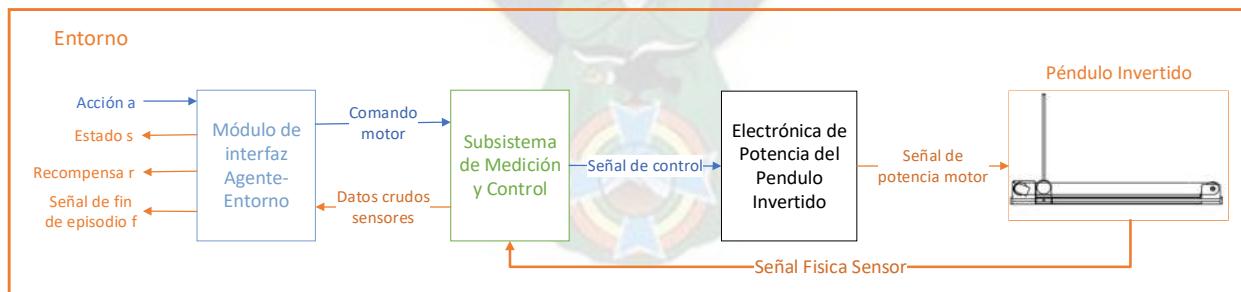


Ilustración 23 Arquitectura del entorno. Fuente: Elaboración propia.

A partir de este esquema, se procede a explicar cada uno de los subsistemas y módulos que conforman el entorno a continuación.

3.1.5.1. Módulo de Interfaz Agente-Entorno

El módulo de interfaz agente-entorno, se propuso como una clase denominada Entorno. Su funcionalidad permite al agente interactuar con el entorno, facilitando el intercambio de señales necesario para llevar a cabo la interacción episódica entre el agente y el péndulo invertido.

Para lograr este propósito, este módulo se definió como una clase con la capacidad de realizar las siguientes funciones:

- Recibir datos obtenidos de los sensores presentes en el péndulo invertido, los cuales son enviados desde el subsistema de medición y control mediante una conexión serial.
- Convertir los datos de los sensores recibidos del sistema de medición y control en cantidades expresadas en el sistema métrico.
- Usar los datos convertidos de los sensores para formar un vector de observación del estado de la forma $[x \ \dot{x} \ \theta \ \dot{\theta}]$.
- Calcular la recompensa usando una función de recompensa $r(s)$.
- Generar la señal de fin de episodio f , que indica si se ha alcanzado un estado final o si se ha alcanzado el máximo número de pasos de tiempo especificado para cada episodio.
- Entregar la observación del estado, la recompensa calculada y la señal de fin de estado al módulo principal.
- Enviar comandos numéricos al subsistema de medición y control mediante la conexión serial.

La ilustración 24 muestra el funcionamiento del módulo de interfaz agente-entorno y las tareas que lleva a cabo.

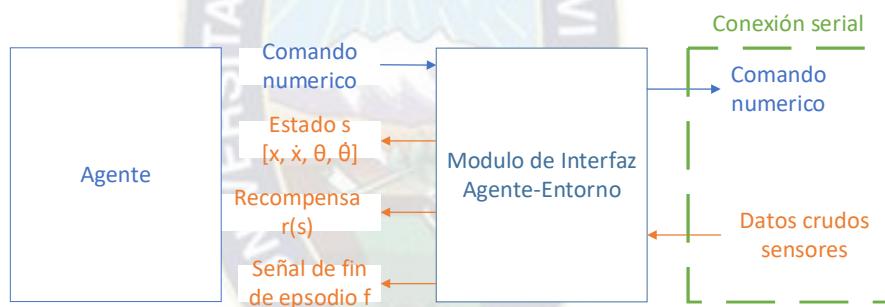


Ilustración 24 Módulo de interfaz agente-entorno. Fuente: Elaboración propia.

3.1.5.2. Subsistema de Medición y Control

El subsistema de medición y control se propuso con el propósito de actuar como un nexo entre el módulo de interfaz agente entorno, la electrónica de potencia del péndulo invertido y los sensores del mismo.

Para lograr este propósito, este subsistema se propuso como un programa que realiza las funciones listadas a continuación.

- Recibe comandos numéricos enviados por el módulo de interfaz agente-entorno mediante una conexión serial.
- Interpreta los comandos numéricos recibidos y los transforma en una señal de control apropiada para la electrónica de potencia del péndulo invertido, de manera que estos sean ejecutados como acciones reales.
- Ejecuta una rutina que le permite retornar al carro del péndulo invertido a su estado inicial en el centro de la pista (con $x = 0$), cuando se recibe un comando numérico especial, denominado comando de reseteo.

- Envía datos muestreados de los sensores del péndulo invertido periódicamente con tasa de muestreo T_s , después de haber recibido un comando numérico especial, denominado comando de inicio de interacción.

La ilustración 25 muestra el funcionamiento del módulo de interfaz agente entorno, con las señales que recibe y genera.

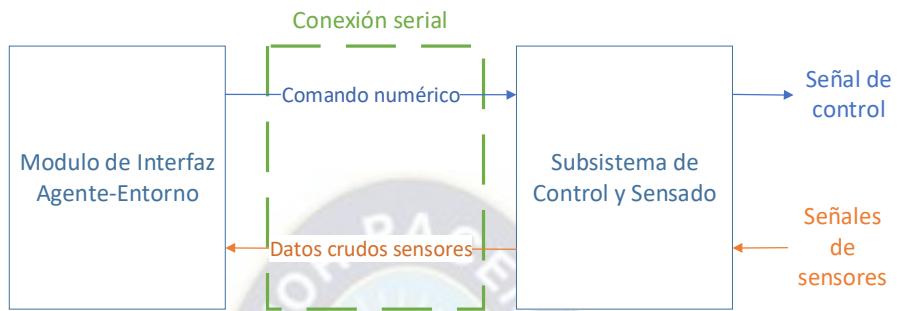


Ilustración 25 Subsistema de medición y control. Fuente: Elaboración propia.

3.1.6. Módulo de Exploración

Dado que el dilema exploración-explotación es un problema de importancia que se presenta en todo sistema de aprendizaje reforzado profundo, se puso especial énfasis en reducir y mitigar sus efectos sobre el proceso de entrenamiento, usando diferentes estrategias de exploración diseñadas con el fin de acelerar la convergencia a una política que logre alcanzar el objetivo de controlar el péndulo invertido.

Por esto, se propuso y diseñó el módulo de exploración, el cual es representado mediante una clase denominada `Explorador`. Ella se encuentra encargada de realizar las siguientes funciones:

- Generar acciones para mover el carro del péndulo invertido hacia los límites laterales de la pista a velocidades diferentes. Esta estrategia de exploración arbitraria tiene el fin de generar experiencias acerca de los límites de la pista para el entrenamiento del agente desde temprano.
- Generar acciones que son tomadas aleatoriamente de una distribución normal. Esta estrategia de exploración arbitraria tiene el fin de generar experiencias que muestran al agente el comportamiento del sistema en la proximidad del estado inicial.
- Tomar como entrada una acción generada por la red política μ^φ y añadir a ella ruido según lo propuesto por los autores de TD3; esta estrategia permite agregar un componente de exploración a la política determinística.

3.1.7. Módulo de Métrica

Con el fin de cuantificar el rendimiento del sistema para un análisis posterior al proceso de entrenamiento, se propuso el módulo de métrica, representado mediante una clase denominada `Metrica`, la cual posee métodos encargados de realizar las funciones detalladas a continuación.

- Calcular la recompensa promedio después de cada episodio que se lleva a cabo.
- Almacenar en una lista la recompensa promedio calculada.
- Almacenar el número de episodios transcurridos a lo largo del entrenamiento del sistema.

- Guardar a disco duro la lista de recompensas promedio y el número de episodios.
- Cargar desde disco duro la lista de recompensas promedio y el número de episodios.

3.1.8. Módulo Principal

El módulo principal fue propuesto como un sistema encargado de instanciar todas las clases descritas anteriormente y usarlas para realizar el entrenamiento del agente, implementando un lazo de control en el cual se lleva a cabo la interacción episódica entre el entorno y el agente, junto al entrenamiento usando el algoritmo de TD3. La ilustración 26 muestra el funcionamiento del módulo principal, con todas las instancias que encapsula.

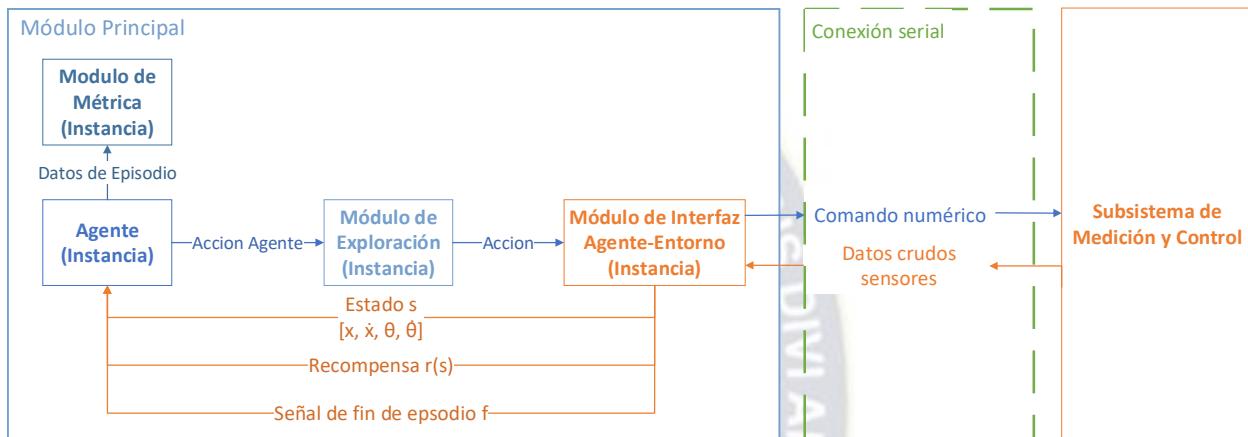


Ilustración 26 Modulo principal. Fuente: Elaboración propia.

Con el fin de lograr la interacción episódica requerida para el entrenamiento del agente, el módulo principal fue diseñado para realizar las tareas que se describen a continuación.

- Aceptar un argumento de entrada para seleccionar si se desea continuar el entrenamiento del agente a partir de una versión guardada con anterioridad o si se desea iniciar un nuevo proceso de entrenamiento.
- Llevar a cabo el ciclo de interacción episódica entre el agente y el entorno durante un numero de episodios n_{ep} .
- Terminar cada episodio cuando recibe la señal de finalización de episodio $f = 1$ desde el módulo de interfaz agente-entorno.
- Enviar el comando de reseteo al subsistema de medición y control a la finalización de cada episodio para devolverlo a su estado inicial; de esta manera el entorno se encuentra listo para llevar a cabo el siguiente episodio.
- Ejecutar el entrenamiento del agente a la finalización de cada episodio.
- Guardar el estado del agente y del módulo de métrica después de haber ejecutado el entrenamiento.

3.1.9. Módulo de Demostración

El módulo de demostración, fue propuesto como un sistema encargado de permitir visualizar el comportamiento del agente fuera del proceso de entrenamiento. El propósito del módulo de demostración es ser usado para observar el rendimiento del agente entrenado, obtener datos para el análisis de su comportamiento como controlador del péndulo invertido, y realizar una demostración en vivo de la política entrenada fuera del marco de un sistema de aprendizaje reforzado profundo en el cual se está llevando a cabo una interacción episódica. La ilustración 27 muestra el funcionamiento del módulo de demostración.

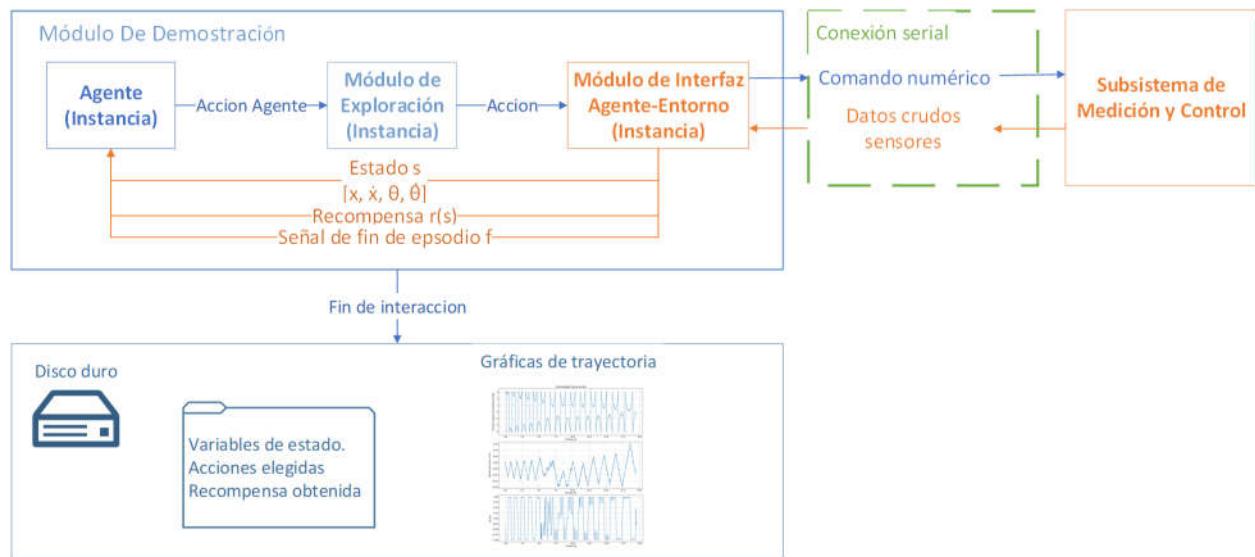


Ilustración 27 Modulo de demostración. Fuente: Elaboración propia.

Para lograr la funcionalidad ideada, el módulo de demostración fue diseñado para realizar las tareas que se describen a continuación.

- Aceptar un argumento de entrada, mediante el cual se selecciona si se desea llevar a cabo una trayectoria de interacción entre el agente y el entorno con longitud igual a un episodio, o si se desea llevar a cabo una interacción arbitrariamente larga.
- Almacenar los valores del estado del entorno, las acciones escogidas por el agente y la recompensa a lo largo de la trayectoria generada.
- Una vez terminada la trayectoria de interacción entre el agente y el entorno, guardar los datos previamente almacenados a disco duro.
- Realizar graficas de las cuatro variables de estado del entorno, de las acciones elegidas por el agente y de la recompensa recibida a lo largo de la trayectoria.

3.1.10. Arquitectura Hardware del Sistema

Dado que los varios componentes de la arquitectura descrita anteriormente son entes computacionales, destinados a interactuar con un entorno real, considerando además que la ejecución y entrenamiento de redes neuronales dentro el agente requiere de prestaciones de hardware altas, mientras que la generación y obtención de señales del péndulo invertido requiere de hardware especializado, se ideó una arquitectura de hardware que separa la arquitectura global

descrita en la secciones anteriores, dentro dos componentes de hardware. La arquitectura de hardware ideada muestra en la ilustración 28.

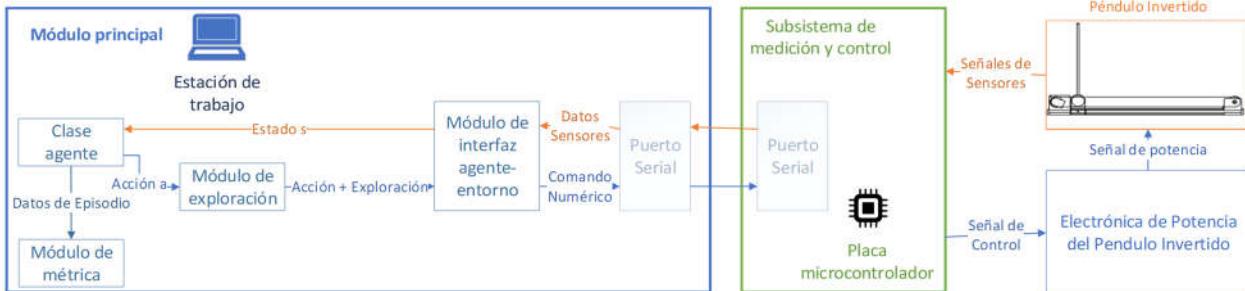


Ilustración 28 Arquitectura hardware del sistema. Fuente: Elaboración propia.

Dentro la arquitectura de hardware mostrada en la ilustración 28, se observa que existen dos componentes de hardware principales que se encuentran en comunicación mediante una conexión serial. Ambos enmarcan los diferentes elementos computacionales de la arquitectura general descrita en las subsecciones previas y se describen a continuación.

- **Estación de trabajo:** Es un computador dentro el cual se ejecutan el módulo principal, el módulo de demostración y todos los submódulos relacionados a ambos. Dentro de la estación de trabajo se generan acciones para interactuar con el entorno y se lleva a cabo el entrenamiento del agente, usando aceleración de hardware provista por un GPU.
- **Placa de microcontrolador:** La cual posee un microcontrolador con prestaciones más discretas y componentes de hardware dedicados a obtener datos de los sensores y generar señales de control para la electrónica de potencia del péndulo invertido. Su propósito es realizar las tareas del subsistema de medición y control de manera eficiente.

3.2. Requerimientos de Hardware del Proyecto

Con el fin de implementar la arquitectura para el control del péndulo invertido mencionada en las secciones anteriores, se especificaron los requerimientos de hardware descritos a continuación para los diferentes componentes de hardware que conforman el sistema.

3.2.1. Planta

Como se mencionó en el capítulo 1, el presente proyecto no buscó desarrollar al péndulo invertido como plataforma de experimentación, sino aprovechar una de las plataformas disponibles en el mercado.

Esto implicó la adquisición de una planta de péndulo invertido, para la cual se usaron los siguientes requerimientos:

- **Sensor de posición del péndulo:** Para obtener la posición angular del péndulo, la planta debe poseer un sensor de tipo encoder incremental con una resolución mínima de **720 $\frac{\text{pulsos}}{\text{revolucion}}$** .
- **Motor:** Para poder desplazar el carro lateralmente, la planta debe poseer un motor. Se especificó como requerimiento es que este debe estar estandarizado y debe ser de fácil

obtención, con el propósito de que este pueda ser reemplazado futuramente o para replicar el experimento.

- **Electrónica de potencia:** Para poder accionar el motor a usando señales generadas por el subsistema de medición y control, se requiere que la planta posea un sistema de electrónica de potencia. El sistema de potencia incluido debe consistir en un driver que permita mover al motor en ambas direcciones a partir de señales de control enviadas desde el módulo de medición y control.

3.2.2. Placa de Microcontrolador

Como se especificó en la sección 3.1.10, el subsistema de medición y control fue implementado sobre un microcontrolador, el cual envía y recibe datos desde un computador que corre el módulo de interfaz-agente entorno y el agente.

Para esto se eligió una placa de microcontrolador con hardware capaz de comunicarse con un computador periódicamente mediante una conexión serial usando USB, pero que además posee componentes que permiten muestrear el estado del encoder presente en el péndulo invertido de manera eficiente.

Por lo tanto, se usaron los requerimientos listados a continuación para la selección de la placa de microcontrolador.

- **Interfaz de comunicación serial:** Debe poseer una interfaz de comunicación serial capaz de enviar y recibir datos mediante puerto USB a una tasa de 500600[baud] o mayor; este requerimiento relativamente alto tuvo el fin de reducir el retardo con en el cual los datos del estado se hacen disponibles al agente.
- **Temporizador:** Debe poseer por lo menos un temporizador capaz de generar una interrupción en un intervalo de tiempo de 10[ms]. El propósito de este requerimiento fue ayudar a tomar las muestras de los sensores del péndulo invertido en intervalos de tiempo espaciados uniformemente.
- **Periférico de generación de señales PWM:** Debe poseer por lo menos una salida digital con conexión a un temporizador capaz de generar señales PWM. Este requerimiento se impuso para poder generar señales adecuadas para el driver del motor de la planta eficientemente.
- **Contador con interfaz de encoder:** Debe poseer por lo menos un contador rápido bidireccional con la capacidad de configurarse como interfaz de encoder. Este requerimiento se propuso de manera que no se requiera el uso de interrupciones para llevar la cuenta de los pulsos del encoder.
- **Programador y depurador integrado:** Debe poseer un programador y depurador integrado, este requerimiento se puso con el fin de llevar a cabo el desarrollo del programa del subsistema de medición y control de manera rápida, pudiendo probarlo y depurarlo constantemente a lo largo de su desarrollo.

3.2.3. Estación de Trabajo

Según la arquitectura de hardware propuesta especificada en la sección 3.1.10, la estación de trabajo es utilizada para correr el módulo principal; esto a su vez implica correr todos los módulos

mediante los cuales se lleva a cabo el entrenamiento de las 6 redes neuronales, como especifica el algoritmo de TD3.

Por lo tanto, se requirió prestaciones relativamente altas en cuanto a capacidad de computo por parte de la estación de trabajo, razón por la cual se usaron los requerimientos listados a continuación para su elección.

- **GPU:** Con el fin de reducir el tiempo de entrenamiento de las redes neuronales, pero también con el objetivo de reducir el tiempo en el cual se generan las acciones para ser enviadas al módulo de medición y control, se requirió contar con un GPU que sea capaz de acelerar la ejecución de las redes neuronales.
- **Sistema Operativo:** Con el fin de desarrollar el sistema completamente usando componentes gratuitos y de código abierto, se requirió que la estación de trabajo use un sistema operativo basado en GNU/LINUX.
- **Puerto USB:** Para establecer la conexión serial que permite a la estación de trabajo conectarse con la placa de microcontrolador en la cual se encuentra implementado el subsistema de medición y control, se requirió contar con un puerto USB en la estación de trabajo.

3.3. Requerimientos de Software del Proyecto

Con el fin de implementar los diferentes componentes de software de la arquitectura propuesta, se especificaron los requerimientos de software que se presentan a continuación.

3.3.1. Lenguaje de Programación

Dentro el presente proyecto se buscó desarrollar un sistema modular que permita experimentar y modificar los diferentes componentes individualmente, abstrayéndolos como clases de objetos que pueden ser instanciados.

Adicionalmente, como se mencionó en el capítulo 1, el proyecto se limitó a utilizar librerías de aprendizaje profundo para la implementación del agente; se buscó aprovechar las herramientas ya desarrolladas y probadas en la industria y en la academia para implementar las redes neuronales, junto a los componentes relacionados al entrenamiento de las mismas.

Por lo tanto, la elección del lenguaje de programación usado para implementar los diferentes módulos que conforman la arquitectura de sistema se llevó a cabo en base a los siguientes requerimientos.

- El lenguaje de programación usado debe soportar el uso del paradigma de programación orientada a objetos, con el fin de proveer la posibilidad de abstraer los diferentes módulos.
- El lenguaje de programación usado debe poseer aceptación amplia en la industria y academia como herramienta para desarrollo de sistemas de aprendizaje profundo.
- El lenguaje de programación usado debe poseer un ecosistema amplio de librerías especializadas para realizar tareas de aprendizaje profundo.

3.3.2. Librería de Aprendizaje Profundo

Una vez realizada la elección de un lenguaje de programación para desarrollar la mayoría del proyecto, tomando en cuenta que se especificó que el lenguaje de programación debe poseer

librerías para la implementación de sistemas de aprendizaje profundo, se usaron los requerimientos listados a continuación para la elección de dicha librería.

- **Aceleración de GPU:** La librería de aprendizaje profundo elegida debe tener la capacidad de poder aprovechar la aceleración de una unidad de procesamiento de gráficos (GPU), ya que como se describió en la sección 2.4, el uso de GPUs reduce substancialmente el tiempo de ejecución de las redes neuronales.
- **Definición de las redes neuronales:** La librería de aprendizaje profundo elegida debe soportar la definición de redes neuronales feedforward como objetos.
- **Cálculo de gradientes:** La librería de aprendizaje reforzado profundo elegida debe ser capaz de calcular gradientes para el entrenamiento de las redes neuronales.
- **Optimizadores:** La librería debe proveer funciones que implementan algoritmos de optimización para llevar a cabo el entrenamiento de redes neuronales, tales como descenso de gradiente estocástico y ADAM.

3.4. Elección de las herramientas hardware del proyecto

3.4.1. RoboBalance Store Linear Inverted Pendulum

Para elegir una planta de péndulo invertido, se hizo un reconocimiento del mercado de compras por internet, del cual se obtuvo las opciones que se resumen en la tabla 1.

Plantas de Péndulo Invertido Consideradas					
Fabricante	Nombre	Sensor	Motor	Electrónica de potencia	
Quanser	Linear Servo Base Unit with Inverted Pendulum	Encoder incremental 4096 [$\frac{\text{pulsos}}{\text{revolución}}$]	inde	Motor de corriente continua de 6[v] con reducción 3.71 : 1 propietario	Quanser VoltPAQ-X1 propietario
RoboBalance Store	Linear Inverted Pendulum	Encoder incremental 1000 [$\frac{\text{pulsos}}{\text{revolución}}$]	inde	Motor paso a paso estándar NEMA-17 de 65[N * cm]	StepperOnline DM-542T
Technosys Systems	Aluminium Alloy Direct Drive Inverted Pendulum System	Encoder incremental 2400 [$\frac{\text{pulsos}}{\text{revolución}}$]	inde	Motor AC no especificado*	No especificado o no incluido*

Tabla 1 Plantas de péndulo invertido consideradas. Fuente: Elaboración propia.

*Nota: El fabricante no provee las especificaciones, se realizó un intento de contacto con el fabricante, sin éxito.

En base a la tabla 1, tomando en cuenta los requerimientos especificados en la sección 3.2.1, se eligió el péndulo ofrecido por RoboBalance Store. La principal razón fue que incluye un motor que se adhiere al estándar NEMA-17 de motores paso a paso, el cual se encuentra disponible ampliamente. De manera similar, la electrónica de potencia se encuentra compuesta por un driver estándar para motores paso a paso, el cual provee opciones para micropasos.

A continuación, se describe los componentes de la planta elegida a continuación mediante la ilustración 29.

- **(1) Motor:** El motor paso a paso NEMA-17 se encuentra montado al extremo derecho de la pista y usa una polea para mover el carro.
- **(2) Pista:** La planta posee una pista de fricción reducida con una longitud útil de 0.4[m], sobre la cual el carro se puede desplazar.
- **(3) Carro:** El carro se desplaza lateralmente sobre la pista y se encuentra acoplado al motor mediante la polea flexible. El carro sostiene el encoder incremental.
- **(4) Encoder rotacional incremental:** El encoder incremental de 1000 $\left[\frac{\text{pulsos}}{\text{revolución}}\right]$ se encuentra montado sobre el carro y su eje se encuentra directamente acoplado al péndulo, permitiendo la medición de su posición angular.
- **(5) Péndulo:** El péndulo consiste una barra metálica que se encuentra acoplada al eje del encoder incremental.
- **(6) Sensor de final de carrera:** El sensor de final de carrera consiste en un switch situado al extremo izquierdo de la pista, cuyo propósito es detectar cuando el carro ha alcanzado ese extremo. Este switch es activo en bajo; emite una señal baja cuando el carro alcanza dicho extremo y lo presiona, caso contrario siempre emite una señal alta.



Ilustración 29 Péndulo invertido elegido. Fuente: Elaboración propia.

El motor de la planta es un motor paso a paso StepperOnline de 200 $\left[\frac{\text{pasos}}{\text{revolucion}}\right]$ con un torque de 65[N cm] a 2.1 [A] [60].

Para control el motor, la planta incluye el driver para motores paso a paso StepperOnline DMT-542T, del cual se destaca la capacidad para generar micropasos. Usando la función de micropasos, es posible incrementar el número de pasos por revolución de un motor paso a paso por múltiplos enteros, a costo de una reducción de torque. Las resoluciones de micropasos disponibles en el

driver varían desde resolución x2 hasta x125 y son seleccionables mediante dip-switches [61]. La ilustración 30, muestra el driver DMT-425T y detalla los diferentes componentes presentes en él.

- **(1) Entrada de alimentación:** El driver es alimentado con 24[v] mediante esta entrada.
- **(2) Salidas para motor:** Salidas para las fases A y B del motor paso a paso.
- **(3) Entrada de habilitación:** Permite habilitar el driver con una señal en alto
- **(4) Entrada de dirección:** Selecciona movimiento del motor en sentido horario con una señal baja y movimiento antihorario con una señal alta.
- **(5) Entrada de frecuencia:** Permite controlar la velocidad del motor mediante una señal de entrada a una frecuencia determinada, moviendo el motor a una velocidad de $\left[\frac{\text{pasos}}{\text{s}} \right]$ igual a la frecuencia de dicha señal de entrada.
- **(6) Dip-switches de configuración:** Permiten configurar características del driver tales como la corriente máxima y el factor de incremento de resolución de micropasos.

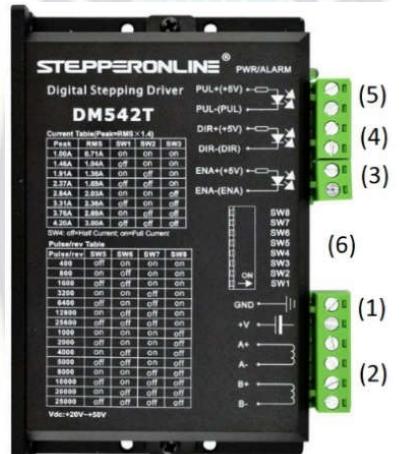


Ilustración 30 Driver DMT-542T. Fuente: [62].

Finalmente, la planta incluye un encoder incremental de 1000 $\left[\frac{\text{pulsos}}{\text{revolución}} \right]$, el cual es usado como sensor de la posición angular del péndulo. Dicho sensor permite medir el ángulo de la rotación del péndulo usando dos trenes de pulsos, denominados Fase A y Fase B, los cuales se encuentran desfasados 90 grados. El funcionamiento del encoder en base a señales de fase A y fase B, se muestra en la ilustración 31.

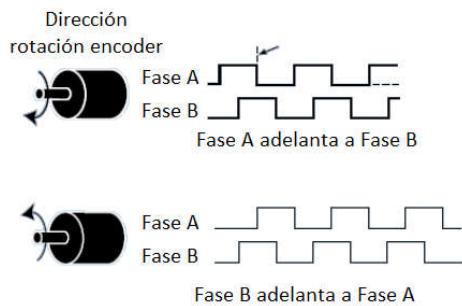


Ilustración 31 Funcionamiento de un encoder incremental. Fuente: Elaboración propia.

Como se observa en la ilustración 31, cuando el tren de pulsos de la fase A realiza un cambio de estado bajo a alto mientras la fase B se encuentra en estado alto, entonces se dice que la fase A adelanta a la fase B y se detecta que el encoder está rotando en sentido horario.

Si en vez, la fase A realiza un cambio de estado alto a bajo mientras la fase B se encuentra en estado bajo, entonces se dice que la fase B adelanta a la fase A y se detecta que el encoder está rotando en sentido antihorario.

Contando el número de pulsos generados por la fase A e incrementando o decrementando la cuenta de pulsos de acuerdo al estado de la fase B es posible medir la posición angular del péndulo.

3.4.2. Núcleo STM32F767ZI

Un reconocimiento del mercado a nivel Bolivia, produjo varias opciones de placas con microcontroladores de diversas arquitecturas, los cuales están orientados a diferentes mercados. Un resumen de las principales placas consideradas se muestra en la tabla 2.

Placas de Microcontrolador Consideradas			
Nombre	Arquitectura del Microcontrolador	Frecuencia de Reloj	Enfoque
Arduino UNO	AVR – 8 bits	16[MHz]	Electrónica como pasatiempo, artistas
Nucleo STM32F767ZI	ARM M7 – 32 bits	216[MHz]	Sistemas de control en tiempo real
chipKIT Uno32	PICMX32 – 32 bits	80[MHz]	Sistemas de control en tiempo real
Bluepill STM32F103C8T6	ARM M3 – 32 bits	72[MHz]	Sistemas adquisición de datos o control con bajo consumo de energía

Tabla 2 Placas de microcontrolador consideradas. Fuente: Elaboración propia

Dado que las placas Arduino se encuentran orientadas a uso en proyectos y prototipos dentro un ámbito informal, se las excluyó del proceso de selección. La tabla 3 muestra un resumen de las características de cada una de las placas restantes consideradas, relacionándolas a los requerimientos especificados en la sección 3.2.2. [63] [64]

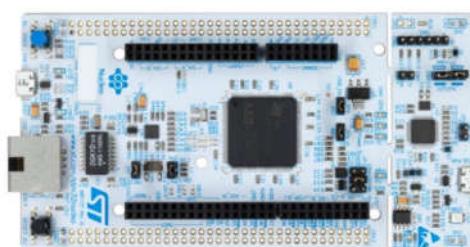


Ilustración 32 Placa de desarrollo Nucleo STM32F767ZI. Fuente: [63].

Placas de Microcontrolador Consideradas - Especificaciones					
Nombre	Interfaz Serial	Temporizadores	Hardware de Generación de PWM	Interfaces de Encoder	Programador y depurador integrado
Nucleo STM32F767ZI [61]	Si, acceso directo a memoria y generación de interrupciones	Si, dos temporizadores de 32 bit	Si, dos contadores para generación de PWM	Si, hasta dos encoders	Si, programador/depurador ST-LINK
Bluepill STM32F103C8T6 [62]	Si	Si, tres temporizadores de 16 bit	Si, un contador para generación de PWM	No	No, requiere un programador aparte
chipKIT Uno32 [63]	Si	Si, cinco temporizadores de 16 bit	Si, 5 contadores para generación de PWM	No	Si

Tabla 3 Placas de microcontrolador consideradas, especificaciones. Fuente: Elaboración propia.

Se observó de la comparación realizada mediante la tabla 3, que la placa Nucleo STM32F767ZI (mostrada en la ilustración 32), cumple con los requerimientos especificados en la sección 3.2.2, principalmente considerando que incluye interfaces hardware para encoder, pero también tomando en cuenta que incluye una interfaz serial con funciones avanzadas. Por ello, se la eligió para implementar el subsistema de medición y control.

A continuación, se describen algunos de los módulos especializados de hardware del microcontrolador STM32F767ZI utilizados en el subsistema de medición y control, los cuales facilitaron su implementación usando rutinas pequeñas y sencillas.

- **Interfaz serial:** Es la primera las características especializadas presente en el microcontrolador; la interfaz serial es capaz de recibir a través de su puerto serial uno o más bytes de datos y escribirlos a una región definida de RAM del microcontrolador, usando acceso directo a memoria (DMA), sin requerir la intervención del procesador.

El periférico es programado con un número de bytes predeterminado que esperar, también es asignado una región de RAM con un tamaño igual en bytes a la cual tiene acceso; al completarse la recepción de dicho número de bytes, la interfaz serial genera una interrupción que informa al procesador que la transferencia fue completada, entonces el procesador puede realizar operaciones con los datos recibidos, los cuales ya se encuentran disponibles en su memoria RAM. La ilustración 33 resume el proceso usado para recibir datos de la manera descrita.

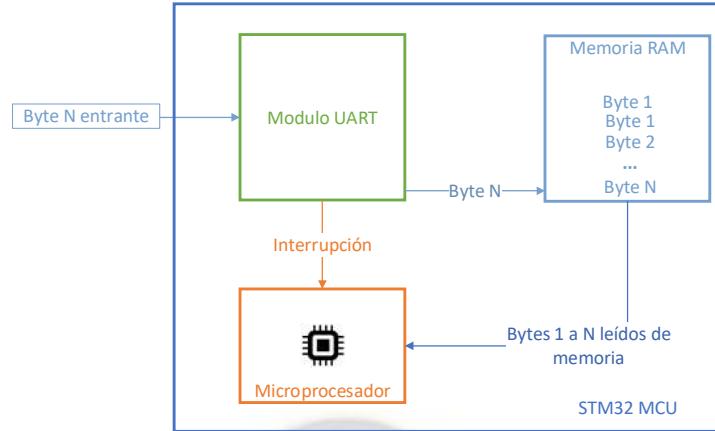


Ilustración 33 Funcionamiento simplificado de la interfaz serial con acceso directo a memoria. Fuente: Elaboración propia.

- **Interfaz de encoder:** La interfaz de encoder, es un periférico de entrada que se encuentra conectado a un contador, está destinada a permitir llevar la cuenta de un encoder incremental sin intervención del procesador. Tradicionalmente, el procesador atendería una interrupción cada vez que la fase A realice un cambio de estado, leyendo el estado de la fase B e incrementando o decrementando la cuenta.

Como se muestra en la ilustración 34, la interfaz de encoder posee elementos dedicados a llevar la cuenta de pulsos del encoder usando un contador hardware, sin necesidad de usar el procesador, liberando la carga de interrupciones que se occasionaría usando el método anteriormente descrito; esto se logra gracias a lógica implementada en hardware que genera pulsos que alimentan la entrada de reloj del contador y la señal que controla si la cuenta es hacia arriba o abajo. En cualquier momento, para obtener la cuenta, basta con que el procesador lea el dato almacenado en el contador.

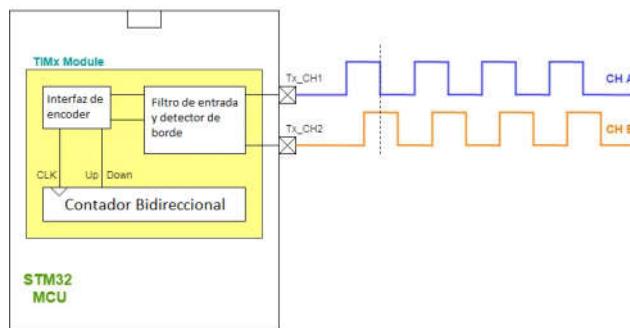


Ilustración 34 Funcionamiento de la interfaz de encoder. Fuente: [66].

Una característica adicional del hardware de la interfaz de encoder, es que permite generar pulsos para el contador no solamente en un cambio de estado bajo a alto de la fase A, sino también en un cambio de alto a bajo. En adición, permite generar también pulsos usando la fase B en adición a la fase A; esto permite a la interfaz de encoder generar 2 y 4 veces más pulsos por revolución del encoder, mediante modos denominados de resolución x2 y resolución x4, como se muestra en la ilustración 35.

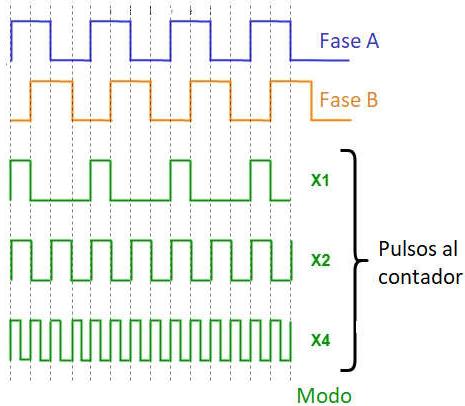


Ilustración 35 Funcionamiento de la interfaz de encoder en modos x2 y x4. Fuente: [66].

3.4.3. Estación de Trabajo

En base a los requerimientos especificados en la sección 3.2.3, se usó como estación de trabajo un computador portátil provisto de un GPU marca Nvidia, lo cual permitió la aceleración del proceso de entrenamiento. Se resumen las especificaciones de la estación de trabajo elegida en la tabla 4.

Estación de Trabajo - Especificaciones	
Nombre	Acer Predator Helios 300 PH315-51
CPU	Intel Core i7 8750H @2.2[GHz]
Memoria RAM	16[GB] DDR4@2667[MHz]
GPU	NVidia GTX1060m 6[GB] VRam
Puertos USB	1x USB 3.0, 2x USB 2.0, 1x USB 3.1 gen 1
Sistema Operativo	Arch Linux Kernel 5.10

Tabla 4 Estación de trabajo, especificaciones. Fuente: Elaboración propia.

3.5. Elección de las herramientas software del proyecto

3.5.1. Python

En base a los requerimientos especificados en la sección 3.3.1, se eligió el lenguaje de programación Python, ya que satisface los requerimientos listados en dicha sección como se describe a continuación.

Python es un lenguaje de programación interpretado de alto nivel, cuyo paradigma de programación principal es la programación orientada a objetos [67]. Python ha sido adoptado y usado ampliamente en diferentes campos, gracias a la facilidad que provee para el desarrollo rápido de software.

Además de manera particular, ha sido adoptado como lenguaje para el desarrollo de sistemas de aprendizaje automático durante los últimos años. Python es la elección más popular para la implementación de sistemas de aprendizaje automático según una encuesta realizada el año 2017; goza de una preferencia por parte del 57% de los desarrolladores de sistemas de aprendizaje automático encuestados [68]. Por su masiva adopción, Python ha sido beneficiado con soporte y uso amplio, tanto para tareas de aprendizaje profundo en la industria, como para desarrollo de sistemas experimentales en la academia.

Dado que el aprendizaje profundo y sus técnicas son un componente esencial para la implementación de sistemas de aprendizaje reforzado profundo, se eligió Python para la implementación de la mayoría de los módulos, a excepción del programa utilizado en el subsistema medición y control, ya que debe ser desarrollado en un lenguaje de programación compilado para su ejecución en el microcontrolador.

3.5.2. Pytorch

Sobre la base de la elección de Python como lenguaje de programación para el proyecto, se consideró tres librerías de código abierto utilizadas por la industria y la academia para el desarrollo de sistemas de aprendizaje profundo. Las tres se encuentran entre las más populares y con mayor soporte; se describen a continuación.

- **Pytorch:** Es una librería de código abierto para el desarrollo de sistemas de aprendizaje automático y aprendizaje profundo, es desarrollada bajo supervisión y con el auspicio del laboratorio de investigaciones en inteligencia artificial de Meta (Previamente Facebook) [69].
- **Tensorflow:** Es una librería para la implementación de sistemas de aprendizaje automático y aprendizaje profundo desarrollada con el auspicio de Google. Su principal ventaja es su amplio uso en sistemas utilizados en producción, para este propósito soporta el despliegue a dispositivos móviles y del internet de las cosas (IoT) [70].
- **Keras:** Es una librería para la implementación de sistemas de aprendizaje profundo, la cual utiliza Tensorflow como base; provee una interfaz de programación de aplicaciones de alto nivel que busca simplificar la implementación, mientras se aprovechan las capacidades ya desarrolladas de Tensorflow [71].

La elección fue llevada a cabo en base a una comparación entre las tres librerías mencionadas, tomando en cuenta los requerimientos especificados en la sección 3.3.2, como se muestra en la tabla 5.

De la comparación realizada mediante la tabla 5, se pudo observar las tres librerías poseen características similares en cuanto a sus capacidades de uso de GPU y algoritmos implementados para el entrenamiento de sistemas de aprendizaje profundo.

Dado lo anterior, el principal criterio usado para la elección de Pytorch fue la fácil definición de redes neuronales como objetos, la cual permite integrar código realizado usando Pytorch de manera fácil dentro la arquitectura propuesta a lo largo de la sección 3.1. Sin embargo, también se tomó en cuenta su amplio uso en la academia para el desarrollo y prueba de nuevos sistemas de aprendizaje profundo.

Pytorch realiza operaciones sobre objetos denominados tensores; estos son generalizaciones de vectores y matrices a dimensiones superiores. Los tensores son asignados tipos básicos usados para operaciones aritméticas, tales como enteros de 64 bit o números de punto flotante de 32 bit.

Librerías de aprendizaje profundo - Comparación					
Nombre	Uso de GPUs	Método de definición de redes neuronales	Cálculo de gradientes	Optimizadores	Enfoque
Pytorch	Si	Redes neuronales como objetos, con un enfoque en la reutilización de componentes	Si	Descenso de gradiente estocástico, ADAM	Desarrollo de sistemas de aprendizaje profundo para investigación y la academia
Tensorflow	Si	Redes neuronales como grafos computacionales	Si	Descenso de gradiente estocástico, ADAM	Desarrollo de sistemas de aprendizaje profundo orientados a despliegue en la industria
Keras	Si	Redes neuronales como grafos computacionales	Si	Descenso de gradiente estocástico, ADAM	Prototipado e iteración rápida para el desarrollo de sistemas de aprendizaje profundo

Tabla 5 Fuente: Librerías de aprendizaje profundo, comparación. Fuente: Elaboración propia.

Es posible convertir una lista de Python a un tensor usando el método de Pytorch `torch.tensor` mediante la sintaxis mostrada en la siguiente línea de código.

```
A = torch.tensor([1, 2, 3], dtype=torch.float32)
```

A continuación, se describe brevemente el uso de Pytorch para definir una red neuronal feedforward como una clase denominada `MiRed`.

Pytorch permite definir las dimensiones de entrada y salida de cada una de las capas de una arquitectura de red neuronal de manera sencilla, definiendo las operaciones lineales de cada capa en el constructor del objeto, como se muestra en el siguiente fragmento de código.

```
import torch
import torch.nn as nn
import torch.nn.functional as functional
import torch.optim as optim
class MiRed(nn.Module):
    def __init__(self):
        super(MiRed, self).__init__()
        # Definir la arquitectura de la red neuronal
        self.lineal1 = nn.Linear(6, 256)
        self.salida = nn.Linear(256, 4)
        # Instanciar un optimizador
        self.optim_actor = optim.Adam(self.parameters(), 0.001)
        # Enviar al CPU o GPU
        self.to(torch.device('cuda'))
```

- Primeramente, se define una clase denominada `MiRed`, la cual hereda de la clase `nn.Module` de Pytorch.
- Dentro el constructor `__init__` de la clase, se define la operación lineal de la primera capa escondida, la cual posee 6 entradas y 256 salidas, para ello se usa `self.lineall= nn.Linear(6, 256)`; esta línea define una capa de 256 neuronas con sus parámetros correspondientes.
- Seguidamente se define la operación lineal de la capa de salida, la cual posee 256 entradas y 4 salidas, usando `self.salida= nn.Linear(256, 4)`; esta línea define una capa de 4 neuronas .
- Se define el optimizador que será usado sobre los parámetros de la red neuronal mediante `self.optim_actor= optim.Adam(self.parameters(), 0.001)`, el cual usa el algoritmo ADAM con una tasa de entrenamiento igual a 0.001.
- Finalmente, con el fin de aprovechar la aceleración de un GPU, se asigna la red neuronal a la GPU usando `to(torch.device('cuda'))`.

Para obtener predicciones de la red, Pytorch especifica que se debe definir un método denominado `forward` dentro toda clase que define una arquitectura de red neuronal. En dicho método se aplican las funciones de activación escogidas a las operaciones lineales definidas en el constructor, como se muestra en el siguiente fragmento de código.

```
def forward(self, x):
    y = self.lineall(x)
    y = functional.relu(y)
    y = self.salida(y)
    y = torch.tanh(y)
    return y
```

- El método `forward` toma como argumento un tensor de Pytorch denominado `x`.
- Se obtiene el la operación lineal de `x` para la capa escondida ejecutando `self.lineall(x)` .
- Seguidamente, aplica la función de activación ReLU mediante `functional.relu(y)` .
- A continuación, se obtiene la operación lineal para la capa de salida ejecutando `self.salida(y)` .
- Finalmente se aplica la función de activación TanH, mediante `torch.tanh(y)` , obteniendo la predicción de la red neuronal `y`, que es retornada del método.

Usando un lote o set de datos que consiste en entradas contenidas en una variable `X` y ejemplos de salidas contenidos en una variable `Y`, ambos almacenados como tensores de Pytorch bidimensionales, es posible entrenar la red neuronal definida previamente usando el fragmento de código descrito a continuación.

```
red = MiRed()
X = X.to(torch.device('cuda'))
```

```

Y = Y.to(torch.device('cuda'))

for i in range(100):

    y_pred = red.forward(X)
    perdida = functional.mse_loss(Y, y_pred)
    perdida.backward()
    red.optim.step()

```

- Primeramente, se crea una instancia de la clase MiRed denominada simplemente red.
- Seguidamente, se envía los datos de entrenamiento al GPU.
- Se obtiene las predicciones de la red neuronal ejecutando red.forward(X) .
- Se calcula la función de perdida usando el error cuadrático medio entre los datos de entrenamiento y las predicciones de la red neuronal, mediante funcional.mse_loss(Y, y_pred) .
- Se calcula los gradientes de la función de perdida con respecto a los parámetros de la red neuronal mediante retropropagación, ejecutando perdida.backward() .
- Se ejecuta un paso de entrenamiento del optimizador de la red neuronal mediante red.optim.step() .
- Dado que los 4 anteriores pasos se ejecutan dentro de un bucle de 100 iteraciones, en total se lleva a cabo 100 pasos de entrenamiento de la red neuronal.

3.5.3. Numpy

Numpy es una librería de código abierto orientada a realizar tareas de computación científica y cálculos rápidos usando el lenguaje de programación Python [72].

Numpy implementa objetos de arreglos n-dimensionales de tamaño y tipo fijo similares a los tensores de Pytorch mediante la clase np.ndarray.

Como ejemplo, para crear un arreglo bidimensional inicializado con ceros, con tamaño 200*300 y de tipo float32 se usa la clase zeros de NumPy, como se muestra en el siguiente fragmento de código.

```

import numpy as np
A = np.zeros((200, 300), dtype=np.float32)

```

Para crear un vector a partir de una lista de Python se usa la clase array de la manera que se muestra a continuación.

```
B = np.array([1, 2, 3], dtype=np.float32)
```

Una característica notable que alienta el uso de Numpy, es su interoperabilidad con Pytorch, ya que los arreglos de NumPy pueden ser convertidos a tensores de Pytorch y viceversa usando la sintaxis que se muestra en las dos siguientes líneas de código.

```
B = torch.tensor(B) #Convertir a tensor de PyTorch
```

```
B = B.numpy() #Convertir de nuevo a arreglo de NumPy
```

Gracias a la interoperabilidad y fácil conversión de datos entre ambas librerías, Numpy es por lo general utilizado en conjunto con PyTorch, ya que provee métodos y funciones para manipular arreglos, realizar operaciones matemáticas sencillas sobre los mismos, guardarlos y cargarlos.

Lo anteriormente mencionado hace conveniente el uso de los arreglos de datos y funciones de Numpy en comparación con el uso de listas de Python, en especial porque facilita el almacenamiento de datos en memoria y la realización de operaciones matemáticas sobre ellos para ser utilizados después por Pytorch.

3.5.4. PySerial

Pyserial es una librería de código abierto para el envío y recepción de datos mediante comunicación serial usando el lenguaje de programación Python.

Su objetivo principal es abstraer los detalles de la implementación del protocolo de comunicación serial en los tres sistemas operativos principales; GNU/Linux, Windows y MacOS, proveyendo una interfaz idéntica para el acceso a los puertos seriales mediante clases y objetos en los tres sistemas operativos [73] . A continuación, se describe el uso de PySerial para establecer una conexión serial.

Para inicializar una conexión serial usando Pyserial, es necesario instanciar un objeto de tipo `serial.Serial` usando la sintaxis que se muestra a continuación.

```
conexion=serial.Serial("/dev/ttyACM0", 9600)
```

La sintaxis mostrada instancia la clase `serial.Serial` de Pyserial en un objeto denominado `conexion`, el cual utiliza el puerto `/dev/ttyACM0` a una velocidad de $9600 \frac{\text{baud}}{\text{s}}$.

Una vez instanciado un objeto para la conexión serial, para enviar un mensaje mediante el puerto serial, Pyserial provee el método `write`, el cual permite enviar un mensaje de longitud arbitraria contenido en su argumento como se muestra en la siguiente línea de código.

```
conexion.write(mensaje)
```

Pyserial también provee un método para leer un mensaje de una longitud definida de bytes del puerto serial, utilizando el método `read`. La sintaxis mostrada a continuación usa `read` para leer 100 bytes.

```
recibido = conexion.read(100)
```

La línea de código anterior asigna el mensaje de 100 bytes recibidos mediante `conexion` a una variable denominada `recibido`. Se menciona que el proceso en el cual la llamada al método `read` es realizada, se bloquea hasta que se haya leído la totalidad de bytes del mensaje,

Finalmente, Pyserial provee dos métodos utilitarios para borrar los contenidos del búfer de entrada y el búfer de salida de la conexión serial, con el propósito de evitar errores causados por datos

residuales que puedan quedar almacenados en ellos. El uso de ambos métodos se muestra en las dos siguientes líneas de código.

```
conexion.flushOutput() //Borrar búfer de salida
conexion.flushInput() // Borrar búfer de entrada
```

3.5.5. STM32CubeIDE

El microcontrolador STM32F767ZI usado en la placa de desarrollo elegida cuenta con un amplio ecosistema de herramientas ideadas para el desarrollo de software. Para desarrollar el programa del subsistema de medición y control, se consideró tres opciones, que se listan a continuación.

- **ARM Mbed:** Fue propuesto por ARM como un ecosistema de código abierto basado en el sistema operativo MbedOS. Usa el compilador ARM Compiler y un entorno de desarrollo basado en Theia o en la nube. Mediante Mbed, los diferentes fabricantes de microcontroladores y placas de desarrollo basados en arquitectura ARM, proveen librerías y soporte para que sus productos y periféricos puedan ser programados corriendo MbedOS [74].
- **STM32CUBEIDE:** Fue desarrollado por ST Microelectronics para su gama de microcontroladores basados en arquitectura ARM. STM32CUBEIDE provee un entorno de desarrollo basado en Eclipse junto al compilador GCC. Además, incluye una herramienta denominada STM32CUBEMX, mediante la cual es posible configurar los periféricos de los microcontroladores y placas ofrecidas por ST Microelectronics usando una interfaz gráfica [75].
- **ARM Keil uVision:** ARM Keil uVision es un entorno de desarrollo de software propietario y de pago desarrollado por ARM para microcontroladores basados su arquitectura, el cual fue orientado al desarrollo de aplicaciones en un ámbito industrial. Mientras que existe una versión gratuita, esta se encuentra enfocada como una introducción para estudiantes y posee limitaciones [76].

Entornos de desarrollo para NUCLEO-F767ZI - Comparación						
Nombre	Sistema Operativo	Lenguaje de programación	Nivel de abstracción	Configuración de periféricos	Soporte depurador integrado	Enfoque
ARM Mbed	Windows, GNU/Linux, MacOS, Navegador Web desde la nube	C o C++	Alto	Solamente mediante código	Si	Desarrollo rápido de sistemas de IoT
STM32 CUBEIDE	Windows, GNU/Linux, MacOS	C o C++	Bajo o Alto según se requiera mediante el uso de librerías	Mediante código o mediante interfaz gráfica usando STM32CUBEMX	Si	Desarrollo de sistemas de IoT y de tiempo real
ARM Keil μ Vision 5	Windows	C	Bajo	Solamente mediante código	Si	Desarrollo de sistemas de tiempo real

Tabla 6 Entornos de desarrollo para NUCLEO-F767ZI, comparación. Fuente: Elaboración propia.

La elección se llevó a cabo en base a una comparación entre los tres entornos de desarrollo mencionados, tomando en cuenta los criterios expresados en la tabla 6. Se eligió STM32CUBEIDE, el cual es mostrado en la ilustración 36. Fue elegido principalmente, considerando que posee facilidades específicas, las cuales están orientadas al desarrollo de software en placas de la marca ST Microelectronics, ya que dicha empresa es tanto el fabricante de la placa NUCLEO-F767ZI como el desarrollador del entorno.

Entre las facilidades integradas en STM32CUBEIDE, se destaca la posibilidad de configurar los periféricos del microcontrolador mediante la interfaz gráfica STM32CUBEMX, así como la posibilidad de usar abstracciones de nivel alto para simplificar el desarrollo de software donde sea necesario, pero con la posibilidad de obviarlas donde se necesite un mayor control del hardware y un acceso a nivel más bajo.

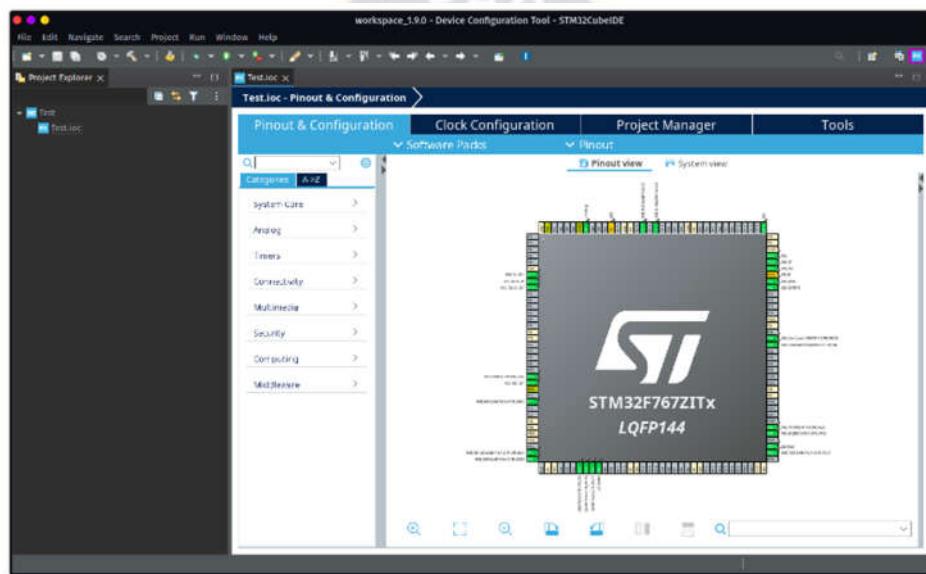


Ilustración 36 STM32CUBEIDE con la herramienta STM32CUBEMX. Fuente: Elaboración propia.

3.6. Implementación del sistema de control

En esta sección, se describe a detalle la implementación de los diferentes módulos de la arquitectura del sistema descrita a lo largo de la selección 3.1, usando las herramientas de hardware y de software elegidas en las dos secciones anteriores.

3.6.1. Proceso de decisión Markov del Péndulo Invertido

A continuación, se muestra la manera en la que el lazo de control del péndulo invertido mediante aprendizaje reforzado, fue enmarcado como un proceso de decisión de Markov, especificando el espacio de estados S , el espacio de acciones \mathcal{A} y la forma de la función de recompensa \mathcal{R} .

3.6.1.1. Espacio de Estados

Para definir el espacio de estados S , se usó el esquema simplificado del péndulo invertido mostrado en la ilustración 37, el cual muestra las medidas de la pista sobre la cuales el carro se puede desplazar. También, muestra la elección de los orígenes de las coordenadas usadas para medir tanto la posición x del carro como la posición angular del péndulo θ .

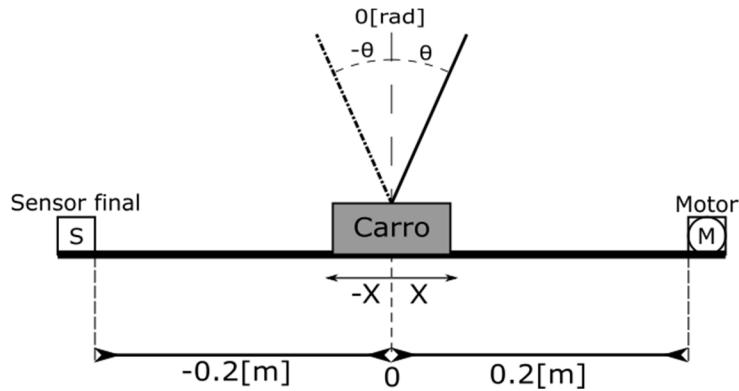


Ilustración 37 Esquema del péndulo invertido adquirido. Fuente: Elaboración propia.

Tomando en cuenta la base teórica desarrollada en la sección 2.3, el espacio de estados del péndulo invertido fue definido como un espacio vectorial en \mathbb{R}^4 , que posee la forma $[x \ \dot{x} \ \theta \ \dot{\theta}]$. Para dicho espacio de estados, se definió los valores máximos y mínimos que pueden tomar la posición del carro x , la velocidad del carro \dot{x} , la posición angular del péndulo θ y la velocidad angular del mismo $\dot{\theta}$ en unidades del sistema internacional, los cuales se muestran a continuación.

En base a los límites físicos de la pista sobre la cual se desplaza el carro del péndulo invertido, mostrados en la ilustración 37, se definió el espacio de estados para la variable de posición del carro x dejando 1[cm] de margen de seguridad al lado derecho, ya que a ese lado no existe un sensor de final de carrera, obteniendo la expresión para el espacio de estados en x que se muestra a continuación.

$$x \in [-0.19, 0.2][m] \quad (3.6.1)$$

En base a la expresión (3.6.1), se tomó como estados terminales del proceso de decisión de Markov, todo estado en el cual la posición del carro se encuentra fuera de los límites establecidos, es decir todo estado para el cual $x > 0.2 \vee x < -0.19$.

Para definir el espacio de estados de la velocidad del carro \dot{x} , primeramente, se delimitó la velocidad máxima del motor paso a paso con un valor de $10000 \frac{\text{pasos}}{\text{s}}$. Usando dicho límite de velocidad de rotación, se calculó la velocidad máxima del carro, obteniendo el intervalo de valores para \dot{x} expresado en (3.6.2).

$$\dot{x} \in [-0.3122, 0.3122] \left[\frac{m}{s} \right] \quad (3.6.2)$$

En base a la ilustración 37, para medir la posición angular del péndulo θ , se definió la posición vertical hacia arriba como el origen, con un valor igual a $0[\text{rad}]$. El desplazamiento angular del péndulo en el sentido de las manecillas del reloj incrementa el ángulo hasta llegar a π en la posición hacia abajo, mientras que desplazamiento angular del péndulo en la dirección contraria reduce el ángulo hasta llegar a $-\pi$ en la posición hacia arriba, intervalo que es expresado en (3.6.3).

$$\theta \in [-\pi, \pi][m] \quad (3.6.3)$$

Finalmente, para definir el espacio de estados de la variable de velocidad angular del péndulo $\dot{\theta}$, se tomo un limite de 18 $\left[\frac{rad}{s}\right]$ como velocidad máxima, como se muestra en el intervalo (3.6.4). Dicho limite fue impuesto con el fin de evitar que el péndulo gire a velocidades altas, protegiendo al sensor.

$$\dot{\theta} \in [-18, 18] \left[\frac{rad}{s}\right] \quad (3.6.4)$$

En base al intervalo (3.6.4), se tomó como estados terminales en el proceso de decisión de Markov todo estado en el cual la velocidad angular del péndulo se encuentra fuera de los límites establecidos, es decir todo estado para el cual $|\dot{\theta}| > 18$.

3.6.1.2. Espacio de Acciones

El espacio de acciones \mathcal{A} fue definido como un espacio continuo sobre el conjunto de los números reales \mathbb{R}^1 tal que cualquier acción a se encontrará dentro el intervalo expresado a continuación.

$$a \in [-1.0, 1.0] \quad (3.6.5)$$

En base a (3.6.5), se aclara que cualquier acción con valor positivo mueve el carro hacia la derecha, mientras que una acción con valor negativo mueve el carro hacia la izquierda. Toda acción con valor absoluto de 1.0 ocasiona que el motor gire a la velocidad máxima definida de 10000 $\left[\frac{pasos}{s}\right]$, lo cual genera un movimiento del carro a $0.3122 \left[\frac{m}{s}\right]$ hacia cualquiera de los dos costados.

Se determinó el espacio de acciones con los limites expresados en (3.6.5) con el fin de simplificar la manera en la cual la red política es implementada, ya que para generar comandos numéricos como requiere el subsistema de medición y control, cualquier acción a generada por el agente entre -1.0 y 1.0 puede ser escalada por el factor de 10000.

3.6.1.3. Función de Recompensa

Como se mencionó en la sección 2.2, la función de recompensa debe codificar de manera implícita el objetivo que se desea llevar a cabo mediante el agente de aprendizaje reforzado profundo. La función de recompensa diseñada para representar el objetivo de llevar al péndulo a la posición vertical y mantenerlo en cercanía de la misma se muestra en la ecuación (3.6.6).

$$r(x, \theta, \dot{\theta}) = \begin{cases} -c_1\theta^2 - c_2x^2 - c_3|\dot{\theta}| - 10000 * B; |\theta| \geq 0.15[rad] \\ 1000 - c_1\theta^2 - c_2x^2 - c_3|\dot{\theta}| - 10000 * B; |\theta| \leq 0.15[rad] \end{cases} \quad (3.6.6)$$

(1) (2) (3) (4) (5)

A continuación, se describe cada término de la función de recompensa (3.6.6) diseñada, explicando la información que proporciona al agente acerca del objetivo.

- El termino (1), asigna una recompensa positiva y constante por cada instante de tiempo que el agente interactúa con el entorno y el péndulo se encuentra en la posición vertical hacia arriba. La recompensa proporcionada por este término se hace cercana a 1000 una vez que ha alcanzado dicha posición, la cual fue considerada entre $-0.15[rad]$ y $0.15[rad]$. Esto informa

al agente que debe buscar alcanzar y mantener la posición vertical para recolectar una recompensa substancialmente alta.

- El término (2) asigna una recompensa negativa proporcional al cuadrado del ángulo del péndulo, penalizando al agente de una manera más agresiva cuanto más alejado este el péndulo de la posición vertical hacia arriba. Este término informa al agente qué tan lejos está de alcanzar el objetivo y que debe buscar llevar el péndulo lo más cercano a la posición vertical para evitar ser penalizado fuertemente.
- El término (3) asigna una recompensa negativa proporcional al cuadrado de la posición del carro, penalizando al agente de una manera más agresiva cuanto más alejado este el carro del centro. Este término informa al agente que debe procurar no alejar demasiado el carro del centro de la pista.
- El término (4) asigna una recompensa negativa proporcional al valor absoluto de la velocidad angular del péndulo. Esto informa al agente que no debe incrementar la velocidad del péndulo a valores demasiado altos.
- El término (5) asigna una recompensa fuertemente negativa en el caso que se haya alcanzado un estado terminal. La variable auxiliar denominada bandera **B**, indica si se ha alcanzado un estado terminal según la ecuación que se muestra a continuación.

$$B = \begin{cases} 1; & |x| \geq 0.2[m] \vee |\dot{\theta}| \geq 18 \left[\frac{\text{rad}}{\text{s}} \right] \\ 0; & |x| < 0.2[m] \vee |\dot{\theta}| < 18 \left[\frac{\text{rad}}{\text{s}} \right] \end{cases} \quad (3.6.7)$$

La bandera **B** se encuentra diseñada para indicar al agente que no debe mover el carro más allá de los límites de seguridad laterales, y que tampoco debe dejar que el péndulo alcance velocidades altas, caso contrario será penalizado fuertemente por arriesgar dañar la planta.

3.6.2. Implementación del Entorno

Dentro esta sección, se describe a la manera en la que se implementaron tanto el subsistema de medición y control como el módulo de interfaz agente-entorno; los dos elementos que conforman la arquitectura propuesta para el entorno.

3.6.2.1. Implementación del Subsistema de Medición y Control

Para implementar el subsistema de medición y control, se hizo amplio uso de las capacidades de hardware presentes en la placa Nucleo F767ZI, usando un esquema basado solamente en interrupciones para realizar las diferentes tareas descritas en la sección 3.1.5.2.

En el programa implementado, el microcontrolador y sus diferentes periféricos hardware son configurados de la manera descrita mediante la tabla 7.

Periféricos del Microcontrolador Utilizados		
Periférico	Modo	Función
Interfaz Serial USART 3	Interfaz serial a 921600 $\frac{baud}{s}$ con acceso directo a memoria y generación de interrupciones	Enviar datos de los sensores y recibir comandos numéricos
Temporizador TIM2	Contador rápido bidireccional con entrada en el pin PA5	Contar los pulsos enviados al driver del motor paso a paso
Temporizador TIM3	Generador de PWM a con salida en el pin PC6	Generar trenes de pulsos a frecuencias variables para controlar el motor paso a paso
Temporizador TIM4	Temporizador con interrupción generada a intervalo de 10[ms]	Generar interrupción para enviar datos de los sensores con frecuencia de muestreo $T_s = 10[ms]$
Temporizador TIM5	Interfaz de encoder modo resolución 4x con entradas en el pin PA0 para la fase 1 del encoder y pin PA1 para la fase 2	Llevar la cuenta de pulsos del sensor de posición del péndulo
Puerto PC8	Salida Digital	Generar una señal digital para controlar la dirección del motor
Puerto PB11	Entrada digital	Leer el estado del sensor de final de carrera

Tabla 7 Periféricos del microcontrolador utilizados. Fuente: Elaboración propia.

El subsistema de medición y control recibe comandos numéricos enviados desde el módulo de interfaz agente-entorno como números enteros de 32 bits (4 bytes), dichos comandos son recibidos a través de USART3; la interfaz de comunicación serial guarda los 4 bytes a RAM mediante acceso directo a memoria (DMA) y genera una interrupción cuando se ha completado la recepción del cuarto byte.

Comandos Numéricos		
Comando	Nombre	Función
1 a 10000	Comando de frecuencia positiva	Mover el motor en sentido horario con frecuencia igual al valor del comando
0	Comando de stop	Detener el motor
-1 a -10000	Comando de frecuencia negativa	Mover el motor en sentido antihorario con frecuencia igual al valor del comando
32768	Comando de Inicio	Iniciar la cuenta del temporizador TIM4
65536	Comando de reseteo	Detener la cuenta del temporizador TIM4 y correr rutina "hallar_medio" para devolver el carro al centro de la pista

Tabla 8 Comandos numéricos del subsistema de medición y control. Fuente: Elaboración propia.

Los comandos numéricos recibidos pueden ser comandos que indican al subsistema de medición y control mover al motor en uno de los dos sentidos a una frecuencia especificada por el comando, pero también existen dos comandos especiales; estos están diseñados para indicar cuando se debe empezar a enviar datos de los sensores y cuando detener el envío para devolver el péndulo invertido a su estado inicial. La tabla 8 detalla todos los comandos que pueden ser recibidos por el subsistema de medición y control.

Los comandos mostrados en la tabla 8, son identificados dentro la rutina de atención a la interrupción de UART3 denominada `HAL_UART_RxCpltCallback`. El nombre de dicha rutina es usado por convención de STM32CubeIDE. El algoritmo usado para implementarla se detalla en el algoritmo 6.

Algoritmo 6 HAL_UART_RxCpltCallback

```

1: si comando  $\geq -10000$  o comando  $\leq 10000$  entonces
2:   Ejecutar frecuencia_driver(comando)
3: else si comando = 32768 entonces
4:   Detener cuenta TIM4
5:   Ejecutar hallar_medio()
6: else si comando == 65536 entonces
7:   Iniciar cuenta TIM4
8: fin de si

```

Dentro el algoritmo 6, primeramente, para lograr ejecutar los tres primeros tipos de comandos, se ejecuta una llamada a una rutina denominada *frecuencia_driver*, la cual consiste en un algoritmo que escribe a los registros de control de frecuencia de TIM3 y al puerto PC8, que permiten controlar el driver del motor, accionándolo en el sentido correcto a la frecuencia especificada por el comando. Esta rutina también modifica el sentido de conteo de TIM2, la manera en la que fue implementada se describe en el algoritmo 7.

Algoritmo 7 frecuencia_driver(frecuencia)

Entradas: *frecuencia*

```

1: Cambiar la frecuencia de TIM3 a  $|frecuencia|$ 
2: si frecuencia > 0 entonces
3:   Escribir 1 a Puerto PC8
4:   Escribir 0 a bit de dirección de conteo de TIM2
5: else
6:   Escribir 1 a Puerto PC8
7:   Escribir 1 a bit de dirección de conteo de TIM2
8: fin de si

```

El comando de inicio ejecuta una función prevista por la librería estándar del microcontrolador, la cual inicia el temporizador TIM4, por lo tanto, no posee algoritmo.

Finalmente, para implementar la rutina *hallar_medio* ejecutada cuando se recibe un comando de reseteo, primeramente, se debió calcular el número de pasos del motor desde cualquier extremo al centro, denominado *n_pasos_centro*. Para calcular dicha cantidad, se usó la ecuación (3.6.8).

$$n_{\text{pasos centro}} = \frac{n_{\text{pasos por revolucion}} d_{\text{centro}}}{2\pi r_{\text{polea}}} \quad (3.6.8)$$

Para realizar el cálculo, se midió la variable $r_{\text{polea}} = 0.0159[\text{m}]$, la cual es el radio de la polea acoplada al eje del motor. Se uso $n_{\text{pasos por revolucion}} = 3200$, el cual es el número de pasos que el motor da por cada revolución de su eje, tomando en cuenta un factor de micropasos igual a 16 en el driver. Finalmente, se observó que existe una distancia de un extremo al centro de la pista al centro $d_{\text{centro}} = 0.2[\text{m}]$. Reemplazando los tres datos, en la ecuación (3.6.8), se obtuvo el resultado $n_{\text{pasos centro}} = 6726$.

Para implementar la rutina `hallar_medio`, se ideó que se debe mover el carro hacia la izquierda hasta chocar con el sensor de final de carrera, alcanzando el extremo izquierdo. Seguidamente, se debe mover el carro hacia la derecha hasta que el contador TIM2 haya alcanzado un número de pulsos igual al valor calculado de $n_{\text{pasos centro}}$. La rutina `hallar_medio` se detalla en el algoritmo 8.

Algoritmo 8 `hallar_medio()`

```

1: Ejecutar frecuencia_driver(-4000)
2: while Puerto PB11 = 1 do
3:   NOP
4: fin de while
5: Ejecutar frecuencia_driver(4000)
6: while Cuenta TIM2 ≠  $n_{\text{pasos centro}}$  do
7:   NOP
8: fin de while
9: Ejecutar frecuencia_driver(0)
10: Cuenta TIM2 ← 0

```

Para enviar los datos de los sensores al módulo de interfaz agente entorno, se ideó el uso de una trama con longitud de 7 bytes, la cual se muestra mediante la ilustración 38 y describe a continuación.

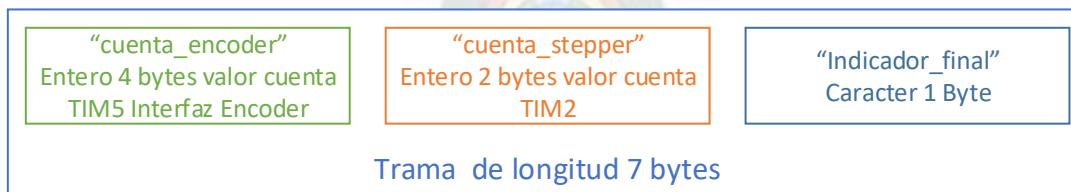


Ilustración 38 Formato de trama enviado por el subsistema de medición y control. Fuente: Elaboración propia.

- **cuenta_encoder:** Contiene la cuenta llevada a cabo por la interfaz de encoder, la cual es proporcional al ángulo del péndulo. Esta puede ser negativa o positiva y puede ser un valor alto, con la posibilidad de ser muy superior a los 4000 $\left[\frac{\text{pulsos}}{\text{revolucion}} \right]$ que se pueden obtener del encoder con la interfaz en modo de resolución 4x, ya que existe la posibilidad que el péndulo lleve a cabo decenas de revoluciones. Por lo tanto, se la almacena en un número entero de 32 bits.

- **cuenta stepper:** Contiene la cuenta llevada por el temporizador TIM2; su entrada se encuentra conectada directamente a la salida de TIM3, por lo tanto contiene el número de pasos llevados a cabo por el motor con respecto al centro de la pista; esta puede ser negativa o positiva, pero nunca mayor a **n_pasos_centro**, por lo tanto, se la almacena en un número entero de 16 bits.
- **indicador_final:** Es una bandera que posee dos estados: **0x00** si el sensor de final de carrera no se ha activado y **0xFF** si el mismo ha sido activado. Por lo tanto, es necesario solo 1 byte para almacenar su información.

Una vez que se ha recibido el comando de inicio, el temporizador TIM4 genera una interrupción cada $10[ms]$. Para atender dicha interrupción, se usa la rutina descrita en el algoritmo 9, la cual primeramente copia las cuentas de los temporizadores TIM5 y TIM2 a un espacio de memoria denominado **buffer_tx** de 7 bytes. Seguidamente, lee el estado del sensor de final de carrera y copia el valor correspondiente en el último byte de **buffer_tx**. Finalmente, envía a **buffer_tx** mediante la interfaz serial.

Algoritmo 9 HAL.TIM_PeriodElapsedCallback

```

1: buffer_tx[0] ← CuentaTIM5
2: buffer_tx[4] ← CuentaTIM2
3: si Puerto PB11 = 0 entonces
4:   buffer_tx[6] ← 0xFF
5: else
6:   buffer_tx[6] ← 0x00
7: fin de si
8: Enviar buffer_tx a través de USART3
  
```

3.6.2.2. Implementación del Módulo de Interfaz Agente-Entorno

El módulo de interfaz agente-entorno se implementó como una clase de Python denominada **Entorno** la cual requiere ser instanciada usando los tres parámetros descritos en la tabla 9.

Parámetros de Instanciación de “Entorno”		
Nombre	Tipo	Descripción
p_muestreo	Número de punto flotante	Valor en segundos del periodo de muestreo usado en el subsistema de medición y control
tam_ep	Número entero	Longitud máxima de experiencias elegida para cada episodio
puerto	Cadena de caracteres	Nombre del puerto usado para comunicación con el subsistema de medición y control

Tabla 9 Parámetros de instanciaación de la clase Entorno. Fuente: Elaboración propia.

Al instanciar **Entorno**, se inicializan las variables numéricas descritas mediante la tabla 10 a cero.

Variables inicializadas al instanciar “Entorno”	
Nombre	Descripción
x_ant	Valor anterior de posición del carro x_{t-1}
theta_ant	Valor anterior de posición angular del péndulo θ_{t-1}
theta_dot	Valor de velocidad angular del péndulo $\dot{\theta}_t$
theta_dot_ant	Valor anterior de velocidad angular del péndulo $\dot{\theta}_{t-1}$
pulsos_encoder_ant	Valor anterior de cuenta de pulsos de encoder $cuenta_encoder_{t-1}$
band_fin	Bandera para indicar fin de episodio f
cont_pasos	Cuenta de pasos de tiempo de episodio

Tabla 10 Variables inicializadas al instanciar la clase Entorno. Fuente: Elaboración propia.

Cuando es instanciada, la implementación del módulo de interfaz agente-entorno posee varios métodos que tienen el propósito de abstraer el funcionamiento del entorno, permitiendo realizar los intercambios de señales propios de un proceso de decisión de Markov entre el entorno y el agente.

El método principal encargado de facilitar la obtención de los datos del entorno y la transformación de los mismos en el vector de estado, la recompensa y la bandera de finalización de episodio, se denomina `est_entorno`. Dicho método realiza sus cálculos a partir de la trama de datos enviada por el subsistema de medición y control, la cual fue descrita en la sección 3.6.2.1.

Para calcular el vector de estado, el método `est_entorno` trasforma el valor de `cuenta_stepper` enviado por el subsistema de control y medición a metros. Usando el radio de la polea acoplada al eje del motor r_{polea} y el número de pasos por revolución del motor $n_{pasos\ por\ revolucion}$, la ecuación (3.6.9) fue ideada para obtener la posición x del carro a partir del valor `cuenta_stepper`.

$$x = \frac{2\pi r_{polea}}{n_{pasos\ por\ revolucion}} \text{cuenta_stepper} \quad (3.6.9)$$

Para calcular la posición angular del péndulo, se considera que el péndulo inicia en la posición vertical hacia abajo, para la cual se tiene siempre que el valor de `cuenta_encoder` es cero. En la ilustración 39 se muestra un análisis de dos casos considerados para el cálculo del ángulo; cuando `cuenta_encoder` es un valor positivo, o cuando es un valor negativo.

A partir del análisis realizado en la ilustración 39, para obtener la posición angular del péndulo θ a partir del valor `cuenta_encoder`, primeramente, se calcula el valor $\theta_n = \text{cuenta_encoder mod}(4000)$, ya que una revolución completa del encoder genera 4000 pulsos. Seguidamente, se usa la ecuación (3.6.10), derivada directamente del análisis ilustrado.

$$\theta = \begin{cases} -\frac{\pi}{2000}(\theta_n - 2000); & \theta_n \geq 0 \\ -\frac{\pi}{2000}(\theta_n + 2000); & \theta_n < 0 \end{cases} \quad (3.6.10)$$

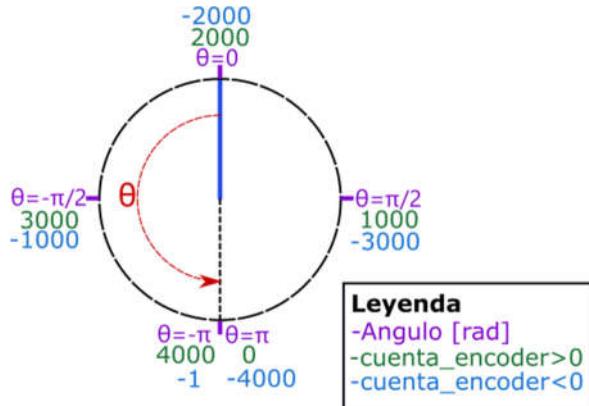


Ilustración 39 Análisis del valor de cuenta_encoder con relación al ángulo del péndulo. Fuente: Elaboración propia.

Para calcular la velocidad del carro \dot{x} , se usa la posición anterior del carro guardada x_{ant} junto al periodo de muestreo mediante la ecuación (3.6.11).

$$\dot{x} = \frac{x - x_{t-1}}{p_muestreo} \quad (3.6.11)$$

Finalmente, para calcular la velocidad angular del péndulo $\dot{\theta}$ se usa el valor de cuenta de pulsos del encoder anterior, denominado `cuenta_encoder_ant`, en la ecuación (3.6.12).

$$\dot{\theta} = \frac{2\pi}{4000p_muestreo} (cuenta_encoder - cuenta_encoder_ant) \quad (3.6.12)$$

Con el fin de reducir el ruido en las mediciones del estado relacionadas al péndulo, los valores de θ y $\dot{\theta}$ son filtrados si $-\frac{\pi}{14} \leq \theta \leq \frac{\pi}{14}$ usando una regla exponencial, mediante las ecuaciones (3.6.13) y (3.6.14).

$$\theta = \vartheta\theta + (1 - \vartheta)\theta_{t-1} \quad (3.6.13)$$

$$\dot{\theta} = \vartheta\dot{\theta} + (1 - \vartheta)\dot{\theta}_{t-1} \quad (3.6.14)$$

Para calcular la recompensa dentro `est_entorno`, se usa la ecuación (3.6.1), reemplazando en ella los datos del estado calculados mediante las ecuaciones descritas en la presente sección.

Para obtener la bandera de fin de episodio f , denominada `band_fin`, se usa la ecuación (3.6.15), la cual toma en cuenta si se ha alcanzado un estado terminal, pero también toma en cuenta si se ha alcanzado el número de pasos de tiempo máximo establecido para un episodio.

$$f = \begin{cases} 1; & |x| \geq 0.2[m] \vee |\dot{\theta}| \geq 18 \left[\frac{rad}{s} \right] \vee cont_pasos = pasos_ep \\ 0; & |x| < 0.2[m] \vee |\dot{\theta}| < 18 \left[\frac{rad}{s} \right] \vee cont_pasos \neq pasos_ep \end{cases} \quad (3.6.15)$$

Sobre la base de los análisis, cálculos y ecuaciones descritas anteriormente, el algoritmo 10 detalla el funcionamiento del método `est_ent`.

Algoritmo 10 est_ent

```

1: Leer cuenta_encoder, cuenta stepper, indicador_final del puerto serial
2:  $x \leftarrow \frac{2\pi r_{polea}}{n\_pasos\_por\_revolucion} cuenta\_stepper$ 
3: si  $0 \leq cuenta\_encodermod(4000) < 4000$  entonces
4:    $\theta \leftarrow -\frac{\pi}{2000}(cuenta\_encodermod(4000) - 2000)$ 
5: else si  $-4000 \leq cuenta\_encodermod(4000) < -1$  entonces
6:    $\theta \leftarrow -\frac{\pi}{2000}(cuenta\_encodermod(4000) + 2000)$ 
7: fin de si
8:  $\dot{x} \leftarrow -\frac{x - x_{t-1}}{p\_muestreo}$ 
9:  $\dot{\theta} \leftarrow -\frac{2\pi}{4000p\_muestreo}(cuenta\_encoder - cuenta\_encoder\_ant)$ 
10: si  $-\frac{\pi}{14} \leq \theta < \frac{\pi}{14}$  entonces
11:    $\theta \leftarrow \vartheta\theta + (1 - \vartheta)\theta_{t-1}$ 
12:    $\dot{\theta} \leftarrow \vartheta\dot{\theta} + (1 - \vartheta)\dot{\theta}_{t-1}$ 
13: fin de si
14:  $x_{t-1} \leftarrow x, \theta_{t-1} \leftarrow \theta, \dot{\theta}_{t-1} \leftarrow \dot{\theta}, pulsos\_encoder\_ant \leftarrow pulsos\_encoder$ 
15:  $r \leftarrow R(x, \theta, \dot{\theta}, indicador\_final)$ 
16: si cont_pasos > tam_ep entonces
17:    $f \leftarrow 1$ 
18: fin de si
19:  $cont\_pasos \leftarrow cont\_pasos + 1$ 
20: retornar  $([x, \dot{x}, \theta, \dot{\theta}], r, f, cont\_pasos)$ 

```

Para enviar comandos numéricos al subsistema de medición y control, Entorno posee un método denominado `enviar`, el cual toma como argumento un número entero denominado `cmd` y lo envía a través de la conexión serial.

Finalmente, tomando ventaja del método `enviar`, se implementó dos métodos especiales, los cuales son usados para iniciar un episodio y terminarlo. Dichos métodos se describen en la tabla 11.

Métodos especiales de "Entorno"	
Nombre del método	Funcionamiento
<code>comenzar</code>	Ejecutar <code>enviar(cmd)</code> para enviar el comando de inicio (32768)
<code>reset_ent</code>	Ejecutar <code>enviar(cmd)</code> para enviar el comando de reseteo (65536)

Tabla 11 Método especiales de Entorno. Fuente: Elaboración propia.

3.6.3. Implementación del Agente

Se describe a continuación la manera en la se implementaron tanto el módulo de memoria como el módulo de redes neuronales, los cuales son usados por la clase agente para implementar la arquitectura definida para el agente.

3.6.3.1. Implementación del Módulo de Memoria de Experiencias

El módulo de memoria de experiencias fue implementado como una clase de Python denominada `Memoria`, la cual requiere ser instanciada usando los dos parámetros descritos en la tabla 12.

Parámetros de Instanciación de “Memoria”		
Nombre	Tipo	Descripción
tamano	Número entero	Tamaño del búfer de reproducción t_{mem}
tam_estado	Número entero	Longitud del vector de estado

Tabla 12 Parámetros de instanciaión de la clase Memoria. Fuente: Elaboración propia.

Al instanciar `Memoria` con los parámetros necesarios, se inicializan las variables y objetos usados para el almacenamiento de experiencias descritos en la tabla 13.

Variables y objetos inicializados al instanciar “Memoria”	
Nombre	Descripción
almacen	Búfer de reproducción vacío, implementado mediante un arreglo bidimensional de numpy para almacenar experiencias
puntero	Puntero para indicar el numero de experiencias que se encuentran ya almacenadas en el búfer
band_lleno	Bandera que indica si el búfer de reproducción fue llenado por lo menos una vez

Tabla 13 Variables y objetos inicializados al instanciar la clase Memoria. Fuente: Elaboración propia.

Dado que cada experiencia almacenada en el búfer de reproducción es una tupla de la forma (s_t, a, r, s_{t+1}, f) y que tanto s_t como s_{t+1} son vectores de estado con dimensión `tam_estado`, entonces el búfer de reproducción denominado `almacen` debe ser instanciado como un arreglo bidimensional, el cual debe tener dimensiones $t_{mem} \times (2 * tam_estado + 3)$.

Para almacenar una experiencia al búfer de reproducción, se implementó un método dentro el módulo de memoria de experiencias denominado `almacenar_paso`, el cual toma los argumentos de entrada mostrados en la tabla 14.

Argumentos de ”almacenar_paso”		
Nombre	Tipo	Descripción
estado	Vector de numpy con dimensiones $1 \times tam_estado$	Vector de estado para instante de tiempo $t s_t$
accion	Número de punto flotante	Acción ejecutada por el agente en instante de tiempo $t a_t$
recompensa	Número de punto flotante	Recompensa obtenida por transición de estado en el instante de tiempo $t + 1 r_{t+1}$
estado_sig	Vector de numpy con dimensiones $1 \times tam_estado$	Vector de estado para instante de tiempo $t + 1 s_{t+1}$
fin	Número de punto flotante	Bandera que indica si el estado s_{t+1} alcanzado durante la experiencia fue un estado final en el episodio

Tabla 14 Argumentos de entrada del método `almacenar_paso`. Fuente: Elaboración propia.

El funcionamiento del método `almacenar_paso` se detalla en el algoritmo 11.

Algoritmo 11 `almacenar_paso`

Entradas: estado, accion, recompensa, estado.sig, fin

- 1: `datos` \leftarrow [estado, accion, recompensa, estado.sig, fin]
 - 2: `almacen[puntero]` \leftarrow `datos`
 - 3: `puntero` \leftarrow `puntero` + 1
 - 4: **si** `puntero` = `tamano` **entonces**
 - 5: `puntero` \leftarrow 0
 - 6: `band_lleno` \leftarrow 1
 - 7: **fin de si**
-

Del algoritmo 11, se observa que cuando el búfer de reproducción es llenado, entonces el puntero es reestablecido con cero, para apuntar al inicio de dicho búfer, comenzando a sobrescribir las primeras experiencias almacenadas.

Para recuperar un lote de experiencias del búfer de reproducción durante el proceso de entrenamiento, se implementó en el módulo de memoria de experiencias un método denominado `tomar_muestras`, el cual toma un único argumento de entrada descrito en la tabla 15.

Argumento de "tomar_muestras"		
Nombre	Tipo	Descripción
<code>num_muestras</code>	Número entero	Número de experiencias tomadas aleatoriamente del búfer de experiencias para formar el lote

Tabla 15 Argumento de entrada del método `tomar_muestras`. Fuente: Elaboración propia

El funcionamiento del método `tomar_muestras` se detalla en el algoritmo 12.

Algoritmo 12 `tomar_muestras`

Entradas: num_muestras

- 1: **si** `band_lleno`=1 **entonces**
 - 2: `muestras[0 : num_muestras]` \sim `almacen`
 - 3: **else**
 - 4: `muestras[0 : num_muestras]` \sim `almacen[0 : puntero]`
 - 5: **fin de si**
 - 6: **retornar** `muestras`
-

En el algoritmo anterior, se tuvo cuidado en tomar las muestras del búfer de reproducción entero, solo en caso que este haya sido llenado por lo menos una vez, ya que, si no se hubiese implementado dicha precaución, se podría tomar datos de experiencias para las cuales todos los valores son cero; los cuales no contienen información que ayude al agente a evaluar y mejorar su rendimiento, e incluso podrían ser perjudiciales para el proceso de entrenamiento.

Adicionalmente, se implementaron dos métodos de carácter utilitario en `Memoria`; tienen el fin de permitir guardar el búfer de experiencias para pausar el entrenamiento, y cargarlo para continuar cuando sea necesario. Ambos métodos son descritos en la tabla 16.

Métodos utilitarios de "Memoria"		
Nombre del método	Argumento de entrada	Funcionamiento
guardar	<code>direccion</code> : Literal que especifica una ruta de almacenamiento del sistema operativo	Guardar almacén, puntero y band_lleno a la ruta en disco duro especificada mediante el argumento <code>direccion</code> usando el formato comprimido de NumPy "npz"
cargar	<code>direccion</code> : Literal que especifica una ruta de almacenamiento del sistema operativo	Cargar almacén, puntero y band_lleno desde la ruta en disco duro especificada mediante el argumento <code>direccion</code>

Tabla 16 Métodos utilitarios de la clase Memoria. Fuente: Elaboración propia.

3.6.3.2. Implementación del Módulo de Redes Neuronales

El módulo de redes neuronales fue implementado mediante dos clases denominadas `RedActor` y `RedCritico`, las cuales son instanciadas múltiples veces para poder llevar a cabo el algoritmo de TD3; una vez instanciadas según lo requerido, estas se usan para aproximar una la política y la función de valor de estado-acción a lo largo del proceso de entrenamiento del agente.

Dado que `RedActor` debe representar una red neuronal que aproxima un mapeo del estado a una acción, la arquitectura debe tomar como entrada el vector de estado s de la forma $[x \ \dot{x} \ \theta \ \dot{\theta}]$ y transformarlo en una acción adimensional $a \in [-1.0, 1.0]$; dicha acción puede ser después escalada a un comando de frecuencia para ser enviado al subsistema de medición y control.

Tomando en cuenta que se desea generar la acción en el menor tiempo posible, se limitó la profundidad de capas escondidas de la red neuronal de `RedActor` a dos. Esta decisión para su implementación también tiene el propósito de acelerar la convergencia de la política a lo largo del proceso de entrenamiento.

Sin embargo, con el fin de permitir un proceso de experimentación e iteración rápida, se incluyó la posibilidad de instanciar las capas de `RedActor` con un número variable de neuronas, permitiendo modificar la arquitectura de la red neuronal de esa manera.

La implementación de `RedActor`, fue realizada mediante una clase de Python, la cual debe ser instanciada usando los parámetros descritos en la tabla 17.

Parámetros de Instanciación de “RedActor”		
Nombre	Tipo	Descripción
tam_estado	Número entero	Longitud del vector de estado para la capa de entrada $n^{[i]}$
tam_capas	Lista de Python de longitud 2 con números enteros	Números de neuronas para las dos capas escondidas $[n^{[1]}, n^{[2]}]$
tas_apren	Número de punto flotante	Tasa de aprendizaje para el optimizador de la red
procesador	Objeto de tipo torch.device	Nombre del dispositivo usado para llevar a cabo las operaciones de la red neuronal, puede ser el CPU o una GPU

Tabla 17 Parámetros de instanciación de la clase RedActor. Fuente: Elaboración propia.

La arquitectura de red neuronal ideada para RedActor es resumida en la tabla 18 y se muestra a continuación en la ilustración 40.

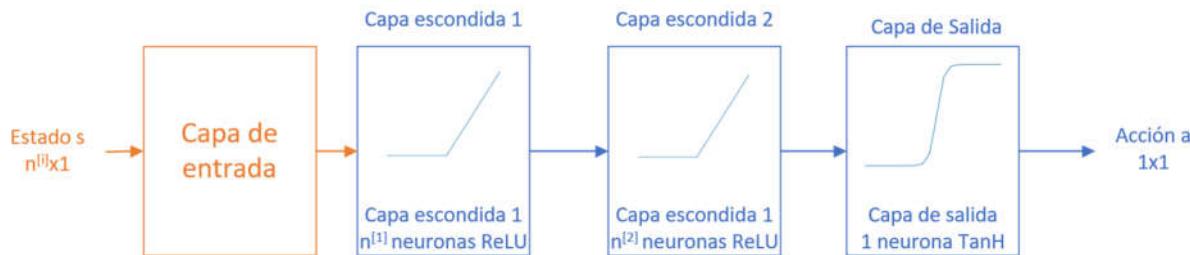


Ilustración 40 Arquitectura de RedActor. Fuente: Elaboración propia.

Arquitectura de “RedActor” - Resumen		
Nombre capa	Número de neuronas	Función de activación
Capa de entrada	<i>tam_estado</i>	Ninguna
Capa Escondida 1	$n^{[1]}$	Unidad lineal rectificada (ReLU)
Capa Escondida 2	$n^{[2]}$	Unidad lineal rectificada (ReLU)
Capa de salida	1	Tangente hiperbólica (TanH)

Tabla 18 Arquitectura de RedActor. Fuente: Elaboración propia.

La clase RedCritico debe representar una red neuronal que aproxima un mapeo de un estado y una acción a un valor esperado del retorno. Por lo tanto, la arquitectura de su red neuronal debe tomar como entrada el vector de estado s junto a la acción a ejecutada en ese estado y transformarlos en un estimado del valor de estado-acción $Q(s, a)$, el cual no se encuentra acotado.

Como en el caso de RedActor, en la implementación de RedCritico, se usó una arquitectura con dos capas escondidas que permite especificar un número variable de neuronas en cada capa al instanciarse. Por lo tanto, los parámetros de instanciación para la arquitectura de RedCritico son los mismos que aquellos usados para RedActor, y se encuentran resumidos en la tabla 17.

Sin embargo, la arquitectura de la red neuronal tuvo que ser diseñada de una manera diferente, dado que se aproxima una función con dimensiones de entrada diferentes y una salida no acotada.

La arquitectura de red neuronal ideada para RedCritico es resumida en la tabla 19 y se muestra a continuación en la ilustración 41.

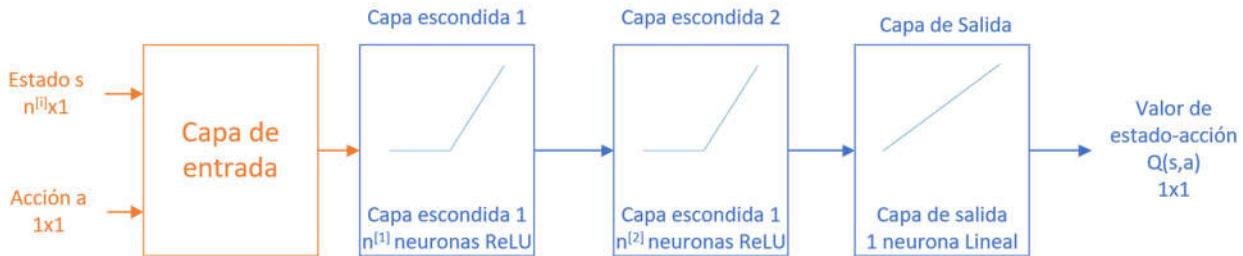


Ilustración 41 Arquitectura de RedCritico. Fuente: Elaboración propia.

Arquitectura de “RedCritico” - Resumen		
Nombre capa	Numero de neuronas	Función de activación
Capa de entrada	$tam_estado + 1$	Ninguna
Capa Escondida 1	$n^{[1]}$	Unidad lineal rectificada (ReLU)
Capa Escondida 2	$n^{[2]}$	Unidad lineal rectificada (ReLU)
Capa de salida	1	Lineal

Tabla 19 Arquitectura de RedCritico. Fuente: Elaboración propia.

Dado que no se conoce los valores límite entre los cuales el valor de estado-acción se puede encontrar, se usó una función de activación lineal para la capa de salida de RedCritico, mientras que en ambas arquitecturas se utilizó funciones de activación ReLU para las capas escondidas.

Ambas redes neuronales fueron implementadas como clases, usando código similar a aquel presentado en la sección 3.5.2, esto significa que cada red posee un método `forward` para obtener su respectiva salida. Sin embargo, con el fin de poder pausar el entrenamiento y continuarlo en otro momento, se implementaron dos métodos extra en cada clase de red neuronal, los cuales se muestran en la tabla 20.

Métodos de guardado y cargado de las redes neuronales		
Nombre del método	Argumento de entrada	Funcionamiento
guardar	direccion: Literal que especifica una ruta de almacenamiento del sistema operativo	Guardar los parámetros de la red neuronal y el estado del optimizador a la ruta especificada como un archivo de Pytorch con extensión .pt
cargar	direccion: Literal que especifica una ruta de almacenamiento del sistema operativo	Cargar los parámetros de la red neuronal y el estado del optimizador desde la ruta especificada.

Tabla 20 Métodos de guardado y cargado de las redes neuronales. Fuente: Elaboración propia.

3.6.3.3. Implementación de la Clase Agente

La clase `Agente` implementa el algoritmo de TD3 con todos los componentes que este requiere, tomando ventaja de las tres clases anteriormente descritas para instanciar dichos componentes y llevar el proceso de entrenamiento.

La implementación de la clase `Agente`, fue diseñada para ser instanciada usando los parámetros descritos en la tabla 21. La posibilidad de cambiar dichos parámetros permite instanciar al agente con diferentes configuraciones, las cuales permiten experimentar y variar el proceso de entrenamiento.

Parámetros de Instanciación de “Agente”		
Nombre	Tipo	Descripción
<code>rho</code>	Número de punto flotante	Parametro ρ para llevar a cabo las actualizaciones de las redes objetivo mediante la media de Polyak
<code>gamma</code>	Número de punto flotante	Tasa de descuento γ para estimación del retorno descontado
<code>interval_act_actor</code>	Número entero	Parámetro p_{retard} ; define cuantos pasos de entrenamiento de las redes de valor se llevan a cabo por cada paso de entrenamiento de la red política
<code>entrenamientos_ep</code>	Número entero	Número de pasos de entrenamiento llevados a cabo después de cada episodio
<code>tam_memoria</code>	Número entero	Tamaño de la memoria de experiencias; numero máximo de experiencias almacenables.
<code>tam_lote</code>	Número entero	Número de experiencias tomadas de la memoria de experiencias para cada paso de entrenamiento

Tabla 21 Parámetros de instanciación de la clase Agente. Fuente: Elaboración propia.

Al instanciar `Agente` con los parámetros necesarios descritos en la tabla 21, se inicializan las variables y objetos descritos en la tabla 22, los cuales toman ventaja de los módulos detallados en las dos secciones anteriores. Son usados para llevar a cabo el proceso de entrenamiento según lo requerido por el algoritmo de TD3.

Dentro la clase agente, es posible a su vez modificar los parámetros bajo los cuales cada uno de los objetos de la tabla 22 son instanciados, por lo cual es posible cambiar de manera rápida las arquitecturas de las redes neuronales usadas para la política y las funciones de valor de estado acción, o los parámetros del módulo de memoria de experiencias.

Variables y objetos inicializados al instanciar “Agente”	
Nombre	Descripción
actor	Instancia de la clase RedActor usada como red política del agente $\pi^\varphi(s)$
actor_obj	Instancia de la clase RedActor usada como red objetivo $\pi^{\varphi obj}(s)$ de la política
critico_1	Instancia de la clase RedCritico usada como red de valor de estado-acción $Q_1^\beta(s, a)$
critico_1_obj	Instancia de la clase RedCritico usada como red objetivo $Q_1^{\beta obj}(s, a)$ de la red de valor de estado-acción 1
critico_2	Instancia de la clase RedCritico usada como red de valor de estado-acción $Q_2^\beta(s, a)$
critico_2_obj	Instancia de la clase RedCritico usada como red objetivo $Q_2^{\beta obj}(s, a)$ de la red de valor de estado-acción 2
mem	Instancia de Memoria con búfer de experiencias de tamaño definido por tam_memoria

Tabla 22 Variables y objetos inicializados al instanciar la clase Agente Fuente: Elaboración propia.

De la manera en que la clase Agente fue implementada, provee un método denominado `entrenar`, el cual se encuentra encargado de llevar a cabo un numero de pasos de entrenamiento definido mediante la variable `entrenamientos_ep`. Su funcionamiento se detalla en el algoritmo 13.

Algoritmo 13 `entrenar`

- 1: **for** `entrenamiento < entrenamientos_ep` **do**
 - 2: Tomar un lote de `tam_lote` experiencias aleatorias del modulo de memoria ejecutando `tomar_muestras`
 - 3: Calcular las acciones objetivo: a_{t+1}^{obj}
 - 4: Calcular el mínimo entre ambas redes neuronales objetivo: $min_Q(s_{t+1})$
 - 5: **si** $f = 1$ en una tupla $(s_t, a_t, r_{t+1}, s_{t+1}, f)$ **entonces**
 - 6: $min_Q(s_{t+1}) = 0$ para esa tupla
 - 7: **fin de si**
 - 8: Calcular la función de perdida para cada red de valor de estado-acción L_1, L_2
 - 9: Calcular la perdida global de estado-acción $L_{criticos} = L_1 + L_2$
 - 10: Calcular el gradiente de la perdida de estado acción $\nabla L_{criticos}$ y usarla para realizar un paso de entrenamiento de las redes de valor de estado-acción $Q_{1,2}$
 - 11: **si** Si $entrenamiento \bmod interval_act_actor = 0$ **entonces**
 - 12: Calcular la función objetivo de la red política J
 - 13: Calcular el gradiente ∇J y realizar un paso de entrenamiento sobre la red politica μ^φ
 - 14: Copiar los parámetros a las redes objetivo usando la media de Polyak
 - 15: **fin de si**
 - 16: Guardar las redes neuronales y el modulo de memoria
 - 17: **fin de for**
-

Finalmente, la clase `Agente` provee también cuatro métodos adicionales, los cuales se encuentran encargados de generar acciones, almacenar experiencias en el módulo de memoria de experiencias, y cargar y guardar el estado de los objetos. Dichos métodos se resumen en la tabla 23.

Métodos adicionales de "Agente"			
Nombre del método	Argumentos(s) de entrada	Funcionamiento	
<code>generar_accion</code>	<code>estado</code> ; vector de estado	Usar a <code>actor</code> para obtener una acción a partir de <code>estado</code>	
<code>almacenar_paso</code>	<code>s, a, r, s.sig, fin</code> ; datos de una experiencia	Almacenar una experiencia en <code>mem</code>	
<code>guardar</code>	Ninguna	Guardar los parámetros de las redes neuronales <code>actor</code> , <code>actor_obj</code> , <code>critico_1</code> , <code>critico_1_obj</code> , <code>critico_2</code> , <code>critico_2_obj</code> y de <code>mem</code> a disco duro	
<code>cargar</code>	Ninguna	Cargar los parámetros de las redes neuronales <code>actor</code> , <code>actor_obj</code> , <code>critico_1</code> , <code>critico_1_obj</code> , <code>critico_2</code> , <code>critico_2_obj</code> y de <code>mem</code> desde disco duro	

Tabla 23 Métodos de la clase `Agente`. Fuente: Elaboración propia.

3.6.4. Implementación del Módulo de Exploración

El módulo de exploración fue implementado mediante una clase denominada `Exploracion`, la cual está destinada a ayudar a mitigar los efectos del dilema exploración-explotación, generando nuevas acciones de manera que el agente pueda tomar parte en nuevas experiencias para su entrenamiento. Esto se logra a través de 3 estrategias descritas a continuación.

- Generar acciones empezando de $a = -1.0$, incrementando su valor progresivamente en pasos de 0.1 para cada episodio que se lleva a cabo, hasta llegar a $a = 1.0$. Esto ocasiona que durante los episodios en los cuales se usa esta estrategia de exploración arbitraria, el carro se mueva a distintas velocidades hasta alcanzar los extremos de la pista, generando experiencias que contienen los estados finales localizados en los límites de x , lo cual ayuda al agente a aprender que debe evitar empujar el carro a dichos límites.
- Generar acciones aleatorias tomadas de una distribución normal. Esto ocasiona que durante el número de episodios en los cuales se usa esta estrategia de exploración arbitraria, el carro se mueva a distintas velocidades en cercanía del estado inicial, generando experiencias que contienen pequeñas oscilaciones del péndulo, lo cual ayuda al agente a aprender que debe alejar el péndulo de la posición hacia abajo y que también debe mantener el carro en cercanía del centro de la pista.
- Añadir ruido a las acciones generadas por la política según la estrategia de exploración propuesta por los autores de TD3. Como una modificación a esta estrategia de exploración, el factor de escala del ruido e_{ruido} se reduce a medida que se incrementa el número de experiencias llevadas a cabo. Esto ocasiona que, en episodios tempranos, el ruido de exploración añadido sea grande, favoreciendo la exploración al inicio, mientras que a medida que se incrementa el número de experiencias se favorece la explotación de la política.

La clase `Exploracion` debe ser instanciada con un único parámetro, el cual fue denominado `esc_ruido`. Dicho parámetro, junto a una lista de Python que es inicializada al instanciar la clase, se muestran en la tabla 24.

Variables y objetos inicializados al instanciar “Exploracion”	
Nombre	Descripción
<code>esc_ruido</code>	Numero de punto flotante entre 0 y 1, especifica la escala del ruido de exploración e_{ruido} inicial.
<code>ACC_INICIO</code>	Lista de Python que contiene las acciones iniciales para los primeros 21 episodios [-1.0, -0.9, -0.8, ... 0.8, 0.9, 1.0]

Tabla 24 Variables y objetos inicializados al instanciar la clase `Exploracion`. Fuente: Elaboración propia.

Para lograr el funcionamiento descrito anteriormente, la clase `Exploracion` fue implementada para proveer 5 métodos, los cuales son descritos en la tabla 25.

Métodos de ”Exploracion”			
Nombre del método	Argumentos(s) de entrada	Funcionamiento	
<code>gen_prim_accion</code>	<code>num_ep</code>	Devuelve la acción inicial correspondiente al numero de episodio $ACC_INICIO[num_ep]$	
<code>gen_acc_ruido</code>	Ninguno	Toma una acción aleatoria de una distribución normal con escala 0.5	
<code>anadir_ruido_accion</code>	<code>accion</code>	Añade ruido tomado de una distribucion normal escalado por e_{ruido} a <code>accion</code> , luego reduce e_{ruido}	
<code>guardar</code>	Ninguno	Guardar la escala de ruido e_{ruido} a disco duro	
<code>cargar</code>	Ninguno	Cargar la escala de ruido e_{ruido} desde disco duro	

Tabla 25 Métodos de la clase `Exploracion`. Fuente: Elaboración propia.

El método `anadir_ruido_accion` descrito en la tabla 25, fue implementado de manera que la variable e_{ruido} sea decrementada exponencialmente con cada experiencia llevada a cabo. Su funcionamiento se detalla en el algoritmo 14.

Algoritmo 14 `anadir_ruido_accion`

Entradas: `accion`

- 1: $ruido \leftarrow \epsilon \sim N$
 - 2: $accion_ruido \leftarrow accion + ruido$
 - 3: $accion_ruido \leftarrow clip(accion_ruido, -1,0, 1,0)$
 - 4: $esc_ruido \leftarrow esc_ruido \times 0,9999985$
 - 5: **retornar** `accion_ruido`
-

3.6.5. Implementación del Módulo de Métrica

El módulo de métrica fue implementado como una clase denominada `Metrica`, la cual está destinada a calcular y almacenar datos de la recompensa promedio obtenida en cada uno de los episodios de entrenamiento. Para lograr dicho propósito, al ser instanciada, inicializa una lista de Python cuyo propósito se describe en la tabla 26.

Lista inicializada al instanciar “Metrica”	
Nombre	Descripción
lista_rec	Lista de Python; usada para almacenar las recompensas promedio de los episodios

Tabla 26 Lista inicializada al instanciar la clase Metrica. Fuente: Elaboración propia.

Para llevar a cabo sus funciones, el módulo de Métrica fue implementado para proveer los métodos descritos en la tabla 27.

Métodos de ”Metrica”		
Nombre del método	Entrada(s)	Funcionamiento
prom_rec	rec_ep; Recompensa total acumulada a lo largo del episodio pasos_ep: Numero total de experiencias llevadas a cabo a lo largo el episodio	Añade a lista_rec el promedio de recompensa del episodio
guardar	Ninguna	Guardar lista_rec a disco duro
cargar	Ninguna	Cargar lista_rec desde disco duro

Tabla 27 Métodos de la clase Metrica. Fuente: Elaboración propia.

3.6.6. Implementación del Módulo Principal

El módulo principal fue implementado como un script de Python, el cual, al ser ejecutado, lleva a cabo la interacción episódica y el entrenamiento del agente, aprovechando los módulos y clases descritos anteriormente. En esencia, por la manera en la que el módulo principal fue implementado, crea un lazo de control realimentado donde el controlador se encuentra compuesto por el agente.

Para lograr lo descrito anteriormente de una manera que permite controlar y modificar ciertos aspectos del proceso de entrenamiento y del lazo de control, el módulo principal posee un número de parámetros y variables, los cuales se describen en la tabla 28.

Además de permitir modificar los parámetros descritos en la tabla 28, el módulo principal crea instancias de Agente, Entorno, Exploracion y Metrica con los parámetros de instanciación requeridos para cada uno, los cuales se pueden modificar también con el fin de cambiar características del proceso de entrenamiento y experimentar.

Añadido a lo anterior, el módulo principal es capaz de tomar un parámetro literal como argumento de entrada, el cual indica si se va a continuar el entrenamiento o se empieza un nuevo proceso de entrenamiento. El parámetro de entrada puede tomar dos valores: cargar indica al módulo principal que debe cargar los parámetros de Agente, Entorno, Exploracion y Metrica desde el disco duro para continuar el proceso de entrenamiento, mientras que nocargar indica que se debe iniciar un nuevo proceso de entrenamiento sin carga de parámetros.

Para ejecutar el módulo principal, basta con llamarlo con el argumento de entrada desde una línea de comandos en un sistema GNU/Linux, como se muestra en la línea de código a continuación.

```
python Principal.py cargar
```

Parámetros del modulo principal		
Nombre	Tipo	Descripción
EPS_INICIALES	Número entero	Número de episodios iniciales durante los cuales generar acciones usando <i>gen_prim_acc</i>
EPS_RUIDO	Número entero	Número de episodios iniciales durante los cuales generar acciones de ruido usando <i>gen_acc_ruido</i>
MAX_EPS	Número entero	Número máximo de episodios a llevarse a cabo durante el proceso de entrenamiento
MAX_PASOS_EP	Número entero	Número máximo de experiencias a llevarse a cabo durante cada episodio
MAX_CAMBIO_ACCION	Número de punto flotante	Límite de cambio máximo entre una acción anterior a_{t-1} y una acción actual a_t
ep_actual	Número entero	Variable para almacenar el numero de episodio de entrenamiento llevándose a cabo

Tabla 28 Parámetros del módulo principal. Fuente: Elaboración propia.

El funcionamiento del módulo principal se detalla en el algoritmo 15.

Algoritmo 15 Módulo principal

```

1: Instanciar objetos agente, entorno, exploracion, metrica
2: for ep_actual < max_eps do
3:   rec_ep, num_paso, accion_ant ← 0
4:   Ejecutar entorno.comenzar()
5:   while fin_ep ≠ 1 do
6:     Recibir est_ant, rec, fin_ep, num_paso ejecutando entorno.est_ent()
      {Elegir la estrategia de exploración de acuerdo al numero de episodio}
7:     si ep_actual ≤ EPS_INICIALES entonces
8:       accion ← exploracion.gen_prim_acc(ep_actual)
9:     else si ep_actual ≥ EPS_INICIALES y ep_actual ≤ EPS_RUIDO entonces
10:      accion ← exploracion.gen_acc_ruido()
11:    else
12:      accion ← agente.generar_accion(est_ant)
13:      accion ← exploracion.anadir_ruido_accion(accion)
14:    fin de si
15:    si |accion – accion_ant| > MAX_CAMBIO_ACCION entonces
16:      Limitar cambio de acción a MAX_CAMBIO_ACCION
17:    fin de si
18:    Convertir accion a comando numerico cmd ← accion * 10000
19:    entorno.enviar(cmd)
20:    Recibir est_act, rec, fin_ep, num_paso ejecutando entorno.est_ent()

```

```

21:   Guardar experiencia agente.almacenar_paso(est_ant, accion, rec, est_act, fin_ep)
22:   rec_ep  $\leftarrow$  rec_ep + rec, est_ant  $\leftarrow$  est_act, accion_ant  $\leftarrow$  accion
23: fin de while
24: Reiniciar el entorno entorno.reset_ent()
25: Ejecutar función de la métrica,
    prom_rec  $\leftarrow$  metrica.prom_rec(rec_ep, num_paso)
26: si ep_actual  $\geq$  EPS_RUIDO entonces
27:     Entrenar el agente ejecutando agente.entrenar()
28: fin de si
29: Guardar estado del modulo de metrica y el modulo de exploracion
30: ep_actual  $\leftarrow$  ep_actual + 1
31: fin de for

```

3.6.7. Implementación del Módulo de Demostración

El módulo de demostración fue implementado como un lazo de control que no toma parte en el proceso de entrenamiento del agente. Para esto, permite al agente interactuar con el entorno sin las restricciones de tiempo impuestas por la interacción episódica.

Para lograr la funcionalidad descrita, el módulo de demostración crea instancias de *Agente*, *Entorno*, *Exploracion* y *Metrica*, las cuales poseen los mismos parámetros de instancia usados para llevar a cabo el proceso de entrenamiento.

Los objetos instanciados son usados dentro un lazo de control que no lleva a cabo entrenamiento del agente, pero que en cambio almacena y guarda datos de las variables de estado y acciones elegidas por el agente.

El módulo de demostración usa como argumento de entrada un parámetro literal, el cual indica si se desea llevar a cabo una interacción de longitud igual a un episodio o si se desea llevar una interacción de longitud mayor. Dicho parámetro puede tomar dos valores: *correr_corto* para una interacción episódica y *correr_largo* para una interacción arbitrariamente larga de 1000000 pasos de tiempo.

Para ejecutar el módulo principal, basta con llamarlo con el argumento de entrada desde una línea de comandos en un sistema GNU/Linux, como se muestra en la línea de código a continuación.

```
python Demostracion.py correr_corto
```

El algoritmo 16 detalla el funcionamiento del módulo de Demostración.

Algoritmo 16 Modulo de demostracion

```

1: Instanciar objetos agente, entorno, exploracion, metrica
2: rec_ep, num_paso, accion_ant  $\leftarrow 0$ 
3: Ejecutar entorno.comenzar()
4: while fin_ep  $\neq 1$  do
5:   Recibir est_ant, rec, fin_ep, num_paso ejecutando entorno.est_ent()
6:   accion  $\leftarrow$  agente.generar_accion(est_ant)
7:   accion  $\leftarrow$  exploracion.anadir_ruido_accion(accion)
8:   si  $|accion - accion\_ant| > MAX\_CAMBIO\_ACCION$  entonces
9:     Limitar cambio de acción a MAX_CAMBIO_ACCION
10:    fin de si
11:   Convertir accion a comando numerico cmd  $\leftarrow accion * 10000$ 
12:   entorno.enviar(cmd)
13:   Almacenar est_ant, accion, rec
14:   Recibir est_act, rec, fin_ep, num_paso ejecutando entorno.est_ent()
15:   Almacenar est_ant, accion, rec
16:   rec_ep  $\leftarrow rec\_ep + rec, est_ant  $\leftarrow est\_act, accion_ant  $\leftarrow accion
17: fin de while
18: Reiniciar el entorno entorno.reset_ent()
19: Graficar x, ẋ, θ, θ̇, accion, rec a lo largo de la trayectoria$$$ 
```

3.7. Pruebas del sistema

3.7.1. Entrenamiento del Agente

Para llevar a cabo el proceso de entrenamiento, primeramente, se eligieron los valores para los parámetros c_1 , c_2 y c_3 de la función de recompensa. Se estableció mediante dichos parámetros una jerarquía de importancia de los estados θ , x y $\dot{\theta}$, de manera que las penalizaciones asignadas por los términos que incluyen cada uno de dichos estados, señalen la importancia de cada estado con respecto a los otros dos.

La tabla 29 muestra los valores elegidos para los parámetros c_1 , c_2 y c_3 , describiendo la manera en que los valores asignan importancia a cada uno de los estados asociados.

Parámetros de la función de recompensa usados para el entrenamiento		
Nombre	Valor	Descripción
c_1	$100 \frac{1}{2\pi^2}$	Normaliza la penalización asignada por el ángulo del péndulo θ con respecto a la vertical a un valor máximo de 100, haciéndola la más grande entre las tres y por lo tanto la más importante.
c_2	$0.25 \frac{100}{0.2^2}$	Normaliza la penalización asignada por alejarse del centro de la pista a un valor máximo de 25, haciéndola la segunda más grande y por lo tanto segunda en importancia.
c_3	$0.005 \frac{100 \times 0.01}{\pi}$	Normaliza la penalización asignada por incrementar la velocidad angular del péndulo arriba de $2\pi[\frac{rad}{s}]$ a 0.5, haciéndola la más pequeña y por lo tanto la menos importante.

Tabla 29 Parámetros de la función de recompensa usados para el entrenamiento. Fuente: Elaboración propia.

La ilustración 42, muestra el comportamiento de la función de recompensa usando los parámetros de la tabla 29, graficando los valores obtenidos de ella dentro los límites impuestos para los tres estados θ , x y $\dot{\theta}$ en la sección 3.6.1.1.

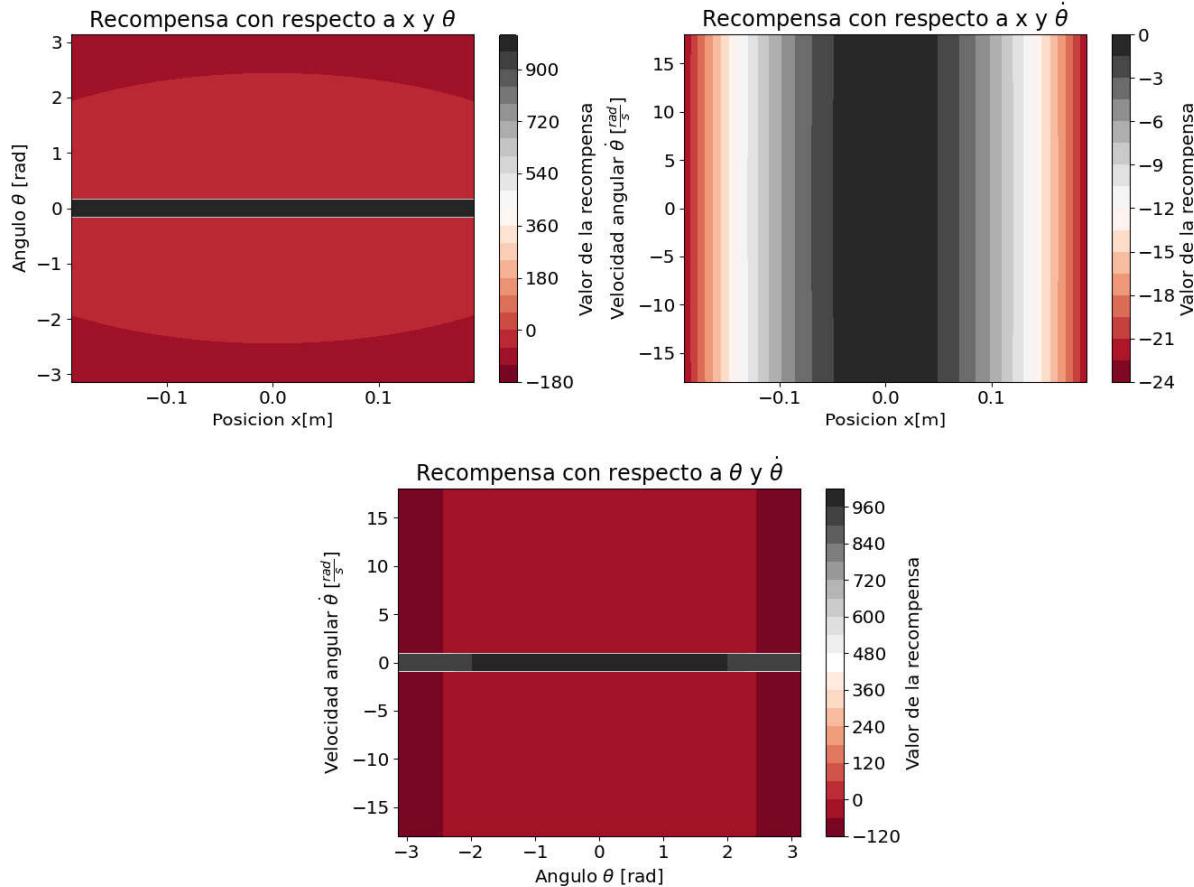


Ilustración 42 Comportamiento de la función de recompensa para los parámetros elegidos. Fuente: Elaboración propia.

En la ilustración 42, se muestra la manera en la cual la función de recompensa penaliza de manera más fuerte, estados que se encuentran más alejados del objetivo de mantener al péndulo en la posición vertical hacia arriba, mientras que la recompensa se hace substancialmente más alta cuando se alcanza dicho objetivo.

De dicha ilustración es posible también observar como la función de recompensa penaliza de manera proporcional a la medida en la que el carro es alejado del centro de la pista, aunque de una manera más leve comparada con la penalización asignada por el valor del ángulo del péndulo.

Seguidamente, se llevó a cabo una elección de los tamaños de capa para las instancias de `RedActor` y `RedCritico`. Tomando en cuenta que la instancia de `RedActor` es usada como red política, y por lo tanto debe ser ejecutada en una cantidad de tiempo restringida por la tasa de muestreo, se eligió un tamaño más pequeño para su primera capa con respecto a las instancias de

RedCriticó, las cuales pueden ser más grandes y no poseen restricciones de tiempo de ejecución, ya que se ejecutan una vez que se haya acabado un episodio. La tabla 30 resume las arquitecturas con los parámetros elegidos para las instancias de ambas redes.

Arquitectura de “RedActor” usada para el entrenamiento		
Nombre capa	Número de neuronas	Función de activación
Capa de entrada	$tam_estado = 4$	Ninguna
Capa Escondida 1	$n^{[1]} = 256$	Unidad lineal rectificada (ReLU)
Capa Escondida 2	$n^{[2]} = 256$	Unidad lineal rectificada (ReLU)
Capa de salida	1	Tangente hiperbólica (Tanh)
Arquitectura de “RedCriticó” usada para el entrenamiento		
Nombre capa	Número de neuronas	Función de activación
Capa de entrada	$tam_estado + 1$	Ninguna
Capa Escondida 1	$n^{[1]} = 512$	Unidad lineal rectificada (ReLU)
Capa Escondida 2	$n^{[2]} = 256$	Unidad lineal rectificada (ReLU)
Capa de salida	1	Lineal

Tabla 30 Parámetros para las arquitecturas de RedActor y RedCriticó. Fuente: Elaboración propia.

Para instanciar la clase Agente, tanto en el módulo principal como en el módulo de demostración, se usaron los parámetros resumidos en la tabla 31.

Parámetros de Instanciación de “Agente” usados para el entrenamiento	
Nombre	Valor
rho	0.005
gamma	0.99
interval_act_actor	2
entrenamientos_ep	3000
tam_memoria	1×10^6
tam_lote	256

Tabla 31 Fuente: Parámetros de instanciación del agente usados para el entrenamiento. Fuente: Elaboración propia.

Para instanciar el módulo de exploración, se usó el único parámetro requerido; la escala de ruido inicial, por lo tanto, se usó el valor `esc_ruido = 0.2`.

Finalmente, el proceso de entrenamiento fue llevado a cabo ejecutando el módulo principal, el cual tuvo una duración total de 1400 episodios. Los parámetros utilizados en el módulo principal para el proceso de entrenamiento son aquellos descritos en la tabla 32.

Parámetros del modulo principal usados para el entrenamiento		
Nombre	Valor	Descripción
EPS_INICIALES	21	Su valor es igual al número de acciones presentes en la lista <i>ACC_INICIO</i> del módulo de exploración
EPS_RUIDO	10	Se limitó el número de episodios que usan acciones aleatorias de una distribución normal usando <i>gen_accion_ruido</i> a 10.
MAX_EPS	1400	Se limitó el máximo número de episodios usados para el entrenamiento del agente a 1400
MAX_PASOS_EP	4500	Se limitó el máximo número de pasos de tiempo por episodio a 4500, los cuales a la tasa de muestreo de $10[ms]$ son iguales a $45[s]$ de interacción entre el agente y el entorno
MAX_CAMBIO_ACCION	0.5	Se limitó el cambio máximo de las acciones generadas por el agente a 0.5 con el propósito de evitar que el carro se sacuda demasiado y que el motor salte pasos.

Tabla 32 Parámetros del módulo principal usados para el entrenamiento Fuente: Elaboración propia.

3.7.2. Pruebas de rendimiento

Una vez llevado a cabo el entrenamiento del agente, para cuantificar su rendimiento después de haber sido entrenado y comprobar que cumplió con el objetivo, se diseño y llevo a cabo una serie de pruebas bajo diferentes condiciones, mediante las cuales se buscó observar y analizar el comportamiento del agente como controlador del péndulo invertido. Las pruebas de rendimiento fueron diseñadas para observar el rendimiento del agente en tres categorías, las cuales se describen a continuación.

- **Análisis de desempeño:** En esta categoría, se realizaron 15 pruebas usando el agente entrenado, obteniendo datos de 15 trayectorias generadas mediante el módulo de demostración. Se uso los datos recopilados a lo largo de las 15 pruebas analizar el rendimiento del agente como controlador, tanto en el proceso de columpiado hacia arriba (swing up), como en la estabilización en la posición vertical.
- **Análisis de perturbaciones de carga:** Dentro esta categoría, se modificó la planta con una barra de péndulo diferente, esencialmente modificando el entorno. Se realizaron 15 pruebas sobre el entorno modificado, generando datos de 15 trayectorias adicionales. Los datos recopilados dentro esta categoría fueron usados para observar el comportamiento del agente, tanto durante el proceso de columpiado hacia arriba como en la estabilización en la posición vertical, permitiendo analizar la manera en la que perturbaciones de carga y cambios de parámetros del entorno afectan la capacidad del agente entrenado para cumplir con el objetivo.
- **Análisis de perturbaciones externas:** Dentro esta tercera y última categoría, se sometió al agente a perturbaciones externas, las cuales consistieron en modificar el ángulo del péndulo de manera manual, aplicando fuerza sobre él. De los datos recopilados dentro esta categoría, se pudo analizar el comportamiento del agente ante experiencias inesperadas y que no fueron encontradas a lo largo del proceso de entrenamiento, probando su eficacia como controlador para rechazar perturbaciones externas.

Capítulo 4

Análisis y discusión de los resultados

4.1. Experimentación y resultados

Para caracterizar el progreso del agente a lo largo del proceso de entrenamiento bajo los parámetros especificados en la sección 3.7.1, en la presente sección se realiza un análisis de los datos de entrenamiento recopilados mediante el módulo de métrica; estos proveen una vista amplia a de la evolución del rendimiento en dicho proceso.

La ilustración 43 es una gráfica la evolución de la recompensa a lo largo del proceso de entrenamiento, en la cual se muestra la recompensa promedio por episodio, se observa una subida gradual de su valor a lo largo de los 1400 episodios de entrenamiento.

Se denota en la ilustración 43, que la recompensa mínima es obtenida en el episodio 21, con un valor igual a -166.783. Obtener la recompensa mínima durante los primeros episodios fue esperado, ya que durante dichos episodios se usó el módulo de exploración para generar acciones mediante estrategias de exploración arbitrarias, las cuales fueron diseñadas para generar experiencias acerca del comportamiento de la recompensa en los límites de la pista. Al alcanzar dichos límites, se genera una recompensa fuertemente negativa, según se especifica mediante la función de recompensa diseñada en la sección 3.6.1.3.

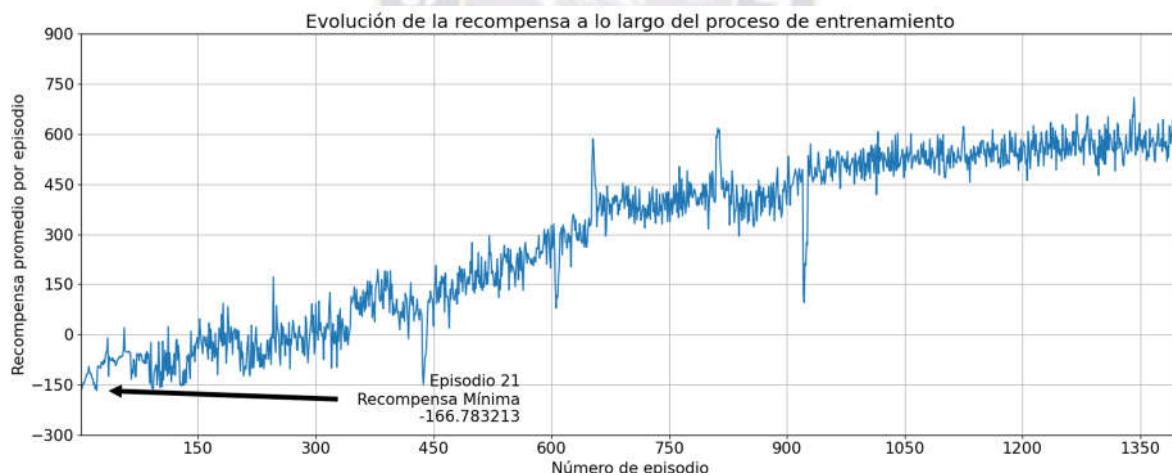


Ilustración 43 Evolución de la recompensa a lo largo del proceso de entrenamiento. Fuente: Elaboración propia.

Es posible observar también en la ilustración 43, que a lo largo del proceso de entrenamiento se generaron caídas de la recompensa promedio por episodio. Este comportamiento es atribuible al ruido de exploración añadido a las acciones generadas por la política, el cual puede ocasionar que se alcancen estados del entorno que aún no fueron explorados. Cuando esto sucede, es posible que la política no posea conocimiento suficiente para elegir una acción apropiada que lleve a recompensas más altas. Dado que, por diseño, el ruido de exploración fue reducido a medida que se llevó a cabo un número mayor de experiencias según lo especificado en la sección 3.6.4, dichas caídas de rendimiento también se redujeron con el incremento del número de episodios completados.

La ilustración 44, es una gráfica de la evolución de recompensa promedio por episodio a lo largo de los primeros 31 episodios, esto permite observar en detalle el comportamiento de dicho valor durante los episodios que se llevaron a cabo usando las dos estrategias de exploración descritas en la sección 3.6.4.



Ilustración 44 Evolución de la recompensa durante los primeros 32 episodios. Fuente: Elaboración propia.

Primeramente, de la ilustración 44 se observa la recompensa promedio por episodio a lo largo de los primeros 21 episodios, los cuales contienen experiencias que fueron llevadas a cabo usando la primera estrategia de exploración arbitraria; aquella diseñada para generar acciones que mueven el carro a los dos extremos de la pista, variando las velocidades en cada episodio. De dicha ilustración es posible observar que la estrategia de exploración fue exitosa en generar experiencias que ejemplifican un hecho importante; llegar a los límites de la pista ocasiona recibir recompensas substancialmente bajas.

Los siguientes 10 episodios mostrados en la ilustración 44, ejemplifican el comportamiento de la recompensa promedio por episodio usando la segunda estrategia de exploración arbitraria; aquella diseñada para mantener el carro en proximidad del centro de la pista usando acciones aleatorias tomadas de una distribución normal. Se puede observar que usar esa estrategia ocasionó que el agente reciba recompensas más altas en relación a los primeros 21 episodios. De esta manera, se generó de manera exitosa datos que representan un ejemplo contrario a aquel mostrado usando la primera estrategia de exploración, reforzando el hecho que el agente debe evitar alejarse del centro de la pista.

Adicionalmente, la segunda estrategia de exploración arbitraria ocasionó que el péndulo se balancee alrededor de la posición vertical hacia abajo, lo cual generó experiencias con recompensas levemente más altas cuando el péndulo se mueve hacia arriba. Esto pudo mostrar al agente que, para obtener recompensas más altas, debe alejar al péndulo de dicha posición.

Con el fin de demostrar la importancia de los primeros 31 episodios y el impacto de haber utilizado las dos estrategias de exploración arbitrarias al inicio del proceso de entrenamiento, la ilustración 45 muestra la evolución de recompensa promedio a lo largo de los primeros 75 episodios.

De la ilustración 45, se observa que a partir del episodio 32, en el cual se empieza a entrenar y usar la política para generar acciones, existe una tendencia del agente a mantener el valor de la recompensa promedio por episodio alrededor -75, el cual se encuentra significativamente alejado del valor mínimo de -166.783.

En adición, se nota una leve tendencia de subida de la recompensa promedio por episodio a partir del episodio 32, esto se atribuye a que el agente empezó a ser entrenado usando el conjunto de experiencias almacenadas a lo largo de los primeros 31 episodios, los cuales ejemplifican un conjunto de estados que no deben ser alcanzados; esto evitó que el agente deba explorar dichos estados en episodios futuros.

También, gracias a las experiencias de balanceo leve del péndulo generadas usando la segunda estrategia de exploración, se observa que el agente realizó esfuerzos de llevar al péndulo lejos de la posición hacia abajo, incluso logrando por primera vez en el episodio 56, un valor positivo de la recompensa promedio.



Ilustración 45 Evolución de la recompensa durante los primeros 75 episodios. Fuente: Elaboración propia.

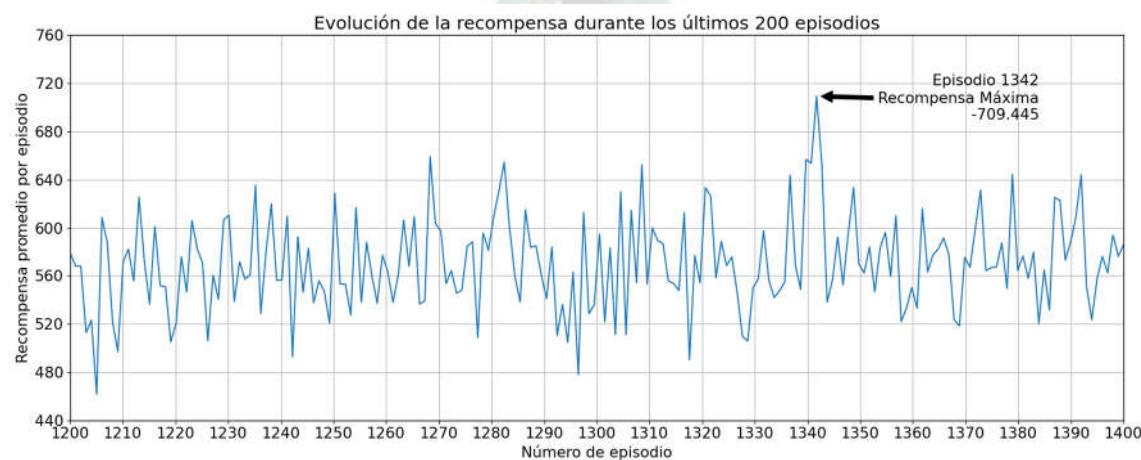


Ilustración 46 Evolución de la recompensa durante los últimos 200 episodios. Fuente: Elaboración propia.

Finalmente, en la ilustración 46 se muestra la recompensa promedio por episodio durante los últimos 200 episodios; se observa que su valor es mantenido alrededor de 560. Sin embargo, también se observa que se alcanza la recompensa promedio máxima de 709.445 en el episodio 1342, por lo cual se evidencia que el agente puede seguir mejorando su rendimiento, si se decidiera realizar más episodios de entrenamiento.

4.2. Análisis de desempeño

Además de un análisis de la evolución de la recompensa promedio obtenida a lo largo del proceso de entrenamiento, para cuantificar el desempeño del agente como controlador de una manera más detallada, se llevó a cabo un análisis de su comportamiento una vez que el proceso de entrenamiento fue completado. Para este propósito, se realizó 15 pruebas usando el módulo de demostración, de las cuales se obtuvieron datos de 15 trayectorias de 45 segundos.

4.2.1. Columpiado Hacia Arriba (Swing Up)

La tarea del columpiado hacia arriba (denominada swing up en inglés), consiste en llevar al péndulo desde la posición inicial apuntando hacia abajo, para la cual el ángulo del péndulo $\theta = \pi[\text{rad}]$, a la posición vertical hacia arriba, la cual en las pruebas realizadas, fue considera en un ángulo $\theta = \pm 0.2[\text{rad}]$.

La tabla 33 resume el tiempo empleado en lograr que el péndulo alcance la posición vertical hacia arriba por primera vez a lo largo de las 15 pruebas realizadas. En adición, se muestra la posición del carro sobre la pista en el instante que dicha posición fue alcanzada.

Tiempo mínimo de columpiado hacia arriba		
Número de prueba	Tiempo de columpiado [s]	Posición x [m]
1	16.05	0.013786
2	14.52	0.063928
3	14.71	0.094546
4	14.7	-0.10577
5	20.35	-0.088713
6	15.95	0.0039841
7	14.72	0.054838
8	19.48	-0.10254
9	12.66	-0.097293
10	18.54	-0.11863
11	13.42	0.013984
12	16.57	-0.052683
13	14.58	-0.094569
14	18.27	-0.077179
15	15.59	-0.11732
Promedio	16.01	-0.040642
Desviación estándar	2.16	0.071278

Tabla 33 Tiempo mínimo de balanceo hacia arriba. Fuente: Elaboración propia.

De la tabla 33 se observa que, en promedio, la tarea de balanceo hacia arriba es llevada a cabo en 16.01 segundos con una desviación estándar de 2.16. La variabilidad relativamente alta, es atribuida principalmente al ruido de exploración añadido.

Además del efecto del ruido de exploración, se debe tomar en cuenta que, al tratarse de interacción un sistema real, existen otros factores no ideales que influencian el rendimiento del agente, tales como el retardo de comunicación entre la estación de trabajo y la placa de microcontrolador usada para implementar el subsistema de medición y control, así como la elasticidad de la polea usada para mover al carro. Se remarca que a pesar de esos u otros factores no ideales que puedan influir sobre el rendimiento del controlador, el agente entrenado siempre fue capaz de realizar la tarea del columpiado hacia arriba en las pruebas realizadas.

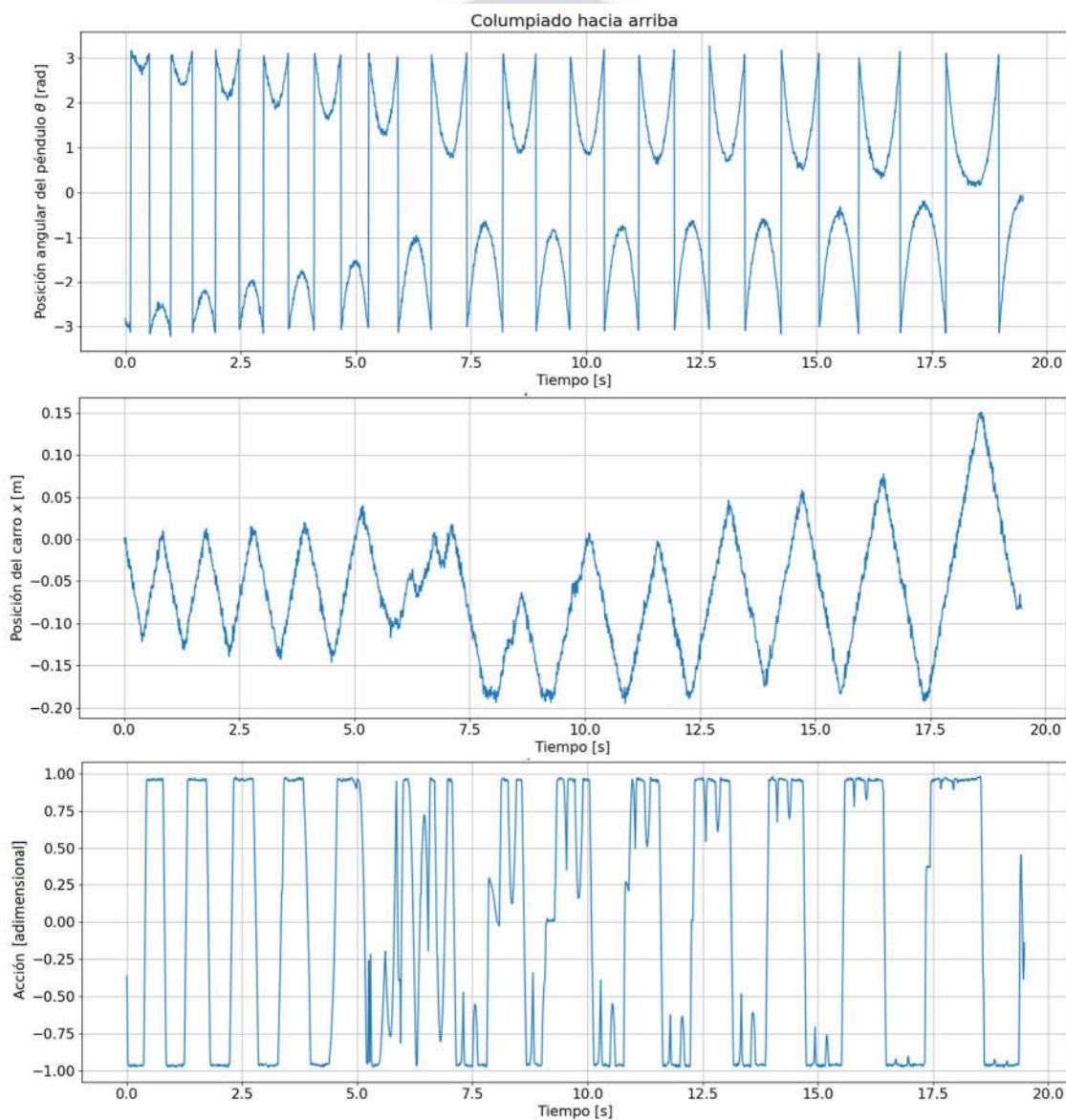


Ilustración 47 Columpiado hacia arriba de la prueba 8. Fuente: Elaboración propia.

De la tabla 33, se observa también que la desviación estándar para la posición en la cual se encuentra el carro al alcanzar la posición hacia arriba es de 0.71278 metros; un valor relativamente grande que representa una longitud de más de un cuarto de la pista hacia ambos lados. De la desviación estándar calculada, se tiene que para un rango de confiabilidad de 95%, el péndulo alcanza la posición vertical hacia arriba dentro los límites de la pista.

Del resultado anterior, se puede afirmar que el agente fue capaz de generalizar a partir de las experiencias y llevar al péndulo a la posición hacia arriba en un amplio rango de posiciones del carro, pero que también aprendió a evitar sobrepasar los límites laterales de la pista, los cuales fueron especificados en la sección 3.6.1.1.

La ilustración 47 muestra la evolución de la posición angular del péndulo θ y la posición del carro x junto a las acciones elegidas por el agente, dentro el proceso de columpiado hacia arriba para la trayectoria 8.

4.2.2. Estabilización en la Posición Vertical

La tarea de estabilización en la posición vertical consiste en mantener al péndulo en la posición hacia arriba una vez que se haya realizado el columpiado hacia arriba.

En las 15 pruebas realizadas, se observó que el agente entrenado es capaz de estabilizar al péndulo en la posición vertical hacia arriba todas las veces, manteniéndolo en cercanía de dicha posición hasta los 45 segundos de tiempo límite para cada prueba realizada.

En adición, se pudo observar que el agente aprendió simultáneamente a mantener el carro en cercanía del centro de la pista, ya que esta es la única manera de maximizar la recompensa obtenida. Según el diseño de la función de recompensa, tanto las penalidades por alejar el péndulo de la posición vertical hacia arriba y de alejar el carro del centro se acercan a cero solamente si el carro se encuentra en cercanía de $x = 0$ mientras la posición angular del péndulo se aproxima a $\theta = 0$, hecho que fue notado y aprovechado por el agente.

Dado que el agente aprendió a mantener el carro en proximidad del centro de la pista, también fue posible observar que según las importancias asignadas a θ y x mediante los parámetros de la función de recompensa, el agente fue capaz de aprender comportamientos complejos que muestran preferencias por mantener el ángulo del péndulo en la posición hacia arriba, incluso si esto implica mover el carro lejos del centro. El comportamiento de preferencia por mantener el ángulo del péndulo, es observable y evidente entre los segundos 30 y 40 de la ilustración 48, la cual muestra la evolución de las cuatro variables de estado del péndulo a lo largo de la prueba 14.

En dicha prueba, es posible notar que el carro es desplazado hacia el lado izquierdo para mantener el ángulo del péndulo cercano a cero; mientras que existen cambios notables en la posición del carro y su velocidad, tanto la posición angular como la velocidad del péndulo se mantienen en valores cercanos a cero.

Se muestra en la ilustración 48 también, que una vez que el ángulo se encuentra estabilizado, el agente toma acciones que devuelven el carro a la posición en el centro, para maximizar la recompensa recolectada.

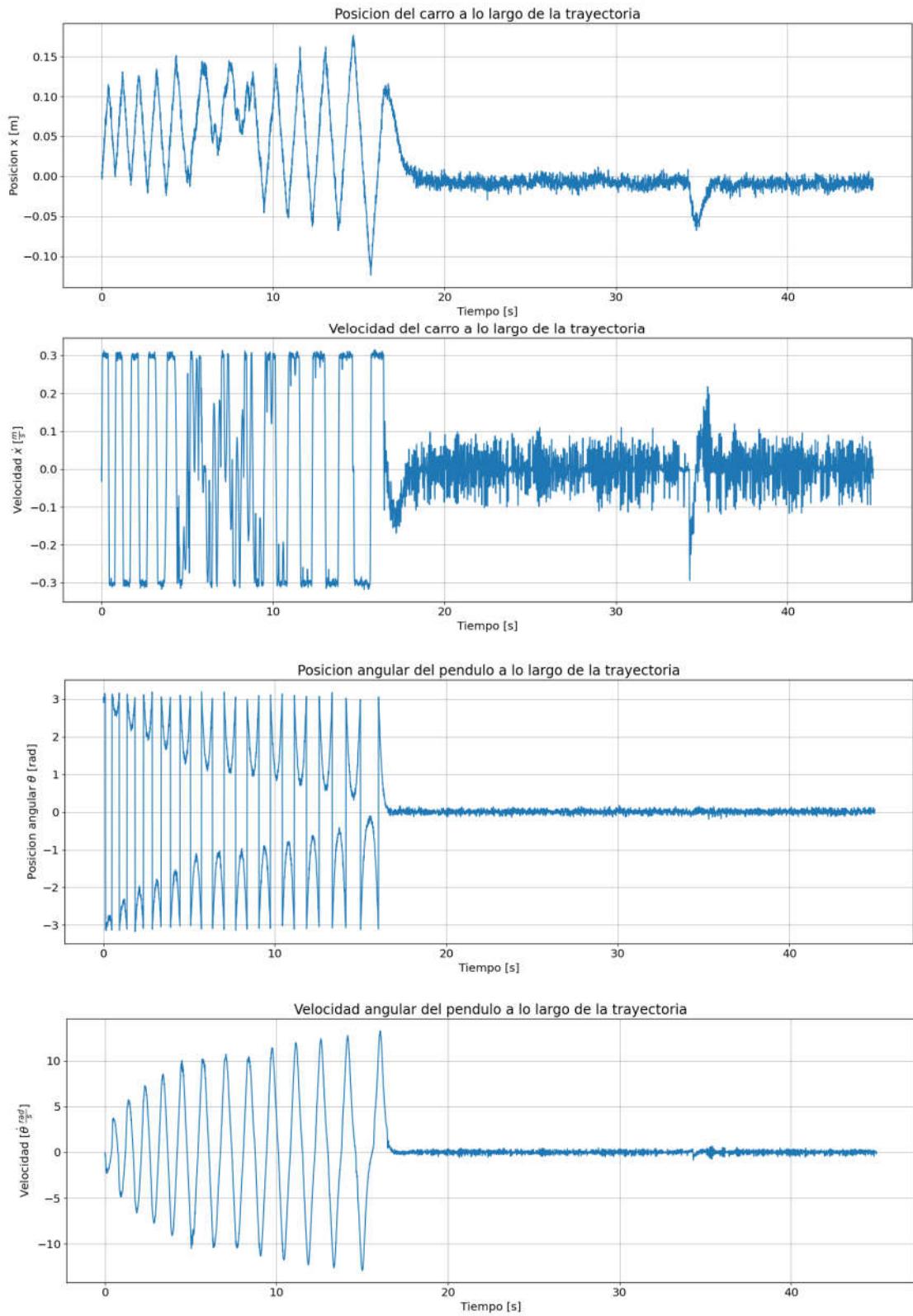


Ilustración 48 Evolución de las variables de estado a lo largo de la prueba 14. Fuente: Elaboración propia.

4.3. Análisis de Perturbaciones de Carga

En la presente sección, se analiza y describe los resultados obtenidos de someter al agente a una perturbación de carga, la cual fue introducida mediante un cambio del péndulo usado en la planta por uno con parámetros físicos diferentes.

La tabla 34 resume las características físicas de la barra metálica usada como péndulo a lo largo del proceso de entrenamiento, también describe las características de una segunda barra más liviana y corta usada para las pruebas de perturbación de carga de la presente sección.

Comparación de longitud y peso de las dos barras		
Barra	Peso [g]	Longitud [cm]
Barra usada para el entrenamiento	90.2	40.3
Barra usada para pruebas de perturbación de carga	75.9	34.2
Cambio porcentual	-18.8%	-17.8%

Tabla 34 Comparación de las dos barras usadas. Fuente: Elaboración propia.

Se eligió usar una barra más corta y menos pesada para realizar estas pruebas ya que al tener una barra más corta, el centro de masa del péndulo se encuentra a una longitud menor de su punto de giro, lo cual ocasiona que controlar el péndulo modificado tenga una mayor dificultad.

Una reducción de la longitud al centro de masa, genera que se requiera un esfuerzo de control más alto por parte de cualquier controlador para estabilizar al péndulo en la posición hacia arriba; se requieren acciones con cambios más bruscos y de mayor magnitud para poder controlar el péndulo. Además, pone a prueba la capacidad de generalización del agente entrenado, ya que permite observar su comportamiento en un entorno similar al cual fue utilizado para su entrenamiento, pero con parámetros diferentes.

La tabla 35 resume el tiempo empleado en lograr que el péndulo alcance la posición vertical hacia arriba por primera vez, a lo largo de las 15 pruebas realizadas usando la barra para pruebas de perturbaciones de carga. En adición, se muestra también la posición del carro sobre la pista para el instante que dicha posición fue alcanzada.

De los datos presentados tabla 35, se observa que el tiempo empleado para llevar a cabo el balanceo hacia arriba es en promedio 1.07 segundos más alto en comparación con las pruebas llevadas a cabo en la sección anterior; la modificación del entorno tiene un efecto adverso sobre el rendimiento del agente en la tarea del columpiado hacia arriba. Sin embargo, se subraya que el agente es capaz de llevar a cabo dicha tarea de todas formas.

En cuanto a la posición del carro en la tarea de columpiado hacia arriba, se observa un promedio de -0.011365 [m] con una deviación estándar de 0.10064[m]; de nuevo es posible observar el efecto adverso de la segunda barra sobre el rendimiento del agente, ya que los datos muestran que el agente se acerca más a los límites de la pista. De la deviación estándar calculada, es posible calcular para un rango de confiabilidad de 68%, que el péndulo alcanza la posición vertical hacia arriba dentro los límites de la pista.

Tiempo mínimo de columpiado hacia arriba - Barra para pruebas de perturbaciones de carga		
Número de prueba	Tiempo de columpiado [s]	Posición x [m]
1	17.78	0.039831
2	20.67	-0.099729
3	19.28	0.043749
4	17.61	-0.071019
5	14.24	-0.12498
6	20.92	0.13862
7	18.29	-0.11238
8	16.62	0.10515
9	16.69	0.15910
10	15.76	0.019446
11	15.75	-0.11105
12	15.77	0.082916
13	16.69	-0.12161
14	14.75	-0.13049
15	15.43	0.011965
Promedio	17.08333333	-0.011365
Desviación estándar	1.94	0.10064

Tabla 35 Tiempo mínimo de balanceo hacia arriba con barra para pruebas de perturbación. Fuente: Elaboración propia.

De los datos recopilados y su análisis, es posible observar que el agente es capaz de rechazar perturbaciones de carga, pero que la introducción de ellas genera un rendimiento menor, como es de esperarse.

En la ilustración 49 se muestra el comportamiento de las variables de estado x y θ a lo largo de la prueba 13. Se enmarca en dicha ilustración un comportamiento notorio; al terminar el balanceo hacia arriba, el agente realiza un intento de estabilizar al péndulo en la posición vertical, el cual no es exitoso en una primera instancia.

De la ilustración 49 y los datos recopilados a lo largo de dicha prueba, se describe a continuación el comportamiento y las acciones tomadas por el agente para poder realizar un segundo intento de estabilización en la posición hacia arriba.

- Primeramente, el agente intenta estabilizar al péndulo en la posición vertical haría arriba en cercanía del centro de la pista.
- Seguidamente, al no conseguirse esto, el agente mueve el carro hacia el extremo derecho de la pista, dirigiéndose hacia el límite de seguridad.
- Una vez que el carro se encuentra próximo al límite de seguridad derecho de la pista, el carro es detenido; se observa que las acciones generadas por el agente son casi nulas.
- Finalmente, el agente genera acciones de valor máximo hacia el lado izquierdo de la pista y prosigue a realizar el columpiado hacia arriba hasta un nuevo intento de estabilización en la posición vertical, el cual es exitoso.

De lo anterior, es posible observar nuevamente que un cambio en los parámetros y el comportamiento del entorno puede llevar a un rendimiento más bajo del agente como controlador.

Sin embargo, se remarca que el agente fue capaz de generar acciones correctivas y proseguir a completar el objetivo de control, lo cual indica que es tolerante a fallas.

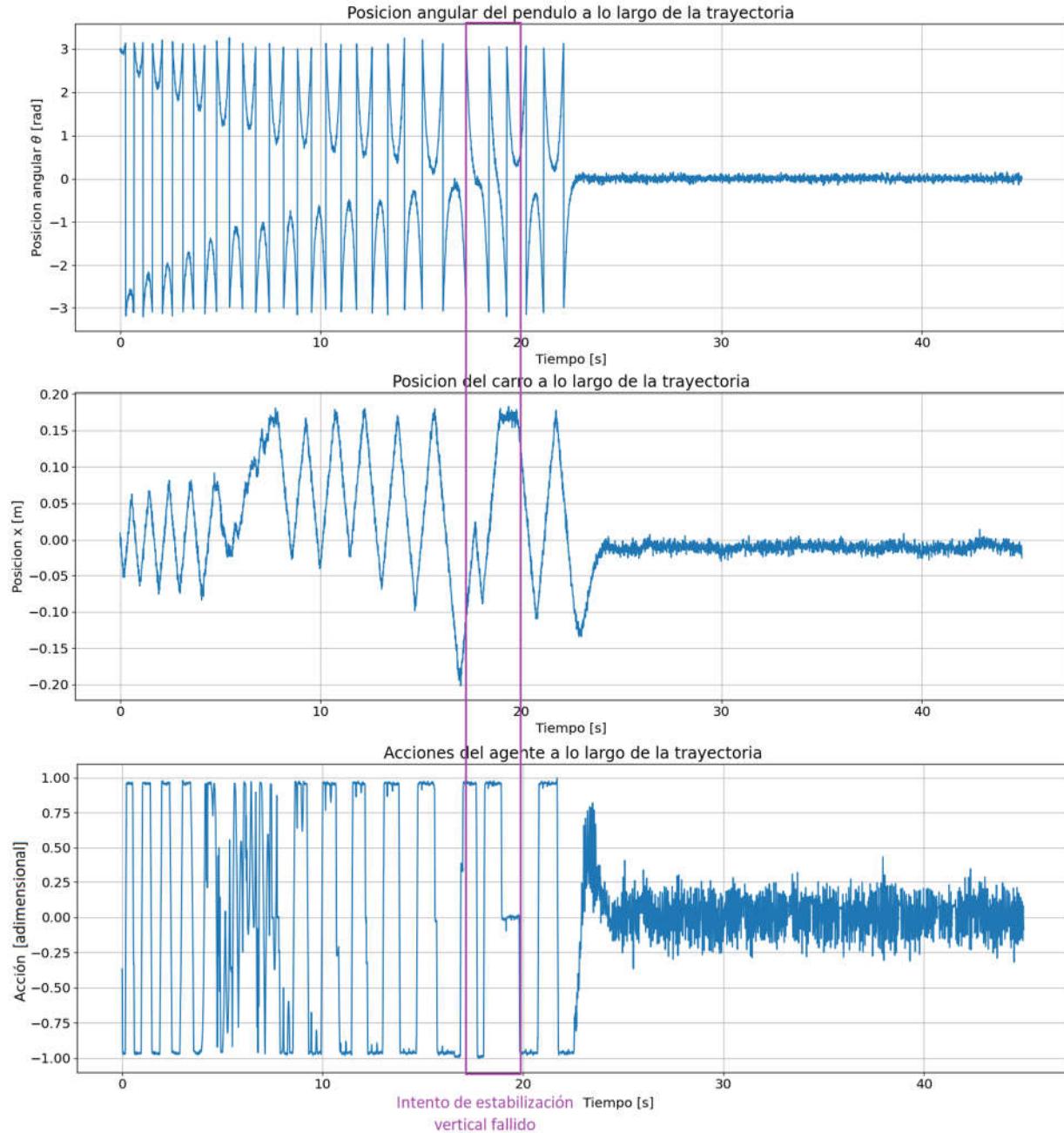


Ilustración 49 Trayectoria 13 con barra de perturbación de carga. Fuente: Elaboración propia.

4.4. Análisis de Perturbaciones Externas

En esta sección, se observa y analiza el comportamiento del agente ante perturbaciones externas que fueron aplicadas de manera manual, modificando el ángulo del péndulo. Se aplicaron 4 perturbaciones en sucesión; dos empujando el péndulo hacia la izquierda y dos empujando el

péndulo hacia la derecha. Las variables de estado x y θ junto a las acciones elegidas por la política fueron graficadas en la ilustración 50.

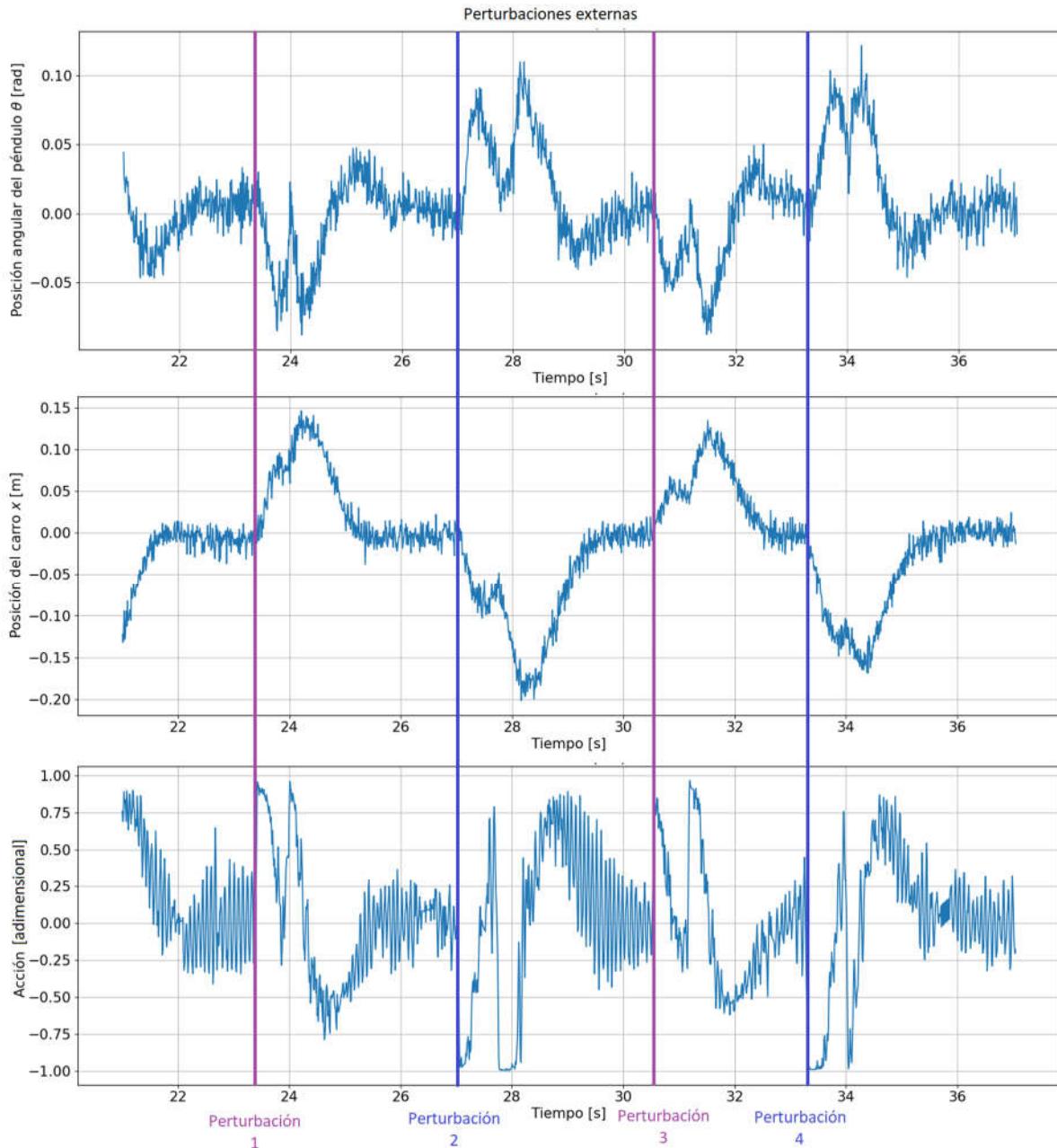


Ilustración 50 Respuesta del sistema a perturbaciones externas. Fuente: Elaboración propia.

De la ilustración 50, es posible observar que después de la aplicación de la perturbación 1, en la cual se modifica el ángulo del péndulo hacia la derecha, las acciones elegidas por el agente son cercanas al valor máximo $a = 1.0$. Dichas acciones desplazan el carro a velocidad máxima hacia la derecha, contrarrestando la perturbación introducida y manteniendo el ángulo del péndulo en

cercanía a la posición vertical ($\theta = 0$). La perturbación 3 introducida es similar a la perturbación 1 y muestra un comportamiento comparable por parte del agente en respuesta.

También es posible observar de la ilustración 50, que después de la aplicación de la perturbación 2, en la cual se modifica el ángulo del péndulo hacia la izquierda, las acciones elegidas por el agente son cercanas al valor mínimo $a = -1.0$. Las acciones elegidas desplazan el carro hacia la izquierda con velocidad máxima, contrarrestando la perturbación 2 y manteniendo el ángulo del péndulo en cercanía a la posición vertical. La perturbación 4 introducida es similar a la perturbación 2 y se observa una respuesta comparable por parte del agente.

Finalmente, se observa de la ilustración 50 que, una vez que las perturbaciones del ángulo del péndulo fueron contrarrestadas, el agente elige retornar el carro al centro de la pista, mostrando de nuevo que el agente prefiere mantener el péndulo en la posición vertical hacia arriba antes que mantener al carro en la posición central según lo especificado mediante los parámetros de la función de recompensa, pero que busca maximizar la recompensa devolviendo el carro al centro de la pista, si es posible.

De las pruebas realizadas en la presente sección y el comportamiento observado, se subraya que el agente entrenado fue capaz de generalizar de una manera suficiente para rechazar perturbaciones externas, incluso si el proceso de entrenamiento no incluyó experiencias en las cuales se hayan introducido perturbaciones.



Capítulo 5

Conclusiones y recomendaciones

5.1. Conclusiones

A continuación, se detalla conclusiones correspondientes directamente a cada uno de los objetivos específicos del proyecto, listados en la sección 1.4.2.

- ***Proponer una arquitectura básica de software para el aprendizaje reforzado profundo aplicado al control del péndulo invertido.***

Se propuso una arquitectura básica que permitió el uso de aprendizaje reforzado profundo para el control del péndulo invertido (Detallada en la sección 3.1.2, ilustración 20), la cual fue basada en la arquitectura desarrollada dentro el campo y mantiene estrecha relación con ella.

Sobre la arquitectura básica propuesta, se propusieron sub-arquitecturas para cada uno de los componentes que la conforman, las cuales fueron detalladas en las secciones 3.1.3 a 3.1.9. La implementación de cada una de ellas, como se detalló a lo largo de la sección 3.6, resultó en sistema modular, el cual posee parámetros fácilmente modificables que alientan la experimentación y permiten la implementación rápida de modificaciones.

De ello, se remarca que la arquitectura implementada puede ser adaptada para el uso de otro tipo de agentes, siempre que ellos sean diseñados para usar las señales proporcionadas por el módulo de interfaz agente-entorno.

De igual manera, se destaca que la arquitectura puede ser adaptada para uso con entornos diferentes, si se realizan modificaciones al subsistema de medición y control y al módulo de interfaz agente-entorno.

- ***Realizar una elección justificada de las herramientas hardware y software a utilizarse en el proyecto.***

En base a los requerimientos de hardware expresados a lo largo de la sección 3.2, se realizó una elección tanto para la planta de péndulo invertido adquirida, como para la placa de microcontrolador; en la sección 3.4, se llevó a cabo comparaciones entre diferentes opciones disponibles, mediante tablas elaboradas en base a los requerimientos previamente establecidos. Se señala que sobre la elección de la placa Nucleo STM32F767ZI, se pudo implementar la totalidad del programa que realiza las tareas del subsistema de medición y control en base a interrupciones, las cuales son generadas por los diferentes periféricos que el microcontrolador provee. Gracias a esto, se pudo realizar la obtención de datos y la ejecución de las acciones de una manera confiable, aprovechando elementos de hardware dedicados que reducen la latencia de envío y recepción de los datos.

De igual manera, una serie de requerimientos de las herramientas de software fueron detallados a lo largo de la sección 3.3. De dichos requerimientos, primeramente, se eligió utilizar el lenguaje de programación Python, ya que este se encuentra afianzado en la industria y la academia como una herramienta esencial dentro el campo del aprendizaje profundo a lo largo de la última década.

Gracias a la elección de usar Python, fue posible implementar cada uno de los módulos y subsistemas de la arquitectura propuesta usando la programación orientada a objetos, llevando a cabo la visión modular que se propuso al inicio del capítulo 3.

Además, gracias al amplio ecosistema de librerías de código abierto para Python, se hizo posible elegir Pytorch entre otras dos librerías con amplio soporte en la industria. Sobre dicha elección y su uso en conjunto a Numpy, la implementación de las redes neuronales y del algoritmo TD3 para el entrenamiento se pudo incorporar de una manera acorde con la arquitectura propuesta.

- ***Proponer las escalas de medición del estado del sistema, de manera que dicho estado pueda ser usado para expresar el objetivo de control del agente.***

En la sección 3.6.1, las escalas de medición del estado del sistema fueron propuesta como parte de la especificación del proceso de decisión de Markov para el péndulo invertido. La elección de dichas escalas, particularmente en cuanto a los orígenes y valores máximos de la posición angular θ del péndulo y de la posición del carro x , fueron realizadas específicamente con el propósito de facilitar el diseño de la función de recompensa, y fueron esenciales en la posibilidad de expresar penalidades por alejarse de la realización del objetivo de control de una manera sencilla e interpretable.

- ***Proponer un criterio de desempeño (denominado función de recompensa) que permita al agente evaluar y mejorar su propio rendimiento.***

En la sección 3.6.3, se propuso una función de recompensa como parte de la especificación del proceso de Markov para el péndulo invertido. La función de recompensa que fue propuesta posee 5 términos, cada uno destinado explícitamente a codificar de manera implícita un aspecto del objetivo de control que se buscó lograr mediante el entrenamiento del agente.

La función que diseñada posee 3 parámetros (c_1, c_2 y c_3), los cuales fueron destinados a asignar importancia a cada una de las variables de estado usadas para calcular la recompensa ($\dot{\theta}$, x y θ), esto permitió informar al agente cual de dichos estados debió ser priorizado para cumplir con el objetivo.

Se destaca que, dadas las escalas de medición propuestas, el diseño de la función de recompensa fue intuitivo; se agregó términos a su ecuación según se observó la necesidad de señalar al agente cuál de los estados debe tomar en cuenta para lograr el objetivo de controlar el péndulo invertido, como resultado se obtuvo una función de recompensa que expresa dicho objetivo de una manera interpretable.

De igual manera, la elección de los parámetros c_1, c_2 y c_3 usados en la función de recompensa para el entrenamiento pudo realizarse en base a un razonamiento lógico, usando valores más altos para priorizar informar al agente que el péndulo debe acercarse y mantenerse en la posición vertical, asignando una prioridad más pequeña a informar que se debe mantener al carro en la posición central y la prioridad más baja a informar que el péndulo no debe alcanzar velocidades altas.

- ***Proponer un agente basado en la literatura de aprendizaje reforzado profundo que, a partir de las mediciones del estado, genere la acción adecuada para posicionamiento del péndulo en su posición vertical.***

Se propuso un agente basado en la literatura de aprendizaje reforzado profundo, el cual usa el algoritmo TD3. Fue implementado como una clase de Python que puede ser instanciada, la cual aprovecha las capacidades de Pytorch para implementar el algoritmo de entrenamiento y entrenar una política capaz de realizar la tarea de control del péndulo.

El agente propuesto fue capaz de generar experiencias propias, almacenarlas y usarlas a lo largo del proceso de entrenamiento para mejorar su rendimiento, aprendiendo a seleccionar acciones adecuadas y generalizando la manera en la cual deben ser elegidas para lograr el objetivo de controlar al péndulo invertido sobre un carro.

El agente propuesto uso 6 redes neuronales, como se especifica en el algoritmo de TD3 mostrado en la sección 2.2.14, razón por la cual se debió dedicar una estación de trabajo con aceleración GPU. Por lo tanto, se denota que el agente implementado posee requerimientos de hardware relativamente altos para su ejecución y entrenamiento.

- ***Mostrar y analizar los resultados del sistema de control propuesto.***

A lo largo de la sección 4.1, se mostró y analizó los resultados del proceso de entrenamiento, el cual duro 1400 episodios de interacción entre el agente y el entorno sobre la arquitectura propuesta.

Se pudo observar que, a lo largo de dicho proceso, existió una tendencia del agente a mejorar su rendimiento, hasta lograr que el péndulo sea estabilizado en la posición vertical hacia arriba. Se remarca del análisis del proceso de entrenamiento, que el agente fue beneficiado de manera significativa de experiencias generadas usando estrategias de exploración arbitrarias diseñadas en el presente proyecto.

En la sección 4.2 se mostró y analizó el rendimiento del agente como resultado directo del proceso de entrenamiento. Se pudo observar a lo largo de 15 pruebas que el agente entrenado fue capaz de cumplir con el objetivo de controlar el péndulo invertido, realizando consistentemente las tareas de llevar al péndulo a la posición vertical hacia arriba y estabilizarlo en dicha posición.

Continuando, en la sección 4.3, se pudo observar que el agente entrenado fue capaz de generalizar a partir de sus experiencias, de manera que logró cumplir el objetivo bajo un entorno modificado con parámetros y comportamiento diferentes, los cuales fueron impuestos mediante un cambio de barra del péndulo, aunque se pudo observar un rendimiento levemente más bajo.

Finalmente, en la sección 4.4, se mostró que el agente también fue capaz de generalizar a lo largo del proceso de entrenamiento de manera que, incluso el agente no tomo parte en experiencias que incluyen la introducción de perturbaciones externas, la política pudo corregir y rechazar las perturbaciones externas introducidas manualmente.

A partir de lo desarrollado en el presente proyecto de grado y a partir de los resultados mostrados, se concluye que es posible utilizar sistemas de aprendizaje reforzado profundo como controladores, lo cual permite proponerlos como una alternativa al flujo de diseño de controladores descrito en la sección 1.1.1.2.

La conclusión anterior se afirma dado que, considerando las propiedades del péndulo invertido detalladas dentro la sección 2.3, las cuales lo caracterizan como un sistema complejo y desafiante para los controladores, el sistema de aprendizaje reforzado profundo propuesto e implementado fue capaz de completar con éxito el objetivo de llevar el péndulo a la posición vertical hacia arriba y estabilizarlo en la misma, como se mostró a lo largo del capítulo 4. En base a los resultados obtenidos, es posible afirmar la capacidad de usar sistemas de aprendizaje reforzado profundo en

lugar de controladores convencionales; por lo tanto, se puede obviar la necesidad del modelado matemático de la planta y del diseño de controladores en base a dicho modelo.



5.2. Recomendaciones

En base a las conclusiones detalladas en la sección anterior y tomando en cuenta los resultados obtenidos tanto del proceso de entrenamiento como de las pruebas de rendimiento, se realiza la serie de recomendaciones detallada a continuación.

- Dado que en el presente proyecto se demostró la capacidad de uso de sistemas de aprendizaje reforzado profundo como controlador sobre un sistema con dinámica no-lineal e inestable, se recomienda realizar un proyecto futuro dentro en el cual se aplica uno o más agentes de aprendizaje reforzado profundo, a plantas o procesos que son aplicables a tareas de la industria. Dos ejemplos de entornos sugeridos para desarrollar una línea de investigación de aprendizaje reforzado profundo aplicado al control para la industria, son el control de velocidad de motores o control de temperatura.
- Aprovechando la arquitectura modular del sistema desarrollado, se recomienda modificar el agente con la implementación de otros algoritmos disponibles en la literatura de aprendizaje reforzado profundo, tales como DDPG o SAC, los cuales pueden ser después comparados con el agente basado en TD3 usado en el presente proyecto de grado.
- De igual manera, aprovechando la modularidad y fácil modificación de los componentes del sistema, se sugiere usar el trabajo desarrollado como base para aplicar aprendizaje reforzado profundo a otros entornos usados en la academia. Dos ejemplos de entornos sugeridos, son el sistema de barra y bola o el péndulo de Furuta.
- Se recomienda también el uso del agente en conjunción con una simulación de la planta, la cual puede ser usada para entrenar al agente previamente a que este sea usado para interactuar con el entorno real. De esta manera, se hace posible acelerar el proceso de entrenamiento de manera significativa.
- Continuando, se recomienda la realización de un sistema de control usando aprendizaje reforzado profundo con un enfoque híbrido, en el cual se entrena el agente usando una simulación de la planta durante un número determinado de episodios. Seguidamente, se puede continuar el entrenamiento del agente usando la planta real durante otro número de episodios. El uso de este enfoque podría también acelerar substancialmente el entrenamiento del agente.
- Tomando en cuenta la reciente inclusión de GPUs en sistemas embebidos orientados a combinar el internet de las cosas con tareas de aprendizaje automático, tales como la serie de placas Jetson de Nvidia, se recomienda la implementación y ejecución del módulo principal en ellas o dispositivos similares. Con la inclusión de un dispositivo similar, se eliminaría la necesidad de dedicar una estación de trabajo para llevar a cabo el proceso de entrenamiento.
- Finalmente, considerando que el desarrollo e investigación en sistemas de aprendizaje reforzado profundo es reciente y constituye un campo activo de interés a nivel mundial, dentro el cual tanto la academia como la industria buscan desarrollar aplicaciones innovativas haciendo uso de sus técnicas, se recomienda continuar sobre la base del proyecto desarrollado, o en campos relacionados, proyectos y líneas de investigación de aprendizaje reforzado profundo dentro la universidad, con el fin de beneficiar a la sociedad Boliviana.

Bibliografía

- [1] M. Sewak, «What is artificial intelligence and how does reinforcement learning relate to It?», de *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*, Springer, 2019, p. 1.
- [2] P. H. R. I. M. G. B. J. P. Vincent Francois-Lavet, «An Introduction to Deep Reinforcement Learning», *Foundations and Trends in Machine Learning*, vol. 11, nº 3-4, 2018.
- [3] O. Mayr, «The Origins of Feedback Control», MIT Press, 1970, pp. 11-13.
- [4] H. Nyquist, «Regeneration Theory», *Bell System Technical Journal*, vol. 11, nº 1, pp. 3-24, 1932.
- [5] H. W. Bode, Network Analysis and Feedback Amplifier Design, Princeton, New Jersey: D. Van Nostrand Company, 1945.
- [6] R. Dorf, «History of automatic control», de *Modern Control Systems 7th Edition*, Addisson Wesley, 1995, p. 5.
- [7] R. Dorf, «Closed-loop feedback control systems», de *Modern Control Systems 7th Edition*, Addisson Wesley, 1995, p. 3.
- [8] P. J. Antsalakis, «Fundamental Characteristics of Feedback Mechanisms», University of Notre Dame, Notre Dame, Indiana, 2011.
- [9] N. Nise, «The design process», de *Control Systems Engineering 7th Edition*, Wiley, 2015, pp. 14-17.
- [10] A. D. J. L. C. W. Sean Ashley, «The Stability of an Inverted Pendulum», The University of Arizona, Tucson, Arizona, 2013.
- [11] R. v. d. Boomgaard, «Classical Control Theory», 2018. [En línea]. Available: <https://staff.fnwi.uva.nl/r.vandenboomgaard/SignalProcessing/Applications/ControlTheory/index.html>. [Último acceso: 23 Noviembre 2019].
- [12] O. Boubaker, «The Inverted Pendulum Benchmark in Nonlinear Control Theory: A Survey», *International Journal of Advanced Robotic Systems*, vol. 213, nº 1, pp. 233-242, 2013.
- [13] A. b. Richard Sutton, «Reinforcement learning», de *Reinforcement Learning: An Introduction 2nd Edition*, MIT Press, 2018, pp. 1-5.

- [14] R. Sutton, «History of Reinforcement Learning,» 04 Enero 2005. [En línea]. Available: <http://incompleteideas.net/book/ebook/node12.html>. [Último acceso: Diciembre 2019].
- [15] K. K. D. S. A. G. I. A. D. W. M. R. Volodymyr Mnih, «Playing Atari with Deep Reinforcement Learning,» de *NIPS 2013: Neural Information Processing Systems*, Lake Tahoe, Nevada, 2013.
- [16] J. Vincent, «DeepMind's AI agents conquer human pros at StarCraft II,» The Verge, 24 Enero 2019. [En línea]. Available: <https://www.theverge.com/2019/1/24/18196135/google-deepmind-ai-starcraft-2-victory>. [Último acceso: Diciembre 2019].
- [17] A. N. Matt Vitelli, «CARMA: A Deep Reinforcement Learning Approach to Autonomous Driving,» Stanford University, Stanford, California, 2016.
- [18] D. K. G. V. W. M. J. B. A. Rupam Mahmood, «Benchmarking Reinforcement Learning Algorithms on Real-World Robots,» 2018. [En línea]. Available: arXiv: 1809.07731.
- [19] D. T. S. S. J. L. J. M. G. W. Y. T. T. E. Z. W. S. M. A. E. M. R. D. S. Nicolas Heess, «Emergence of Locomotion Behaviours in Rich Environments,» 2017. [En línea]. Available: arXiv: 1707.02286.
- [20] J. M. Z. W. Nicolas Heess, «Producing flexible behaviours in simulated environments,» Deepmind, 10 Julio 2017. [En línea]. Available: <https://www.deepmind.com/blog/producing-flexible-behaviours-in-simulated-environments>. [Último acceso: 5 Enero 2022].
- [21] B. Copeland, «Encyclopedia Britannica,» Encyclopædia Britannica, 14 December 2021. [En línea]. [Último acceso: 16 December 2021].
- [22] F. Chollet, "Machine learning," in *Deep Learning With Python 2nd Edition*, New York, Manning Publications, 2021, p. 4.
- [23] F. Chollet, «Learning rules and representations from data,» de *Deep Learning with Python 2nd Edition*, New York, Manning Publications, 2021, p. 6.
- [24] J. Grus, «What is machine learning?,» de *Data Science From Scratch: First Principles with Python 2nd Edition*, Sebastopol, California, O'Reilly, 2019, p. 148.
- [25] M. Nielsen, «Chapter 1 - Using neural nets to recognize handwritten digits,» Diciembre 2019. [En línea]. Available: <http://neuralnetworksanddeeplearning.com/chap1.html>. [Último acceso: 25 Octubre 2021].
- [26] A. Glassner, «Fully connected Layers,» de *Deep Learning: A visual Approach*, San Francisco, No Starch Press, 2021, p. 328.

- [27] E. C. Kai Fong, «A Closer Look At The Approximation Capabilities of Neural Networks,» de *ICLR 2020 : International Conference on Learning Representations*, Addis Ababa, Ethiopia, 2020.
- [28] U. Michelucci, «Network Architecture,» de *Deep Learning: A Case Based Approach*, New York, Apress, 2017, pp. 84-85.
- [29] A. Burkov, «Shallow vs Deep Learning,» de *The Hundred-Page Machine Learning Book*, 2019, p. 21.
- [30] Y. B. A. C. Ian Goodfellow, «Historical Notes,» de *Deep Learning*, Cambridge, Massachusetts, MIT Press, 2017, p. 226.
- [31] «Initializing neural networks,» DeepLearning.AI, 2019. [En línea]. Available: <https://www.deeplearning.ai/ai-notes/initialization/>. [Último acceso: 24 Noviembre 2021].
- [32] P. Werbos, «Backpropagation through time: what it does and how to do it,» *Proceedings of the IEEE*, vol. 78, nº 10, pp. 1550 - 1560, 1990.
- [33] K. Nikhil, «Batch, stochastic (Single and Mini-batch) descent,» de *Deep Learning with Python: A Hands-on Introduction*, New York, Apress, 2017, pp. 113-132.
- [34] J. L. B. Diederik Kingma, «ADAM: A Method for Stochastic Optimization,» de *3rd International Conference on Learning Representations*, San Diego, CA, 2015.
- [35] H. B. Youssef Fenjiro, «Deep Reinforcement Learning Overview of the State of the Art,» *Journal of Automation, Mobile Robotics & Intelligent Systems*, vol. 12, nº 3, pp. 20-39, 2018.
- [36] M. P. D. M. B. A. A. B. Kai Arulkumaran, «A Brief Survey of Deep Reinforcement Learning,» *IEEE Signal Processing Magazine, Special Issue on Deep Learning for Image Understanding*, pp. 26-38, Noviembre 2017.
- [37] L. G. Keng Wah Loon, «Reinforcement Learning,» de *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*, Boston, MA, Addison-Wesley Professional, 2019, p. 3.
- [38] M. Morales, «The environment: Everything else,» de *Grokking Deep Reinforcement Learning*, Shelter Island, NY, Manning Publications, 2020, p. 37.
- [39] M. Morales, «Deep reinforcement learning agents improve their behavior through trial-and-error learning,» de *Grokking Deep Reinforcement Learning*, Shelter Island, NY, Manning Publications, 2020, p. 10.
- [40] M. Morales, «States: Specific configurations of the Environment,» de *Grokking Deep Reinforcement Learning*, Manning Publications, 2020, p. 49.

- [41] M. Sewak, «The Markov decision process,» de *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*, Springer, 2019, p. 19.
- [42] M. Morales, «States: Specific configurations of the environment,» de *Grokking Deep Reinforcement Learning*, Manning Publications, 2020, p. 47.
- [43] M. W. Martijn van Otterlo, «Reinforcement Learning: State of the Art,» Berlin, Germany, Springer-Verlag, 2012, p. 11.
- [44] P. WInder, «Reward Engineering,» de *Reinforcement Learning: Industrial Applications of Intelligent Agents*, Sebastopol, CA, O'Reilly, 2020, p. 26.
- [45] P. Winder, «Discounted Return,» de *Reinforcement Learning: Industrial Applications of Intelligent Agents*, Sebastopol, CA, O'Reilly, 2020, pp. 42-43.
- [46] M. Lapan, «Policy,» de *Deep Reinforcement Learning Hands-On*, Packt, 2020, p. 23.
- [47] S. Ravichandiran, «State value function,» de *Hands-On Reinforcement Learning with Python: Master reinforcement and deep reinforcement learning using OpenAI Gym and TensorFlow*, Packt, 2018, p. 46.
- [48] S. Ravichandiran, «State-action value function (Q function),» de *Hands-On Reinforcement Learning with Python: Master reinforcement and deep reinforcement learning using OpenAI Gym and TensorFlow*, Packt, 2018, p. 46.
- [49] L. G. Keng Wah Loon, «Reinforcement Learning as MDP,» de *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*, Boston, MA, Addison-Wesley Professional, 2019, p. 8.
- [50] M. Hensel, «Exploration Methods in Sparse Reward Environments,» de *Reinforcement Learning Algorithms: Analysis and Applications*, Springer, 2021, p. 36.
- [51] J. J. H. A. P. N. H. T. E. Y. T. D. S. D. W. Timothy P. Lillicrap, «Continuous control with deep reinforcement learning,» Google Deepmind, 2016.
- [52] OpenAI, «Twin Delayed DDPG,» OpenAI, 2018. [En línea]. Available: Twin Delayed DDPG. [Último acceso: 12 Febrero 2022].
- [53] H. v. H. D. M. Scott Fujimoto, «Addressing Function Approximation Error in Actor-Critic Methods,» de *International Conference on Machine Learning 2018*, Stockholm, 2018.
- [54] J.-J. E. S. W. L. Jean-Jacques E. Slotine, «Lyapunov Theory,» de *Applied Nonlinear Control*, Englewood Cliffs, NJ, Prentice-Hall, 1991, pp. 15-16.

- [55] I.-C. M. S.-I. N. Islam Boussaada, «Inverted pendulum stabilization: Characterization of codimension-three triple zero bifurcation via multiple delayed proportional gains,» *Systems and Control Letters*, vol. 82, pp. 1-9, 2015.
- [56] B. W. Bequette, «Linear State Space Models,» de *Process Control: Modeling, Design and Simulation*, Prentice Hall Professional, 2003, p. 82.
- [57] Seti@Home, «CPU performance,» Seti@Home, 23 Noviembre 2021. [En línea]. Available: https://setiathome.berkeley.edu/cpu_list.php. [Último acceso: 23 Noviembre 2021].
- [58] Heavy.AI, «Parallel Computing,» Heavy.AI, [En línea]. Available: <https://www.heavy.ai/technical-glossary/parallel-computing>. [Último acceso: 23 Noviembre 2021].
- [59] TechPowerUp, «Nvidia GeForce GTX 1050,» TechPowerUp, [En línea]. Available: <https://www.techpowerup.com/gpu-specs/ geforce-gtx-1050.c2875>. [Último acceso: 23 Noviembre 2021].
- [60] StepperOnline, «Stepper Motor 17HS24-2104S,» [En línea]. Available: <https://www.omc-stepperonline.com/download/17HS24-2104S.pdf>. [Último acceso: 14 Diciembre 2021].
- [61] StepperOnline, «User's Manual for DM542T Full Digital Stepper Drive,» 2017. [En línea]. Available: <https://www.omc-stepperonline.com/download/DM542T.pdf>. [Último acceso: 14 Diciembre 2021].
- [62] StepperOnline, «Digital Stepper Driver 1.0-4.2A 20-50VDC for Nema 17, 23, 24 Stepper Motor,» [En línea]. Available: <https://www.omc-stepperonline.com/digital-stepper-driver-1-0-4-2a-20-50vdc-for-nema-17-23-24-stepper-motor-dm542t>. [Último acceso: 7 Julio 2022].
- [63] S. Microelectronics, «STM32 Nucleo-144 development board with STM32F767ZI MCU, supports Arduino, ST Zio and morpho connectivity,» ST Microelectronics, [En línea]. Available: <https://www.st.com/en/evaluation-tools/nucleo-f767zi.html>. [Último acceso: 23 Abril 2020].
- [64] B. G. Electronics and Robotics Club, «Blue Pill (STM32F103C8T6),» Electronics and Robotics Club, BITS Goa, [En línea]. Available: https://erc-bpgc.github.io/handbook/electronics/Development_Boards/STM32/. [Último acceso: 22 Abril 2020].
- [65] Digilent, «chipKIT Uno32 Reference Manual,» Digilent, [En línea]. Available: https://digilent.com/reference/chipkit_uno32/refmanual. [Último acceso: 22 Abril 2020].

- [66] «STM32 Timer Encoder Mode – STM32 Rotary Encoder Interfacing,» DeepBlue Embedded, 12 Marzo 2021. [En línea]. Available: <https://deepbluembedded.com/stm32-timer-encoder-mode-stm32-rotary-encoder-interfacing/>. [Último acceso: 13 Mayo 2021].
- [67] «General Python FAQ,» Python Software Foundation, 8 Diciembre 2021. [En línea]. Available: <https://docs.python.org/3/faq/general.html>. [Último acceso: 11 Diciembre 2021].
- [68] C. Voskoglou, «What is the best programming language for Machine Learning?,» 5 Mayo 2017. [En línea]. Available: <https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7>. [Último acceso: 11 Diciembre 2021].
- [69] M. AI, «Meta AI Tools - Pytorch,» Meta AI, [En línea]. Available: <https://ai.facebook.com/tools/pytorch/>. [Último acceso: 12 Enero 2022].
- [70] G. O. Source, «Tensorflow,» Google Open Source, [En línea]. Available: <https://www.tensorflow.org>. [Último acceso: 23 Enero 2022].
- [71] Keras, «Keras,» Keras, [En línea]. Available: <https://keras.io>. [Último acceso: 22 Enero 2022].
- [72] Numpy Developers, «What is NumPy?,» NumPy, [En línea]. Available: <https://numpy.org/devdocs/user/whatisnumpy.html>.
- [73] C. Liechti, «PySerial Overview,» 2021. [En línea]. Available: <https://pyserial.readthedocs.io/en/latest/pyserial.html>. [Último acceso: 3 Enero 2022].
- [74] A. Mbed, «Mbed Rapid IoT device development,» ARM, [En línea]. Available: <https://os.mbed.com>. [Último acceso: 26 Enero 2021].
- [75] S. Microelectronics, «STM32CubeIDE Integrated Development Environment for STM32,» ST Microelectronics, [En línea]. Available: <https://www.st.com/en/development-tools/stm32cubeide.html>. [Último acceso: 26 Enero 2021].
- [76] «μVision® IDE,» ARM Keil, [En línea]. Available: <https://www2.keil.com/mdk5/uvision/>. [Último acceso: 27 Enero 2022].
- [77] F. Chollet, «Inference: Using a model after training,» de *Deep Learning with Python 2nd Edition*, New York, Manning Publications, 2021, p. 93.

Apéndice

A.1. Código del Subsistema de Medición y Control

```
#include "main.h"
#include "Peripherals.h"

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
TIM_HandleTypeDef htim5;
UART_HandleTypeDef huart3;
DMA_HandleTypeDef hdma_usart3_rx;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);
static void MX_TIM5_Init(void);

void hallar_medio(void);
void frecuencia_driver(int);
void activar_driver();
void desact_driver();

// Para almacenar valores de contadores
int cuenta_pulsos_stepper;
int cuenta_encoder;
int16_t cuenta_encoder_msg;
int16_t cuenta_pulsos_stepper_msg;
// Para recepcion de comandos numericos
int comandoInt= 0;
//Buffers de envio y recepcion
uint8_t buffer_RX[4];
uint8_t buffer_TX[7] = {'0', '0', '0', '0', '0', '0', '0'};
// Constantes para hallar el medio
const float METERS_PER_PULSE= 2* 3.14159265358979323846* 0.0159/ 3200;
const int PULSES_TO_MID= (int)(0.2/ METERS_PER_PULSE);
int main(void){
    /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_USART3_UART_Init();
    MX_TIM2_Init();
    MX_TIM3_Init();
    MX_TIM4_Init();
    MX_TIM5_Init();
    HAL_GPIO_WritePin(motor_enable_GPIO_Port, motor_enable_Pin, GPIO_PIN_SET);
    HAL_TIM_Encoder_Start(&htim5, TIM_CHANNEL_ALL);
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
    HAL_TIM_Base_Start(&htim2);
    HAL_Delay(100);
```

```
    HAL_UART_Receive_DMA (&huart3, buffer_RX, 4);
    HAL_Delay(1000);

}

void SystemClock_Config(void) {
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
    HAL_PWR_EnableBkUpAccess();
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 4;
    RCC_OscInitStruct.PLL.PLLN = 216;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    RCC_OscInitStruct.PLL.PLLR = 2;
    HAL_RCC_OscConfig(&RCC_OscInitStruct);
    HAL_PWREx_EnableOverDrive();
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
    HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7);
}

static void MX_TIM2_Init(void) {
    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 0;
    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim2.Init.Period = 4294967295;
    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    HAL_TIM_Base_Init(&htim2);
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_ETRMODE2;
    sClockSourceConfig.ClockPolarity = TIM_CLOCKPOLARITY_NONINVERTED;
    sClockSourceConfig.ClockPrescaler = TIM_CLOCKPRESCALER_DIV1;
    sClockSourceConfig.ClockFilter = 0;
    HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig);
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig);
}

static void MX_TIM3_Init(void) {

    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_OC_InitTypeDef sConfigOC = {0};

    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 5;
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 2399;
```

```
htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
HAL_TIM_PWM_Init(&htim3);
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig);
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_1);
HAL_TIM_MspPostInit(&htim3);
}

static void MX_TIM4_Init(void) {
    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    htim4.Instance = TIM4;
    htim4.Init.Prescaler = 1000;
    htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim4.Init.Period = 1080;
    htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
    HAL_TIM_Base_Init(&htim4);
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig);
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig);
}

static void MX_TIM5_Init(void) {
    TIM_Encoder_InitTypeDef sConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    htim5.Instance = TIM5;
    htim5.Init.Prescaler = 0;
    htim5.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim5.Init.Period = 4294967295;
    htim5.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim5.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    sConfig.EncoderMode = TIM_ENCODERMODE_TI12;
    sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
    sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
    sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
    sConfig.IC1Filter = 0;
    sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
    sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
    sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
    sConfig.IC2Filter = 0;
    HAL_TIM_Encoder_Init(&htim5, &sConfig);
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    HAL_TIMEx_MasterConfigSynchronization(&htim5, &sMasterConfig);
}

static void MX_USART3_UART_Init(void) {
    huart3.Instance = USART3;
    huart3.Init.BaudRate = 921600;
    huart3.Init.WordLength = UART_WORDLENGTH_8B;
    huart3.Init.StopBits = UART_STOPBITS_1;
```

```
        huart3.Init.Parity = UART_PARITY_NONE;
        huart3.Init.Mode = UART_MODE_TX_RX;
        huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
        huart3.Init.OverSampling = UART_OVERSAMPLING_16;
        huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
        huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
        HAL_UART_Init(&huart3);
    }
static void MX_DMA_Init(void){
    __HAL_RCC_DMA1_CLK_ENABLE();
    HAL_NVIC_SetPriority(DMA1_Stream1_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Stream1_IRQn);
}
static void MX_GPIO_Init(void){
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();
    __HAL_RCC_GPIOG_CLK_ENABLE();
    HAL_GPIO_WritePin(motor_enable_GPIO_Port, motor_enable_Pin,
GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_6, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(direc_motor_GPIO_Port, direc_motor_Pin, GPIO_PIN_RESET);
    GPIO_InitStruct.Pin = button_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(button_GPIO_Port, &GPIO_InitStruct);
    GPIO_InitStruct.Pin = sensor_final_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(sensor_final_GPIO_Port, &GPIO_InitStruct);
    GPIO_InitStruct.Pin = motor_enable_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(motor_enable_GPIO_Port, &GPIO_InitStruct);
    GPIO_InitStruct.Pin = GPIO_PIN_6;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
    GPIO_InitStruct.Pin = USB_OverCurrent_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(USB_OverCurrent_GPIO_Port, &GPIO_InitStruct);
    GPIO_InitStruct.Pin = direc_motor_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(direc_motor_GPIO_Port, &GPIO_InitStruct);
    HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
}
}
```

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    memcpy(&comandoInt, &buffer_RX, 4);
    // Si el comando se encuentra en un rango valido para PWM
    if(comandoInt>= -10000 && comandoInt<= 10000){
        frecuencia_driver(comandoInt);
    // Si el comando es de reseteo del sistema
    } else if(comandoInt== 32768){
        HAL_TIM_Base_Stop_IT(&htim4);
        hallar_medio();
    // Si el comando es de empezar muestreo
    } else if(comandoInt== 65536){
        HAL_TIM_Base_Start_IT(&htim4);
        activar_driver();
    }

    HAL_UART_Receive_DMA(&huart3, buffer_RX, 4);
}

//Enviar muestras en interrupcion del timer
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim) {
    cuenta_encoder= (int) TIM5->CNT;
    cuenta_pulsos_stepper= TIM2-> CNT;
    cuenta_pulsos_stepper_msg= (int16_t) cuenta_pulsos_stepper;
    memcpy(&buffer_TX, &cuenta_pulsos_stepper_msg, 2);
    memcpy(&buffer_TX[2], &cuenta_encoder, 4);
    if(!HAL_GPIO_ReadPin(sensor_final_GPIO_Port, sensor_final_Pin)){
        buffer_TX[6]= 0xFF;
    } else{
        buffer_TX[6]= 0x00;
    }
    HAL_UART_Transmit(&huart3, &buffer_TX, 7, 100);

}

//Devolver el carro a la mitad de la pista
void hallar_medio(void){
    activar_driver();
    //Avanzar hacia el sensor de final de carrera
    frecuencia_driver(-4000);
    while(HAL_GPIO_ReadPin(sensor_final_GPIO_Port, sensor_final_Pin)){}
    TIM2-> CNT= 0;
    //Desplazar el carro desde el borde de la pista hasta el medio
    frecuencia_driver(4000);
    while(abs(TIM2-> CNT)< PULSES_TO_MID){}
    frecuencia_driver(0);
    TIM2-> CNT= 0;
    desact_driver();
}

void activar_driver(){
    HAL_GPIO_WritePin(motor_enable_GPIO_Port, motor_enable_Pin,
    GPIO_PIN_RESET);
}

void desact_driver(){
    HAL_GPIO_WritePin(motor_enable_GPIO_Port, motor_enable_Pin,
    GPIO_PIN_RESET);
}

void frecuencia_driver(int frecuencia){

    int contador= 0;
```

```

if(frecuencia!= 0){
    if(abs(frecuencia)>= 400){
        contador= (uint32_t)108000000/ (abs(frecuencia)* (5+ 1));
        __HAL_TIM_SET_PRESCALER(&htim3, 5);
    } else if(abs(frecuencia< 400)){
        contador= (uint32_t)108000000/ (abs(frecuencia)* (499+ 1));
        __HAL_TIM_SET_PRESCALER(&htim3, 499);
    }
    __HAL_TIM_SET_AUTORELOAD(&htim3, contador);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, contador/ 2);
} else{
    __HAL_TIM_SET_PRESCALER(&htim3, 499);
    __HAL_TIM_SET_AUTORELOAD(&htim3, 1000);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, 0);
}
if (frecuencia> 0){
    HAL_GPIO_WritePin(direc_motor_GPIO_Port, direc_motor_Pin,
GPIO_PIN_SET);
    TIM2->CR1&= 0xFFFFFEF;
} else{
    HAL_GPIO_WritePin(direc_motor_GPIO_Port, direc_motor_Pin,
GPIO_PIN_RESET);
    TIM2->CR1|= 0x00000010;
}
void Error_Handler(void)
{
    __disable_irq();
}

```

A.2. Código del Módulo de Interfaz Agente-Entorno

```

import struct
import math
import serial
import numpy as np

class FiltroExponencial(object):

    def __init__(self, alfa):
        self.alfa = alfa

    def filtrar(self, anterior, actual):
        return self.alfa * actual + (1 - self.alfa) * anterior

class Entorno(object):

    def __init__(self, p_muestreo, tam_ep, puerto='/dev/TTYACM0'):
        # Definir constantes
        self.MEN_RESET = np.int32(32768)
        self.MEN_COMIENZO = np.int32(65536)
        self.T_S = p_muestreo
        self.TAM_EP = tam_ep

        self.FACTOR_ANGULO = -math.pi / 2000
        self.FACTOR_VELOCIDAD= (2 * math.pi / 4000) / self.T_S

```

```
self.PULSOS_REV = 3200
self.RAD_POLEA = 0.0159
self.METROS_POR_PULSO = 2 * math.pi * self.RAD_POLEA /
self.PULSOS_REV

self.LIMITE_REC = 0.15
self.LIMITE_FILT = math.pi / 14

self.C1 = 100 / (math.pi ** 2)
self.C2 = 0.25 * 100 / (0.2 ** 2)
self.C3 = 0.005 * 100 / (math.pi / self.T_S)

self.x_ant = 0
self.theta_ant = 0
self.x_dot = 0
self.theta_dot = 0
self.theta_dot_ant = 0
self.pulsos_enc_ant = 0
self.band_fin = False
self.cont_pasos = 0

self.pend = serial.Serial(puerto, 921600)

self.filt_ang = FiltroExponencial(0.95)
self.filt_vel = FiltroExponencial(0.9)
self.reset_ent()

def enviar(self, cmd):
    mensaje = bytearray(struct.pack("i", cmd))
    self.pend.write(mensaje)
    self.pend.reset_output_buffer()

def conv_angulo(self, pulsos_encoder):
    if(pulsos_encoder >= 0 and pulsos_encoder < 4000):
        angulo = self.FACTOR_ANGULO * (pulsos_encoder - 2000)
    elif(pulsos_encoder > -4000 and pulsos_encoder <= -1):
        angulo = self.FACTOR_ANGULO * (pulsos_encoder + 2000)

    return angulo

def est_ent(self):
    frame = self.pend.read(7)
    valores = struct.unpack('=hi?', frame)
    self.pend.reset_input_buffer()
    # Calcular x, x_dot, theta, theta_dot
    x = self.METROS_POR_PULSO * valores[0]
    theta = self.conv_angulo(math.fmod(valores[1], 4000))
    self.theta_dot = (valores[1] - self.pulsos_enc_ant) *
self.FACTOR_VELOCIDAD

    if theta <= self.LIMITE_FILT and theta >= -self.LIMITE_FILT:
        theta = self.filt_ang.filtrar(self.theta_ant, theta)
        self.theta_dot = self.filt_vel.filtrar(self.theta_dot_ant,
self.theta_dot)
        self.x_dot = (x - self.x_ant) / self.T_S
        self.x_ant = x
```

```

        self.theta_ant = theta
        self.theta_dot_ant = self.theta_dot
        self.pulsos_enc_ant= valores[1]

        if valores[2] or x > 0.19 or (abs(self.theta_dot) > 18 and
self.cont_pasos > 10):
            self.band_fin = True

        # Calcular la recompensa
        if abs(theta) <= self.LIMITE_REC:
            rec = 1000 -self.C1 * (theta ** 2) - self.C2 * (x ** 2) - self.C3
* abs(self.theta_dot) - 10000* self.band_fin
        else:
            rec = -self.C1 * abs(theta ** 2) - self.C2 * (x ** 2) - self.C3 *
abs(self.theta_dot) - 10000 * self.band_fin

        if self.cont_pasos > self.TAM_EP:
            self.band_fin = True

        self.cont_pasos += 1
        return np.array([x, self.x_dot, theta, self.theta_dot],
dtype='float32'), rec, self.band_fin, self.cont_pasos

    def reset_ent(self):
        self.pend.flushOutput()
        self.pend.flushInput()
        self.enviar(self.MEN_RESET)
        self.x_ant = 0
        self.theta_ant = 0
        self.theta_dot_ant = 0
        self.x_dot = 0
        self.theta_dot = 0
        self.pulsos_enc_ant = 0
        self.band_fin = False
        self.cont_pasos = 0

    def comenzar(self):
        self.pend.flushOutput()
        self.pend.flushInput()
        self.enviar(self.MEN_COMIENZO)

```

A.3. Código del Módulo de Memoria de Experiencias

```

import numpy as np

class Memoria:

    #Inicializar el buffer
    def __init__(self, tamano, tam_estado):
        self.tamano = tamano
        self.tam_estado = tam_estado
        self.almacen = np.zeros((tamano, tam_estado * 2 + 3))
        self.puntero = 0
        self.band_lleno = 0

    def almacenar_paso(self, estado, accion, recompensa, estado_siguiente,
fin):

```

```

        datos = np.hstack((estado, accion, recompensa, estado_siguiente,
fin))
        self.almacen[self.puntero, :] = datos
        self.puntero = self.puntero + 1
        if self.puntero == self.tamano:
            self.puntero = 0
            self.band_lleno = 1
    #Escoger un numero de muestras del almacenamiento al azar
    def tomar_muestras(self, numero_muestras):
        if self.band_lleno== 1:
            muestras= np.random.choice(self.tamano, numero_muestras)
            return self.almacen[muestras, :]
        else:
            muestras= np.random.choice(self.puntero, numero_muestras)
            return self.almacen[muestras, :]

    # Funciones para pasar el buffer de RAM a almacenamiento no volatil
    def guardar(self, direccion):
        info = np.array([self.puntero, self.band_lleno])
        np.savez_compressed(f"{direccion}buffer.npz",
                            buf=self.almacen, inf=info)

    def cargar(self, direccion):
        self.almacen = np.load(f"{direccion}buffer.npz",
allow_pickle=True) ["buf"]
        self.puntero, self.band_lleno = np.load(f"{direccion}buffer.npz",
allow_pickle=True) ["inf"].tolist()

```

A.4. Código del Módulo de Redes Neuronales

```

import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as functional
import torch.optim as optim

class RedActor(nn.Module):
    def __init__(self, tam_estado, tam_capas, tasa_apren, procesador):
        super(RedActor, self).__init__()
        self.tam_estado = tam_estado
        self.tam_capas = tam_capas
        self.tasa_apren = tasa_apren
        self.procesador = procesador

        self.lineal1 = nn.Linear(self.tam_estado, self.tam_capas[0])
        self.lineal2 = nn.Linear(self.tam_capas[0], self.tam_capas[1])
        self.salida = nn.Linear(self.tam_capas[1], 1)

        self.optim_actor = optim.Adam(self.parameters(), self.tasa_apren)

        self.to(self.procesador)

    def forward(self, estado):
        a = self.lineal1(estado)
        a = functional.relu(a)
        a = self.lineal2(a)
        a = functional.relu(a)

```

```

    a = self.salida(a)
    a = torch.tanh(a)

    return a

def guardar(self, direccion):
    torch.save(self.state_dict(), direccion)

def cargar(self, direccion):
    self.load_state_dict(torch.load(direccion))

class RedCritico(nn.Module):
    def __init__(self, tam_estado, tam_capas, tasa_apren, procesador):
        super(RedCritico, self).__init__()
        self.tam_estado = tam_estado
        self.tam_capas = tam_capas
        self.tasa_apren = tasa_apren
        self.procesador = procesador

        self.lineal1 = nn.Linear(self.tam_estado + 1, self.tam_capas[0])
        self.lineal2 = nn.Linear(self.tam_capas[0], self.tam_capas[1])
        self.salida = nn.Linear(self.tam_capas[1], 1)

        self.optim_critico = optim.Adam(self.parameters(), self.tasa_apren)

        self.to(self.procesador)

    def forward(self, estado, accion):
        entrada = torch.cat([estado, accion], 1)
        q = self.lineal1(entrada)
        q = functional.relu(q)
        q = self.lineal2(q)
        q = functional.relu(q)
        q = self.salida(q)

        return q

    def guardar(self, direccion):
        torch.save(self.state_dict(), direccion)

    def cargar(self, direccion):
        self.load_state_dict(torch.load(direccion))

```

A.5. Código de la Clase Agente

```

import numpy as np
import torch
import torch.nn.functional as functional

from Memoria import Memoria
from Redes import RedActor, RedCritico

class Agente():
    def __init__(self, rho, gamma, interval_act_actor, entrenamientos_ep,
tam_memoria, tam_lote):
        self.gamma = gamma

```

```
self.rho = rho
self.interval_act_actor = interval_act_actor
self.entrenamientos_ep = entrenamientos_ep
self.tam_memoria = tam_memoria
self.tam_lote = tam_lote
if torch.cuda.is_available():
    self.procesador = torch.device("cuda")
else:
    self.procesador = torch.device("cpu")

# Crear instancias de las redes neuronales.
self.actor = RedActor( 4, [256, 256], 0.001, self.procesador)
self.actor_obj = RedActor( 4, [256, 256], 0.001, self.procesador)
self.copiar_red(self.actor, self.actor_obj)
self.critico_1 = RedCritico(4, [512, 256], 0.001, self.procesador)
self.critico_1_obj = RedCritico(4, [512, 256], 0.001,
self.procesador)
    self.copiar_red(self.critico_1, self.critico_1_obj)
self.critico_2 = RedCritico(4, [512, 256], 0.001, self.procesador)
self.critico_2_obj = RedCritico(4, [512, 256], 0.001,
self.procesador)
    self.copiar_red(self.critico_2, self.critico_2_obj)

# Crear instancia del buffer
self.mem = Memoria(self.tam_memoria, 4)

def copiar_red(self, red, red_obj):
    param_red = dict(red.named_parameters())
    param_obj = dict(red_obj.named_parameters())

    for nombre in param_red:
        param_obj[nombre] = param_red[nombre].clone()

    red_obj.load_state_dict(param_obj)

def perturbar_red(self, red, red_obj):
    param_red = dict(red.named_parameters())
    param_obj = dict(red_obj.named_parameters())

    for nombre in param_red:
        param_obj[nombre] = self.rho* param_red[nombre].clone() + \
            (1 - self.rho) * param_obj[nombre].clone()

    red_obj.load_state_dict(param_obj)

def generar_accion(self, estado):
    accion = self.actor.forward(estado)
    accion = accion.to(torch.device("cpu")).detach().numpy()
    return accion[0]

def almacenar_paso(self, s, a, r, s_sig, fin):
    self.mem.almacenar_paso(s, a, r, s_sig, fin)

def entrenar(self):

    for entrenamiento in range(self.entrenamientos_ep):
        muestras = self.mem.tomar_muestras(self.tam_lote)
```

```
estado = muestras[:, 0: 4]
estado = torch.FloatTensor(estado).to(self.procesador)
accion = np.array(muestras[:, 4]).reshape(self.tam_lote, 1)
accion = torch.FloatTensor(accion).to(self.procesador)
recompensa = np.array(muestras[:, 5]).reshape(self.tam_lote, 1)
recompensa = torch.FloatTensor(recompensa).to(self.procesador)
estado_sig = muestras[:, 6: 10]
estado_sig = torch.FloatTensor(estado_sig).to(self.procesador)
fin= np.array(muestras[:, 10]).reshape(self.tam_lote, 1)
fin= torch.BoolTensor(fin).to(self.procesador)

acciones_obj = self.actor_obj.forward(estado_sig)
ruido_reg = torch.tensor(np.random.normal(scale=0.15))
acciones_obj = acciones_obj + torch.clamp(ruido_reg, -0.5, 0.5)
acciones_obj = torch.clamp(acciones_obj, -1.0, 1.0)

q_1_obj = self.critico_1_obj.forward(estado_sig, acciones_obj)
q_1_obj[fin] = 0.0
q_2_obj = self.critico_2_obj.forward(estado_sig, acciones_obj)
q_2_obj[fin] = 0.0
q_obj_min = torch.min(q_1_obj, q_2_obj)

q_1 = self.critico_1.forward(estado, accion)
q_2 = self.critico_2.forward(estado, accion)

obj = recompensa + self.gamma * q_obj_min

self.critico_1.optim_critico.zero_grad()
self.critico_2.optim_critico.zero_grad()

perdida_critico_1 = functional.mse_loss(obj, q_1)
perdida_critico_2 = functional.mse_loss(obj, q_2)
perdida_criticos = perdida_critico_1 + perdida_critico_2
perdida_criticos.backward()
self.critico_1.optim_critico.step()
self.critico_2.optim_critico.step()

if entrenamiento % self.interval_act_actor == 0 and entrenamiento > 0:
    self.actor.optim_actor.zero_grad()
    acciones = self.actor.forward(estado)
    q_actor = self.critico_1.forward(estado, acciones)
    perdida_actor = -torch.mean(q_actor)
    perdida_actor.backward()
    self.actor.optim_actor.step()

    self.perturbar_red(self.actor, self.actor_obj)
    self.perturbar_red(self.critico_1, self.critico_1_obj)
    self.perturbar_red(self.critico_2, self.critico_2_obj)

self.guardar()

def guardar(self):
    self.mem.guardar("Guardados/")
    self.actor.guardar("Guardados/actor.pt")
    self.actor_obj.guardar("Guardados/actor_obj.pt")
    self.critico_1.guardar("Guardados/critico_1.pt")
```

```

    self.critico_1_obj.guardar("Guardados/critico_1_obj.pt")
    self.critico_2.guardar("Guardados/critico_2.pt")
    self.critico_2_obj.guardar("Guardados/critico_2_obj.pt")

def cargar(self):
    self.mem.cargar("Guardados/")
    self.actor.cargar("Guardados/actor.pt")
    self.actor_obj.cargar("Guardados/actor_obj.pt")
    self.critico_1.cargar("Guardados/critico_1.pt")
    self.critico_1_obj.cargar("Guardados/critico_1_obj.pt")
    self.critico_2.cargar("Guardados/critico_2.pt")
    self.critico_2_obj.cargar("Guardados/critico_2_obj.pt")

```

A.6. Código del Módulo de Exploración

```

import numpy as np

class Exploracion:

    def __init__(self, esc_ruido, direccion):
        self.direccion = direccion
        self.escala_ruido = esc_ruido
        self.ACC_INICIO = [i * 0.1 for i in range(-10, 11)]

    def gen_prim_acc(self, num_ep):
        accion = self.ACC_INICIO[num_ep]
        accion = self.anadir_ruido_accion(accion)
        return accion

    def gen_acc_ruido(self):
        ruido = np.random.normal(scale=0.5)
        return ruido

    def anadir_ruido_accion(self, accion):
        ruido = np.random.normal(scale=self.escala_ruido)
        accion_ruido = accion + ruido
        accion_ruido = np.clip(accion_ruido, -1, 1)
        self.escala_ruido = self.escala_ruido * 0.9999985
        return accion_ruido

    def guardar(self):
        escala= np.array([self.escala_ruido])
        np.savez_compressed(f"{self.direccion}escala_ruido.npz",
                            esc=escala)

    def cargar(self):
        self.escala_ruido = np.load(f"{self.direccion}escala_ruido.npz",
                                    allow_pickle=True) ["esc"]
        self.escala_ruido = float(self.escala_ruido)

```

A.7. Código del Módulo Principal

```

import time
import signal
import sys
import math

from Entorno import Entorno

```

```
from Agente import Agente
from Metrica import Metrica
from Exploracion import Exploracion

import torch
import numpy as np

# Tomar argumentos de la linea de comando del sistema
args = sys.argv

# Parametros del lazo de entrenamiento
EPS_INICIALES = 20
EPS_RUIDO = EPS_INICIALES + 10
MAX_EPS = 1400
MAX_PASOS_EP = 4500
MAX_CAMBIO_ACCION = 5000 / 10000
ep_actual = 0

# Instanciar objetos
entorno = Entorno(0.01, MAX_PASOS_EP, '/dev/ttyACM0')
agente = Agente(0.005, 0.99, 2, 3000, 1000000, 256)
metrica = Metrica("Guardados/")
explorador = Exploracion(0.2, "Guardados/")

if args[1] == "cargar":
    agente.cargar()
    ep_actual = metrica.cargar()
    explorador.cargar()

# Para salir.
def salida(signal, frame):
    print("Salida")
    entorno.reset_ent()
    sys.exit(0)
signal.signal(signal.SIGINT, salida)

while ep_actual <= MAX_EPS:

    #Inicializar el episodio, resetear posicion planta
    time.sleep(10)
    rec_ep= 0
    num_paso= 0
    accion_ant = 0

    #Empezar episodio
    entorno.comenzar()
    est_ant, rec, fin_ep, num_paso = entorno.est_ent()
    est_ant, rec, fin_ep, num_paso = entorno.est_ent()
    est_ant, rec, fin_ep, num_paso = entorno.est_ent()

    while True:
        if fin_ep:
            break
        if ep_actual <= EPS_INICIALES:
            accion = explorador.gen_prim_acc(ep_actual)
        elif ep_actual > EPS_INICIALES and ep_actual< EPS_RUIDO:
```

```

        accion = float(explorador.gen_acc_ruido())
    else:
        accion =
float(agente.generar_accion(torch.from_numpy(est_ant).to(agente.procesador)))
        accion = explorador.anadir_ruido_accion(accion)
        accion = np.clip(accion, -1, 1)
        dif_accion = accion - accion_ant
        if abs(dif_accion) > MAX_CAMBIO_ACCION:
            accion = accion_ant + math.copysign(MAX_CAMBIO_ACCION,
dif_accion)
            cmd= np.int32(accion * 10000)
            entorno.enviar(cmd)
            est_act, rec, fin_ep, num_paso = entorno.est_ent()
            agente.almacenar_paso(est_ant, accion, rec, est_act, fin_ep)
            rec_ep += rec
            est_ant= est_act
            accion_ant = accion
#Reset del entorno
entorno.reset_ent()

#Ejecutar las funciones de metrica
prom_rec= metrica.prom_rec(rec_ep, num_paso)
print("Episodio numero: ", ep_actual)
print("La recompensa promedio del episodio fue: ", prom_rec)

#Correr entrenamiento
if ep_actual >= EPS_RUIDO:
    agente.entrenar()

metrica.guardar(ep_actual)
explorador.guardar()
ep_actual += 1

```

A.8. Código del Módulo de Demostración

```

import time
import signal
import sys
import math

from Entorno import Entorno
from Agente import Agente
from Metrica import Metrica
from Exploracion import Exploracion

import torch
import numpy as np
from mpl_toolkits import mplot3d
from matplotlib import pyplot as plt

# Tomar argumentos de la linea de comando del sistema
args = sys.argv

if args[1] == "correr_largo":
    MAX_PASOS_EP = 1000000
elif args[1] == "correr_corto":

```

```
MAX_PASOS_EP = 4500

# Parametros del lazo de entrenamiento
T_S = 0.01
MAX_CAMBIO_ACCION = 5000 / 10000
DPI = 96
recompensas_trayectoria = []
acciones_trayectoria = []
x_trayectoria = []
x_dot_trayectoria = []
theta_trayectoria = []
theta_dot_trayectoria = []
plt.rcParams["font.size"] = 15

# Instanciar objetos
entorno = Entorno(0.01, MAX_PASOS_EP, '/dev/ttyACM0')
agente = Agente(0.005, 0.99, 2, 3000, 1000000, 256)
metrica = Metrica("Guardados/")
explorador = Exploracion(0.2, "Guardados/")

agente.cargar()
ep_actual = metrica.cargar()
explorador.cargar()

def plot_2d(t, y, titulo, nombre_t, nombre_y, nombre_archivo, res_x, res_y):
    dib = plt.figure(figsize=(res_x / DPI, res_y / DPI), dpi=DPI)
    ax = dib.add_subplot(111)
    ax.plot(t, y)
    for axis in ['top', 'bottom', 'left', 'right']:
        ax.spines[axis].set_linewidth(1.0)
    ax.set_xlabel(nombre_t)
    ax.set_ylabel(nombre_y)
    ax.set_title(titulo)
    ax.grid()
    plt.savefig(nombre_archivo)

def guardar_datos(x, x_dot, theta, theta_dot, accion, rec):
    np.savez_compressed(f"Guardados/trayectoria.npz", x=x, x_dot=x_dot,
                        theta=theta, theta_dot=theta_dot, accion=accion, rec=rec)

# Plotear el rendimiento del sistema
def plot_trayectoria(recompensa, accion, x, x_dot, theta, theta_dot):
    # Plotear la recompensa
    eje_tiempo = np.linspace(0, len(recompensa), len(recompensa)) * T_S
    eje_recompensa = np.array(recompensa)
    eje_accion = np.array(accion)
    eje_x = np.array(x)
    eje_x_dot = np.array(x_dot)
    eje_theta = np.array(theta)
    eje_theta_dot = np.array(theta_dot)
    plot_2d(eje_tiempo, eje_recompensa, "Recompensa a lo largo de la
trayectoria",
            "Tiempo [s]", "Recompensa", "plot_recompensa.png", 1920, 600)
    # Plotear la trayectoria
    plot_2d(eje_tiempo, eje_x, "Posicion del carro a lo largo de la
trayectoria",
            "Tiempo [s]", "Posicion x [m]", "plot_x.png", 1920, 600)
```

```
plot_2d(eje_tiempo, eje_x_dot, "Velocidad del carro a lo largo de la trayectoria",
         "Tiempo [s]", r"Velocidad $\dot{x}$ $\frac{m}{s}$",
"plot_x_dot.png", 1920, 600)
plot_2d(eje_tiempo, eje_theta, "Posicion angular del pendulo a lo largo de la trayectoria",
         "Tiempo [s]", r"Posicion angular $\theta$ [rad]", "plot_theta.png",
1920, 600)
plot_2d(eje_tiempo, eje_theta_dot, "Velocidad angular del pendulo a lo largo de la trayectoria",
         "Tiempo [s]", r"Velocidad $\dot{\theta}$ $\frac{rad}{s}$",
"plot_theta_dot.png", 1920, 600)
plot_2d(eje_tiempo, eje_accion, "Acciones del agente a lo largo de la trayectoria",
         "Tiempo [s]", r"Accion a", "plot_accion.png", 1920, 600)
guardar_datos(eje_x, eje_x_dot, eje_theta, eje_theta_dot, eje_accion,
eje_recompensa)
# Para salir.
def salida(signal, frame):
    print("Salida")
    entorno.reset_ent()
    plot_trayectoria(recompensas_trayectoria, acciones_trayectoria,
x_trayectoria, x_dot_trayectoria,
theta_trayectoria, theta_dot_trayectoria)
    sys.exit(0)
signal.signal(signal.SIGINT, salida)

time.sleep(10)
rec_ep= 0 #Recompensa total episodio
num_paso= 0
accion_ant = 0

entorno.comenzar()
est_ant, rec, fin_ep, num_paso = entorno.est_ent()
est_ant, rec, fin_ep, num_paso = entorno.est_ent()
est_ant, rec, fin_ep, num_paso = entorno.est_ent()

while True:
    if fin_ep:
        break
    accion =
float(agente.generar_accion(torch.from_numpy(est_ant).to(agente.procesador)))
    accion = explorador.anadir_ruido_accion(accion)
    accion = np.clip(accion, -1, 1)
    dif_accion = accion - accion_ant
    if abs(dif_accion) > MAX_CAMBIO_ACCION:
        accion = accion_ant + math.copysign(MAX_CAMBIO_ACCION, dif_accion)
    cmd= np.int32(accion * 10000)
    entorno.enviar(cmd)
    acciones_trayectoria.append(accion)
    recompensas_trayectoria.append(rec)
    x_trayectoria.append(est_ant[0])
    x_dot_trayectoria.append(est_ant[1])
    theta_trayectoria.append(est_ant[2])
    theta_dot_trayectoria.append(est_ant[3])
```

```
est_act, rec, fin_ep, num_paso = entorno.est_ent()
rec_ep += rec
est_ant= est_act
accion_ant = accion
#Reset del entorno
entorno.reset_ent()

#Ejecutar las funciones de metrica
prom_rec= metrica.prom_rec(rec_ep, num_paso)
print("La recompensa promedio de la trayectoria fue: ", prom_rec)
plot_trayectoria(recompensas_trayectoria, acciones_trayectoria,
x_trayectoria, x_dot_trayectoria,
theta_trayectoria, theta_dot_trayectoria)
```

Correo: rhc Calderon11@gmail.com

Celular: 75809660