

Benemérita Universidad Autónoma de Puebla

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

DOCTORADO EN INGENIERÍA DEL LENGUAJE Y DEL
CONOCIMIENTO



**“Agente que aprende a controlar un robot con técnicas de
aprendizaje profundo por refuerzo”**

TESIS

Presentada para obtener el grado de:

Doctor en ingeniería del lenguaje y del conocimiento

Presenta:

M. en C. Francisco José López Cortés

Asesor:

Dr. David Eduardo Pinto Avendaño

Puebla, Pue. Diciembre 2025.

Dedicatoria

Dedico este trabajo de investigación a los seres que más amo en este mundo, a mi esposa Carolina quien siempre me ha inspirado a seguir mejorando y ser un hombre de bien.

A mis padres María Isabel y Alfredo, quienes con su ejemplo y dedicación fueron inspiración para llegar a esta etapa de mi vida.

A mi hermano Alfredo quien siempre me motiva a luchar por mis sueños.

Agradecimientos

Quiero agradecer en primer lugar a Dios por permitirme culminar esta etapa de mi vida, por brindarme salud y entusiasmo en todo momento, y sobre todo, agradecerle por la salud y vida de mi esposa y mi familia quienes me han acompañado a lo largo de este camino.

A mi esposa Carolina, por el cariño y amor tan sublime que me das, gracias mi amor por tantas aventuras juntos y sobre todo por ser mi compañera de vida, de proyectos, de sueños, por todo el apoyo incondicional que me brindas.

A mis padres María Isabel y Alfredo, por todo el apoyo que me brindan en cada etapa de mi vida. Gracias por todos sus consejos y su ejemplo de lucha y perseverancia.

A mi hermano Alfredo, por siempre acompañarme con tus pláticas y entusiasmo.

A mis amigos Rigoberto, Marcos y Rabi, por hacer grato el tiempo de estudio y eventos en los que participamos. A mi asesor de tesis el Dr. David Pinto Avendaño, a quien considero un amigo ya que siempre me apoyo y confío en mí para culminar este trabajo.

También agradecer al Dr. Fernando Perez-Tellez quien supervisó mi estancia en la TU Dublin, por compartir sus conocimientos y enriquecer mi investigación.

Por último, agradecer a la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (Secihti) por el apoyo económico brindado, a la Vicerrectoría de Investigación y Estudios de Posgrado (VIEP) y la Technological University Dublin, por financiar estancias de investigación que aportaron contenido esencial y de alto valor en esta tesis.

Resumen

Los robots cuadrúpedos son un tipo de robot de servicio que se adaptan a cualquier terreno, sin embargo, establecer un modelo completo del mismo es una tarea muy complicada debido a su alto número de grados de libertad y a su no linealidad. Por lo que el aprendizaje profundo por refuerzo (DRL, por sus siglas en inglés) es una solución práctica para realizar tareas de control del sistema, ya que promete que los robots autónomos aprendan habilidades de comportamiento con una mínima intervención humana. Además, es posible diseñar algoritmos que permiten que un robot aprenda una tarea específica por sí mismo y llegue a una solución óptima sin estar programado específicamente para ello.

Aunque ya existen novedosos algoritmos de DRL para controlar dichos robots, estos se enfocan principalmente a optimizar los parámetros de los controladores analíticos basados en trayectorias predefinidas que deben seguir las patas del robot, por ejemplo, variando el tiempo que permanecen las patas en el suelo o el tiempo de transición entre los movimientos. De igual manera, se ha trabajado en maximizar la recompensa que obtiene el agente con el diseño de una función de recompensa especializada para cada tarea, es decir, para cada forma de desplazamiento del robot. No se ha reportado a detalle un análisis sobre cuáles son las variables del espacio observable que optimizan el proceso de aprendizaje para este tipo de robots en especial.

En este trabajo se plantea el diseño de un agente inteligente que aprende a controlar un robot cuadrúpedo por medio de estrategias de aprendizaje profundo por refuerzo, con el principal enfoque de identificar qué variables optimizan el proceso de aprendizaje, así como detallar qué estructuras de redes profundas realizan una mejor identificación del sistema y logren generar patrones de marcha o comportamientos específicos en un robot cuadrúpedo. Se presenta una revisión del estado del arte de los diversos métodos utilizados, el diseño y la construcción de un robot cuadrúpedo manufacturado en impresión 3D, la instrumentación del mismo y el desarrollo de sus archivos URDF para implementarlo en un ambiente de simulación, el cual permite realizar exploración de comportamientos logrando un aprendizaje controlado y seguro. Por último, se presenta una nueva forma de generar patrones de marcha en robots cuadrúpedos documentados en la literatura y se explora un tipo de controlador inteligente basado en redes neuronales.

Abstract

Quadruped robots are a type of service robot that can adapt to any terrain. However, establishing a complete model of them is a very complicated task due to their high number of degrees of freedom and their non-linearity. Therefore, deep reinforcement learning (DRL) is a practical solution for performing system control tasks, as it promises that autonomous robots will learn behavioral skills with minimal human intervention. In addition, it is possible to design algorithms that allow a robot to learn a specific task on its own and develop an optimal solution without being specifically programmed for it.

Although innovative DRL algorithms already exist to control such robots, they focus mainly on optimizing the parameters of analytical controllers based on predefined trajectories that the robot's legs must follow, for example, by varying the time the legs remain on the ground or the transition time between movements. Similarly, work has been done to maximize the reward obtained by the agent with the design of a specialized reward function for each task, that is, for each form of robot movement. No detailed analysis has been reported on which variables in the observable space optimize the learning process for this type of robot in particular.

This thesis proposes the design of an intelligent agent that learns to control a quadruped robot through deep reinforcement learning strategies, with the main focus on identifying the variables that optimize the learning process, as well as detailing which deep neuronal network structures best identify the system and generate specific gait patterns or behaviors in a quadruped robot. A review of the state of the art of the various methods used is presented, as well as the design and construction of a quadruped robot manufactured using 3D printing is presented, along with its instrumentation and the development of its URDF files for implementation in a simulation environment, which allows for behavior exploration, achieving controlled and safe learning. Finally, a new way of generating gait patterns in quadruped robots documented in the literature is presented, and an intelligent controller based on neural networks is explored.

Índice general

Índice de figuras	VI
Índice de tablas	XI
1. Introducción	1
1.1. Justificación	2
1.2. Planteamiento del Problema	3
1.3. Preguntas de Investigación	4
1.4. Objetivos	4
1.4.1. General	4
1.4.2. Específicos	5
1.5. Hipótesis	5
1.6. Alcances y Limitaciones	5
1.7. Descripción del Documento	6
2. Marco Teórico	8
2.1. Robots Cuadrúpedos	8
2.2. Control de Robots Cuadrúpedos	10
2.2.1. Control basado en modelo	11
2.2.2. Control sin modelo	12
2.3. Aprendizaje Profundo por Refuerzo	14

ÍNDICE GENERAL

2.3.1. Redes Neuronales Artificiales	14
2.3.2. Aprendizaje Profundo	16
2.3.3. Aprendizaje por Refuerzo	17
2.4. Simuladores	24
2.4.1. Nvidia Isaac Sim	25
2.4.2. Nvidia Isaac Lab	27
2.5. Robot Operating System (ROS)	29
2.5.1. ROS2	29
2.6. Conclusiones	32
3. Estado del Arte	33
3.1. Síntesis de los Estudios	34
3.1.1. Aprendizaje de locomoción en robots cuadrúpedos físicos	35
3.1.2. Aprendizaje de locomoción en robots cuadrúpedos simulados	37
3.2. Conclusiones	39
4. Metodología	40
4.1. Diseño Metodológico	40
4.2. Población	41
4.3. Muestra	42
4.4. Instrumentos de Medición	42
4.5. Conclusiones	44
5. Resultados	45
5.1. Diseño y construcción de Robot Cuadrúpedo	45
5.1.1. Diseño del robot cuadrúpedo	45
5.1.2. Instrumentación	49
5.1.3. Construcción del robot	51
5.2. Aprendizaje de controlador PID para balance con RNA	54
5.2.1. Plataforma Freenove Robot Dog	54

5.2.2. Data set del control PID de balance	57
5.2.3. Exploración Arquitecturas de Redes Neuronales	59
5.3. Transferencia de conocimiento	64
5.4. Patrones de marcha emergentes	66
5.5. Ambiente de Simulación	67
5.5.1. Archivo URDF	67
5.5.2. Configuración de Isaac Lab	69
5.6. Patrones de marcha derivados del equilibrio	76
5.6.1. Variación del ángulo Roll	79
5.6.2. Variación del ángulo Pitch	82
5.7. Aprendizaje por Refuerzo Profundo DRL	84
5.7.1. Aprendizaje de Política de incorporación estable	84
5.7.2. Aprendizaje de Política de balanceo del torso	92
Conclusiones y Trabajo Futuro	98

Índice de figuras

2.1. Estructura de robots cuadrúpedos: tipo reptil (izquierda), tipo mamífero (derecha).	9
2.2. Patrones de marcha más comunes en robots cuadrúpedos.	10
2.3. Estructura celular de la neurona.	15
2.4. Neurona artificial.	15
2.5. Interacción de un agente con su ambiente. Tomada de (Sutton & Barto, 2018).	18
2.6. Taxonomía de algoritmos de RL.	19
2.7. Flujo de trabajo de Nvidia Isaac Sim. Tomado de (Nvidia, 2025).	26
2.8. Nvidia Isaac Sim, entorno de simulación de robots con el concepto de digital twin. Tomado de (Nvidia, 2025).	26
2.9. Estructura de Isaac Lab. Tomado de (Nvidia, 2022).	27
2.10. Ecosistemas de aprendizaje con robots en Isaac Lab. Tomado de (Nvidia, 2022).	28
2.11. Modelo simple del grafo de ROS, donde se observa dos nodos, un publicador y suscriptor comunicando un mensaje a través de un tópico. Tomado de (OpenRobotics, 2025b)	30
2.12. Representación de un servicio en ROS. Tomado de (OpenRobotics, 2025b)	30
2.13. Representación de un servicio en ROS. Tomado de (OpenRobotics, 2025b)	31
2.14. Representación de una acción en ROS. Tomado de (OpenRobotics, 2025b)	32
4.1. Metodología propuesta.	41

4.2. Plataforma Robótica (Robot Caudrúpedo): a) Diseño CAD, b) Robot ensamblado.	42
4.3. Diagrama de instrumentación del sistema robótico cuadrúpedo.	43
4.4. Servomotores marca DCServo.	43
4.5. Sensor inercial para medir avance e inclinación del robot en el entorno. . .	44
4.6. Placa integrada Jetson NANO de la familia NVIDIA.	44
5.1. Diseño CAD de robot cuadrúpedo.	45
5.2. Diseño CAD de la parte superior de las patas del robot, la cual conecta toda la extremidad con el torso del robot.	46
5.3. Diseño CAD de la parte media de las patas del robot, donde (a) muestra el ensamble con los 2 GDL permitiendo al robot realizar movimientos de flexión y extensión en cada extremidad, en (b) se presenta el compartimiento interior para alojar cables derivados de las conexiones entre servomotores.	46
5.4. Diseño original vs diseño modificado para extremidades del robot.	47
5.5. Diseño de la pata para colocar el sensor de contacto.	47
5.6. Diseño de las cubiertas frontal (a) y posterior (b), donde se albergará un sensor de ultrasonido en conjunto con una cámara de 2 Mp y un sensor de voltaje junto con el interruptor general respectivamente.	48
5.7. Diseño de la tapa superior (a) y placa donde se fija la electrónica dentro del torso del robot (b).	48
5.8. Diseño CAD del robot cuadrúpedo completo, integrando los nuevos diseños y modificaciones.	49
5.9. Servomotores de arquitectura abierta.	49
5.10. Componentes para el robot: a) sensor fin de carrera, b) sensor ultrasónico y c) cámara 2Mp.	50
5.11. Placa basada en microcontrolador de 32 bits para el control de bajo nivel de los servomotores y adquisición de información de sensores ultrasónicos. .	50
5.12. Componentes: a) batería de litio y b) voltímetro amperímetro digital. . . .	51
5.13. Manufactura de piezas en PLA.	52
5.14. Extremidad inferior con sensor de límite de carrera.	52
5.15. Ensamble de pierna implementando sensor de contacto y servomotores. .	53

ÍNDICE DE FIGURAS

5.16. Electrónica del robot instalada dentro del torso.	53
5.17. Plataforma robótica implementada: a) vista frontal, b) vista trasera y c) vista lateral.	54
5.18. Plataforma Robótica Robot Dog Kit desarrollada por Freenove.	55
5.19. Componentes y piezas de acrílico que conforman el Robot Dog Kit.	55
5.20. Posicionamiento de cada motor a 90° para un correcto ensamble.	56
5.21. Herramienta “install manager tool” propia de Raspberry.	56
5.22. Aplicación para control remoto vía TCP/IP del robot.	56
5.23. Robot cuadrúpedo en plataforma de inclinación.	58
5.24. Movimientos realizados para extraer la base de datos, a) y b) movimiento en Pitch, c) y d) movimiento de Roll.	58
5.25. Archivos generados que conforman la base de datos utilizada para el entrenamiento de los modelos de redes neuronales.	59
5.26. Muestra de datos adquiridos realizando movimientos de inclinación en Pitch y Roll.	59
5.27. Gráfica de la función de pérdida MSE durante el entrenamiento para la red neuronal con 2 capas y 18 neuronas cada capa.	61
5.28. Gráfica de la métrica MAE para la red neuronal con 2 capas y 18 neuronas cada capa.	61
5.29. Gráfica de diferencia entre el valor real expresada en grados (°) y la predicción del modelo de 2 capas con 14 neuronas cada una.	62
5.30. Gráfica de la función de pérdida MSE durante el entrenamiento de la red con una capa de 18 neuronas.	62
5.31. Gráfica de la métrica MAE para la red neuronal con 1 capa y 18 neuronas.	63
5.32. Gráfica de diferencia entre el valor real expresada en grados (°) y la predicción del modelo de 1 capa con 18 neuronas cada una.	63
5.33. Eje de referencia inercial con el cual se generó la red neuronal artificial.	64
5.34. Ensamble del robot con los motores a 90°.	65
5.35. Valores de Roll, Pitch y Yaw con filtros aplicados y la inferencia de la red neuronal para estos valores.	65
5.36. Prueba de balance en la nueva plataforma utilizando la red neuronal desarrollada.	66

5.37. Patrones de marcha emergentes variando el giro en el eje x (Roll).	67
5.38. Archivo URDF de descripción del robot.	68
5.39. Representación de las piezas del robot en el simulador.	68
5.40. Cálculo de parámetros físicos de masa y momentos de inercia, señalados en rectángulos rojos, de cada pieza del robot.	69
5.41. Módulos principales del simulador Isaac lab.	70
5.42. Robot digitalizado y configurado para utilizarlo en simulación.	70
5.43. Configuraciones realizadas para una conversión del archivo URDF del robot cuadrúpedo al archivo USD que maneja el simulador.	71
5.44. Configuraciones realizadas en los parámetros de cada articulación.	72
5.45. Configuración de posiciones a 90° en cada articulación del robot.	72
5.46. Extremos de cada articulación sombreadas en verde, configuradas como pieza independiente de toda la extremidad.	73
5.47. Diseño y configuración de una plataforma que servirá como ambiente de aprendizaje para el robot cuadrúpedo, la cual puede variar su inclinación y posee un coeficiente de fricción dinámico y estático alto.	74
5.48. Instancia del robot suspendido en el aire dentro de la escena de simulación, sin interactuar con la fuerza de gravedad.	79
5.49. Movimiento en grados de las patas al variar el ángulo Roll linealmente.	80
5.50. Movimiento en grados de las patas al variar el ángulo Roll de manera no lineal.	81
5.51. Patrón de marcha "Paso", donde se observa en (a) cuando las dos extremidades del un solo lado están en el suelo, y (b) donde alterna las patas que están tocando el suelo.	81
5.52. Movimiento en grados de las patas al variar el ángulo pitch de manera lineal.	82
5.53. Movimiento en grados de las patas al variar el ángulo pitch de manera no lineal.	83
5.54. Patrón de marcha "Galope", donde se observa en (a) cuando las dos extremidades traseras están tocando el suelo y (b) donde alterna a las patas delanteras que están tocando el suelo.	83
5.55. Promedio de pesos ocupados en los experimentos.	86
5.56. Gráfica del promedio de pasos de simulación en cada episodio para el grupo 1.	87

ÍNDICE DE FIGURAS

5.57. Gráfica del promedio de pasos de simulación en cada episodio para el grupo 2.	87
5.58. Gráfica del promedio de pasos de simulación en cada episodio para el grupo 3.	88
5.59. Gráfica del promedio de recompensa por episodio para el grupo 1.	88
5.60. Gráfica del promedio de recompensa por episodio para el grupo 2.	89
5.61. Gráfica del promedio de recompensa por episodio para el grupo 3.	89
5.62. Gráfica de la desviación estándar de la política en cada experimento en el grupo 1.	90
5.63. Gráfica de la desviación estándar de la política en cada experimento en el grupo 2.	91
5.64. Gráfica de la desviación estándar de la política en cada experimento en el grupo 3.	91
5.65. Secuencia de pasos que realiza el robot con la política aprendida.	92
5.66. Escenario de entrenamiento con 1024 instancias de robot posicionados en un plataforma con diferente inclinación.	93
5.67. Promedio de pasos por episodio para cada una de las arquitecturas propuestas.	94
5.68. Promedio de recompensas total de cada una de las arquitecturas propuestas.	94
5.69. Desviación estándar de la política de cada arquitectura propuesta.	95
5.70. Tasa de aprendizaje para la política como para el estimador de valor.	96
5.71. Pérdida de la política para cada arquitectura utilizada en el experimento.	96
5.72. Pérdida del estimador de valor para cada arquitectura utilizada en el experimento.	97
5.73. Tarea de balance en una plataforma con diferentes inclinaciones.	97

Índice de tablas

2.1. Métodos de control de movimiento de robots cuadrúpedos.	11
5.1. Comparación de métricas MSE y MAE de redes neuronales con diferentes neuronas.	60
5.2. Configuración de parámetros para el algoritmo PPO	85
5.3. Configuración de parámetros para el algoritmo PPO	90

Introducción

Con el progreso de la ciencia, la tecnología y el desarrollo de la sociedad, la inteligencia artificial (IA) y la robótica se han vuelto temas de mayor interés. Por una parte, la IA es una ciencia que estudia las actividades cognitivas del ser humano, por ejemplo: pensar, razonar e interactuar con el ambiente que lo rodea; por otra parte, pretende dotar de esta inteligencia a elementos como sistemas computacionales (software) o a sistemas físicos (hardware) como los sistemas robóticos capaces de obtener información del medio que los rodea a través de sensores y así poder manipular, modificar o interactuar con su entorno a través de actuadores.

A medida que se desarrollan diversos tipos de robots para las exigencias tecnológicas que se enfrentan hoy en día, necesitamos que los robots completen tareas más complejas para el ser humano. El robot inteligente pertenece al producto experimental de la IA que puede realizar tareas complejas y se muestran más flexibles ante la incertidumbre. Una de las áreas que se ha destacado últimamente es la de robótica de servicio, la cual consta de emplear los robots no solo en ambientes industriales, sino también en labores cotidianas del ser humano que son desgastantes o peligrosas, por ejemplo, el transporte de materiales pesados o el reconocimiento de áreas potencialmente peligrosas.

Un robot cuadrúpedo es uno de los robots de campo más utilizados para transportar cosas automáticamente y reconocer terrenos desconocidos en la industria militar. En la industria de la construcción se utilizan estos robots para realizar recorridos en las construcciones e identificar potenciales peligros para el ser humano. Los robots de campo con patas se caracterizan por una buena movilidad accesible al entorno en comparación con los de ruedas. Por ejemplo, un robot cuadrúpedo puede conducirse en condiciones de terreno accidentado y puede superar obstáculos.

El control de un robot cuadrúpedo es una tarea que requiere el desarrollo de un modelo matemático preciso, ya que se debe conocer el comportamiento dinámico del robot. Para lograr esto se requiere la identificación de todos los parámetros físicos del robot como lo

son: pesos y longitudes de eslabones, inercias, centros de gravedad y fuerzas de fricción que actúan en sus articulaciones al contacto con el suelo. Detectar o extraer con precisión estos parámetros del modelo físico es una tarea laboriosa y tiende a presentar errores, además de tomar en cuenta la dificultad para medir directamente variables como las fricciones internas o la distribución exacta del peso.

A pesar de estas dificultades, el modelo matemático es esencial para derivar ecuaciones de control que permitan manipular las variables del sistema y lograr el movimiento deseado, principalmente los ángulos y velocidades de los motores. Los robots cuadrúpedos cuentan normalmente con 12 grados de libertad (GDL), esto representa una complejidad considerable debido a sus múltiples articulaciones y restricciones cinemáticas impuestas por la geometría del robot, lo que dificulta aún más el desarrollo de controladores precisos.

Aunque hasta ahora se han desarrollado muchos algoritmos de control basados en aprendizaje, la mayoría de ellos toman los algoritmos como una herramienta opcional para ayudar a los algoritmos de control clásicos a sintonizar sus parámetros ya que dependen en gran medida de las ecuaciones cinemáticas.

Las técnicas de Machine Learning (ML) como los algoritmos de aprendizaje por refuerzo y el aprendizaje profundo, (RL y DL respectivamente, por sus siglas en inglés), han sido adoptados cada vez más por la comunidad robótica en los últimos años para controlar robots complejos y proporcionar políticas integrales de extremo a extremo y guiados por datos desde la percepción hasta el control (Cheng et al., 2019), logrando optimizar el proceso de diseño de un controlador específico para los robots cuadrúpedos.

1.1. Justificación

Una de las aplicaciones con mayor interés es la de robótica de servicio, en especial los robots cuadrúpedos que pretenden en un futuro apoyar al ser humano en diversas tareas cotidianas monótonas y de riesgo. Se sabe que el diseño de controladores de locomoción para robots cuadrúpedos es de alta dificultad y requieren de conocimiento experto para poder desarrollarlos, además debe ser flexible y adaptarse a los diversos entornos en los cuales serán puestos en funcionamiento.

Los métodos actuales de vanguardia suelen emplear un enfoque en cadena, que consta de componentes como la estimación de estado (Leziart et al., 2021), la programación de contactos en las extremidades (H. Kim et al., 2005), la planificación de la colocación de los pies (Wen et al., 2012), el control del espacio operativo (Kimura et al., 2007), el control predictivo del modelo (MPC) (Saraf et al., 2021) (que debe diseñarse con conocimientos profesionales) y un modelo dinámico preciso del robot, que es difícil de obtener.

El desarrollo de controladores de locomoción en robot cuadrúpedos ha sido un área en constante crecimiento. Aunque se han diseñado controladores que aprenden patrones de locomoción exhibidas por sus análogos biológicos, estos se basan en la idea de tener un

previo reconocimiento del sistema y sus parámetros físicos, para posteriormente, dotar al sistema de trayectorias predefinidas o puntos en el espacio que deberán seguir las patas para realizar el correcto patrón de marcha. En este caso, se utiliza el aprendizaje por refuerzo en conjunto con la técnica llamada moldeo de recompensa para lograr optimizar la velocidad de transición o parámetros, como el tiempo que pasa cada extremidad pegado al suelo o en el aire, esto limita a explorar solo la gamma de características que contengan las trayectorias predefinidas.

Han surgido otros métodos que aprenden el patrón de marcha sin algún tipo de trayectoria definida, pero previamente se han realizado identificaciones del sistema y se utiliza la cinemática inversa del mecanismo para obtener las trayectorias. Por otra parte, agregando una técnica conocida como *aleatorización*, aumenta el tiempo de aprendizaje y la cantidad de datos que se ocupan para entrenar al modelo. Una idea importante en aprendizaje por refuerzo es la capacidad del modelo para explorar el ambiente a través de sus acciones. En la actualidad, este reto se afronta tomando una acción aleatoria o sumando ruido a la acción estimada por el modelo. Detrás de estos métodos hace falta un análisis de las variables que el robot toma en cuenta para lograr algún comportamiento óptimo.

Por lo anterior, surge la necesidad de conocer si es posible abordar el problema de aprendizaje de locomoción con la perspectiva de no contar con información de patrones de marcha ni trayectorias predefinidas, recurriendo al moldeo de recompensa y al análisis de las variables que aportan información relevante para lograr comportamientos de locomoción, además de proponer una metodología que oriente la exploración del ambiente por parte del agente hacia la meta de maximizar la recompensa.

1.2. Planteamiento del Problema

Muchos trabajos de investigación de control de locomoción de robots cuadrúpedos complementados con IA emplean el aprendizaje por refuerzo, ya que el RL puede hacer que un robot se enfrente a un entorno desconocido e irregular interactuando constantemente con él. Aunque casi todos los trabajos se han efectuado en simulaciones por computadora, existe una gran brecha entre los experimentos in situ o el RL online, es decir, en cuanto el robot está interactuando con el entorno se genera un aprendizaje al mismo tiempo que se efectúa la salida de una acción.

Debido a lo costoso que pueden ser estos sistemas, el generar acciones aleatorias puede incurrir en daños para la mecánica o electrónica del robot. Se ha implementado una metodología llamada *Sim-to-real* que consta de transferir conocimiento adquirido en simulación hacia el robot real, aleatorizar ciertos parámetros dinámicos del robot o introducir ruido a las observaciones de los estados ha dado buenos resultados, pero extiende el tiempo de entrenamiento por la cantidad de datos necesarios para el entrenamiento y esto contribuye a que las acciones generadas por el robot sean estocásticas y sin un método que oriente la toma de decisiones sobre qué acciones obtienen una mejor recompensa.

Por otra parte, la gran mayoría se enfoca en el diseño de la recompensa que el agente tiene para cada acción en específico y se centran en optimizar trayectorias y puntos de apoyo de las patas, lo cual limita el descubrimiento de patrones de marcha por sí solo. Muchos trabajos plantean un enfoque de extremo a extremo, que significa el contemplar todas las entradas que el agente puede percibir y tratar de mapearlo a los estados y acciones que este realice, pasando desapercibido el análisis sobre qué entradas realmente son las que aportan información de relevancia, es decir, qué características del espacio observable optimizan el proceso de aprendizaje del agente. Además, la exploración del ambiente se realiza con aleatorizar la acción o sumarle algún tipo de ruido a la salida del modelo, aunque estos métodos han mostrado buenos resultados es necesario un análisis de las variables que son indispensables para lograr patrones de marcha o comportamientos de balance en un robot cuadrúpedo.

1.3. Preguntas de Investigación

- ¿Qué algoritmos de aprendizaje profundo por refuerzo existen que realizan el control de un robot cuadrúpedo desde un enfoque extremo a extremo?
- ¿Qué características debe tener un algoritmo de aprendizaje profundo por refuerzo para realizar tareas de control de un robot cuadrúpedo del cual se desconocen sus características de peso, dimensiones y su modelo dinámico?
- ¿Qué características debe tener la función de recompensa en un algoritmo de aprendizaje profundo por refuerzo que realice el aprendizaje de marcha y equilibrio de los robots cuadrúpedos?
- ¿Cuáles son las variables del espacio observable en un robot cuadrúpedo que aportan información relevante para el aprendizaje de marcha de un robot cuadrúpedo?

1.4. Objetivos

1.4.1. General

Diseñar e implementar un sistema inteligente que aprenda a generar movimientos de marcha en un robot cuadrúpedo por medio de técnicas de aprendizaje por refuerzo profundo identificando las características de una función de recompensa y variables del espacio observable que aporten información relevante para el aprendizaje del robot.

1.4.2. Específicos

- Identificar los algoritmos de aprendizaje profundo por refuerzo que realicen el control de un robot cuadrúpedo basado en un enfoque extremo a extremo.
- Diseñar una función de recompensa que realice el aprendizaje en los robots cuadrúpedos al aplicar un algoritmo de aprendizaje profundo por refuerzo.
- Realizar un análisis sobre qué variables aportan información relevante y optimizan el aprendizaje.
- Diseñar e implementar un algoritmo de aprendizaje profundo por refuerzo que sea capaz de controlar un robot cuadrúpedo del cual se desconoce su modelo y parámetros como pesos y longitudes.

1.5. Hipótesis

Un agente basado en redes neuronales profundas entrenado con técnicas de aprendizaje por refuerzo, en un ambiente de simulación variable, aprende a controlar un robot cuadrúpedo en tareas de marcha o balance sin necesidad de contar con el modelo matemático ni con el conocimiento exacto de las variables físicas del robot.

Además, el análisis de diferentes escenarios y experimentos considerando diversas combinaciones de variables en el aprendizaje podría derivar una conclusión sobre cuales son primordiales para generar el comportamiento de marcha o balance y así sintetizar los modelos inteligentes de control.

1.6. Alcances y Limitaciones

Uno de los alcances de este estudio es que, a través de una plataforma robótica cuadrúpea comercial, se demuestre que partiendo del balance y desbalance del torso del robot se pueden generar de manera natural patrones de marcha sin requerir trayectorias preestablecidas para cada una de las patas del robot. Esto nos hace pensar en que la locomoción puede surgir como resultado de mantener el equilibrio dinámico del cuerpo, lo cual representa un enfoque innovador a los métodos convencionales basados en planificaciones cinemáticas o dinámicas detalladas.

Este trabajo de investigación se enfoca principalmente en generar que un agente inteligente aprenda a controlar un robot cuadrúpedo, especialmente en tareas de balance y locomoción, capaz de generar patrones de marcha en un ambiente de simulación sin contar con un modelo matemático o trayectorias de marcha preestablecidos. Esto implica generar un ambiente de entrenamiento para aplicar técnicas de aprendizaje por refuerzo

(RL) en conjunto con aprendizaje profundo (DL), donde se experimentará específicamente con algoritmos basados en políticas de gradiente (PPO). De igual manera, se contempla un análisis sobre qué arquitecturas de redes profundas tienen mejor desempeño, por lo cual, en este trabajo solo se contemplan redes neuronales profundas perceptrón multicapa en las que solo se variará la densidad de capas y neuronas.

Para el robot físico de este trabajo se utilizó un diseño de licencia libre y se realizaron modificaciones antes de la manufactura en impresión 3D, por lo cual, el sistema se considera de bajo costo y una herramienta funcional para poder ejercer experimentos de aprendizaje directamente en el robot.

Algunas limitantes del sistema robótico son el sistema de adquisición de datos y procesamiento, ya que, se utilizó la tarjeta de desarrollo Nvidia Jetson Nano que fue diseñada para realizar experimentos de IA, cuenta con una GPU basada en arquitectura NVIDIA Maxwell de 128 núcleos y un CPU ARM A57 de cuatro núcleos. Al ser una tarjeta de desarrollo existen problemas de compatibilidad en las paqueterías instaladas. Por otra parte, se requiere un sistema de cómputo potente para realizar los entrenamientos en escenarios diversos, esto conlleva mucha carga computacional y al contar con una estación de trabajo Desktop con procesador Intel core i9 generación 12, con 64Gb en RAM y una GPU Nvidia RTX 4070 de 12 Gb de Vram, esto presenta una limitante en tiempo de entrenamiento y capacidades reducidas de generar largos entrenamientos con diversas instancias en paralelo.

1.7. Descripción del Documento

Este trabajo está dividido de la siguiente manera:

El capítulo 2 describe la información teórica a utilizar para el desarrollo de este trabajo, es decir, los conceptos conforme a los agentes inteligentes, redes neuronales, definiciones de los tipos de aprendizaje y la combinación de ellos como lo es el aprendizaje por refuerzo profundo.

En el capítulo 3 se presenta la revisión sistemática del estado del arte, el protocolo, su formación y la ejecución de esta. Además, se describen los criterios para la extracción de datos y la síntesis de los estudios.

En el capítulo 4 se describe la metodología propuesta para el desarrollo de este trabajo, se expone la población del estudio, la muestra y los instrumentos de medición.

En el capítulo 5 se presentan los resultados obtenidos durante el desarrollo de esta tesis, así como la discusión de los mismos.

Por último, el capítulo 6 consiste en las conclusiones generales obtenidas. Se menciona el trabajo a futuro y las posibles mejoras que pueden realizarse.

En el apartado bibliográfico se encuentran los datos de la literatura que sirvió de apoyo para el desarrollo de esta tesis.

Marco Teórico

En esta sección se presentan las teorías y métodos que fundamentan y anteceden a este trabajo de tesis. El control de movimiento de los robots cuadrúpedos requiere definir en un principio la estructura mecánica o la morfología de dichos robots, por lo que se abordarán las ventajas con las que cuentan y las topologías que se han estudiado a través de los años. Posteriormente, se exponen brevemente los métodos de control clásicos y las características que estos poseen, así como las nuevas metodologías que no requieren un modelo matemático del sistema y que prometen controlar un robot cuadrúpedo reduciendo la complejidad que esto representa, al mismo tiempo explorando el área de aprendizaje máquina. Por último, se detallan los métodos de control basados en datos como el aprendizaje profundo por refuerzo (DRL) y los métodos actuales que ofrecen una alternativa prometedora para controlar sistemas de alta dimensionalidad.

2.1. Robots Cuadrúpedos

Aunque los vehículos de ruedas se han utilizado por varios años, es necesario que el pavimento cuente con características especiales para que estos funcionen de manera correcta, como la rugosidad o que sea una superficie no accidentada, por lo que es necesario encontrar un medio de transporte que pueda desplazarse sobre superficies accidentadas.

Por la evolución a través de miles de millones de años, el resultado de la selección natural sobre el modo de movimiento de los animales terrestres ha demostrado la excelente adaptabilidad, movilidad y capacidad de carga en terrenos accidentados de los mamíferos cuadrúpedos.

Desde hace 100 años, los investigadores han estudiado los robots cuadrúpedos que toman a los mamíferos como modelo de inspiración. Se espera que el robot cuadrúpedo pueda influir en campos como el transporte de material, la detección medioambiental, la

ayuda en caso de catástrofe, etc. El objetivo se centra en mejorar la capacidad de los robots cuadrúpedos para cargar, manipular y adaptarse, formando una nueva rama en el campo de la robótica.

(Todd, 1985) menciona algunas ventajas de los robots cuadrúpedos, las cuales son:

- Las patas pueden superar obstáculos y subir y bajar escaleras.
- La locomoción con patas puede incluso llevar un vehículo sobre terrenos muy accidentados.
- Un vehículo con patas puede transitar suavemente en terrenos accidentados variando la longitud efectiva de sus patas para adaptarse a las ondulaciones del suelo.
- En terreno blando, una rueda siempre sale de un bache que ella misma ha creado, lo que supone un desperdicio de energía. En un caso extremo, la rueda puede profundizar hasta que el vehículo se detenga.
- Las patas dañan menos el suelo que las orugas y otros tipos de ruedas.

La estructura topológica del robot cuadrúpedo se refiere a la configuración de la pata y la articulación de esta. Una estructura topológica adecuada es la base para que el robot consiga un movimiento estable y flexible en un entorno complejo. Durante el desarrollo del robot cuadrúpedo, la configuración de la pata puede dividirse en estructura de tipo mamífero y de tipo reptil (Todd, 2013) como se presenta en la figura 2.1.

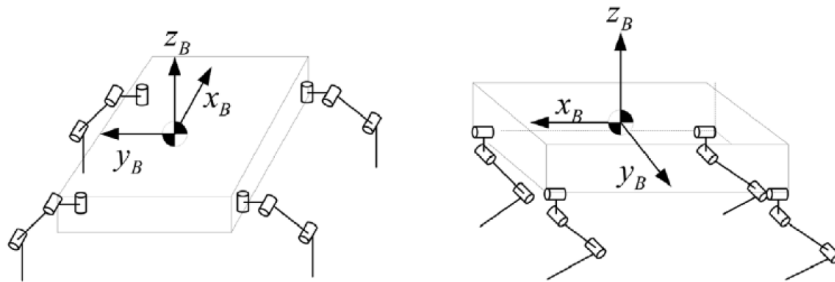


Figura 2.1: Estructura de robots cuadrúpedos: tipo reptil (izquierda), tipo mamífero (derecha).

Se han obtenido numerosos logros notables en este campo. Por ejemplo, los robots cuadrúpedos equipados con actuadores hidráulicos, como *Big Dog* (Raibert et al., 2008) de Boston Dynamics y *HyQ2Max* (Semini et al., 2017) del MIT, aprovechan el combustible convencional con alta densidad de energía para emitir grandes fuerzas en cada articulación. Otra importante familia de robots cuadrúpedos están equipados con actuadores eléctricos que recientemente han mostrado grandes perspectivas para realizar maniobras dinámicas y ágiles junto con una gran mitigación de impactos, un control de fuerza excepcional y capacidades de control de posición, como *ANYmal* en ETH (Hutter et al., 2016) que presenta

un actuador bio inspirado que lo hace robusto frente a impactos, y varias generaciones de *Cheetah* del MIT, (Bosworth et al., 2015) y (Katz et al., 2019), un robot cuadrúpedo diseñado como plataforma de investigación optimizada principalmente para la velocidad aunque no se ha evaluado a fondo en lo que respecta a la duración de la batería, la capacidad de giro, la robustez mecánica y la aplicabilidad en exteriores.

2.2. Control de Robots Cuadrúpedos

La esencia del método de control de movimiento es controlar la marcha del robot para que siga una trayectoria determinada. El método de control de movimiento de los robots cuadrúpedos es el que determina directamente el movimiento del robot, así como su flexibilidad, estabilidad y adaptabilidad al terreno (Chai et al., 2021). Los robots con patas se mueven levantando y pisando cada pata en secuencia y este comportamiento se denomina *marcha*.

La marcha de un robot con patas suele describirse con tres términos técnicos: *zancada*, *factor de trabajo* y *fase relativa* (Alexander, 1984). La zancada es la distancia del movimiento del torso durante un ciclo de marcha. El factor de trabajo es la relación entre la fase de apoyo de una pierna y el ciclo de marcha. La fase relativa de una pierna se define como la relación entre el retraso y el periodo completo.

Según el polígono de apoyo, la marcha del robot cuadrúpedo puede dividirse en marcha estática y marcha dinámica. La marcha estática se refiere principalmente al avance sencillo y más lento, mientras que la marcha dinámica puede dividirse en trote, paso, salto, pronación, galope, etc. Entre ellos, el trote y el galope son los más utilizados por los mamíferos cuadrúpedos. La figura 2.2 presenta los gráficos de marcha más comunes para los robots cuadrúpedos, donde las fases de apoyo se dibujan como barras negras, mientras que las fases de balanceo se dibujan como barras en blanco.

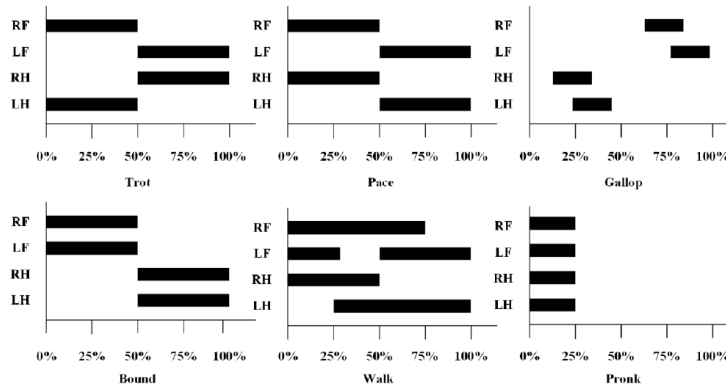


Figura 2.2: Patrones de marcha más comunes en robots cuadrúpedos.

El método de control de movimiento de los robots cuadrúpedos es un gran reto en el

campo de la robótica, ya que el robot cuadrúpedo es un sistema no lineal de alta dimensión y variable en el tiempo que interactúa con varios entornos (Bellicoso et al., 2016). Se han desarrollado muchos métodos para el control del movimiento, que pueden agruparse en dos categorías: métodos de control basados en modelos y métodos de control sin modelos, como se observa en el cuadro 2.1.

Tabla 2.1: Métodos de control de movimiento de robots cuadrúpedos.

Métodos de control basados en modelos	Métodos de control sin modelo
Punto de Momento Cero (ZMP) Péndulo invertido con carga de muelle (SLIP) Control de modelo virtual (VMC) Dinámica inversa (ID) Control predictivo de modelos (MPC) Control de todo el cuerpo (WBC)	Generador central de patrones (CPG) Control inteligente (CI)

2.2.1. Control basado en modelo

Los métodos basados en modelos se derivan de la teoría de control, donde el requisito principal es conocer de forma exacta el mecanismo y sus propiedades físicas, como parámetros de masa, inercia, longitudes, entre otros, lo que conlleva a un arduo trabajo de análisis matemático y complejidad en las ecuaciones. El modelo matemático del sistema no siempre es preciso, ya que existen algunos parámetros intrínsecos del sistema que son difíciles de obtener y requieren una serie de técnicas para su obtención como lo es la *parametrización*.

Método de control basado en el criterio de estabilidad

La condición estable de un robot cuadrúpedo es que la proyección del Centro de Masa (CM) sobre el suelo se encuentre dentro del polígono formado por los puntos de apoyo, por lo que se puede indicar la estabilidad del robot por la relación entre el CM y los pies de apoyo. Este tipo de norma se denomina *criterio de estabilidad*, (J. Chen et al., 2022).

La marcha estática de un robot cuadrúpedo incluye tres pasos: elegir una pierna de balanceo, mover el CM a una nueva posición y balancear la pierna de balanceo hasta un punto de apoyo deseado, (J. Chen et al., 2022). Para garantizar el equilibrio durante la marcha, el robot debe mantener siempre la proyección del CM en el polígono de apoyo. El método de control basado en el criterio de estabilidad requiere una planificación detallada de las extremidades y la trayectoria del CM mejora la estabilidad del robot sacrificando su rendimiento dinámico. Este método se utiliza sobre todo para el control estático de la marcha del robot.

Modelo SLIP

El Péndulo Invertido con Resorte (SLIP, por sus siglas en inglés *Spring-Loaded Inverted Pendulum*) es un modelo dinámico utilizado como una representación simplificada para capturar la dinámica básica de la locomoción con patas y utiliza un mecanismo de recuperación de energía similar al del péndulo invertido para describir los cambios de energía cinética y potencial. El modelo SLIP consiste en una masa conectada a una pata sin masa con un resorte unido en su articulación. Cuando la pata está en contacto con el suelo, el resorte se comprime almacenando energía potencial. A medida que la pata se separa del suelo, la energía almacenada en el resorte se libera, impulsando la masa hacia arriba. El ciclo se repite cuando la pata impacta nuevamente en el suelo. Una zancada en el modelo SLIP consta de las fases de apoyo y vuelo. En la fase de vuelo, la pierna elástica ajusta el ángulo de aterrizaje para oscilar hasta la posición deseada y el cuerpo avanza en la fase de apoyo comprimiendo y descomprimiendo el resorte. El modelo SLIP proporciona una representación simple pero efectiva de las dinámicas involucradas. Al aplicar estrategias de control al modelo SLIP, se pueden optimizar parámetros como la rigidez de la pata, la longitud de la pata y el sincronismo para lograr una locomoción estable (M. Li et al., 2014), (Takemura et al., 2005).

2.2.2. Control sin modelo

En el método de control sin modelo, no hay un modelo cinemático ni dinámico del robot implicado en el control. Por su sencillez y generalidad, el método de control sin modelo se ha convertido en un método atractivo para el control de locomoción de robots cuadrúpedos. Tiene la ventaja de adaptarse a sistemas complejos y no lineales donde es difícil o costoso obtener un modelo matemático preciso. Sin embargo, también puede requerir grandes cantidades de datos y tiempo de entrenamiento para lograr un buen rendimiento y puede ser más susceptible a problemas de generalización en diferentes situaciones.

En el control de robots sin modelo, se utilizan métodos basados en neurociencia y algoritmos de aprendizaje automático, como redes neuronales, algoritmos genéticos o métodos de optimización, para aprender y ajustar los parámetros de control en función de los datos recopilados durante la interacción con el entorno. Estos algoritmos permiten al robot aprender a través de la experiencia y mejorar su rendimiento a medida que se realiza el control en tiempo real.

Una de las técnicas más utilizadas en el control de robots sin modelo es el RL (por sus siglas en inglés, *Reinforcement Learning*). En el RL, el robot interactúa con el entorno, recibe retroalimentación en forma de recompensas o penalizaciones, y ajusta sus acciones para maximizar la recompensa acumulada a lo largo del tiempo. De esta manera, el robot puede aprender políticas de control óptimas sin la necesidad de un modelo preciso del sistema.

Generador Central de Patrones

En Neurociencia se estudian los movimientos repetitivos (cíclicos) que subyacen a comportamientos cruciales como la locomoción (caminar, nadar, arrastrarse y volar), así como a funciones vegetativas esenciales como la respiración, el latido del corazón en algunos invertebrados, la masticación y los movimientos intestinales. Aunque tanto la retroalimentación sensorial como la dinámica neuro-muscular desempeñan un papel importante en la conformación de la producción motora rítmica, los patrones básicos de actividad rítmica son generados por circuitos centrales llamados generadores de patrones centrales (CPG). Cuando los CPGs se activan adecuadamente, pueden producir actividad rítmica en la red en ausencia de señales externas de tiempo, es decir, sin retroalimentación sensorial rítmica o activación rítmica por parte de las neuronas descendentes (Bucher, 2009).

El método de control basado en la CPG es un método de control biónico. La idea del CPG es imitar la estructura nerviosa central de los organismos inferiores para construir múltiples osciladores periódicos y generar un movimiento rítmico coordinando la conexión de los osciladores. El controlador CPG no sólo puede coordinar el movimiento de las piernas, sino también completar la transición de la marcha mediante simples señales de control (Ijspeert, 2014). Además, el controlador puede utilizar un oscilador neural (Endo et al., 2008) con función de aprendizaje de refuerzo para obtener parámetros en línea y optimizar cada parámetro (Park & Kim, 2013).

Control Inteligente

Es posible realizar el control de movimiento de un robot sin establecer el modelo, y este método se ha presentado últimamente en conjunto con técnicas de aprendizaje máquina dando origen a los controladores basados en datos o comúnmente llamados *controladores extremo a extremo*. Algunos algoritmos de IA se utilizan en la implementación de este tipo de controladores para realizar tareas de locomoción en robots cuadrúpedos, como el control difuso, controladores neuronales y algoritmos genéticos.

En la mayoría de los casos, los algoritmos de IA son utilizados como herramientas de optimización para los controladores convencionales, donde el principal objetivo es encontrar parámetros del controlador que logren un mejor desempeño. Se busca lograr un comportamiento adaptativo en estos y que puedan enfrentar diversos escenarios sin tener que configurar los parámetros para cada uno. Un ejemplo claro de esto sería el control difuso, donde la parte de lógica difusa puede gestionar el acoplamiento entre múltiples entradas y salidas de un sistema para ejecutar una tarea, seleccionando parámetros que se adapten mejor a la situación que enfrenta el sistema (Palmer & Orin, 2010).

Algunos enfoques de control inteligente abordan la idea de controladores basados en datos, los cuales evitan la ardua tarea de contar con el modelo dinámico del robot para poder controlarlo, apoyándose de métodos como el aprendizaje profundo y redes neuronales artificiales (RNA) para lograr aproximar el comportamiento del sistema ante la interacción con su entorno. Por ejemplo, en (Y. Chen et al., 2015) realizan el entrenamiento de una red neuronal para lograr que un robot cuadrúpedo se incorpore después de una caída y

en (Hwangbo et al., 2019) adoptaron un método donde se entrena una red neuronal de ejecución de acciones utilizando directamente los datos reales del actuador de la articulación en el que lograron que el robot AnYmal (Hutter et al., 2016) fuera más rápido que antes en simulación.

Aunque existen enfoques muy diversos, esencialmente se basan en un conocimiento previo del dominio del problema y muchos en demostraciones como los datos de captura de movimiento. El uso de un marco de aprendizaje como el que ofrece RL ha ido desvelando mejores resultados y se han logrado notables avances dentro del área de control de robots cuadrúpedos con controladores extremo a extremo.

2.3. Aprendizaje Profundo por Refuerzo

Una rama del aprendizaje máquina que está siendo explotada últimamente es el aprendizaje por refuerzo profundo (DRL, por sus siglas en inglés), ya que es la combinación de dos tipos de aprendizaje: *no supervisado* que describe los elementos participantes y la relación entre ellos, y *aprendizaje por refuerzo* que permite la tarea de aprendizaje por medio de prueba y error. Esta combinación de técnicas es la que ha demostrado que un sistema robótico pueda controlarse por medio de procesos de aprendizaje de sus parámetros intrínsecos e incluso de las acciones necesarias para lograr un objetivo, es decir, moverse en su entorno hacia una posición en específico.

2.3.1. Redes Neuronales Artificiales

Las redes neuronales artificiales (RNA) son técnicas populares de aprendizaje automático que simulan el mecanismo de aprendizaje de los organismos biológicos. El sistema nervioso humano contiene células, que se denominan neuronas. Las neuronas están conectadas entre sí mediante axones y dendritas, y las regiones de conexión entre axones y dendritas se denominan sinapsis (Daly et al., 2012).

El cerebro de los seres humanos, formado por miles de millones de neuronas interconectadas, tiene la capacidad de realizar tareas complejas y exigentes desde el punto de vista informático, como el reconocimiento de caras, la planificación del movimiento corporal y el control de las actividades musculares. La figura 2.3 presenta la estructura celular de una neurona del ser humano.

Inspirada en la estructura de la neurona, la RNA se desarrolló para emular la capacidad de aprendizaje del sistema de neuronas biológicas. El concepto de RNA fue investigado inicialmente por McCulloch y Pitts en la década de 1940 (Kober et al., 2013). El modelo matemático de la RNA consiste en tres capas: la capa de entrada, la capa oculta y la capa de salida, que son de estructura computacional paralela simple, pero con una capacidad de aprendizaje y poder computacional para predecir patrones dinámicos no lineales.

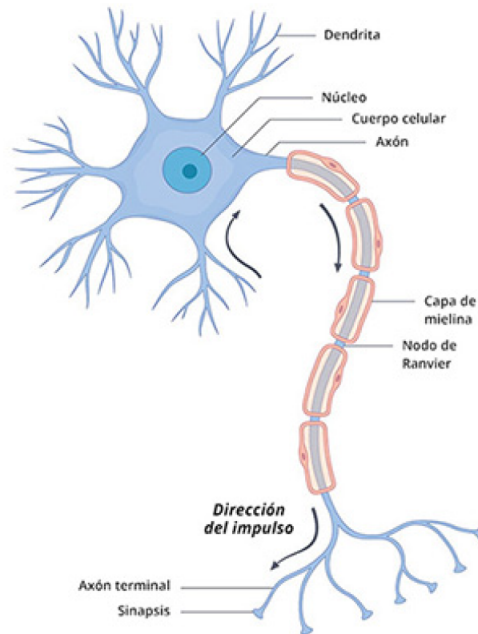


Figura 2.3: Estructura celular de la neurona.

En la figura 2.4 se observa una neurona artificial de una sola entrada, la entrada escalar p se multiplica por el peso escalar w para formar wp y se envía al bloque de suma, así sucesivamente. La salida del sumador pasa a una función de transferencia, que produce la salida escalar de la neurona artificial.

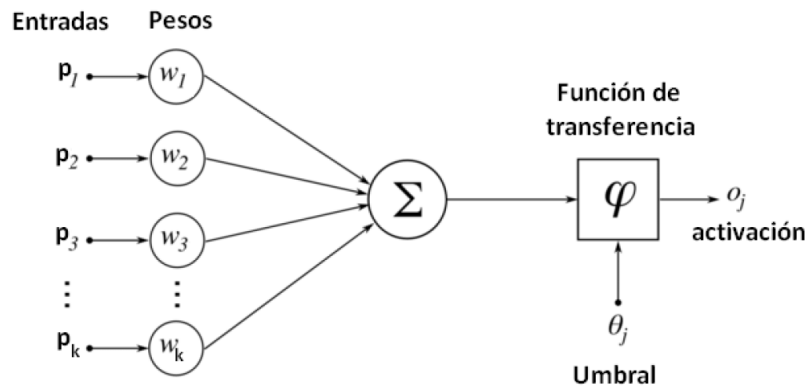


Figura 2.4: Neurona artificial.

En el campo de la IA, existen varias funciones de activación utilizadas en las RNA. A continuación, se mencionan algunas de las funciones de activación más comunes.

- **Función Sigmoide:** toma una entrada y la mapea a un valor entre 0 y 1. Es una función no lineal y se utiliza principalmente en problemas de clasificación binaria,

donde la salida puede interpretarse como una probabilidad. Sin embargo, en capas ocultas de redes neuronales profundas, puede tener problemas de desvanecimiento del gradiente.

- *Función ReLU (Rectified Linear Unit)*: es una función no lineal que mapea las entradas negativas a cero y mantiene las entradas positivas sin cambios. Es computacionalmente eficiente y ha sido ampliamente utilizada en redes neuronales convolucionales (CNN) debido a su capacidad para aprender características discriminativas.
- *Función Tangente Hiperbólica (Tanh)*: es similar a la función sigmoide, pero su rango es de -1 a 1. Al igual que la función sigmoide, es útil en problemas de clasificación binaria y también en casos donde se requiere una salida centrada en cero.
- *Función Softmax*: se utiliza comúnmente en la capa de salida de una red neuronal para problemas de clasificación multiclase. Toma un vector de entradas y produce una distribución de probabilidad sobre las clases, donde la suma de todas las salidas es igual a 1.

Estas son solo algunas de las funciones de activación más utilizadas en IA que han dado mejores resultados (Rasamoelina et al., 2020). Existen otras funciones que también se utilizan en diferentes contextos y pueden adaptarse a diferentes tipos de problemas y arquitecturas de red. La elección de la función de activación depende del tipo de tarea que se esté abordando y la naturaleza de los datos.

2.3.2. Aprendizaje Profundo

El aprendizaje profundo (DL, por sus siglas en inglés *Deep Learning*) consta de múltiples capas ocultas de neuronas entre la entrada y la salida de una red neuronal. Estas capas adicionales permiten la extracción jerárquica de características a medida que los datos pasan a través de la red. A diferencia de las redes neuronales tradicionales, las redes profundas pueden aprender representaciones de alto nivel a partir de características más básicas, lo que mejora la capacidad de modelado y la capacidad de generalización del sistema.

Una de las características distintivas del DL es su capacidad para aprender de manera no supervisada. Esto significa que las redes neuronales pueden extraer patrones y características, es decir, la red neuronal realiza un proceso para decidir qué características de un conjunto de datos pueden utilizarse como indicadores para etiquetar esos datos de forma fiable (Patterson & Gibson, 2017). Se beneficia del autoaprendizaje, donde las redes neuronales pueden aprender características representativas sin supervisión externa.

En DL se ha desarrollado diversas arquitecturas de redes neuronales que se utilizan de manera específica y complementaria para abordar diferentes tipos de problemas en el campo de la IA. Por ejemplo, las Redes Neuronales Convolucionales (CNN, por sus siglas en

inglés) (Z. Li et al., 2022) se emplean para el procesamiento de datos estructurados, como imágenes, gracias a su capacidad de extraer características relevantes a través de capas de convolución y pooling. Por otro lado, las Redes Neuronales Recurrentes (RNN, por sus siglas en inglés) (Y. Yu et al., 2019) son adecuadas para el procesamiento de secuencias de datos, como texto, y pueden ser utilizadas en combinación con las Redes Neuronales Transformer (Vaswani et al., 2017), que se basan en mecanismos de atención para mejorar la capacidad de capturar dependencias temporales y contextuales en el procesamiento del lenguaje natural. Las Redes Neuronales Generativas (GAN, por sus siglas en inglés) (Gonog & Zhou, 2019) pueden ser empleadas para generar datos sintéticos que luego pueden ser utilizados como parte del conjunto de entrenamiento en otras arquitecturas.

Estas son solo algunas de las clasificaciones más comunes en el campo de aprendizaje profundo. Es importante tener en cuenta que estas clasificaciones no son mutuamente excluyentes y que algunas arquitecturas pueden combinar características de diferentes tipos de redes neuronales para abordar problemas específicos. DL está en constante evolución, surgen nuevas arquitecturas y enfoques de aprendizaje con regularidad.

Para entrenar los pesos ideales para cada capa en el aprendizaje profundo, el error de la última capa se propaga hacia atrás, estas capas se actualizan tomando el gradiente de la función de pérdida para que los pesos puedan ser actualizados en una dirección tal que la pérdida pueda ser minimizada. Las pérdidas pueden minimizarse moviéndose hacia un mínimo en la función de pérdida y este mínimo se puede alcanzar moviéndose en la dirección de los gradientes calculados de las funciones de pérdida (Sewak, 2019). Esta función de pérdida toma los valores reales y los predichos como entrada para obtener la pérdida.

Por otra parte, la redes neuronales profundas al contar con más de una capa oculta de neuronas surgen algunos problemas para realizar un entrenamiento efectivo, ya que existe un fenómeno llamado *desvanecimiento del gradiente*, el cual al propagar el error a las capas iniciales requiere una larga cadena de multiplicaciones de gradiente y el uso de una función de activación como la Sigmoide y Tanh conduce a un valor absoluto pequeño menor que 1 del gradiente resultante, que cuando se multiplica continuamente con una serie de valores absolutos igualmente pequeños, como en la regla de la cadena, lleva a que el gradiente tienda a cero.

DL desempeña un papel crucial en el RL, una rama de la IA que se centra en el aprendizaje de agentes que toman decisiones en entornos dinámicos. En RL, DL se utiliza para construir funciones de valor o políticas que permiten al agente aprender y tomar decisiones basadas en la información del entorno.

2.3.3. Aprendizaje por Refuerzo

El aprendizaje por refuerzo (RL) es un enfoque computacional para entender y automatizar el aprendizaje dirigido a objetivos y a la toma de decisiones. Se distingue de

otros enfoques computacionales por su énfasis en el aprendizaje de un agente a partir de la interacción directa con su entorno, sin requerir una supervisión ejemplar ni modelos completos del entorno. El RL es el primer campo que aborda seriamente las cuestiones computacionales que surgen cuando se aprende de la interacción con un entorno para alcanzar objetivos a largo plazo.

El RL utiliza el marco formal de los procesos de decisión de Markov para definir la interacción entre un agente de aprendizaje y su entorno en términos de estados, acciones y recompensas, en la figura 2.5 se observa una representación de esta interacción. Este marco pretende ser una forma sencilla de representar características como un sentido de causa y efecto, de incertidumbre y no determinismo, y la existencia de objetivos explícitos (Sutton & Barto, 2018).

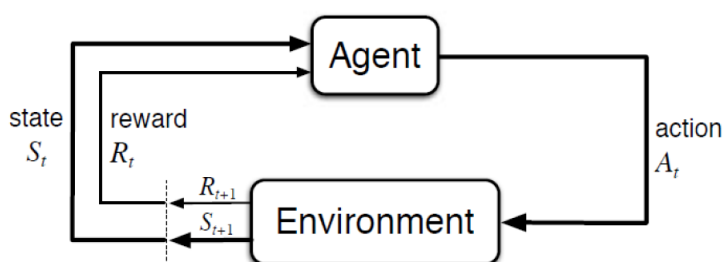


Figura 2.5: Interacción de un agente con su ambiente. Tomada de (Sutton & Barto, 2018).

En el marco de RL, el agente se programa para percibir el entorno, realizar acciones, recibir información y tomar decisiones basadas en esa información. El entorno puede ser tanto real como simulado, y el agente interactúa con él a través de un ciclo continuo de observación, acción y retroalimentación. El agente debe tomar decisiones óptimas basándose en el estado actual para maximizar las recompensas futuras, las cuales son la retroalimentación que el agente recibe después de realizar una acción. El estado representa la percepción o configuración actual del entorno que el agente puede observar, puede haber un número finito o infinito de posibles estados, dependiendo de la complejidad del entorno. Previamente se tiene que definir la tarea u objetivo a realizar y se detalla una serie de reglas para obtener una recompensa, esta puede ser positiva o negativa, dependiendo de si la acción tomada fue beneficiosa o perjudicial para alcanzar los objetivos.

El objetivo del agente es maximizar la recompensa global a largo plazo, considerando tanto las recompensas inmediatas como las futuras. Además, las acciones representan las diferentes opciones disponibles para el agente en un determinado estado, pueden ser acciones discretas o acciones continuas, dependiendo de la naturaleza del problema.

Durante el RL, el conjunto de estado, acción y recompensa es fundamental para aprender y ajustar las estrategias del agente a lo largo del tiempo, ya que cada interacción entre el agente y el entorno se representa mediante este conjunto el cual se considera un paso de ejecución en un episodio. Un episodio en RL representa una ejecución completa de la tarea, desde el inicio hasta la finalización, durante un episodio, el agente toma decisiones,

realiza acciones, recibe recompensas y ajusta su comportamiento en función de la retroalimentación recibida. El objetivo del agente es aprender una política óptima que maximice las recompensas acumuladas a lo largo de múltiples episodios.

Los fundamentos del RL se basan en tres temas. El más importante es el proceso de decisión de Markov (MDP), un marco que le ayuda a describir su problema. La programación dinámica (PD) y los métodos de Montecarlo son la base de todos los algoritmos que pretenden resolver los MDP (Winder, 2018).

Existen dos tipos de algoritmos de RL, los basados en modelos y sin modelo. Los algoritmos basados en modelos utilizan el conocimiento definitivo del entorno en el que operan para mejorar el aprendizaje. Puede utilizar este conocimiento para restringir el algoritmo de manera que no proporcione acciones inválidas y mejorar el rendimiento proyectando hacia adelante en el tiempo.

Al igual que los sistemas expertos, las soluciones basadas en modelos aprenden de forma eficiente porque no pierden tiempo buscando caminos inadecuados. Por otra parte, los algoritmos sin modelo pueden, en teoría, aplicarse a cualquier problema y aprenden estrategias a través de la interacción, absorbiendo cualquier regla del entorno en el proceso. Algunos algoritmos pueden aprender modelos del entorno al mismo tiempo que aprenden estrategias óptimas. Una representación compacta de los algoritmos de RL que existen se observa en la figura 2.6.

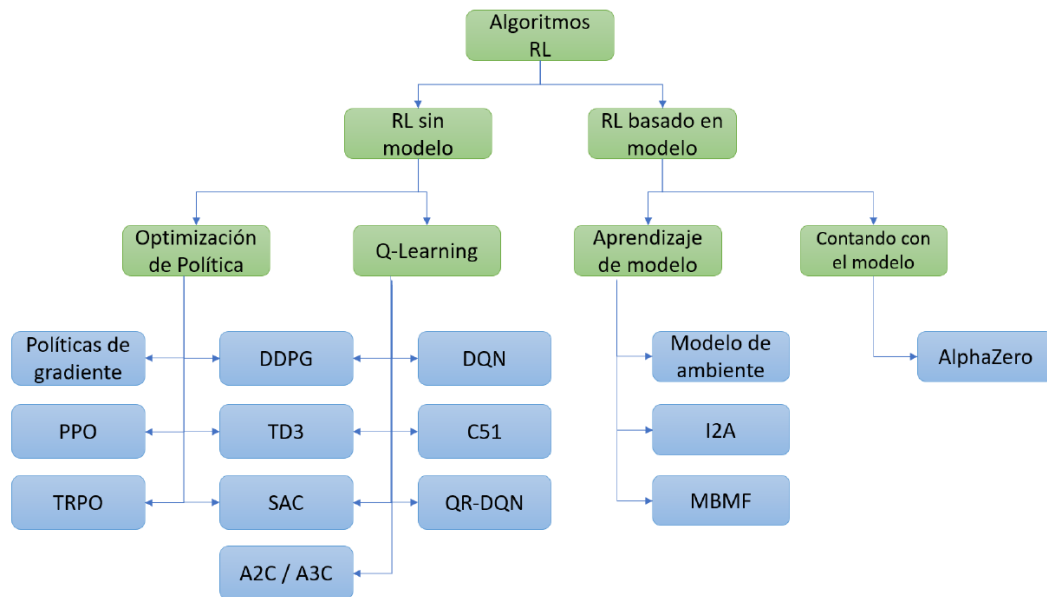


Figura 2.6: Taxonomía de algoritmos de RL.

Algoritmo Deep Deterministic Policy Gradient

Entre los algoritmos más destacados en RL se encuentra el Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016), que aborda eficientemente problemas con espacios

de acción continuos. DDPG es un algoritmo basado en la combinación de dos conceptos clave: Redes Neuronales Profundas y Política Determinista. Utiliza una arquitectura compuesta de dos redes neuronales profundas, conocida como Actor-Crítico.

El *actor* es responsable de aprender y generar una política determinista que mapea directamente los estados a acciones. En lugar de estimar la función Q , el actor aprende directamente la mejor acción a tomar en función del estado actual. Esto se logra mediante la optimización de la función de pérdida, que busca minimizar la diferencia entre las acciones seleccionadas por el actor y las acciones óptimas. La fórmula para la actualización del actor en el algoritmo DDPG se representa con la ecuación 2.1:

$$\theta' = \theta + \alpha \nabla Q(s, a | \theta) \quad (2.1)$$

Donde,

- θ es el vector de parámetros del actor
- α es la tasa de aprendizaje
- $\nabla Q(s, a | \theta)$ es el gradiente de la función Q con respecto a los parámetros del actor

El *crítico*, por otro lado, se encarga de evaluar la calidad de las acciones tomadas por el actor. Utiliza la función Q para estimar el valor esperado de la recompensa acumulada a lo largo del tiempo. Al actualizar el crítico, se busca minimizar la diferencia entre las estimaciones Q actuales y las recompensas reales obtenidas. La fórmula de actualización del crítico en DDPG se expresa con la ecuación 2.2:

$$L = (r + \gamma Q'(s', \mu'(s')) - Q(s, a))^2 \quad (2.2)$$

Donde,

- r es la recompensa obtenida
- γ es el factor de descuento
- s' es el próximo estado
- μ' es la política determinista objetivo del actor
- Q' es la función Q objetivo del crítico

Una característica distintiva del DDPG es el uso de un mecanismo de replay buffer (Zhang & Sutton, 2018), que almacena y recupera las experiencias pasadas del agente. Esto

permite un muestreo eficiente de experiencias aleatorias para el aprendizaje, evitando la correlación temporal inherente en la secuencia de interacciones.

Otra técnica clave utilizada en DDPG es el uso de dos redes objetivo (target networks), una para el actor y otra para el crítico. Estas redes se utilizan para estimar los valores objetivo de las estimaciones Q y la política determinista, lo que estabiliza el aprendizaje y evita la sobre estimación de los valores Q . Las ecuaciones para la actualización de las redes objetivo se expresan en 2.3 y 2.4:

$$\theta' = \tau\theta + (1 - \tau)\theta' \quad (2.3)$$

$$w' = \tau w + (1 - \tau)w' \quad (2.4)$$

Donde,

- θ' y w' son los vectores de parámetros de las redes objetivo
- θ y w son los vectores de parámetros de las redes principales
- τ es un hiperparámetro que controla la suavidad de la actualización

El algoritmo DDPG se entrena mediante iteraciones de exploración y explotación en el entorno. En cada iteración, el agente toma acciones basadas en la política determinista generada por el actor y recibe retroalimentación en forma de recompensas. Utilizando esta retroalimentación, se actualizan tanto el crítico como el actor a través de técnicas de optimización basadas en gradientes.

Este algoritmo ha demostrado ser efectivo en una amplia gama de aplicaciones de RL. Su capacidad para lidiar con espacios de acción continuos y generar políticas deterministas ha permitido abordar problemas complejos en áreas como robótica, control de procesos y juegos. Al combinar el poder de las redes neuronales profundas con la optimización basada en gradientes, cuenta con la capacidad para aprender de forma eficiente en espacios de acción continuos, que lo hace especialmente adecuado para aplicaciones del mundo real donde las acciones no se limitan a opciones discretas.

Algoritmo SAC: Soft Actor-Critic

El algoritmo SAC (Haarnoja et al., 2018) es un enfoque avanzado en el campo del RL que combina elementos de la teoría de la información y el aprendizaje profundo para aprender políticas de decisión óptimas en entornos de RL.

El objetivo principal del algoritmo SAC es maximizar la recompensa acumulada en un entorno dado. A diferencia de otros algoritmos de RL, el SAC utiliza una política estocástica en lugar de una política determinista. Esto significa que la política SAC no solo genera acciones, sino que también proporciona una distribución de probabilidad sobre las

posibles acciones a tomar. Esto permite una mayor exploración del espacio de acciones y una mejor adaptabilidad a entornos cambiantes.

La formulación matemática básica del algoritmo SAC se basa en la optimización de una función objetivo conocida como Entropy-Regularized Reinforcement Learning. La función objetivo busca maximizar la recompensa esperada, al mismo tiempo que maximiza la entropía de la política. La entropía de la política se refiere a la incertidumbre o diversidad de acciones que la política puede generar. Esto fomenta la exploración y ayuda a evitar la convergencia prematura a políticas subóptimas.

El algoritmo SAC consta de tres componentes principales: el actor, el crítico y el regulador de entropía. El actor es responsable de generar acciones basadas en el estado actual y su objetivo es maximizar la recompensa esperada. El crítico, por otro lado, evalúa la calidad de las acciones tomadas por el actor y proporciona una estimación del valor Q , que representa la recompensa acumulada esperada. El regulador de entropía, como su nombre lo indica, regula la entropía de la política para fomentar la exploración.

La función objetivo del actor en SAC está compuesta por dos términos: el valor Q del crítico y la entropía de la política. La ecuación 2.5 expresa la función objetivo del actor.

$$J(\theta_{actor}) = \mathbb{E}_{s \sim \rho, \alpha \sim \pi} [\alpha \log \pi_{\theta}(\alpha|s) - Q_{critico}(s, \alpha)] \quad (2.5)$$

Donde,

- θ_{actor} representa los parámetros del actor
- ρ es la distribución de estados
- π es la política generada por el actor
- $Q_{critico}$ es la función Q estimada por el crítico

La función objetivo del crítico en SAC se basa en la estimación de la función Q y se compone de dos términos: el error cuadrático medio (MSE) entre la estimación Q y el valor objetivo, y el término de regularización de entropía. La ecuación para la función objetivo del crítico se expresa como en 2.6:

$$J(\theta_{critico}) = \mathbb{E}_{s, \alpha \sim \rho, \epsilon \sim clip} [(Q_{critico}(s, \alpha) - y)^2 - \alpha \log(\pi_{\theta}(\alpha|s))] \quad (2.6)$$

Donde,

- $\theta_{critico}$ representa los parámetros del crítico
- ρ es la distribución de estados

- ϵ es una variable de ruido añadida a las acciones para la exploración
- y es el valor objetivo calculado como la recompensa inmediata más la estimación del valor Q del próximo estado ponderado por el factor de descuento

El algoritmo SAC ha demostrado un rendimiento sobresaliente en una variedad de aplicaciones de RL y se ha utilizado con éxito en tareas de control robótico, juegos y optimización de carteras, entre otros. Los experimentos han mostrado que el SAC puede lograr políticas óptimas y alcanzar altas recompensas en entornos desafiantes y complejos.

El éxito del SAC en una variedad de aplicaciones de RL destaca su importancia y relevancia en la búsqueda de agentes inteligentes capaces de tomar decisiones óptimas en entornos dinámicos y complejos.

Algoritmo Proximal Policy Optimization (PPO)

El algoritmo Proximal Policy Optimization (PPO) (Schulman et al., 2017) es uno de los más usados en aprendizaje por refuerzo profundo debido a su efectividad y estabilidad en el entrenamiento de agentes para tareas de control continuo, un ejemplo de estas tareas de control es la locomoción en robots cuadrúpedos. Es un algoritmo on-policy, entrena solo con los datos recolectados por la misma política que se está optimizando.

PPO fue propuesto por OpenAI y pertenece a los métodos de optimización basados en políticas, surge como una alternativa más sencilla y eficiente al algoritmo Trust Region Policy Optimization (TRPO), manteniendo la idea central de restringir los cambios abruptos en la política para preservar la estabilidad del entrenamiento.

El funcionamiento de PPO contempla una mejora iterativa de la política actual, al inicio, los datos se adquieren ejecutando la política actual en el entorno durante varios pasos. Esta ejecución genera trayectorias compuestas por secuencias de estados, acciones, recompensas y predicciones del valor del estado, los cuales son almacenados para el posterior entrenamiento. Posteriormente, basadas en estas trayectorias se calcula la *advantage function*, la cual mide qué tan buena fue una acción en relación a lo esperado por el valor del estado. Una forma común de calcular esta ventaja es utilizando la técnica conocida como Generalized Advantage Estimation (GAE), lo reduce la varianza en las estimaciones y permite un entrenamiento más estable. La fórmula general para la ventaja es:

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \quad \text{donde} \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

donde γ es el factor de descuento para las recompensas futuras, λ controla el sesgo-varianza de la estimación, r_t es la recompensa recibida en el paso t , y $V(s_t)$ es la estimación del valor del estado actual.

El siguiente paso en PPO consiste en comparar la nueva política con la política anterior mediante un cociente de probabilidad conocido como la razón de cambio:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Esta razón de cambio representa cuanto difiere la nueva política con respecto a la anterior al escoger la misma acción a_t en el mismo estado s_t . Un cambio excesivo en este valor puede generar que el agente se aleje demasiado de políticas previamente útiles, lo que causa inestabilidad o pérdida de rendimiento. Para evitarlo, PPO introduce una función de pérdida que restringe este cambio mediante un operador de recorte o *clipping*, dado por:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t [\text{mín}(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon)A_t)]$$

donde ε es un hiperparámetro que define el margen aceptable de cambio en la política. Si la razón de cambio r_t permanece dentro del intervalo $[1 - \varepsilon, 1 + \varepsilon]$, la política se actualiza de manera normal. Si el cambio sobrepasa esos límites, se detiene la actualización para esa muestra específica, manteniendo así el cambio de política dentro de una región confiable.

Además de la política, PPO entrena una red crítica la cual estima el valor de los estados, esta se optimiza utilizando una función de pérdida tipo error cuadrático medio entre el valor estimado y el retorno acumulado descontado:

$$L_V = (V(s_t) - \hat{R}_t)^2$$

donde $\hat{R}_t$ es la suma de recompensas descontadas desde el paso t , y $V(s_t)$ es el valor predicho por la red crítica. Tanto la política como la función de valor se actualizan mediante el gradiente descendente durante varios ciclos de entrenamiento.

La combinación de estas técnicas permite a PPO lograr un balance entre estabilidad y rendimiento. Su estructura hace que sea fácil de implementar y ajustar, a la vez que mantiene un comportamiento robusto frente a cambios en el entorno o en la política del agente. En aplicaciones de locomoción robótica, PPO ha demostrado ser altamente eficaz para aprender comportamientos complejos de manera emergente, partiendo únicamente de observaciones sensoriales y sin necesidad de trayectorias de referencia ni modelos dinámicos precisos. Esto lo convierte en una herramienta fundamental para el desarrollo de controladores inteligentes en robots cuadrúpedos.

2.4. Simuladores

Las simulaciones se convirtieron en una parte cada vez más importante parte del desarrollo de hardware, especialmente en el campo de la robótica y debido a esto existen diversos entornos de simulación que proporcionan información de calidad al momento de compararlos con la implementación real, ya que utilizan motores físicos existentes o

personalizados para calcular el comportamiento de cuerpos con masa bajo la influencia de fuerzas como la gravedad, la inercia o las restricciones de contacto.

2.4.1. Nvidia Isaac Sim

Nvidia Isaac Sim es un simulador de robótica desarrollado por Nvidia como parte de sus aplicaciones basadas en NVIDIA Omniverse. Este simulador está diseñado para ser utilizado en el desarrollo y entrenamiento de algoritmos de IA para robots. Dentro de las características principales de Isaac Sim se cuenta con la capacidad de generar entornos de simulación altamente realistas que permiten probar y validar algoritmos en un entorno virtual antes de implementarlos en robots físicos, además de ofrecer la posibilidad de integrarse con Nvidia Omniverse, una plataforma de simulación y colaboración en tiempo real.

El simulador aprovecha las capacidades de hardware de Nvidia, como las tarjetas gráficas RTX, para ofrecer un rendimiento y una representación visual avanzados. Gracias al hardware de Nvidia se puede utilizar CUDA para entrenar algoritmos de IA, como aprendizaje profundo y técnicas de visión por computadora, para tareas específicas de robótica, contando con la capacidad de simular una variedad de sensores, como cámaras y LiDAR, para replicar el entorno del mundo real (Nvidia, 2025).

Isaac Sim permite a los desarrolladores simular y probar soluciones robóticas con un ambiente totalmente flexibles y personalizadas basadas en “Universal Scene Description (OpenUSD)”. Universal Scene Description (USD) es el primer software disponible públicamente que aborda la necesidad de intercambiar y ampliar de forma robusta y escalable escenas 3D arbitrarias que pueden estar compuestas por muchos activos elementales, además permite el intercambio de activos elementales (por ejemplo, modelos) o animaciones. A diferencia de otros paquetes de intercambio, USD también permite ensamblar y organizar cualquier número de activos en conjuntos virtuales, escenas, tomas y mundos, transmitiéndolos de una aplicación a otra y editándolos de forma no destructiva (como sustituciones).

Los procesos de producción de películas y videojuegos con gráficos por ordenador suelen generar, almacenar y transmitir grandes cantidades de datos 3D, lo que se denomina *descripción de escena*. Cada una de las numerosas aplicaciones que cooperan en el proceso (modelado, sombreado, animación, iluminación, efectos especiales, renderizado) suele tener su propia forma especial de descripción de escena, adaptada a las necesidades y flujos de trabajo específicos de la aplicación, que no es legible ni editable por ninguna otra aplicación.

Isaac Sim facilita tres flujos de trabajo esenciales: generar datos sintéticos para entrenar y ajustar los modelos básicos de robots, realizar pruebas de software en el circuito para células de robots y habilitar el aprendizaje de robots a través de Isaac Lab. En la figura 2.7 se presenta la estructura de isaac sim.

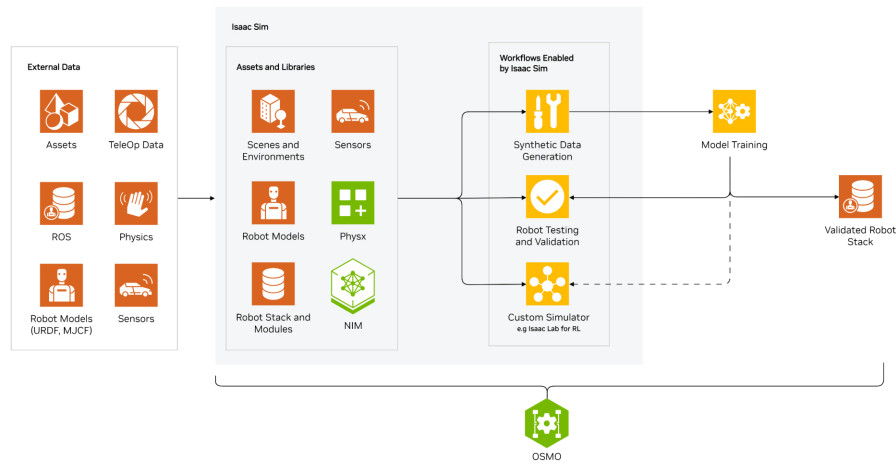


Figura 2.7: Flujo de trabajo de Nvidia Isaac Sim. Tomado de (Nvidia, 2025).

Nvidia lanzó el concepto de *digital twin*, que se puede pensar como un análogo digital del robot real, lo cual reduce la llamada brecha entre la realidad y la simulación. En la figura 2.8 se observa un ambiente creado en Isaac sim.



Figura 2.8: Nvidia Isaac Sim, entorno de simulación de robots con el concepto de digital twin. Tomado de (Nvidia, 2025).

Isaac Sim tiene una serie de modelos de robots de terceros creados en OpenUSD que tienen las propiedades físicas ideales para acelerar la simulación de robots.

- **Humanoides:** 1X, Agility, Fourier Intelligence, and Sanctuary.
- **Manipuladores:** Fanuc, KUKA, Universal Robots, and Techman.
- **Cuadrúpedo:** ANYbotics, Boston Dynamics y Unitree.

- **Robots móviles:** idealworks, iRobot.

2.4.2. Nvidia Isaac Lab

NVIDIA Isaac Lab es un marco unificado de código abierto para el aprendizaje robótico, diseñado para ayudar a entrenar políticas robóticas. Proporciona simulación física de alta fidelidad mediante NVIDIA PhysX y renderizado basado en la física de NVIDIA RTX (Nvidia, 2022), está construido sobre Isaac Sim como se presenta en la figura 2.9.

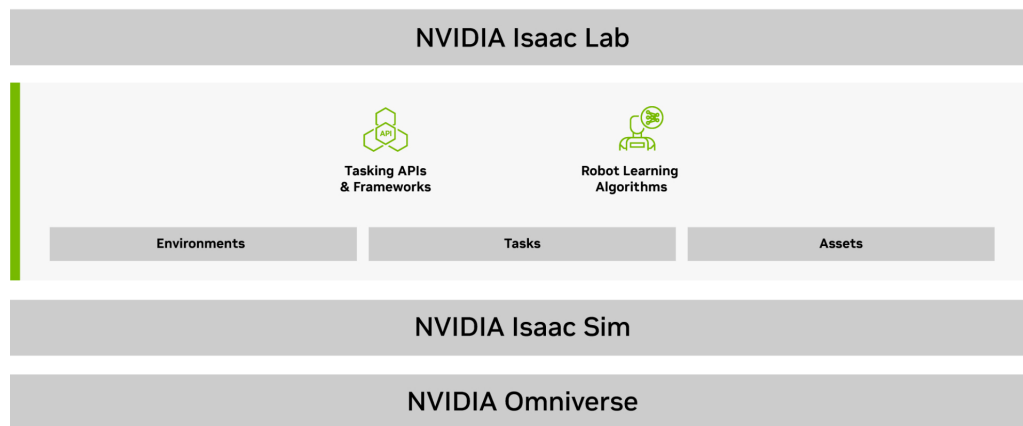


Figura 2.9: Estructura de Isaac Lab. Tomado de (Nvidia, 2022).

La arquitectura modular de Isaac Lab y la paralelización basada en GPU NVIDIA lo hacen ideal para desarrollar políticas robóticas que abarcan una amplia gama de encarnaciones, incluyendo robots humanoides, manipuladores y robots móviles autónomos. Es un ambiente ideal para realizar experimentos de aprendizaje robótico, desde la configuración del entorno hasta el entrenamiento y la implementación de políticas. Tiene la capacidad de personalizar y ampliar sus capacidades con diversos motores de física, como NVIDIA PhysX, Warp y MuJoCo.

En Isaac Lab se pueden crear sistemas robóticos más robustos, eficientes y capaces mediante la enseñanza de nuevas habilidades a los robots por medio de simulación. El aprendizaje robótico es una parte fundamental del proceso de entrenamiento de los robots. Dado que el entrenamiento de los robots puede requerir mucho tiempo y recursos, el entrenamiento físico puede complementarse con el entrenamiento en entornos simulados para la IA física. La aplicación de técnicas de aprendizaje robótico en entornos simulados puede acelerar los tiempos de entrenamiento y permitir la escalabilidad, ya que muchos

CAPÍTULO 2. MARCO TEÓRICO

robots pueden aprender y entrenarse simultáneamente. En la figura 2.10 se presentan ejemplos de ecosistemas de aprendizaje con robots.



Figura 2.10: Ecosistemas de aprendizaje con robots en Isaac Lab. Tomado de (Nvidia, 2022).

En la simulación, los operadores también pueden añadir fácilmente varianza y ruido a cada escena con un robot, lo que le proporciona más experiencia y materiales para aprender. Existen varios enfoques de aprendizaje que pueden utilizarse para enseñar a los robots nuevas habilidades tanto en entornos físicos como simulados. Uno de estos enfoques es el aprendizaje por refuerzo que pretende imitar el proceso de aprendizaje de los seres humanos basado en experiencia, esto es, aprender a través de ensayo y error iterativos, guiado por una función que proporciona recompensas o penalizaciones en función de las acciones del robot, orientando al robot a determinar los mejores pasos a seguir. El aprendizaje por refuerzo es más útil en entornos simulados, donde los robots pueden explorar y aprender de forma segura a partir de una amplia gama de escenarios sin consecuencias en el mundo real.

Después del entrenamiento y de completar con éxito sus acciones, un modelo de IA puede implementarse en un robot del mundo real. El éxito de esta transferencia depende de varios factores, entre ellos la brecha entre los entornos virtual y real, la dificultad de la tarea de aprendizaje y la complejidad de la propia plataforma robótica. Otro enfoque ampliamente usado es el aprendizaje por imitación, donde los robots observan y replican las demostraciones de un experto, ya sean vídeos reales de humanos o datos simulados. El aprendizaje por imitación utiliza conjuntos de datos etiquetados para enseñar a los robots acciones complejas que son difíciles de definir mediante programación. Es especialmente útil para robots humanoides diseñados para funcionar y colaborar con humanos. Aunque grabar una demostración puede ser más sencillo que especificar una política de recompensas, como en el aprendizaje por refuerzo, crear una demostración perfecta puede ser complicado y los robots pueden tener dificultades con situaciones imprevistas.

2.5. Robot Operating System (ROS)

En robótica y sistemas de control complejos se demanda una alta carga de procesamiento para leer datos de sensores y enviar comandos de control a cada parte del sistema, esto requiere un controlador potente que pueda manejar todas las variables y calcular cada una de las acciones, para sistemas robóticos se requiere que el tiempo de control sea en mili segundos y es por esto que existen varias estrategias para lograr el control en tiempo real.

ROS no es un sistema operativo que gestiona y optimiza procesos, más bien proporciona una capa de comunicaciones estructurada encima del sistema operativo en el que se trabaje (Quigley et al., 2009), por otra parte, proporciona un conjunto de programas y drivers de comunicación de módulos que conviven en un ecosistema basado en Nodos, estos se comunican por medio de mensajes enviados por canales llamados tópicos. Los sistemas a controlar pueden verse como nodos, ya que estos ejecutan una tarea en específico haciendo modular al sistema. En robótica por ejemplo, la tarea de control se basa en leer sensores específicos del robot y calcular las acciones de cada uno de sus motores dependiendo de su modo de control y la dinámica del robot, pero esto requiere que un controlador, computadora o sistema embebido, ejecute cada una de las acciones de manera seriada lo que resulta en carga computacional y tiempos excesivos además de contar con cada una de las interfaces para comunicarse con los motores y sensores. ROS propone un compendio de librerías reutilizables con interfaces para comunicarse con sistemas de manera genérica, al ser open source se puede utilizar desarrollos previos, la ideología detrás de ROS es "No reinventes la rueda. Crea algo nuevo y hazlo más rápido y mejor construyendo sobre ROS"(OpenRobotics, 2025a).

2.5.1. ROS2

ROS no cuenta con capacidad de realizar comunicaciones en tiempo real dada su estructura de diseño, alcanzaba un rendimiento óptimo pero se considera un sistema que no trabaja en tiempo real al carecer de las características de este tipo de sistemas, como el cumplimiento de tareas en tiempo establecido o un planificador de tareas. Dado que los sistemas de control críticos requieren de sistemas en tiempo real, se presento en 2014 la actualización de ROS2, esta versión fue implementada con un protocolo estándar industrial llamado Servicios de distribución de datos, DDS por sus siglas en ingles, este protocolo sirve de middleware integral que proporciona un transporte de publicación suscripción muy similar al de ROS. Utiliza el lenguaje de descripción de interfaces (IDL) definido por Object Management Group para la definición y serialización de mensajes (Macenski et al., 2022). DDS se ha utilizado en diversas aplicaciones, desde grandes instalaciones como presas hasta sistemas espaciales utilizados por la NASA.

La principal característica de ROS es que cuenta con un sistema de comunicación mediante publicadores y suscriptores, estos son identificados como nodos que intercambian

información por medio de tópicos y mensajes. Cuando se tienen varios elementos de ROS trabajando al mismo tiempo e interconectados en una red se le conoce como Grafo de ROS. Se puede observar de manera gráfica este grafo con cada uno de sus elementos, el ejemplo mas sencillo se muestra en la figura 2.11, consta de un nodo publicador, 1 nodo suscriptor, 1 tópico y el mensaje que quieren comunicar.

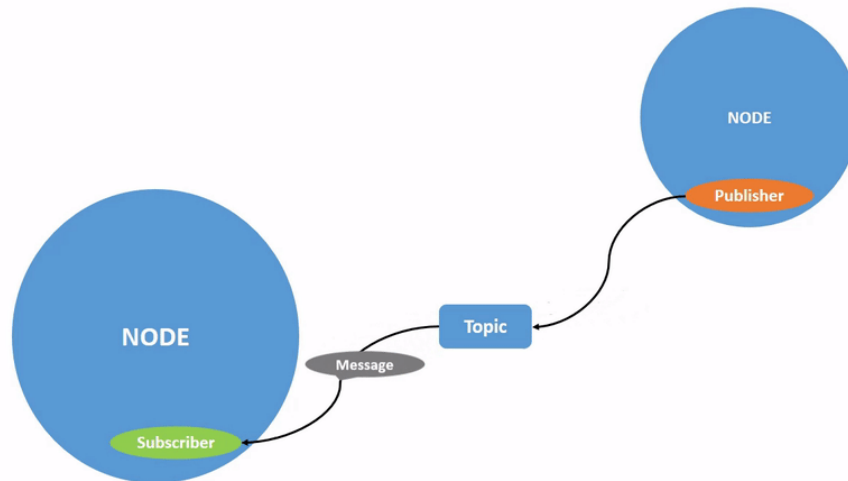


Figura 2.11: Modelo simple del grafo de ROS, donde se observa dos nodos, un publicador y suscriptor comunicando un mensaje a través de un tópico. Tomado de (OpenRobotics, 2025b)

Los tópicos son de vital importancia en los elementos de ROS, ya que son el medio por el cual viaja la información o el mensaje, los nodos pueden publicar información a diversos tópicos y al mismo tiempo pueden estar suscritos a otros tópicos, así mismo, varios nodos pueden publicar información al mismo tópico como se muestra en la figura 2.12.

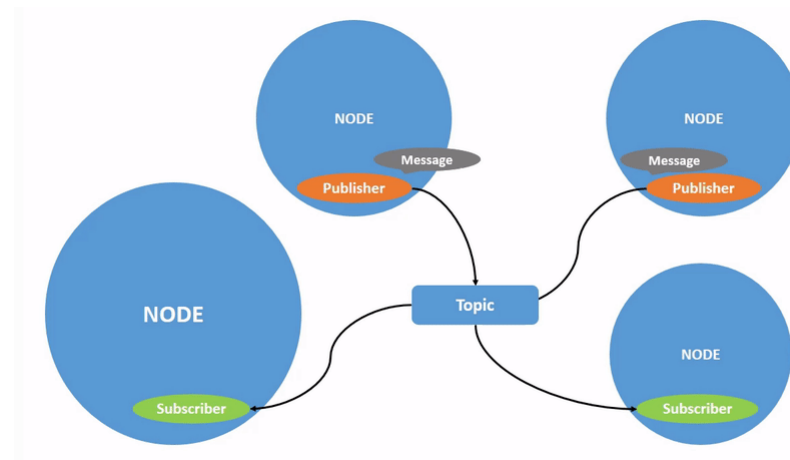


Figura 2.12: Representación de un servicio en ROS. Tomado de (OpenRobotics, 2025b)

ROS2 incorpora nuevas formas de comunicación entre nodos que permite aumentar la flexibilidad de los sistemas, además de la forma tradicional de publicador-suscriptor, ROS2 cuenta con Servicios y Acciones que son métodos que controlan la comunicación entre nodos. Los servicios en ROS2 se basan en petición y respuesta, esto es, solo mandan información cuando un cliente solicita específicamente la información. En la figura 2.13 se observa el esquema de dos nodos comunicándose por medio de un servicio, donde el nodo cliente de color verde manda una petición al nodo servidor de color naranja, todo lo gestiona el servicio de ROS2. Para los servicios se considera que solo existe un nodo servidor y pueden existir varios clientes.

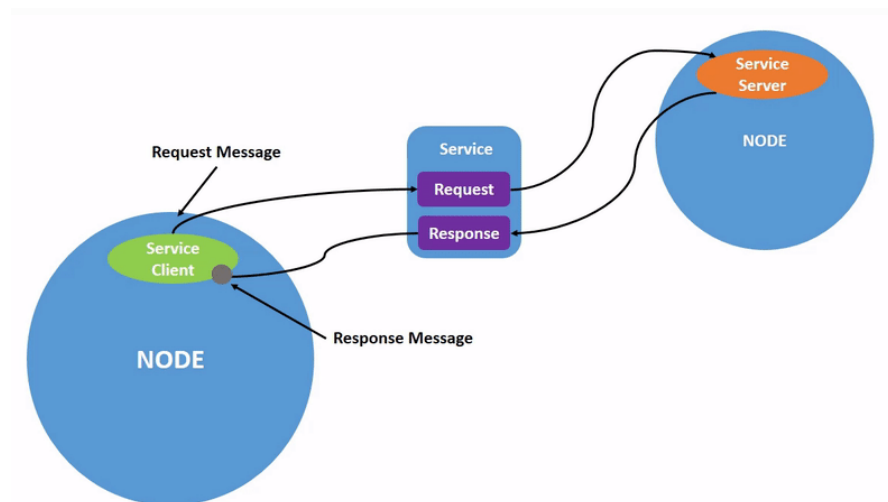


Figura 2.13: Representación de un servicio en ROS. Tomado de (OpenRobotics, 2025b)

Por último, se tienen a las acciones que están conformadas por tópicos y servicios, esta forma de comunicación sirve para tareas que requieren un seguimiento con retroalimentación y están pensadas para tareas de larga duración, consta de 3 partes, una meta, retroalimentación y un resultado. Su funcionalidad es similar a la de los servicios solo que las acciones pueden ser canceladas en determinadas situaciones y estas entregan una retroalimentación de manera constante a diferencia de los servicios que emiten una sola respuesta. En la figura 2.14 se observa la estructura de una acción en el grafo de ROS.

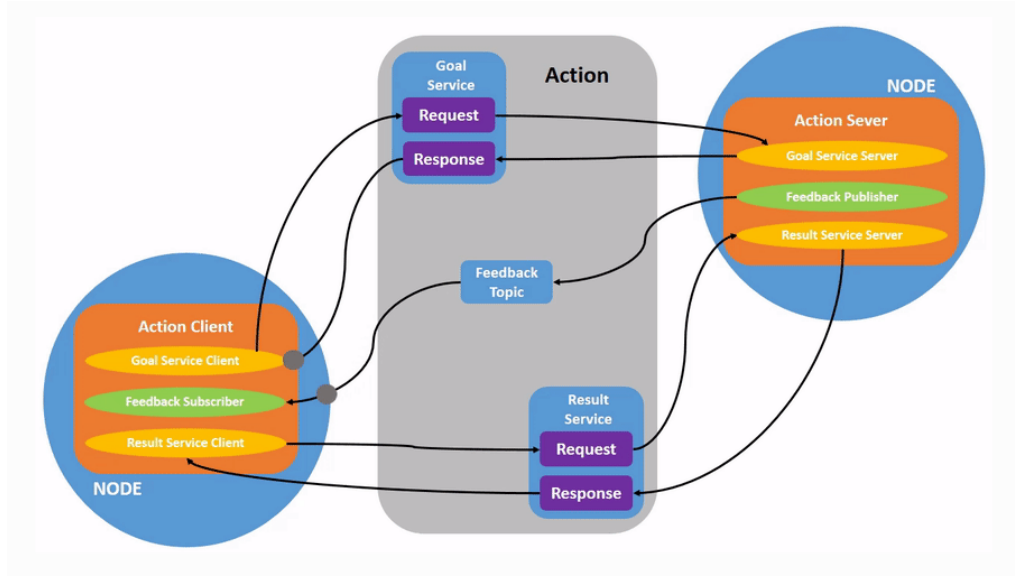


Figura 2.14: Representación de una acción en ROS. Tomado de (OpenRobotics, 2025b)

2.6. Conclusiones

En este capítulo se presentaron las teorías y métodos que anteceden a este trabajo de tesis, empezando por la definición de los métodos de control para robots cuadrúpedos. Se expuso una breve recopilación sobre los métodos basados en modelos y sin modelo y este apartado enfatizó en los enfoques basados sin modelo y las nuevas metodologías como el DRL.

Por otra parte, se definieron conceptos para conocer las partes fundamentales del marco necesario de RL, así como también, se abordaron dos algoritmos que han mostrado resultados contundentes al momento de enfrentar espacios de estados y acciones continuas, lo cual es el motivo de este trabajo de investigación.

En el siguiente capítulo conoceremos los diversos trabajos que existen en la literatura que abordan el tema de control de robot cuadrúpedo con técnicas de DRL.

Estado del Arte

Para la ejecución de la revisión del estado del arte se incluyeron los trabajos que demostraron una metodología explícita en la experimentación con algoritmos DRL en robots cuadrúpedos físicos, simulados y en ambos casos. Se revisaron trabajos que realicen una comparativa con otros algoritmos utilizados en el estado del arte, así como los trabajos que se centren únicamente en el aprendizaje de locomoción en robots cuadrúpedos sin modelo matemático, ya que muchos trabajos se centran en optimización de controladores.

El propósito de la revisión del estado del arte principalmente es realizar una recopilación y búsqueda de algoritmos de DRL que logren realizar tareas de control de marcha en un robot cuadrúpedo para poder responder a las preguntas de investigación siguientes:

- ¿Qué características destacan al controlar un robot cuadrúpedo con las técnicas de aprendizaje profundo por refuerzo en sistemas sin modelo en comparación con las técnicas de control clásico de robots que requieren un modelo del sistema?
- ¿Qué mejoras ofrece el aprendizaje profundo por refuerzo para controlar un sistema robótico cuadrúpedo?
- ¿Cuál es la efectividad al controlar un robot cuadrúpedo con aprendizaje profundo por refuerzo?

La revisión de la literatura se realizó principalmente en revistas indexadas en las bases de datos de Journal Citation Report (JCR), Scopus, EBSCO, IEEE Xplore, Springer y Elsevier. Además, se contempló la búsqueda en congresos como ICRA e IROS.

3.1. Síntesis de los Estudios

Es difícil establecer un modelo completo de un robot cuadrúpedo debido a su gran número de GDL y su fuerte no linealidad. El algoritmo de control del robot cuadrúpedo suele ser jerárquico, esto es, se divide en dos capas que incluyen el control del cuerpo y el control de una pierna (Jinhui et al., 2016). Por otra parte, en los últimos años se han estudiado métodos de control que dependen solo de los datos obtenidos por el robot y su interacción con el entorno, ya que dichos métodos requieren herramientas de procesamiento de datos potentes y complejos que hasta ahora se han logrado explotar, como las técnicas de aprendizaje máquina dentro de la rama de IA y el aprendizaje de extremo a extremo (E2E), el cual se refiere al entrenamiento de un sistema de aprendizaje posiblemente complejo representado por un único modelo (concretamente una red neuronal profunda) que representa el sistema objetivo completo.

El aprendizaje por refuerzo (Hwangbo et al., 2019) puede aprender directamente controladores efectivos en la práctica, con la esperanza de superar las limitaciones sufridas por controladores convencionales. Dado que es difícil el entrenar algoritmos de RL en sistemas físicos, se aplica a los robots cuadrúpedos sólo mediante la simulación en la mayoría de los casos. La locomoción de los robots cuadrúpedos es un problema de control que requiere alta velocidad de los actuadores, así como una coordinación precisa entre sus extremidades. Además de la locomoción básica en diferentes terrenos, tareas o condiciones ambientales pueden requerir comportamientos primitivos específicos (Jain et al., 2019).

RL es un método basado en datos en el que el algoritmo estima la función de recompensa óptima entre el estado del sistema y la acción a través de una gran cantidad de datos de retroalimentación, es decir, la estrategia de acción óptima. Sin embargo, durante el proceso de aprendizaje producirá la dinámica de modelado rígido, los parámetros de simulación incorrectos y errores numéricos. Por lo cual, no solo es difícil reiniciar automáticamente los experimentos, recopilando datos continuamente, sino que también puede causar daños potenciales al robot (Singla et al., 2019). RL, DRL y otros métodos sin modelo para derivar los comportamientos de locomoción robustos de los cuadrúpedos pueden resolver las dificultades de reconocimiento y modelado de robots (Tan et al., 2018). Pero el entrenamiento de RL sin modelo generalmente se realiza en una simulación, esto deriva a tener consideraciones al momento de transferir lo aprendido al robot real, por lo general tiene una brecha de realidad debido a la discrepancia en el modelo entre el sistema físico real y el simulado.

RL es una opción ampliamente utilizada en la literatura para lograr la tarea de control de un sistema robótico con alta dimensionalidad, como se observa a continuación, existen diversos algoritmos de aprendizaje por refuerzo en conjunto con el aprendizaje profundo para controlar un robot cuadrúpedo y podemos encontrar en la literatura que los trabajos se encuentran divididos en 2 categorías:

3.1.1. Aprendizaje de locomoción en robots cuadrúpedos físicos

(Wang et al., 2020) propone un método heurístico de aprendizaje de la marcha para robots cuadrúpedos basado en DDPG, inspirado en el algoritmo de aprendizaje Double Q, utilizaron dos críticos independientes para seleccionar el valor más pequeño para actualizar los parámetros. Realiza una verificación experimental demostrando que el algoritmo DDPG presenta un mejor rendimiento al compararlo con el algoritmo PPO.

(Haarnoja et al., 2019) presenta una extensión del algoritmo *Soft Actor Critic (SAC)* con la mejora de auto ajustar el parámetro de temperatura de dicho algoritmo, emplea la optimización basada en el gradiente de la temperatura hacia la entropía esperada en los estados visitados. La contribución más importante de este artículo es que plasman las bases con un marco teórico para controladores de extremo a extremo de robots cuadrúpedos físicos por medio de aprendizaje por refuerzo, ya que logran el control del robot sin el acceso a ningún diseño de trayectoria, planificador de puntos de apoyo o modelo de dinámica del robot. Todo el aprendizaje se realiza por completo a través de interacción con el mundo real únicamente con los datos de los ángulos de las articulaciones, datos obtenidos de la IMU en el robot, las posiciones deseadas de los motores y sin un pre entrenamiento en un simulador, logrando un aprendizaje de marcha en solo 2 horas de entrenamiento sin ningún conocimiento de la dinámica del robot.

(Hu et al., 2019) realiza el aprendizaje del movimiento de un sistema robótico cuadrúpedo llamado *Doggo* a través de un conocimiento previo de su controlador para movimientos de trote, los parámetros del controlador son aprendidos por el algoritmo de DRL utilizando el algoritmo *Distributed Proximal Policy Optimization (DPPO)* que es un algoritmo de aprendizaje extremo a extremo basado en el gradiente de la póliza que emite directamente las acciones en lugar de solo los valores de estas, es un algoritmo adecuado para control continuo de modelos complejos y está basado en el algoritmo tradicional SAC. Las redes neuronales utilizan la función de activación Tanh y penalizan los movimientos de desviación hacia la derecha o izquierda, así como los movimientos oscilatorios hacia arriba o abajo y también se penaliza el consumo de energía. Como resultado final se concluye que el algoritmo DPPO obtiene mejores resultados cuando se ocupa conocimiento previo de la marcha del robot y converge 0.5 veces más rápido.

(Isken et al., 2018) propone una arquitectura para el aprendizaje de comportamientos controlables complejos mediante políticas simples que modulan los generadores de trayectorias (PMTG), lo cual puede proporcionar tanto memoria como conocimiento previo al controlador. Para aprender la locomoción controlada por velocidad de un robot cuadrúpedo se utiliza DRL y estrategias evolutivas. Se demostró que una política lineal simple, cuando se combina con un generador de trayectorias paramétrico para marcha en robots cuadrúpedos, puede inducir comportamientos con velocidad controlable a partir de observaciones de la IMU en 4 dimensiones.

(Lele et al., 2020) presentan un controlador basado en un generador de patrones centrales (CPG) basado en neuronas tipo *Spiking* de extremo a extremo que lleva a cabo el

procesamiento en línea de la información sensorial y el aprendizaje de la generación de la marcha en robots hexápodos. El autor afirma que es el primer trabajo que describe el aprendizaje en línea para la locomoción en robots utilizando la dinámica de las redes de *spiking* que puede encontrar una aplicación potencial en la robótica *on the Edge* y otros sistemas embebidos de baja potencia.

(Gehring et al., 2013) presenta un marco de control para un robot cuadrúpedo capaz de realizar el control de locomoción utilizando varios patrones de marcha, tales como caminata estática, trote caminando y trote corriendo, asegurando una transición fluida entre ellas. Además, demostró la flexibilidad del algoritmo realizando experimentos con el robot cuadrúpedo *StarlETH*.

(Barter & Yin, 2021) presentan un robot cuadrúpedo capaz de controlar la postura y la locomoción dirigida a un objetivo en un terreno irregular. La arquitectura de control es una red jerárquica de sistemas simples de control con retroalimentación negativa inspirados en la organización del sistema nervioso de los vertebrados. Este robot es capaz de controlar la postura y la locomoción de forma robusta en entornos nuevos con perturbaciones imprevisibles.

(Gangapurwala et al., 2020) introduce la optimización guiada de política restringida (GCPO), un marco de RL basado en la optimización del algoritmo de política proximal restringida (CPPO) para el seguimiento de los comandos de velocidad de la base del robot mientras se siguen las restricciones definidas. Presentan resultados experimentales del método de entrenamiento y lo implementan en el robot cuadrúpedo ANYmal real, comparando su enfoque con el método RL sin restricciones y mostrando que el GCPO ofrece una convergencia más rápida cerca del óptimo deseado, resultando en un control robótico sin necesidad de un ajuste preciso de la función de recompensa.

(Mitchell et al., 2020) enmarca el control de los cuadrúpedos como una optimización en un espacio latente estructurado y utiliza un modelo generativo profundo que captura una representación estadística de configuraciones conjuntas factibles de las articulaciones del robot, mientras que las restricciones dinámicas complejas se expresan a través de indicadores semánticos de alto nivel y se representan mediante clasificadores aprendidos que operan sobre el espacio latente. Como consecuencia, las restricciones complejas se hacen diferenciables y evalúan un orden de magnitud más rápido que los enfoques analíticos. Validan la viabilidad de las trayectorias de locomoción optimizadas con su enfoque tanto en simulación como en un cuadrúpedo del mundo real, demostrando que este enfoque es capaz de generar trayectorias suaves y realizables. Hasta donde se sabe, es la primera vez que se aplica con éxito el control del espacio latente a una plataforma robótica compleja y real.

(Jain et al., 2019) presenta un enfoque de control jerárquico especialmente para los robots cuadrúpedos al separar la arquitectura en dos partes, alto y bajo nivel, que se entrenan en conjunto. El enfoque se basa principalmente en que la póliza de alto nivel emite comandos en el espacio latente para la póliza de bajo nivel y decide por qué tiempo estará

en ejecución dicho comando, al mismo tiempo, la póliza de bajo nivel utiliza este comando y los sensores que se encuentran en el robot para controlar sus motores. Este esquema se probó en un robot cuadrúpedo físico de 8 GDL y se enfatiza en que comandos de control de dirección automáticamente emergen en cuanto la póliza de bajo nivel requiere ejecutar estas acciones. Por último, el autor sugiere que en lo que a esta investigación concierne, este trabajo es la primera aplicación de un aprendizaje jerárquico extremo a extremo que realiza una tarea de locomoción en un robot.

(Sun et al., 2021) proporciona una arquitectura de control basada en el aprendizaje para el auto equilibrio de cuadrúpedos, que es adaptable a múltiples escenas impredecibles de perturbación continua externa. A diferencia de los métodos convencionales que construyen modelos analíticos que razonan explícitamente el proceso de equilibrio, este trabajo utiliza RL y RNA para evitar el modelado matemático. La política de control está compuesta por una red neuronal y una política Tanh Gaussiana, que establece implícitamente el mapeo difuso de las señales propioceptivas a los comandos de acción. Durante el proceso de entrenamiento, se emplea el método de máxima entropía (algoritmo SAC) para dotar a la póliza de una potente capacidad de exploración y generalización. La póliza entrenada se valida tanto en simulaciones como en experimentos realistas con un robot cuadrúpedo personalizado.

3.1.2. Aprendizaje de locomoción en robots cuadrúpedos simulados

(Liu et al., 2020) utiliza un algoritmo genético *PSO* para mejorar la convergencia del algoritmo *Policy Gradient*. Además, permite optimizar los parámetros de la red neuronal profunda que funciona como póliza de control sin estimar el gradiente, lo que resulta en un método que requiere menos costo computacional y menor tiempo de entrenamiento. Se realizan simulaciones creando una variedad de agentes inicializando sus hiper parámetros en base a una distribución gaussiana para seleccionar la mejor póliza de control. Se realizan experimentos en varios tipos de robots entre ellos un cuadrúpedo y se utiliza una red neuronal profunda y algoritmo RL sin modelo *Policy Gradient* combinado con algoritmo genético *PSO* para explorar parámetros para que aprenda a controlar un robot cuadrúpedo de tal manera que camine hacia una dirección lo más rápido posible. Todos los experimentos se realizaron mediante simulación en el motor físico MuJoCo.

Basados en algoritmos de DRL sin modelo, (Jones et al., 2020) dan solución al problema de planificación de movimiento de un robot cuadrúpedo que se mueve de un terreno plano a un terreno inclinado. Utilizando una ligera modificación del algoritmo Distributed Proximal Policy Optimization (DPPO), realizaron únicamente simulaciones en un ambiente virtual utilizando el robot Climbot de 12 GDL. Por medio de los experimentos que presentan fueron capaces de obtener un movimiento dinámico y adaptable, capaz de pasar de un plano a una pendiente utilizando una retroalimentación háptica indirecta.

(Tsounis et al., 2020) propone un nuevo enfoque para el entrenamiento de una póliza

jerárquica que realiza la locomoción tomando en cuenta el terreno para los robots cuadrúpedos. Descomponen la locomoción en dos partes que se entrenan de forma independiente utilizando DRL sin modelo y se interconectan a través de una parametrización cuidadosamente diseñada de los movimientos de los cuadrúpedos. Como la simulación física tiene un alto coste computacional para su uso con DRL, este método reduce la complejidad de la muestra para el entrenamiento del planificador de la marcha. Han demostrado la eficacia de este enfoque entrenando y probando con éxito en una serie de escenarios difíciles. Las pólizas generadas no sólo resultan eficientes en el conjunto de terrenos rígidos no planos, sino que también consiguen generalizarse a casos nunca antes vistos.

(Schilling et al., 2021) proporcionan un análisis y una comparación de ocho arquitecturas de control para la locomoción adaptativa que desarrollaron para un agente cuadrúpedo, pero con su grado de descentralización variando sistemáticamente entre los extremos totalmente centralizado y totalmente descentralizado. Este trabajo muestra que la velocidad de aprendizaje mejora significativamente en las arquitecturas distribuidas logrando el mismo nivel de rendimiento de las arquitecturas centralizadas, debido a que los espacios de búsqueda son más pequeños y los costes locales proporcionan una información más focalizada para el aprendizaje. Realizan simulaciones en el software MuJoCo, utilizando el algoritmo PPO para el entrenamiento de las pólizas.

(W. Yu et al., 2018) proponen un nuevo método de aprendizaje de políticas que minimiza el consumo de energía y fomenta la simetría de la marcha para mejorar la naturalidad de la locomoción en diferentes sistemas simulados, dentro de ellos un cuadrúpedo. El primer elemento de este enfoque es proporcionar asistencia virtual por medio de fuerzas externas para ayudar al personaje a aprender a equilibrarse y a alcanzar una velocidad objetivo. El segundo elemento es fomentar el comportamiento simétrico mediante el uso de un término adicional en la función de pérdida. De igual manera, utilizan la optimización de política proximal (PPO) para aprender la política de locomoción óptima en conjunto con un sistema de aprendizaje curricular para facilitar el proceso de aprendizaje y este se elimina cuando el agente es lo suficientemente hábil en la tarea original.

(Wang et al., 2020) automatizan la selección de patrones de marcha para robots con patas, logrando seleccionar sistemáticamente los patrones de marcha según las especificaciones de la tarea y el entorno, sin necesidad de recurrir a la heurística o a la entrada manual. Esto lo logran al construir un mapa desde la tarea hasta la selección del patrón de marcha para un entorno y un objetivo de rendimiento determinados y demuestran que, para un robot cuadrúpedo, se puede establecer un mapa directo entre una tarea dada y un patrón de marcha óptimo.

(Tang et al., 2020) utilizan un sistema de aprendizaje multi agente, en el que un robot cuadrúpedo (protagonista) aprende a perseguir a otro robot (adversario) mientras este último aprende a escapar. Este proceso de entrenamiento adversario no sólo fomenta los comportamientos ágiles, sino que también trata eficazmente el laborioso esfuerzo de diseño del entorno.

(Truong et al., 2021) desarrolla políticas de navegación jerárquicas que dividen el control del robot en una política de alto nivel que se enfoca en el movimiento del centro de masa y una política de bajo nivel que convierte las órdenes de alto nivel (velocidad) en posiciones deseadas. Este trabajo aborda el aprendizaje de representaciones de dinámica de los robots con redes neuronales y pone de manifiesto la importancia de incorporar la dinámica específica del robot a la hora de aprender las políticas de navegación para los robots zoomórficos, así como los retos que plantea la generalización de dichas políticas aprendidas a nuevos robots.

(Pei et al., 2020) abordó el problema de locomoción para un robot cuadrúpedo en terrenos desconocidos y no estructurados utilizando el algoritmo DDPG. El sistema de aprendizaje propuesto incorpora redes actor-crítico en el esquema de RL sin modelo. El diseño de la función de recompensa contempla el conocimiento en el área de control con el fin de aumentar la eficiencia del aprendizaje durante el entrenamiento. Por último, el controlador basado en DDPG se emplea en un robot cuadrúpedo y su rendimiento se valida en simulación midiendo el promedio de la recompensa por episodio de simulación.

(Luo et al., 2020) desarrollaron un agente cuadrúpedo controlable basado en datos y en la física, adoptando la capacidad de interactuar físicamente con los entornos dinámicos y los movimientos naturales aprendidos a partir de la imitación de clips de movimiento de referencia. Se trata de un proceso de tres etapas que comienza con el aprendizaje de los movimientos de referencia mediante la imitación. El agente aprende a mapear los controles de alto nivel del usuario, como la velocidad y el rumbo, en acciones conjuntas con redes adversariales generativas (GAN). Mediante DRL, el agente adquiere la capacidad de adaptarse y recuperarse de escenarios no vistos, lo que nos permite sintetizar reacciones significativas frente a perturbaciones externas.

3.2. Conclusiones

En este capítulo se presentaron los diferentes trabajos encontrados en la literatura, los cuales abordan el problema de generar patrones de marcha en robots cuadrúpedos. Se observa que se han realizado muchos esfuerzos para lograr el control de locomoción de un robot cuadrúpedo, por lo cual, este paradigma se presenta como una problemática resuelta, ya que hay ciertos trabajos que reportan el logro de aprendizaje de patrones de marcha. Sin embargo, estos enfoques se centran en optimizar trayectorias de las patas o puntos de contacto del robot, o bien, muy pocos trabajos presentan solución cuando no se conocen dichas trayectorias. Por otra parte, no realizan un análisis para conocer que variables físicas impactan en el descubrimiento de patrones de marcha y optimizan el entrenamiento. Por último, ya que este tipo de robots son costosos, existen muy pocos trabajos de aprendizaje realizados directamente en el robot físico.

En el siguiente capítulo se presenta la metodología propuesta para este trabajo de tesis, así como los sistemas de medición de la plataforma robótica utilizada.

Metodología

En el presente capítulo se abordan los aspectos metodológicos que han orientado este trabajo de investigación para lograr la implementación del área de IA en la robótica y realizar la tarea de aprendizaje de locomoción de un robot cuadrúpedo.

4.1. Diseño Metodológico

Se proponen las siguientes etapas metodológicas para el desarrollo de la presente investigación como se presenta en la figura 4.1:

1. Construcción de la plataforma robótica:

Para realizar este estudio se requiere de un robot cuadrúpedo que cuente con la instrumentación necesaria para obtener los datos del entorno y del mismo robot en todo momento. Esta plataforma robótica se tomó de un proyecto de código libre que solo aporta los modelos gráficos en 3D. Esta primer etapa de la metodología consiste en manufacturar el robot en impresión 3D y dotarlo de sensores y toda la electrónica necesaria para obtener la información tanto de las posiciones del robot como del entorno en el cual interactúa, además de realizar modificaciones al diseño de ser necesario.

2. Identificación de algoritmos de aprendizaje por refuerzo:

Ya que la presente investigación se aborda desde el enfoque de extremo a extremo, se requiere investigar sobre los algoritmos de aprendizaje profundo por refuerzo que logren mapear los estados y observaciones de todo el entorno hacia las acciones del robot y lograr controlar el sistema robótico dependiendo de sus entradas. En esta etapa se debe seleccionar un algoritmo de aprendizaje por refuerzo profundo y realizar pruebas con diversas arquitecturas de redes neuronales para identificar cuál de estas se desempeñan mejor en la tarea propuesta.

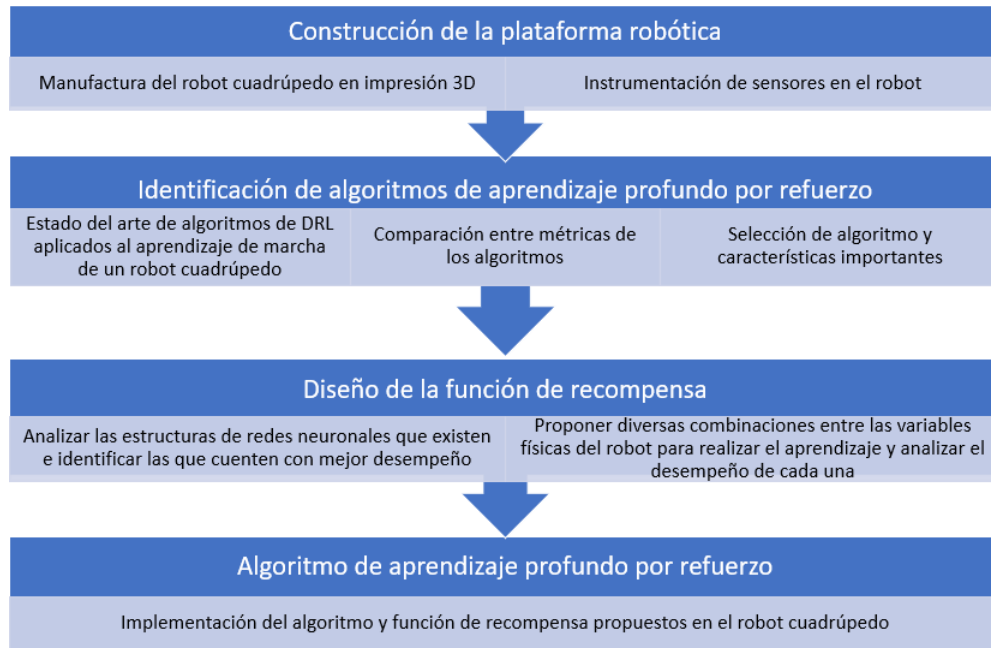


Figura 4.1: Metodología propuesta.

3. Diseño de la función de recompensa:

El diseño de una función de recompensa es sin duda una parte importante, ya que define en cierta forma cómo el agente inteligente aprende. Por lo tanto, en esta etapa se plantea un análisis de qué características son de mayor importancia y que conllevan a un rápido y óptimo aprendizaje. Esta función nos ayudará a comprobar qué variables físicas obtenidas por los sensores aportan información relevante para una optimización del aprendizaje y permiten el control del robot ya sea en tarea de balance o que emerjan patrones naturales de marcha.

4. Algoritmo de aprendizaje profundo por refuerzo:

Por último, se integrarán los pasos anteriores y se explorarán los diversos algoritmos de aprendizaje profundo por refuerzo del estado del arte, ya que al realizar esta metodología se espera encontrar una mejora en los tiempos de aprendizaje y la calidad de la recompensa. Se analizan y comparan diversos tipos de redes, así como también se diseñan los escenarios de entrenamiento, definiendo las tareas que el robot deberá aprender.

4.2. Población

El presente trabajo es un caso de estudio en el que se planea realizar experimentaciones con una plataforma robótica (figura 4.2). Se implementan técnicas de aprendizaje por refuerzo, el cual forma parte de un paradigma de aprendizaje que difiere del supervisado y

no supervisado, ya que no utiliza una base de datos para su entrenamiento. Por otro lado, en el RL se va construyendo el conocimiento a partir de la interacción con el sistema y su entorno, por lo que puede considerarse que su base de datos se va formando al momento que se realizan los experimentos.

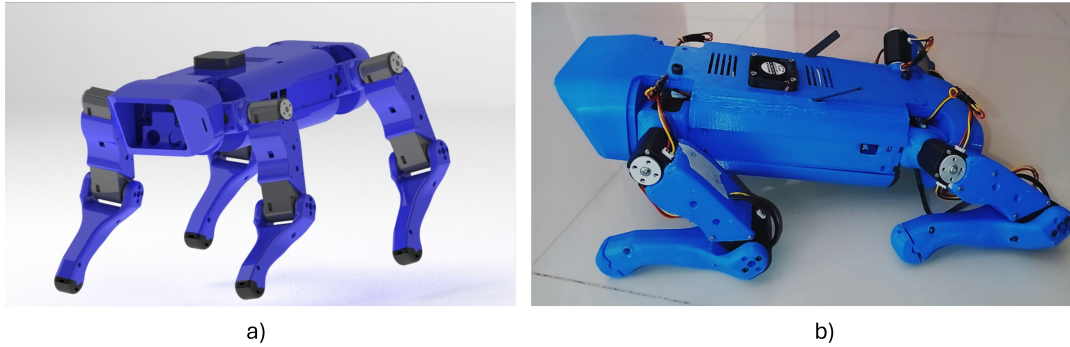


Figura 4.2: Plataforma Robótica (Robot Caudrúpedo): a) Diseño CAD, b) Robot ensamblado.

4.3. Muestra

Debido a que esta investigación es un caso de estudio, la muestra se restringe a las observaciones que se realicen de la plataforma. El aprendizaje por refuerzo sobresale por no necesitar un compendio extenso de casos u observaciones previas para poder realizar una generalización del fenómeno que ocurre, por lo que las muestras se limitarán a una serie de experimentos que se realizarán en el robot físico o en simulación y los datos a recopilar serán los voltajes, amperajes y posiciones de cada actuador, así como también las mediciones de distancias de avance del robot, altura del suelo y contactos de las patas con el suelo. Por lo tanto, la muestra se define por el tipo de algoritmo de RL y el análisis que se realizará de cada observación para conocer cuál de todas las variables son primordiales o relevantes para el aprendizaje de las tareas por parte del agente inteligente que controlará el robot.

4.4. Instrumentos de Medición

Para realizar la medición de las variables y fenómenos físicos necesarios para realizar la investigación se cuentan con diversos sistemas electrónicos que permiten realizar las observaciones pertinentes. De igual manera, se cuenta con una unidad central de adquisición y procesamiento basado en una tarjeta integrada del fabricante NVIDIA, la cual se destaca por tener una GPU integrada para realizar operaciones de punto flotante más rápido que un procesador. En la figura 4.3 se detalla el diagrama de instrumentación de la plataforma experimental (Robot Cuadrúpedo).

Para medir el contacto en las extremidades del robot se modificó el diseño de la extremidad inferior (como se presentará en el capítulo 5) para dotarlo de sensores y componentes necesarios.

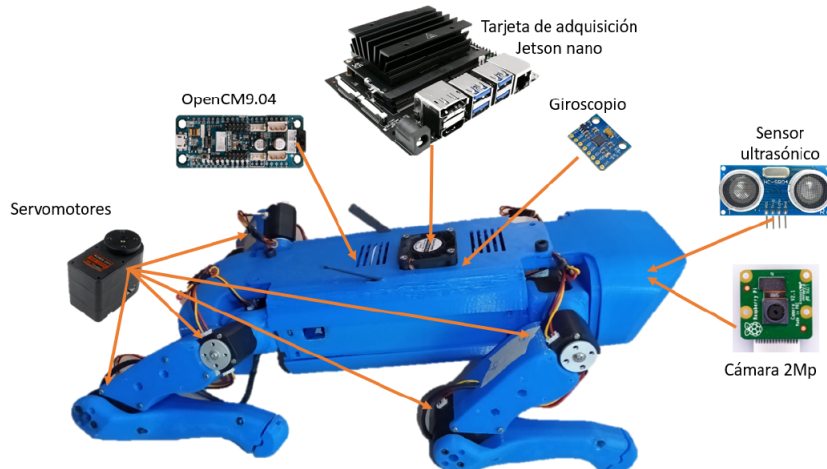


Figura 4.3: Diagrama de instrumentación del sistema robótico cuadrúpedo.

Para medir el avance o posición de cada actuador, se implementaron sensores que permiten retroalimentar la posición, corriente, voltaje y temperatura. Este tipo de servomotores, ver figura 4.4, se comunican con un protocolo especial que consta de 3 cables, el voltaje de alimentación es de 12V.



Figura 4.4: Servomotores marca DCServo.

En la plataforma robótica se desea medir la inclinación o elevación del robot cuadrúpedo. Para este fin se cuenta con un sensor inercial IMU MPU6050, ver figura 4.5, el cual puede realizar mediciones en 6 GDL, ya que combina un giroscopio de 3 ejes y un acelerómetro de 3 ejes en el mismo chip. Es útil para sistemas de control, medición de vibración, sistemas de medición inercial (IMU), detección de caídas, sensor de distancia y velocidad o goniometría (medición de ángulos). Este sensor se comunica a través de la interfaz I2C y cuenta con un regulador de tensión a 3.3V.

Por último, para la integración de todas las mediciones realizadas y para el control del sistema por medio de técnicas de aprendizaje profundo y por refuerzo, se utilizó una placa



Figura 4.5: Sensor inercial para medir avance e inclinación del robot en el entorno.

integrada Jetson Nano del fabricante NVIDIA, ver figura 4.6. Esta placa está diseñada para máquinas autónomas, de bajo consumo y asequible con un alto nivel de potencia informática que permite realizar operaciones de visión por computadora en tiempo real y de aprendizaje profundo a nivel móvil. Esta placa integrada será el centro de adquisición de datos, análisis de la investigación y a su vez servirá como controlador del sistema robótico.



Figura 4.6: Placa integrada Jetson NANO de la familia NVIDIA.

4.5. Conclusiones

En el presente capítulo se abordaron los aspectos metodológicos de este trabajo de investigación para lograr la implementación del área de IA en la robótica y realizar la tarea de aprendizaje de locomoción de un robot cuadrúpedo. El desarrollo del robot cuadrúpedo se realizó en manufactura aditiva, impresión 3D, lo cual representa bajo costo en comparación con los robots vistos en otros trabajos de investigación. Además, se presentaron los instrumentos de medición para dotar al robot de sentidos propioceptivos, se contemplaron estos instrumentos de medición por su bajo costo y facilidad de compra en nuestra ciudad.

Resultados

5.1. Diseño y construcción de Robot Cuadrúpedo

El diseño, construcción y puesta en marcha de un robot, en este caso del robot cuadrúpedo, es un proceso complejo, pero de gran importancia, ya que requiere la integración de varios componentes electrónicos y mecánicos que permitan su implementación y funcionamiento adecuados. Esta primera parte se puede utilizar como guía para que los lectores sigan y repliquen el proceso de diseño, construcción y puesta en marcha de un robot cuadrúpedo.

5.1.1. Diseño del robot cuadrúpedo

El diseño de la plataforma robótica se tomó de un proyecto de código libre que solo aporta los modelos gráficos, (D.-y. Kim, 2021), el cual está protegido bajo licencia, (4.0, 2021). En la figura 5.1 se presenta el diseño en CAD del robot cuadrúpedo descargado.

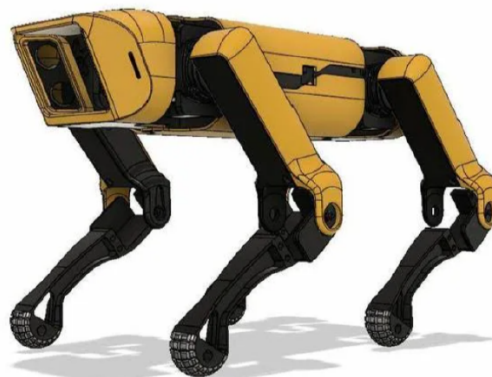


Figura 5.1: Diseño CAD de robot cuadrúpedo.

Sin embargo, ya que este modelo está diseñado para ser implementado con servomotores *MG 996 R*, se tuvo que modificar el diseño de las extremidades del robot para acoplar los actuadores propuestos para este trabajo (*ROBS-251*), los cuales se detallan más adelante. Por lo tanto, se modificó el diseño completo de las extremidades del robot cuadrúpedo, comenzando por la cadera o parte superior de las patas, esta pieza ensambla completamente la extremidad con el torso de robot y le otorga su primer grado de libertad. Se observa en la figura 5.4 (a) la pieza diseñada y en 5.4 (b) el ensamble en conjunto con el servomotor.

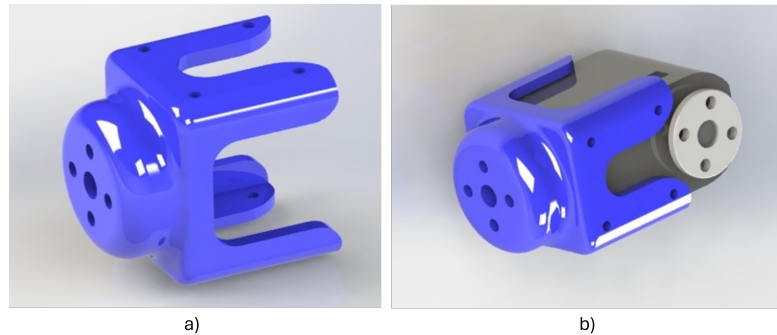


Figura 5.2: Diseño CAD de la parte superior de las patas del robot, la cual conecta toda la extremidad con el torso del robot.

La parte media de la extremidad del robot esta conformada por una pieza que une 2 motores, esta pieza se diseñó con un compartimiento en la parte interna, con el motivo de alojar el excedente de cableado por parte de los 2 servomotores, esta pieza representa los grados de libertad que realizan el movimiento de flexión y extensión de cada extremidad. En la figura 5.4 (a) se presenta la pieza diseñada y su ensamble con los 2 servomotores, y en 5.4 (b) se aprecia el compartimiento que resguarda el excedente de cable que resulta de la conexión de los 2 servomotores.

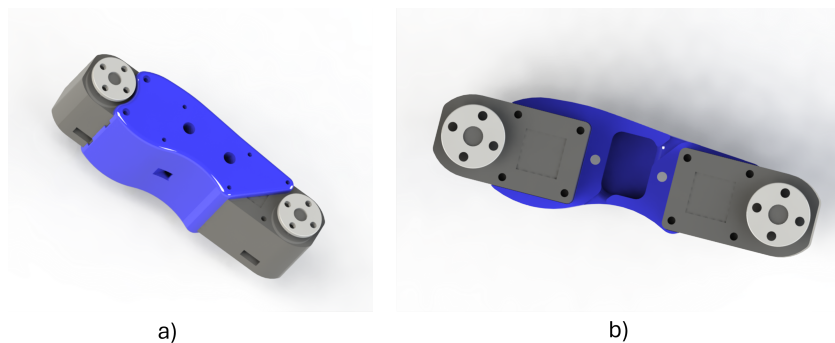


Figura 5.3: Diseño CAD de la parte media de las patas del robot, donde (a) muestra el ensamble con los 2 GDL permitiendo al robot realizar movimientos de flexión y extensión en cada extremidad, en (b) se presenta el compartimiento interior para alojar cables derivados de las conexiones entre servomotores.

En la figura 5.4 se hace una comparación del diseño original de una de las extremidades del robot contra el diseño modificado, este último ya contempla las dimensiones requeridas para alojar a los nuevos motores. Cuenta con un diseño más sólido y resistente, ya que se contempla fijar el motor por ambos lados y no solamente de uno como en el diseño anterior. Dentro de estas piezas se consideró un espacio para alojar el sensor de contacto con el suelo, que consta de un límite de carrera.

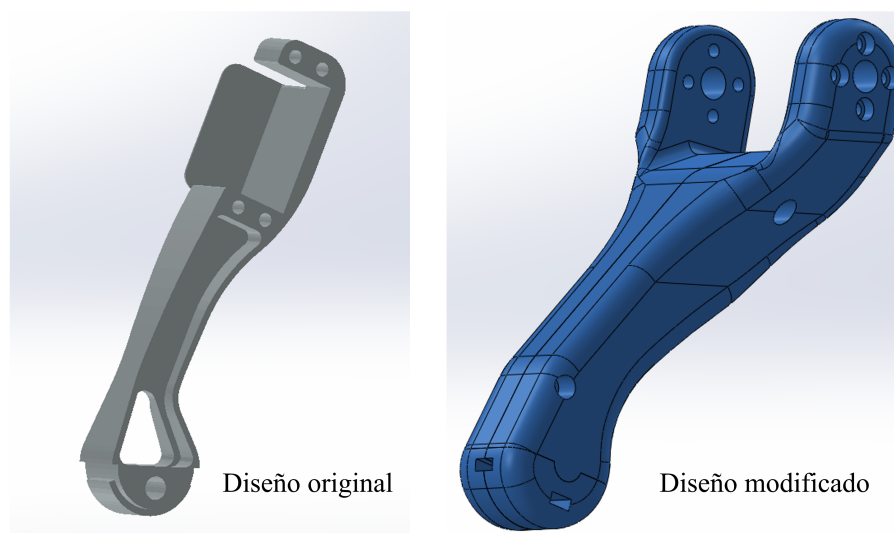


Figura 5.4: Diseño original vs diseño modificado para extremidades del robot.

En la figura 5.5 se observa el diseño de la pata donde se colocará un sensor que se activará al hacer contacto con el suelo, el cual queda en la parte interna de la pierna. Esta pieza del robot se diseñó en 2 partes con la finalidad de poder esconder los cables y sensores.

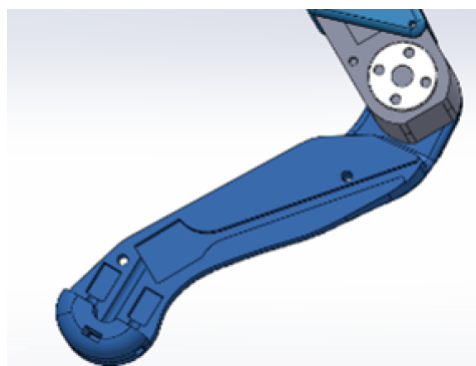


Figura 5.5: Diseño de la pata para colocar el sensor de contacto.

Se modificaron las tapas frontal y trasera, ya que en la frontal se optó por dotar al robot de un sensor de ultrasonido y una cámara de 2 Mp, por lo que se realizaron las

modificaciones para esta integración, como se observa en la figura 5.6 (a). Para la parte posterior del robot se modificó la cubierta para albergar un sensor de voltaje y corriente, así como el interruptor de prendido y apagado de la energía general, como se observa en la figura 5.6 (b).

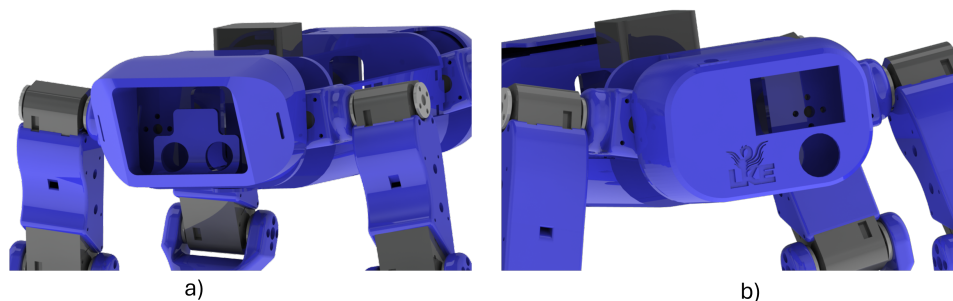


Figura 5.6: Diseño de las cubiertas frontal (a) y posterior (b), donde se albergará un sensor de ultrasonido en conjunto con una cámara de 2 Mp y un sensor de voltaje junto con el interruptor general respectivamente.

Se diseñó la tapa superior que cubre la electrónica del robot, esta debe contemplar el ventilador de la tarjeta de desarrollo Nvidia Jetson Nano, ya que sobresale del espacio determinado para la electrónica, también se contemplaron rendijas para ventilación de todo el sistema, ver figura 5.7 (a). La electrónica debe ir montada dentro del torso del robot, por lo que se diseñó una placa interna que va fija en las barras laterales del torso, como se presenta en la figura 5.7 (b).

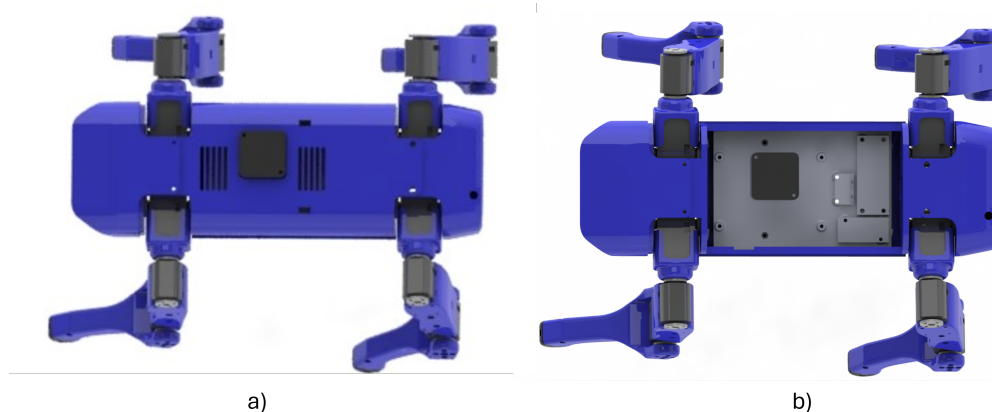


Figura 5.7: Diseño de la tapa superior (a) y placa donde se fija la electrónica dentro del torso del robot (b).

Por último, se realizó el ensamble completo con todas las piezas modificadas, integrando al diseño anterior como se presenta en la figura 5.8.

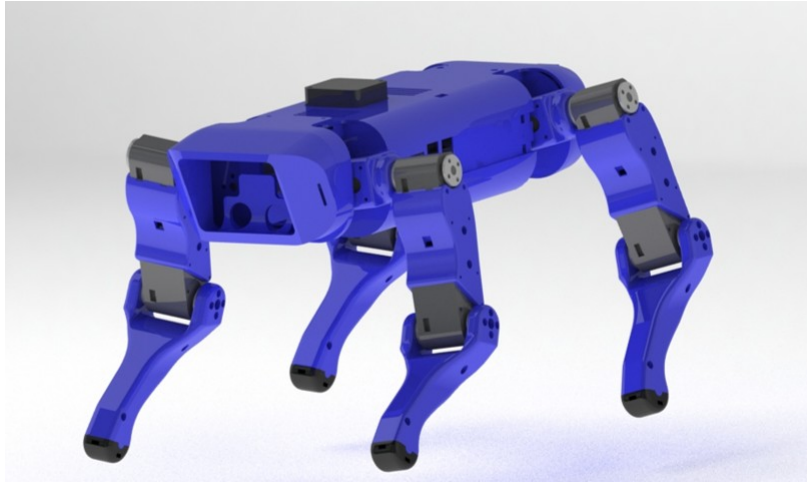


Figura 5.8: Diseño CAD del robot cuadrúpedo completo, integrando los nuevos diseños y modificaciones.

5.1.2. Instrumentación

Se dotó al mecanismo con motores de arquitectura abierta que proporcionan información relevante como posición, par, velocidad y temperatura. Los servomotores *ROBS-251* (figura 5.9) son una opción de bajo costo entre los diversos servomotores de tipo compacto en el mercado. Este tipo de servomotores se puede alimentar con 12V y alcanza un par de 25kg.cm a una velocidad máxima de $360^\circ / 0.86$ segundos, lo que es completamente compatible para la aplicación que se le desea dar a la plataforma robótica. Además, es importante señalar que los servomotores se pueden conectar en forma de cadena y así reducir el cableado, ya que funcionan con un protocolo por IDs donde cada motor se etiqueta con un valor y así es como se puede obtener información de la articulación deseada.



Figura 5.9: Servomotores de arquitectura abierta.

Se implementó un sensor tipo fin de carrera en cada pata para medir cada extremidad del robot al momento de hacer contacto con el suelo, ver figura 5.10a. De igual manera, se utiliza un sensor ultrasónico como sensor de proximidad (figura 5.10b) y una cámara Módulo 2 de Raspberry Pi (figura 5.10c) que incluye un sensor *Sony IMX219* de 8Mp para

el reconocimiento de imagen; estos podrían considerarse los ojos del robot para tareas de detección y/o evasión de obstáculos.

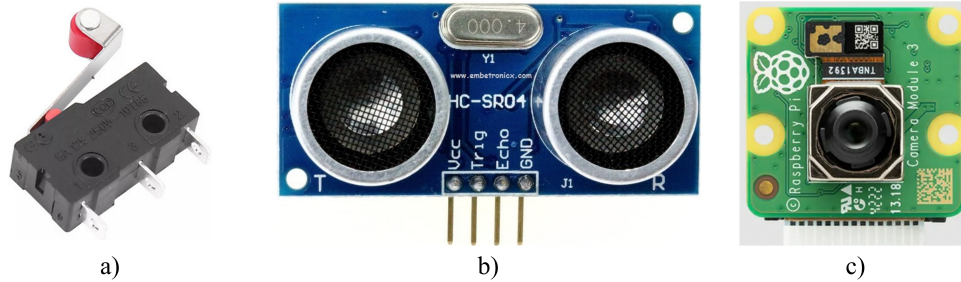


Figura 5.10: Componentes para el robot: a) sensor fin de carrera, b) sensor ultrasónico y c) cámara 2Mp.

En este caso, como se utilizará aprendizaje por refuerzo profundo es necesario contar con una plataforma que tenga la capacidad de ejecutar modelos de IA. Por otra parte, debido a que se carga un sistema operativo genérico, la tarjeta de desarrollo no es capaz de comunicarse con los motores en tiempo real. Debido a los procesos que corre el sistema operativo, no se puede confiar en que la comunicación con los servomotores sea implementada correctamente, ya que requiere de tiempos muy pequeños del orden de microsegundos. Por tal motivo, se decidió utilizar un microcontrolador *OpenCM9.04*, el cual se basa en un microprocesador ARM Cortex-M3 de 32 bits.

Los esquemas y códigos fuente de *OpenCM9.04* son de código abierto y se puede programar con el IDE de Arduino, lo que facilita su implementación. En la figura 5.11 se presenta la placa basada en microcontrolador. Esta placa sirve para comunicarse con los servomotores de la marca *DCServo*, ya que tienen el mismo protocolo que los motores Dynamixel.

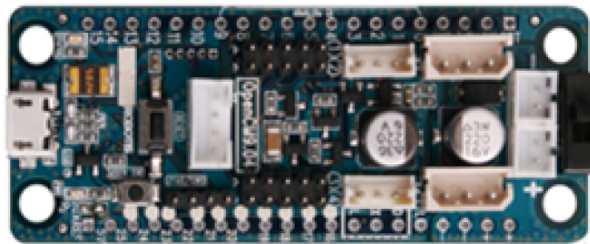


Figura 5.11: Placa basada en microcontrolador de 32 bits para el control de bajo nivel de los servomotores y adquisición de información de sensores ultrasónicos.

La plataforma se alimentó con una batería de iones de litio con capacidad de 12V y 5.6Ah (figura 5.12a), y se implementó un voltímetro amperímetro digital (figura 5.12b) para tener la información visual de voltaje y corriente del robot a través de un panel digital.

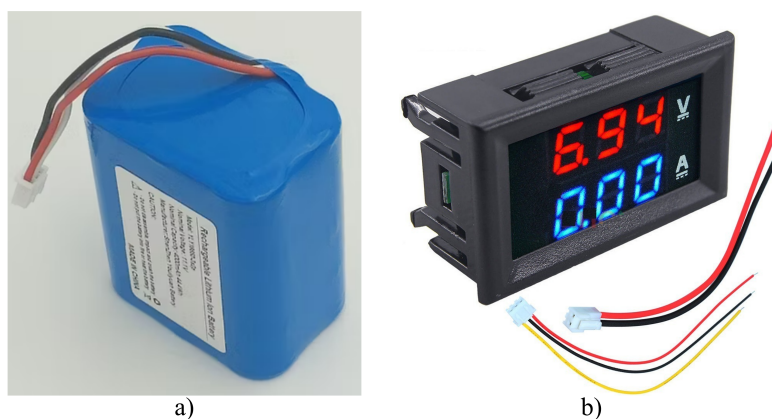


Figura 5.12: Componentes: a) batería de litio y b) voltímetro amperímetro digital.

Por último, la plataforma robótica se controla con una tarjeta de desarrollo Jetson nano del fabricante Nvidia (figura 4.6), ya que esta tarjeta tiene el poder de procesamiento para ejecutar el sistema operativo Ubuntu 20.04, que servirá para realizar las tareas de alto nivel necesarias, por ejemplo, el procesamiento de las imágenes u órdenes que se le den al robot. Es útil para la integración de todas las mediciones realizadas y para el control del sistema por medio de técnicas de aprendizaje profundo y por refuerzo.

5.1.3. Construcción del robot

La plataforma robótica desarrollada, que contiene los sensores necesarios para obtener información del ambiente en el que se desempeñaron los experimentos, se considera de bajo costo ya que se manufacturó en impresión 3D, lo que supone una ventaja al momento de realizar modificaciones en la estructura del mismo. La manufactura de las piezas del robot se realizaron en plástico PLA para reducir el peso de toda la estructura, ver figura 5.13.



Figura 5.13: Manufactura de piezas en PLA.

En cada extremidad se colocó un sensor de límite de carrera y un pequeño vástago que se conecta a la punta de la pata, de modo que cuando esta presiona el suelo el vástago acciona el límite de carrera. También se le agregaron dos resortes para empujar la parte móvil de la punta de la pata y así tener el comportamiento deseado, ver figura 5.14.



Figura 5.14: Extremidad inferior con sensor de límite de carrera.

En conjunto con los servomotores y el sensor de contacto, el ensamble de cada pierna se presenta en la figura 5.15, donde se aprecia el cable de conexión del sensor de límite de carrera, así como las conexiones de los motores.

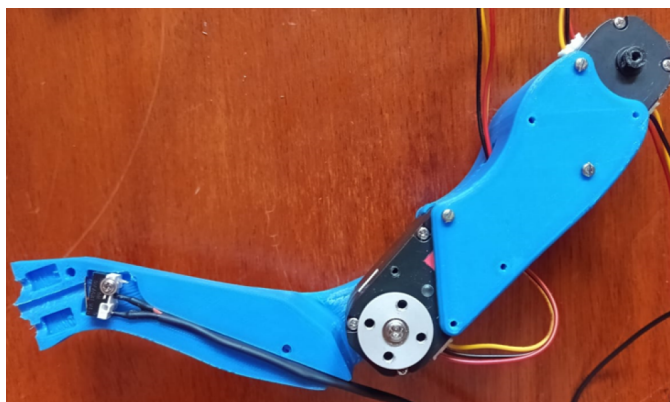


Figura 5.15: Ensamble de pierna implementando sensor de contacto y servomotores.

Se instalaron los componentes electrónicos dentro del torso del robot y se realizó el cableado correspondiente, como se presenta en la figura 5.16.

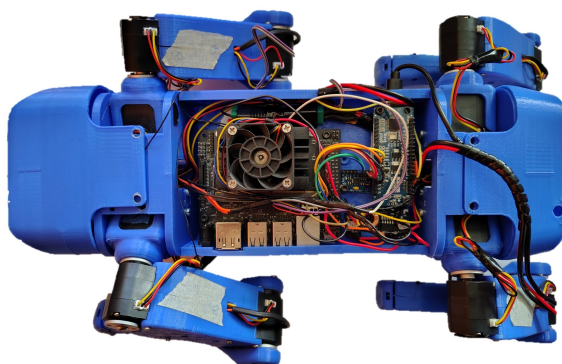


Figura 5.16: Electrónica del robot instalada dentro del torso.

El sensor ultrasónico y la cámara se colocaron en la parte frontal del robot (figura 5.17a) para medir el avance o la distancia hacia algún objeto, y el panel digital se colocó en la parte trasera del robot, ver figura 5.17b, para visualizar el rendimiento de la batería. Finalmente, en la figura 5.17 se presenta la manufactura y ensamble completo de la plataforma robótica.

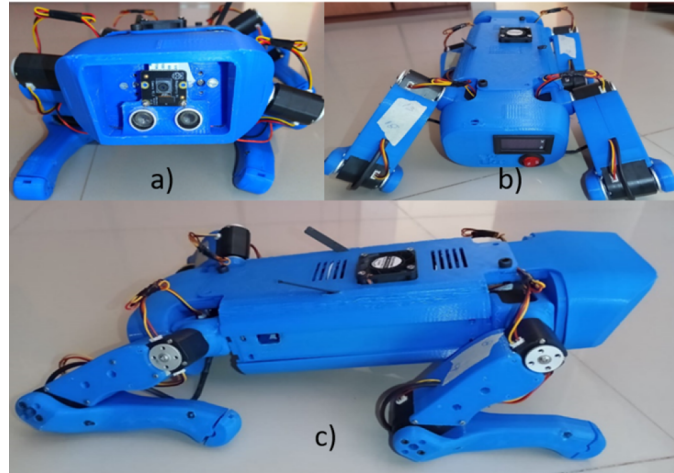


Figura 5.17: Plataforma robótica implementada: a) vista frontal, b) vista trasera y c) vista lateral.

5.2. Aprendizaje de controlador PID para balance con RNA

En este apartado, se presentan los resultados de dos experimentos realizados como parte de una estancia de investigación para analizar el comportamiento de un controlador PID en la tarea de balance del cuerpo en robots cuadrúpedos. El primer experimento consiste en la implementación de un robot cuadrúpedo llamado Robot Dog Kit de la empresa Freenove (Freenove, 2016), que fue utilizado para extraer el comportamiento del controlador PID en diferentes condiciones experimentales. Los resultados obtenidos fueron utilizados como base para el segundo experimento, que consistió en la exploración de diferentes arquitecturas de redes neuronales para replicar el comportamiento del controlador PID.

La implementación del robot cuadrúpedo de Freenove permitió extraer un conjunto de datos significativos sobre el comportamiento del controlador PID para posteriormente realizar la exploración de diferentes arquitecturas de redes neuronales para replicar el comportamiento del controlador PID. Los resultados obtenidos utilizando redes neuronales mostraron una buena capacidad para replicar el comportamiento del controlador PID, lo que sugiere que estas redes pueden ser utilizadas como alternativa a los controles tradicionales en la tarea de balance del torso.

5.2.1. Plataforma Freenove Robot Dog

Como parte de una estancia doctoral se trabajó con un robot cuadrúpedo llamado *Robot dog kit* fabricado por la empresa Freenove (Freenove, 2016), ver figura 5.18. Se trata de

un robot cuadrúpedo con servomotores (mg90) para control por posición en grados y de la electrónica necesaria para que el robot opere con baterías de iones de litio y con una raspberry pi 4 para el control de los motores. Este robot cuenta con funciones predefinidas como avance hacia el frente, hacia atrás, paso hacia los lados y giro en ambas direcciones. Esto lo logra utilizando la cinemática inversa del mecanismo para posteriormente calcular la trayectoria que deben seguir cada una de sus extremidades. Sabemos que esta tarea conlleva al análisis del mecanismo, sus dimensiones y tipo de morfología.

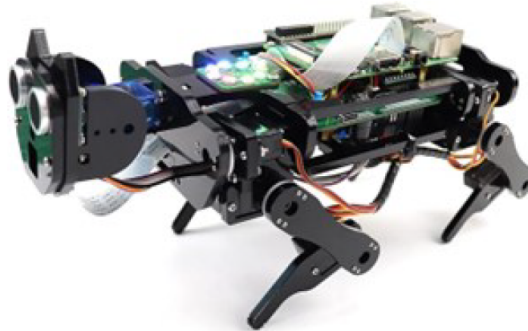


Figura 5.18: Plataforma Robótica Robot Dog Kit desarrollada por Freenove.

El Robot Dog Kit consta de piezas de acrílico fabricadas en corte laser, tornillos, circuitos electrónicos y herramientas que se requieren para su ensamblaje, como se observa en la figura 5.19

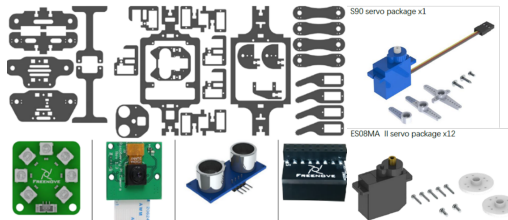


Figura 5.19: Componentes y piezas de acrílico que conforman el Robot Dog Kit.

Aunque el kit consta de todo lo necesario para ensamblar, no cuenta con herramientas que permitan operarlo como lo son el módulo de control y las baterías. El módulo de control consta de una Raspberry pi 4 y las baterías deben de cumplir con la característica de alta descarga, el valor mínimo que recomienda la empresa Freenove es de 10 Amperes. El robot cuenta con un repositorio para descargar en su página web, donde se encuentran las instrucciones de ensamblaje y el software de control. Este sistema cuenta con una configuración inicial de ensamble muy peculiar que nos va a ayudar a tener una referencia de los servomotores para posteriores aplicaciones. Una vez realizada la mayor parte del ensamblaje, se requiere que los servomotores se encuentren encendidos para poder posicionarlos de manera adecuada, esto es en la posición de 90°, como lo presenta la figura 5.20,

CAPÍTULO 5. RESULTADOS

por lo que fue necesario instalar el sistema operativo con el que se trabajará y ejecutar un script de Python.

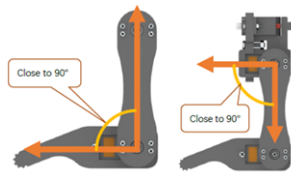


Figura 5.20: Posicionamiento de cada motor a 90° para un correcto ensamblaje.

El sistema operativo que se instaló en la raspberry pi 4 fue Raspberry Pi OS with Desktop.^{en} su versión de 32 bits, una versión de Debian 11 (Bullseye). Se utilizó la herramienta "install manager tool" propia de Raspberry, como se observa en la figura 5.21. Se instalaron los paquetes para continuar el ensamblaje.



Figura 5.21: Herramienta "install manager tool" propia de Raspberry.

Freenove proporciona una aplicación para controlar el robot a distancia por medio de una conexión TCP/IP, por lo que se descargó del repositorio Git dicha aplicación, ver figura 5.22. Con ayuda de esta aplicación se pueden realizar varias tareas con el robot, las cuales son: caminar en diferentes direcciones, realizar streaming de video, controlar el balance del robot, controlar la altura y funciones especiales para reconocimiento de personas y perseguir una pelota roja. Todas estas funciones las realiza con un modelo matemático previamente calculado y está limitado a los parámetros físicos que tiene este robot en específico, es decir, que ningún otro robot podría manejar las mismas funciones ya que cambian los parámetros físicos como longitudes de extremidades y configuraciones.

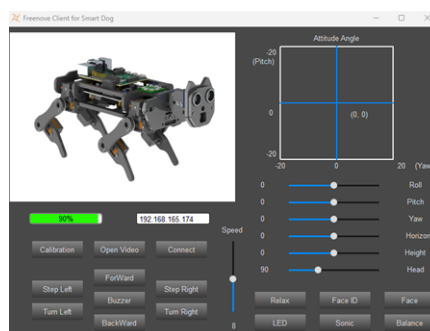


Figura 5.22: Aplicación para control remoto vía TCP/IP del robot.

5.2.2. Data set del control PID de balance

La función control de balance que viene configurada en el robot dog kit, brinda la capacidad al robot de reaccionar cuando está en un plano inclinado, compensando esa inclinación con el movimiento de sus patas para que el torso del robot en todo momento esté de forma horizontal logrando así balancear el sistema. La acción de balanceo involucra un sensor llamado unidad de medida inercial (IMU) (ver figura 4.5), este sensor combina un acelerómetro y giroscopio de tres ejes. El acelerómetro mide la aceleración lineal en las direcciones X, Y y Z, lo que incluye la aceleración debido a la gravedad, permitiendo así determinar la inclinación o el ángulo de orientación del dispositivo. Por otro lado, el giroscopio mide la velocidad angular en las mismas direcciones, proporcionando información crucial sobre rotaciones y movimientos angulares. Además, el sistema cuenta con ecuaciones matemáticas que detallan la orientación del sistema por medio de una unidad de medida inercial, así como filtros de Kalman y técnicas que permiten obtener los ángulos de Euler (Roll, Pitch, Yaw).

Para lograr el control de balance, se utiliza un controlador PID que pretende manipular la variable de los ángulos de Euler anteriormente mencionados, y así, con ayuda de matrices de rotación, filtros de Kalman y la morfología del robot pueden lograr el control de la orientación del sistema. Cabe destacar que también se hace uso de la cinemática inversa para conocer los valores que se le deben de proporcionar a los motores en formato de ángulo.

Sabemos que para lograr lo anterior mencionado, es necesario contar con un análisis previo del mecanismo, su morfología, así como el modelo matemático en este caso la cinemática directa y la cinemática inversa para poder calcular en qué medida las extremidades del robot pueden compensar la inclinación. Por lo que se planteó lograr que una red neuronal aprendiera el comportamiento del controlador PID sobre las variables de los ángulos Euler que proporciona la IMU, y así, realizar experimentos para entender qué importancia tiene el balance en el aprendizaje de patrones de marcha.

Se procedió a adquirir una base de datos del comportamiento del controlador PID sobre los motores, esto es, se requiere extraer el comportamiento que tiene el PID través de todas las mediciones posibles y las acciones que los motores generan para compensar los movimientos de inclinación que registran el sensor. Se necesitó que el robot estuviera en una plataforma donde se realizaran movimientos de inclinación para obtener las mediciones del sensor y la acción del controlador PID en planos inclinados, por lo que se puso el robot en una plataforma y a través del movimiento de la plataforma, como se observa en la figura 5.23, se registraban los valores en cada escenario.

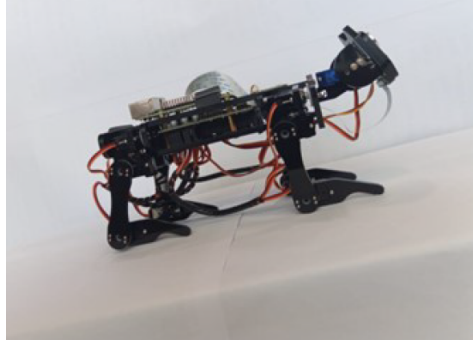


Figura 5.23: Robot cuadrúpedo en plataforma de inclinación.

Las variables que se adquirieron constan de la acción de los 12 motores, las mediciones sin procesamiento de (x, y, z) del sensor IMU, los 3 ángulos de Euler (Roll, Pitch, Yaw) y las 12 salidas anteriores de los motores, estos son los datos que se guardaron en la base de datos. Los movimientos que se realizaron para extraer el comportamiento del controlador PID se presentan en la figura 5.24, donde (a) y (b) presentan el movimiento de inclinación positivo y negativo del ángulo Pitch respectivamente, (c) y (d) el movimiento de inclinación positivo y negativo en el ángulo Roll, respectivamente.

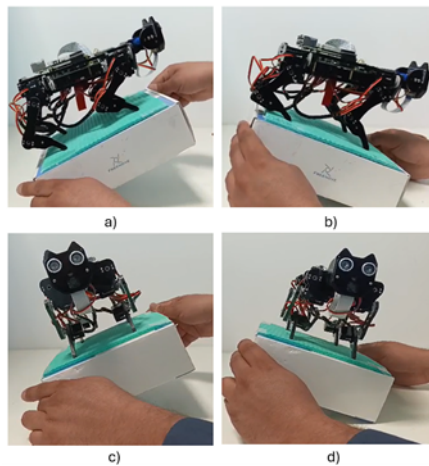


Figura 5.24: Movimientos realizados para extraer la base de datos, a) y b) movimiento en Pitch, c) y d) movimiento de Roll.

Se generaron 14 archivos, ver figura 5.25, con 4,100 muestras aproximadamente cada uno, con los cuales se construyó una base de datos que consta de 49,876 datos. Para la base de datos no se contempló el archivo *Data acquisition 5.csv* ya que se encontraron valores fuera de rango. El archivo *Data acquisition 8.csv* se excluyó para efectos de validación, el cual consta de 4,067 datos.

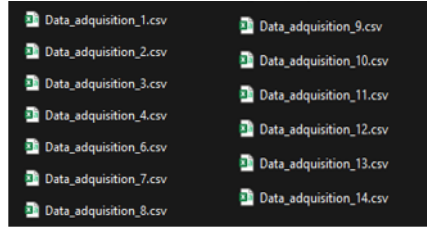


Figura 5.25: Archivos generados que conforman la base de datos utilizada para el entrenamiento de los modelos de redes neuronales.

En la figura 5.26 se observan los tipos de datos adquiridos, donde (mXX_o) es el valor anterior del motor, (x, y, z) los valores sin procesar del sensor IMU y (mXX) es el valor actual que deben aplicar los motores.

m00_o	m01_o	m02_o	m10_o	m11_o	m12_o	m20_o	m21_o	m22_o	m30_o	m31_o	m32_o	x	y	z	m00	m01	m02	m10	m11	m12	m20	m21	m22	m30	m31	m32
76	139	102	94	131	103	91	53	78	97	34	63	0.359	-12.069	0.916	76	138	101	94	130	102	91	52	77	97	34	62
76	138	101	94	130	102	91	52	77	97	34	62	1.389	-14.542	4.534	76	138	100	94	130	101	91	52	76	97	33	62
76	138	100	94	130	101	91	52	76	97	33	62	2.641	-16.931	6.015	76	137	100	94	130	101	91	52	76	97	33	61
76	137	100	94	130	101	91	52	76	97	33	61	2.412	-18.397	7.221	76	137	99	94	129	100	91	51	76	97	32	61
76	137	99	94	129	100	91	51	76	97	32	61	2.779	-18.733	9.565	76	137	99	94	129	100	91	51	75	97	32	60
76	137	99	94	129	100	91	51	75	97	32	60	3.046	-14.824	12.824	76	137	98	94	129	99	91	51	74	97	32	59
76	137	98	94	129	99	91	51	74	97	32	59	2.580	-14.038	14.878	76	137	98	94	129	99	91	51	74	97	32	59
76	137	98	94	129	99	91	51	74	97	32	59	3.084	-14.145	16.473	76	137	98	94	129	98	91	51	74	97	32	59
76	137	98	94	129	98	91	51	74	97	32	59	3.137	-13.962	17.756	76	137	97	94	129	98	91	51	73	97	32	58
76	137	97	94	129	98	91	51	73	97	32	58	3.076	-12.756	18.687	76	137	97	95	129	97	91	51	73	97	32	58
76	137	97	95	129	97	91	51	73	97	32	58	4.191	-11.534	19.420	77	137	96	95	129	97	91	51	73	97	32	58
77	137	96	95	129	97	91	51	73	97	32	58	2.626	-13.588	19.939	77	137	96	95	128	97	91	51	73	97	32	58

Figura 5.26: Muestra de datos adquiridos realizando movimientos de inclinación en Pitch y Roll.

5.2.3. Exploración Arquitecturas de Redes Neuronales

Las variables que se adquirieron en el apartado anterior constan de la acción de los 12 motores más 3 entradas del sensor IMU con las que se calculan los ángulos de Euler, estos son los datos que se ocuparon para entrenar el robot. Una vez obtenida la base de datos con la cual se va entrenar la red neuronal, se procedió a utilizar el framework de tensorflow para crear el modelo.

Se planteó, por lo tanto, conocer qué tipo de arquitectura es la conveniente para realizar la tarea de control de balance en el robot y que muestre un aprendizaje óptimo del controlador PID, por lo que se comenzó utilizando una red perceptrón multicapa y se realizaron experimentos para determinar la mejor arquitectura que realice con éxito la tarea de balanceo.

Los experimentos constan en entrenar con la misma base de datos una serie de redes neuronales que varían desde una neurona hasta 20 neuronas, así mismo, se experimentó con una y dos capas ocultas en cada arquitectura. Para la función de activación se utilizó la función *ReLU* y como función de pérdida el *mean squared error*, el cual se utiliza para problemas de regresión. Todos los entrenamientos se realizaron con una época de 50.

CAPÍTULO 5. RESULTADOS

Una vez realizados los experimentos, se seleccionó la mejor arquitectura calculando las métricas de error absoluto medio (MAE) y error cuadrático medio (MSE), con estas métricas se seleccionó el número de neuronas y de capas ideales para tener una arquitectura óptima al realizar el trabajo de aprendizaje.

En la tabla 5.1 se presentan los resultados de las métricas calculadas con los datos de validación y se observa que la arquitectura de dos capas con 14 neuronas cuenta con mejor desempeño, con un valor de MSE de 0.2363 y un MAE de 0.2399. También se observa que existe una red neuronal con una capa y 18 neuronas que cuenta con un desempeño ligeramente menor, con un valor de MSE de 0.26062 y un MAE de 0.2524.

Tabla 5.1: Comparación de métricas MSE y MAE de redes neuronales con diferentes neuronas.

Red neuronal 1 capa			Red neuronal 2 capas		
Neuronas	MSE	MAE	Neuronas por capa	MSE	MAE
18	0.26062	0.25241	14 -14	0.23634	0.23299
17	0.31753	0.30081	8 -8	0.25719	0.25164
14	0.32096	0.30351	16 -16	0.27813	0.27039
19	0.32676	0.29505	17 -17	0.27996	0.27148
10	0.34511	0.30179	10 -10	0.28447	0.27684
13	0.34582	0.31588	12 -12	0.28803	0.28222
12	0.41526	0.37572	4 -4	0.32811	0.29782
11	0.53823	0.39626	18 -18	0.34210	0.32788
5	0.54264	0.40549	11 -11	0.35039	0.33671
4	0.54407	0.41224	13 -13	0.35077	0.31601
16	0.55667	0.43297	6 -6	0.35464	0.32212
15	0.55823	0.40960	19 -19	0.37177	0.34225
8	0.56823	0.43116	5 -5	0.38674	0.35156
3	0.56898	0.43218	9 -9	0.38730	0.36499
9	0.56915	0.42769	15 -15	0.40518	0.37782
6	0.56921	0.43048	20 -20	0.46303	0.42260
7	0.86466	0.61360	3 -3	0.53846	0.42620
20	1.18497	0.76728	7 -7	0.80416	0.60000
2	10.06871	2.16316	2 -2	47.11017	4.66455
1	46.45846	4.66416	1 -1	2278.72757	40.30010

En la figura 5.27 se presenta la evolución de la función de pérdida durante el entrenamiento, en este caso es el promedio del error cuadrático. Se observa que la red neuronal converge aproximadamente en los valores de 5 épocas.

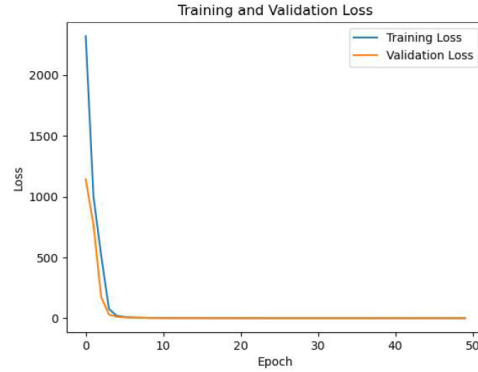


Figura 5.27: Gráfica de la función de pérdida MSE durante el entrenamiento para la red neuronal con 2 capas y 18 neuronas cada capa.

De igual manera, se graficó la métrica de MAE. En la figura 5.28 se observa como converge el modelo aproximadamente a la 5ta época.

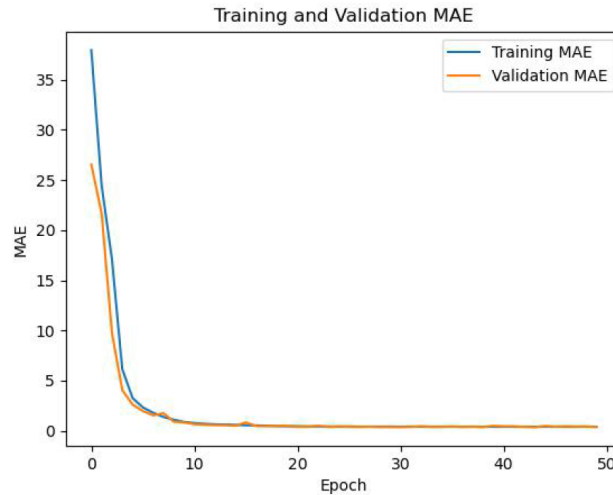


Figura 5.28: Gráfica de la métrica MAE para la red neuronal con 2 capas y 18 neuronas cada capa.

En la figura 5.29 se presenta la gráfica del error en la validación de la red neuronal de 2 capas con 14 neuronas, donde se observan 12 sub gráficas que corresponden a cada uno de los motores del robot cuadrúpedo. Estas gráficas representan el error de la red neuronal al realizar una inferencia y se compara con el valor real que está en el data set.

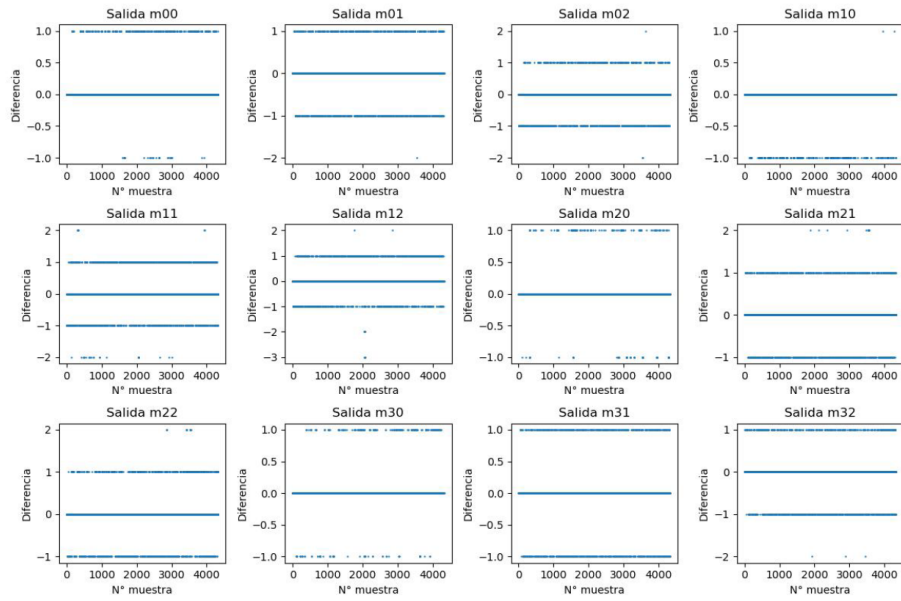


Figura 5.29: Gráfica de diferencia entre el valor real expresada en grados ($^{\circ}$) y la predicción del modelo de 2 capas con 14 neuronas cada una.

En la figura 5.30 se presenta la evolución de la función de pérdida durante el entrenamiento, en este caso es el promedio del error cuadrático. Se observa que la red neuronal converge aproximadamente en los valores de 10 épocas.

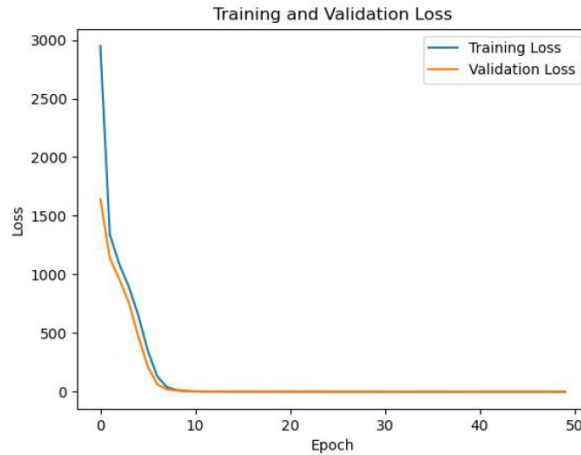


Figura 5.30: Gráfica de la función de pérdida MSE durante el entrenamiento de la red con una capa de 18 neuronas.

Por otra parte, también se realizó la gráfica con una métrica adicional que se desea. En la figura 5.31 se observa cómo converge el modelo aproximadamente a las 20 épocas.

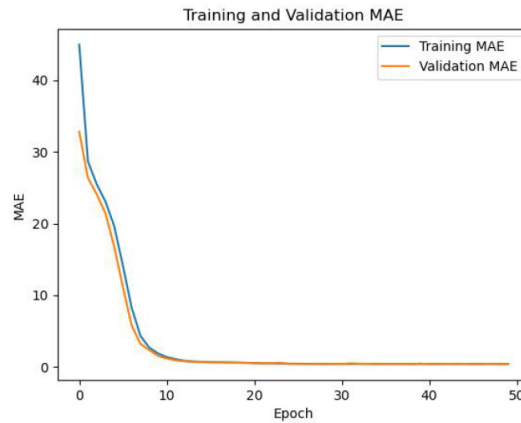


Figura 5.31: Gráfica de la métrica MAE para la red neuronal con 1 capa y 18 neuronas.

En la figura 5.32 se presenta la gráfica del error en la validación de la red neuronal de una capa con 18 neuronas, donde se observan 12 sub gráficas que corresponden a cada uno de los motores del robot cuadrúpedo. Estas gráficas representan el error de la red neuronal al realizar una inferencia y se compara con el valor real que está en el data set.

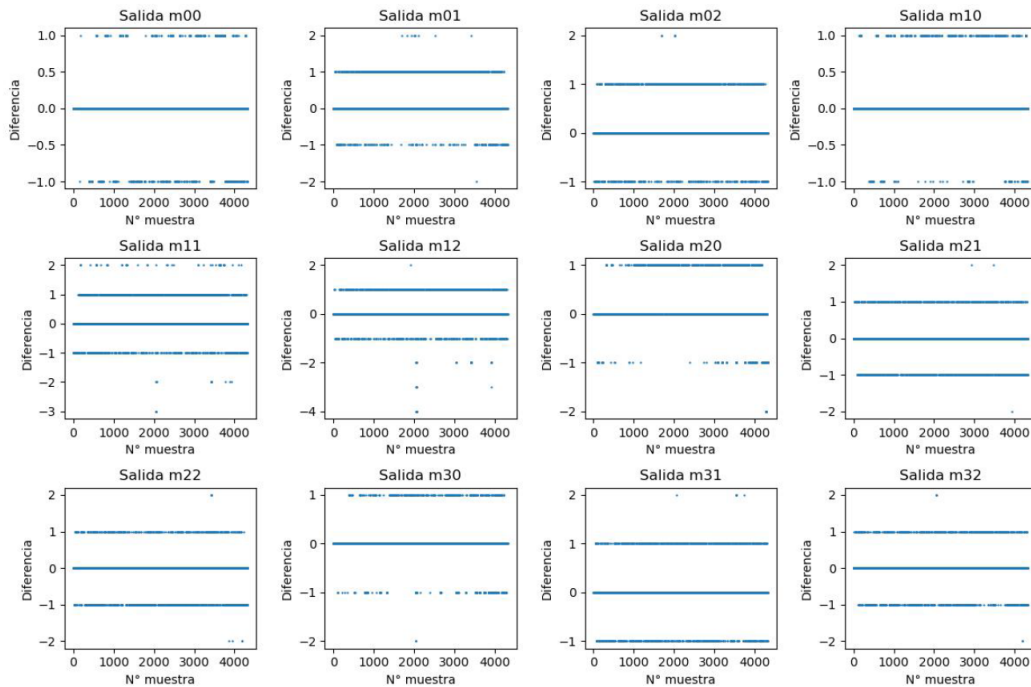


Figura 5.32: Gráfica de diferencia entre el valor real expresada en grados ($^{\circ}$) y la predicción del modelo de 1 capa con 18 neuronas cada una.

Se seleccionó la arquitectura de una sola capa con 18 neuronas para mantener simple el modelo, ya que será utilizado en un hardware con pocos recursos.

5.3. Transferencia de conocimiento

Una vez que se diseñó la red neuronal y se puso a prueba para saber cuál es la que tiene el mejor desempeño, se procedió a transferir el conocimiento adquirido por el modelo hacia la plataforma robótica. Sin embargo, es necesario cuidar ciertos detalles para que la transferencia pueda funcionar y que el modelo pueda realizar las acciones de balanceo en un robot del cual desconocemos su modelo matemático y en donde no se aplica ninguno de los métodos convencionales, como lo es el cálculo de trayectorias para las extremidades.

Para lograr tal cometido, primero es necesario utilizar un sensor que proporcione la información sobre el cambio de estado en los ángulos de Euler, es decir, un giroscopio y acelerómetro, ya que con el debido software puede generar esta información. De igual manera, se deben tomar en cuenta las direcciones de los ejes de este sensor, ya que las variables deben tener el mismo punto de referencia que el sensor con el cual se generó la base de datos, ver figura 5.33.

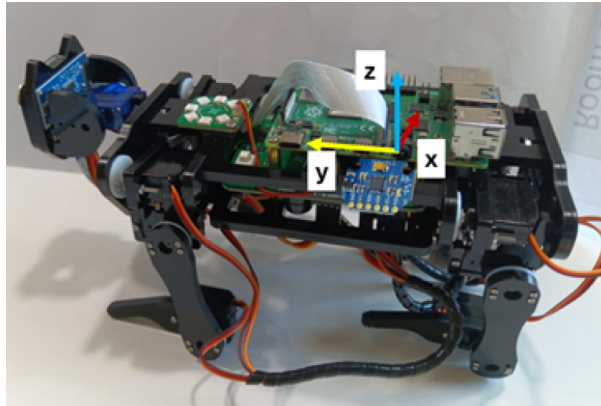


Figura 5.33: Eje de referencia inercial con el cual se generó la red neuronal artificial.

Otros aspectos a tener en cuenta, son que el sensor implementado en el robot propuesto en este trabajo, en comparación con el sensor de la otra máquina no está situado en la misma parte del torso del robo. Sin embargo, tienen un posicionamiento parecido, esto impacta en la forma que el robot se encuentra balanceado en el plano horizontal, pero este problema se puede solucionar compensando los motores manualmente. Cabe señalar que el proceso de calibración se podría automatizar. Así como también, el proceso de ensamble del robot, ya que este debe ser de tal manera que los motores se ensamblen cuando estén en un valor predefinido de 90° , esto permite que el robot quede ensamblado como se observa en la figura 5.34, de esta manera se estaría ensamblado igual que la plataforma robótica de Freenove.

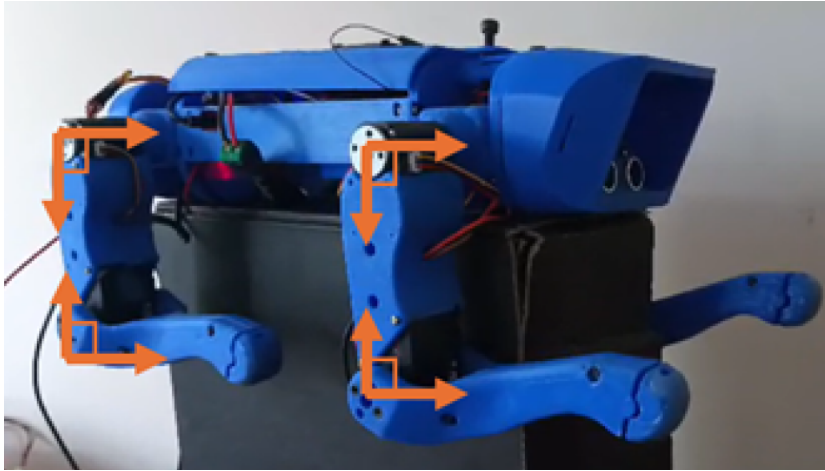


Figura 5.34: Ensamble del robot con los motores a 90°.

La salida de la red neuronal otorga valores en grados ($^{\circ}$), por lo que los motores del robot propuesto deberán funcionar de la misma manera. Nuestra plataforma robótica cuenta con sistemas embebidos para correr un sistema operativo Ubuntu, el problema al que nos enfrentamos es que no tienen mucho soporte para Tensorflow por lo que se debió convertir el modelo previamente generado a una versión que ocupa menos recursos computacionales como lo es Tensorflow Lite.

Una vez instaladas todas las dependencias en el robot, se procedió a generar el código que extrae la información del sensor y aplicar filtros de Kalman para obtener una medición precisa sobre los ángulos (Roll, Pitch, Yaw). En la figura 5.35 se presentan los datos adquiridos del sensor, así como la respuesta de la red neuronal a estas lecturas.

```

Temperature: 19.824117647588824
Roll: 1.363909817494026
Pitch: -2.784463636644484
Yaw: -17.11335899952662
Prediccion del modelo: [ 76 137 68 93 132 104 90 52 76 97 33 92]
Temperature: 19.68294117647059
Roll: 1.3390050292192353
Pitch: -2.7873943631291827
Yaw: -17.145111903324697
Prediccion del modelo: [ 76 137 68 93 132 104 90 52 76 97 33 92]
Temperature: 19.73
Roll: 1.3255518193231559
Pitch: -2.788228532015596
Yaw: -17.155702749125542
Prediccion del modelo: [ 76 137 68 93 132 104 90 52 76 97 33 92]
Temperature: 19.635882352941177
Roll: 1.3240098385423997
Pitch: -2.7876380099347033
Yaw: -17.16734405872699
Prediccion del modelo: [ 76 137 68 93 132 104 90 52 76 97 33 92]
Temperature: 19.73
Roll: 1.3309314493912543
Pitch: -2.792948805176764
Yaw: -17.181309567758707
Prediccion del modelo: [ 76 137 68 93 132 104 90 52 76 97 33 92]
Temperature: 19.73

```

Figura 5.35: Valores de Roll, Pitch y Yaw con filtros aplicados y la inferencia de la red neuronal para estos valores.

Una vez probado el sistema se procedió a ejecutar el control de balance en la nueva plataforma. En la figura 5.36 se observa la respuesta de las extremidades delanteras a la inclinación de la plataforma de tal manera que solo se afecta el ángulo Pitch. La reacción del modelo es rápida y no presenta saltos en los valores de las inferencias, por lo que se aprecia un movimiento controlado y ligero.

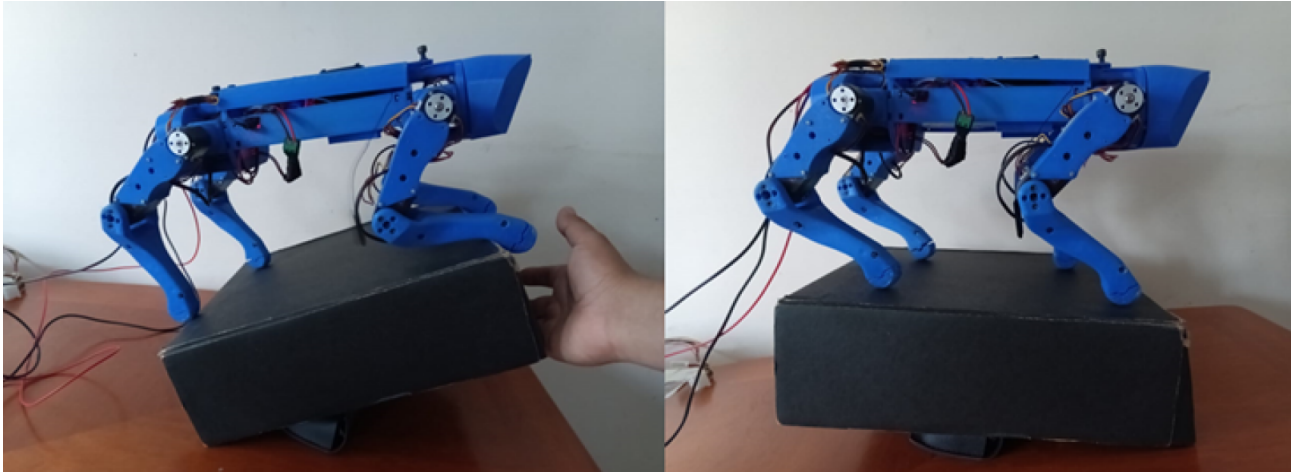


Figura 5.36: Prueba de balance en la nueva plataforma utilizando la red neuronal desarrollada.

5.4. Patrones de marcha emergentes

Al realizar pruebas con el modelo se observó que existían patrones de marcha que solamente se presentaban al variar el ángulo de giro en el eje x , esto es, realizar movimientos en *Roll*. Esto nos ayuda a entender que tan importante es el balance para que un robot cuadrúpedo pueda caminar y cómo es posible que a través de esta acción pueda descubrir patrones de marcha.

Se pretende realizar experimentos en el cual se plantee una metodología de cómo es posible que estos patrones emerjan de una manera más natural, esto quiere decir que necesitamos orientar el aprendizaje para que naturalmente exista un patrón de marcha y en este trabajo se plantean las bases de un marco de aprendizaje guiado que es principalmente entrenar el robot en primer lugar para que pueda mantener el equilibrio y permitir que el robot encuentre la relación que tiene cada una de sus extremidades para balancearse, es decir, mantener en todo momento el torso horizontal.

En la figura 5.37 se presentan los patrones de marcha que emergen al girar el torso del robot sobre el eje x , es decir, variar el ángulo *Roll*. Es importante señalar que estos patrones de marcha emergen al manipular el robot físicamente, pero es posible manipular los datos del sensor y hacerle creer al robot que esta desbalanceado para poder realizar el patrón de marcha. Se puede plantear una analogía al decir que la acción de caminar es

básicamente balancear y desbalancear el torso del robot, ya que existe un sistema encargado de reaccionar ante este estado de desbalanceo y como resultado se obtiene un patrón de marcha haciendo que el robot se desplace.

Una de las ventajas que ofrece este patrón de marcha emergente es sin duda la simplicidad de variables con las cuales puede realizarse, pues solo depende del valor del ángulo Roll para generar este patrón de marcha. La red neuronal que se utilizó para controlar el balance del robot solo consta de 1 capa con 18 neuronas, comparado con otros trabajos que utilizan redes neuronales con 3 capas y más de 50 neuronas por capa, al mismo tiempo que requieren de más variables para generar patrones de marcha.

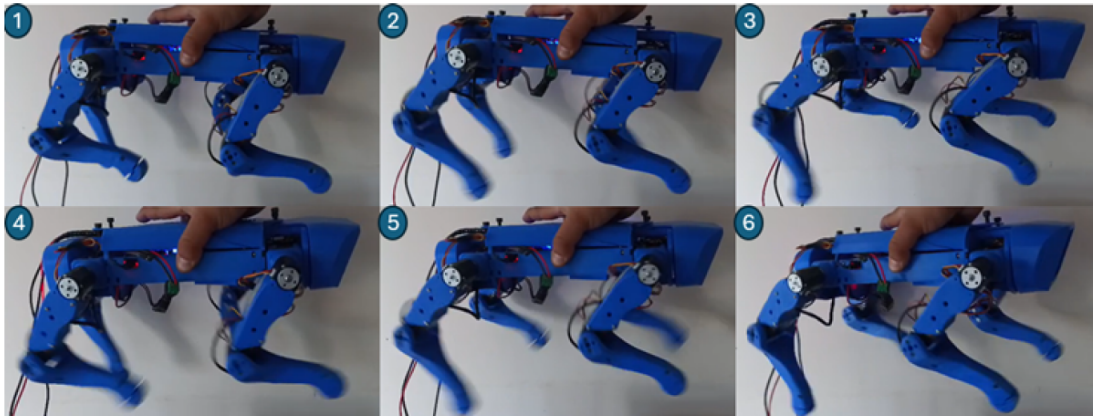


Figura 5.37: Patrones de marcha emergentes variando el giro en el eje x (Roll).

5.5. Ambiente de Simulación

5.5.1. Archivo URDF

Para que el simulador pueda calcular todos los fenómenos físicos del robot se requiere especificar un archivo de descripción del robot llamado URDF, el cual tiene las características más importantes de cada pieza del robot como tamaños, peso, inercia y su apariencia. En el archivo URDF se especifica cada una de las piezas del robot así como los sensores con los que recabará información, en la figura 5.38 se presenta en el lado derecho el archivo de descripción y en el lado izquierdo una pieza del robot.



Figura 5.38: Archivo URDF de descripción del robot.

El simulador realiza una representación de cada pieza del robot con formas básicas llamadas *Box* o cajas, con las que calcula si existen colisiones en cada parte. Por lo cual, para cada pieza se calculó el tamaño de la caja de colisión y estos datos se expresan en el archivo URDF, como se observa en la figura 5.39.

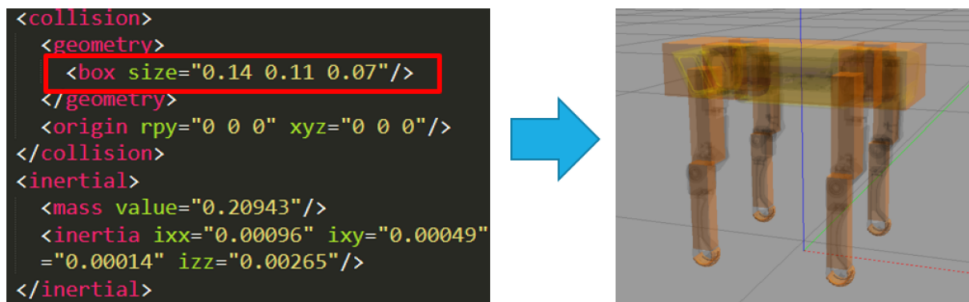


Figura 5.39: Representación de las piezas del robot en el simulador.

Para lograr una simulación bastante apegada a la realidad fue necesario calcular las propiedades físicas del robot, como su masa y sus momentos de inercia, que son datos que dictan la dinámica de un cuerpo rígido y el simulador se encarga de realizar todos estos cálculos. Al contar con los diseños CAD del robot, el cual se manufacturó en plástico PLA (ácido poliláctico), el software SolidWorks es de gran ayuda para calcular este tipo de parámetros. Se aprecia en la figura 5.40 como el software de diseño proporciona la información requerida.



Figura 5.40: Cálculo de parámetros físicos de masa y momentos de inercia, señalados en rectángulos rojos, de cada pieza del robot.

Una vez implementada la descripción del robot con todos los parámetros dinámicos calculados con ayuda del software CAD, se requirió dotar al sistema robótico con sensores para realizar el monitoreo y retroalimentación de información en cada momento, esto es, la posición de los motores, el contacto de las patas con el suelo y la información del sensor de inercia IMU.

5.5.2. Configuración de Isaac Lab

El ambiente de entrenamiento en la plataforma Isaac Lab pertenece a la empresa Nvidia, este software permite simulaciones en tiempo real y muy cercanas a la realidad, ya que cuenta con un simulador muy potente y la gran característica de utilizar la paralelización de las tarjetas gráficas. Toda la simulación se va a ejecutar paralelamente en la GPU.

Se utilizó Isaac Lab que está construido sobre Isaac Sim para proporcionar un marco unificado y flexible para el aprendizaje de robots que aprovecha las últimas tecnologías de simulación. Está diseñado para ser modular y extensible, y tiene como objetivo simplificar los flujos de trabajo comunes en la investigación robótica (como el aprendizaje por repetición, el aprendizaje a partir de demostraciones y la planificación de movimientos). Si bien incluye algunos entornos, sensores y tareas pre-construidos, su objetivo principal es proporcionar una interfaz de código abierto, unificada y fácil de usar para desarrollar y probar entornos personalizados y algoritmos de aprendizaje de robots. En la figura 5.41 se presentan los módulos principales del simulador Isaac lab.

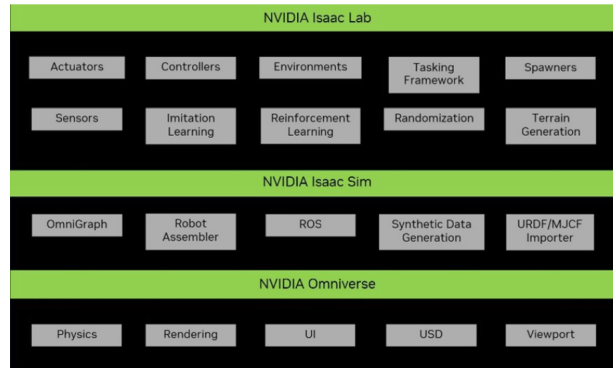


Figura 5.41: Módulos principales del simulador Isaac lab.

Para trabajar con Isaac Lab es necesario contar con los archivos URDF del robot para realizar su digitalización, como se observa en la figura 5.42, y contar con una simulación con los parámetros cercanos a los reales.

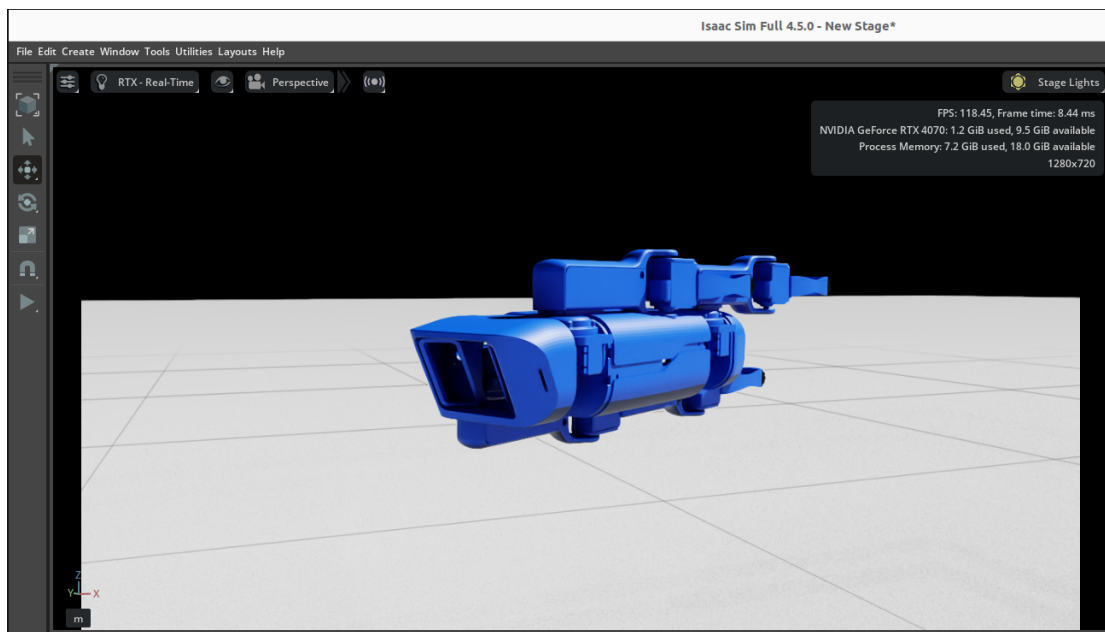


Figura 5.42: Robot digitalizado y configurado para utilizarlo en simulación.

Para lograr tal cometido se utilizó un importador de URDF que trae el mismo simulador para realizar la transformación del archivo URDF del robot al formato con el que trabaja el simulador, es decir, “*Universal Scene Description (USD)*”. Se deben seleccionar las configuraciones adecuadas para que la conversión de formatos sea correcta, esto debido a que el archivo URDF describe cada componente del ensamble donde muchas veces es necesario que las partes fijas que no son accionadas por un motor se consideren como una

sola pieza en conjunto y el archivo original, al ser ensamble, toma cada parte como una pieza aislada sin importar si forman parte de un ensamble que se mueve en conjunto o no.

Por lo tanto, en la configuración del importador URDF de Isaac Sim, se debe seleccionar que se desea fusionar las piezas fijas para que se tomen como una sola independientemente si es ensamble o no. Además, se debe configurar que la base del robot, en este caso el torso del cuadrúpedo, sea una base móvil y no este anclada al ambiente. Por otra parte, es de vital importancia configurar las unidades métricas en las que está definido el archivo URDF, ya que de esto dependerá como calcule las iteraciones físicas del robot, este parámetro lo define como “*Unidades del escenario por metro*”, el simulador Isaac sim tiene como unidades predefinidas los centímetros, en este caso, el archivo URDF también está configurado en centímetros por lo que se debe dejar el valor por defecto del importador. En la figura 5.43 se observan las configuraciones realizadas en el importador del simulador Isaac Sim.

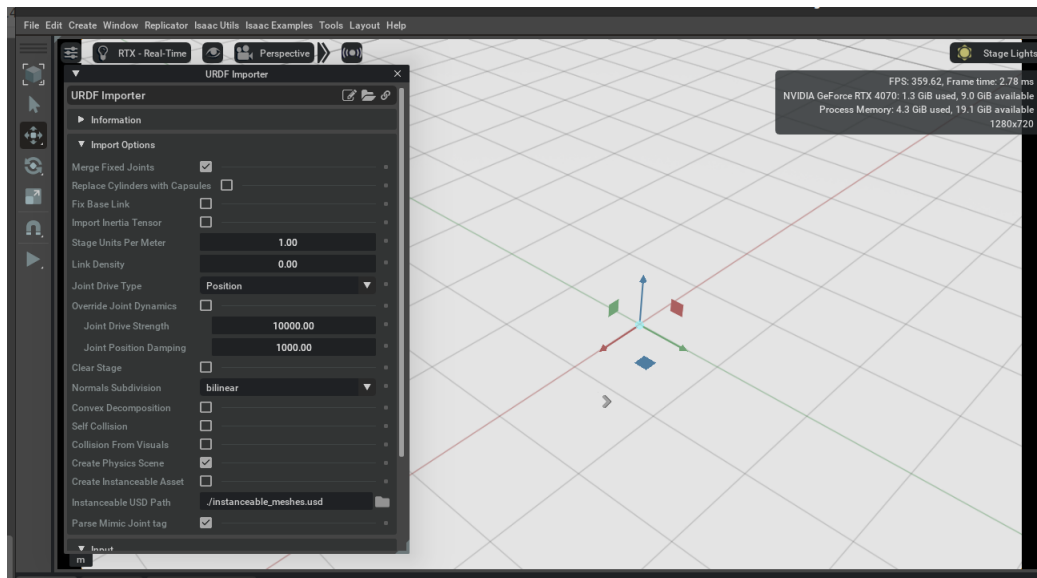


Figura 5.43: Configuraciones realizadas para una conversión del archivo URDF del robot cuadrúpedo al archivo USD que maneja el simulador.

El simulador ahora ya cuenta con el robot en el escenario, aunque Isaac Sim realiza la conversión del archivo URDF con sus propias herramientas, se debe de corroborar ciertos aspectos del proceso de importación. Dentro del simulador se seleccionan cada una de las articulaciones que son comandadas por un motor y se configuran los parámetros deseados acorde a las características del robot físicos, valores como límites de posición, fricción, límites de par, límites de velocidad, etc. En la figura 5.44 se observa una de las configuraciones de parámetros dentro del simulador.

CAPÍTULO 5. RESULTADOS

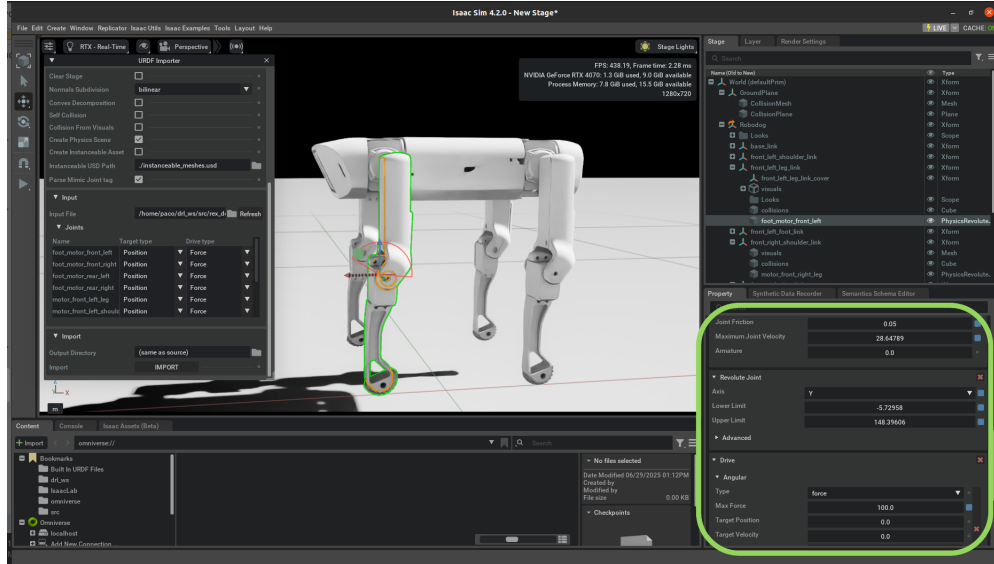


Figura 5.44: Configuraciones realizadas en los parámetros de cada articulación.

Se configuraron las posiciones iniciales o de home, las cuales representan cuando las articulaciones están en 0° grados, tal cual como se tiene en nuestro robot real, con el fin de poder utilizar el modelo que se entrenó previamente con el controlador PID en la sección 5.2.3. Se presenta en la figura 5.45 el ajuste de las posiciones de cada articulación en 90° y el robot muestra la misma configuración que el robot de (Freenove, 2016).

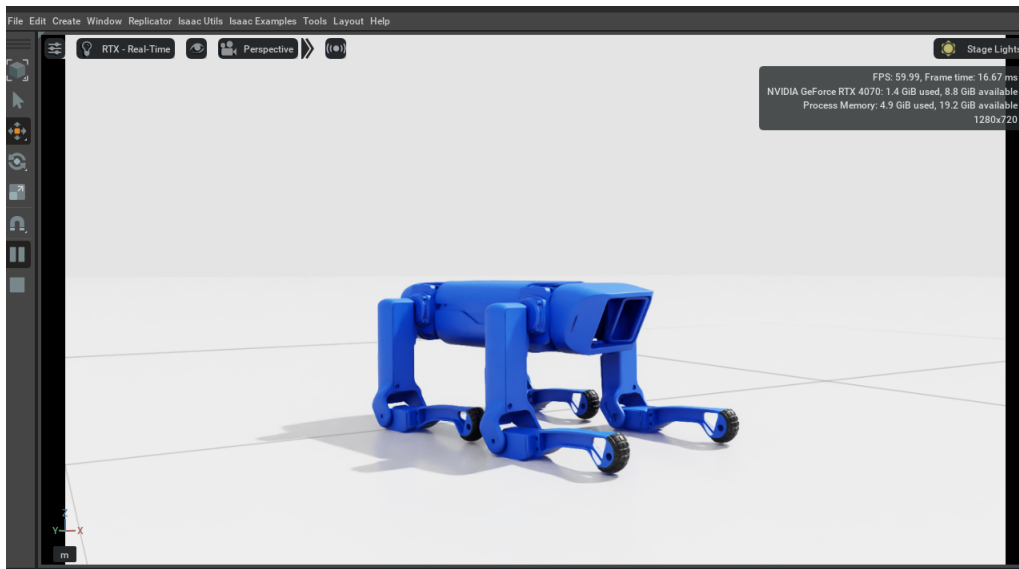


Figura 5.45: Configuración de posiciones a 90° en cada articulación del robot.

Fue importante corroborar, una vez exportado el robot en el simulador, que las piezas que conforman las patas y que están en contacto con el suelo se hayan importado como

partes independientes de la estructura de la pata, ya que al momento de colocar sensores que registren la interacción entre el suelo y la pata del robot se desea que se registre la interacción solo en el extremo de la pata. Si la pieza se importó fusionada con toda la extremidad, el sensor va a registrar cualquier interacción no sólo con el final de la pata sino con toda la extremidad. En la figura 5.46 se observa de color verde cada uno de los extremos de las patas como una pieza aislada de toda la extremidad, donde se posicionarán los sensores de contacto.

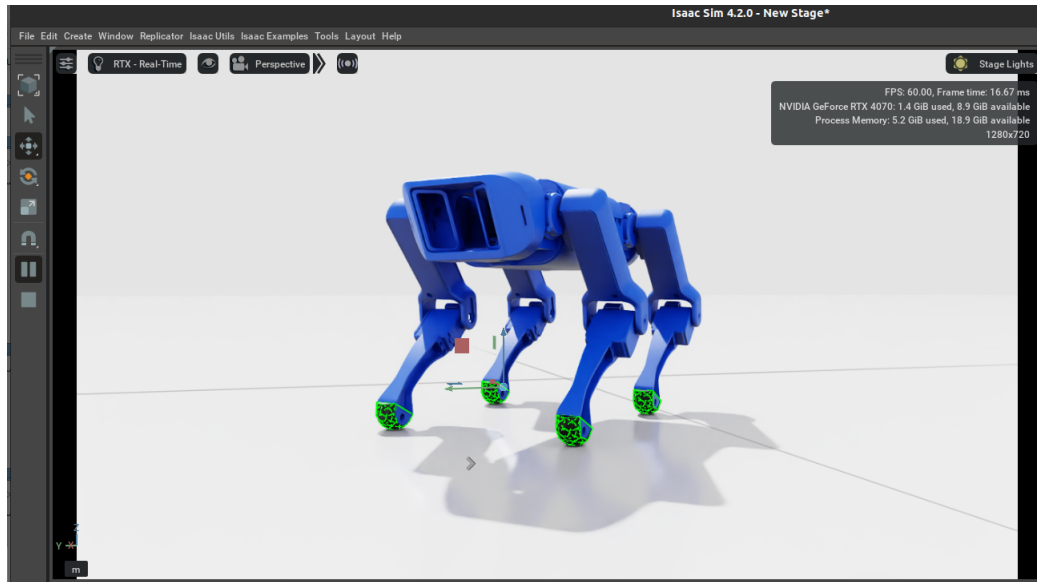


Figura 5.46: Extremos de cada articulación sombreadas en verde, configuradas como pieza independiente de toda la extremidad.

En el desarrollo de esta tesis se planteó crear un ambiente de aprendizaje donde el robot interactúe con su entorno y por medio de una red neuronal logre aprender una política que realice tareas de control, en primer instancia de equilibrio y posteriormente, patrones de marcha derivados de esta acción, como se expuso en 5.4.

Para lograr esto, se requiere crear los elementos que estarán dentro de la simulación y ayudarán al robot en su camino de aprendizaje, por lo que se requirió una plataforma de tamaño considerable pensada como escenario donde el robot se posicionará encima y posteriormente se modificara la inclinación de la plataforma, con la finalidad de que el robot aprenda a reaccionar a las alteraciones de inclinación del entorno, esta plataforma debe de tener una configuración específica debido a que se modificará la inclinación aleatoriamente, el robot cuadrúpedo podría resbalarse si no existe un coeficiente de fricción dinámico y estático alto en la plataforma. En la figura 5.47 se presenta la plataforma que servirá para el ambiente de aprendizaje y se observa el coeficiente de fricción alto en las configuraciones.

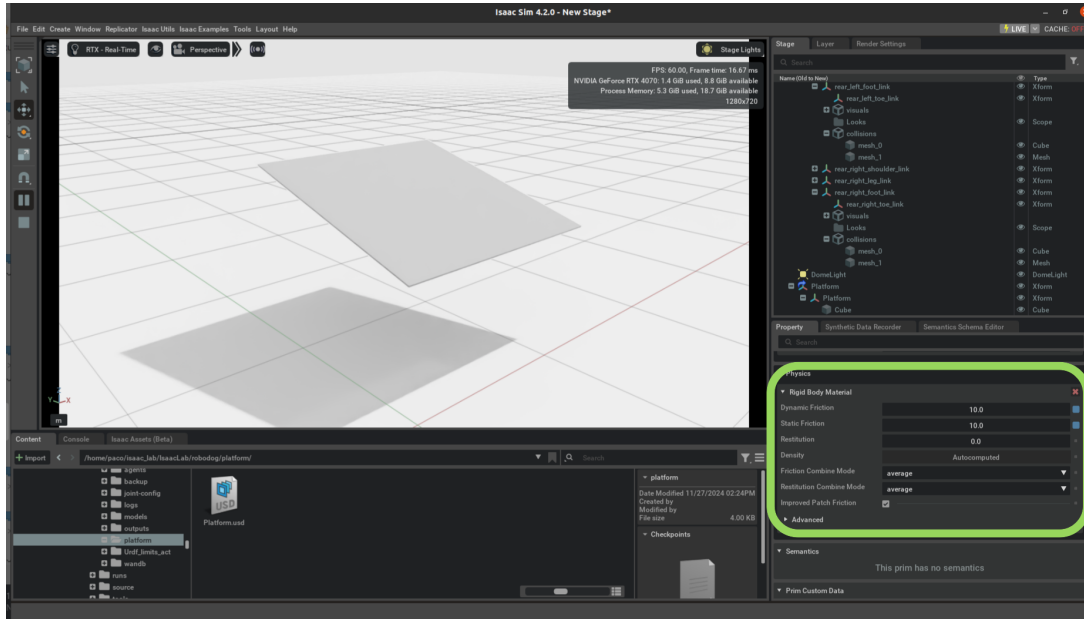


Figura 5.47: Diseño y configuración de una plataforma que servirá como ambiente de aprendizaje para el robot cuadrúpedo, la cual puede variar su inclinación y posee un coeficiente de fricción dinámico y estático alto.

Una vez configurado el robot, el simulador Isaac lab cuenta con diferentes métodos para realizar un ambiente de entrenamiento para tareas de aprendizaje por refuerzo y tiene la característica de realizar entornos vectorizados, lo que ayuda a poder entrenar a más de una instancia de robot o escenario al mismo tiempo y unificar los datos adquiridos para un entrenamiento masivo.

El utilizar instancias de robots requiere de configuraciones predefinidas par que pueda replicar el robot en un entorno, debido a esto se realizó el archivo de configuración del robot llamado “*Robodog_CFG*” donde se define una instancia de la clase *ArticulationCfg* propia del ecosistema de Isaac Sim, esta clase contiene las configuraciones básicas para un elemento digital dentro una simulación que contiene partes móviles. En este script se utilizan las herramientas propias del simulador, ya que está dividido por clases, y utilizándolas en conjunto se pueden crear escenarios modulares en los cuales cada instancia puede interactuar dentro de escenario entre si.

Para definir las configuraciones iniciales de cada instancia de robot se hace uso en primer lugar de una herramienta que posiciona el robot físico en el entorno de simulación a través de las herramientas de simulación que manda a leer la ruta del archivo USD que hemos configurado.

En el fragmento de código 5.1 se presenta parte de las configuraciones importantes que se tomaron en cuenta para este caso, por ejemplo, la ruta donde se encuentra el archivo USD, propiedades de cuerpos rígidos propias del simulador como deshabilitar la gravedad,

las velocidades lineales y angulares máximas, así como las propiedades de la articulación raíz, en este caso sería el torso del robot. Se pueden hacer modificaciones para que el robot no pueda colisionar entre sí mismo, o que la base del robot se quede fija en el entorno y que no sea perturbada por otras fuerzas, esto sería útil para probar en primer lugar las trayectorias de cada extremidad del robot y posteriormente, deshabilitar la opción para corroborar que estas realmente generan un movimiento de caminata en el robot.

Código 5.1 : Fragmento del código “Robodog_CFG” donde se configuran las propiedades iniciales del robot en el simulador

```

13
14  ROBODOG_CFG = ArticulationCfg(
15      # Ruta del archivo USD que contiene el CAD del robot
16      spawn=sim_utils.UsdFileCfg(usd_path="/robodog.usd",
17      rigid_props=sim_utils.RigidBodyPropertiesCfg(
18          disable_gravity=False, # Deshabilitar gravedad
19
20      # Velocidades máximas
21      max_linear_velocity=100.0,
22      max_angular_velocity=100.0)
23
24      articulation_props=sim_utils.ArticulationRootPropertiesCfg(
25          enabled_self_collisions=False, # No colisiona el mismo
26          fix_root_link=False ), # Fijar la base al entorno
27
28      # Activar sensores de contacto
29      activate_contact_sensors=True)
30      ...

```

Otro aspecto importante que se consideró en este archivo son las posiciones iniciales que el robot tendrá en cada episodio de entrenamiento, tanto de la base del robot en el espacio, como cada una de las posiciones articulares, de esto depende el curso que tome el aprendizaje. Por lo tanto, en el fragmento de código 5.2 se presentan las configuraciones de posiciones iniciales a través de la clase *.ArticulationCfg.InitialStateCfg*", en la línea 35 se define la posición en metros del robot respecto al origen del entorno de simulación y en la línea 36 se definen cada una de las posiciones articulares en radianes, seleccionándolas por el nombre que tiene cada una dentro del archivo USD.

Las configuraciones mostradas en este apartado se utilizan cada vez que se hace una instancia del robot, el código *Robodog_CFG*" completo se encuentra en la sección de Anexos de esta tesis.

Código 5.2 : Fragmento del código “Robodog_CGF” donde se inicializan las posiciones de cada articulación

```
34  init_state=ArticulationCfg.InitialStateCfg(  
35  pos=(0,0,0.24),  
36  joint_pos={"motor_front_left_shoulder": 1.57,  
37            "motor_front_left_leg":1,  
38            "foot_motor_front_left":1.23,  
39            "motor_front_right_shoulder":1.57,  
40            "motor_rear_left_leg":1,  
41            "foot_motor_rear_left":1.23,  
42            "motor_rear_left_shoulder":1.57,  
43            "motor_rear_right_leg":1,  
44            "foot_motor_rear_right":1.23,  
45            "motor_rear_right_shoulder":1.57,  
46            "motor_front_right_leg":1,  
47            "foot_motor_front_right":1.23}  
48  ),  
49  ...
```

5.6. Patrones de marcha derivados del equilibrio

En el desarrollo de esta tesis se encontró que es posible generar patrones de marcha para robots cuadrúpedos a través de un control del balance, específicamente se encontró la relación que tienen los patrones de marcha con las variables de orientación del torso del robot y como consecuencia de tratar de estabilizar el torso del robot cuando hay una perturbación, se generan patrones de marcha conocidos.

Además de contar con el escenario y la configuración a punto del robot, se requiere configurar el entorno de simulación, donde se posiciona el robot sobre el suelo, se definen características como luces y sensores en el robot, además se carga el modelo de red neuronal que se entrenó previamente, en la sección 5.2.3, para realizar exploraciones sobre el tipo de patrones de marcha que se pueden generar.

El simulador Isaac Lab tiene la capacidad de ser modular y realizar ambientes basados en “*managers*”, esto quiere decir que existe una clase que concentra y maneja las interacciones de cada objeto dentro de la simulación y así coordina cada una de las interacciones entre los elementos contenidos, por ejemplo, el ambiente y el robot.

Para este apartado se utilizó solo la clase “*SceneManager*” la cual es responsable de importar los objetos dentro del simulador llamando las clases correspondientes por cada uno. En el fragmento de código 5.3 se muestra la forma de instanciar el terreno sobre el cual se posiciona el robot, las luces del escenario y el robot. En esta configuración se debe instanciar los sensores como objetos de la clase *Sensor_CFG* para que el manager pueda registrar cada interacción de estos en el entorno.

Código 5.3 : Fragmento del código “Patrones_marcha.py” donde se muestra la configuración del ambiente de simulación

```

47 @configclass
48 class RobodogSceneCfg(InteractiveSceneCfg):
49
50     # ----- Superficie de suelo
51     ground = AssetBaseCfg(prim_path="/World/defaultGroundPlane",
52     spawn=sim_utils.GroundPlaneCfg(
53     physics_material=materials.RigidBodyMaterialCfg()))
54
55     # ----- Luces
56     dome_light = AssetBaseCfg(
57     prim_path="/World/Light", spawn=sim_utils.DomeLightCfg(intensity
58     =1000.0, color=(0.75, 0.75, 0.75)))
59
60     # ----- Robot
61     robodog: ArticulationCfg = ROBODOG_CFG.replace(prim_path="{
62     ENV_REGEX_NS}/Robot")
63
64     # ----- IMU
65     imu_sensor = ImuCfg(prim_path="{ENV_REGEX_NS}/Robot/rex/base_link",
66     update_period = 0.0,
67     visualizer_cfg= FRAME_MARKER_CFG.replace(prim_path="/Visuals2/
68     IMU_XYZ",),
69     debug_vis = False)
70
71     # ----- Sensores de contacto
72     contact = ContactSensorCfg(
73     prim_path="{ENV_REGEX_NS}/Robot/rex/.*_toe_link",
74     update_period=0.0,
75     visualizer_cfg=CONTACT_SENSOR_MARKER_CFG.replace(prim_path="/
76     Visuals/ContactSensors"),
77     history_length=1, debug_vis=True, track_air_time=True )
78     ...

```

Posteriormente se importo el modelo de red neuronal con mejor desempeño a la tarea de balance del torso, se selecciono la arquitectura de dos capas ocultas con 14 neuronas cada una, ya que esta red neuronal presenta el mejor desempeño de acuerdo con la tabla comparativa 5.1. En el fragmento de código 5.4 se muestra la clase que define la arquitectura de la red neuronal, posteriormente se debe importar los pesos del modelo entrenado, con el archivo “modelo_pytorch.pth” e indicar que el modelo correrá desde la gpu en modo evaluación.

Código 5.4 : Fragmento del código “Patrones_marcha.py” donde se muestra como cargar el modelo de red neuronal

```

83 class Modelo(nn.Module):
84     def __init__(self, entradas = 3, capas = 14, salidas = 12):
85         super(Modelo, self).__init__()
86         self.fc1 = nn.Linear(entradas, capas)
87         self.fc2 = nn.Linear(capas, capas)
88         self.fc3 = nn.Linear(capas, salidas)
89         self.relu = nn.ReLU()
90
91     def forward(self, x):
92         x = self.relu(self.fc1(x))
93         x = self.relu(self.fc2(x))
94         x = self.fc3(x)
95         return x
96
97 model = Modelo()
98 model.load_state_dict(torch.load(r'/robodog/models/modelo_pytorch.
    pth'))
99 model = model.to('cuda')
100 model.eval() # Establecer el modelo en modo de evaluación
101
102 ...

```

Se llevaron a cabo experimentos modificando los ángulos *Roll* y *Yaw* de dos formas distintas. La primera, el ángulo se incrementó de manera lineal desde 0° hasta 25°. Al alcanzar los 25°, comenzó a disminuir en pasos de 1° hasta llegar a -25°. Una vez alcanzado el valor mínimo de -25°, se volvió a aumentar progresivamente hasta alcanzar nuevamente los 25°.

En la segunda forma, de tipo no lineal, también se incrementó el ángulo desde 0° hasta 25°. Al alcanzar los 25°, el valor se fijó en 0°, para luego disminuir en pasos de 1° hasta llegar a -25°. Una vez alcanzado el mínimo, se volvió a fijar el valor en 0°.

En las siguientes gráficas se presentan los movimientos, en grados, de cada uno de los motores que controlan las patas. La nomenclatura utilizada para identificarlas es la siguiente:

- *FR (Front Right)*: Pata delantera derecha.
- *FL (Front Left)*: Pata delantera izquierda.
- *RR (Rear Right)*: Pata trasera derecha.
- *RL (Rear Left)*: Pata trasera izquierda.

5.6.1. Variación del ángulo Roll

Se posicionó el robot en la escena con la particularidad de estar suspendido en el aire, sin interacción con la fuerza de gravedad, como se muestra en la figura 5.48, con el motivo de observar el movimiento de las patas del robot y corroborar que existe un patrón de marcha.

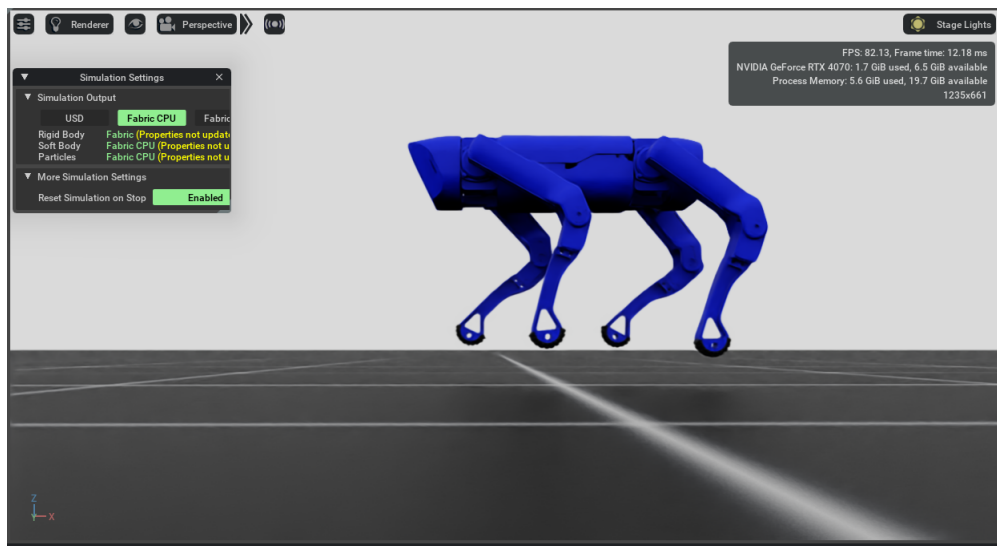


Figura 5.48: Instancia del robot suspendido en el aire dentro de la escena de simulación, sin interactuar con la fuerza de gravedad.

Posteriormente se procedió a generar el desbalance virtual con incrementos en el ángulo Roll de manera lineal hasta 25° y al llegar a ese punto se comienza a disminuir hasta llegar a -25° , donde se vuelve a incrementar linealmente hasta el punto máximo de 25° . Con esta forma de desbalance, se observó que las extremidades del robot generaban el patrón de marcha dinámico de paso lateral sincronizado “Pace”, donde ambas patas de cada lado se mueven al mismo tiempo, en la figura 5.49 se presenta el movimiento en grados de cada articulación que maneja los extremos de las patas del robot, donde los valores cercanos a 0° indican que la pata está en contacto con el suelo, caso contrario con valores alejados de 0° . En este caso se observa como las patas delantera y trasera derechas no están en contacto con el suelo al tener valores alejándose de 0° , por otra parte, las patas de lado izquierdo están en contacto con el suelo. Esta representación del patrón de marcha “Pace” se genera al variar linealmente el ángulo de inclinación Roll.

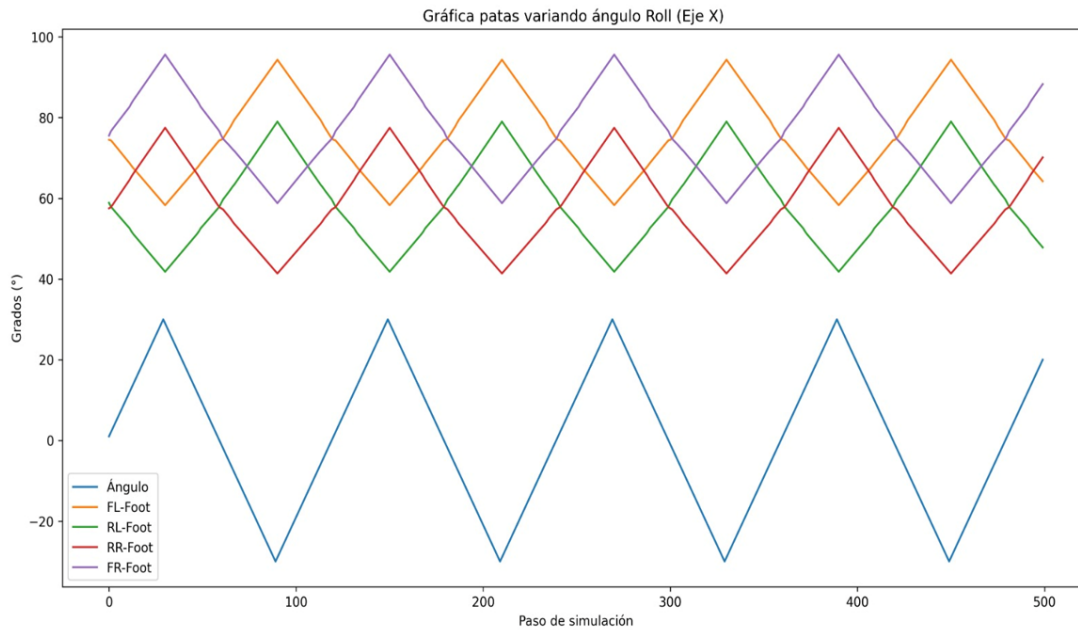


Figura 5.49: Movimiento en grados de las patas al variar el ángulo Roll linealmente.

Aunque el robot no presenta movimiento de avance cuando esta posicionado en la superficie y se realiza el desbalance lineal, ya que solo presenta movimientos de balance del torso, debido a que se esta replicando el comportamiento del controlador PID que reacciona a los desbalances.

El patrón de marcha generado al realizar inclinaciones virtuales de manera lineal en el torso del robot tiene un comportamiento suave en la transición de incremento a decremento, aunque al ser un patrón de marcha dinámico requiere un movimiento de impulso al momento de realizar esta transición, por tanto, al variar la inclinación del angulo Roll de manera no lineal, es decir, cuando llega al punto máximo de inclinación se procede a dar un cambio brusco en decremento a 0, esto genera un impulso en el robot y alterna el movimiento en las extremidades de lado contrario. En la figura 5.50 se observa el mismo comportamiento de un patrón de marcha “Pace”, aunque con la particularidad de que la frecuencia al alternar las patas es más rápido y por lo tanto existe un impulso en esta transición. En ambos casos se logra apreciar la correlación directa del movimiento de cada pata con la variación del ángulo Roll.

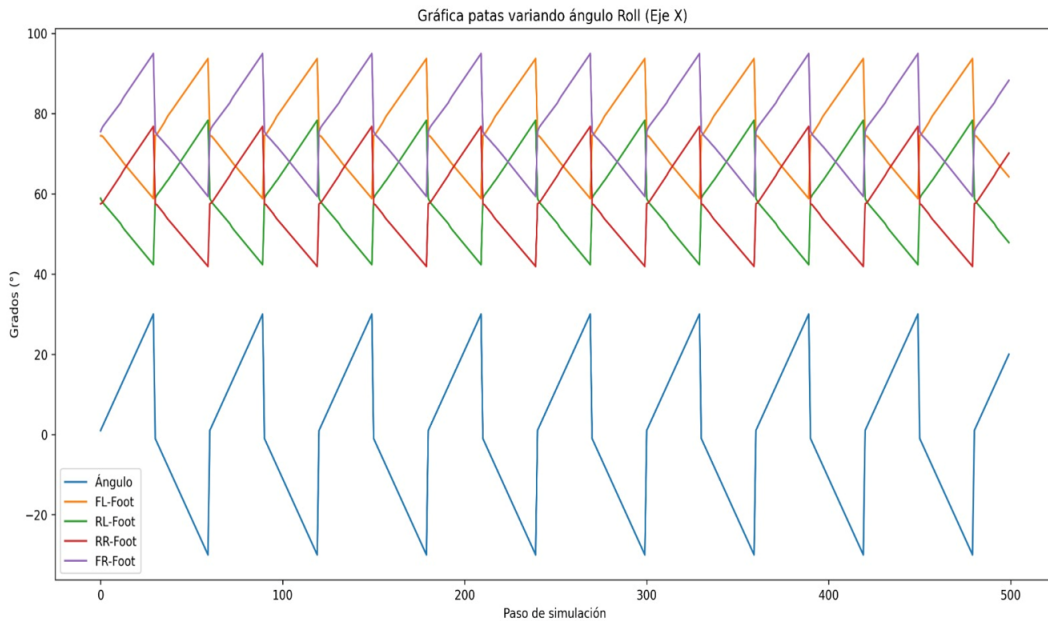


Figura 5.50: Movimiento en grados de las patas al variar el ángulo Roll de manera no lineal.

Con esta variación del ángulo Roll emergió un patrón de marcha conocido y documentado en la literatura, el patrón de marcha llamado "*Paso*", este patrón de marcha de dos tiempos donde las extremidades de un mismo lado tocan el suelo en un tiempo y para el siguiente tiempo alternan la posición, ahora se apoyan las otras dos patas del mismo lado como se logra apreciar en la figura 5.51. En su mayoría, el paso lo utilizan los animales con patas largas, aparentemente porque son más propensos a enredarse al trotar.

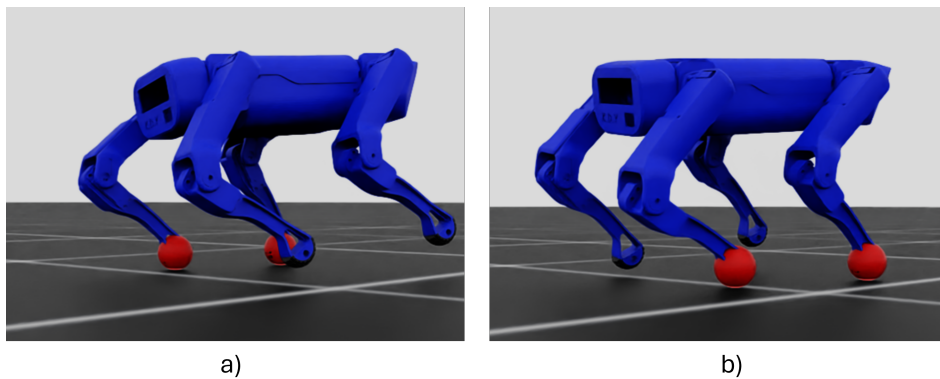


Figura 5.51: Patrón de marcha "*Paso*", donde se observa en (a) cuando las dos extremidades del un solo lado están en el suelo, y (b) donde alterna las patas que están tocando el suelo.

5.6.2. Variación del ángulo Pitch

Se realizó la misma metodología para el ángulo pitch, esto es, generar un desbalance virtual con incrementos de manera lineal hasta 25° y al llegar a ese punto se comienza a disminuir hasta llegar a -25° , posteriormente se vuelve a incrementar linealmente hasta el punto máximo de 25° . Con esta forma de desbalance, se observó que el robot generaba el patrón de marcha dinámico “*Galope*”, en el que las patas delanteras tocan el suelo en un momento y alternan con las patas traseras para el siguiente paso, este tipo de marcha está bien documentado y se puede observar en animales cuando estos quieren avanzar a mayor velocidad.

En la figura 5.52 se presenta el movimiento de cada pata al realizar un patrón de marcha de galope. Este patrón se realiza al mover ambas patas traseras al mismo tiempo seguidas de las patas delanteras, impulsando al robot a avanzar. Se observa que las patas traseras se alejan del suelo al mismo tiempo que el ángulo pitch se incrementa, por otra parte, las patas delanteras (FR, FL) se acercan al suelo al mismo tiempo.

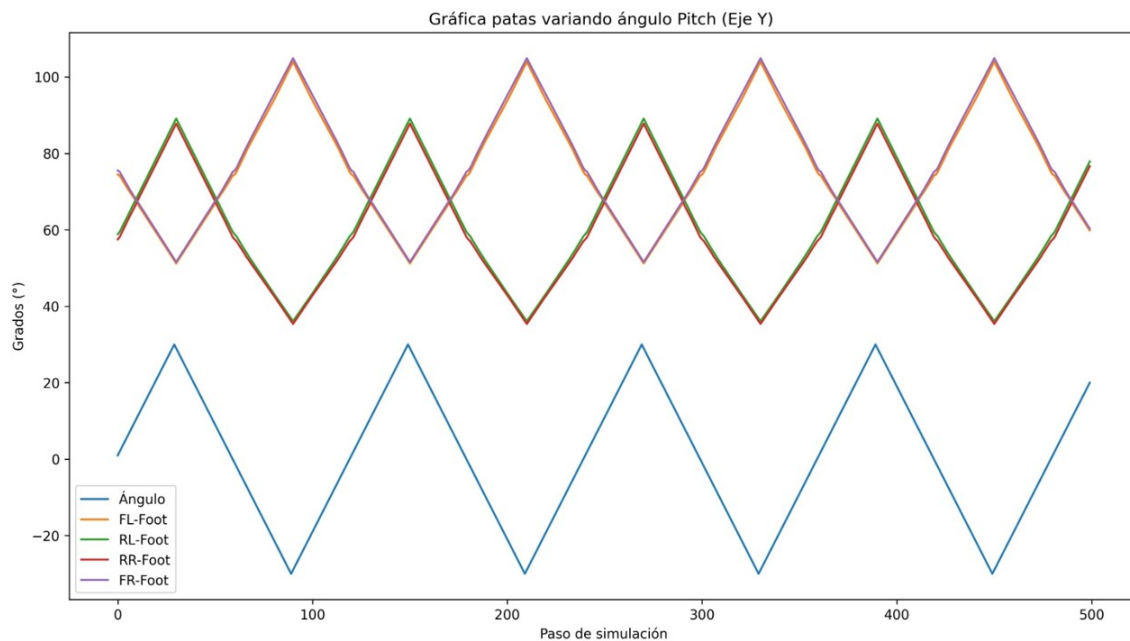


Figura 5.52: Movimiento en grados de las patas al variar el ángulo pitch de manera lineal.

Como anteriormente se describe, al posicionar al robot en la superficie del suelo, este desbalance solo genera que el robot se incline hacia adelante y atrás, no genera un avance y esto es debido a que al ser patrones de marcha dinámicos, requieren un impulso al alternar las patas. Por lo anterior, el desbalance no lineal que se propone, logra que el robot avance sobre el suelo al realizar un impulso cuando hay un cambio abrupto de 25° a 0° . En la figura 5.53 se presenta el mismo comportamiento de galope, con la diferencia de que la frecuencia para alternar las patas aumenta. En la figura se logra apreciar de nuevo la relación directa

que tiene el movimiento de las patas con el ángulo de inclinación del torso. Además, al ser un patrón de marcha dinámico, en este caso, aumentar la frecuencia de alternación de patas le proporciona mayor estabilidad al robot.

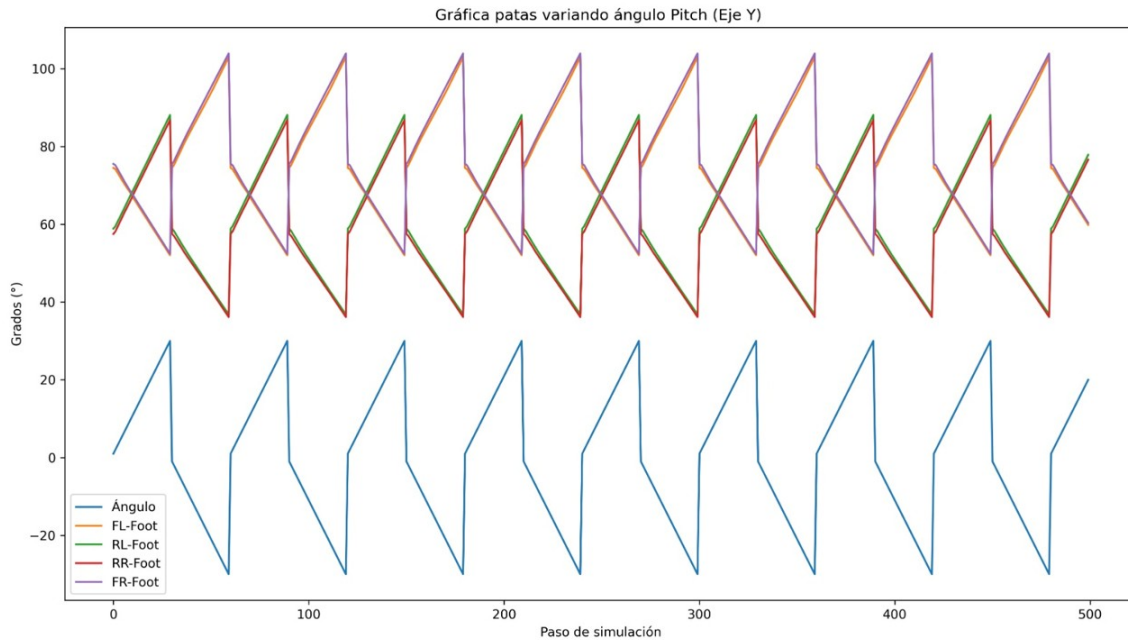


Figura 5.53: Movimiento en grados de las patas al variar el ángulo pitch de manera no lineal.

En la figura 5.54 se puede observar el patrón de marcha galope que se genera con el desbalance no lineal, donde 5.54(a) se muestra al robot alzando las patas delanteras al mismo tiempo que las patas traseras están tocando el suelo y 5.54(b) alternando las patas traseras ahora en el aire y delanteras tocando el suelo.

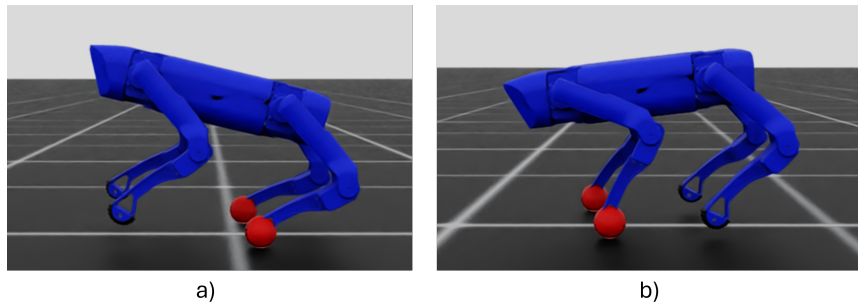


Figura 5.54: Patrón de marcha "Galope", donde se observa en (a) cuando las dos extremidades traseras están tocando el suelo y (b) donde alterna a las patas delanteras que están tocando el suelo.

5.7. Aprendizaje por Refuerzo Profundo DRL

El objetivo de la presente tesis es desarrollar un agente (red neuronal artificial) capaz de realizar tareas de control de equilibrio o marcha en un robot cuadrúpedo, esto mediante aprendizaje por refuerzo profundo. Esta metodología se presenta como una alternativa para el control de robots cuadrúpedos que requieren del modelo matemático completo, así como los parámetros físicos reales del sistema. Se realizaron experimentos bajo diversas configuraciones utilizando el algoritmo “Optimización de Políticas Próximas (PPO)”, seleccionado por su estabilidad de aprendizaje y por los casos de éxito reportados en la literatura en tareas de control de robots manipuladores y cuadrúpedos.

5.7.1. Aprendizaje de Política de incorporación estable

En primer lugar, se estableció el modelo en el simulador directamente en el suelo, y se realizaron experimentos en donde las extremidades del robot se posicionaban al azar al iniciar cualquier episodio. La tarea que se desea aprender es incorporarse si está en el suelo y mantenerse erguido, sin ninguna otra acción. Esto sirve para que el agente aprenda todas las posiciones posibles que pueden tomar cada una de sus articulaciones y como afectan directamente a la inclinación del torso.

Para esta tarea se definieron las recompensas que dictaminarán el comportamiento del aprendizaje. En primer lugar, se penalizará al robot por tener aceleración en los ejes XYZ, así como si la acción de cada una de sus articulaciones es muy grande. También se considera el tiempo en el que está en el aire cada una de sus patas y la altura del torso, es decir, se penaliza que no respete la altura a la cual se le está pidiendo que esté el torso cuando esta parado en las 4 patas. Además, se penalizan las posiciones de las articulaciones si están fuera de rango y sobre todo, se monitorearán las inclinaciones medidas en ángulos de Euler, que son Roll, Pitch y Yaw. Por último, se penaliza con funciones de terminación de episodio, las cuales se activan cuando el robot golpea el suelo con la cabeza o cuando tiene una inclinación mayor a 60° en cualquiera de los ángulos de Euler anteriormente medidos.

Se realizaron 57 experimentos donde se variaron los pesos de las funciones de recompensa, se configuró un número máximo de pasos por episodio a 500 y en total 50,000 pasos de duración de experimento, se determinó una estructura de red neuronal de 2 capas con 200 neuronas cada una, función de activación elu. Se utilizó la configuración por defecto de la librería skrl de Isaac lab, la cual tiene valores utilizados en otras tareas de aprendizaje para el algoritmo PPO, como se muestran en la tabla [5.2](#)

Tabla 5.2: Configuración de parámetros para el algoritmo PPO

Parámetro	Valor
rollouts	16
learning epochs	8
mini batches	8
discount factor	0.99
lambda	0.95
learning rate	0.0003
entropy loss scale	0.0
value loss scale	2.0

La función de recompensa que se utilizó en este experimento se muestra en la ecuación 5.1.

$$R = -w_1 Roll - w_2 Pitch - w_3 Yaw - w_4 JP_{limit} + w_5 Contact - w_6 Air_{time} - w_7 Ax - w_8 Ay + w_9 Alive - w_{10} Term - w_{11} H \quad (5.1)$$

Donde;

$w_1 \dots w_{11}$ = Pesos a variar para cada término

$Roll$ = Inclinación eje x

$Pitch$ = Inclinación eje y

Yaw = Inclinación eje z

JP_{limit} = Exceder límites de articulación

$Contacto$ = Contacto patas-suelo

Air_{time} = Tiempo en el aire cada pata

Ax = Aceleración eje x

Ay = Aceleración eje y

$Alive$ = Seguir en el mismo experimento cada paso

$Term$ = Cumplir alguna condición de terminación

H = Altura del torso

Una vez variando los pesos de cada término de la función de recompensa se obtuvo una tabla del promedio de cada reward, como se muestra en la figura 5.55, lo cual nos indica la importancia que tiene cada término. Se logró observar que la altura juega un papel muy importante en el aprendizaje de la tarea en robots cuadrúpedos, así mismo, se nota que el término de finalización de experimentos también juega un papel muy importante ya que presentó valores altos en comparación con las demás. Se observó que la altura es de gran importancia debido a que el robot debe de tener una referencia estable que le permita mover

CAPÍTULO 5. RESULTADOS

sus extremidades libremente, el robot debe ser posicionado erguido con las 4 extremidades parcialmente extendidas.



Figura 5.55: Promedio de pesos ocupados en los experimentos.

A continuación se presentan una serie de gráficos, los cuales nos indican el comportamiento de aprendizaje en la tarea de incorporación estable. Se muestran en primer lugar tres gráficos sobre la duración de cada episodio del entrenamiento. En la figura 5.56 se puede observar el promedio de los pasos por episodio durante el entrenamiento, esta gráfica tiene una media móvil de 6 periodos para fines de visualización. La gráfica despliega la evolución de la duración de cada episodio medida en pasos de simulación. Entre mas tiempo dure activo el robot en el mismo episodio sin incurrir en una posición de reinicio, más pasos por episodio se registran, teniendo en mente que el limite se definió en 500 pasos de simulación por episodio. En la gráfica podemos observa que el experimento 14, 19 y 20 se acercan mas al valor máximo de 500, esto quiere decir que encontró una configuración en la que evita caerse o inclinarse de más.

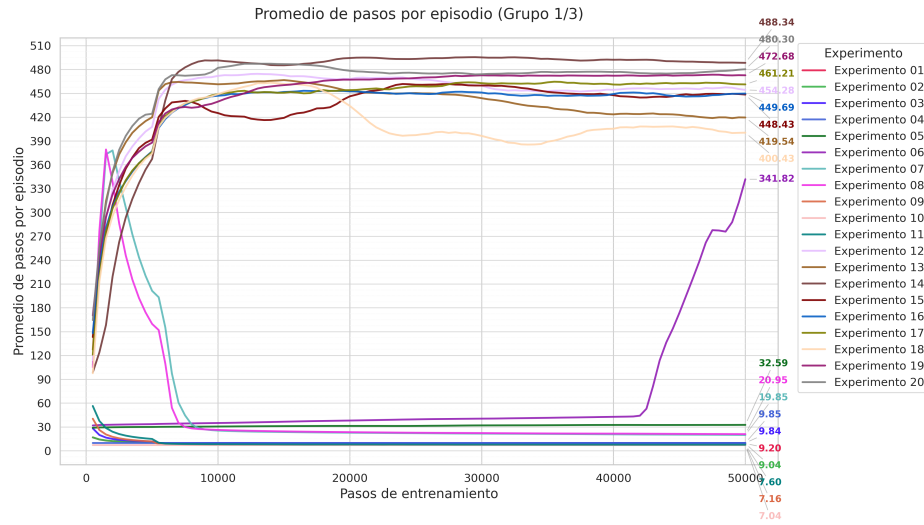


Figura 5.56: Gráfica del promedio de pasos de simulación en cada episodio para el grupo 1.

El grupo 2 presenta valores más estables en los entrenamientos, es decir, el robot cuadrúpedo aprendió a no realizar acciones que incurran en el reinicio del episodio. En la gráfica 5.57 se puede observar que el experimento 25 y 26 llegan al valor de 488.70 y 491.25 respectivamente, este valor es alto pero solo indica que el robot no ha realizado alguna acción que reinicie el episodio, para saber si realmente es funcional la política aprendida, se debe de visualizar las gráficas de recompensa.

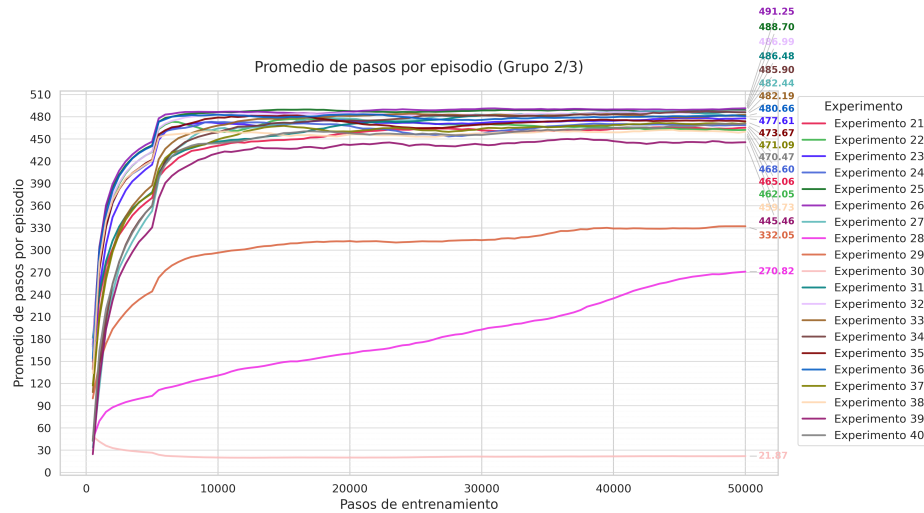


Figura 5.57: Gráfica del promedio de pasos de simulación en cada episodio para el grupo 2.

En la figura 5.58 se puede observar que varios experimentos encontraron un óptimo local, ya que se estabilizan en un valor por debajo de los 500 pasos por episodio, lo cual

CAPÍTULO 5. RESULTADOS

indica que después de un punto, el robot hace una acción que reinicia el episodio, es decir, si sobrepaso el umbral de inclinación en roll o pitch. Aunque los experimentos 45 y 46 muestran un buen crecimiento, no son constantes.

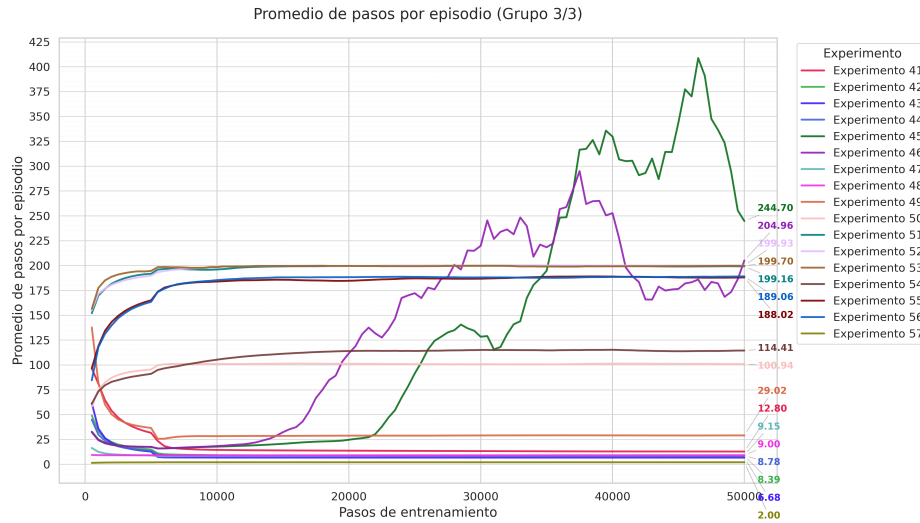


Figura 5.58: Gráfica del promedio de pasos de simulación en cada episodio para el grupo 3.

Se presentan las gráficas de recompensa en las cuales se observo y corroboro el desempeño de los agentes. En la figura 5.59 se observan los promedios de recompensa para cada experimento del grupo 1, donde destacan los experimentos 19, 13 y 12, por obtener mayor recompensa en la tarea, además, los experimentos 12 y 19 mostraron a su vez mayor desempeño en la gráfica 5.56.

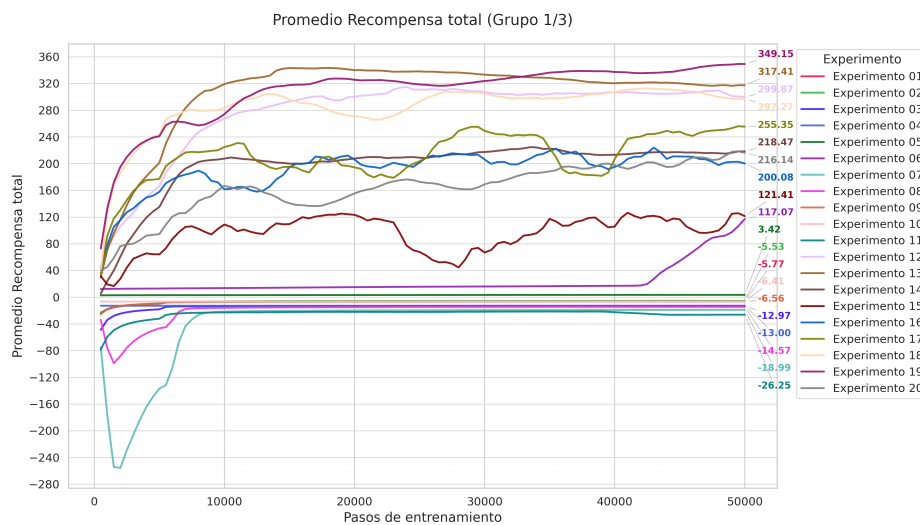


Figura 5.59: Gráfica del promedio de recompensa por episodio para el grupo 1.

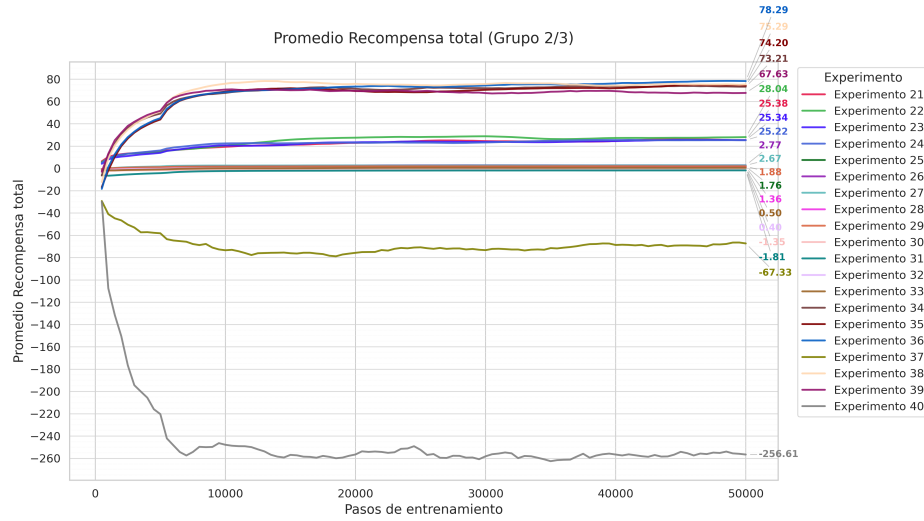


Figura 5.60: Gráfica del promedio de recompensa por episodio para el grupo 2.

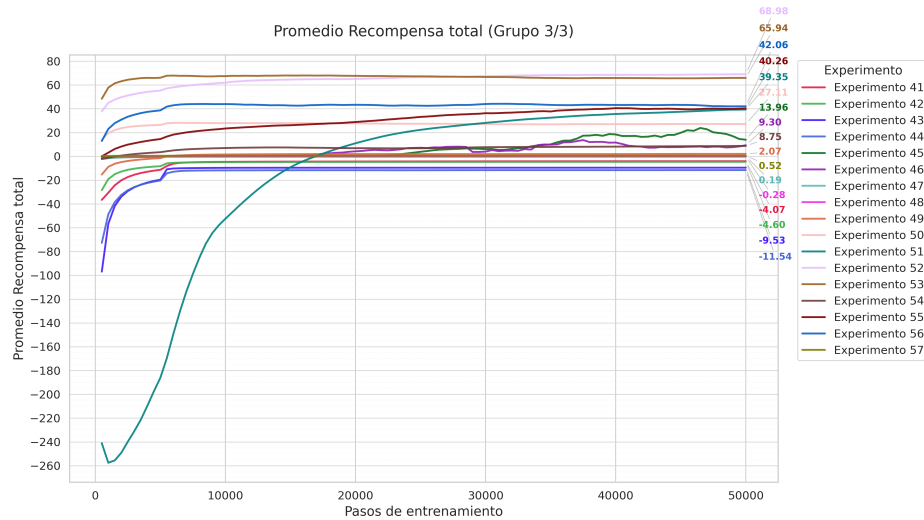


Figura 5.61: Gráfica del promedio de recompensa por episodio para el grupo 3.

Los grupos 2 y 3 no presentaron algún modelo que concuerde entre las gráficas de pasos por episodio y recompensa. Además, se lo gro aprender la política que realiza la tarea de incorporarse y mantenerse erguido para los experimentos 19, 12 y 20, aunque la mayoría si lograban mantenerse de pie, realizaban la tarea con algún defecto, como no estar completamente erguido o tener una pata levantada. En la tabla 5.3 se presentan los parámetros del experimento 19, el cual logró aprender una política que realiza la tarea con mejor desempeño, tanto en las gráficas como visualmente en el simulador.

CAPÍTULO 5. RESULTADOS

Tabla 5.3: Configuración de parámetros para el algoritmo PPO

Air_{time}	Contacts	Height	JP_{limit}	Roll	Pitch	Yaw	Terminating
-5	5	-50	-3.0	-0.3	-0.3	-0.3	-50

La política que se aprendió con el experimento 19, logra mantener el robot en una posición erguido y estable, aunque todas los experimentos que se realizaron presentan un control con mucha variación en la salida, el agente tiene desviación estándar alta por que lo que sigue explorando y como consecuencia se generan oscilaciones en las extremidades. A continuación se presentan las gráficas de la evolución de la desviación estándar de la política para cada experimento.

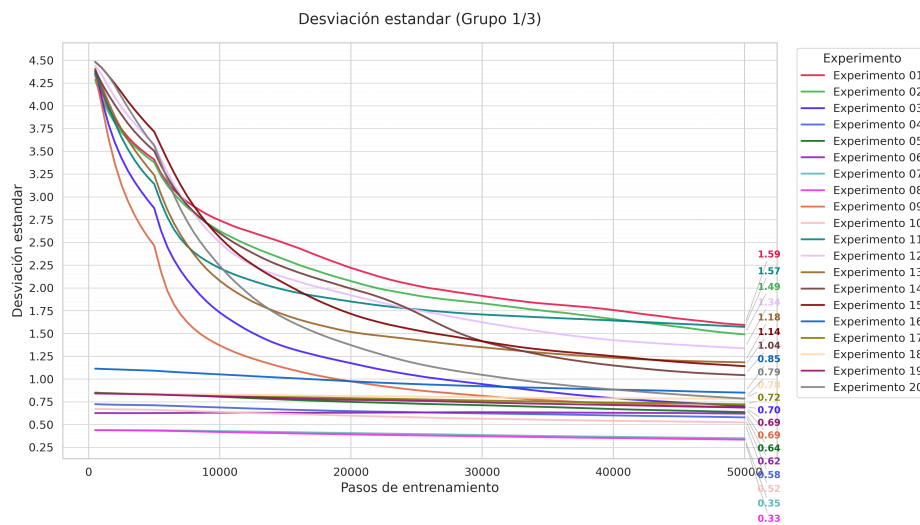


Figura 5.62: Gráfica de la desviación estándar de la política en cada experimento en el grupo 1.

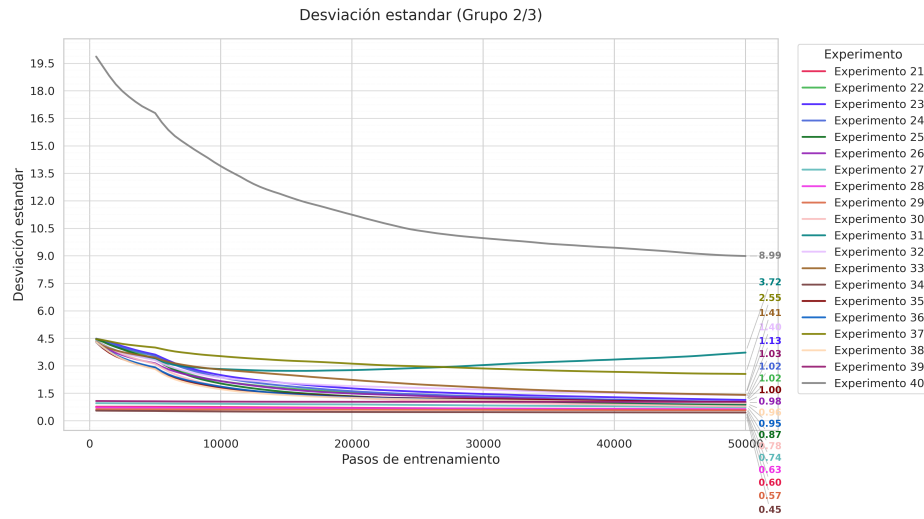


Figura 5.63: Gráfica de la desviación estándar de la política en cada experimento en el grupo 2.

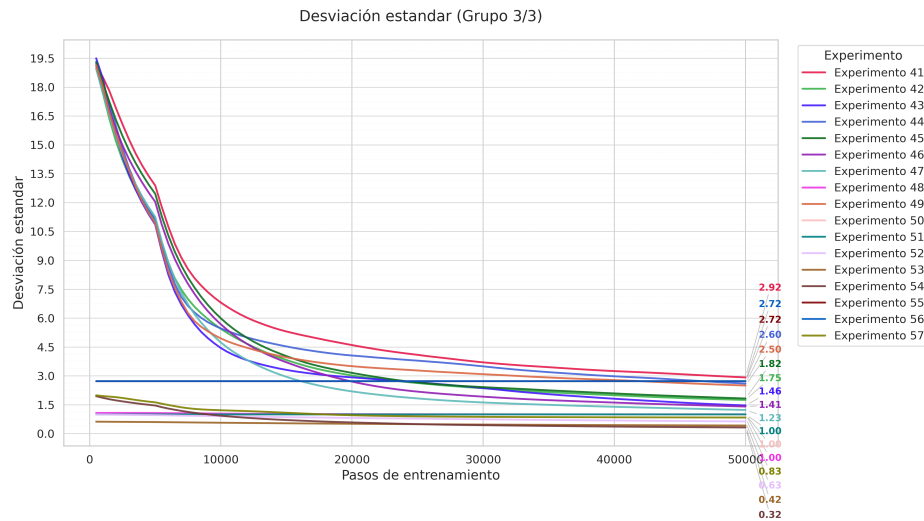


Figura 5.64: Gráfica de la desviación estándar de la política en cada experimento en el grupo 3.

El entrenamiento se realizó con una posición aleatoria de cada una de sus extremidades y para comprobar que el robot podría levantarse y mantenerse erguido se propuso iniciar al robot con sus extremidades en una posición predeterminada de 90° grados. Se observa en la figura 5.65 la secuencia de pasos que el robot realizó, cumpliendo completamente con la tarea para la cual fue entrenado.

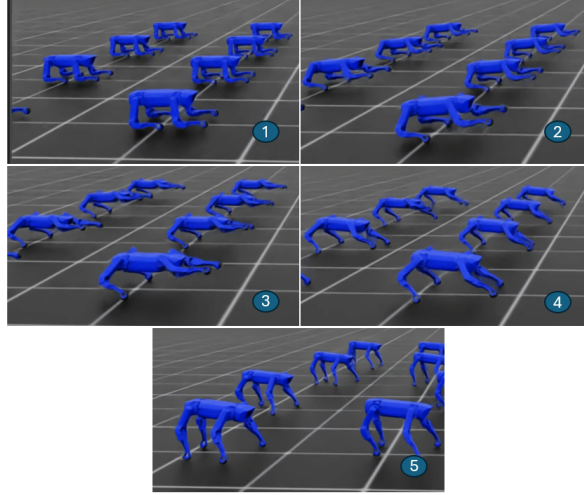


Figura 5.65: Secuencia de pasos que realiza el robot con la política aprendida.

5.7.2. Aprendizaje de Política de balanceo del torso

Para realizar el aprendizaje de una política que logre balancear el torso del robot y lo mantenga horizontalmente en todo momento, se definió un escenario en donde se pueda variar la inclinación de la superficie sobre la cual estará posicionado el robot cuadrúpedo.

Anteriormente se diseñó y configuró una plataforma en el simulador Isaac Lab para esta tarea. Se utilizó el algoritmo PPO, con los parámetros encontrados en el aprendizaje de la política de incorporación estable mostrados en la tabla 5.3.

Este experimento se basó en incentivar el aprendizaje de la tarea solo con los datos propioceptivos del robot; esto es, solo los sensores con los que cuenta, como posición de los motores, altura del torso al suelo, contacto de los pies con el suelo y rotaciones del torso con ayuda de una unidad inercial (IMU) que proporciona mediciones de los ángulos de Euler.

Además, se realizó una exploración de diferentes arquitecturas de redes neuronales para los agentes. El objetivo fue definir la mejor de ellas, considerando que serán utilizadas en tarjetas de desarrollo con poder de cómputo reducido para ejecutar las tareas de balance y control, por lo que la exploración se limitó a redes de 2 capas desde 10 neuronas por capa hasta 200 neuronas por capa. Todos los entrenamientos se realizaron con 1024 ambientes paralelos de Isaac lab, esto nos ayudó a recabar mas información en cada paso de simulación y a entrenar al agente con más datos y experiencias de distintas configuraciones. El entrenamiento se realizó posicionando 1024 instancias del robot cuadrúpedo, cada una sobre un plataforma que varia su inclinación tanto en el eje x como en el eje y , que son los ángulos de euler, la forma de variar estos ángulos fue determinada por una función que muestrea el valor de una distribución uniforme de un rango, en este caso el rango utilizado fue de $[-35, 35]$. Se configuró una duración de 500 pasos de simulación por cada

episodio de entrenamiento, una vez alcanzados estos el ambiente se reiniciaba posicionando al robot de manera erguida sobre la plataforma y se cambiaba la inclinación de esta; cuando se alcanzaban los 50,000 pasos de simulación se daba por terminado el experimento. En la figura 5.66 se observa el escenario con las 1024 instancias de robot posicionados en cada plataforma con diferente inclinación. La función de recompensa utilizada para este experimento es la presentada en la ecuación 5.1.

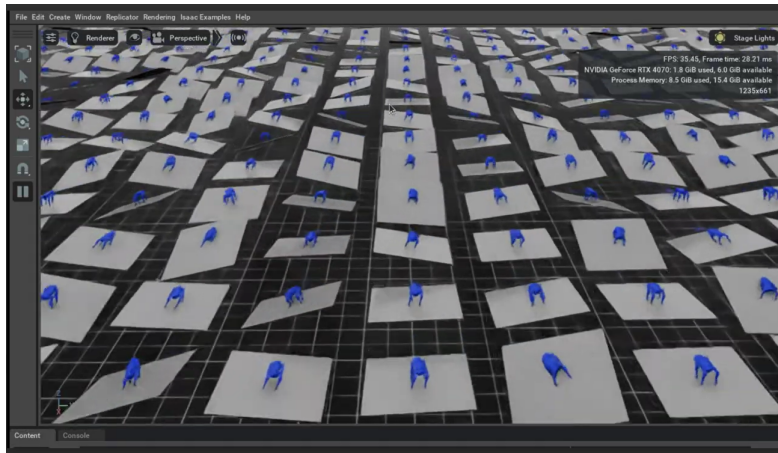


Figura 5.66: Escenario de entrenamiento con 1024 instancias de robot posicionados en una plataforma con diferente inclinación.

Para este experimento se posicionó al robot ya erguido, esto con la finalidad de que el agente se enfocara a mantener la altura y la orientación del torso de manera horizontal, ya que la plataforma cambiaba su inclinación en los ángulos roll y pitch. En la figura 5.67 se presenta la gráfica de promedio de pasos por episodio de cada uno de los experimentos, es posible notar que todos tuvieron un desempeño en el que el robot permanece erguido la mayor parte del tiempo ya que el máximo de pasos por episodio son 500 y al finalizar el entrenamiento se aprecia que la mayoría se encuentra por encima de 450 pasos por episodio. Para llegar a una conclusión sobre la arquitectura con mejor desempeño y optimizada es necesario comparar la gráfica de recompensas.

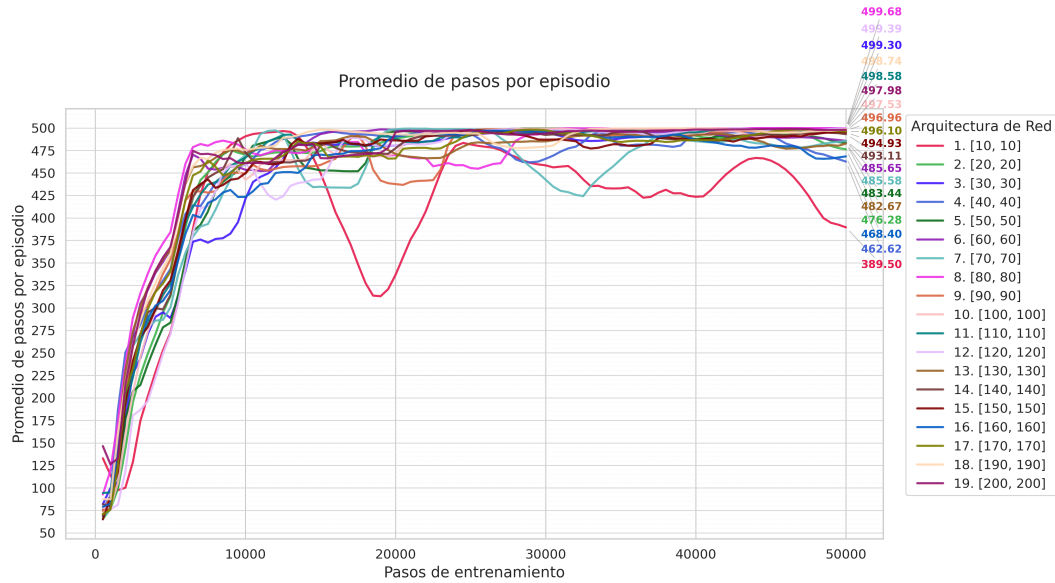


Figura 5.67: Promedio de pasos por episodio para cada una de las arquitecturas propuestas.

En la figura 5.68 se observan las recompensas totales de cada episodio durante el entrenamiento, la evolución de las curvas denota un aprendizaje continuo, mostrando valores cada vez más altos. La arquitectura [80,80] neuronas para la política y el estimador de valor presentó el mejor desempeño seguida por arquitectura [170, 170] y [200, 200].

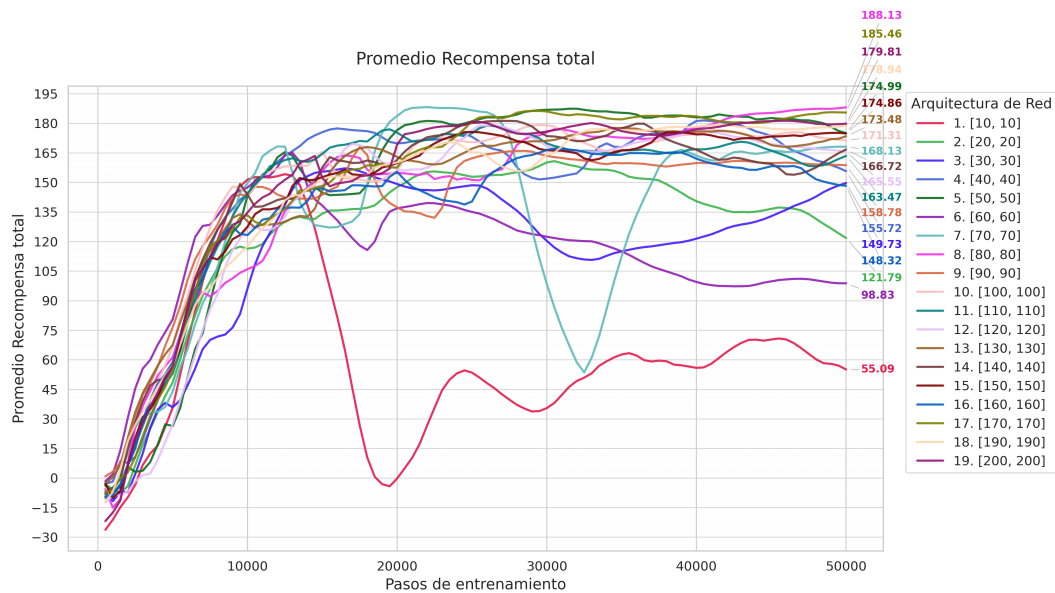


Figura 5.68: Promedio de recompensas total de cada una de las arquitecturas propuestas.

En la figura 5.69 se presenta la gráfica de la desviación estándar de la política para cada arquitectura, esta nos muestra cuanta exploración sigue realizando el agente para las nuevas observaciones. Se puede observar valores mas conservadores en comparación con la tarea de incorporación estable y en la simulación se logro mitigar las oscilaciones de las extremidades del robot. La arquitectura de 80 neuronas presenta un valor de 0.32 lo cual es considerable para mantener la exploración mientras cumple con la tarea.

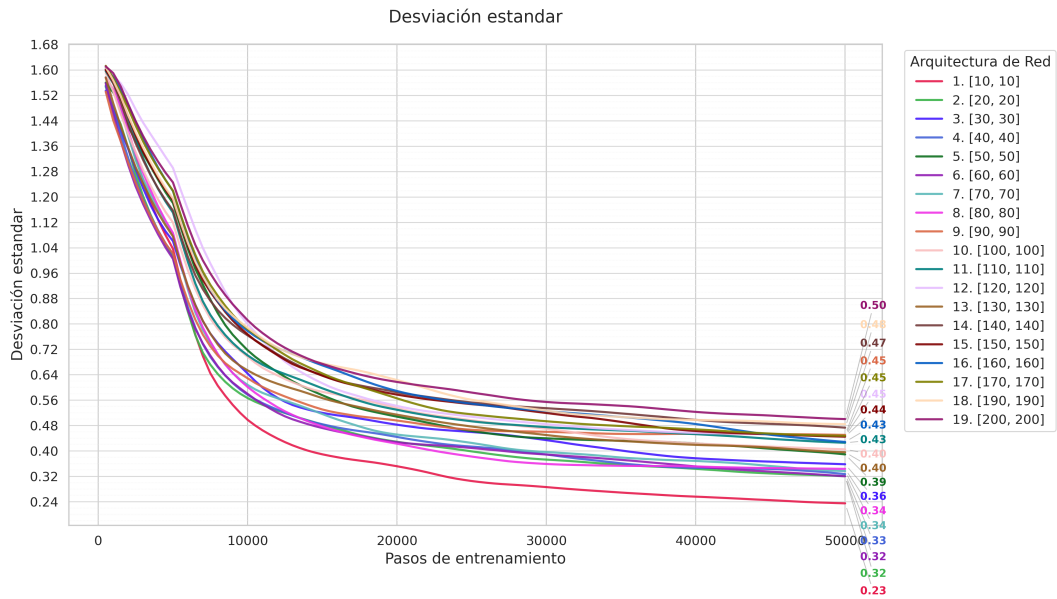


Figura 5.69: Desviación estándar de la política de cada arquitectura propuesta.

En las siguientes figuras se presentan las gráficas de tasa de aprendizaje 5.70, pérdida de la política 5.71 y pérdida del estimador de valor 5.72 de cada una de las arquitecturas. Se puede observar que la tasa de aprendizaje va decrementando y acercándose a cero, esto para lograr que la política no tenga cambios bruscos en aprendizaje. La pérdida de la política se va acercando a cero, esto es crucial y denota que la política elige acciones cada vez más acertadas. Por último, se observa que la pérdida del estimador de valor para cada arquitectura oscila en valores altos, esto se debe a que el escenario que fue propuesto para el entrenamiento es muy dinámico, es decir, el punto de inicio del agente siempre cambia, por lo tanto, no ejecuta la tarea en las mismas condiciones y esto produce que el estimador de valor mantenga errores altos.

CAPÍTULO 5. RESULTADOS

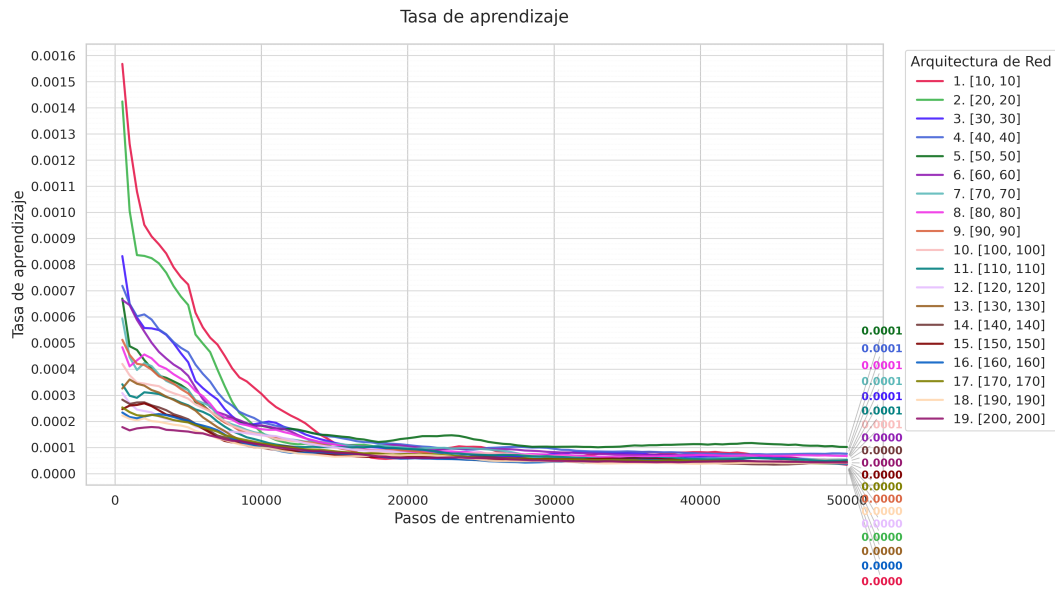


Figura 5.70: Tasa de aprendizaje para la política como para el estimador de valor.

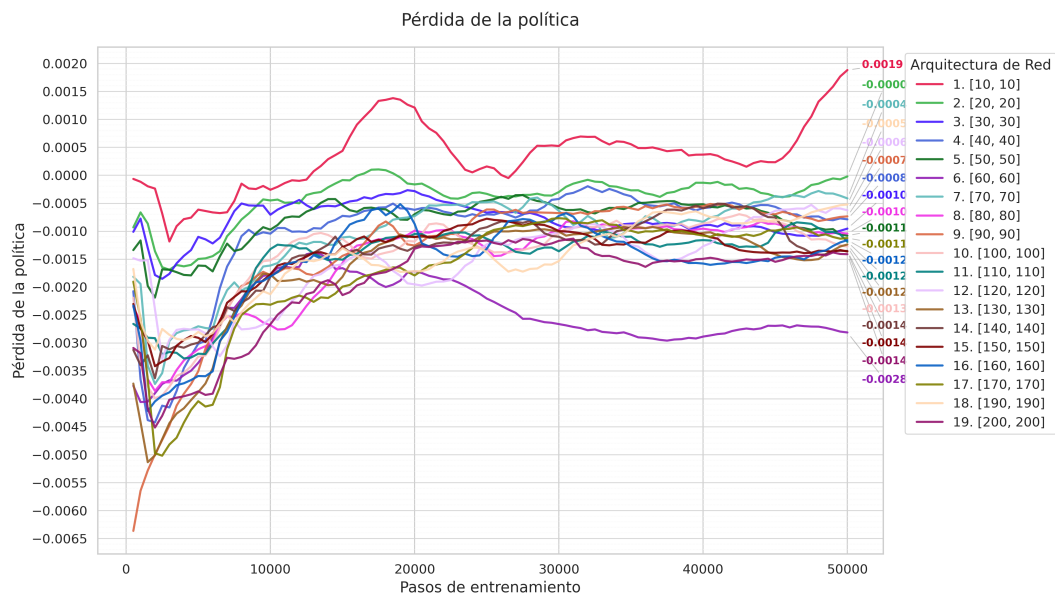


Figura 5.71: Pérdida de la política para cada arquitectura utilizada en el experimento.

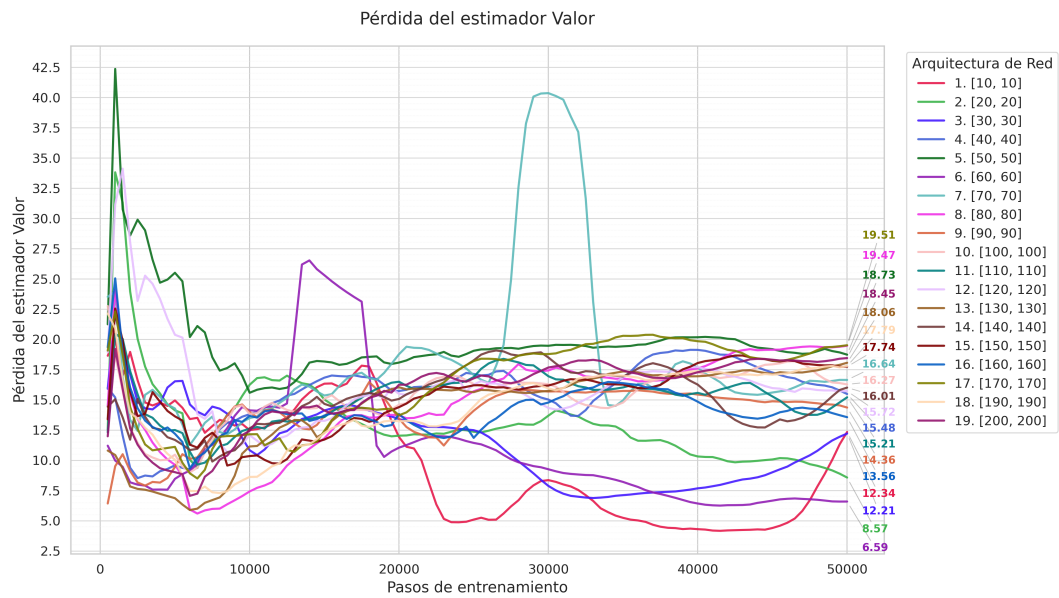


Figura 5.72: Pérdida del estimador de valor para cada arquitectura utilizada en el experimento.

En la figura 5.73 se observa al robot cumpliendo con la tarea de balance en una plataforma con diferentes inclinaciones. El robot mantiene su torso en una orientación horizontal mitigando la inclinación de la plataforma con un error de aproximadamente 8.

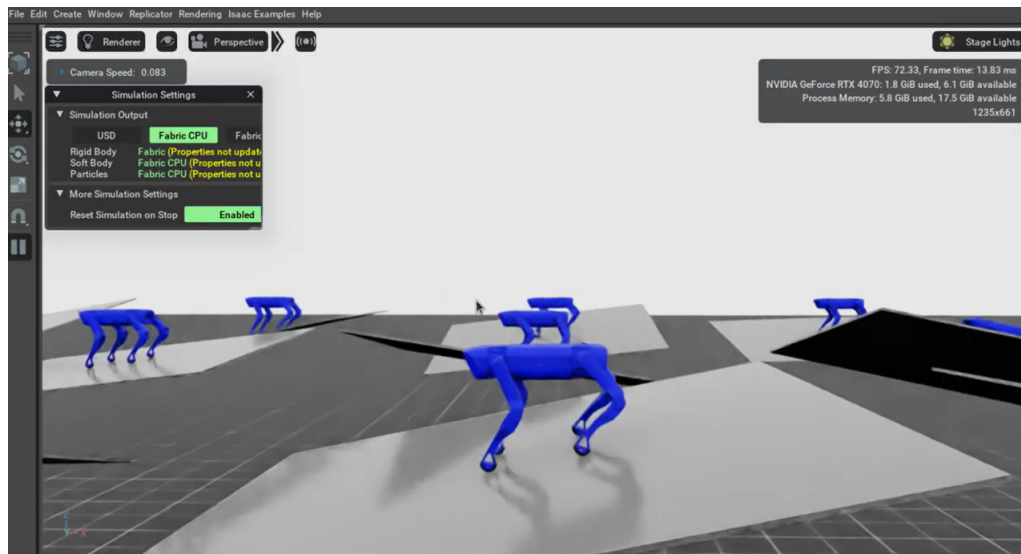


Figura 5.73: Tarea de balance en una plataforma con diferentes inclinaciones.

Conclusiones y Trabajo Futuro

En este trabajo de tesis se modificó y manufacturó un robot cuadrúpedo a escala en impresión 3D. Este robot fue instrumentado con sensores para poder percibir el contacto con el suelo en cada una de sus extremidades. Además, se dotó de sensores ultrasónicos para detectar distancias, así como de un giroscopio para poder detectar las rotaciones que hay en el torso del robot. Por otro lado, se logró manufacturar completamente y ensamblar el robot para poder realizar tareas de marcha y balance.

En ese trabajo se mostró la metodología para transferir el conocimiento ya existente en un robot caudrúpedo diseñado por la empresa Freenove, el cual puede caminar hacia adelante, hacia atrás y hacia los lados, esto lo realiza utilizando técnicas de control clásico como el calculo de una trayectoria que las extremidades del robot deben seguir y como consecuencia logra generar patrones de marcha. Este robot cuenta con una función llamada balanceo la cual tiene como finalidad mantener el torso del robot en todo momento de forma horizontal, esto fue de gran ayuda ya que se logró obtener una base de datos del comportamiento del controlador que utilizan para este cometido y las mediciones de los sensores (giroscopio y acelerómetro) con los que cuenta el robot.

Al mismo tiempo se realizó una comparación de varias arquitecturas de redes neuronales mediante el entrenamiento con la base de datos adquirida y se logró identificar el número de neuronas y capas óptimo para que la red neuronal tuviera un buen desempeño, presentando un error muy bajo. Se logró transferir este conocimiento a otra plataforma robótica cuadrúpeda tomando algunas consideraciones como los ejes de referencia inercial y los ejes de referencia de los motores, con estas consideraciones el robot cuadrúpedo sin conocimientos del modelo dinámico ni de su morfología logró imitar el comportamiento del balance del robot freenove, teniendo un buen desempeño a la hora de ejecutar la tarea, aunque se tuvieron que calibrar los motores manualmente, pero es un paso que podría posteriormente automatizarse. Además durante el proceso de transferencia de conocimiento se observó que existían patrones de marcha lo cual es uno de los objetivos de esta tesis, también se logró identificar cuáles serían las variables que promueven a un robot de este tipo generar patrones de marcha.

Se desarrollo un ambiente de simulación en el software Isaac Lab, el cual cuenta con un potente simulador de física y la ventaja de utilizar la gpu para la simulación. Se logró diseñar la tarea y el escenario para que un robot cuadrúpedo realice el entrenamiento para aprender a balancear su torso. El ambiente consta de una plataforma elevada la cual varia su inclinación aleatoriamente. Por otra parte, se presentaron graficas del comportamiento de las extremidades del robot al variar los ángulos Roll y Pitch, donde se encontró que emergen patrones de marcha documentados en la literatura y se logró evaluar en el ambiente de simulación dichos patrones de marcha.

Si bien no son secuencias o patrones robustos para que un robot camine, es importante tener en cuenta que las trayectorias que generan las patas están definidas por una sola variable (ángulo Roll ó ángulo Pitch) esto reduce la complejidad de programar dichas trayectorias para cada una de las patas y evita la necesidad de sincronizar los movimientos de cada pata. Los enfoques actuales para determinar los patrones de marcha suelen recurrir al modelo dinámico y a trayectorias predefinidas que deben seguir las patas, por lo que resulta costoso en términos de carga computacional y sobre todo en complejidad del algoritmo al tener en cuenta la sincronización de cada movimiento. Este método no es robusto, pero faltan realizar diversas pruebas para concluir el grado de robustez o eficiencia, ya que se podría seguir explorando el tipo de funciones se pueden generar para los ángulos roll y pitch.

En el simulador Isaac Sim se realizaron dos experimentos. En primer lugar, para evaluar qué variables físicas del robot aportan mayor información o ayudan a que el robot aprenda la tarea de pararse y quedar erguido, así como la tarea de balancear su torso en una plataforma que cambia su inclinación.

Para esto se utilizó la librería skrl y el algoritmo PPO. Se mantuvieron las configuraciones que en el estado del arte han mostrado buen desempeño para tareas de control de robots manipuladores y cuadrúpedos. Se logró obtener una política que aprendió la tarea de incorporarse y mantenerse erguido en un suelo plano.

Asimismo, se realizó una exploración de la importancia que tiene cada una de las variables propuestas en estos experimentos para lograr el aprendizaje óptimo. De esta forma, se obtuvieron los parámetros para posteriormente realizar un experimento en donde se aprendió la tarea de balance del torso en una plataforma con diferentes inclinaciones.

Se logró aprender una política que mantiene el torso del robot cuadrúpedo de manera horizontal pese a cada una de las variaciones de inclinación de la plataforma en la que está sobrepuesto. Finalmente, se logró concluir que, para este tipo de tareas, basta una arquitectura de 2 capas ocultas con 80 neuronas cada una para poder realizar con éxito esta tarea.

Como trabajo futuro, se plantea transferir las políticas aprendidas con el algoritmo PPO en el simulador hacia el robot manufacturado en 3D, tanto la tarea de incorporación y mantenerse erguido, así como para la de balance en una plataforma con diferentes inclinaciones.

CONCLUSIONES

Por otro lado, se pretende tener todo un ecosistema en donde se genere un gemelo digital del robot para realizar más experimentaciones y corroborar todas las simulaciones en el gemelo digital, para posteriormente transferirlo al robot físico.

Bibliografía

- 4.0, C. B. (2021). *Attribution 4.0 International*. <https://creativecommons.org/licenses/by/4.0/#ref-indicate-changes>
- Alexander, R. M. (1984). The Gaits of Bipedal and Quadrupedal Animals. *The International Journal of Robotics Research*, 3(2), 49-59. <https://doi.org/10.1177/027836498400300205>
- Barter, J. W., & Yin, H. H. (2021). Achieving natural behavior in a robot using neurally inspired hierarchical perceptual control. *iScience*, 24(9), 102948. <https://doi.org/10.1016/j.isci.2021.102948>
- Bellicoso, C. D., Gehring, C., Hwangbo, J., Fankhauser, P., & Hutter, M. (2016). Perception-less terrain adaptation through whole body control and hierarchical optimization. *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 558-564. <https://doi.org/10.1109/HUMANOIDS.2016.7803330>
- Bosworth, W., Kim, S., & Hogan, N. (2015). The MIT Super Mini Cheetah: A Small, Low-Cost Quadrupedal Robot for Dynamic Locomotion. *2015 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 1-8. <https://doi.org/10.1109/SSRR.2015.7443018>
- Bucher, D. (2009). Central Pattern Generators. En L. R. Squire (Ed.), *Encyclopedia of Neuroscience* (pp. 691-700). Academic Press. <https://doi.org/10.1016/B978-008045046-9.01944-6>
- Chai, H., Li, Y., Song, R., Zhang, G., Zhang, Q., Liu, S., Hou, J., Xin, Y., Yuan, M., Zhang, G., & Yang, Z. (2021). A survey of the development of quadruped robots: Joint configuration, dynamic locomotion control method and mobile manipulation approach. *Biomimetic Intelligence and Robotics*, 100029. <https://doi.org/10.1016/j.birob.2021.100029>
- Chen, J., Xu, K., & Ding, X. (2022). Adaptive gait planning for quadruped robot based on center of inertia over rough terrain. *Biomimetic Intelligence and Robotics*. <https://doi.org/10.1016/j.birob.2021.100031>
- Chen, Y., Hou, W.-Q., Wang, J., Wang, J.-W., & Ma, H.-x. (2015). A strategy for push recovery in quadruped robot based on reinforcement learning. *2015 34th Chinese*

- Control Conference (CCC)*, 3145-3151. <https://doi.org/10.1109/ChiCC.2015.7260125>
- Cheng, R., Orosz, G., Murray, R., & Burdick, J. (2019). End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 3387-3395. <https://doi.org/10.48550/arXiv.1903.08792>
- Daly, B. P., Eichen, D. M., Bailer, B., Brown, R. T., & Buchanan, C. L. (2012). Central nervous system. En V. S. Ramachandran (Ed.), *Encyclopedia of Human Behavior (Second Edition)* (2.^a ed., pp. 454-459). Academic Press. <https://doi.org/10.1016/B978-0-12-375000-6.00084-7>
- Endo, G., Morimoto, J., Matsubara, T., Nakanishi, J., & Cheng, G. (2008). Learning CPG-based biped locomotion with a policy gradient method: Application to a humanoid robot. *The International Journal of Robotics Research*, 27(2), 213-228. <https://doi.org/10.1177/0278364907084980>
- Freenove. (2016). *Robot Dog Kit*. <https://freenove.com/>
- Gangapurwala, S., Mitchell, A., & Havoutis, I. (2020). Guided constrained policy optimization for dynamic quadrupedal robot locomotion. *IEEE Robotics and Automation Letters*, 5(2), 3642-3649. <https://doi.org/10.1109/LRA.2020.2979656>
- Gehring, C., Coros, S., Hutter, M., Bloesch, M., Hoepflinger, M. A., & Siegwart, R. (2013). Control of dynamic gaits for a quadrupedal robot. *2013 IEEE International Conference on Robotics and Automation*, 3287-3292. <https://doi.org/10.1109/ICRA.2013.6631035>
- Gonog, L., & Zhou, Y. (2019). A review: Generative adversarial networks [ISSN: 2158-2297]. *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 505-510. <https://doi.org/10.1109/ICIEA.2019.8833686>
- Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., & Levine, S. (2019). Learning to Walk via Deep Reinforcement Learning [Preprint]. *arXiv preprint arXiv:1812.11103*. <http://arxiv.org/abs/1812.11103>
- Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor [Preprint]. *arXiv preprint arXiv:1801.01290*. <http://arxiv.org/abs/1801.01290>
- Hu, B., Shao, S., Cao, Z., Xiao, Q., Li, Q., & Ma, C. (2019). Learning a Faster Locomotion Gait for a Quadruped Robot with Model-Free Deep Reinforcement Learning. *Proceedings of the 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 1097-1102. <https://doi.org/10.1109/ROBIO49542.2019.8961651>
- Hutter, M., Gehring, C., Jud, D., Lauber, A., Bellicoso, C. D., Tsounis, V., Hwangbo, J., Bodie, K., Fankhauser, P., Bloesch, M., Diethelm, R., Bachmann, S., Melzer, A., & Hoepflinger, M. (2016). ANYmal – a highly mobile and dynamic quadrupedal robot. *Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 38-44. <https://doi.org/10.1109/IROS.2016.7758092>
- Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., & Hutter, M. (2019). Learning agile and dynamic motor skills for legged robots [arXiv:

- 1901.08652]. *Science Robotics*, 4(26), eaau5872. <https://doi.org/10.1126/scirobotics.aau5872>
- Ijspeert, A. J. (2014). Biorobotics: Using robots to emulate and investigate agile locomotion. *Science*, 346(6206), 196-203. <https://doi.org/10.1126/science.1254486>
- Iscen, A., Caluwaerts, K., Tan, J., Zhang, T., Coumans, E., Sindhwani, V., & Vanhoucke, V. (2018). Policies Modulating Trajectory Generators [ISSN: 2640-3498]. *Proceedings of the 2nd Conference on Robot Learning*, 916-926. <https://proceedings.mlr.press/v87/iscen18a.html>
- Jain, D., Iscen, A., & Caluwaerts, K. (2019). Hierarchical reinforcement learning for quadruped locomotion. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 7551-7557. <https://doi.org/10.1109/IROS40897.2019.8967913>
- Jinhui, R., Honglei, A., Taihui, Z., Yangzhen, C., & Hongxu, M. (2016). Single leg operational space control of quadruped robot based on reinforcement learning [No date]. *Unpublished manuscript*, 1-6.
- Jones, W., Blum, T., & Yoshida, K. (2020). Adaptive Slope Locomotion with Deep Reinforcement Learning. *2020 IEEE/SICE International Symposium on System Integration (SII)*, 546-550. <https://doi.org/10.1109/SII46433.2020.9025928>
- Katz, B., Di Carlo, J., & Kim, S. (2019). Mini Cheetah: A Platform for Pushing the Limits of Dynamic Quadruped Control. *2019 International Conference on Robotics and Automation (ICRA)*, 6295-6301. <https://doi.org/10.1109/ICRA.2019.8793865>
- Kim, D.-y. (2021). *Spotmicro - robot dog*. https://www.thingiverse.com/thing:3445283#google_vignette
- Kim, H., Kang, T., Vo, G. L., & Choi, H. R. (2005). Gait Planning of Quadruped Walking and Climbing Robot for Locomotion in 3D Environment. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2733-2738. <https://doi.org/10.1109/ROBOT.2005.1570527>
- Kimura, H., Fukuoka, Y., & Cohen, A. (2007). Adaptive Dynamic Walking of a Quadruped Robot on Natural Ground Based on Biological Concepts. *International Journal of Robotics Research*, 26(5), 475-490. <https://doi.org/10.1177/0278364907078089>
- Kober, J., Bagnell, J., & Peters, J. (2013). Reinforcement Learning in Robotics: A Survey. *International Journal of Robotics Research*, 1238-1274.
- Lele, A. S., Fang, Y., Ting, J., & Raychowdhury, A. (2020). Learning to Walk: Bio-Mimetic Hexapod Locomotion via Reinforcement-Based Spiking Central Pattern Generation. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4), 536-545. <https://doi.org/10.1109/JETCAS.2020.3033135>
- Leziart, P.-A., Flayols, T., Grimminger, F., Mansard, N., & Souères, P. (2021). Implementation of a Reactive Walking Controller for the New Open-Hardware Quadruped Solo-12, 5007-5013. <https://doi.org/10.1109/ICRA48506.2021.9561559>
- Li, M., Jiang, Z., Wang, P., Sun, L., & Ge, S. S. (2014). Control of a quadruped robot with bionic springy legs in trotting gait. *Journal of Bionic Engineering*. [https://doi.org/10.1016/S1672-6529\(14\)60043-3](https://doi.org/10.1016/S1672-6529(14)60043-3)

- Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2022). A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33, 6999-7019. <https://doi.org/10.1109/TNNLS.2021.3084827>
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning [Preprint]. *arXiv preprint arXiv:1509.02971*, 1-14.
- Liu, T., Li, L., Shao, G., Wu, X., & Huang, M. (2020). A novel policy gradient algorithm with PSO-based parameter exploration for continuous control. *Engineering Applications of Artificial Intelligence*, 90, 103525. <https://doi.org/10.1016/j.engappai.2020.103525>
- Luo, Y.-S., Soeseno, J. H., Chen, T. P.-C., & Chen, W.-C. (2020). CARL: Controllable agent with reinforcement learning for quadruped locomotion. *ACM Transactions on Graphics*, 39(4), 38:1-38:10. <https://doi.org/10.1145/3386569.3392433>
- Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66), eabm6074. <https://doi.org/10.1126/scirobotics.abm6074>
- Mitchell, A. L., Engelcke, M., Jones, O. P., Surovik, D., Gangapurwala, S., Melon, O., Havoutis, I., & Posner, I. (2020). First steps: Latent-space control with semantic constraints for quadruped locomotion. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5343-5350. <https://doi.org/10.1109/IROS45743.2020.9340737>
- Nvidia. (2022). *Nvidia Isaac Lab*. Consultado el 10 de agosto de 2022, desde <https://developer.nvidia.com/isaac/lab>
- Nvidia. (2025). *Nvidia Isaac Sim*. Consultado el 10 de mayo de 2021, desde <https://developer.nvidia.com/isaac/sim>
- OpenRobotics. (2025a). *ROS*. Consultado en 2025, desde <https://www.ros.org/blog/why-ros/>
- OpenRobotics. (2025b). *ROS2 Documentación:Humble*. Consultado el 10 de octubre de 2022, desde <https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html>
- Palmer, L. R., & Orin, D. E. (2010). Intelligent Control of High-Speed Turning in a Quadruped. *Journal of Intelligent and Robotic Systems*, 27(2), 213-228. <https://doi.org/10.1007/s10846-009-9345-7>
- Park, C.-S., & Kim, J.-H. (2013). Stable modifiable walking pattern algorithm with constrained optimized central pattern generator. En J.-H. Kim, E. T. Matson, H. Myung & P. Xu (Eds.), *Robot Intelligence Technology and Applications 2012: An Edition of the Presented Papers from the 1st International Conference on Robot Intelligence Technology and Applications* (pp. 223-230). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-37374-9_22
- Patterson, J., & Gibson, A. (2017). *Deep Learning: A Practitioner's Approach*. O'Reilly.
- Pei, M., Wu, D., & Wang, C. (2020). Quadruped robot locomotion in unknown terrain using deep reinforcement learning. *Proceedings of the 2020 3rd International Conference*

- on Unmanned Systems (ICUS)*, 517-522. <https://doi.org/10.1109/ICUS50048.2020.9274920>
- Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., & Ng, A. (2009). ROS: an open-source Robot Operating System [No date]. *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*.
- Raibert, M., Blankespoor, K., Nelson, G., & Playter, R. (2008). BigDog, the rough-terrain quadruped robot [17th IFAC World Congress]. *IFAC Proceedings Volumes*, 41(2), 10822-10825. <https://doi.org/10.3182/20080706-5-KR-1001.01833>
- Rasamoelina, A. D., Adjailia, F., & Sincak, P. (2020). A review of activation function for artificial neural network. *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 281-286. <https://doi.org/10.1109/SAMI48414.2020.9108717>
- Saraf, P., Sarkar, A., & Javed, A. (2021). Terrain Adaptive Gait Transitioning for a Quadruped Robot using Model Predictive Control. *2021 26th International Conference on Automation and Computing (ICAC)*, 1-6. <https://doi.org/10.23919/ICAC50006.2021.9594065>
- Schilling, M., Melnik, A., Ohl, F. W., Ritter, H. J., & Hammer, B. (2021). Decentralized control and local information for robust and adaptive decentralized Deep Reinforcement Learning. *Neural Networks*, 144, 699-725. <https://doi.org/10.1016/j.neunet.2021.09.017>
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. <https://arxiv.org/abs/1707.06347>
- Semini, C., Barasuol, V., Goldsmith, J., Frigerio, M., Focchi, M., Gao, Y., & Caldwell, D. G. (2017). Design of the hydraulically actuated, torque-controlled quadruped robot HyQ2Max. *IEEE/ASME Transactions on Mechatronics*, 22(2), 635-646. <https://doi.org/10.1109/TMECH.2016.2616284>
- Sewak, M. (2019). *Deep Reinforcement Learning - Frontiers of Artificial Intelligence*. Springer.
- Singla, A., Bhattacharya, S., Dholakiya, D., Bhatnagar, S., Ghosal, A., Amrutur, B., & Kolathaya, S. (2019). Realizing learned quadruped locomotion behaviors through kinematic motion primitives. *2019 International Conference on Robotics and Automation (ICRA)*, 7434-7440. <https://doi.org/10.1109/ICRA.2019.8794179>
- Sun, H., Fu, T., Ling, Y., & He, C. (2021). Adaptive quadruped balance control for dynamic environments using maximum-entropy reinforcement learning. *Sensors*, 21(17), 5907. <https://doi.org/10.3390/s21175907>
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press.
- Takemura, H., Deguchi, M., Ueda, J., Matsumoto, Y., & Ogasawara, T. (2005). Slip-adaptive walk of quadruped robot. *Robotics and Autonomous Systems*, 53, 124-141. <https://doi.org/10.1016/j.robot.2005.07.002>

- Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., & Vanhoucke, V. (2018). Sim-to-real: Learning agile locomotion for quadruped robots [arXiv:1804.10332]. *arXiv preprint arXiv:1804.10332*. <http://arxiv.org/abs/1804.10332>
- Tang, Y., Tan, J., & Harada, T. (2020). Learning agile locomotion via adversarial training. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6098-6105. <https://doi.org/10.1109/IROS45743.2020.9341777>
- Todd, D. J. (1985). *Walking Machines: An Introduction to Legged Robots*. Springer.
- Todd, D. J. (2013). *Walking Machines: An Introduction to Legged Robots*. Springer Science & Business Media.
- Truong, J., Yarats, D., Li, T., Meier, F., Chernova, S., Batra, D., & Rai, A. (2021). Learning navigation skills for legged robots with learned robot embeddings. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 484-491. <https://doi.org/10.1109/IROS51168.2021.9635911>
- Tsounis, V., Alge, M., Lee, J., Farshidian, F., & Hutter, M. (2020). DeepGait: Planning and Control of Quadrupedal Gaits Using Deep Reinforcement Learning [Conference Name: IEEE Robotics and Automation Letters]. *IEEE Robotics and Automation Letters*, 5(2), 3699-3706. <https://doi.org/10.1109/LRA.2020.2979660>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need [arXiv:1706.03762 [cs]]. *arXiv preprint arXiv:1706.03762*. <https://doi.org/10.48550/arXiv.1706.03762>
- Wang, M., Ruan, X., & Zhu, X. (2020). Heuristic gait learning of quadruped robot based on deep deterministic policy gradient algorithm. *2020 Chinese Automation Congress (CAC)*, 1046-1049. <https://doi.org/10.1109/CAC51589.2020.9326973>
- Wen, J., Wang, J., Chen, W., & Zhang, J. (2012). A gait planning approach for locomotion stability of four-legged robots. *2012 7th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 324-329. <https://doi.org/10.1109/ICIEA.2012.6360745>
- Winder, P. (2018). *Reinforcement learning: Industrial applications of intelligent agents*. O'Reilly.
- Yu, W., Turk, G., & Liu, C. (2018). Learning symmetric and low-energy locomotion. *ACM Transactions on Graphics*, 37(4). <https://doi.org/10.1145/3197517.3201397>
- Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A review of recurrent neural networks: LSTM cells and network architectures. *Neural Computation*, 31(7), 1235-1270. https://doi.org/10.1162/neco_a_01199
- Zhang, S., & Sutton, R. S. (2018). A deeper look at experience replay [Preprint]. *arXiv preprint arXiv:1712.01275*. <http://arxiv.org/abs/1712.01275>