



UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

Tesis para obtener el grado de:

Maestro en Robótica

**Control de estabilidad en la marcha de un robot bípedo
mediante ZMP y aprendizaje por refuerzo**

Presenta:

Ing. Miguel de Jesús Santiago Vásquez

Director de Tesis:

Dr. Oscar David Ramírez Cárdenas

Co-director de Tesis:

Dr. José Aníbal Arias Aguilar

H. Cd. de Huajuapán de León, Oaxaca, México, septiembre de 2026

Dedicatoria

*A mi esposa la **Ing. Fátima Itandehui Martínez Santos**, por su apoyo incondicional, su paciencia y la confianza que depositó en mí durante cada etapa de este proceso.*

A mis padres, por su amor, sus enseñanzas y los valores que me han transmitido. Su ejemplo de esfuerzo, responsabilidad y perseverancia ha sido una guía fundamental en mi formación personal y profesional.

A mi hermano, por su cariño. Gracias por estar presente y por motivarme a seguir adelante a lo largo de este camino.

Agradecimientos

En primer lugar, agradezco a Dios por concederme la fortaleza, la sabiduría y la perseverancia necesarias para afrontar cada desafío presentado a lo largo de este proceso. Su presencia fue mi guía y sustento en los momentos de dificultad, y me permitió culminar satisfactoriamente esta etapa de mi vida.

Agradezco profundamente a mi esposa por su amor, paciencia y apoyo incondicional. Su compañía, comprensión y confianza fueron fundamentales para superar los momentos difíciles y mantenerme firme en el cumplimiento de esta meta. Gracias por compartir cada esfuerzo, cada sacrificio y, finalmente, la alegría de este logro.

Expreso mi agradecimiento al Dr. Oscar David Ramírez Cárdenas, director de esta tesis, por su orientación, disposición y acompañamiento durante el desarrollo de la investigación.

Asimismo, agradezco al Dr. José Aníbal Arias Aguilar, codirector de la tesis, por el apoyo brindado, sus aportaciones técnicas y el tiempo dedicado a la revisión del proyecto.

Agradezco a la Universidad Tecnológica de la Mixteca y a la División de Estudios de Posgrado por proporcionar la formación académica y el entorno necesarios para llevar a cabo este trabajo. También reconozco el apoyo de los profesores que, mediante sus conocimientos, comentarios y experiencias, contribuyeron a mi formación profesional.

Agradezco profundamente a mi familia y a las personas que me acompañaron a lo largo de este proceso por su motivación y respaldo constante.

Finalmente, agradezco a la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI) por la beca otorgada para la realización de mis estudios de posgrado.

Resumen

La locomoción bípeda representa un problema complejo dentro de la robótica debido a la naturaleza no lineal del sistema, la coordinación de múltiples grados de libertad y la necesidad de mantener estabilidad dinámica durante el desplazamiento. En este contexto, el criterio del Punto de Momento Nulo (ZMP, por sus siglas en inglés) permite analizar la estabilidad del robot respecto al polígono de soporte. Sin embargo, su uso aislado depende de modelos precisos y puede resultar limitado ante la complejidad de la marcha. Por ello, este trabajo integra el criterio ZMP con aprendizaje por refuerzo para estudiar la generación de marcha estable en un robot bípedo simulado.

El objetivo de este trabajo fue diseñar y validar, en simulación, una metodología para la locomoción de un robot bípedo de 12 grados de libertad. Para ello, se combinan el modelado cinemático, el cálculo del centro de masa y del ZMP, y el entrenamiento de una política de control mediante aprendizaje por refuerzo. La finalidad es obtener una política capaz de producir desplazamiento sostenido y conservar condiciones de estabilidad durante la marcha.

La metodología incluyó el diseño del modelo del robot, el desarrollo del modelo cinemático mediante parámetros de Denavit-Hartenberg y la validación progresiva del sistema en distintos entornos de simulación. CoppeliaSim se utilizó para instrumentar y visualizar el ZMP, Gazebo/RVIZ para verificar el modelo robótico y, finalmente, Isaac Sim con Isaac Lab para realizar el entrenamiento masivo en paralelo. El algoritmo seleccionado fue *Proximal Policy Optimization* (PPO), incorporando el ZMP como un término de penalización dentro de la función de recompensa. Se evaluaron configuraciones de 500, 1000 y 4000 entornos paralelos, y posteriormente se realizó un análisis estadístico con cinco semillas distintas para verificar la robustez de los resultados.

Los resultados muestran que el nivel de paralelismo tuvo una influencia importante en la calidad de la política aprendida. La configuración con 4000 entornos fue la única que produjo locomoción sostenida de manera consistente y presentó un error promedio de seguimiento de velocidad sobre el eje x de 0.71 m/s, frente a 1.00 m/s y 0.97 m/s para las configuraciones con 500 y 1000 entornos, respectivamente. Esto representa una reducción relativa del error de 29.0 % respecto a 500 entornos y de 26.8 % respecto a 1000 entornos. Asimismo, el análisis con cinco semillas mostró que la configuración con 4000 entornos mantuvo el mejor compromiso entre reducción del error lineal, distancia

recorrida y generación de locomoción sostenida.

Se concluye que la hipótesis se cumplió parcialmente dentro del alcance de la simulación. La política entrenada con la función de recompensa propuesta fue capaz de producir marcha bípeda sostenida y conservar condiciones de estabilidad; además, se obtuvo una reducción relativa del error de seguimiento al incrementar el número de entornos paralelos. Sin embargo, el error de 0.71 m/s no puede considerarse bajo en términos absolutos, por lo que esta parte de la hipótesis no se cumplió completamente. La tendencia observada sugiere que el error podría reducirse mediante un mayor número de entornos e iteraciones, el ajuste de la función de recompensa y la optimización de los hiperparámetros, aunque no fue posible evaluar más de 4000 entornos debido a las limitaciones del equipo utilizado. Dado que los resultados se obtuvieron exclusivamente en simulación, la implementación en un prototipo físico y la reducción de la brecha *sim-to-real* se plantean como líneas futuras de investigación.

Índice general

Dedicatoria	III
Agradecimientos	V
Resumen	VII
1. Introducción	3
1.1. Antecedentes	4
1.2. Planteamiento del problema	9
1.3. Justificación	10
1.4. Hipótesis	10
1.5. Objetivos	11
1.5.1. Objetivo general	11
1.5.2. Objetivos específicos	11
1.6. Metas	11
1.7. Limitaciones de la tesis	12

1.8. Metodología	13
2. Marco teórico	17
2.1. Modelado	17
2.1.1. Cinemática directa	18
2.1.2. Cinemática inversa	18
2.1.3. Cinemática diferencial	19
2.1.4. Convención de Denavit-Hartenberg	20
2.1.5. Desacople cinemático	21
2.2. Movimientos de los miembros inferiores	21
2.3. Ciclo de la marcha	23
2.3.1. Fase de apoyo	23
2.3.2. Fase de balanceo	23
2.4. Centro de presión	24
2.5. Punto de momento nulo	27
2.6. Inteligencia artificial	28
2.6.1. Perceptrón.	29

Índice general	XI
2.6.2. Perceptrón multicapa	29
2.6.3. Aprendizaje máquina	30
2.6.4. Aprendizaje supervisado	31
2.6.5. Aprendizaje no supervisado	31
2.6.6. Aprendizaje por refuerzo	32
2.6.7. Proximal Policy Optimization	34
2.7. ROS	36
2.7.1. Nodo	37
2.7.2. Temas	37
2.8. Gazebo	37
2.9. Isaac Sim	38
3. Diseño y simulación	41
3.1. Diseño conceptual	41
3.2. Diseño CAD	44
3.3. Exportación a GAZEBO	45
3.3.1. Importación a Isaac Sim	49

4. Modelo matemático	53
4.1. Cinemática directa	53
4.2. Cinemática inversa	57
4.2.1. Cinemática de posición	57
4.2.2. Cinemática de orientación	59
4.3. Pruebas de lazo abierto	60
4.3.1. Cinemática directa	60
4.4. Cálculo de velocidades mediante la matriz Jacobiana	62
5. Punto de momento nulo y aprendizaje por refuerzo	65
5.1. Cálculo del Punto de Momento Nulo	65
5.2. Definición del espacio de estados	67
5.3. Definición de las acciones del agente	68
5.4. Selección de algoritmo de aprendizaje reforzado a utilizar	69
5.5. Arquitectura del ciclo de control	70
5.6. Diseño de la función de recompensa	71
5.7. Configuración experimental	77

Índice general	1
5.7.1. Condiciones iniciales del entorno durante el entrenamiento. . . .	77
5.7.2. Etapa de inferencia	80
6. Resultados	83
7. Conclusiones	97
7.1. Conclusiones por objetivo	97
7.2. Aportaciones	98
7.3. Trabajo Futuro	99
Referencias	106
Apéndices	107
A. Matriz de rotación de la pierna derecha	107
B. Programas en Python para cinemática directa y Jacobiano.	108
C. Material audiovisual complementario	120
D. Productos de investigación	120

Capítulo 1

Introducción

El desarrollo tecnológico en el ámbito de la robótica ha impulsado la creación de sistemas autónomos capaces de interactuar con su entorno y reproducir, en cierta medida, habilidades asociadas al movimiento humano. Dentro de este campo, la robótica humanoide busca emular la complejidad y versatilidad de la anatomía humana mediante mecanismos articulados y sistemas de control avanzados [1,2]. Esta línea de investigación resulta relevante por sus posibles aplicaciones en asistencia, rehabilitación, interacción humano-robot y ejecución de tareas en entornos diseñados para personas.

En particular, la locomoción bípeda constituye uno de los retos más importantes de la robótica humanoide, ya que requiere coordinar múltiples grados de libertad mientras se conserva la estabilidad dinámica del sistema [3]. A diferencia de robots con bases móviles o configuraciones con mayor número de apoyos, un robot bípedo debe alternar fases de soporte y balanceo, controlar el desplazamiento del centro de masa y evitar la pérdida de equilibrio durante la marcha. Por ello, el diseño de estrategias de control para robots que caminan sobre dos piernas exige considerar tanto el modelado cinemático y dinámico como criterios de estabilidad específicos.

Uno de los criterios más utilizados para analizar la estabilidad en robots con patas es el Punto de Momento Nulo (ZMP, por sus siglas en inglés), el cual permite evaluar la relación entre la dinámica del robot y la región de soporte definida por sus puntos de contacto con el suelo. No obstante, el uso exclusivo de métodos analíticos puede resultar limitado cuando el sistema presenta alta no linealidad, incertidumbre en los parámetros físicos o una gran cantidad de configuraciones posibles. En este contexto, el aprendizaje por refuerzo (RL, por sus siglas en inglés) ofrece una alternativa para entrenar políticas de control capaces de aprender comportamientos locomotores a partir de la interacción con un entorno simulado.

El avance reciente de la inteligencia artificial y el incremento en la capacidad de procesamiento han favorecido el uso de simuladores físicos para entrenar y evaluar algoritmos de control en robótica. La simulación permite reducir costos, evitar daños en

prototipos físicos y realizar pruebas repetidas bajo condiciones controladas. Sin embargo, también introduce una limitación importante: los resultados obtenidos en entornos virtuales no garantizan por sí mismos el mismo desempeño en un sistema real, debido a diferencias entre el modelo simulado y las condiciones físicas del mundo real.

Con base en lo anterior, esta tesis se enfoca en la generación de marcha y el análisis de estabilidad de un robot bípedo simulado, integrando el criterio ZMP dentro de una estrategia de aprendizaje por refuerzo. El propósito es estudiar cómo la combinación de criterios clásicos de estabilidad con técnicas de aprendizaje permite entrenar políticas capaces de producir locomoción bípeda sostenida, manteniendo la estabilidad durante el desplazamiento.

1.1 Antecedentes

En esta sección se analizarán trabajos realizados anteriormente para comprender la relevancia y el estado actual de estas áreas dentro del campo de estudio de esta investigación.

Kurazume et.al. [4] utilizan el ZMP para mantener la estabilidad del robot, pero proponen que el robot realice la caminata con las piernas rectas, esto mejora el consumo energético ya que las articulaciones de la rodilla tienen menos esfuerzo, se menciona que la estabilidad es más difícil al realizar la caminata con las piernas rectas ya que el centro de masa se encuentra en una posición más alta. El esquema de la caminata propuesta se muestra en la figura 1.1.

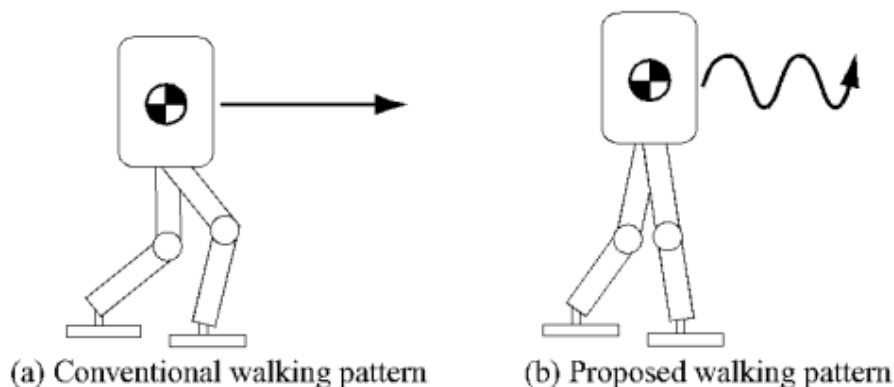


Figura 1.1: Modo de caminata propuesto por Karuzume et.al [4].

El modelado de un robot con múltiples grados de libertad (como es el caso de los robots bípedos) resulta ser una tarea complicada incluso utilizando manipuladores simbólicos como MATLAB [5]. Una alternativa a estos métodos es simplificar el modelo dinámico, por ejemplo, utilizando el modelo de un péndulo invertido [6] como se muestra en la figura 1.2.

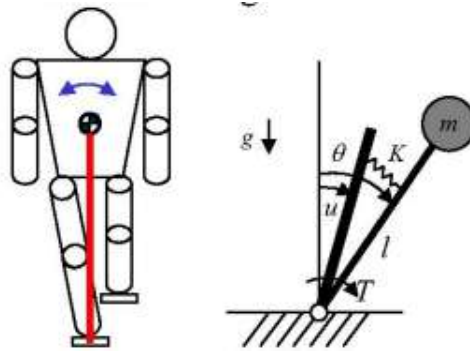


Figura 1.2: Modelo de péndulo invertido comparado con el modelo de un robot [6]

El criterio del ZMP y utilizar algún método de captura de movimiento humano para replicarlo en el robot es una técnica que se ha planteado para resolver el problema de la movilidad de robots bípedos, como se muestra en el trabajo de Deok-Won et.al. [2], se discute la generación de poses consecutivas en robots humanoides, esto se refiere a calcular una serie de posiciones estables del robot y reproducirlas consecutivamente para generar la marcha, este es un proceso que conlleva un alto costo computacional. Para evitar el complejo modelo analítico se propone emplear el criterio del ZMP con el fin de mantener el equilibrio del robot. Asimismo, se sugiere el uso de una herramienta de captura de movimiento para generar los movimientos del robot, lo que podría ser una solución más eficiente y efectiva en comparación con la generación de poses sucesivas. En esta investigación, se destaca una desventaja: al procurar mantener el equilibrio del robot a través del uso del ZMP, se observa una disminución en la movilidad del robot.

Xuechao Chen et al. mencionan que las acciones 'Heel-strike' y 'Toe-off' pueden mejorar la eficiencia energética del movimiento en los robots bípedos [7], para lograr que el robot realice estas dos etapas de la caminata utilizan el criterio del ZMP para mantener el equilibrio y un sistema de captura de movimiento para imitar los movimientos humanos.

Como otra alternativa surge la aplicación de aprendizaje por refuerzo. S. Phaniteja et.al. [8] atacan el problema generando trayectorias en el espacio articular del robot, las cuales deben asegurar la estabilidad del robot, esta estabilidad se calcula nuevamente

con el criterio del ZMP y un polígono de soporte generado por la geometría de los pies que se encuentran en contacto con el suelo o la superficie de apoyo (figura 1.3). Es importante destacar que el robot permanece en posición vertical, es decir, no camina, pero logra mantener la estabilidad.

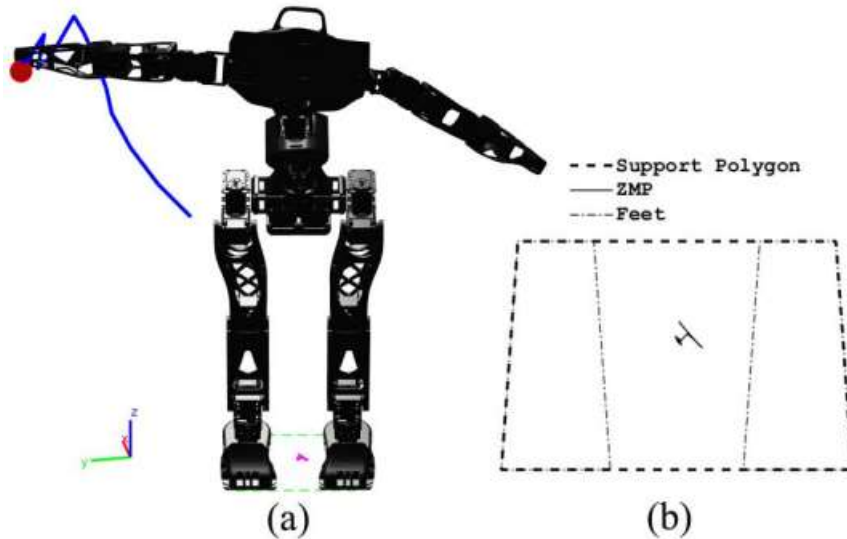


Figura 1.3: a) trayectoria generada por IA b) Trayectoria del ZMP [8]

Elmer R. Magsino [3] utiliza un robot bípedo en su investigación. Destaca que, en posiciones estáticas o caminatas lentas, se prioriza el cálculo del Centro de Masa del robot (COM). Sin embargo, en una marcha a velocidades mayores, factores como la aceleración, la fuerza centrífuga y términos de Coriolis afectan considerablemente el movimiento. En el modelado del robot, se emplea la convención de Denavit-Hartenberg (DH), considerando al robot como un conjunto de robots manipuladores que operan de manera desacoplada entre sí. En este caso el control solo se aplica al plano sagital del robot, como se puede observar en la figura 1.4 (El robot utilizado por R. Masigno), el robot no cuenta con articulaciones que controlen el plano frontal.

Donghyeon Seo et al. muestran como el aprendizaje por refuerzo puede hacer que un robot humanoide recupere el equilibrio después de una perturbación (figura 1.5) [9]. En esta investigación se utiliza aprendizaje por refuerzo para controlar la estabilidad del robot y la recuperación.

Boying Liu et al. mencionan que el aprendizaje por refuerzo ha demostrado buenos resultados en robots cuadrúpedos y hexápodos [10, 11]. En este caso las pruebas solamente se realizan en simulaciones (Véase figura 1.6), mostrando que puede llegar a

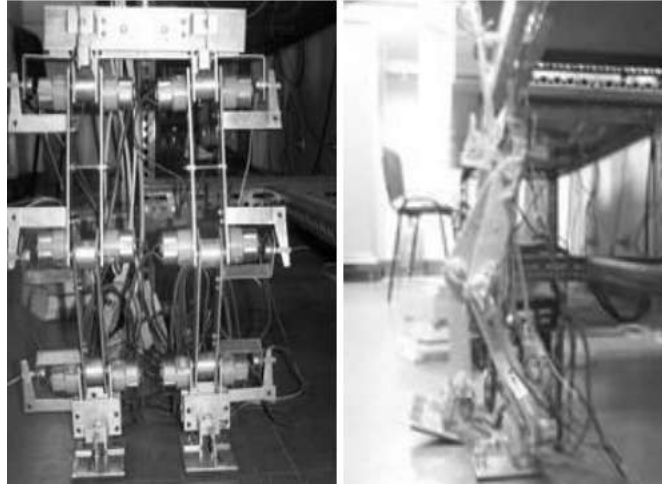


Figura 1.4: Robot bípedo en vista frontal y sagital [3]



Figura 1.5: Robot Darwin recuperándose de una perturbación [9]

funcionar [10], sin embargo en el entorno de simulación no se han tomado en cuenta variables importantes presentes en un ambiente real.

Szabo [12] menciona que es importante que un robot bípedo se pueda recuperar de las caídas(reforzando el trabajo de Boying Liu et al. [9]), debido a que esto es algo muy común, si el robot esta realizando una actividad y se cae, al poder levantarse puede continuar con su actividad. Programar el robot para que se levante es sumamente complicado, es por esto que Szabo propone utilizar algoritmos genéticos para controlar los movimientos del robot, lo cual dio muy buenos resultados experimentales.

Alemayoh et.al. [13] utilizan un sistema de captura de movimiento para obtener los movimientos humanos y replicarlos en el robot, a diferencia de los demás trabajos, se utiliza una red Bi-LSTM (*Bidirectional Long short-term memory*) que calcula la velocidad y la fase de la caminata del operador, la información calculada pasa a una red

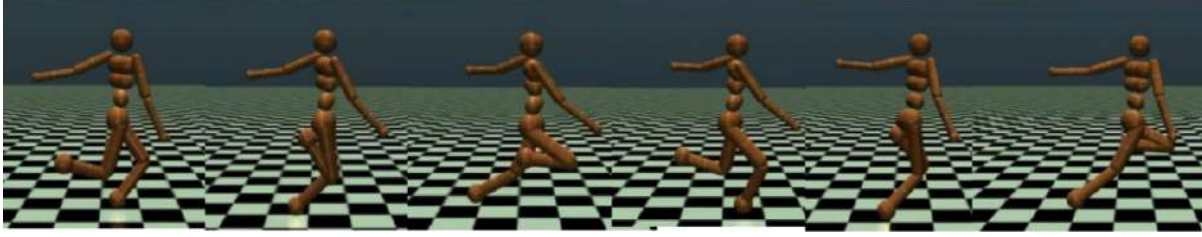


Figura 1.6: Simulación de la marcha de un humanoide generada por IA [10].

neuronal decodificadora que calcula los ángulos de cada articulación y estos ángulos son transferidos a un controlador de seguimiento, el esquema de este sistema se puede observar en la figura 1.7. Como se puede notar no sólo utilizaron la captura de movimiento, está técnica fue complementada con un par de redes neuronales, lo cual dio buenos resultados.

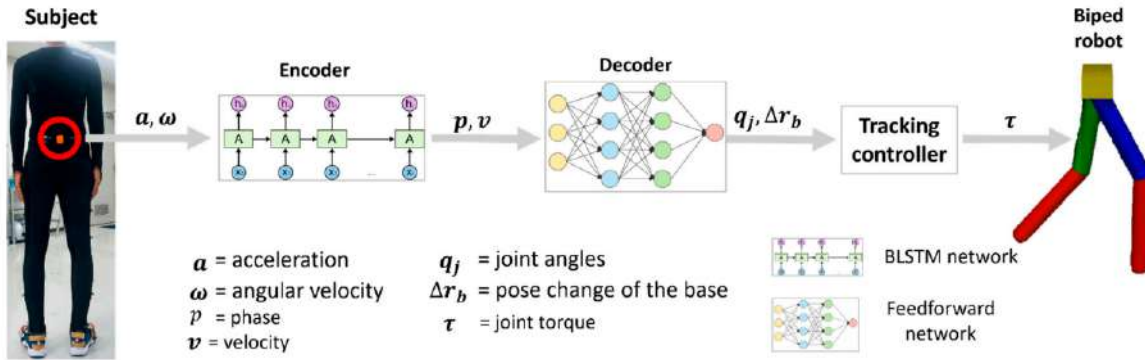


Figura 1.7: Esquema de control para robot bípedo. [13]

Nikita et.al [14] presentan un trabajo, endonde un robot cuadrúpedo aprende a caminar en menos de 20 minutos, utilizan Isaac Gym como simulador al momento de entrenar el robot, para el entrenamiento utilizaron Proximal Policy Optimization, utilizando entrenamiento en paralelo con multiples politicas (figura 1.8), entrenando hasta 16384 politicas simultaneamente, los autores expresan que no buscan un control robusto o una politica de alto rendimiendo, sino demostrar que con observaciones simples se puede transferir el aprendizaje de la simulación a la realidad en un periodo corto de tiempo.

Como se ha podido observar, el criterio de estabilidad mediante el ZMP toma un papel muy importante en el diseño y control de sistemas de locomoción bípeda. La utilización de ZMP como métrica de estabilidad permite evaluar y gestionar eficientemente la distribución del peso del robot sobre sus puntos de apoyo, contribuyendo así a su

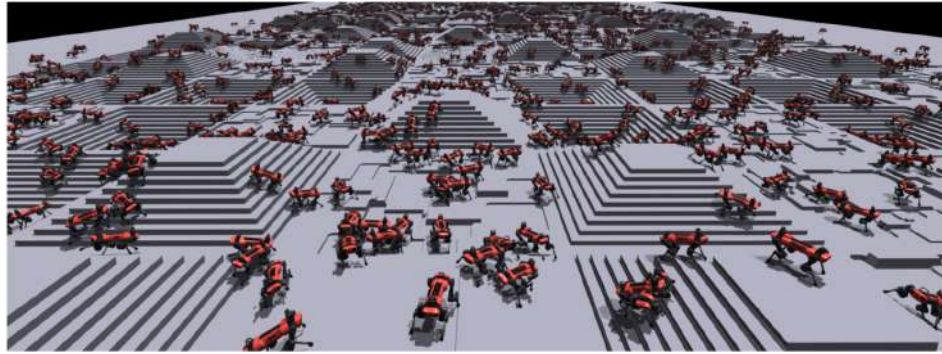


Figura 1.8: Miles de robots aprendiendo a caminar simultáneamente [14]

equilibrio.

En conjunto, la combinación de ZMP y técnicas de inteligencia artificial, como las redes neuronales convolucionales, algoritmos genéticos y aprendizaje por refuerzo, no solo ayudan a mantener la estabilidad del robot, sino que también simplifican y automatizan el proceso de desarrollo y control.

1.2 Planteamiento del problema

En la investigación sobre la estabilidad en la marcha de robots bípedos, se ha observado que el criterio del ZMP es ampliamente utilizado para analizar la estabilidad[2,4,7]. Aunque esta aproximación se considera eficaz, aún presenta deficiencias como la dependencia de un modelado preciso y la falta de exactitud de los modelos matemáticos, por lo que no se abordan todos los factores que afectan la estabilidad.

La complejidad de los robots bípedos, con su gran número de grados de libertad y la naturaleza no lineal de sus sistemas, complica la tarea de modelado y requiere técnicas de control más avanzadas. Se han utilizado modelos simplificados, como el del carro péndulo, para reducir la complejidad, pero presentan limitaciones en su efectividad debido a su simplificación.

Una alternativa es replicar los movimientos de un operador humano y aplicar el criterio del ZMP para mantener la estabilidad. Sin embargo, la diferencia en la distribución del centro de masa entre humanos y robots reduce la movilidad del último.

Para superar estas limitaciones, algunos investigadores han recurrido a la IA, empleando técnicas como el aprendizaje por refuerzo y algoritmos genéticos para entrenar robots y mejorar su estabilidad. Sin embargo, gran parte de esta investigación se ha centrado en condiciones estáticas, y aún hay oportunidades para explorar la aplicación efectiva de IA en escenarios dinámicos de caminata. A pesar de que la aplicación de IA ha demostrado ser efectiva en la locomoción de robots cuadrúpedos o hexápodos [11], la investigación en el ámbito de robots bípedos aún está en una fase emergente.

1.3 Justificación

La mayoría de los estudios previos sobre la estabilidad en la marcha de robots bípedos se han centrado predominantemente en el control mediante métodos analíticos [3, 5, 6, 15]. Otros trabajos que han explorado la perspectiva de la inteligencia artificial tienden a enfocarse en la estabilidad del robot en posición estática o se limitan al cálculo del Punto de Momento Nulo (ZMP) [8, 9]. Estos estudios destacan la importancia de una investigación que integre métodos analíticos con técnicas de inteligencia artificial, tanto para la marcha como para la estabilidad.

Máximo Roa et al [16] destacan la complejidad de la actividad de caminar, que demanda la coordinación de movimientos en diversas articulaciones. Además, subrayan la importancia de adquirir conocimiento del entorno a través de sensores. Al investigar la estabilidad de un robot bípedo, se abre la posibilidad de transferir este conocimiento al control de prótesis de miembros inferiores. Otra aplicación potencial de este estudio se encuentra en robots de servicio. Si el robot bípedo se diseña con proporciones humanas, podría ser capaz de desempeñar tareas similares a las realizadas por humanos. Es en estas posibles aplicaciones que reside la importancia de investigar la estabilidad en la marcha de un robot bípedo.

El desarrollo de técnicas para la locomoción bípeda, centradas en aspectos principales como el ZMP y el aprendizaje por refuerzo, podría beneficiar directamente a investigadores de este campo y, además, abrir nuevas perspectivas para la aplicación de estos avances.

1.4 Hipótesis

Mediante la combinación de las técnicas de aprendizaje por refuerzo y el criterio del punto de momento nulo, se conseguirá control sobre la estabilidad de la marcha de

un robot bípedo, resultando en un sistema capaz de alcanzar una referencia con bajo porcentaje de error.

1.5 Objetivos

1.5.1 Objetivo general

Diseñar un robot bípedo y aplicar las técnicas de estabilidad del criterio de Punto de Momento Nulo y aprendizaje por refuerzo en la marcha del robot, utilizando el modelo cinemático como apoyo, con el fin de conseguir estabilidad en la marcha y generar los movimientos de la caminata en un entorno simulado.

1.5.2 Objetivos específicos

- Investigar las bases del ZMP y RL.
- Diseño del modelo CAD del robot bípedo.
- Realizar el modelo cinemático del robot.
- Definir el algoritmo de RL a utilizar.
- Generar algoritmo para calcular el centro de masa y ZMP del robot.
- Definir las variables a utilizar en el entrenamiento por refuerzo.
- Entrenar el modelo de RL en un entorno simulado.
- Evaluación de los resultados.

1.6 Metas

- Realizar un informe que explique el concepto, los fundamentos y la implementación del criterio de ZMP y RL en la locomoción de robots bípedos.
- Diseñar el robot bípedo en un entorno CAD.

- Exportar y simular el modelo CAD a un entorno robótico.
- Realizar pruebas en lazo abierto de los eslabones del bípedo para corroborar su correcto modelado.
- Mostrar en una tabla las ventajas y desventajas de los algoritmos de aprendizaje reforzado y ponderar las necesidades, para elegir el algoritmo más adecuado.
- Simular en GAZEBO el robot bípedo para corroborar que los cálculos del centro de masa y ZMP sean correctos.
- Generar una tabla dónde se relacionen las acciones del robot y las variables del entorno a utilizar.
- Implementar el modelo de RL entrenado en ROS y simularlo en GAZEBO.
- Verificar la estabilidad y marcha en el robot bípedo, utilizando como referencia los ángulos deseados, así como la orientación y posición deseada.

1.7 Limitaciones de la tesis

1. **Grados de Libertad:** El robot contará con 6 grados de libertad (GDL) por pierna, teniendo un total de 12 GDL
2. **Simulación con ROS-GAZEBO:** La simulación se llevará a cabo mediante la plataforma GAZEBO impulsada por ROS.
3. **Posición de Inicio:** El robot comenzará la simulación desde una posición de triple flexión.

4. **Obtención de Caminata:** El objetivo principal será lograr la caminata del robot bípedo en el entorno de simulación.
5. **Enfoque Exclusivo en Tren Inferior:** El diseño y análisis se centrarán exclusivamente en el tren inferior del robot bípedo.
6. **Limitación a Simulación:** El proyecto se limitará a la simulación y no incluirá la construcción física del prototipo.
7. **Resultados Basados en Simulación:** Todas las conclusiones y resultados se derivarán de la simulación virtual.

1.8 Metodología

En la metodología empleada, se llevó a cabo la combinación de dos enfoques distintos: la "Metodología para Implementación Flexible en Robots Colaborativos" [17] y la "Metodología de Diseño Mecatrónico de Robots Industriales en Detalle" [18]. Dada la naturaleza específica del proyecto, se encontró necesario fusionar elementos de ambas metodologías.

La "Metodología de Diseño de Robots Industriales" aborda aspectos como el proceso de diseño, la selección de actuadores, estrategias de control y pruebas para la implementación. Por otro lado, se incorpora una sección de la "Metodología para Implementación Flexible en Robots Colaborativos" que se centra en el aprendizaje automático.

Luego de un análisis detenido de ambas metodologías y la identificación de aquellas partes que resultan más pertinentes para el presente trabajo de tesis, se propone una nueva metodología que integra ambos elementos. La estructura resultante se describe a continuación:

- **Identificación de Necesidad:** En esta fase, se busca comprender y definir la necesidad o problema que se abordará. Esto puede incluir la identificación de desafíos, oportunidades o mejoras.

- **Diseño Conceptual y Funcional:** Se realiza la conceptualización de la solución, definiendo sus características principales y funciones. Esto implica la creación de un diseño inicial que sirve como base para el desarrollo posterior.
- **Selección de Actuadores:** Se eligen los actuadores que serán responsables de llevar a cabo las acciones necesarias para la ejecución de la solución diseñada. Esta elección está vinculada a las funciones y requerimientos identificados.
- **Diseño CAM (Computer-Aided Manufacturing):** En esta etapa, se utiliza software de diseño asistido por computadora para crear modelos detallados de los componentes físicos de la solución.
- **Modelo Matemático:** Se desarrolla un modelo matemático que describe las relaciones y comportamientos esperados del sistema. Este modelo puede ser utilizado para análisis teóricos y simulaciones.
- **Verificación del Modelo Matemático:** Se valida y verifica el modelo matemático para asegurar que refleje el comportamiento esperado del sistema.
- **Selección de algoritmo para RL:** En esta fase, se realiza la elección del algoritmo de RL que mejor se adapte a las características específicas del problema y del entorno. Este proceso implica evaluar diversas opciones de algoritmos RL en función de factores como la complejidad del problema, la naturaleza del espacio de acciones y estados, la cantidad de datos disponibles y los requisitos de tiempo real.
- **Configuración del Simulador:** Se configura un simulador para reproducir virtualmente el comportamiento del sistema. Esto permite probar y ajustar la estrategia de control en un entorno controlado.
- **Entrenamiento:** Esta fase implica la recopilación y preparación de datos, así como el entrenamiento de modelos.
- **Simulación:** Se lleva a cabo una simulación integral del sistema, utilizando la estrategia de control ajustada y los parámetros finales, para evaluar el comportamiento global y validar la eficacia de la solución antes de la implementación física.
- **Conclusión.** En esta etapa final, se extraen conclusiones basadas en los resultados obtenidos durante todas las fases de la metodología. Se analizan los datos recopilados, los rendimientos simulados y/o experimentales, y se evalúa el cumplimiento de los objetivos establecidos.

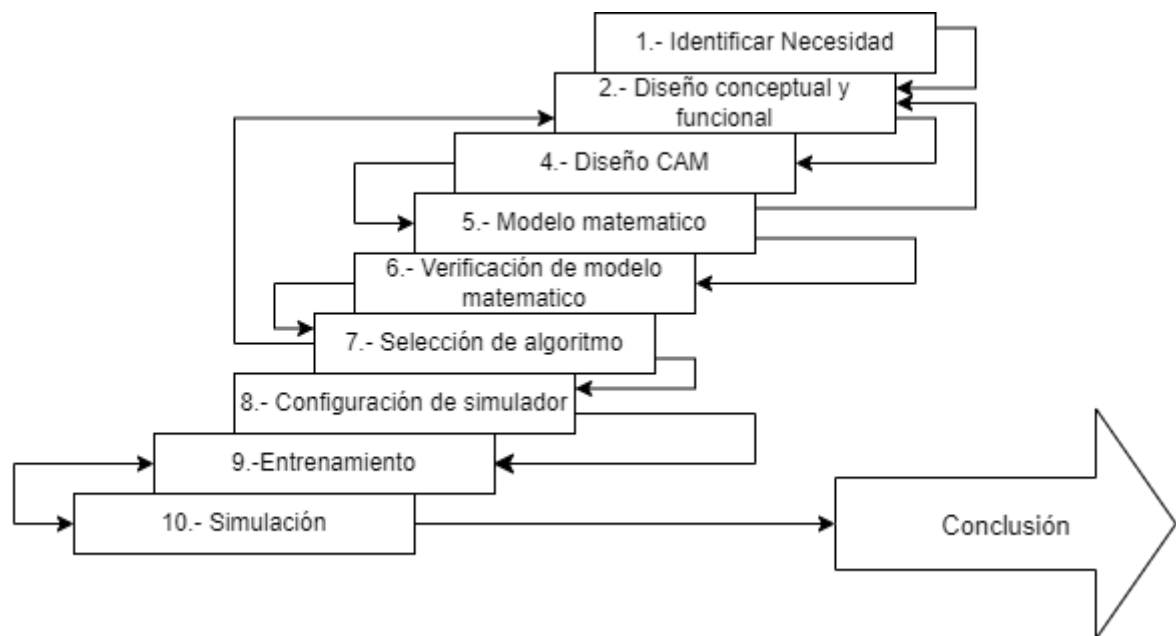


Figura 1.9: Diagrama de la metodología a utilizar.

Capítulo 2

Marco teórico

El presente marco teórico aborda los temas necesarios para poder comprender el desarrollo de esta tesis, enfocada en el control de la estabilidad durante la marcha de un robot bípedo.

En la actualidad, se han realizado múltiples desarrollos por parte de empresas privadas e investigadores de distintas instituciones. Este tema ha destacado debido a que los robots con locomoción bípeda pueden tener distintas aplicaciones en la robótica de servicio, así como en la robótica de rehabilitación y en el desarrollo y mejora de prótesis. Sin embargo, las empresas privadas no publican las técnicas ni metodologías utilizadas en el desarrollo de sus investigaciones. Por su parte, los investigadores de universidades e instituciones afines enfrentan limitaciones presupuestarias que ralentizan el desarrollo. Actualmente, el aprendizaje reforzado es una de las técnicas más utilizadas por la industria para el control de robots bípedos, aunque esto suponga un costo computacional muy elevado. Por lo tanto, es muy importante establecer un marco teórico que sirva como base para comprender el control de estabilidad en la marcha de un robot bípedo.

El objetivo principal de este marco teórico es proporcionar el conocimiento básico para comprender de manera adecuada los conceptos y teorías que sustentan esta investigación. Se pretende revisar la literatura existente sobre los robots manipuladores, conceptos para la estabilidad en la marcha, inteligencia artificial, entre otros.

Esto permitirá no solo clarificar el panorama teórico sobre el control de estabilidad en robots bípedos, sino también identificar áreas de oportunidad y posibles líneas de investigación futura, contribuyendo así al avance del conocimiento en este campo.

2.1 Modelado

Los robots son sistemas mecánicos complejos. El objetivo principal de los robots es seguir un comportamiento deseado por el usuario. Para que el robot pueda cumplir con

estas características, es necesario conocer el modelo matemático para poder generar las acciones de control [19]. El estudio de estos sistemas mecánicos incluye la estática, la cinemática y la dinámica [20].

2.1.1 Cinemática directa

La cinemática es una rama de la mecánica que describe el movimiento de las partículas [21]. En relación con la robótica de manipuladores, la cinemática directa describe la posición y orientación del efector final a lo largo de los eslabones de un robot manipulador. Esta relación es el punto de interés en la cinemática de los robots [20].

Encontrar las ecuaciones que describan la relación entre las variables articulares, y el efector final es un problema que puede solucionarse mediante la cinemática directa, la cual describe la posición y orientación del efector final en términos de las variables articulares [22].

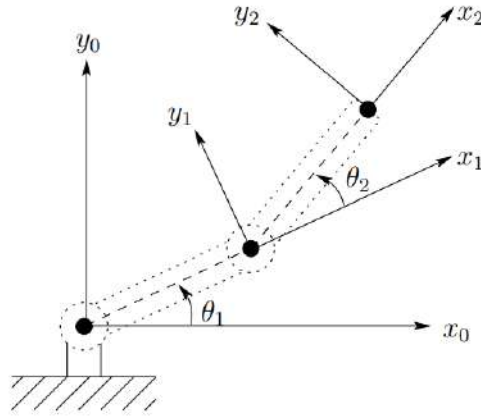


Figura 2.1: Marcos coordenados de un robot planar de 2 GDL.

2.1.2 Cinemática inversa

Para que el efector final de un robot manipulador llegue a una posición deseada, es necesaria la cinemática inversa. Con esto, se busca encontrar el valor de las variables articulares en función de la posición y orientación del efector final [22].

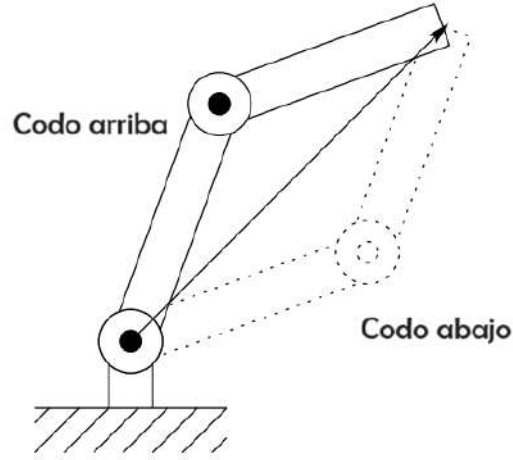


Figura 2.2: Múltiples soluciones a un problema de cinemática inversa [22].

2.1.3 Cinemática diferencial

Una forma adicional para controlar la posición y orientación del efector final del robot es la cinemática diferencial [22]. Este enfoque relaciona las velocidades articulares con las velocidades lineales y angulares del efector final mediante la derivación temporal de la cinemática directa. La matriz que concentra estas relaciones se denomina “jacobiano analítico” y permite calcular, a partir de las velocidades articulares las velocidades lineales y angulares del efector final. Esta relación puede ser calculada con la ayuda del jacobiano analítico como se muestra en 2.1

$$\dot{x}_e = \begin{bmatrix} \dot{p}_e \\ \dot{\phi}_e \end{bmatrix} = \begin{bmatrix} J_p(q) \\ J_\phi(q) \end{bmatrix} \dot{q} = J_A(q) \dot{q} \quad (2.1)$$

donde P_e representa la posición del efector final, $\dot{P}_e$ las velocidades lineales del efector final, ϕ_e es la representación mínima de la orientación del efector final, $\dot{\phi}_e$ la velocidad angular del efector final, q las posiciones articulares, $\dot{q}$ las velocidades articulares, $J_p(q)$ el jacobiano para las velocidades lineales, $J_\phi(q)$ el jacobiano para las velocidades angulares y $\dot{x}_e$ las velocidades de traslación y rotación del efector final.

2.1.4 Convención de Denavit-Hartenberg

Jacques Denavit y Richard Scheunemann Hartenberg en 1955 introdujeron un método para describir completamente mecanismos cinemáticos utilizando una notación basada en matrices. Este método, conocido como la convención de Denavit-Hartenberg(DH), ha sido ampliamente utilizada para describir la cinemática de robots manipuladores debido a su capacidad para simplificar el modelado y análisis de estos sistemas. La convención de DH consta de 9 pasos [22]:

1. Localizar y etiquetar los ejes articulares($z_0, z_1, \dots, z_{n-1}$), las rotaciones siempre deben hacerse en el eje z del marco coordenado del eje articular.
2. Establecer un marco coordenado de base (x_0, y_0, z_0) respetando la regla de la mano derecha
Para los pasos 3 al 5 se repite para $i = 1, \dots, n - 1$
3. Localizar el origen O_i donde la normal común a z_i y z_{i-1} intersectan z_i
4. Localizar el origen O_i donde la normal común a z_i y z_{i-1} intersecta a z_i . Si z_i interseca a z_{i-1} , localizar O_i en esta intersección. Si z_i y z_{i-1} son paralelos, localizar O_i en cualquier posición conveniente a lo largo de z_i .
5. Establecer x_i a lo largo de la normal común entre z_{i-1} y z_i a través de O_i , o en la dirección normal al plano $z_{i-1} - z_i$ si z_{i-1} y z_i se intersectan.
6. Establecer y_i para completar un marco respetando la regla de la mano derecha.
7. Establecer el marco del efector final en x_n, y_n, z_n . Suponiendo que la n -ésima articulación es de tipo revoluta, establece $z_n = a$ en la dirección de z_{n-1} . Establecer el origen convenientemente a lo largo de z_n , preferiblemente en el centro del gripper o en la punta de cualquier herramienta que pueda estar llevando el manipulador. Establecer $y_n = s$ en la dirección del cierre del gripper y establece $x_n = s \times a$. Si la herramienta no es un simple gripper, establece x_n e y_n convenientemente para formar un marco de referencia respetando la regla de la mano derecha.
8. Crea una tabla de parámetros de eslabones $a_i, d_i, \alpha_i, \theta_i$:

Eslabón i	a_i	d_i	α_i	θ_i
-------------	-------	-------	------------	------------

Donde:

- a_i : distancia a lo largo de x_i desde O_i hasta la intersección de los ejes x_i y z_{i-1} .
 - d_i : distancia a lo largo de z_{i-1} desde O_{i-1} hasta la intersección de los ejes x_i y z_{i-1} . d_i es variable si la articulación i es prismática. - α_i : el ángulo entre z_{i-1} y z_i medido alrededor de x_i . - θ_i : el ángulo entre x_{i-1} y x_i medido alrededor de z_{i-1} . θ_i es variable si la articulación i es revoluta.
9. Formar las matrices de transformación homogéneas A_i sustituyendo los parámetros anteriores.
 10. Formar $T_n^0 = A_1 \cdot \dots \cdot A_n$. Esto luego proporciona la posición y orientación del marco de la herramienta expresada en coordenadas de base.

2.1.5 Desacople cinemático

La cinemática inversa de un robot manipulador puede ser compleja, y esta complejidad aumenta a medida que se incrementan los grados de libertad del robot. Para robots con seis grados de libertad, el problema de la cinemática inversa se puede simplificar dividiendo el robot en dos partes: cinemática de posición inversa y cinemática de orientación inversa [22]. Para que esta propiedad se cumpla, el robot debe tener una característica específica: las últimas tres articulaciones deben intersectarse en un punto. De esta manera, el robot se puede visualizar como dos robots distintos: uno encargado de la posición y otro encargado de la orientación del efector final. En la figura 2.3 se muestra un ejemplo apto para aplicar desacople cinemático, el vector d_0^c apunta al punto final para la cinemática de posición inversa y el vector d_0^6 apunta a el efector final tomando en cuenta tanto la cinemática de posición inversa como la cinemática de orientación inversa.

2.2 Movimientos de los miembros inferiores

Los miembros inferiores se encargan de sostener el peso del cuerpo, así como mantenerlo en equilibrio en bipedación o durante la locomoción (caminata) [23]. Los miembros inferiores son indispensables para la locomoción bípeda, por lo que también es importante conocer los movimientos que estas extremidades pueden realizar. Estos movimientos se pueden clasificar en 3 tipos principales [24]:

- Flexión/extensión.

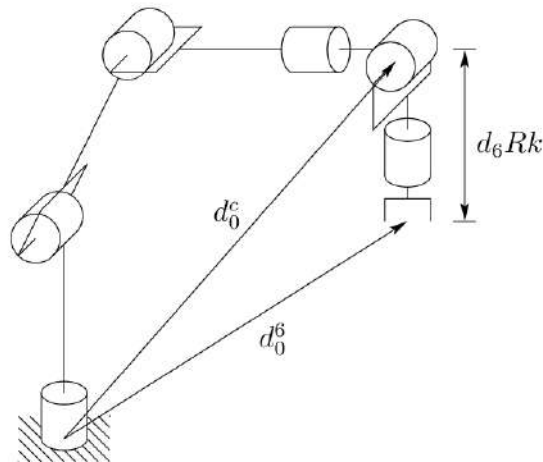


Figura 2.3: Ejemplo de desacople cinemático [22].

- Abducción/Aducción.
- Rotación interna/externa

En la figura 2.4 se muestran los movimientos principales de los miembros inferiores, con los nombres específicos del movimiento con relación a la articulación.

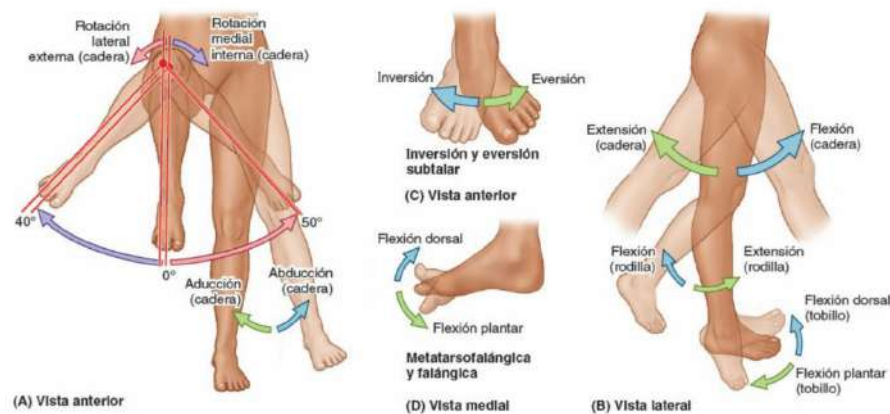


Figura 2.4: Movimientos del miembro inferior [23].

2.3 Ciclo de la marcha

"El ciclo de la marcha se define como el intervalo de tiempo entre dos pasos sucesivos" [24]. Para describir el ciclo de la marcha generalmente se descompone en componentes más simples, estos componentes cambian dependiendo de la fuente consulta, para investigaciones sobre locomoción bípeda en robots se suelen dividir en la fase de balanceo y la fase de apoyo [25].

2.3.1 Fase de apoyo

La fase de apoyo se puede dividir en cinco componentes [25] (mostrados en la figura 2.5):

- Contacto del talón: Es el momento en el que el talón de la pierna de referencia toca el suelo.
- Apoyo plantar: Se refiere al contacto de la parte anterior del pie con el suelo.
- Apoyo medio: Ocurre cuando el trocánter mayor (una prominencia ósea en la parte superior del fémur) está alineado verticalmente con el centro del pie, visto desde un plano sagital.
- Elevación del talón: Ocurre cuando el talón se eleva del suelo.
- Despegue del pie: Es cuando los dedos se elevan del suelo.

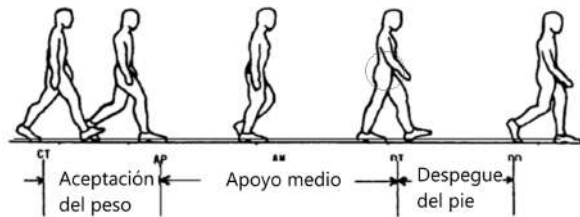


Figura 2.5: Componentes de la fase de apoyo de la marcha [25].

2.3.2 Fase de balanceo

La fase de balanceo se divide en tres componentes [25]:

- **Aceleración:** El pie acelera después de despegar los dedos del suelo, justo después del apoyo plantar.
- **Balanceo medio:** La pierna en movimiento se desplaza hacia adelante, posicionándose delante de la pierna de apoyo.
- **Deceleración:** La pierna en movimiento disminuye su velocidad para pasar a la siguiente fase de apoyo.



Figura 2.6: Componentes de la fase de balanceo [25].

2.4 Centro de presión

Consideremos un bípedo cuando un solo pie está en contacto con el suelo. El campo de fuerzas de presión (normal a la suela) equivale a una única fuerza resultante, ejercida en el punto donde el momento resultante es nulo, este punto se denomina centro de presión (CoP, por sus siglas en inglés). En cualquier punto P de la suela, la fuerza de contacto elemental $\mathbf{c}(P)$ puede dividirse en dos componentes, un componente normal $p(P)\mathbf{n}$, que es la fuerza de presión local, y un componente tangencial $\mathbf{f}(P)$ que es la fuerza de fricción local. Siendo $\mathbf{n}$ el vector unitario normal a la suela (S), orientado hacia el pie, hacia el exterior de la superficie de apoyo. En consecuencia, la resultante de estas fuerzas de contacto elementales aparece como la suma, de las fuerzas locales de presión.

$$\begin{aligned}
 P \in (S), \quad \mathbf{c}(P) &= p(P)\mathbf{n} + \mathbf{f}(P) \\
 p(P) &\geq 0, \quad \mathbf{f}(P) \cdot \mathbf{n} = 0
 \end{aligned}
 \tag{2.2}$$

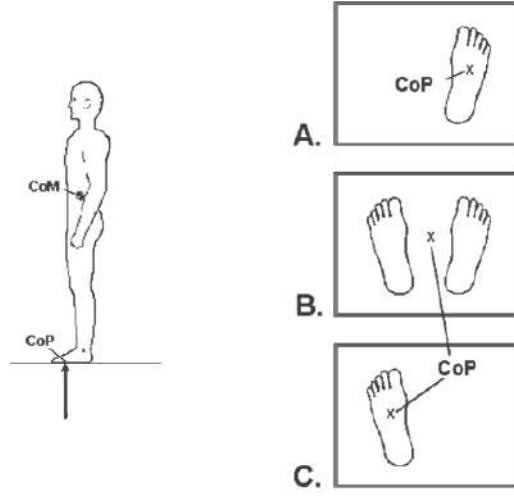


Figura 2.7: Diagrama que ilustra el CoP en función del contacto de la suela del pie con el suelo [26].

$$\mathbf{R}^p = \left(\int_S p(P) dS \right) \mathbf{n} \equiv R^p \mathbf{n}, \quad R^p > 0 \quad (2.3)$$

cuyo momento alrededor de cualquier punto Q es:

$$\mathbf{M}_Q^p = \left(\int_S p(P) \mathbf{Q} \mathbf{P} dS \right) \times \mathbf{n} \quad (2.4)$$

y por otro lado, las fuerzas de fricción locales

$$\mathbf{R}^f = \int_S \mathbf{f}(P) dS, \quad \mathbf{R}^f \cdot \mathbf{n} = 0 \quad (2.5)$$

cuyo momento alrededor de Q es

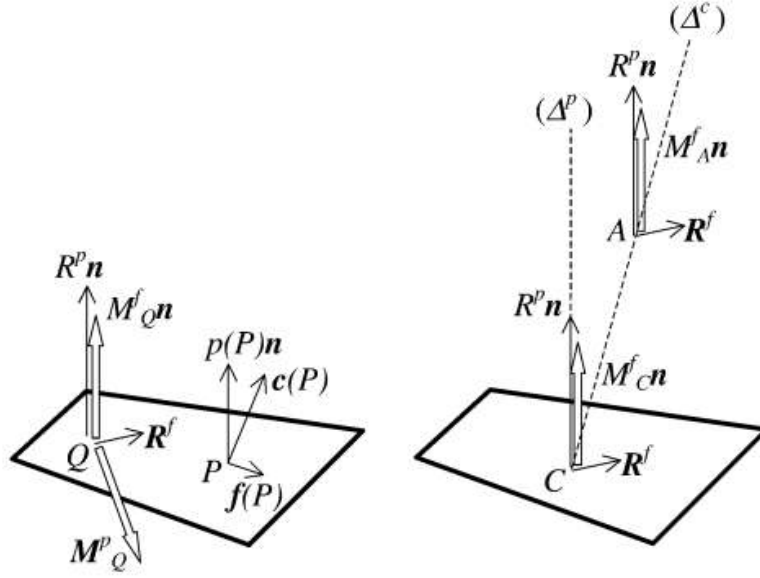


Figura 2.8: Fuerzas de contacto y momentos actuando en la suela [27].

$$M_Q^f = \int_S \mathbf{QP} \times \mathbf{f}(P) dS. \quad (2.6)$$

Con los superíndices p y f denotando los componentes de presión y fricción respectivamente. La ecuación 2.4 muestra que el momento de las fuerzas de presión es siempre perpendicular al vector normal $\mathbf{n}$, cualquiera que sea el punto Q . En consecuencia, la resultante de las fuerzas de presión se dirige a lo largo de $\mathbf{n}$, luego existe un eje Δ^p , donde el momento $\mathbf{M}_Q^p$ desaparece en cada punto de este eje. Se puede observar que Δ^p es el eje central de las fuerzas de presión. El CoP que se define como el punto de la suela donde es la intersección entre el eje y el plano de la suela.

2.5 Punto de momento nulo

El Punto de Momento Nulo (ZMP, por sus siglas en inglés) es un punto en el suelo donde el momento resultante debido a las fuerzas de gravedad e inercia es igual a cero, cabe mencionar que el término ZMP no es una expresión perfectamente exacta ya que la componente normal del momento generado por las fuerzas de inercia que actúan sobre el bípedo no es necesariamente cero. La resultante de la gravedad más las fuerzas de inercia pueden ser expresadas como:

$$\mathbf{R}^{gi} = m\mathbf{g} - m\mathbf{a}_G \quad (2.7)$$

y el momento al rededor de cualquier punto Q como:

$$\mathbf{M}_Q^{gi} = \mathbf{QG} \times m\mathbf{g} - \mathbf{QG} \times m\mathbf{a}_G - \dot{\mathbf{H}}_G \quad (2.8)$$

Donde m es la masa total, $\mathbf{g}$ es la aceleración de la gravedad, G es el centro de masa $\mathbf{a}_G$ es la aceleración de G y $\dot{\mathbf{H}}_G$ la tasa del momento angular en G .

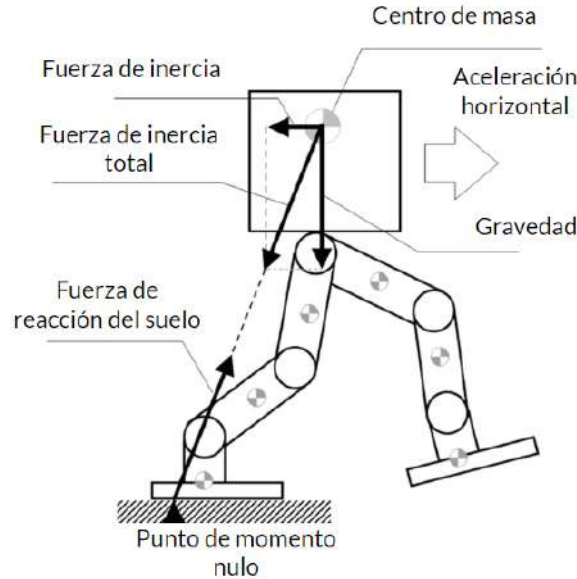


Figura 2.9: Diagrama básico que ilustra el ZMP en función de las fuerzas que actúan sobre el centro de masa del robot [28].

Existe un eje Δ^{gi} , en el que el momento es paralelo al vector normal $\mathbf{n}$, sobre cada punto B del eje. El ZMP, cuyo momento se dirige por definición a lo largo del vector

normal $\mathbf{n}$, pertenece necesariamente a este eje. Así pues, el ZMP puede definirse como la intersección entre el eje y la superficie del suelo, de forma que:

$$\mathbf{M}_D^{gi} = \mathbf{DG} \times m\mathbf{g} - \mathbf{DG} \times m\mathbf{a}_G - \dot{\mathbf{H}}_G, \quad \mathbf{M}_D^{gi} \times \mathbf{n} = \mathbf{0} \quad (2.9)$$

Hay que señalar que este eje concreto no pasa por el centro de masa global como se muestra en la figura 2.10. Estas ecuaciones expresan el hecho de que el bípodo está equilibrado dinámicamente si las fuerzas de contacto y las fuerzas de gravedad-inercia son estrictamente opuestas.

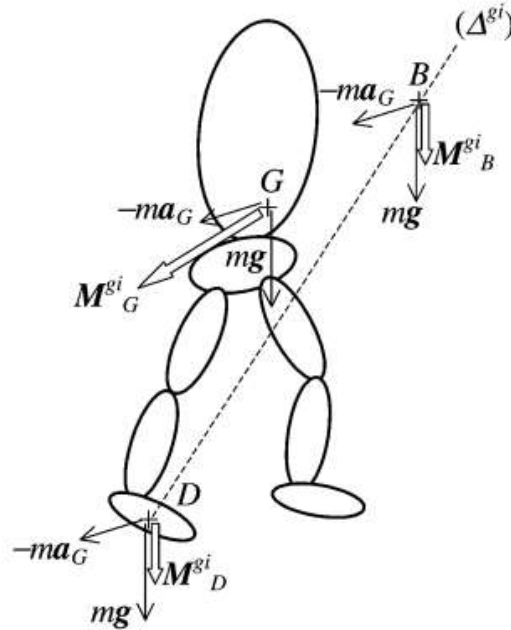


Figura 2.10: Gravedad-inercia, fuerzas-momentos en el centro de masa G, y el mismo transportado al ZMP D [27].

2.6 Inteligencia artificial

La inteligencia artificial (IA) se define como "el campo de estudio que se centra en el desarrollo de sistemas informáticos capaces de realizar tareas que requieren inteligencia

humana" [29]. A partir de esta idea, han surgido diversas técnicas para permitir que las computadoras realicen tareas de forma automática. Entre estas técnicas, destaca el aprendizaje máquina o *machine learning*.

2.6.1 Perceptrón.

Se puede definir al perceptrón como: “un dispositivo que puede responder “sí” o “no” a un problema en el dominio bajo consideración" [30]. El perceptrón es la unidad básica utilizada en las redes neuronales. Este cuenta con n entradas, las cuales se multiplican por n pesos. La suma de los resultados de estas multiplicaciones se ingresa a una función de activación, que produce como salida un nivel alto o bajo.

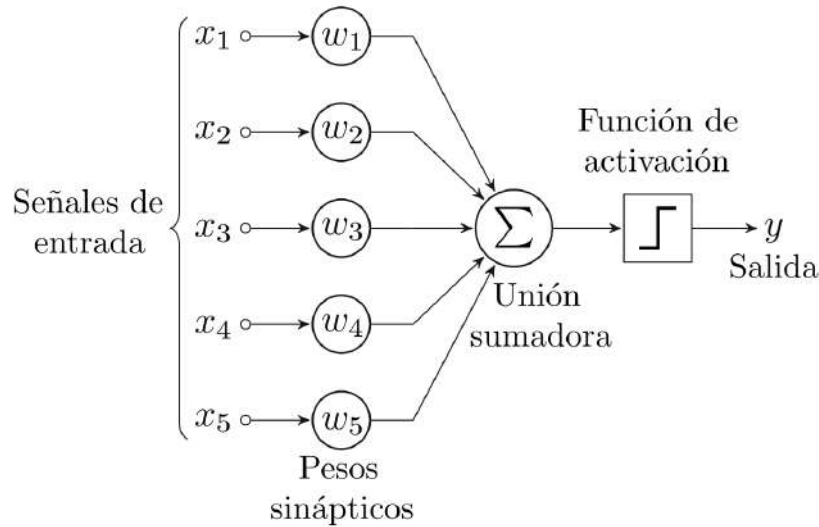


Figura 2.11: Modelo gráfico de un perceptrón [31].

2.6.2 Perceptrón multicapa

Las redes neuronales profundas de tipo *feedforward*, también conocidas como redes neuronales hacia adelante o perceptrón multicapa (MLP, por sus siglas en inglés), constituyen uno de los modelos fundamentales del aprendizaje profundo.

El objetivo principal de una red *feedforward* es aproximar una función objetivo f^* . Por ejemplo, en tareas de clasificación, dicha función asigna una entrada x a una salida o categoría y , es decir, $y = f^*(x)$. La red neuronal define una función aproximada

$y = f(x; \theta)$, donde θ representa el conjunto de parámetros del modelo, los cuales son ajustados durante el proceso de entrenamiento con el fin de lograr la mejor aproximación posible de la función objetivo [32].

En resumen, el perceptrón multicapa es un conjunto de neuronas artificiales organizadas en capas (una capa de entrada, una o varias capas ocultas y una capa de salida), donde cada neurona está conectada con las de la siguiente capa mediante pesos ajustables. Su función principal es aproximar funciones complejas a partir de datos de entrada, aprendiendo mediante el ajuste iterativo de estos pesos durante el entrenamiento.

2.6.3 Aprendizaje máquina

El concepto de *machine Learning* (ML) fue acuñado en 1959 por Arthur Samuel (Bell Labs, IBM, Stanford). "Un sistema de IA necesita tener la capacidad de aprender su propio conocimiento a partir de los datos brutos. Esta capacidad de aprendizaje se conoce como ML" [29]. Una definición más técnica es la siguiente: "El *machine learning* es el campo de estudio que da a los ordenadores la capacidad de aprender sin ser programados de manera explícita" [33].

Según la UC Barkeley (Universidad de California en Barkeley) el ML consta de tres componentes importantes [34]:

- Un proceso de decisión: Una receta de cálculos u otros pasos que toma los datos y "adivina" qué tipo de patrón busca encontrar el algoritmo.
- Una función de error: Un método para medir la calidad de la suposición comparándola con ejemplos conocidos (cuando están disponibles). ¿Ha acertado el proceso de decisión? Si no es así, ¿cómo se cuantifica "lo malo" que ha sido el fallo?
- Un proceso de actualización u optimización: Un método en el que el algoritmo examina el fallo y luego actualiza cómo el proceso de decisión llega a la decisión final, por lo que la próxima vez el fallo no será tan grande.

Tomando como base estos componentes se han generado múltiples técnicas y algoritmos de aprendizaje máquina. De acuerdo con la UC Barkeley el ML se puede dividir en tres categorías principales [33]:

- Aprendizaje supervisado.
- Aprendizaje no supervisado.
- Aprendizaje reforzado.

2.6.4 Aprendizaje supervisado

Según Isha Salian "En el aprendizaje supervisado se tiene un conjunto completo de datos totalmente etiquetados mientras se entrena un algoritmo".

"Totalmente etiquetados" significa que cada ejemplo del conjunto de datos de entrenamiento está etiquetado con la respuesta que el algoritmo debería dar por sí mismo. Así, un conjunto de datos etiquetados de imágenes de flores indicaría al modelo qué fotos son de rosas, margaritas y narcisos. Cuando se le muestra una nueva imagen, el modelo la compara con los ejemplos de entrenamiento para predecir la etiqueta correcta [35].

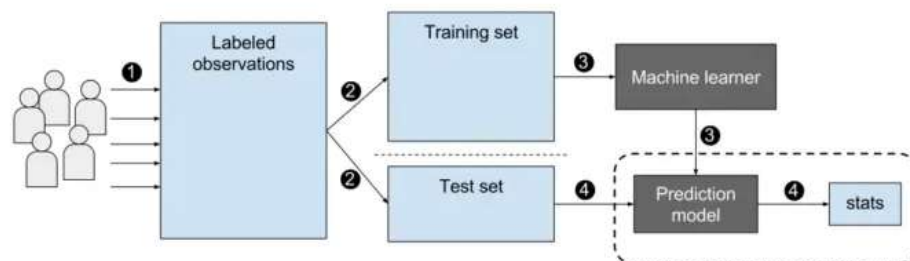


Figura 2.12: Diagrama de funcionamiento básico del aprendizaje supervisado [35].

2.6.5 Aprendizaje no supervisado

Tomando nuevamente el trabajo de Isha Salian "En el aprendizaje no supervisado, un modelo de aprendizaje profundo recibe un conjunto de datos sin instrucciones explícitas sobre qué hacer con él. El conjunto de datos de entrenamiento es una colección de ejemplos sin un resultado específico deseado o una respuesta correcta. A continuación, la red neuronal intenta encontrar automáticamente una estructura en los datos extrayendo características útiles".

Dentro de esta técnica se encuentra el *clustering* o agrupamiento de datos. "El análisis clúster o análisis de conglomerados es una técnica estadística multivariante que busca agrupar individuos con el fin de lograr la máxima homogeneidad en cada grupo (alta similitud intra-clase) y la mayor diferencia entre los grupos (baja similitud inter-clase). En este caso los conglomerados son desconocidos y el método busca definir dichos grupos de forma óptima a partir de los mismos datos. En este punto, la tarea más importante es definir etiquetas para esos grupos encontrados" [36].

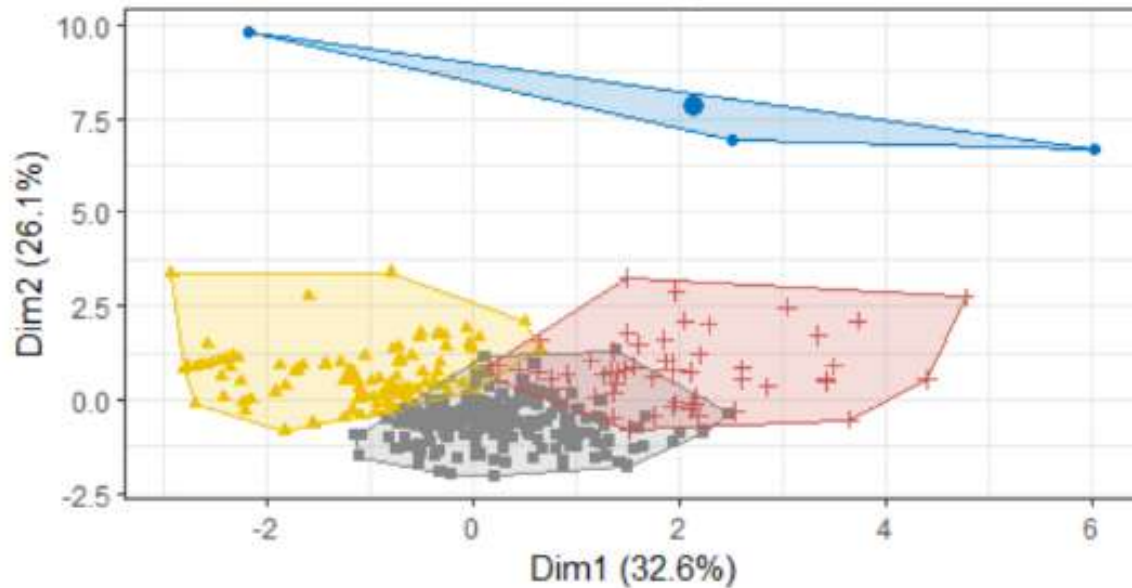


Figura 2.13: Diagrama que ilustra los resultados obtenidos mediante algoritmo k-means (clustering) [36].

2.6.6 Aprendizaje por refuerzo

El RL es un subconjunto del ML que permite a los ordenadores aprender interactuando con el entorno. RL separa el mundo real en dos componentes: un entorno y un agente [29].

Un agente es una entidad que puede interactuar con el entorno y tomar decisiones. Recibe información del entorno a través de sensores y realiza acciones en función de esa información para lograr un objetivo específico. El agente busca maximizar una recompensa o minimizar una penalización a lo largo del tiempo [29].

El ambiente es la entidad con la que el agente interactúa. El ambiente recibe las acciones del agente y en respuesta produce un nuevo estado y una recompensa, los cuales son observados por el agente. El ambiente puede ser cualquier sistema con el que el agente interactúa, como un videojuego, un robot físico o cualquier otro entorno controlado por computadora [29]. En un paso de tiempo arbitrario, el agente observa el estado actual del entorno (S_t) y la recompensa correspondiente (R_t). Luego, toma una decisión sobre la siguiente acción (A_t) basándose en la información del estado y la recompensa. La acción afecta al entorno, generando un nuevo estado ($S_t + 1$) y recompensa ($R_t + 1$) (figura 2.14). La observación del estado desde la perspectiva del agente puede ser parcial o completa, lo que determina si el entorno es parcial o completamente observable.

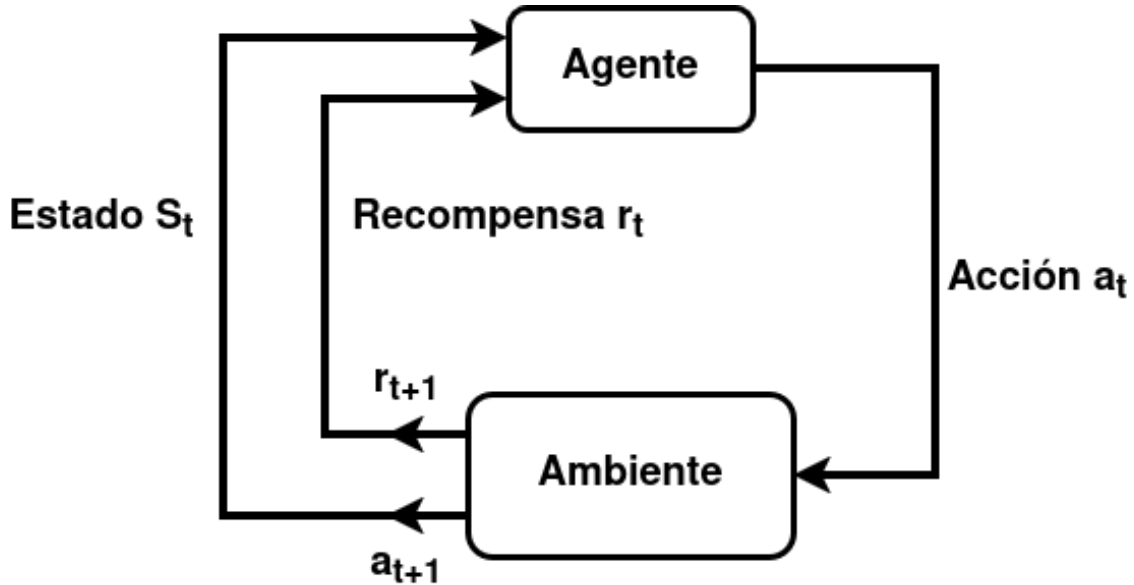


Figura 2.14: Diagrama que muestra la interacción entre el agente y el ambiente, adaptado de [29]

Para proporcionar información del entorno al agente, una función genera una recompensa R_t en función del estado del entorno y la envía al agente, en algunos casos, la función de recompensa depende únicamente del estado actual, es decir, $R_t = R(S_t)$, sin embargo, a veces la función de recompensa puede depender no sólo del estado y la acción actuales, sino también de los estados y las acciones en otros pasos temporales, entonces, la recompensa dependerá no sólo de un par estado-acción sino también de la secuencia de pares estado-acción.

Aprendizaje por refuerzo profundo.

El aprendizaje por refuerzo profundo (DRL) se obtiene cuando se utilizan redes neuronales profundas para aproximar la función de valor, la política o el modelo (función de transición de estado o función de recompensa). Usualmente, se utiliza descenso de gradiente estocástico para actualizar los parámetros [37].

2.6.7 Proximal Policy Optimization

Proximal Policy Optimization (PPO) es un algoritmo de DRL que optimiza la política del agente de manera estable y eficiente. Es un algoritmo sobre la política actual (on-policy), lo que significa que aprende usando únicamente los datos generados por la propia política que se está entrenando. Durante el entrenamiento, los datos de experiencia se dividen en mini-batches (mini-lotes), pequeños subconjuntos que permiten actualizar la red neuronal de manera más estable y eficiente [38].

El algoritmo PPO puede dividirse en dos etapas principales: La recolección de experiencia (rollout) y la fase de actualización de los parámetros de la política y de la función de valor.

En la etapa de rollout, el agente interactúa con el entorno durante un horizonte temporal finito. En cada instante t , el agente observa el estado S_t y, a partir de la política estocástica del actor $\pi_\theta(a_t | S_t)$, selecciona una acción a_t . Para espacios de acción continuos, esta política suele modelarse como una distribución normal parametrizada por una media $\mu_\theta(S_t)$ y una desviación estándar $\sigma_\theta(S_t)$, de la cual se muestrea la acción. Tras ejecutar a_t , el entorno devuelve la recompensa inmediata r_t . De forma paralela, el crítico estima el valor del estado mediante $V_\theta(S_t)$, que aproxima el retorno esperado a partir de dicho estado.

Como resultado del rollout, se construye una trayectoria de transiciones para el entrenamiento posterior. Usualmente, se almacenan tuplas del tipo $(S_t, a_t, r_t, V_\theta(S_t), \log \pi_\theta(a_t | S_t))$. Con esta información se calculan las ventajas y los retornos objetivo, que posteriormente se utilizan para actualizar de manera estable la política en PPO. El flujo de información durante esta etapa se ilustra en la figura 2.15.

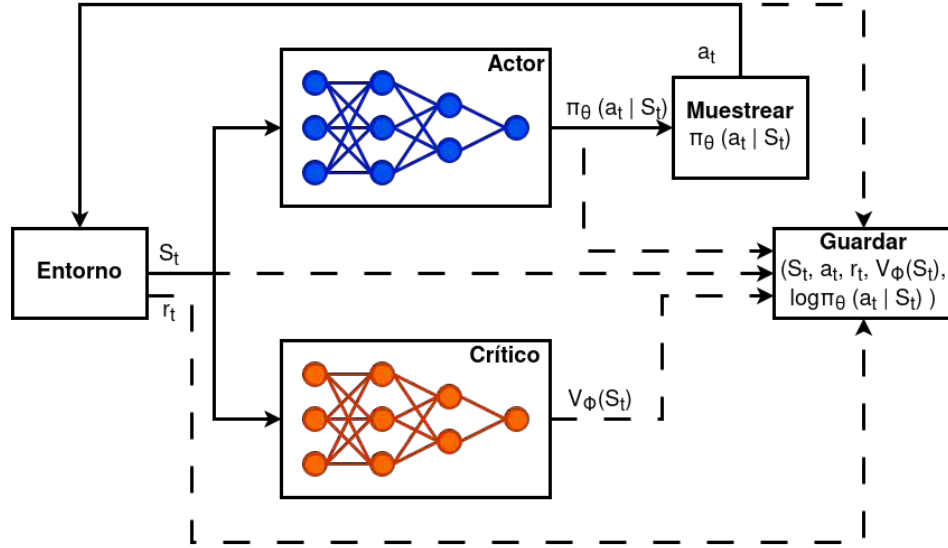


Figura 2.15: Etapa de rollout del algoritmo PPO para acciones continuas.

En la etapa de actualización de parámetros se utilizan las trayectorias de transición almacenadas durante el rollout. La ventaja $\hat{A}_t$ representa cuánto mejor resulta una acción respecto al valor esperado del estado $V_\phi(S_t)$. Esta ventaja se calcula mediante el Estimador Generalizado de Ventaja (GAE, por sus siglas en inglés), como se muestra en 2.10.

$$\hat{A}_t = \sum_{l=0}^{T-t+1} (\gamma\lambda)^l \delta_{t+l} \quad (2.10)$$

donde $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

La función de pérdida del crítico, también denominada pérdida de la función de valor, se muestra en 2.11.

$$L_t^{VF}(\theta) = (V_\theta(S_t) - V_t^{\text{objetivo}})^2 \quad (2.11)$$

La función de pérdida del actor se muestra en 2.12.

$$L^{\text{actor}} = L_t^{\text{CLIP}}(\theta) + c_2 S[\pi_\theta](S_t) \quad (2.12)$$

$$\text{dónde, } L_t^{\text{CLIP}} = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

$$r_t(\theta) = e^{\log \pi_\theta(a_t|S_t) - \log \pi_{\theta_{\text{anterior}}}(a_t|S_t)} \quad (2.13)$$

Por lo tanto, el procedimiento de entrenamiento se resume en el algoritmo 1.

Algorithm 1 PPO estilo Actor-Critic

```

1: for iteración  $k = 1, 2, \dots$  do
2:   for actor  $i = 1, 2, \dots, N$  do
3:     Ejecutar la política  $\pi_{\theta_{\text{old}}}$  en el entorno durante  $T$  pasos
4:     Calcular las ventajas  $\hat{A}_1, \dots, \hat{A}_T$ 
5:   end for
6:   Optimizar el objetivo sustituto  $L$  respecto a  $\theta$ 
7:   usando  $K$  épocas y minibatches de tamaño  $M \leq NT$ 
8:   Actualizar política antigua:
9:    $\theta_{\text{old}} \leftarrow \theta$ 
10: end for

```

2.7 ROS

ROS (Robot Operating System por sus siglas en inglés) "es un conjunto de bibliotecas de software y herramientas que ayudan a desarrollar aplicaciones de robots. (...) Y es de fuente abierta." [39]

Hay que aclarar que ROS no es un sistema operativo. Como se menciona en la página oficial, es en cambio, un conjunto de bibliotecas y herramientas. Este software es ampliamente utilizado en distintas aplicaciones robóticas, ya que facilita y acelera el desarrollo de estas aplicaciones.

2.7.1 Nodo

Los nodos son archivos ejecutables que forman parte de un paquete de ROS. Estos utilizan librerías para comunicarse con otros nodos a través de los temas(topics). Los nodos pueden publicar o suscribirse a un tema, así como solicitar o proveer un servicio. En general, los nodos no saben con quién se están comunicando. En cambio, los nodos que están interesados en los datos se suscriben al tema relevante. [40]

2.7.2 Temas

Los temas (topics) son buses de datos etiquetados sobre los cuales los nodos intercambian mensajes. Estos temas tienen una semántica de publicación/suscripción anónima, que desacopla la producción de información de su consumo. [41]

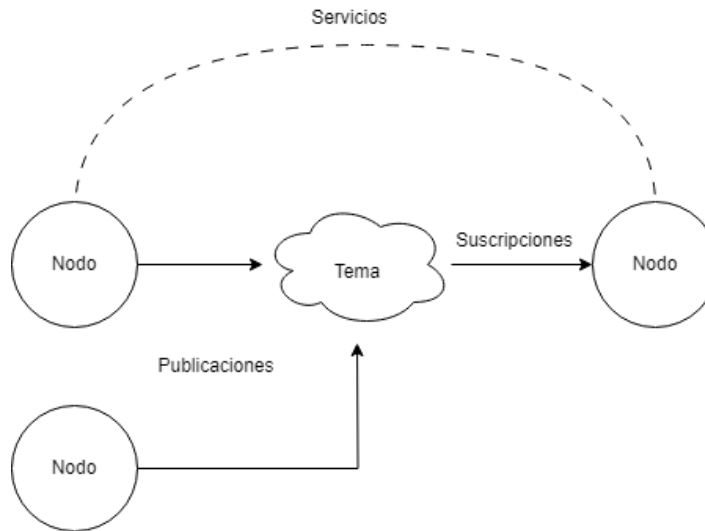


Figura 2.16: Esquema básico de comunicación en ROS.

2.8 Gazebo

”Gazebo permite hacer pruebas y experimentos con robots en escenarios físicos simulados de forma realista y con gráficas de alta calidad.” [42]

Además de su capacidad para simular entornos complejos, Gazebo ofrece un conjunto de características avanzadas, como la simulación de sensores y actuadores, la integración

con ROS y la posibilidad de desarrollar controladores y algoritmos de navegación dentro de su entorno de simulación.



Figura 2.17: Imgen que muestra ejemplo de simulación en gazebo, tomada de [43]

2.9 Isaac Sim

Isaac Sim es una plataforma de simulación desarrollada sobre Omniverse de NVIDIA, diseñada para probar, entrenar y desplegar robots que utilizan inteligencia artificial [44].

Además, Isaac Sim permite entrenar una sola política en múltiples entornos que se ejecutan en paralelo, lo que acelera el proceso de entrenamiento. Esto se logra al aprovechar principalmente la GPU, mientras que la CPU está limitada por la cantidad de hilos disponibles. Se ha demostrado que este enfoque reduce tanto el tiempo de entrenamiento como el hardware necesario para llevarlo a cabo [45]. También ofrece una integración sencilla con ROS, ya que permite publicar y suscribirse a temas e intercambiar mensajes con nodos externos. Esta interoperabilidad facilita la validación de algoritmos de percepción, planificación y control.

Isaac Sim puede ejecutarse en sistemas operativos Windows y Linux. Para su implementación adecuada en entornos de simulación robótica, es necesario considerar requisitos mínimos de hardware y software, así como configuraciones recomendadas para obtener un desempeño estable. Estos requerimientos se presentan en la tabla 2.1.



Figura 2.18: Entorno de simulación en Isaac Sim para pruebas de locomoción y control de robots [44].

Tabla 2.1: Requisitos del sistema para ejecutar Isaac Sim (x86_64) [46].

Elemento	Mínimo	Bueno	Ideal
OS	Ubuntu 22.04/24.04; Windows 10/11	Ubuntu 22.04/24.04; Windows 10/11	Ubuntu 22.04/24.04; Windows 10/11
CPU	Intel Core i7 (7th gen) o AMD Ryzen 5	Intel Core i7 (9th gen) o AMD Ryzen 7	Intel Core i9 serie X (o superior) o AMD Ryzen 9/Threadripper (o superior)
Núcleos	4	8	16
RAM	32 GB	64 GB	64 GB
Almacenamiento	50 GB SSD	500 GB SSD	1 TB NVMe SSD
GPU	GeForce RTX 4080	GeForce RTX 5080	RTX PRO 6000 Blackwell
VRAM	16 GB	16 GB	48 GB
Driver	Linux: 580.65.06; Windows: 580.88	Linux: 580.65.06; Windows: 580.88	Linux: 580.65.06; Windows: 580.88

Capítulo 3

Diseño y simulación

Este capítulo abarca tres fases principales: el diseño conceptual, el diseño del modelo en un software CAD especializado, y la exportación del modelo a un entorno de simulación virtual. Inicialmente, se describirá el enfoque y las metodologías utilizadas para el desarrollo del diseño del robot bípedo, incluyendo una visión general de las consideraciones para su diseño, como la selección preliminar de los actuadores necesarios para el movimiento. La segunda parte del capítulo se centrará en el diseño detallado del modelo 3D y los ensamblajes correspondientes. En esta fase, se abordará en profundidad la selección de los actuadores, especificando los tipos, tamaños y características técnicas necesarias para cumplir con los requisitos de movimiento del robot. También se explicarán las herramientas empleadas para crear el modelo tridimensional. Finalmente, se presentará la importación del modelo al entorno de simulación. Se abordarán los pasos necesarios para integrar el diseño en la plataforma de simulación, así como las configuraciones y parámetros ajustados para replicar con una buena aproximación las condiciones del mundo real. Además, se discutirán los resultados obtenidos de las simulaciones iniciales y se propondrán posibles mejoras y ajustes para optimizar el rendimiento de la simulación.

3.1 Diseño conceptual

Para iniciar el diseño conceptual, se analizó la configuración de los modelos funcionales más frecuente para robots bípedos, o el tren inferior de los robots humanoides, con 6 grados de libertad para cada pierna. Como se puede apreciar en distintos estudios [47–49], el modelo funcional más utilizado comúnmente es el que se muestra en la figura 3.1.

Luego, se llevó a cabo un análisis para desarrollar el modelo cinemático, el cual incluyó la creación de tres propuestas mostradas en la figura 3.2. La propuesta a) refleja el modelo funcional previamente mencionado, mientras que la propuesta b) intercambia las articulaciones 2 y 3, y luego las articulaciones 1 y 2. Finalmente, en la propuesta c), se cambia la articulación responsable de la rotación interna/externa de la pierna entre



Figura 3.1: Modelo funcional para una pierna comúnmente utilizado para robots bípedos de 12 grados de libertad.

las articulaciones de la rodilla y el tobillo.

Tras considerar las tres propuestas, se llegó a la conclusión de que utilizando la propuesta a), no habría mayores problemas con la cinemática directa. Sin embargo, al realizar ciertos ajustes, se podría simplificar la cinemática inversa mediante el desacople cinemático, particularmente al considerar la propuesta c).

Una vez decidido el modelo funcional a utilizar, se procedió con la selección de los actuadores. Aunque el robot solo será implementado en simulación, es importante tener en cuenta los actuadores para su eventual construcción.

Se optó por utilizar servomotores como actuadores debido a su capacidad para proporcionar alto torque y su capacidad de control de posición, lo que facilitará la implementación del robot en un entorno real. Esta elección elimina la necesidad de diseñar e implementar sistemas de control adicionales, permitiendo concentrarse en el control de los movimientos del robot. Se evaluaron cinco modelos de servomotores: Dynamixel AX-18A, TowerPro MG996R, Hiwonder HTS-16L, Hiwonder HTD-45H y Hiwonder HX-35HM. En la tabla 3.1 se muestran las características principales de los motores considerados.

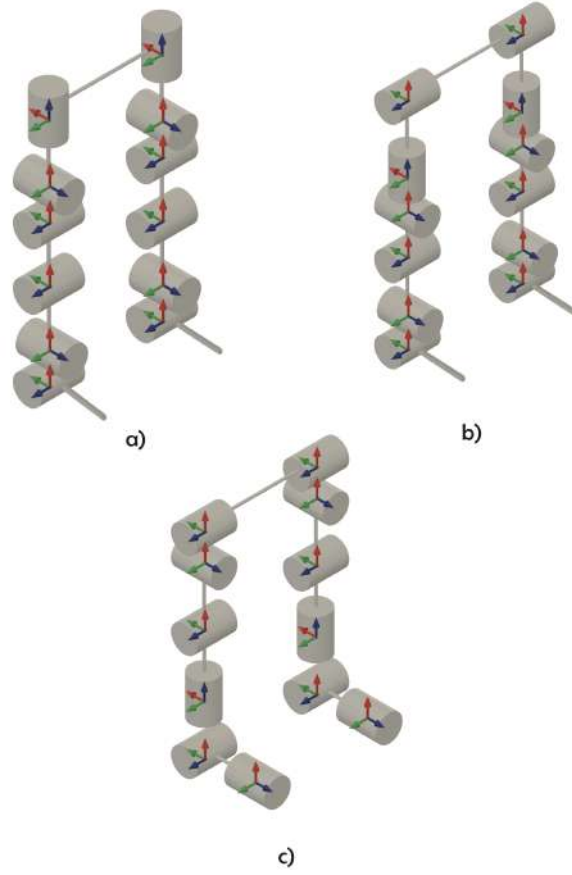


Figura 3.2: Propuestas de modelos funcionales para el robot bípedo.

Tabla 3.1: Características de los motores considerados

Modelo	Comunicación	Voltaje operativo	Torque	Corriente máxima
AX-18A	Serial TTL	11.1V	$18Kg \cdot cm$	1.6A
MG996R	PWM	7.2V	$11Kg \cdot cm$	1A
HTS-16L	Serial TTL	7.2V	$16Kg \cdot cm$	1A
HTD-45H	Serial TTL	11.1	$45Kg \cdot cm$	3A
HX-35HM	Serial TTL	11.1	$35Kg \cdot cm$	3A

Después de analizar las características y necesidades, se tomó la decisión que para el diseño se utilizarán los servomotores de la marca Hiwonder, específicamente el modelo HTD-45H.

3.2 Diseño CAD

Para el diseño CAD, se optó por utilizar Fusion 360® de Autodesk® bajo la licencia de estudiante. Se eligió este software debido a que ofrece un paquete completo con la licencia de estudiante, que incluye trabajo colaborativo en línea y procesamiento en la nube de forma gratuita (renderizado y simulaciones de elemento finito). Esta opción resulta muy conveniente ya que no se requiere una computadora potente para el procesamiento de los modelos. Una vez seleccionados los servomotores, se procedió a realizar un diseño preliminar del robot, para los eslabones de la cadera al tobillo se tomó como base el diseño de eslabones comerciales y se modificaron atendiendo las necesidades del robot, esto con el fin de reducir el tiempo de diseño, para la unión de las articulaciones del tobillo, el diseño se realizó completamente desde cero. El diseño preliminar se muestra en la figura 3.3.

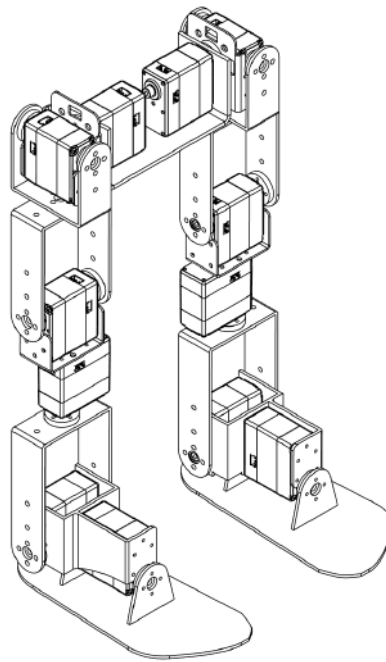


Figura 3.3: Diseño preliminar del robot bípedo.

La elección de materiales es importante ya que las propiedades de estos pueden ser tomadas en cuenta por el simulador, aumentando la similitud entre las simulaciones y el mundo real. Se ha decidido utilizar lámina de acero para los eslabones que van desde la cadera hasta el tobillo, debido a su resistencia y costo económico. Dado que estos

eslabones son de lámina, el peso no representa un factor significativo. Para el eslabón que une las articulaciones del tobillo, se ha seleccionado ABS (Acrilonitrilo butadieno estireno) debido a la complejidad de la forma del eslabón, la cual sería complejo replicar con dobleces de lámina, se ha seleccionado ABS debido a su buena resistencia mecánica, capaz de soportar las fuerzas aplicadas sobre este material. En cuanto al pie, se ha elegido lámina de acero como material.

3.3 Exportación a GAZEBO

Para poder exportar un modelo CAD a un simulador de entornos robóticos existe un formato llamado URDF (Unified Robot Description Format), este tipo de archivo contiene una gran cantidad de información que puede ser editada manualmente por el usuario para poder tener un control total de lo que sucede con las características del robot, para poder facilitar la edición manual se puede utilizar el formato XACRO (XML Macro), con este formato la edición de las características del robot resulta más sencilla. En el software Fusion 360® no se cuenta con una herramienta que exporte el modelo CAD al formato XACRO o URDF es por esto que se instala un plugin que lo hace de manera automática, las instrucciones de instalación y uso de este se puede encontrar en [50], cabe destacar que es importante seleccionar la versión adecuada ya que existe una versión para ROS y otra para ROS2, estas funcionan independientemente de la distribución de ROS que se esté utilizando. La distribución de ROS2 que se está utilizando para este trabajo de tesis es Irion Irwin. La ventaja que ofrece este plugin es que al exportar el modelo CAD genera un grupo de archivos en forma de paquete (figura 3.4), en este caso para ROS2, este paquete contiene los archivos necesarios para visualizar el robot en RVIZ y simularlo en GAZEBO, lo que agiliza el proceso de pruebas para la simulación.

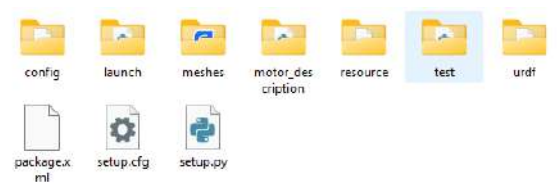


Figura 3.4: Archivos para paquete de ROS2.

El software de simulación utiliza las mallas en 3D generadas por el software CAD para realizar la simulación, estas mallas cuentan con un alto número de triángulos (más de 3500 triángulos en promedio), esta complejidad causa que el simulador realice

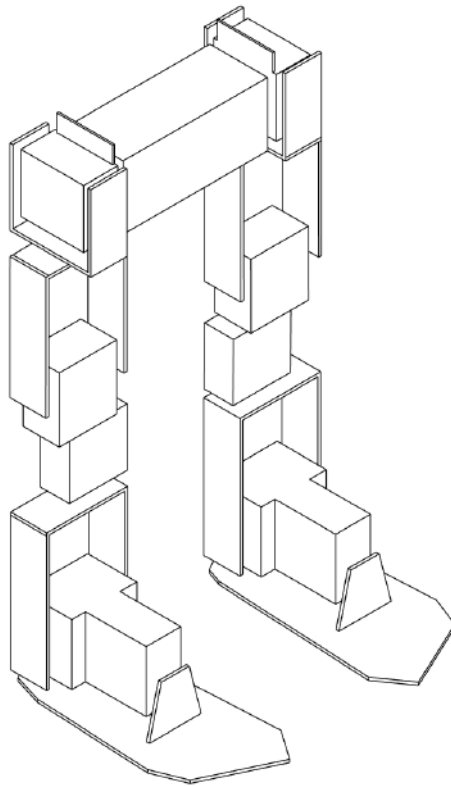


Figura 3.5: Polígonos de colisión.

múltiples operaciones por triángulo, lo que se traduce en un tiempo de simulación largo, se busca que las simulaciones se ejecuten en tiempo real, para lograrlo es necesario simplificar las piezas que conforman el robot disminuyendo el número de triángulos que las conforman, estos polígonos se conocen como polígonos de colisión. En la figura 3.5 se muestran los polígonos de colisión creados para el robot bípodo.

Una vez generados estos archivos, podemos ejecutar RVIZ para verificar la correcta relación entre articulaciones. La figura 3.6 muestra el robot en el entorno de RVIZ con los marcos coordenados visibles. En la figura 3.7 se muestra la visualización del robot, esta vez sin marcos coordenados para una visualización más clara del robot. Para validar la integridad cinemática del diseño, se llevó a cabo una prueba mediante la asignación manual de valores a las articulaciones del modelo. En la figura 3.8 se muestra el robot con una posición para verificar la correcta relación de las articulaciones.

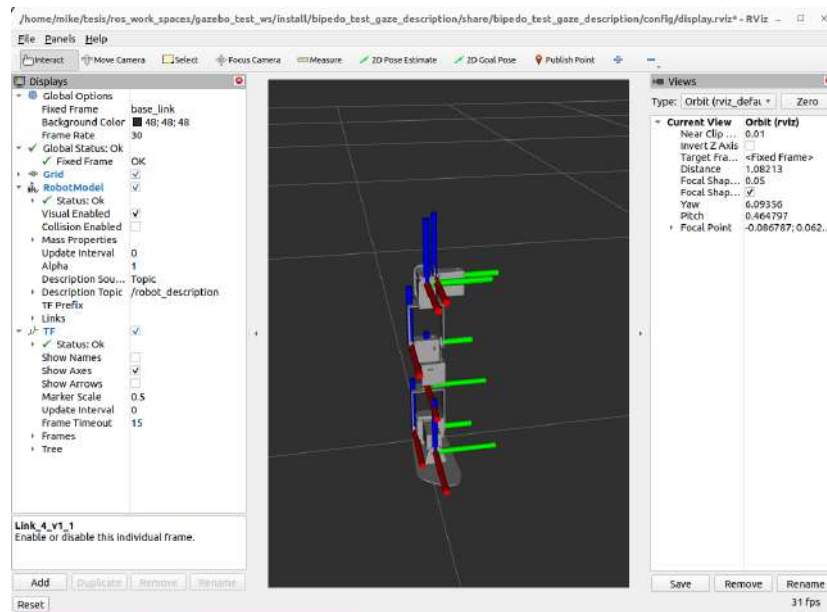


Figura 3.6: Visualización del robot en RVIZ con marcos coordenados.

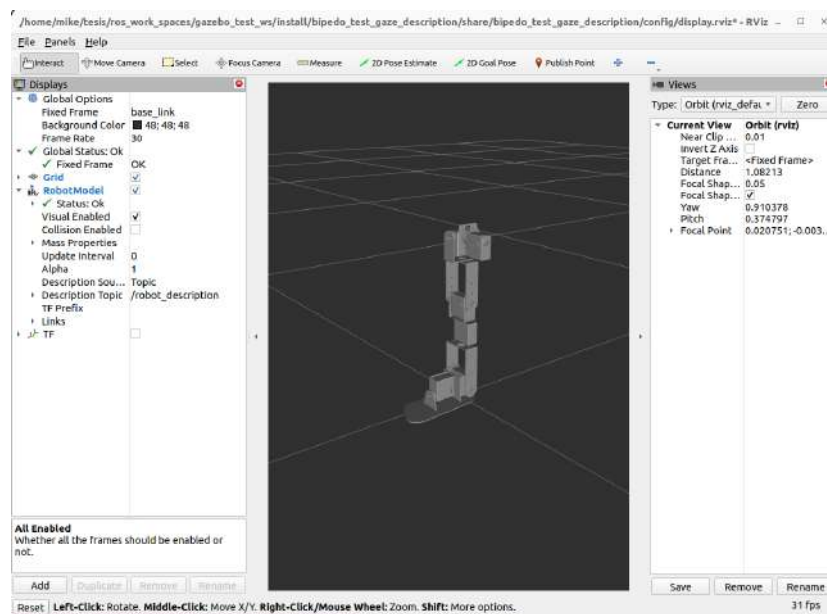


Figura 3.7: Visualización del robot en RVIZ sin marcos coordenados.

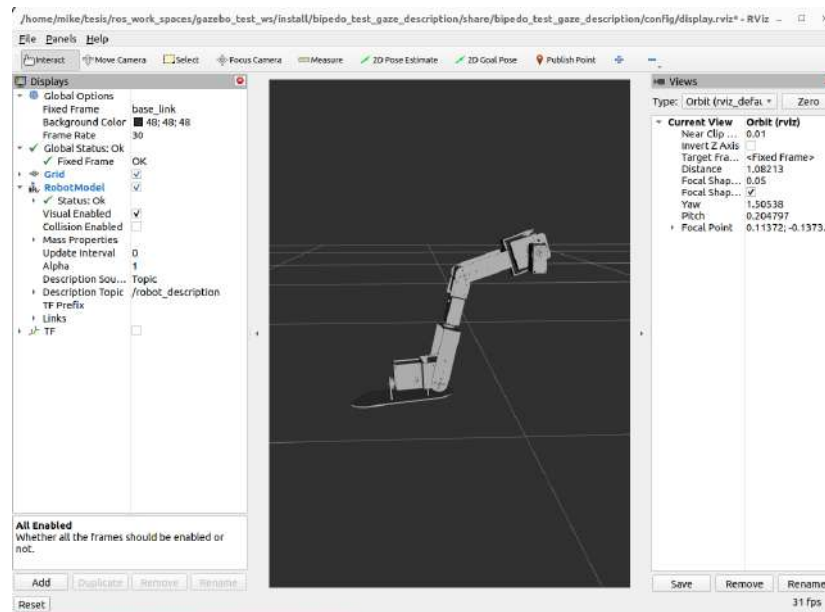


Figura 3.8: Prueba de movimientos de las articulaciones del robot.

Una vez verificado el correcto funcionamiento de las articulaciones del robot se realizaron pruebas en el simulador GAZEBO, las pruebas realizadas consistieron en verificar la correcta visualización del modelo del robot, así como la correcta interacción con la gravedad.

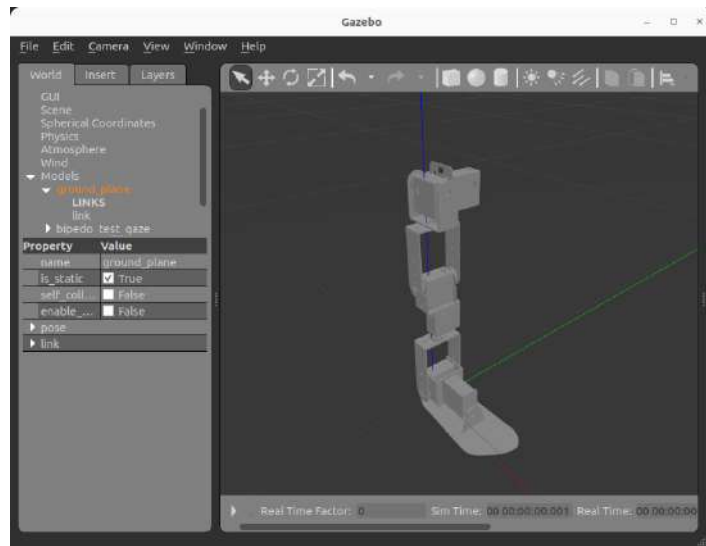


Figura 3.9: Modelo del robot en GAZEBO.

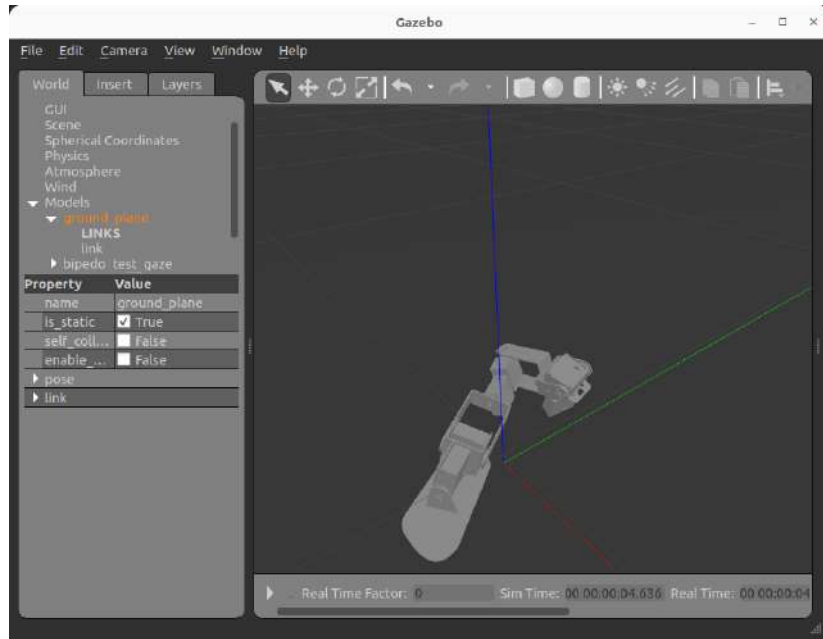


Figura 3.10: Interacción del robot con la gravedad del simulador.

Por último, la exportación del modelo a un entorno de simulación, apoyada en el uso de complementos, facilitó su integración en la plataforma ROS2. Las pruebas realizadas en RVIZ y GAZEBO confirmaron la correcta relación entre las articulaciones y la adecuada interacción del modelo con la gravedad. Asimismo, la importación del robot y los resultados favorables obtenidos en las simulaciones iniciales ponen de manifiesto la efectividad de las herramientas y metodologías empleadas. En conjunto, estas actividades han establecido una base sólida para el desarrollo de la investigación.

3.3.1 Importación a Isaac Sim

Isaac Sim soporta el formato URDF (Unified Robot Description Format); por lo tanto, permite importar robots modelados en este formato y convertirlos a una representación compatible con el entorno de simulación de NVIDIA Omniverse. Esta característica facilita la reutilización del modelo generado previamente para ROS2 y Gazebo, manteniendo la estructura de eslabones, articulaciones, masas, inercias y mallas asociadas al robot. Para iniciar el proceso de importación, en la esquina superior izquierda de la interfaz, dentro de la pestaña File, se debe seleccionar la opción de importación, como se muestra en la figura 3.11. A partir de esta herramienta, Isaac Sim interpreta el

archivo URDF y genera el modelo correspondiente dentro de la escena de simulación.

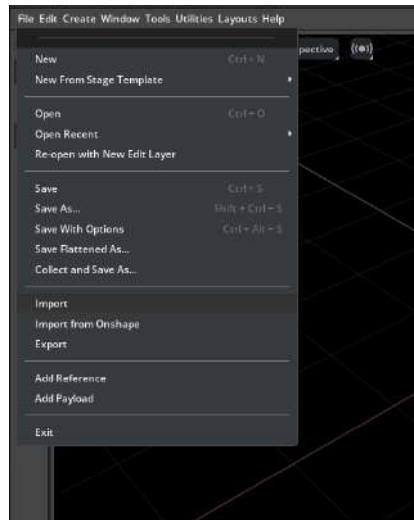


Figura 3.11: Opción para importar URDF en Isaac Sim.

Al hacerlo, se abrirá una ventana desde la cual se podrá navegar por los archivos de la computadora hasta localizar el archivo URDF. Una vez ubicado, basta con seleccionarlo y presionar el botón de importación (Figura 3.12). En la figura 3.13 se muestra el modelo importado en el simulador Isaac Sim.

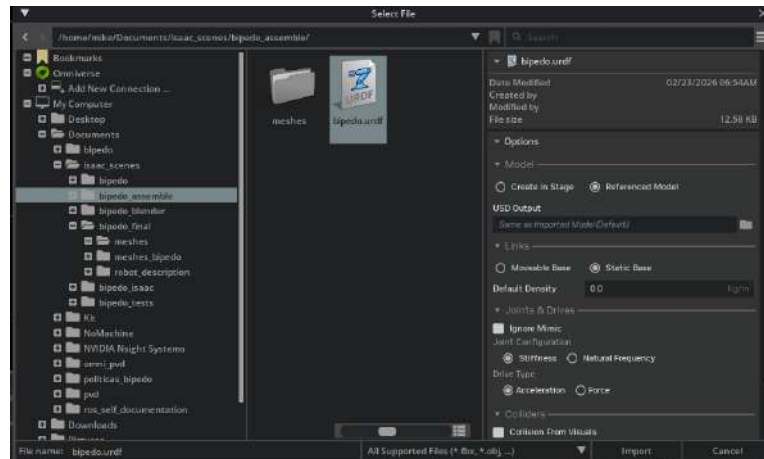


Figura 3.12: Ventana para importar el archivo URDF.

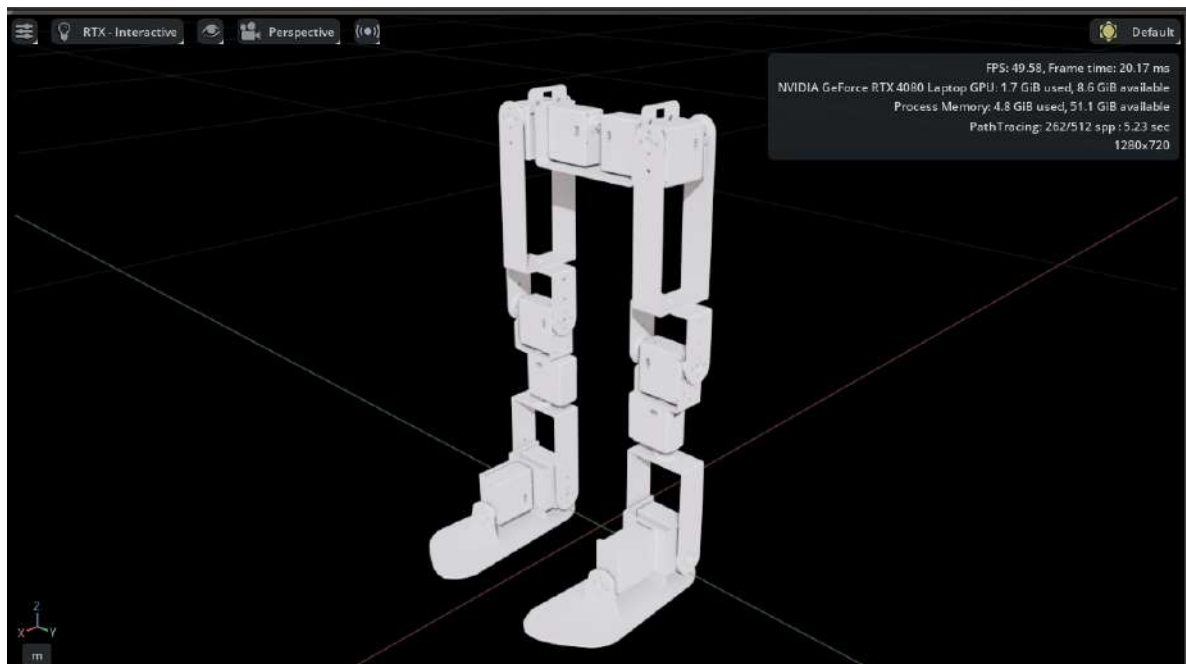


Figura 3.13: Modelo URDF importado en Isaac Sim.

Capítulo 4

Modelo matemático

La cinemática de los robots manipuladores se encarga del estudio y control de los movimientos de los robots. Este capítulo se enfoca en los modelos de cinemática directa e inversa, dos modelos necesarios para la determinación de la posición y orientación de las piernas consideradas como manipuladores.

Recordando el marco teórico del capítulo 2. La cinemática directa consiste en calcular la posición y orientación del efector final de un robot a partir de los valores de las variables articulares, es decir, de las posiciones de las articulaciones. En cambio, la cinemática inversa, dada una posición y orientación deseadas para el efector final, se busca encontrar los valores correspondientes de las articulaciones. A lo largo de este capítulo, se presentarán los principios matemáticos que sustentan estos modelos y los métodos de resolución empleados para cada uno de ellos.

4.1 Cinemática directa

El objetivo de la cinemática directa es encontrar la posición y la orientación del efector final del robot [22] (en este caso el pie del robot) en términos del origen (centroide de la cadera del robot) con base en la posición articular del robot, esta relación está dada por las matrices de transformación homogénea que se forman a partir de las traslaciones y rotaciones dadas a partir de los parámetros DH. Las matrices de transformación homogénea se obtuvieron con el software MATLAB, los scripts se encuentran en el Apéndice B.

Para realizar el modelo de cinemática directa, cada pierna se considera como un robot manipulador de 6 GDL (Grados De Libertad). Tomando en cuenta la metodología de DH se encuentran los ejes articulares y se determina la orientación de los marcos coordenados. En la figura 4.1a se muestra la disposición de los marcos coordenados de la pierna derecha del robot y en la figura 4.1b se muestran para la pierna izquierda.

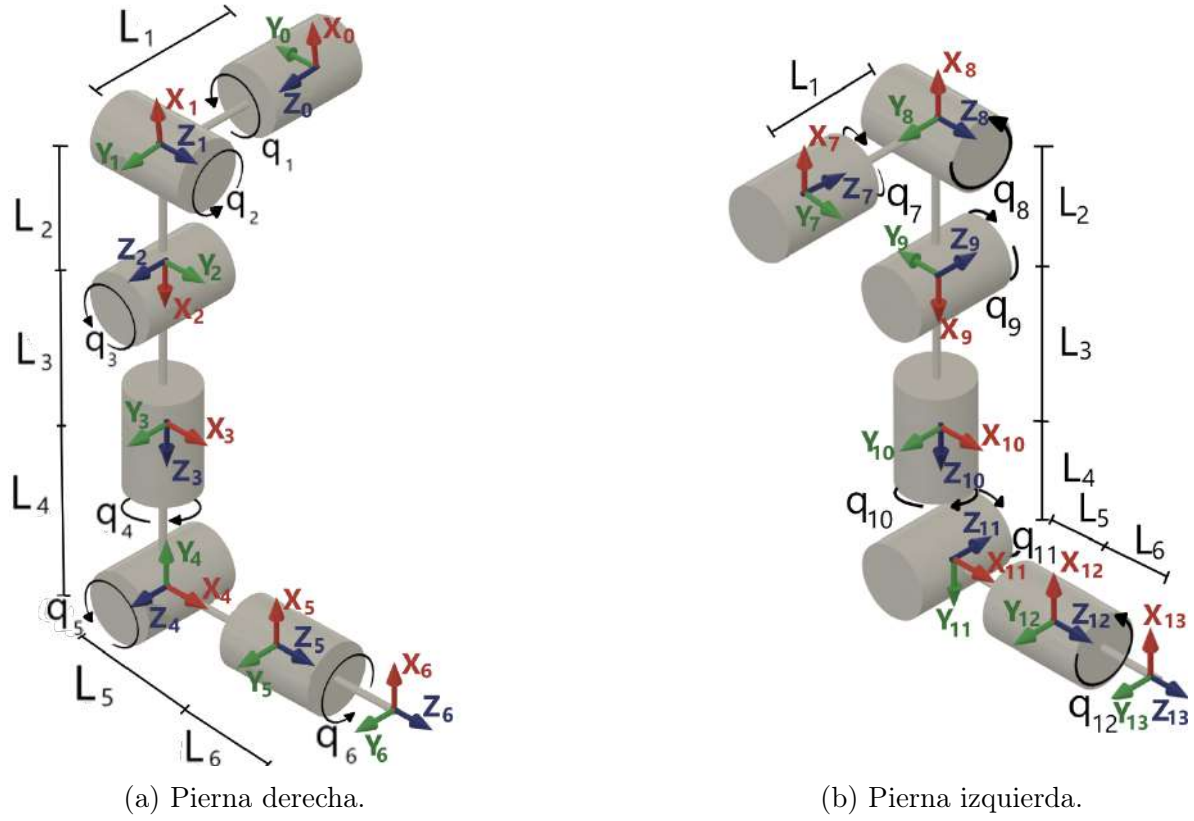


Figura 4.1: Marcos coordenados para los parámetros DH de ambas piernas.

Una vez definidos los sentidos de giro de las articulaciones y las orientaciones de los marcos coordenados, se obtienen los parámetros DH, estos se muestran en la tabla 4.1 y los parámetros para la pierna izquierda se muestran en la tabla 4.2.

Las longitudes geométricas utilizadas en el modelo cinemático, expresadas en metros, son

$$\begin{aligned}
 L_1 &= 0.069 \text{ m}, & L_2 &= 0.132 \text{ m}, & L_3 &= 0.080 \text{ m}, \\
 L_4 &= 0.084 \text{ m}, & L_5 &= 0.068 \text{ m}, & L_6 &= 0.020 \text{ m}.
 \end{aligned}
 \tag{4.1}$$

Tabla 4.1: Parámetros DH para la pierna derecha del robot.

Articulación	Z		X	
	θ_i	d_i	a_i	α_i
1	q_1	L_1	0	$\frac{\pi}{2}$
2	$q_2 + \pi$	0	L_2	$\frac{\pi}{2}$
3	$q_3 + \frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
4	q_4	$L_3 + L_4$	0	$-\frac{\pi}{2}$
5	$q_5 + \frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
6	q_6	$L_5 + L_6$	0	0

Tabla 4.2: Parámetros DH para la pierna izquierda del robot.

Articulación	Z		X	
	θ_i	d_i	a_i	α_i
7	q_7	L_1	0	$-\frac{\pi}{2}$
8	$q_8 + \pi$	0	L_2	$-\frac{\pi}{2}$
9	$q_9 - \frac{\pi}{2}$	0	0	$-\frac{\pi}{2}$
10	q_{10}	$L_3 + L_4$	0	$\frac{\pi}{2}$
11	$q_{11} - \frac{\pi}{2}$	0	0	$-\frac{\pi}{2}$
12	q_{12}	$L_5 + L_6$	0	0

Una vez obtenidos los parámetros de DH, se forman las matrices de transformación homogénea A_i sustituyendo los parámetros obtenidos anteriormente. Matrices A_i para la pierna derecha.

$$A_1^0 = \begin{bmatrix} \cos(q_1) & 0 & \sin(q_1) & 0 \\ \sin(q_1) & 0 & -\cos(q_1) & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2^1 = \begin{bmatrix} -\cos(q_2) & 0 & -\sin(q_2) & -L_2 \cdot \cos(q_2) \\ -\sin(q_2) & 0 & -\cos(q_2) & -L_2 \cdot \sin(q_2) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^2 = \begin{bmatrix} \cos(q_3 + \frac{\pi}{2}) & 0 & \sin(q_3 + \frac{\pi}{2}) & 0 \\ \sin(q_3 + \frac{\pi}{2}) & 0 & \cos(q_3 + \frac{\pi}{2}) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4^3 = \begin{bmatrix} \cos(q_4) & 0 & -\sin(q_4) & 0 \\ \sin(q_4) & 0 & \cos(q_4) & 0 \\ 0 & -1 & 0 & L_3 + L_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5^4 = \begin{bmatrix} \cos(q_5 + \frac{\pi}{2}) & 0 & \sin(q_5 + \frac{\pi}{2}) & 0 \\ \sin(q_5 + \frac{\pi}{2}) & 0 & -\cos(q_5 + \frac{\pi}{2}) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_6^5 = \begin{bmatrix} \cos(q_6) & 0 & -\sin(q_6) & 0 \\ \sin(q_6) & 0 & \cos(q_6) & 0 \\ 0 & 0 & 1 & L_5 + L_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Continuando con la metodología de DH para determinar la cinemática directa de un robot manipulador, se crea la matriz T_6^0 .

$$T_6^0 = A_1^0 \cdot A_2^1 \cdot A_3^2 \cdot A_4^3 \cdot A_5^4 \cdot A_6^5 \quad (4.2)$$

En la ecuación 4.3 se observa la matriz de transformación homogénea genérica. Para observar la matriz completa del robot véase apéndice A.

$$T = \begin{bmatrix} n_x & o_x & a_x & t_x \\ n_y & o_y & a_y & t_y \\ n_z & o_z & a_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.3)$$

Como puede observarse, la matriz contiene explícitamente los términos de traslación t_x , t_y y t_z ; por ello, dichas traslaciones se expresan como se muestra en la ecuación 4.4.

$$\begin{aligned} t_x &= T_{61,4}^0 \\ t_y &= T_{62,4}^0 \\ t_z &= T_{63,4}^0 \end{aligned} \quad (4.4)$$

Para obtener las rotaciones alrededor de los ejes es necesario realizar algunas operaciones con los elementos de la matriz de transformación homogénea. La matriz de transformación homogénea alrededor de los ejes x , y y z con ángulos α , β y γ respectivamente ($T_{\alpha\beta\gamma}$), sin considerar traslaciones, se define en la ecuación 4.5. En esta sección, la función seno se representa como S y la función coseno como C ; el subíndice indica el ángulo (por ejemplo, $\cos(\alpha) = C_\alpha$):

$$T_{\alpha\beta\gamma} = \begin{bmatrix} C_\beta \cdot C_\gamma & -C_\beta \cdot S_\gamma & S_\beta & 0 \\ C_\alpha \cdot S_\gamma + C_\gamma \cdot S_\alpha \cdot S_\beta & C_\alpha \cdot C_\gamma - S_\alpha \cdot S_\beta \cdot S_\gamma & -C_\beta \cdot S_\alpha & 0 \\ S_\alpha \cdot S_\gamma - C_\alpha \cdot C_\gamma \cdot S_\beta & C_\gamma \cdot S_\alpha + C_\alpha \cdot S_\beta \cdot S_\gamma & C_\alpha \cdot C_\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.5)$$

De la relación entre 4.3 y 4.5 se pueden obtener las ecuaciones mostradas en 4.6.

$$\begin{aligned} \alpha &= \text{atan}\left(\frac{-a_y}{a_z}\right) \\ \beta &= \text{asin}(a_x) \\ \gamma &= \text{atan}\left(\frac{-o_x}{n_x}\right) \end{aligned} \quad (4.6)$$

4.2 Cinemática inversa

El desacoplo cinemático es una técnica utilizada para simplificar el problema de la cinemática inversa en robots manipuladores. Recordando el marco teórico en el capítulo 2. El desacoplo cinemático consiste en descomponer el movimiento del efector final en dos o más componentes independientes, permitiendo que el sistema se resuelva de manera más eficiente y menos compleja, la obtención de las variables articulares comienza con la identificación de las relaciones entre las variables de posición y orientación del efector final y las variables articulares del manipulador. A través de un proceso de desacoplo, se pueden separar las variables de posición de las variables de orientación.

4.2.1 Cinemática de posición

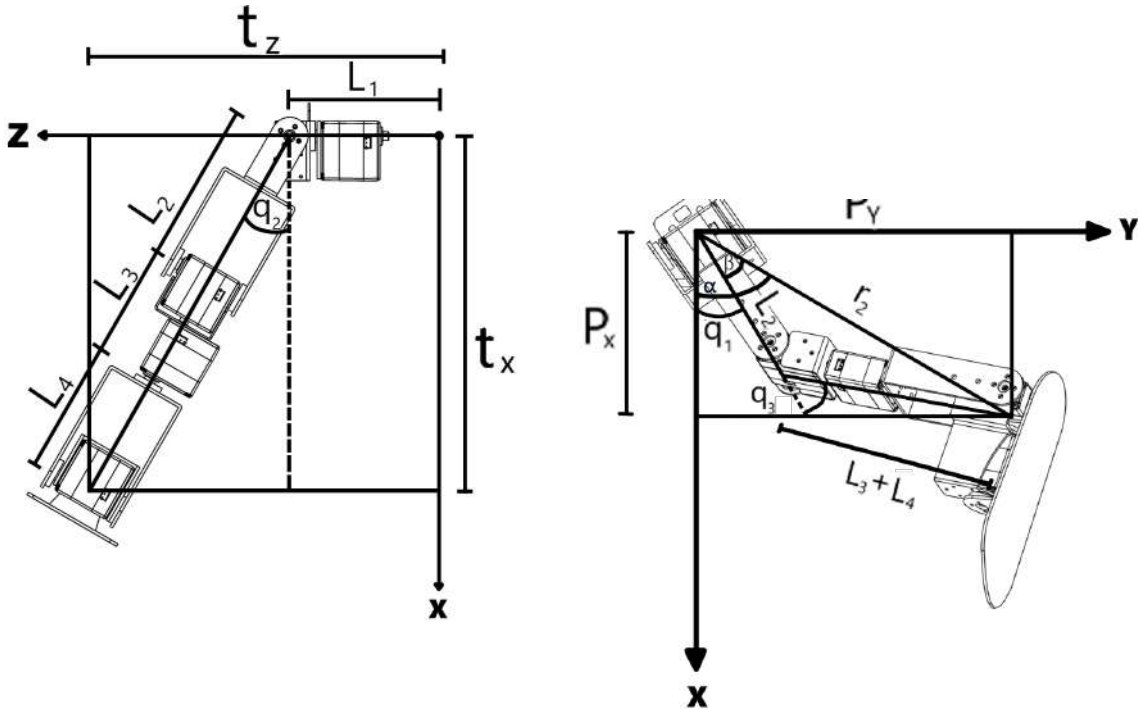
Para el cálculo de la cinemática inversa se utilizó el método geométrico mencionado en [22]. Para poder analizar el modelo, el robot se observó desde el plano (Z,X) y el plano normal al eje de la articulación q_3 , tomando en cuenta que el método a utilizar para la cinemática inversa será desacoplo cinemático primero se calcularán las 3 primeras variables articulares (posición). En las figuras 4.2a y 4.2b se muestra la geometría para calcular la articulación q_2 .

$$q_2 = \text{atan}\left(\frac{t_z - L_1}{t_x}\right) \quad (4.7)$$

$$r_1 = \sqrt{(t_x)^2 + (t_z - L_1)^2}$$

$$r_2 = \sqrt{t_y^2 + r_1^2}$$

Para encontrar q_1 se debe tomar en cuenta el triángulo interno de la figura 4.2b, tomando en cuenta que solo se conocen los lados, para encontrar los ángulos se utiliza la ley de los cosenos.



(a) Geometría para calcular la articulación q_2 .

(b) Geometría para calcular las articulaciones q_1 y q_3 .

Figura 4.2: Geometría utilizada para el cálculo de las articulaciones en la cinemática inversa.

$$\alpha = \text{atan}\left(\frac{t_y}{r_1}\right)$$

$$\beta = \text{acos}\left(\frac{(L_3 + L_4)^2 - L_2^2 - r_2^2}{-2 \cdot L_2 \cdot r_2}\right)$$

$$q_1 = \alpha - \beta \quad (4.8)$$

$$q_3 = \arccos\left(\frac{r_2^2 - L_2^2 - (L_3 + L_4)^2}{2 \cdot L_2 \cdot (L_3 + L_4)}\right) \quad (4.9)$$

4.2.2 Cinemática de orientación

Una vez determinados los primeros tres grados de libertad que definen la posición del centro de la muñeca, es necesario obtener la matriz R_6^3 . Mediante la cinemática directa y los tres primeros grados de libertad, se puede calcular la matriz R_3^0 . Con la rotación deseada, se obtiene la matriz R_6^0 y despejando de la cinemática directa, se puede encontrar la matriz R_6^3 .

$$R_6^0 = R_3^0 \cdot R_6^3$$

$$R_6^3 = (R_3^0)^{-1} \cdot R_6^0$$

Considerando que las matrices de rotación son "*special ortogonal*", es decir, que su inversa es igual a su transpuesta

$$R_6^3 = (R_3^0)^T \cdot R_6^0$$

donde

$$R_6^3 = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

$$R_6^3 = \begin{bmatrix} C_{q_4} \cdot C_{q_6} \cdot S_{q_5} - S_{q_4} \cdot S_{q_6} & -C_{q_6} \cdot S_{q_4} - C_{q_4} \cdot S_{q_5} \cdot S_{q_6} & C_{q_4} \cdot C_{q_5} \\ C_{q_4} \cdot S_{q_6} + C_{q_6} \cdot S_{q_4} \cdot S_{q_5} & C_{q_4} \cdot C_{q_6} - S_{q_4} \cdot S_{q_5} \cdot S_{q_6} & C_{q_5} \cdot S_{q_4} \\ -C_{q_5} \cdot C_{q_6} & C_{q_5} \cdot S_{q_6} & S_{q_5} \end{bmatrix}$$

Los tres últimos grados de libertad están dispuestos de tal manera que forman una muñeca esférica. De esta forma, las rotaciones del manipulador corresponden a las rotaciones de los ángulos de Euler ZYZ , los cuales están representados por las variables articulares q_4 , q_5 y q_6 . En las ecuaciones 4.10-4.12 se presentan las ecuaciones correspondientes a estas últimas variables articulares.

$$q_4 = \text{atan}\left(\frac{R_{23}}{R_{13}}\right) \quad (4.10)$$

$$q_5 = \text{atan}\left(\frac{R_{33}}{\sqrt{1 - (R_{33})^2}}\right) \quad (4.11)$$

$$q_6 = \text{atan}\left(\frac{R_{32}}{-R_{31}}\right) \quad (4.12)$$

4.3 Pruebas de lazo abierto

4.3.1 Cinemática directa

Para verificar que el modelo de cinemática directa es correcto, el robot se simuló, obteniendo la posición exacta del pie respecto al centroide de la cadera del robot. Se realizaron 3 pruebas para distintos puntos, en la figura 4.3 se muestra la prueba para el punto 1, en la tabla 4.3 se muestran los resultados para los 3 puntos.

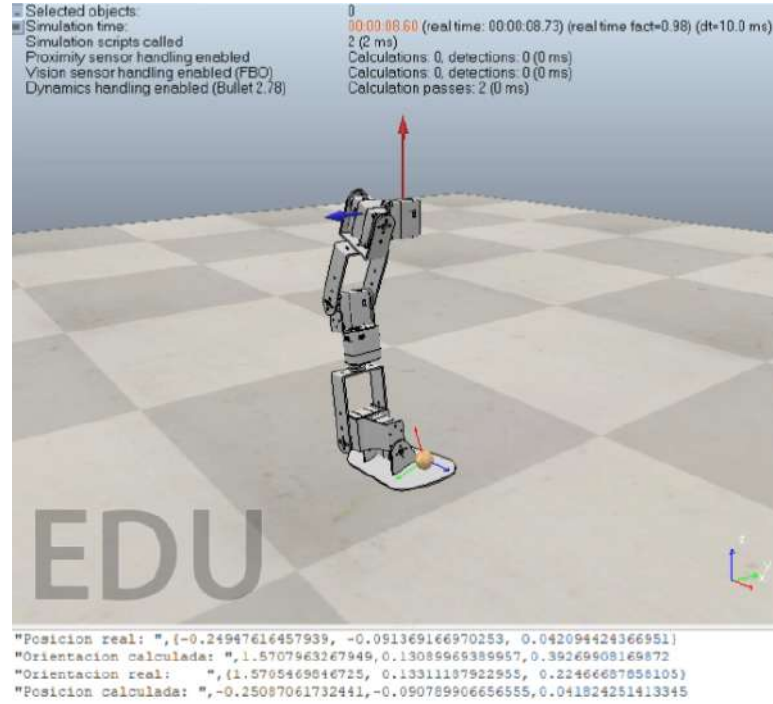


Figura 4.3: Prueba de lazo bierto para el punto 1

Tabla 4.3: Comparación entre posiciones medidas y calculadas

Punto 1	X	Y	Z
Posición medida	-0.2490	-0.0913	0.0420
Posición calculada	-0.2510	-0.09	0.0418
	α	β	γ
Orientación medida	1.57	0.1310	0.3930
Orientación calculada	1.57	0.1330	0.2250
Punto 2	X	Y	Z
Posición medida	-0.1370	-0.1790	0.2370
Posición calculada	-0.1370	-0.1800	0.2360
	α	β	γ
Orientación medida	1.57	0.3930	-0.7850
Orientación calculada	1.57	0.3930	-0.7850
Punto 3	X	Y	Z
Posición medida	-0.1043	-0.1957	0.1461
Posición calculada	-0.1043	-0.1957	0.1461
	α	β	γ
Orientación medida	2.18	0.5235	-0.9553
Orientación calculada	2.18	0.5236	-0.9551

4.4 Cálculo de velocidades mediante la matriz Jacobiana

Como se mencionó en el capítulo 2.1.3 se puede calcular la velocidad del efector final en función de las velocidades articulares, para el caso particular de este robot se utilizó el jacobiano analítico, para obtener la relación que existe entre las articulaciones y la pose del efector final se utilizó la cinemática directa quedando como se muestra en 4.13

$$q(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \\ \alpha(t) \\ \beta(t) \\ \gamma(t) \end{bmatrix} \quad (4.13)$$

con esta expresión se puede obtener el jacobiano analítico derivando temporalmente 4.13

$$J(q) = \begin{bmatrix} \frac{\partial x(t)}{\partial q_1} & \frac{\partial x(t)}{\partial q_2} & \cdots & \frac{\partial x(t)}{\partial q_6} \\ \frac{\partial y(t)}{\partial q_1} & \frac{\partial y(t)}{\partial q_2} & \cdots & \frac{\partial y(t)}{\partial q_6} \\ \frac{\partial z(t)}{\partial q_1} & \frac{\partial z(t)}{\partial q_2} & \cdots & \frac{\partial z(t)}{\partial q_6} \\ \frac{\partial \alpha(t)}{\partial q_1} & \frac{\partial \alpha(t)}{\partial q_2} & \cdots & \frac{\partial \alpha(t)}{\partial q_6} \\ \frac{\partial \beta(t)}{\partial q_1} & \frac{\partial \beta(t)}{\partial q_2} & \cdots & \frac{\partial \beta(t)}{\partial q_6} \\ \frac{\partial \gamma(t)}{\partial q_1} & \frac{\partial \gamma(t)}{\partial q_2} & \cdots & \frac{\partial \gamma(t)}{\partial q_6} \end{bmatrix} \quad (4.14)$$

como resultado se obtiene una matriz $J(q) \in \mathbb{R}^{6 \times 6}$, con esto se puede calcular la velocidad del efector final con

$$\begin{bmatrix} V \\ W \end{bmatrix} = J(q)\dot{q} \quad (4.15)$$

si se desea controlar la velocidad del efector final del robot es necesario controlar la velocidad articular, para esto se utiliza la inversa del jacobiano $J(q)^{-1}$

$$\dot{q} = J(q)^{-1} \begin{bmatrix} V \\ W \end{bmatrix} \quad (4.16)$$

de esta forma teniendo una velocidad lineal y angular deseadas se pueden calcular las velocidades articulares necesarias para alcanzar estas velocidades.

Capítulo 5

Punto de momento nulo y aprendizaje por refuerzo

5.1 Cálculo del Punto de Momento Nulo

En el capítulo 2 del marco teórico se abordó el *Zero Moment Point* (ZMP, por sus siglas en inglés), el punto de momento nulo se utiliza para evaluar la estabilidad de robots con patas, en este caso el robot es bípedo, el ZMP es un punto en el suelo, en el cual la sumatoria de los momentos alrededor de los ejes X,Y es cero [27], esto considera la inercia del robot así como la fuerza de gravedad. El polígono de soporte es el cierre convexo de todos los puntos de contacto del robot con el suelo [27], en este caso para el cierre convexo se consideraron solo las plantas de los pies del robot como se muestra en la imagen 5.1.

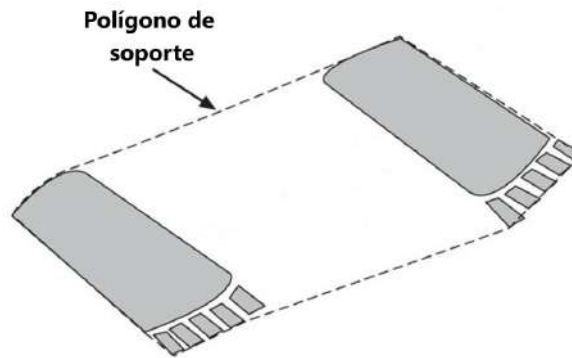


Figura 5.1: Polígono de soporte considerando la planta de los pies del robot, Adaptado de [51].

Si el ZMP se encuentra dentro del polígono de soporte el robot se encuentra estable, si el ZMP se encuentra en los bordes o muy cerca de estos, el robot será inestable.

El cálculo del ZMP involucra tanto el momento de inercia lineal como el momento de inercia angular para cada uno de los cuerpos rígidos del robot en relación con el marco coordinado del origen del mundo, como se muestra en la ecuación 5.1

$$\begin{aligned} X_{ZMP} &= \frac{M \cdot g \cdot X - Z \cdot \dot{\mathcal{P}}_X + \dot{\mathcal{L}}_Y}{M \cdot g + \dot{\mathcal{P}}_Z} \\ Y_{ZMP} &= \frac{M \cdot g \cdot Y + Z \cdot \dot{\mathcal{P}}_Y - \dot{\mathcal{L}}_X}{M \cdot g + \dot{\mathcal{P}}_Z} \end{aligned} \quad (5.1)$$

donde X_{zmp}/Y_{zmp} son la posición en los ejes X/Y del ZMP, M es la masa del robot, g es la gravedad, $X/Y/Z$ son la posición del centro de masa del robot en sus respectivos ejes, $\mathcal{P}_X/\mathcal{P}_Y/\mathcal{P}_Z$ son los momentos lineales en sus respectivos ejes y $\mathcal{L}_X/\mathcal{L}_Y/\mathcal{L}_Z$ son los momentos angulares alrededor de sus respectivos ejes.

En simulación realizar estos cálculos es sencillo, en el caso del mundo real es complicado conocer estas características, por lo que el cálculo del ZMP se puede simplificar de la siguiente manera: si se considera que el centro de masa general del robot se encuentra a la misma altura del origen y el centro de masa permanece a una misma altura, es decir no tiene aceleraciones en el eje Z , se pueden obtener las siguientes ecuaciones para las coordenadas (X,Y) del ZMP como se muestra la ecuación 5.2.

$$\begin{aligned} X_{ZMP} &= X - \frac{Z}{g} \ddot{X} \\ Y_{ZMP} &= Y - \frac{Z}{g} \ddot{Y} \end{aligned} \quad (5.2)$$

Con esta simplificación se puede estimar el zmp, en función de la velocidad del robot y la posición general del centro de masa del robot.

La posición y la velocidad del robot se obtiene directamente del simulador, las coordenadas del ZMP se muestran a través de un indicador visual rojo. En la imagen 5.2 se muestran los resultados obtenidos del cálculo del ZMP en la simulación, el ZMP se encuentra indicado mediante un indicador visual rojo.

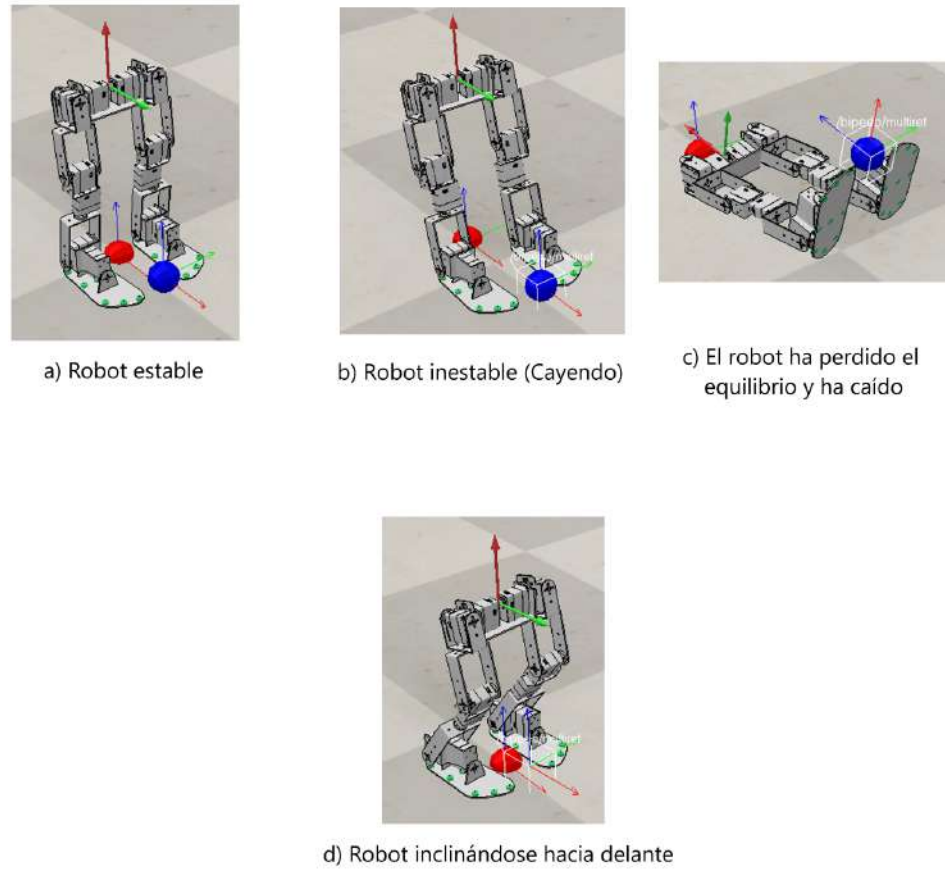


Figura 5.2: ZMP en distintas posiciones en el plano XY en función de la inclinación del robot.

5.2 Definición del espacio de estados

La definición del espacio de estados es fundamental en el aprendizaje por refuerzo, ya que la acción del agente depende de la función de recompensa, la cual, a su vez, está determinada por el estado del ambiente y la acción del agente [29]. Al revisar el estado del arte 9, 10, 13, 14, especialmente en [14] se identifican varios estados relevantes para su utilización, tales como: velocidad lineal, velocidad angular, posición angular de cada articulación, vector de proyección de gravedad, etc.

En la tabla 5.1 se detallan las variables correspondientes al espacio de estados del sistema.

Tabla 5.1: Variables del espacio de estados para el algoritmo de RL a utilizar.

Variable	Descripción
$v \in \mathbb{R}^3$	Velocidad lineal de la cadera del robot
$\omega \in \mathbb{R}^3$	Velocidad angular de la cadera del robot
$g_b \in \mathbb{R}^3$	Vector de proyección de gravedad
$J_w \in \mathbb{R}^{12}$	Velocidad angular de cada articulación
$J_\theta \in \mathbb{R}^{12}$	Posición angular de cada articulación
$V_{cmd} \in \mathbb{R}^3$	Comandos de velocidad
$a_{t-1} \in \mathbb{R}^{12}$	Acción anterior

5.3 Definición de las acciones del agente

Recordando el marco teórico del capítulo 2, un agente interactúa con su entorno realizando acciones con el objetivo de maximizar la recompensa a largo plazo [29]. Ya se ha definido el vector de estados, por lo que ahora es necesario especificar las acciones que el agente llevará a cabo.

Para mantener el equilibrio, es necesario mover las piernas; por lo tanto, la acción del robot consiste en realizar movimientos en las piernas, estos se realizarán controlando la posición articular.

Tabla 5.2: Vector del espacio de acciones del agente basado en posiciones articulares.

Variable	Descripción
θ_1	Posición articular 1 de la pierna derecha
θ_2	Posición articular 2 de la pierna derecha
θ_3	Posición articular 3 de la pierna derecha
θ_4	Posición articular 4 de la pierna derecha
θ_5	Posición articular 5 de la pierna derecha
θ_6	Posición articular 6 de la pierna derecha
θ_7	Posición articular 1 de la pierna izquierda
θ_8	Posición articular 2 de la pierna izquierda
θ_9	Posición articular 3 de la pierna izquierda
θ_{10}	Posición articular 4 de la pierna izquierda
θ_{11}	Posición articular 5 de la pierna izquierda
θ_{12}	Posición articular 6 de la pierna izquierda

5.4 Selección de algoritmo de aprendizaje reforzado a utilizar

El aprendizaje por refuerzo es una técnica de aprendizaje automático ampliamente investigada y utilizada en muchos ámbitos. Debido a esto, existen múltiples algoritmos para abordar diferentes problemas. En el caso de la caminata de un robot bípedo, es necesario analizar y seleccionar un algoritmo que sea capaz de manejar la complejidad de dicho problema. En la tabla 5.3 se muestra una comparativa de algunos de los algoritmos más importantes utilizados en el estado del arte.

Tabla 5.3: Comparativa de algoritmos de aprendizaje reforzado

Algoritmo	Tipo	Exploración vs. Explotación	Ventajas	Desventajas	Aplicación
<i>Q-Learning</i> [52]	Valor	ϵ -greedy	Simple, converge en entornos pequeños	No escala a espacios grandes	Juegos, entornos discretos
DQN [53]	Valor + red neuronal	ϵ -greedy	Maneja espacios grandes	Inestable sin ajustes	Videojuegos, robótica
<i>Policy Gradient</i> [54]	Grad. política	Estocástica	Aprende la política directa	Alta varianza, muchos datos	Robótica, control cont.
PPO [38]	Grad. política	Estocástica	Estable, implementación simple	Requiere cómputo	Juegos, robótica, simul.
<i>Actor-Critic</i> [55]	Política + valor + red neuronal	Estocástica	Menor varianza, estable	Alto costo comput.	Control cont., robótica
<i>Twin Delayed DDPG</i> [56]	Política + valor + red neuronal	Determinista + ruido	Reduce sobreestimación de Q	Alto costo, exploración media	Optimización, robótica, control cont.

Los estados del robot pertenecen a un espacio continuo, es decir, existen infinitas configuraciones posibles. En consecuencia, la complejidad del problema es elevada. Habiendo hecho un análisis del estado del arte, el algoritmo *Proximal Policy Optimization* (PPO) es adecuado para realizar el entrenamiento, ya que, como se menciona en [14], con este algoritmo se pueden entrenar múltiples robots de manera simultánea reduciendo significativamente el tiempo de entrenamiento mostrando que el algoritmo es capaz de transmitir la información aprendida en la simulación a un entorno real. Por lo tanto, en este trabajo se empleará el algoritmo *Proximal Policy Optimization* para resolver el problema.

5.5 Arquitectura del ciclo de control

La figura 5.3 muestra el ciclo cerrado utilizado para controlar el robot y entrenar la política mediante PPO. En cada instante t , los comandos de velocidad se combinan con las mediciones proporcionadas por los sensores para construir el estado $s(t)$. Este vector contiene la velocidad lineal y angular de la cadera, la proyección de la gravedad, las posiciones y velocidades articulares, y la acción aplicada en el instante anterior, de acuerdo con el espacio de estados definido previamente.

El estado $s(t)$ se suministra a las redes neuronales del esquema actor-crítico. La red actor representa la política y genera la acción $a(t)$, correspondiente a las posiciones articulares deseadas que se aplican al robot dentro del entorno de simulación. Como resultado de esta interacción, el estado dinámico del robot cambia y los sensores producen nuevas mediciones, con las cuales se construye el estado del siguiente instante y se cierra el lazo de control.

Durante el entrenamiento, la red crítico estima el valor $V(t)$ del estado, mientras que la función de recompensa calcula $r(t)$ a partir del comportamiento observado en el entorno. La recompensa, la estimación de valor y las trayectorias recopiladas se utilizan en la etapa de optimización de PPO para actualizar los parámetros de las redes actor y crítico. Una vez finalizado el entrenamiento, durante la inferencia se utiliza principalmente el ciclo formado por la medición del estado, la generación de la acción mediante la red actor y la respuesta del entorno.

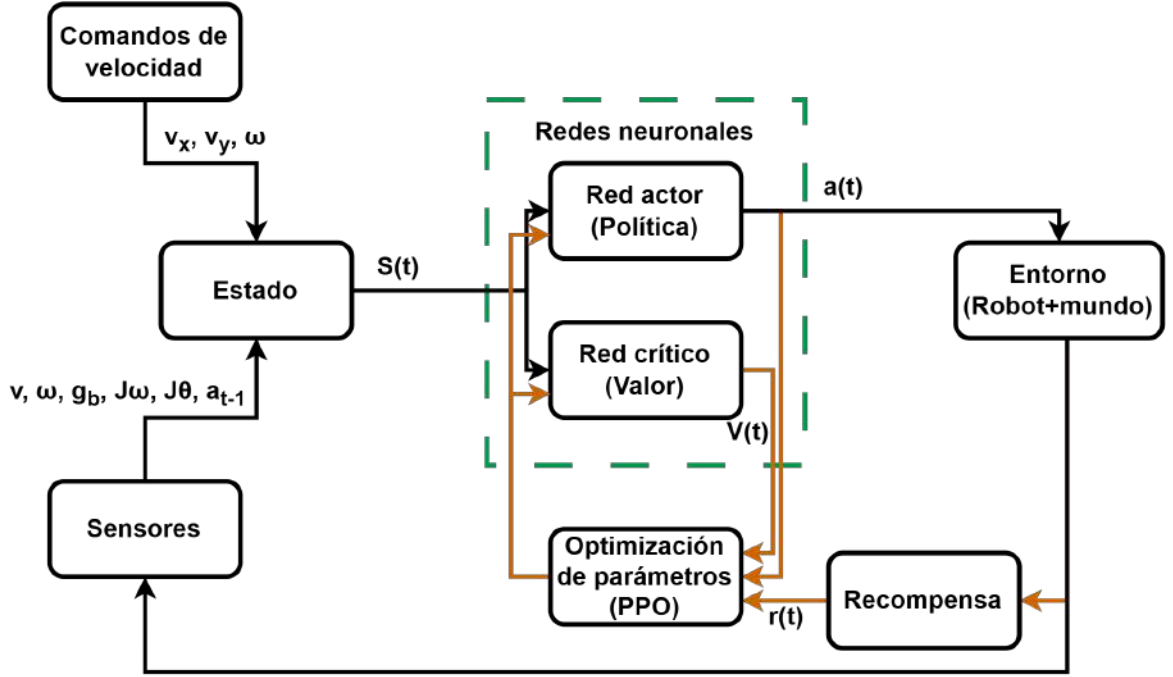


Figura 5.3: Arquitectura del ciclo de control y entrenamiento de la política PPO para el robot bípedo.

5.6 Diseño de la función de recompensa

En aprendizaje reforzado, la función de recompensa es un componente fundamental del sistema, esta se encarga de “calificar” el comportamiento del agente al asignar una puntuación (recompensa) que refleja el grado de éxito de una acción en un entorno determinado. A través de esta puntuación, el agente aprende a tomar decisiones que maximicen su recompensa acumulada a largo plazo.

El diseño de la función de recompensa se basa en el trabajo de Nikita Rudin et al. [14], reutilizando sus términos de recompensa y extendiéndolos con componentes adicionales como se muestra en 5.3.

$$r_t = r_t^{\text{lin}} + r_t^{\text{ang}} + r_t^{\text{step}} + r_t^{v_z} + r_t^{\omega_{xy}} + r_t^{\text{acc}} + r_t^{\Delta a} + r_t^{\tau} + r_t^{\text{fall}} + r_t^{\text{slide}} + r_t^{\text{flat}} + r_t^{\text{zmp}}. \quad (5.3)$$

Para facilitar la interpretación de la ecuación 5.3, los términos pueden analizarse de manera individual según el objetivo que persiguen. Como primer término, se presenta r_t^{lin} , el cual recompensa al agente por seguir los comandos de velocidad lineal y se define como

$$r_t^{\text{lin}} = 2 \cdot \exp\left(-\frac{e_v}{\sigma^2}\right) \quad (5.4)$$

donde e_v es el error entre la velocidad lineal comandada y la velocidad lineal medida en el robot, mientras que σ es un parámetro de escala que regula la sensibilidad de la recompensa ante dicho error. Esta expresión alcanza valores altos cuando el error es pequeño y disminuye exponencialmente conforme el robot se aleja de la referencia deseada. En consecuencia, este término incentiva al agente a reproducir los comandos de velocidad lineal establecidos para la tarea.

De manera análoga, el término r_t^{ang} recompensa al agente por seguir el comando de velocidad angular, particularmente alrededor del eje z del robot. Su propósito es favorecer que la orientación del movimiento coincida con la referencia establecida, lo cual resulta importante durante maniobras de giro o correcciones de trayectoria. Al igual que r_t^{lin} , este término alcanza valores altos cuando el error entre la velocidad angular deseada y la observada es pequeño.

Por su parte, el término r_t^{step} se asocia con el patrón de marcha del robot, ya que incentiva la ejecución adecuada de los pasos durante la locomoción. En este trabajo, dicho término se define como

$$r_t^{\text{step}} = 1.5 \cdot (\|\text{com_vel}\| > 0.1) \cdot \text{mín}(\text{humbral}, \text{soporte_simple} \cdot \text{mín}(\text{tiempo_izq}, \text{tiempo_derecha})) \quad (5.5)$$

donde $\|\text{com_vel}\| > 0.1$ actúa como una condición de activación que habilita esta recompensa únicamente cuando el robot recibe un comando de velocidad suficientemente grande. El término `soporte_simple` indica si el robot se encuentra en fase de apoyo simple, mientras que `tiempo_izq` y `tiempo_derecha` representan el tiempo de apoyo de cada pie. De esta manera, la recompensa aumenta cuando uno de los pies permanece en

el aire y el otro sostiene el cuerpo durante un intervalo compatible con la marcha. La función mínimo con humbral impone una cota superior para evitar que el término crezca de manera excesiva. En consecuencia, este componente favorece patrones de locomoción con pasos definidos.

Una vez descritos los términos asociados al seguimiento de referencia, es posible analizar con mayor detalle algunas de las penalizaciones empleadas para estabilizar y regularizar el movimiento del robot. En primer lugar, la penalización por velocidad vertical se expresa como

$$r_t^{v_z} = -2v_z^2 \quad (5.6)$$

donde v_z representa la velocidad lineal del robot en el eje vertical. Este término penaliza desplazamientos ascendentes o descendentes no deseados del cuerpo, favoreciendo una locomoción más estable y evitando oscilaciones excesivas del centro de masa.

De forma similar, la penalización por rotación alrededor de los ejes laterales se define como

$$r_t^{\omega_{xy}} = -0.05(\omega_x^2 + \omega_y^2) \quad (5.7)$$

donde ω_x y ω_y son las velocidades angulares alrededor de los ejes x e y , respectivamente. Este término desalienta inclinaciones y balanceos excesivos del torso, contribuyendo al mantenimiento de la postura y del equilibrio del robot durante la marcha.

En cuanto a la regularización del movimiento articular, se emplea la penalización

$$r_t^{\text{acc}} = -2.5 \times 10^{-7} \sum_i \ddot{q}_i^2 \quad (5.8)$$

donde $\ddot{q}_i$ representa la aceleración de la articulación i . Este componente reduce la apa-

rición de aceleraciones articulares excesivas, favoreciendo trayectorias más suaves y compatibles con un comportamiento físicamente plausible.

Asimismo, para evitar cambios abruptos entre acciones consecutivas, se considera el término

$$r_t^{\Delta a} = -0.01 \|a_t - a_{t-1}\|^2 \quad (5.9)$$

donde a_t es la acción aplicada en el instante actual y a_{t-1} corresponde a la acción del instante anterior. Esta penalización promueve transiciones suaves en la señal de control, evitando comandos bruscos que puedan comprometer la estabilidad o generar movimientos poco naturales.

Para favorecer movimientos eficientes y evitar esfuerzos excesivos en los actuadores, el término r_t^τ penaliza el par aplicado en las articulaciones

$$r_t^\tau = -2 \times 10^{-6} \cdot \sum_{i=1}^q \tau_i^2 \quad (5.10)$$

donde τ_i representa el torque aplicado en la articulación i y q es el número total de articulaciones consideradas.

Finalmente, se analizan los términos que inducen al robot a conservar la estabilidad durante la marcha. El término r_t^{fall} penaliza la caída del robot con un peso de -200 y provoca el reinicio del episodio. Para determinar esta condición, se detectan las colisiones entre el cuerpo del robot y el suelo, excluyendo deliberadamente los pies, ya que estos constituyen los elementos de contacto esperados durante la locomoción. Con ello, el agente aprende a evitar configuraciones en las que el torso o cualquier otro segmento no diseñado para el apoyo entra en contacto con la superficie.

Por otra parte, el término r_t^{slide} penaliza el deslizamiento de los pies sobre la superficie de apoyo y puede expresarse como

$$r_t^{\text{slide}} = -0.25 \cdot ((v_d c_d) + (v_i c_i)) \quad (5.11)$$

donde v_d y v_i representan la velocidad lineal de los pies derecho e izquierdo, respectivamente, mientras que c_d y c_i son variables indicadoras de contacto con el suelo para cada pie. De esta forma, cuando un pie mantiene contacto con la superficie y al mismo tiempo presenta velocidad tangencial, el término de penalización aumenta, lo que indica la presencia de deslizamiento. Esta formulación desalienta apoyos inestables.

Asimismo, el término r_t^{flat} penaliza la inclinación de la base del robot con respecto a la dirección vertical. Para calcularlo, se utilizan las componentes x e y del vector de gravedad proyectado en el sistema de referencia de la base, de acuerdo con

$$r_t^{\text{flat}} = -(g_{b,x}^2 + g_{b,y}^2) = -\|\mathbf{g}_{b,xy}\|_2^2, \quad (5.12)$$

donde $g_{b,x}$ y $g_{b,y}$ son las componentes horizontales del vector de gravedad proyectado $\mathbf{g}_b$. Cuando la base permanece nivelada y el torso conserva una orientación vertical, ambas componentes se aproximan a cero, por lo que la penalización es mínima. En cambio, su valor aumenta conforme el robot se inclina alrededor de los ejes x o y . Por tanto, este término favorece una orientación estable y aproximadamente vertical del torso durante la locomoción.

Para la penalización r_t^{zmp} , se calcula la posición del ZMP en los ejes x e y , así como la posición del centro del polígono de soporte, y se penaliza la desviación entre ambos puntos. Una forma compacta de expresar este término es

$$r_t^{\text{zmp}} = -1 \times 10^{-5} \|\mathbf{p}_{\text{zmp}} - \mathbf{p}_{\text{cop}}\|^2 \quad (5.13)$$

donde $\mathbf{p}_{\text{zmp}} \in \mathbb{R}^2$ representa la posición del ZMP en el plano de apoyo y $\mathbf{p}_{\text{cop}} \in \mathbb{R}^2$ corresponde a la posición del centro del polígono de soporte. De este modo, cuanto mayor sea la distancia entre el ZMP y el centro de dicha región, mayor será la penalización. Este término favorece que el ZMP permanezca cercano a una zona central del área de soporte, lo cual contribuye a mantener una condición de equilibrio más robusta durante

la marcha.

En conjunto, estas penalizaciones orientan el proceso de aprendizaje lejos de soluciones inestables. Así, la suma de todos los términos de la ecuación 5.3 no solo busca que el robot siga referencias de velocidad, sino que además lo haga con estabilidad y suavidad.

El código 5.1 presenta la clase de recompensas heredada, en la cual se incorporan los términos utilizados para el entrenamiento del robot. Entre ellos, la recompensa asociada al ZMP constituye la aportación desarrollada en esta investigación, al añadirse como un término específico orientado a mejorar la estabilidad del robot durante la marcha.

```

1 class BipedoRewards(RewardsCfg):
2
3     termination_penalty = RewTerm(func=mdp.is_terminated, weight=-200.0)
4
5     track_lin_vel_xy_exp = RewTerm(
6         func=mdp.track_lin_vel_xy_yaw_frame_exp,
7         weight=2.0,
8         params={"command_name": "base_velocity", "std": 0.5},
9     )
10
11     track_ang_vel_z_exp = RewTerm(
12         func=mdp.track_ang_vel_z_world_exp,
13         weight=0.7,
14         params={"command_name": "base_velocity", "std": 0.5},
15     )
16
17     feet_air_time_positive_biped = RewTerm(
18         func=mdp.feet_air_time_positive_biped,
19         weight=1.5,
20         params={
21             "sensor_cfg": SceneEntityCfg(
22                 "contact_forces",
23                 body_names=["link_6", "link_12"]
24             ),
25             "command_name": "base_velocity",
26             "threshold": 0.4,
27         },
28     )
29
30     feet_slide = RewTerm(

```

```

31     func = mdp.feet_slide,
32     weight = -0.25,
33     params={
34         "sensor_cfg": SceneEntityCfg("contact_forces", body_names=["link_6", "link_12"]),
35         "asset_cfg": SceneEntityCfg("robot", body_names=["link_6", "link_12"])
36     }
37 )
38
39 zmp_cop_error = RewTerm(
40     func=com_zmp_reward,
41     weight = -1e-5,
42     params={
43         "sensor_cfg": SceneEntityCfg("zmp_contact_sensor", body_names=["link_6", "link_12"]),
44         "asset_cfg": SceneEntityCfg("robot")
45     }
46 )
47
48 self.rewards.flat_orientation_l2.weight = -1.0

```

Listing 5.1: Recompensas heredadas

5.7 Configuración experimental

5.7.1 Condiciones iniciales del entorno durante el entrenamiento.

Una vez definidas la función de recompensa, el espacio de estados y el espacio de acciones, se procedió al entrenamiento del agente. Para ello, fue necesario establecer las condiciones iniciales del entorno de simulación y del robot. El entorno de entrenamiento consiste en un suelo plano. Para el robot, se utilizó un coeficiente de fricción estática de 0.8 y dinámica de 0.6, correspondiente a una condición intermedia de contacto. Esta elección permite mantener un equilibrio entre estabilidad en el apoyo y facilidad de deslizamiento durante la interacción con el suelo, de modo que la dinámica del sistema dependa principalmente del control articular. Para el suelo, se definió un coeficiente de fricción estática y dinámica de 1.0.

Adicionalmente, se definió una postura inicial del robot mediante la asignación de una configuración articular específica. Los valores iniciales de las articulaciones, ordenados de R1 a R12, son: $[0.243, 0.0, -1.104, 0.0, 0.861, 0.0, -0.243, 0.0, 1.104, 0.0, -0.861, 0.0]$ rad. Esta configuración corresponde a la postura inicial utilizada durante el entrenamiento, la cual se muestra en la figura 5.4. Dicha postura ubica el centro de masa del robot cerca del centro del polígono de soporte, con el propósito de que el entrenamiento inicie desde una condición de estabilidad favorable.

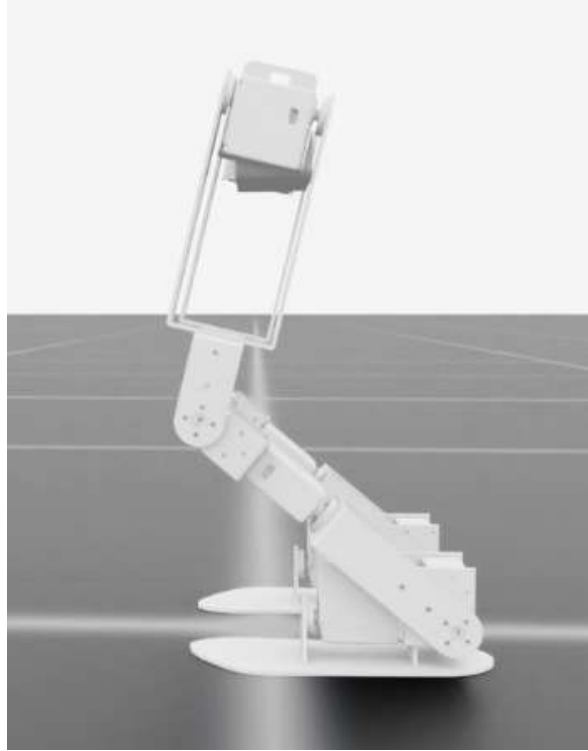


Figura 5.4: Postura inicial del robot durante el entrenamiento.

Con el fin de favorecer la generalización de la política aprendida, la posición inicial del robot también se aleatoriza en los ejes x e y dentro de un rango de 0.5 m. De igual forma, la orientación alrededor del eje z se aleatoriza en el intervalo $[-\pi, \pi]$ rad. Asimismo, las articulaciones se escalan de manera aleatoria entre 0.5 y 1.5 con respecto a su configuración inicial, de modo que el agente enfrente distintas condiciones articulares al inicio de cada episodio.

Para analizar el efecto del paralelismo, se evaluaron tres configuraciones de entrenamiento correspondientes a 500, 1000 y 4000 entornos simulados ejecutados simultáneamente (en la figura 5.5 se muestra un ejemplo de entrenamiento utilizando 500

robots). En todos los casos se mantuvieron fijos la arquitectura de la política, la función de recompensa, el número de pasos por entorno y los principales hiperparámetros de PPO. Sea $\mathcal{D}_i$ el conjunto de trayectorias recolectadas en el entorno i , el lote total de entrenamiento se define como:

$$\mathcal{D} = \bigcup_{i=1}^{N_{\text{env}}} \mathcal{D}_i, \quad (5.14)$$

donde N_{env} corresponde al número de entornos paralelos considerados en cada configuración. Para cada entorno se recolectaron 24 pasos de simulación antes de realizar una actualización, lo que produce 12 000, 24 000 y 96 000 transiciones por iteración para 500, 1000 y 4000 entornos, respectivamente.

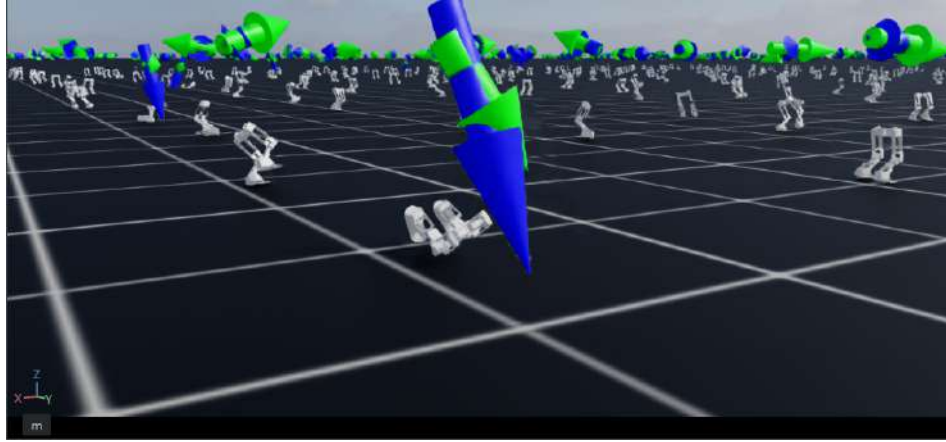


Figura 5.5: Entrenamiento en paralelo utilizando Isaac Sim.

El entrenamiento se ejecutó durante un máximo de 1000 iteraciones, almacenando un *checkpoint* cada 50 iteraciones. Los principales hiperparámetros utilizados fueron: número de pasos por entorno igual a 24, $\epsilon = 0.2$, coeficiente de entropía de 0.01, cinco épocas de aprendizaje por actualización, cuatro mini-lotes, tasa de aprendizaje de 1×10^{-3} , factor de descuento $\gamma = 0.99$, parámetro GAE $\lambda = 0.95$, divergencia KL deseada de 0.01 y norma máxima del gradiente de 1.0. Durante el entrenamiento, el comando de velocidad lineal se muestrea de una distribución uniforme en el intervalo $[0, 1]$ m/s y se mantiene constante a lo largo de cada episodio, actualizándose únicamente al reinicio del entorno. Por su parte, el comando de velocidad angular se fija en cero con el objetivo de favorecer trayectorias rectilíneas y aislar el análisis del comportamiento locomotor en avance.

Los experimentos se realizaron en una sola máquina equipada con un procesador Intel Core i9-13950HX de 13^a generación, 64 GB de memoria RAM DDR5 y una GPU NVIDIA RTX 4080 Laptop GPU con 12 GB de memoria VRAM GDDR6. El entorno de simulación empleado fue Isaac Sim 5.1.0, mientras que la implementación del entrenamiento se desarrolló sobre Isaac Lab v2.3.2. Todos los experimentos se ejecutaron en una sola GPU, considerando configuraciones de 500, 1000 y 4000 entornos paralelos. Los tiempos totales de entrenamiento observados fueron de aproximadamente 6 min 32 s, 6 min 58 s y 10 min 28 s para 500, 1000 y 4000 entornos, respectivamente.

5.7.2 Etapa de inferencia

La fase de inferencia se dividió en tres etapas: bipedestación en triple flexión, transición hacia la postura de marcha y ejecución de la marcha. La triple flexión considera la flexión simultánea de cadera, rodilla y tobillo.

La posición inicial del robot se muestra en la figura 5.6. Esta postura se tomó como configuración de reposo del robot. Posteriormente, la transición consistió en llevar al robot desde la posición de triple flexión hasta la postura inicial de marcha. Para ello, se aplicó control de velocidad en los pies mediante el Jacobiano inverso, considerando los parámetros de Denavit-Hartenberg y la cinemática inversa obtenidos en el capítulo 4.



Figura 5.6: Postura de triple flexión para la inferencia.

Una vez alcanzada la postura inicial de marcha, se cargó el modelo entrenado en Isaac Lab para ejecutar la política aprendida durante la etapa de inferencia. En esta fase, el agente recibe las observaciones del entorno en cada instante de simulación, conformadas por las variables definidas previamente en el espacio de estados, tales como la velocidad lineal y angular de la cadera, la proyección de la gravedad, las posiciones y velocidades articulares, el comando de velocidad y la acción aplicada en el instante anterior.

A partir de estas observaciones, la red neuronal calcula el vector de acciones correspondiente, el cual representa las posiciones deseadas de las articulaciones del robot, permitiendo generar los movimientos necesarios para el desplazamiento bípedo. De esta manera, la política entrenada actúa como controlador de bajo nivel.

Durante la marcha, este proceso se ejecuta de forma iterativa: el simulador actualiza el estado dinámico del robot, se construye el vector de observaciones, la política genera una nueva acción y dicha acción se envía al sistema de control articular. Este ciclo cerrado permite que el robot ajuste continuamente el movimiento de sus piernas en función de su estado actual, con el objetivo de mantener el equilibrio y avanzar de acuerdo con el comando de velocidad establecido.

Capítulo 6

Resultados

Una vez concluido el entrenamiento, se realizó una evaluación comparativa entre las configuraciones de 500, 1000 y 4000 entornos paralelos. El incremento en el número de entornos paralelos introduce una diferencia significativa en la cantidad total de experiencia recolectada. En consecuencia, los resultados obtenidos reflejan no solo el efecto del paralelismo en la estabilidad del entrenamiento, sino también el impacto del volumen de datos disponible para la actualización de la política.

En este contexto, se analizaron los errores de seguimiento de comandos de velocidad lineal y angular, las principales componentes de la función de recompensa, así como métricas de optimización del algoritmo PPO, incluyendo la pérdida de la función de valor y la *surrogate loss* del actor.

Adicionalmente, la capacidad de cada política para generar locomoción sostenida en simulación fue evaluada mediante la interpretación conjunta de las métricas de recompensa y error. Finalmente, la política obtenida con la configuración que presentó el mejor comportamiento locomotor fue analizada en fase de inferencia mediante episodios sin actualización de parámetros.

El enfoque inicial del proyecto consistía en realizar el entrenamiento directamente en Gazebo+ROS2, ya que este era el simulador con el que se había validado el modelo URDF (Sección 3.3). Sin embargo, durante la fase experimental se identificaron limitaciones inherentes de Gazebo para este tipo de entrenamiento masivo:

- **Ejecución de un solo agente:** Gazebo no ofrece paralelización nativa de entornos, por lo que solo es posible ejecutar una simulación a la vez. Esto implica que cada muestra de experiencia se obtiene de forma secuencial, lo que hace inviable recolectar cantidades significativas de datos en tiempos razonables.
- **Inestabilidad del simulador:** De manera aleatoria y sin un patrón reproducible, Gazebo presentaba cierres inesperados (crashes) asociados a errores internos de

plugins del motor de física, lo que interrumpía los procesos de entrenamiento largos y dificultaba la recolección continua de experiencia.

Como resultado, el entrenamiento en Gazebo resultó ineficiente: se realizaron mediciones con un solo agente que arrojaron aproximadamente 1×10^5 transiciones en 12 horas de ejecución continua. La combinación de la baja tasa de muestreo y la inestabilidad del entorno impedía que el algoritmo PPO convergiera dentro de tiempos prácticos.

La Tabla 6.1 resume las principales características computacionales de las tres configuraciones evaluadas utilizando Isaac Sim. Aunque la configuración con 4000 entornos presenta el mayor tiempo total de entrenamiento, también procesa la mayor cantidad de experiencia por iteración, lo que se traduce en el mayor volumen total de transiciones.

Tabla 6.1: Comparación de las configuraciones de entrenamiento paralelo.

Entornos	Tiempo total	Tiempo/iteración	Transiciones/iteración	Transiciones totales
500	6 min 32 s	0.392 s	12 000	12×10^6
1000	6 min 58 s	0.418 s	24 000	24×10^6
4000	10 min 28 s	0.628 s	96 000	96×10^6

La Figura 6.1 muestra la evolución del error de seguimiento de velocidad lineal, mientras que la Figura 6.2 presenta la evolución del error de seguimiento de velocidad angular durante el entrenamiento. En el caso de la velocidad lineal, la configuración con 4000 entornos exhibe la mejor convergencia y el menor error final, lo que sugiere un aprendizaje más eficaz del comportamiento locomotor asociado al desplazamiento sostenido. En contraste, las configuraciones con 500 y 1000 entornos mantienen errores lineales más elevados y una mayor variabilidad, lo que indica una menor estabilidad en el proceso de aprendizaje.

En cuanto al error angular, las configuraciones con 500 y 1000 entornos alcanzan valores finales menores que la configuración con 4000 entornos. No obstante, este resultado no debe interpretarse de manera aislada como una mejora de la locomoción. Durante las pruebas con 500 y 1000 entornos se observó que el robot permanece durante intervalos prolongados en estados con movimiento reducido o permaneció estático, por lo que la velocidad angular tiende a permanecer cercana a cero. En consecuencia, el error angular puede subestimarse de manera artificial. Por el contrario, cuando la política entrenada con 4000 entornos logra una locomoción sostenida, surgen oscilaciones

laterales naturales del torso y la cadera que incrementan la velocidad angular observada y, por ende, el error angular con respecto al comando.

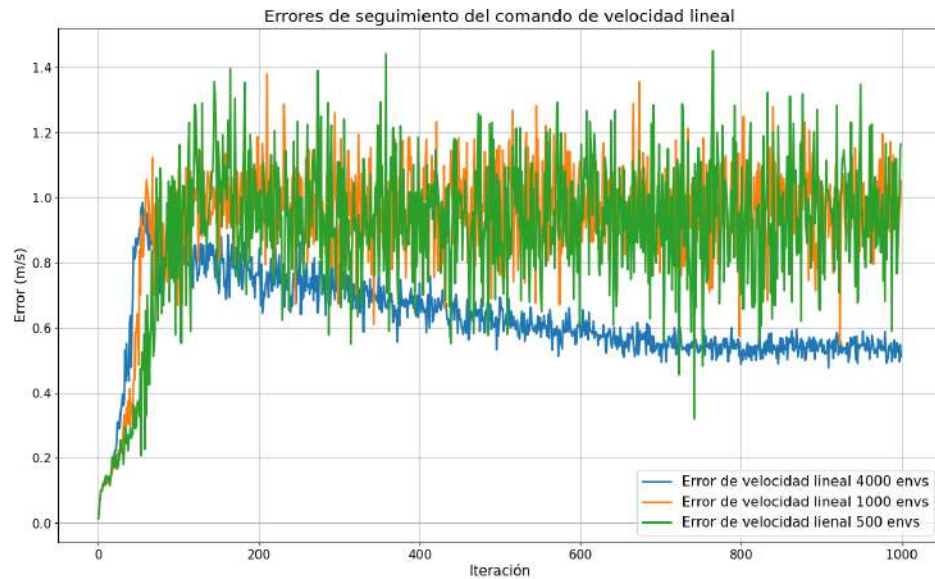


Figura 6.1: Errores de seguimiento al comando de velocidad lineal.

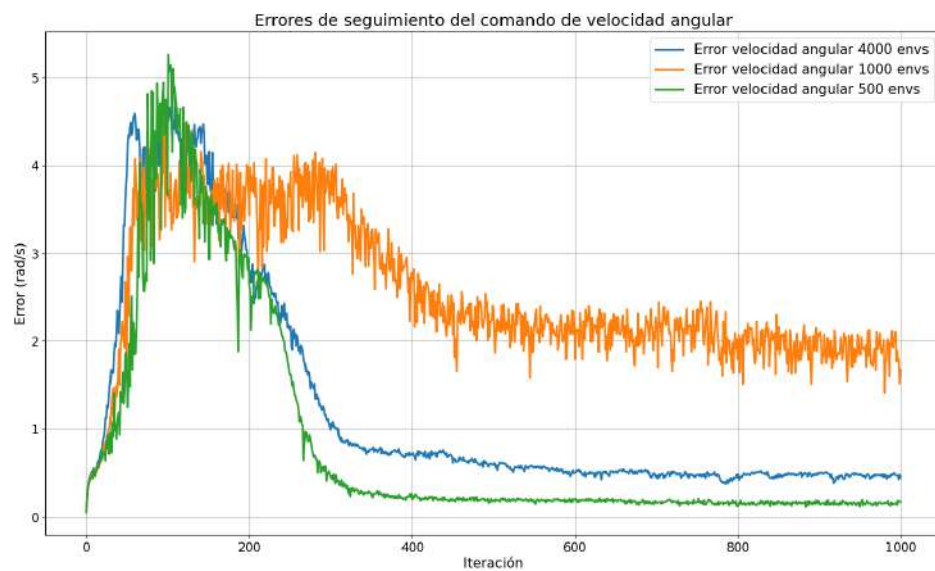


Figura 6.2: Errores de seguimiento al comando de velocidad angular.

La Figura 6.3 presenta la recompensa asociada al seguimiento de velocidad lineal.

En esta gráfica se observa que la configuración con 500 entornos obtiene la menor recompensa promedio y, al mismo tiempo, exhibe el comportamiento más inestable. En este caso, la recompensa parece estabilizarse alrededor de 0.7, sin evidenciar una tendencia clara de mejora. Para la configuración con 1000 entornos se observa un comportamiento similar; sin embargo, el valor de estabilización se sitúa alrededor de 0.9 y se aprecia una tendencia descendente en el desempeño. No obstante, esta tendencia no implica necesariamente un deterioro definitivo de la política, ya que el comportamiento podría modificarse al incrementar el número de iteraciones. En contraste, la configuración con 4000 entornos presenta el mejor desempeño global y una tendencia hacia la estabilización, con indicios de que aún podría mejorar ligeramente.

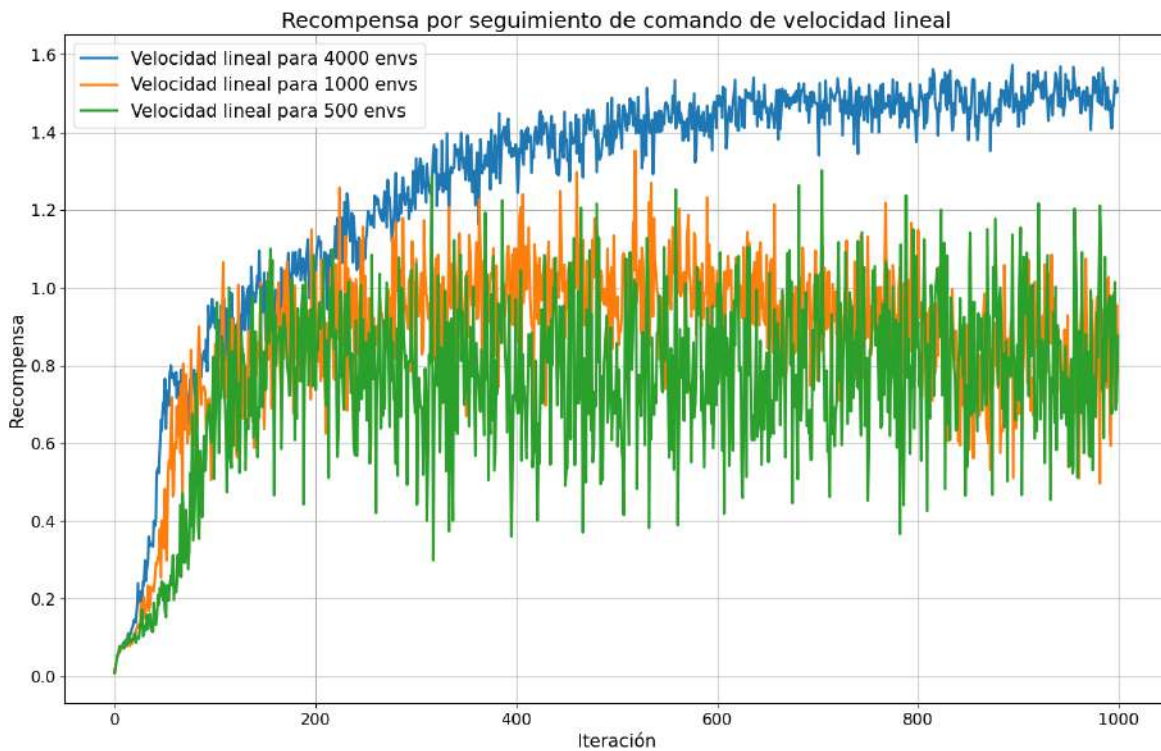


Figura 6.3: Valores de recompensa por seguimiento de velocidad lineal.

En la Figura 6.4 se muestran las recompensas asociadas al seguimiento de velocidad angular. Para este experimento, el comando de velocidad angular se fija en cero, por lo que el objetivo de control consiste en mantener una orientación constante durante la locomoción. En este contexto, valores bajos del error angular no necesariamente implican un mejor desempeño, ya que pueden corresponder a estados en los que el robot no genera locomoción efectiva. Esta situación se observa en las configuraciones

con 500 y 1000 entornos, donde el sistema permanece con frecuencia en estados cercanos al reposo, produciendo errores angularmente pequeños pero poco representativos del comportamiento locomotor deseado.

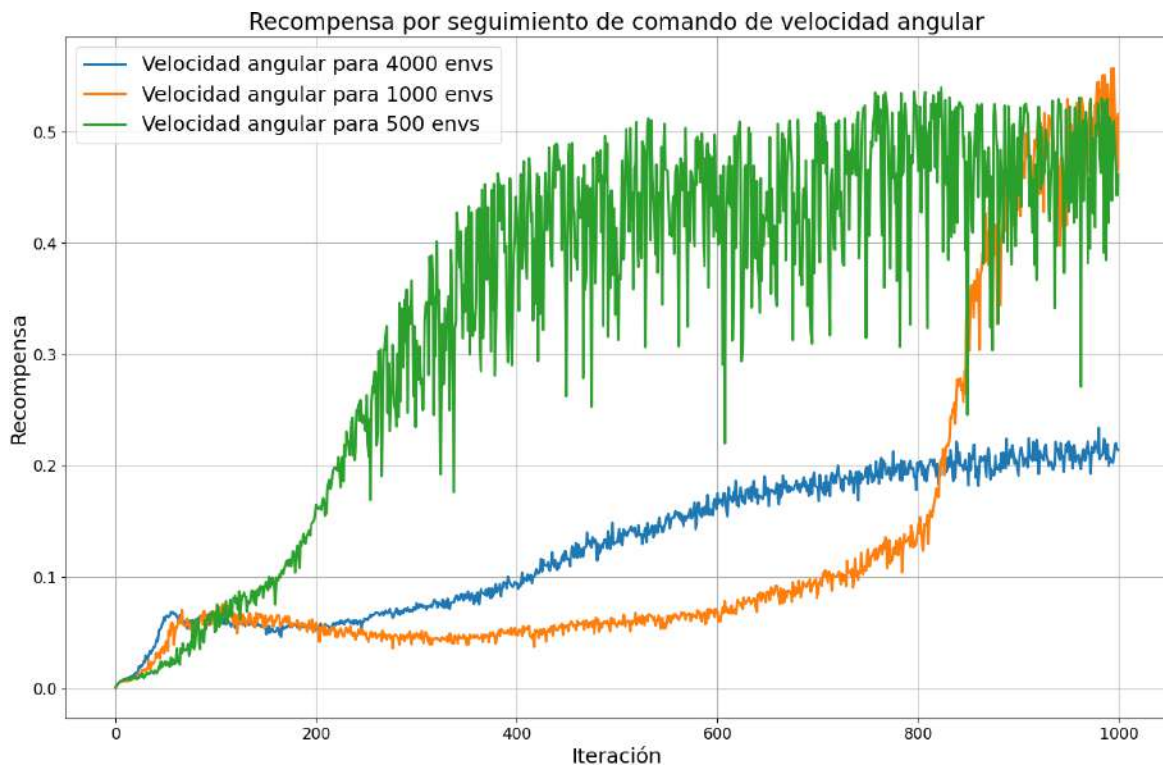


Figura 6.4: Recompensa por seguimiento de velocidad angular.

En la Figura 6.5 se muestran las penalizaciones asociadas al alejamiento del ZMP con respecto al centro del polígono de soporte. Para la configuración con 500 entornos, la penalización se estabiliza alrededor de -0.005 ; para 1000 entornos, alrededor de -0.01 ; y para 4000 entornos, alcanza un valor promedio cercano a -0.04 . La mayor magnitud observada en esta última configuración se relaciona con la generación de una marcha sostenida, durante la cual el ZMP se desplaza continuamente dentro del polígono de soporte. En contraste, las políticas con menor actividad locomotora presentan variaciones más reducidas del ZMP y, por consiguiente, penalizaciones de menor magnitud. Por esta razón, este término no debe interpretarse como un indicador directo de menor estabilidad, sino conjuntamente con la distancia recorrida y las demás métricas de desempeño.

La Figura 6.6 muestra la evolución de la pérdida asociada a la función de valor. Se observa que la configuración con 4000 entornos presenta no solo una menor variabilidad, sino también un comportamiento más suave a lo largo del entrenamiento. En contraste, la configuración con 500 entornos exhibe una mayor dispersión y la presencia de picos pronunciados, lo que sugiere una estimación menos estable del valor esperado.

La configuración con 1000 entornos presenta un comportamiento intermedio entre ambos extremos. Este resultado es consistente con el hecho de que un mayor número de entornos paralelos permite recolectar muestras con mayor diversidad en cada iteración, lo que reduce la varianza en la estimación del retorno y favorece un entrenamiento más estable del crítico. En este sentido, la mejora observada no debe interpretarse únicamente en términos de magnitud de la pérdida, sino principalmente en términos de estabilidad y consistencia del proceso de aprendizaje.

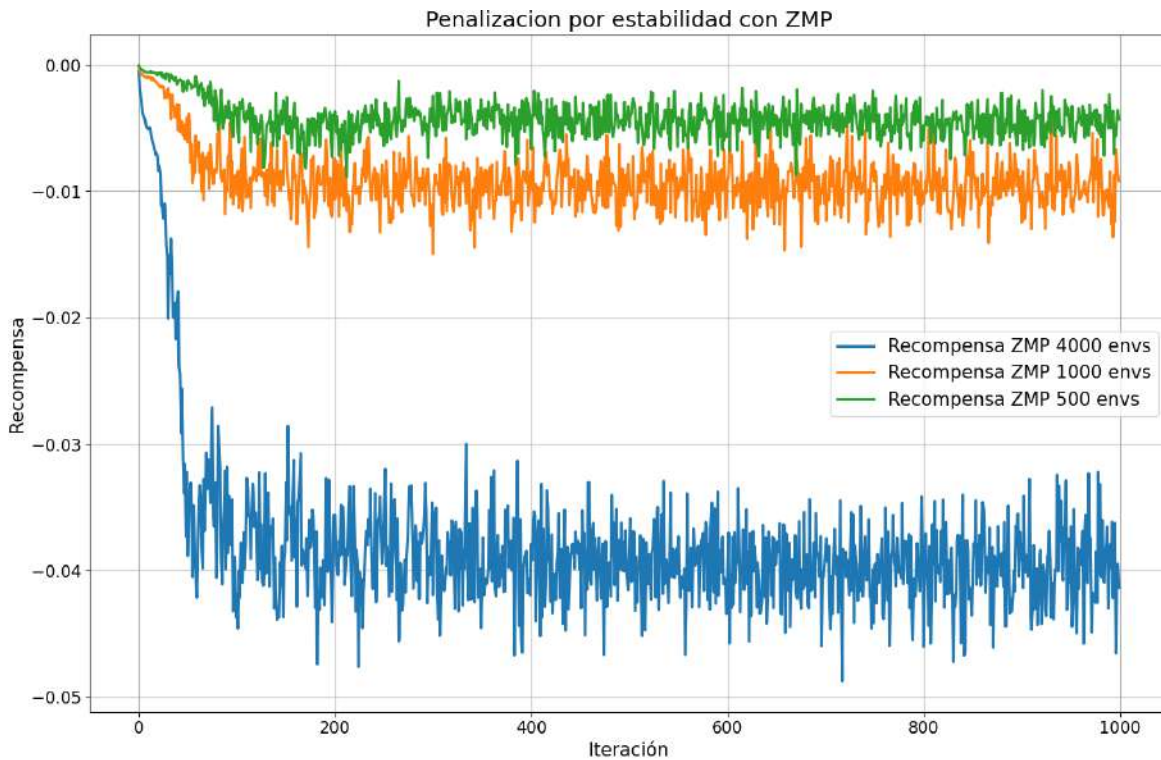


Figura 6.5: Penalización por el alejamiento del ZMP respecto al centro del polígono de soporte.

La Figura 6.7 muestra la evolución del objetivo sustituto del actor para las tres

configuraciones. En todos los casos se observa el comportamiento ruidoso característico de PPO, debido a la dependencia directa del objetivo respecto a estimaciones del *advantage* y a la naturaleza *on-policy* del algoritmo.

Adicionalmente, el uso de funciones de recorte (*clipping*) en PPO introduce puntos no diferenciables en la superficie de optimización, lo que contribuye a la variabilidad observada en esta métrica. A pesar de ello, se aprecia una tendencia general hacia la estabilización conforme avanza el entrenamiento, particularmente en la configuración con 4000 entornos.

El objetivo sustituto no debe interpretarse como una función de pérdida convencional, en este caso el gradiente estima la actualización de la política. En este contexto, su evolución proporciona información sobre la estabilidad de las actualizaciones y el régimen en el que opera el algoritmo, constituyendo así un indicador complementario del proceso de optimización de la política.

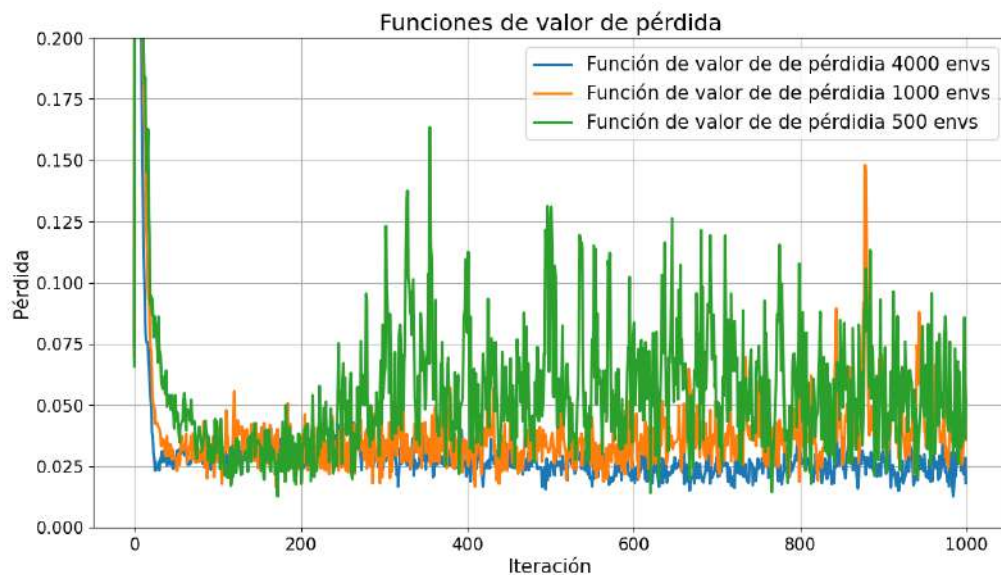


Figura 6.6: Valores de la función de pérdida.

La secuencia completa de inferencia, desde la postura inicial de triple flexión hasta la transición y posterior ejecución de la marcha, se muestra en la figura 6.8. Esta representación permite visualizar la evolución postural del robot y la continuidad del

proceso de control durante las distintas fases del movimiento.

En conjunto, los resultados evidencian que la interpretación aislada de métricas individuales puede conducir a conclusiones erróneas sobre el desempeño del sistema. A pesar de que algunas configuraciones presentan menores errores o mayores recompensas en componentes específicas, únicamente la configuración con 4000 entornos logra generar locomoción sostenida de manera consistente.

Este comportamiento sugiere que el mayor volumen y diversidad de experiencia recolectada durante el entrenamiento favorece la exploración del espacio de estados y permite la adquisición de políticas más robustas, capaces de reproducir dinámicas de marcha. En particular, el incremento en el número de entornos paralelos incrementa la diversidad de los estados presentes en cada actualización, lo que contribuye a una estimación más estable de la función de valor.

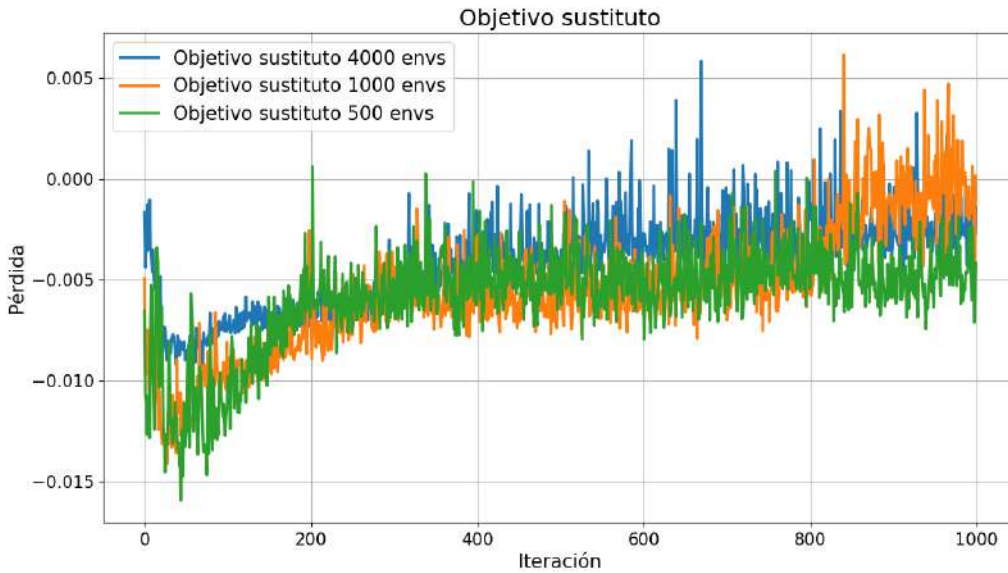


Figura 6.7: Valores del objetivo sustituto.

Con el fin de fortalecer la evidencia experimental y evaluar la robustez de las tendencias observadas, se realizó un análisis estadístico adicional utilizando cinco semillas aleatorias distintas para cada configuración de entrenamiento paralelo: 500, 1000 y 4000 entornos. El conjunto de semillas evaluadas fue $\{0, 50, 500, 1000, 4000\}$. Para cada eje-

cución se analizaron tres indicadores de desempeño: error de velocidad lineal, error de velocidad angular y distancia recorrida.

Las Figuras 6.9, 6.10 y 6.11 presentan los resultados estadísticos correspondientes a las tres métricas evaluadas. Las curvas muestran el comportamiento promedio y la varianza asociada para cada configuración. Las gráficas reportan el comportamiento promedio entre semillas junto con la varianza asociada, lo que permite analizar tanto la tendencia central como la dispersión de los resultados. Los resultados muestran que el patrón general de desempeño se mantiene consistente entre semillas. En particular, la configuración con 4000 entornos continúa presentando el compromiso más favorable, al combinar un menor error de velocidad lineal con una mayor distancia recorrida, mientras que la interpretación del error de velocidad angular sigue condicionada por la aparición efectiva de locomoción sostenida.

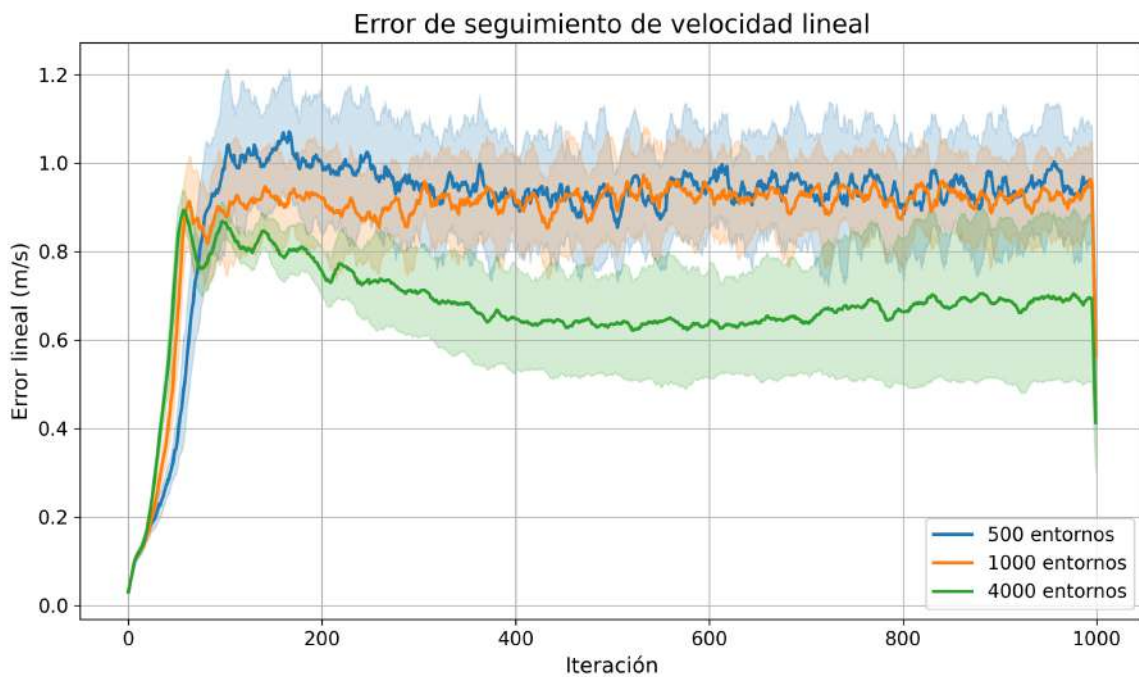


Figura 6.9: Comparación estadística del error de velocidad lineal entre semillas para 500, 1000 y 4000 entornos paralelos.

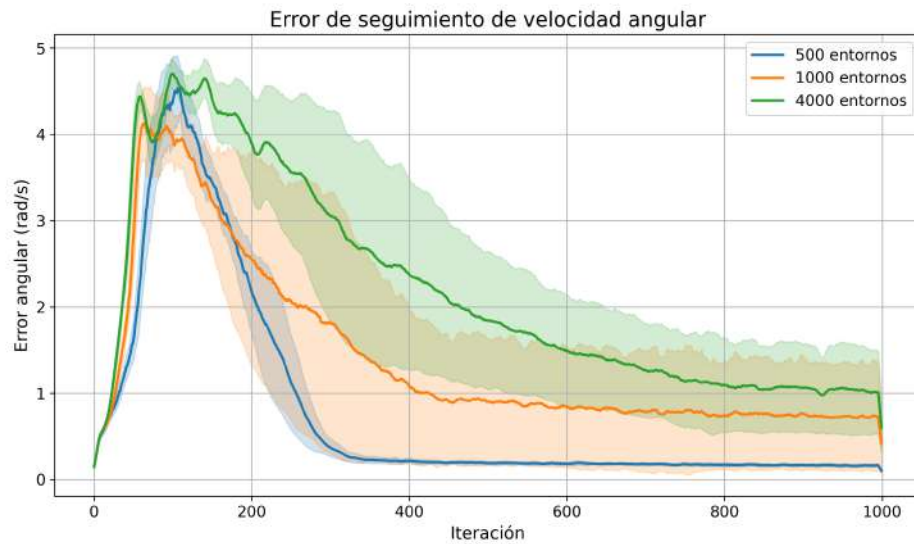


Figura 6.10: Comparación estadística del error de velocidad angular entre semillas para 500, 1000 y 4000 entornos paralelos.

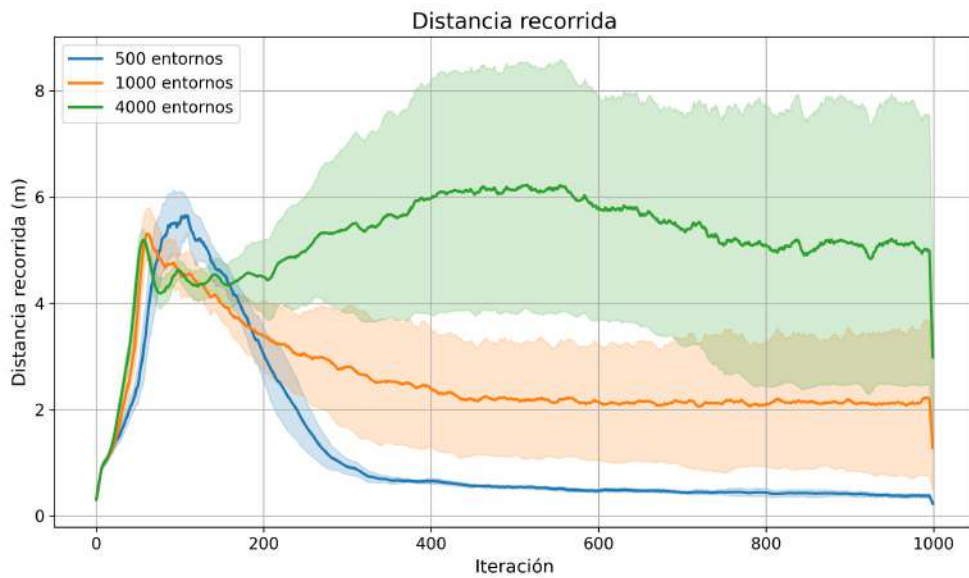


Figura 6.11: Comparación estadística de la distancia recorrida entre semillas para 500, 1000 y 4000 entornos paralelos.

Debido a que el entrenamiento se orientó al desplazamiento hacia adelante, el análisis

del seguimiento lineal se concentró en la componente correspondiente al eje x . Para cuantificar este comportamiento se utilizó el error absoluto instantáneo

$$e_x(t) = |v_{x,\text{cmd}}(t) - v_x(t)|, \quad (6.1)$$

donde $v_{x,\text{cmd}}(t)$ es la velocidad comandada y $v_x(t)$ es la velocidad medida del robot en el instante t . Para obtener los valores reportados, se consideraron las últimas 100 iteraciones de entrenamiento de cada una de las cinco semillas evaluadas. El error promedio de cada configuración se calculó mediante

$$\bar{e}_x = \frac{1}{N_s N_I} \sum_{s=1}^{N_s} \sum_{k=K-N_I+1}^K e_x^{(s)}(k), \quad (6.2)$$

donde $N_s = 5$ es el número de semillas, $N_I = 100$ es el número de iteraciones consideradas por semilla, $K = 1000$ es la última iteración del entrenamiento y $e_x^{(s)}(k)$ es el error de velocidad sobre el eje x registrado en la iteración k para la semilla s . Por tanto, cada valor reportado se obtuvo a partir de 500 observaciones correspondientes a la etapa final de los entrenamientos. Esta métrica corresponde al error absoluto medio y conserva las mismas unidades que la velocidad, por lo que permite interpretar directamente la diferencia promedio entre el comando y la respuesta del robot. Además, al no dividir el error entre la velocidad comandada, evita valores indeterminados o artificialmente elevados cuando el comando se aproxima a cero.

Los errores promedio obtenidos fueron de 1.00 m/s para 500 entornos, 0.97 m/s para 1000 entornos y 0.71 m/s para 4000 entornos. Para comparar la mejora entre configuraciones se empleó la reducción relativa del error, definida como

$$R = \frac{\bar{e}_{x,\text{ref}} - \bar{e}_{x,4000}}{\bar{e}_{x,\text{ref}}} \times 100, \quad (6.3)$$

donde $\bar{e}_{x,\text{ref}}$ es el error promedio de la configuración utilizada como referencia. Esta expresión corresponde al cambio porcentual relativo entre dos valores y permite cuan-

tificar cuánto disminuyó el error de la configuración con 4000 entornos respecto a las configuraciones restantes. Al utilizar como referencia los experimentos con 500 y 1000 entornos, respectivamente, se obtiene

$$R_{500} = \frac{1.00 - 0.71}{1.00} \times 100 = 29.0 \%, \quad R_{1000} = \frac{0.97 - 0.71}{0.97} \times 100 = 26.8 \%. \quad (6.4)$$

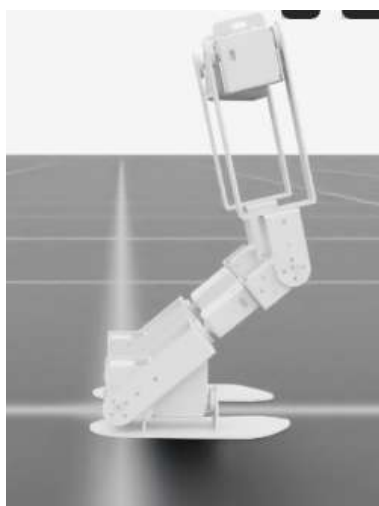
Estos porcentajes representan la reducción del error respecto a las configuraciones de referencia y no un porcentaje de error absoluto. Aunque el valor de 0.71 m/s no puede considerarse un error bajo, los resultados muestran una mejora relativa acompañada de una mayor distancia recorrida y de locomoción sostenida. En consecuencia, los resultados respaldan parcialmente la hipótesis: se consiguió generar locomoción bípeda estable y reducir el error de seguimiento, pero no se alcanzó un error absoluto bajo. El error restante podría reducirse en trabajos posteriores mediante un mayor número de iteraciones de entrenamiento, el ajuste de los términos y pesos de la función de recompensa y la optimización sistemática de los hiperparámetros de PPO.

Este análisis entre distintas semillas aleatorias indica que el comportamiento reportado previamente no corresponde a un artefacto asociado con la selección de la semilla inicial, incluida la semilla utilizada originalmente en los experimentos principales (semilla: 42). Por el contrario, la mejora observada se asocia de manera consistente con el nivel de paralelismo empleado durante el entrenamiento. Por lo tanto, los resultados proporcionan evidencia adicional de que el desempeño locomotor superior obtenido con 4000 entornos constituye un efecto sistemático del entrenamiento paralelo a gran escala, y no una coincidencia inducida por una inicialización aleatoria particular.

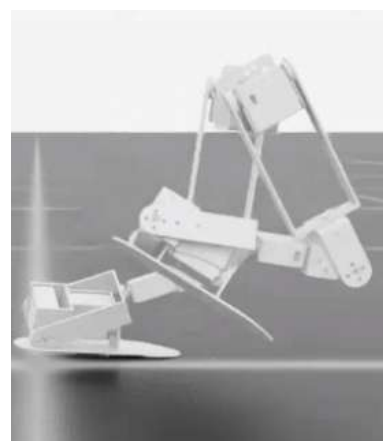
En términos generales, los resultados comparativos muestran que la configuración con 4000 entornos ofrece el mejor compromiso entre estabilidad del aprendizaje, reducción del error lineal y capacidad para generar una marcha sostenida. Aunque las configuraciones con 500 y 1000 entornos requieren un menor tiempo total de entrenamiento, la configuración con 500 entornos conduce a una política estacionaria, en la que el agente aprende a permanecer inmóvil. Por su parte, la configuración con 1000 entornos permite generar locomoción, mientras que la configuración con 4000 entornos produce el mejor comportamiento de marcha.



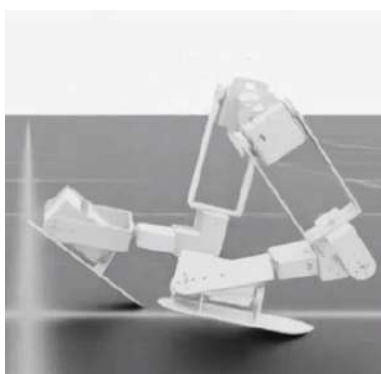
(a) Postura de triple flexión.



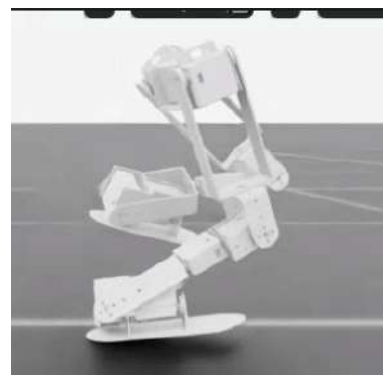
(b) Bipedestación estable.



(c) Fase de soporte simple o balanceo.



(d) Fase de doble soporte.



(e) Fase de soporte simple o balanceo.

Figura 6.8: Secuencia completa de inferencia desde la postura de triple flexión hasta la ejecución de la marcha.

Capítulo 7

Conclusiones

Este trabajo mostró que es posible obtener locomoción bípeda sostenida y estable mediante la integración de criterios clásicos de estabilidad (ZMP) con aprendizaje por refuerzo profundo (PPO), sin necesidad de un prototipo físico. A continuación se presentan las conclusiones principales organizadas por los objetivos planteados en esta investigación, seguidas de las aportaciones científicas y el trabajo futuro.

7.1 Conclusiones por objetivo

Evaluación de la hipótesis Los resultados respaldan parcialmente la hipótesis planteada. La política entrenada con 4000 entornos logró desplazamiento sostenido con alternancia de apoyos y conservó condiciones de estabilidad durante la marcha. En cuanto al seguimiento de la velocidad lineal sobre el eje x , los errores promedio, calculados a partir de las últimas 100 iteraciones de cada una de las cinco semillas, fueron de 1.00 m/s, 0.97 m/s y 0.71 m/s para las configuraciones con 500, 1000 y 4000 entornos, respectivamente. Por tanto, la configuración con 4000 entornos redujo el error en un 29.0 % respecto a la configuración con 500 entornos y en un 26.8 % respecto a la configuración con 1000 entornos. Aunque el error obtenido no puede considerarse bajo en términos absolutos, se verificó una mejora relativa acompañada de locomoción sostenida. En consecuencia, se cumplió la parte de la hipótesis relacionada con la generación de marcha estable y la reducción del error, pero no se alcanzó completamente el criterio de un error bajo.

Influencia del paralelismo en la convergencia PPO El número de entornos concurrentes tuvo una influencia importante en el desempeño del algoritmo PPO. La configuración con 4000 entornos fue la única que produjo locomoción sostenida de manera consistente; la configuración con 500 entornos generó una política estacionaria, mientras que la configuración con 1000 entornos produjo locomoción reducida y un mayor error de seguimiento de velocidad. La disminución progresiva del error promedio sobre el eje x , desde 1.00 m/s con 500 entornos hasta 0.71 m/s con 4000 entornos, muestra una

tendencia favorable al incrementar el número de entornos paralelos. Sin embargo, no fue posible evaluar configuraciones superiores a 4000 entornos debido a las limitaciones de memoria y capacidad de procesamiento del equipo utilizado. Con 4000 entornos y 10 minutos y 28 segundos de entrenamiento en una GPU RTX 4080 Laptop, se recolectaron 96×10^6 transiciones, lo que representó el mejor compromiso alcanzable entre recursos computacionales y calidad de la política.

Interpretación correcta de métricas parciales Los resultados mostraron que errores angulares menores en las configuraciones con 500 y 1000 entornos *no* constituyen una mejora —son un artefacto del robot estático o con locomoción reducida. Esta observación es relevante para futuros trabajos: el error angular aislado no debe usarse como indicador de calidad locomotora sin verificar que la política produce desplazamiento efectivo. La distancia recorrida y la recompensa de seguimiento lineal son métricas más confiables para evaluar locomoción real.

Validación estadística El análisis con cinco semillas aleatorias distintas, correspondientes a 0, 50, 500, 1000 y 4000, confirmó que el desempeño superior de la configuración con 4000 entornos es un efecto sistemático y no un artefacto de inicialización. La mejora se mantuvo consistente en las tres métricas evaluadas: error lineal, error angular y distancia recorrida, con varianza baja entre semillas.

Arquitectura de validación por simulaciones encadenadas La metodología de validación progresiva —ZMP instrumentado con CoppeliaSim, modelo URDF verificado con Gazebo/RVIZ, y entrenamiento masivo en Isaac Lab— garantizó que la política final se apoyara sobre un modelo físico consistente. Cada etapa filtró errores de parametrización antes de llegar al entrenamiento, evitando que problemas del modelo contaminaran los resultados del agente.

7.2 Aportaciones

Este trabajo establece tres aportaciones fundamentales al estado del arte en robótica bípeda:

- 1. Integración del criterio ZMP en la función de recompensa:** Se incorporó el análisis Zero Moment Point (ZMP) como un término de penalización dentro de la función de recompensa del agente de aprendizaje por refuerzo, en lugar de utilizarlo como una restricción externa o un filtro post-entrenamiento. Este enfoque permitió que el robot aprendiera *la necesidad física intrínseca* de mantener el equilibrio dentro del polígono de soporte mientras optimiza globalmente la trayectoria locomotora.
- 2. Validación estocástica y escalable:** La verificación estadística con múltiples semillas aleatorias y pruebas con alta paralelización (4000 entornos) proporcionó evidencia robusta de que el desempeño superior es una característica sistemática del entrenamiento masivo, no un artefacto de la inicialización. Este enfoque sirve como protocolo de validación replicable para futuros trabajos con PPO en robótica.
- 3. Flujo completo de modelado cinemático a política aprendida:** Se estableció y validó el flujo que vincula el modelado cinemático analítico (parámetros Denavit-Hartenberg, Jacobiano inverso para la transición inicial desde bipedestación hasta postura de marcha) con una política de control obtenida por aprendizaje, verificando que es posible mantener consistencia física a lo largo de toda la cadena. La validación progresiva mediante simulaciones encadenadas (CoppeliaSim → Gazebo → Isaac Lab) aseguró que el modelo físico subyacente fuera correcto y correctamente parametrizado antes del entrenamiento.

La configuración óptima demostró que el análisis del paralelismo no solo aumenta la cantidad de datos, sino que directamente incrementa la calidad y robustez del conocimiento adquirido por el agente. La arquitectura de validación encadenada (CoppeliaSim → Gazebo → Isaac Lab) aseguró que el éxito en la simulación estuviera respaldado por un modelo físico correcto y correctamente parametrizado.

7.3 Trabajo Futuro

El trabajo futuro contempla la mejora del control, el escalamiento del entrenamiento, la transición al mundo real y el desarrollo de capacidades de navegación.

Mejora de la locomoción omnidireccional La función de recompensa requiere ajustes en sus términos y pesos para equilibrar mejor el seguimiento angular con el lineal, permitiendo giros completos y marcha en reversa con la misma solidez que la marcha frontal. Asimismo, el error de seguimiento podría reducirse mediante un mayor

número de iteraciones y la optimización sistemática de los hiperparámetros de PPO. Una prioridad es estudiar la sensibilidad de cada término de la recompensa mediante un análisis de ablación que permita identificar compensaciones no deseadas entre objetivos.

Escalamiento del entrenamiento La tendencia observada entre 500, 1000 y 4000 entornos sugiere que incrementar el número de entornos paralelos podría contribuir a reducir adicionalmente el error y mejorar la locomoción. No obstante, esta posibilidad no pudo evaluarse debido a que el equipo utilizado alcanzó su límite de recursos al ejecutar 4000 entornos simultáneos. Como trabajo futuro, se propone evaluar configuraciones con un mayor número de entornos mediante una GPU con mayor capacidad de memoria o utilizando entrenamiento distribuido. Esta evaluación deberá comprobar si la tendencia observada se mantiene y determinar el punto a partir del cual el aumento del paralelismo deja de producir mejoras significativas.

Sim-to-Real Una vez consolidada la locomoción omnidireccional en simulación, el siguiente objetivo es construir el prototipo físico y transferir la política entrenada al robot real. Para mitigar la brecha sim-real, se recomienda introducir aleatorización de parámetros físicos, como la masa, la fricción y los retardos de los actuadores, durante el entrenamiento, así como explorar técnicas de ajuste fino con datos reales.

Control cinemático Se proyecta desarrollar un esquema de navegación basado en comandos de velocidad lineal y angular como entradas al agente, junto con estimación de pose mediante odometría visual o Visual SLAM. Esto dotaría a la plataforma de autonomía funcional: el robot podría seguir rutas sin depender de teleoperación.

Rediseño mecánico Finalmente, se sugiere revisar la arquitectura del diseño para elevar el centro de gravedad, lo cual facilitaría la implementación de estrategias de control más dinámicas en el robot real y permitiría mayores amplitudes de paso sin comprometer la estabilidad dinámica.

Referencias

- [1] J. Miguel Garcia-Haro, S. Martinez, and C. Balaguer, “Balance computation of objects transported on a tray by a humanoid robot based on 3d dynamic slopes,” in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, 2018, pp. 1–6.
- [2] D.-W. Lee, M.-J. Lee, and M.-S. Kim, “Whole body imitation of human motion with humanoid robot via zmp stability criterion,” in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015, pp. 1003–1006.
- [3] E. R. Magsino, “A walking bipedal robot using a position control algorithm based on center of mass criterion,” *ARPJ Journal of Engineering and Applied Sciences*, vol. 14, no. 11, pp. 2029–2038, June 2019.
- [4] R. Kurazume, S. Tanaka, M. Yamashita, T. Hasegawa, and K. Yoneda, “Straight legged walking of a biped robot,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 337–343.
- [5] M. Menad, Z. Derouiche, Z. A. Foitih, and W. Nouibat, “Kinematic modeling of a humanoid robot with 18 dof in a virtual environment,” in *Third International Conference on Innovative Computing Technology (INTECH 2013)*, 2013, pp. 424–429.
- [6] I.-w. Park, J.-y. Kim, and J.-h. Oh, “Online biped walking pattern generation for humanoid robot khr-3(kaist humanoid robot - 3: Hubo),” in *2006 6th IEEE-RAS International Conference on Humanoid Robots*, 2006, pp. 398–403.
- [7] X. Chen, Z. Yu, W. Zhang, Y. Zheng, Q. Huang, and A. Ming, “Bioinspired control of walking with toe-off, heel-strike, and disturbance rejection for a biped robot,” *IEEE Transactions on Industrial Electronics*, vol. 64, no. 10, pp. 7962–7971, 2017.
- [8] S. Phaniteja, P. Dewangan, P. Guhan, A. Sarkar, and K. M. Krishna, “A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots,” in *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2017, pp. 1818–1823.
- [9] H. Kim, D. Seo, and D. Kim, “Push recovery control for humanoid robot using reinforcement learning,” in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 488–492.

- [10] B. Liu, L. Ma, C. Liu, and B. Xu, "Locomotion control method for humanoid robot based on united hierarchical reinforcement learning," in *2020 IEEE 16th International Conference on Control & Automation (ICCA)*, 2020, pp. 1161–1166.
- [11] F. A. Chávez Estrada, J. C. Herrera Lozada, J. Sandoval Gutiérrez, and M. I. Cervantes Valencia, "Performance between algorithm and micro genetic algorithm to solve the robot locomotion," *IEEE Latin America Transactions*, vol. 17, no. 08, pp. 1244–1251, 2019.
- [12] R. Szabo, "Design approach for evolutionary techniques using genetic algorithms: Application for a biped robot to learn to walk and rise after a fall," *Mathematics*, vol. 11, p. 16, 2023, aCS Style. [Online]. Available: <https://doi.org/10.3390/math11132931>
- [13] T. T. Alemayoh, J. H. Lee, and S. Okamoto, "A deep learning approach for biped robot locomotion interface using a single inertial sensor," *Sensors*, vol. 23, no. 24, p. 19, 2023. [Online]. Available: <https://doi.org/10.3390/s23249841>
- [14] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," 2022. [Online]. Available: <https://arxiv.org/abs/2109.11978>
- [15] J. Oh, O. Sim, B. Cho, K. Lee, and J.-H. Oh, "Online delayed reference generation for a humanoid imitating human walking motion," *IEEE/ASME Transactions on Mechatronics*, vol. 26, no. 1, pp. 102–112, 2021.
- [16] M. A. Roa, D. Garzón-Alvarado, and R. Ramirez, *Design Methodology for Biped Robots: Applications in Robotics and Prosthetics*. National University of Colombia, 10 2007, p. 1.
- [17] H. Giberti, T. Abbattista, M. Carnevale, L. Giagu, and F. Cristini, "A methodology for flexible implementation of collaborative robots in smart manufacturing systems," *Robotics*, vol. 11, no. 1, p. 9, January 2022, accedido el 11 de diciembre de 2023. [Online]. Available: <https://doi.org/10.3390/robotics11010009>
- [18] J. H. Carvajal Rojas, "Metodología de diseño mecatrónico de robots," *Épsilon*, vol. 1, no. 25, p. 98, junio 2005, fecha de publicación: 2005-06-01.
- [19] R. Cisneros Limón, "Modelo matemático de un robot paralelo de seis grados de libertad," Tesis de Licenciatura, Universidad de las Américas Puebla, Mayo 2006.
- [20] L.-w. Tsai, *Robot Analysis: The Mechanics of Serial and Parallel Manipulators*. Department of Mechanical Engineering and Institute for Systems Research, University of Maryland: JOHN WILEY & SONS, INC, 1999.

- [21] R. Resnick, D. Halliday, and K. S. Krane, *Física, Vol. 1*, 4th ed. México: Compañía Editorial Continental, 2001.
- [22] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*, ser. Wiley select coursepack. Wiley, 2005.
- [23] K. L. Moore, A. F. Dalley, and A. M. R. Agur, *Anatomía con orientación clínica*, 8th ed. Madrid: Wolters Kluwer, 2001.
- [24] M. W. Whittle, *Gait Analysis: An Introduction*, 4th ed. Heidi Harrison, 2007.
- [25] J. H. Osorio and M. H. Valencia, “Bases para el entendimiento del proceso de la marcha humana,” *Archivos de Medicina (Col)*, vol. 13, no. 1, pp. 88–96, enero-junio 2013.
- [26] S. Shigemi, *ASIMO and Humanoid Robot Research at Honda*. Dordrecht: Springer Netherlands, 2019, pp. 55–90. [Online]. Available: https://doi.org/10.1007/978-94-007-6046-2_9
- [27] P. Sardain and G. Bessonnet, “Forces acting on a biped robot. center of pressure-zero moment point,” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 34, no. 5, pp. 630–637, 2004.
- [28] K. Kirby and I. Precision Intricast, *Foot and Lower Extremity Biomechanics II: Precision Intricast Newsletters, 1997-2002*. Precision Intricast, Incorporated, 2002. [Online]. Available: <https://books.google.com.mx/books?id=TPTjPQAACAAJ>
- [29] Z. Ding, Y. Huang, H. Yuan, and H. Dong, “Introduction to reinforcement learning,” *Deep reinforcement learning: fundamentals, research and applications*, pp. 47–123, 2020.
- [30] H. Adeli and C. Yeh, “Perceptron learning in engineering design,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 4, no. 4, pp. 247–256, 1989.
- [31] Colaboradores de los proyectos Wikimedia, “Perceptrón,” <https://es.wikipedia.org/wiki/Perceptr%C3%B3n>, 30 de agosto 2005, wikipedia, la enciclopedia libre.
- [32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.
- [33] A. Géron, *Aprende Machine Learning con Scikit-Learn, Keras y TensorFlow*, 3rd ed. Anaya Multimedia, junio 2023, fecha de publicación: 15 de junio de 2023.

- [34] (26 de junio de 2020) What is machine learning (ml)? University of California, Berkeley. Fecha de acceso: 17 de enero de 2024. [Online]. Available: <https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/>
- [35] I. Salian. (2 de agosto de 2018) Supervize me: What's the difference between supervised, unsupervised, semi-supervised and reinforcement learning? Fecha de acceso: 17 de enero de 2024. [Online]. Available: <https://blogs.nvidia.com/blog/supervised-unsupervised-learning/>
- [36] F. Plazas Niño, "Introducción al análisis clúster: una aplicación en la clasificación de campos petroleros," Tesis, Universidad Industrial de Santander, abril 2021.
- [37] Y. Li, "Deep reinforcement learning: An overview," 2018. [Online]. Available: <https://arxiv.org/abs/1701.07274>
- [38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: <https://arxiv.org/abs/1707.06347>
- [39] "Ros: Robot operating system," <https://www.ros.org/>, accedido el 6 de mayo de 2024.
- [40] "Understanding ros 2 nodes," <https://wiki.ros.org/es/ROS/Tutoriales/ComprendiendoNodosROS>, accedido el 6 de mayo de 2024.
- [41] "About ros 2 topics," <https://wiki.ros.org/Topics>, accedido el 6 de mayo de 2024.
- [42] MathWorks, "Gazebo Simulation Requirements," documentación oficial de MathWorks, accedido el 6 de mayo de 2024.
- [43] O. Robotics, "Gazebo homepage banner," 2026, accedido el 12 de marzo de 2026. [Online]. Available: <https://gazebosim.org/home-banner.28b26ce62e1ff331.png>
- [44] NVIDIA and Isaac Sim Contributors, "IsaacSim," GitHub repository, 2026, accessed: 2026-03-03. [Online]. Available: <https://github.com/isaac-sim/IsaacSim>
- [45] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," 2021. [Online]. Available: <https://arxiv.org/abs/2108.10470>
- [46] NVIDIA, "Isaac Sim Requirements," NVIDIA Omniverse Documentation, 2026, accessed: 2026-03-03. [Online]. Available: <https://docs.isaacsim.omniverse.nvidia.com/5.1.0/installation/requirements.html>

- [47] Y.-J. Kim, J.-Y. Lee, and J.-J. Lee, “A torso-moving balance control strategy for a walking biped robot subject to external continuous forces,” *International Journal of Humanoid Robotics*, vol. 12, 02 2015.
- [48] Y. Hu and K. Mombaur, “Bio-inspired optimal control framework to generate walking motions for the humanoid robot icub using whole body models,” *Applied Sciences*, vol. 8, no. 2, 2018. [Online]. Available: <https://www.mdpi.com/2076-3417/8/2/278>
- [49] R. Zhang, P. Vadakkepat, C.-M. Chew, and J. Janardhanan, “Mechanical design and control system configuration of a humanoid robot,” *Department of Electrical and Computer Engineering, Department of Mechanical Engineering, National University of Singapore*, 2005, email: {g0202712, prahlad, mpeccm, eng01274}@nus.edu.
- [50] D. Subramanian, “fusion2urdf-ros2,” <https://github.com/dheena2k2/fusion2urdf-ros2>, 2024, versión 1.0, Fecha de acceso: 2024-06-12.
- [51] Y. K. Ryuta Ozawa and J. Ishizaki, “Passivity-based posture control of a humanoid robot using a power-conserved transformation,” *Advanced Robotics*, vol. 35, no. 24, pp. 1485–1499, 2021. [Online]. Available: <https://doi.org/10.1080/01691864.2021.1985605>
- [52] P. Dayan and C. Watkins, “Q-learning,” *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.
- [53] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015. [Online]. Available: <https://doi.org/10.1038/nature14236>
- [54] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 1999. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf
- [55] V. Konda and J. Tsitsiklis, “Actor-critic algorithms,” in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 1999. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

- [56] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

Apéndices

A Matriz de rotación de la pierna derecha

Para compactar las expresiones se define $c_i = \cos(q_i)$ y $s_i = \sin(q_i)$, con $i = 1, \dots, 6$.

$$\begin{aligned}n_x &= c_6 \left[-s_5 (c_4 (s_1 c_3 + c_1 c_2 s_3) - c_1 s_2 s_4) + c_5 (-s_1 s_3 + c_1 c_2 c_3) \right] \\&\quad - s_6 \left[s_4 (s_1 c_3 + c_1 c_2 s_3) + c_1 c_4 s_2 \right], \\n_y &= s_6 \left[s_4 (c_1 c_3 - c_2 s_1 s_3) - c_4 s_1 s_2 \right] \\&\quad - c_6 \left[-s_5 (c_4 (c_1 c_3 - c_2 s_1 s_3) + s_1 s_2 s_4) + c_5 (-c_1 s_3 - c_2 s_1 c_3) \right], \\n_z &= s_6 (c_2 c_4 - s_2 s_3 s_4) \\&\quad - c_6 (c_2 s_4 s_5 - c_3 c_5 s_2 + c_4 s_2 s_3 s_5).\end{aligned}$$

$$\begin{aligned}o_x &= -c_6 \left[s_4 (s_1 c_3 + c_1 c_2 s_3) + c_1 c_4 s_2 \right] \\&\quad - s_6 \left[-s_5 (c_4 (s_1 c_3 + c_1 c_2 s_3) - c_1 s_2 s_4) + c_5 (-s_1 s_3 + c_1 c_2 c_3) \right], \\o_y &= c_6 \left[s_4 (c_1 c_3 - c_2 s_1 s_3) - c_4 s_1 s_2 \right] \\&\quad + s_6 \left[-s_5 (c_4 (c_1 c_3 - c_2 s_1 s_3) + s_1 s_2 s_4) + c_5 (-c_1 s_3 - c_2 s_1 c_3) \right], \\o_z &= s_6 (c_2 s_4 s_5 - c_3 c_5 s_2 + c_4 s_2 s_3 s_5) \\&\quad + c_6 (c_2 c_4 - s_2 s_3 s_4).\end{aligned}$$

$$\begin{aligned}
a_x &= c_1 c_2 c_3 s_5 - s_1 s_3 s_5 + c_3 c_4 c_5 s_1 \\
&\quad - c_1 c_5 s_2 s_4 + c_1 c_2 c_4 c_5 s_3, \\
a_y &= c_1 s_3 s_5 - c_1 c_3 c_4 c_5 + c_2 c_3 s_1 s_5 \\
&\quad - c_5 s_1 s_2 s_4 + c_2 c_4 c_5 s_1 s_3, \\
a_z &= c_2 c_5 s_4 + c_3 s_2 s_5 + c_4 c_5 s_2 s_3.
\end{aligned}$$

B Programas en Python para cinemática directa y Jacobiano.

Durante el desarrollo de la tesis se desarrollaron los siguientes programas para calcular la cinemática directa, obtener la matriz Jacobiana y estimar las velocidades articulares del robot. El primer módulo define la clase `Manipulator`, utilizada para modelar cada pierna como un manipulador serial a partir de los parámetros de Denavit-Hartenberg; el segundo módulo, denominado `utils`, contiene las funciones auxiliares para construir matrices de rotación, traslación y transformación homogénea. Para revisar el código completo con mayor detalle, se puede consultar el repositorio de GitHub donde se alojó el trabajo: <https://github.com/MiguelDevRobotics/Robotics-toolbox>. Repositorio completo donde se realiza el entrenamiento del robot en Isaac Lab: <https://github.com/MiguelDevRobotics/BipedoPPOtesisMaster>

```

1 import numpy as np
2 from numpy import pi
3 import sympy as sp
4 import matplotlib.pyplot as plt
5 from mpl_toolkits import mplot3d
6 from scipy.spatial.transform import Rotation as R
7 from .utils import set_axes_equal, dh_matrix_symbolic, dh_matrix
8
9
10 class Manipulator:
11     """
12     Serial manipulator model based on Denavit-Hartenberg parameters.
13
14     Parameters
15     -----
16     q_offset : array_like
17         Joint angle offsets in radians.
18     d : array_like
19         DH link offsets.

```

```

20     a : array_like
21         DH link lengths.
22     alpha : array_like
23         DH link twists in radians.
24     extra_transform : ndarray, optional
25         Homogeneous transformation applied before the first DH frame.
26         If not provided, the identity matrix is used.
27
28     Attributes
29     -----
30     n : int
31         Number of joints.
32     fk_symbolic_matrix : sympy.Matrix
33         Symbolic forward kinematics transformation matrix.
34     jacobian_matrix : callable
35         Numerical function generated from the symbolic Jacobian.
36     """
37
38     def __init__(self, q_offset, d, a, alpha, extra_transform=None):
39         """
40         Initialize a manipulator model.
41         """
42
43         self.q_offset = np.asarray(q_offset)
44         self.d = np.asarray(d)
45         self.a = np.asarray(a)
46         self.alpha = np.asarray(alpha)
47         self.n = self.q_offset.shape[0]
48         self.qi_sym = None
49
50         if extra_transform is None:
51             extra_transform = np.identity(4)
52         self.extra_transform = np.asarray(extra_transform)
53
54         if self.extra_transform.shape != (4, 4):
55             raise ValueError('extra_transform shape must be 4D ndarray')
56
57         self.fk_symbolic_matrix = self.symbolic_forward_kinematics()
58         jacobian_matrix = self.compute_symbolic_jacobian()
59         self.jacobian_matrix = sp.lambdify(self.qi_sym, jacobian_matrix, 'numpy')
60
61     def forward_kinematics(self, q, d=None, a=None, alpha=None):
62         """
63         Compute the forward kinematics of the manipulator.
64         """
65
66         q = np.asarray(q)
67         if d is None or a is None or alpha is None:
68             d = self.d

```

```

69         a = self.a
70         alpha = self.alpha
71
72     if self.extra_transform is None:
73         T = np.identity(4)
74     else:
75         T = self.extra_transform
76
77     for i in range(len(q)):
78         A_i = dh_matrix(q[i] + self.q_offset[i], d[i], a[i], alpha[i])
79         T = T @ A_i
80     return T
81
82 def draw_kinematic_chain(self, q=None):
83     """
84     Display a 3D visualization of the manipulator.
85     """
86
87     origin = np.array([0, 0, 0])
88     prev_origin = origin
89     xx = np.array([0.01, 0, 0, 1])
90     yy = np.array([0, 0.01, 0, 1])
91     zz = np.array([0, 0, 0.01, 1])
92
93     fig = plt.figure()
94     ax = fig.add_subplot(projection='3d')
95
96     ax.quiver(*origin, 0.01, 0, 0, color='red', linestyle='dashed')
97     ax.quiver(*origin, 0, 0.01, 0, color='green', linestyle='dashed')
98     ax.quiver(*origin, 0, 0, 0.01, color='blue', linestyle='dashed')
99
100    if q is None:
101        q = np.zeros_like(self.q_offset)
102    else:
103        q = np.asarray(q)
104
105    for i in range(q.shape[0]):
106        q_i = [q[j] for j in range(i + 1)]
107        T = self.forward_kinematics(q_i)
108
109        x_ = T @ xx
110        y_ = T @ yy
111        z_ = T @ zz
112
113        origin_new = T @ np.array([0, 0, 0, 1])
114        x_dir = x_[:3] - origin_new[:3]
115        y_dir = y_[:3] - origin_new[:3]
116        z_dir = z_[:3] - origin_new[:3]
117

```

```

118         ax.quiver(*origin_new[:3], *x_dir[:3], color='red')
119         ax.quiver(*origin_new[:3], *y_dir[:3], color='green')
120         ax.quiver(*origin_new[:3], *z_dir[:3], color='blue')
121
122         ax.plot(
123             [prev_origin[0], origin_new[0]],
124             [prev_origin[1], origin_new[1]],
125             [prev_origin[2], origin_new[2]],
126             color='black'
127         )
128         prev_origin = origin_new
129
130     set_axes_equal(ax)
131     plt.show()
132
133     def symbolic_forward_kinematics(self):
134         """
135         Generate the symbolic forward kinematics expression.
136         """
137
138         qi = [sp.symbols(f'q{i + 1}') for i in range(len(self.q_offset))]
139         self.qi_sym = qi
140         qi = [q + sp.pi * offset / np.pi for q, offset in zip(qi, self.q_offset)]
141         di = [d for d in self.d]
142         ai = [a for a in self.a]
143         alphai = [sp.pi * alpha / np.pi for alpha in self.alpha]
144
145         T = sp.Matrix(self.extra_transform)
146         for q, d, a, alpha in zip(qi, di, ai, alphai):
147             A_i = dh_matrix_symbolic(q, d, a, alpha)
148             T = T * A_i
149
150         return T
151
152     def print_forward_kinematics(self):
153         """
154         Pretty-print the symbolic forward kinematics expression.
155         """
156
157         sp.pprint(self.fk_symbolic_matrix)
158
159     def articulation_substitution(self, qi):
160         """
161         Substitute joint values into the symbolic forward kinematics.
162         """
163
164         values = {}
165         for q, q_sym in zip(qi, self.qi_sym):
166             if q_sym != 0:

```

```

167         values[q_sym] = q
168
169     print(values)
170     T = self.fk_symbolic_matrix.subs(values)
171     return T
172
173 def compute_symbolic_jacobian(self):
174     """
175     Compute the symbolic geometric Jacobian.
176     """
177
178     T = sp.Matrix(self.extra_transform)
179
180     z0 = T[:3, 2]
181     o0 = T[:3, 3]
182
183     z_axes = [z0]
184     origins = [o0]
185
186     qi = self.qi_sym
187     qi_offset = [q + sp.pi * off / np.pi for q, off in zip(qi, self.q_offset)]
188
189     for q, d, a, alpha in zip(qi_offset, self.d, self.a, self.alpha):
190         A = dh_matrix_symbolic(q, d, a, alpha)
191         T = T * A
192         z_axes.append(T[:3, 2])
193         origins.append(T[:3, 3])
194
195     o_e = origins[-1]
196
197     Jv = []
198     Jw = []
199
200     for i in range(len(qi)):
201         zi = z_axes[i]
202         oi = origins[i]
203
204         Jv.append(zi.cross(o_e - oi))
205         Jw.append(zi)
206
207     J = sp.Matrix.vstack(
208         sp.Matrix.hstack(*Jv),
209         sp.Matrix.hstack(*Jw)
210     )
211
212     return J
213
214 def compute_velocity(self, q, dq):
215     """

```

```

216         Compute end-effector spatial velocity.
217         """
218
219         J = self.jacobian_matrix(*q)
220         V = J @ np.asarray(dq)
221         return V
222
223     def compute_joint_velocities_analytic(self, v_desired, q):
224         """
225         Compute joint velocities using the symbolic Jacobian.
226         """
227
228         v_desired = np.asarray(v_desired, dtype=float)
229         q = np.asarray(q, dtype=float)
230
231         J = self.jacobian_matrix(*q)
232         dq = np.linalg.pinv(J) @ v_desired
233         return dq
234
235     def f(self, x):
236         """
237         Compute the end-effector pose.
238         """
239
240         x = np.asarray(x)
241
242         if x.ndim != 1:
243             raise ValueError('x must be 1D ndarray or list')
244         if x.shape != self.q_offset.shape:
245             raise ValueError('x and q_offset must have the same lenght')
246
247         gamma = [x[i] for i in range(x.shape[0])]
248         T = self.forward_kinematics(gamma)
249         position = T[:3, 3]
250         r = R.from_matrix(T[:3, :3])
251         orientation = r.as_rotvec()
252         return np.concatenate([position, orientation])
253
254     def jacobian(self, q, h=1e-6):
255         """
256         Numerically estimate the manipulator Jacobian.
257         """
258
259         q = np.asarray(q, dtype=float)
260         n = len(q)
261         f0 = self.f(q)
262         m = len(f0)
263         J = np.zeros((m, n))
264

```

```

265         for i in range(n):
266             dq = np.zeros_like(q)
267             dq[i] = h
268             J[:, i] = (self.f(q + dq) - f0) / h
269
270         return J
271
272     def compute_joint_velocities(self, v_desired, q):
273         """
274         Compute joint velocities using a numerical Jacobian.
275         """
276
277         v_desired = np.asarray(v_desired)
278         q = np.asarray(q)
279
280         J = self.jacobian(q)
281         inverse_jacobian = np.linalg.pinv(J)
282         joint_speeds = inverse_jacobian @ v_desired
283
284         return joint_speeds
285

```

Listing 1: Clase Manipulator para cinemática directa, Jacobiano y velocidades articulares.

```

1  import numpy as np
2  from numpy import pi
3  import sympy as sp
4  import matplotlib.pyplot as plt
5  from mpl_toolkits import mplot3d
6  from scipy.spatial.transform import Rotation as R
7
8
9  def rx(alpha):
10     """
11     Homogeneous rotation about the X axis.
12
13     Parameters
14     -----
15     alpha : float
16         Rotation angle in radians.
17
18     Returns
19     -----
20     ndarray
21         4*4 homogeneous transformation matrix.
22     """
23
24     T = np.array([

```

```
25         [1, 0, 0, 0],
26         [0, np.cos(alpha), -np.sin(alpha), 0],
27         [0, np.sin(alpha), np.cos(alpha), 0],
28         [0, 0, 0, 1]
29     ])
30
31     return T
32
33
34 def ry(beta):
35     """
36     Homogeneous rotation about the Y axis.
37     """
38
39     T = np.array([
40         [np.cos(beta), 0, np.sin(beta), 0],
41         [0, 1, 0, 0],
42         [-np.sin(beta), 0, np.cos(beta), 0],
43         [0, 0, 0, 1]
44     ])
45
46     return T
47
48
49 def rz(gamma):
50     """
51     Homogeneous rotation about the Z axis.
52     """
53
54     T = np.array([
55         [np.cos(gamma), -np.sin(gamma), 0, 0],
56         [np.sin(gamma), np.cos(gamma), 0, 0],
57         [0, 0, 1, 0],
58         [0, 0, 0, 1]
59     ])
60
61     return T
62
63
64 def tx(x):
65     """
66     Homogeneous translation along the X axis.
67     """
68
69     T = np.array([
70         [1, 0, 0, x],
71         [0, 1, 0, 0],
72         [0, 0, 1, 0],
73         [0, 0, 0, 1]
```

```
74
75
76 def ty(y):
77     """
78     Homogeneous translation along the Y axis.
79     """
80     T = np.array([
81         [1, 0, 0, 0],
82         [0, 1, 0, y],
83         [0, 0, 1, 0],
84         [0, 0, 0, 1]
85     ])
86
87     return T
88
89
90 def tz(z):
91     """
92     Homogeneous translation along the Z axis.
93     """
94     T = np.array([
95         [1, 0, 0, 0],
96         [0, 1, 0, 0],
97         [0, 0, 1, z],
98         [0, 0, 0, 1]
99     ])
100
101     return T
102
103
104 def dh_matrix(gamma, z, x, alpha):
105     """
106     Construct a Denavit-Hartenberg transformation matrix.
107
108     Parameters
109     -----
110     gamma : float
111         Joint angle.
112     z : float
113         Link offset.
114     x : float
115         Link length.
116     alpha : float
117         Link twist.
118
119     Returns
120     -----
121     ndarray
122         4*4 DH homogeneous transformation matrix.
```

```
123     """
124
125     T = rz(gamma) @ tz(z) @ tx(x) @ rx(alpha)
126     return T
127
128
129 def rx_symbolic(alpha):
130     """
131     Symbolic homogeneous rotation about the X axis.
132     """
133     T = sp.Matrix([
134         [1, 0, 0, 0],
135         [0, sp.cos(alpha), -sp.sin(alpha), 0],
136         [0, sp.sin(alpha), sp.cos(alpha), 0],
137         [0, 0, 0, 1]
138     ])
139
140     return T
141
142
143 def ry_symbolic(beta):
144     """
145     Symbolic homogeneous rotation about the Y axis.
146     """
147     T = sp.Matrix([
148         [sp.cos(beta), 0, sp.sin(beta), 0],
149         [0, 1, 0, 0],
150         [-sp.sin(beta), 0, sp.cos(beta), 0],
151         [0, 0, 0, 1]
152     ])
153
154     return T
155
156
157 def rz_symbolic(gamma):
158     """
159     Symbolic homogeneous rotation about the Z axis.
160     """
161     T = sp.Matrix([
162         [sp.cos(gamma), -sp.sin(gamma), 0, 0],
163         [sp.sin(gamma), sp.cos(gamma), 0, 0],
164         [0, 0, 1, 0],
165         [0, 0, 0, 1]
166     ])
167     return T
168
169
170 def tx_symbolic(x):
171     """
```

```

172     Symbolic homogeneous translation along the X axis.
173     """
174     T = sp.Matrix([
175         [1, 0, 0, x],
176         [0, 1, 0, 0],
177         [0, 0, 1, 0],
178         [0, 0, 0, 1]
179     ])
180
181     return T
182
183
184 def ty_symbolic(y):
185     """
186     Symbolic homogeneous translation along the Y axis.
187     """
188     T = sp.Matrix([
189         [1, 0, 0, 0],
190         [0, 1, 0, y],
191         [0, 0, 1, 0],
192         [0, 0, 0, 1]
193     ])
194
195     return T
196
197
198 def tz_symbolic(z):
199     """
200     Symbolic homogeneous translation along the Z axis.
201     """
202     T = sp.Matrix([
203         [1, 0, 0, 0],
204         [0, 1, 0, 0],
205         [0, 0, 1, z],
206         [0, 0, 0, 1]
207     ])
208
209     return T
210
211
212 def dh_matrix_symbolic(gamma, z, x, alpha):
213     """
214     Construct a symbolic Denavit-Hartenberg transformation matrix.
215
216     Parameters
217     -----
218     gamma : sympy.Symbol
219         Joint angle.
220     z : sympy.Symbol

```

```

221         Link offset.
222     x : sympy.Symbol
223         Link length.
224     alpha : sympy.Symbol
225         Link twist.
226
227     Returns
228     -----
229     sympy.Matrix
230         Symbolic DH transformation matrix.
231     """
232
233     T = rz_symbolic(gamma) * tz_symbolic(z) * tx_symbolic(x) * rx_symbolic(alpha)
234     return T
235
236
237 def set_axes_equal(ax):
238     """
239     Set equal scaling on all axes of a 3D plot.
240
241     Parameters
242     -----
243     ax : matplotlib.axes.Axes
244         3D axes object.
245     """
246     x_limits = ax.get_xlim3d()
247     y_limits = ax.get_ylim3d()
248     z_limits = ax.get_zlim3d()
249
250     x_range = abs(x_limits[1] - x_limits[0])
251     x_middle = np.mean(x_limits)
252     y_range = abs(y_limits[1] - y_limits[0])
253     y_middle = np.mean(y_limits)
254     z_range = abs(z_limits[1] - z_limits[0])
255     z_middle = np.mean(z_limits)
256
257     # The largest dimension is used as reference to equalize the others.
258     plot_radius = 0.5 * max([x_range, y_range, z_range])
259
260     ax.set_xlim3d([x_middle - plot_radius, x_middle + plot_radius])
261     ax.set_ylim3d([y_middle - plot_radius, y_middle + plot_radius])
262     ax.set_zlim3d([z_middle - plot_radius, z_middle + plot_radius])
263

```

Listing 2: Modulo utils para transformaciones homogeneas y matrices Denavit-Hartenberg.

C Material audiovisual complementario

Los videos de las simulaciones, entrenamientos y pruebas de locomoción desarrollados en esta investigación se encuentran disponibles en la siguiente lista de reproducción:

<https://youtube.com/playlist?list=PLQBIW6DdZczY&si=8E8SDywnBcPs8h3c>

D Productos de investigación

En esta sección se presentan los productos de investigación generados durante el desarrollo de la tesis. El principal producto vinculado de manera directa con los objetivos, la metodología y los resultados de este trabajo es el artículo *Deep Reinforcement for Bipedal Locomotion Using PPO and Parallel Training in Isaac Sim*, en el cual se documenta el entrenamiento de una política de locomoción bípeda mediante el algoritmo PPO y el uso de entornos paralelos en Isaac Sim. Adicionalmente, se desarrolló el artículo *Development of an Autonomous Omnidirectional Robot for Obstacle Avoidance Using Potential Fields*, enfocado en el desarrollo de un robot omnidireccional autónomo y en la evasión de obstáculos mediante el método de campos potenciales.

Deep Reinforcement Learning for Bipedal Locomotion Using PPO and Parallel Training in Isaac Sim

Miguel-de-Jesús Santiago-Vázquez¹[0009-0003-6845-2206], Oscar-D. Ramírez-C.^{2,3}[0000-0003-4284-794X], and José-A. Arias-A.³[0000-0002-8838-877X]

Universidad Tecnológica de la Mixteca, División de Estudios de Posgrado,
Huajuapán de León, Oaxaca 69004 Mexico
h4mike117117@gmail.com,{odramirez,anibal}@mixteco.utm.mx
<http://www.utm.mx>

Abstract. This work presents a deep reinforcement learning approach for bipedal locomotion of a custom designed robot trained in Isaac Sim. Proximal Policy Optimization (PPO) is proposed as the method to train a control policy based on desired joint positions, using a reward function focused on speed command tracking and gait stability. To analyze the effect of parallelism on learning, training configurations with 500, 1000, and 4000 simulated environments executed in parallel were evaluated. The results show that the configuration with 4000 environments offers the best compromise between training stability, reduction of linear tracking error (m/s), and the ability to generate sustained locomotion in simulation. Although the configurations with fewer environments result in shorter total training times, the achieved locomotion performance is lower. Furthermore, training with 4000 environments is integrated in approximately 10 min 28 seg on a single GPU, supporting the feasibility of using parallel training to accelerate bipedal locomotion learning on non-commercial robotic platforms.

Keywords: Biped locomotion · Reinforcement learning · PPO · Parallel training · Isaac Sim · Robotics simulation.

1 Introduction.

Biped locomotion remains one of the hardest problems in robotics because it requires coordinating multiple degrees of freedom while preserving the system's dynamic stability. Unlike other mobile platforms, a biped robot must stay stable across alternating stance and swing phases, which demands tight control over posture, velocity, and environment interaction [1,5].

Classical approaches to biped locomotion have typically relied on dynamic models, predefined trajectories, and analytical control. Although these methods have delivered relevant results, they usually depend on accurate models of both the robot and the environment, which can degrade performance under disturbances or unmodeled variations. In this context, deep reinforcement learning

Development of an Autonomous Omnidirectional Robot for Obstacle Avoidance Using Potential Fields

1st Miguel J. Santiago-Vásquez

*División de estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapán de León, México
h4mike117117@gmail.com*

2nd Deisy Carmona-Ruiz

*División de estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapán de León, México
carmonadeysi8@gmail.com*

3rd Karla I. Salado-Chávez

*División de estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapán de León, Oaxaca, México
itzchav3001@gmail.com*

4th Oscar D. Ramírez-Cárdenas

*División de estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapán de León, Oaxaca, México
odramirez@mixtureco.utm.mx*

Abstract—This work presents the development of an omnidirectional mobile robot with four Mecanum wheels, implementing potential fields algorithm for trajectory generation and obstacle avoidance achieving autonomy on our robot. The processing unit is Nvidia Jetson Nano, a single-board computer compatible with Robot Operating System, enabling communication through topics and nodes. Additionally, an ESP32 microcontroller is used for motor control and a LiDAR sensor is incorporated to get information from environment. Communication between the processing unit and nodes through topics ensures efficient data transfer. This combination, along with effective design and construction, aims to develop an autonomous omnidirectional robot capable of obstacle evasion. Experimental results demonstrate the effectiveness of the evasion technique used.

Index Terms—Omnidirectional, Potential fields, ROS, Jetson Nano, LiDAR, Odometry

I. INTRODUCTION

THERE are many kinematic configurations for mobile wheeled robots; this work focuses on omnidirectional movement. The primary advantage of omnidirectional robots, as mentioned in [1]–[3], is the use of Mecanum wheels, which enable simultaneous translational and rotational movement, improving mobility in restricted spaces.

The research and development of mobile robotics continue to advance, with the main challenge being to create systems capable of operating autonomously in real-world environments. Barrientos et al. [4] mention that for a mobile robot to be considered autonomous, it must navigate and perform tasks in various environments without human intervention. To achieve a certain level of autonomy, these robots are equipped with sensors that allow them to constantly monitor their relative position, both to their starting point and to their destination, avoiding obstacles and seeking the optimal trajectory [5], [6]. Therefore, different navigation architectures have been developed. On one hand, the deliberative architecture calculates the most optimal route to reach a final point on a previously

known map [7]. On the other hand, the architecture known as reactive focuses on unknown or dynamic scenarios, where the robot's movements are continuously calculated in real time until reaching the goal, as mentioned in [8]–[10]. This type of architecture has advantages such as low memory consumption and execution time, as its movements are calculated as it progresses, without the need to recalculate the entire map's trajectory. Moreover, in real-world environments, where conditions are often dynamic or unknown, the reactive architecture provides an effective solution by allowing the robot to move autonomously and reach the goal with greater adaptability [10].

Actually, various studies have investigated different methods for trajectory planning of autonomous mobile robots, as mentioned in [11], [12]; however, one method that has gained popularity due to its simple implementation and effective results is the artificial potential fields method. For instance in [13], it is mentioned that potential fields focus on the idea that the robot behaves like a particle immersed in a potential field. In this approach, obstacles are modeled as repulsive electric charges and the goal as an attractive electric charge. This creates a system where the sum of the attractive and repulsive potentials determines the robot's movement [14]. In recent years, various improvements have been made in the field of potential fields. Gao et al. [15] proposed a trajectory planning method based on potential fields for aerial robots, which increased efficiency in navigation in dynamic environments. Lee et al. [16] presented an integrated approach that combines route planning based on potential fields with deep learning to enhance the robot's adaptability in changing environments. Other works have focused on the physical implementation of potential fields in robots [1]–[3], [17], [18], demonstrating that this technique continues to be widely used and studied.

This work presents the development of a four-wheeled mecanum omnidirectional robot, where the potential fields