Universidad Autónoma de Querétaro
Facultad de Ingeniería
División de Investigación y Posgrado

Aprendizaje por refuerzo aplicado a la locomoción de robots caminantes

Tesis

Que como parte de los requisitos para obtener el Grado de
Maestro en Ciencias en Inteligencia Artificial

Presenta
Ing. Jorge Luis Espinosa Guerrero

Dirigido por
Dr. Efrén Gorrostieta Hurtado

Centro Universitario, Querétaro, Qro.
Diciembre 2021
México

Universidad Autónoma de Querétaro
Facultad de Ingeniería
División de Investigación y Posgrado

Aprendizaje por refuerzo aplicado a la locomoción de robots caminantes

Tesis

Que como parte de los requisitos para obtener el Grado de
Maestro en Ciencias en Inteligencia Artificial

Presenta
Ing. Jorge Luis Espinosa Guerrero

Dirigido por
Dr. Efrén Gorrostieta Hurtado

SÍNODO

Dr. Efren Gorrostieta Hurtado
Presidente

Dr. Juan Manuel Ramos Arreguín
Secretario

Dr. Jesús Carlos Pedraza Ortega
Vocal

Dr. José Emilio Vargas Soto
Suplente

Mtro. Juan Antonio Villeda Resendiz
Suplente

Centro Universitario, Querétaro, Qro.
Diciembre 2021
México

Esta tesis está dedicada a mi madre…

RECONOCIMIENTOS

Primero que nada me gustaría reconocer y agradecer a mi asesor el Dr. Efrén Gorrostieta Hurtado por su paciencia y dedicación al dirigir esta tesis. Por su voluntad de compartir el conocimiento y sus ideas. Sus consejos me llevaron hasta los orígenes de los temas presentados en este trabajo y me enseñaron la importancia de construir una base sólida antes de acercarse a las barreras del conocimiento.

Segundo, al Mtro. Antonio Villeda, por todas las horas de asesorías que me brindó de manera incondicional abordando los temas matemáticos más complejos que se abordan en este trabajo. Esta tesis no hubiese podido ser terminada sin su ayuda.

Tercero, a mi tutor el Dr. Jesús Carlos Pedraza Ortega, por brindar su guía y consejos para realizar un trabajo exitoso y estar constantemente pendiente del alumnado.

Por último, a todo el cuerpo de profesores de la Maestría en Ciencias en Inteligencia Artificial. Por compartir su conocimiento e influir directa o indirectamente en este trabajo. Incluyendo al Coordinador Dr. Saúl Tovar Arriaga por siempre buscar soluciones a los problemas de los alumnos y siempre ver por las personas primero.

RESUMEN

Los robots caminantes hexápodos son útiles en entornos con obstáculos comparables a su tamaño. Sin embargo, el problema de locomoción es complejo de resolver dada su alta dimensionalidad y los entornos desconocidos a los que estos robots son sometidos. El aprendizaje por refuerzo se adecúa naturalmente a resolver este problema. Estos algoritmos han alcanzado mucha popularidad al demostrar control equiparable al de un ser humano al resolver tareas específicas. Esta tesis presenta una revisión del desarrollo de robots hexápodos y el aprendizaje por refuerzo. Muestra la convergencia de estas dos líneas de investigación. Se discuten los algoritmos más populares del estado del arte que han resuelto este problema para algunas clases de robots caminantes y se desarrolla la implementación de un algoritmo de aprendizaje profundo por refuerzo que se aplica a todas las capas de la locomoción en un robot hexápodo simulado.

ABSTRACT

Six-legged robots are very useful in unknown environments with small obstacles compared to the robot size. However, the locomotion problem is complex to solve due to its high dimensionality and the unknown environment. Reinforcement learning fits naturally to solve this issue. Reinforcement learning algorithms have acquired relevance in the last years since they have achieved super human level control in specific tasks. This thesis introduces a brief history on the development of hexapod robots and reinforcement learning. Similarly, it is shown and discussed how this two areas are combined. The state of the art reinforcement learning algorithms applied for robot locomotion are discussed. Lastly, the implementation of a Deep Reinforcement Learning Algorithm from the state of the art is presented using a simulated hexapod robot.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

CAPÍTULO 1 - INTRODUCCIÓN

1.1 Planteamiento del problema

Realizar el control del movimiento de robots con varios grados de libertad utilizando técnicas de control clásico es un desafío (Devjanin et al, 1983). El sinfín de configuraciones de escenarios que se pueden encontrar en la vida real implica que es imposible programar el caminar del robot de forma universal.

El número de grados de libertad que un robot de este tipo puede poseer es alto (Youcef & Pierre, 2004), por lo que es necesario tratar el sistema con un número elevado de dimensiones que se vuelven un problema al tratarse sin algoritmos de inteligencia artificial.

Los robots caminantes tienen un mejor desempeño en terrenos irregulares y en escenarios con obstáculos (Klein, Olson & Pugh, 1983). No obstante, es imposible modelar todas las configuraciones, texturas y formas que el ambiente pueda tener. Esto representa otra variable importante que dificulta la movilidad de estos robots.

1.2 Justificación

En el desarrollo de la robótica, siempre ha sido indispensable la implementación de algoritmos o sistemas inteligentes, que le permitan a un robot interactuar con el entorno y hacer una buena toma de decisiones en las tareas que está desarrollando. También se ha demostrado en el transcurso de la historia, que este tipo de robots pueden contribuir a la sociedad en operaciones de búsqueda y rescate en desastres como incendios (Yu, Chieh & Samani, 2018), así como en tareas especializadas donde un ser humano estaría expuesto a un riesgo elevado. Un ejemplo claro es el COMET-III (Nonami et al., 2003).

La locomoción en los robots hexápodos puede verse como un sistema redundante, ya que existen diferentes configuraciones que nos pueden llevar a un mismo estado, debido al número de extremidades que este tipo de robots ofrecen. Por ello, es importante el desarrollo de algoritmos que permitan explotar estas características. Los algoritmos con inteligencia artificial ofrecerán al robot un mecanismo importante de toma de decisiones dependiendo de las variables del entorno, y el estado en el que el sistema se encuentre en determinado instante de tiempo.

Asimismo, existen investigaciones que concluyen que el aprendizaje por refuerzo puede ayudar a tratar el control de estos robots de tal forma que sea posible que aprendan a moverse en función del ambiente en el que se encuentran (Youcef & Pierre, 2004). Por lo tanto, es necesario investigar a fondo la incorporación de estas técnicas en la movilidad de estos robots.

1.3 Hipótesis

La implementación de algoritmos de aprendizaje por refuerzo permitiría a un robot caminante hexápodo la toma de decisión de los movimientos de cada una de sus patas ayudando a las tareas de locomoción.

1.4 Objetivos

1.4.1 Objetivo general

Implementar y evaluar la locomoción de un robot hexápodo a través de uno o varios algoritmos basados en aprendizaje por refuerzo.

1.4.2 Objetivos específicos

- Realizar el modelado del robot para emular su comportamiento a través de herramientas matemáticas como el análisis de la cinemática y software de simulación.
- Recrear un ambiente de simulación que permita realizar experimentación utilizando herramientas de software.
- Implementar y evaluar algoritmos de aprendizaje por refuerzo para determinar su desempeño.

1.5 Estructura de la tesis

El capítulo primero de la tesis es la sección introductoria y describe los motivos por los cuales se escribe este trabajo. Incluye la descripción del problema, justificación, hipótesis y objetivos. El capítulo dos relata los antecedentes desde dos perspectivas, la primera aborda la historia y algunos trabajos que se consideran el punto de inflexión del aprendizaje por refuerzo. La segunda, da una perspectiva del surgimiento de robots hexápodos. Finalmente, se habla de los trabajos donde estas dos líneas de investigación coinciden a lo largo de la historia.

El capítulo tres es la fundamentación teórica y contiene la base de lo que se utilizó para llegar a los resultados del trabajo. La fundamentación teórica toca brevemente el aprendizaje por refuerzo clásico, las redes neuronales, el aprendizaje por refuerzo profundo con un especial énfasis en el algoritmo que se utiliza en el presente trabajo. Se describe el problema de locomoción y el modelado en la robótica que, si bien, no es requerido por el algoritmo de inteligencia artificial, se describe como complemento para entender el proceso de simulación.

El capítulo cuatro contiene la metodología que se siguió en todas las etapas del desarrollo del trabajo. Esto es, el proceso de investigación, modelado,

implementación e integración. El quinto capítulo es el producto del seguimiento de la metodología y describe los resultados obtenidos aplicando los conceptos de la fundamentación teórica al problema.

Finalmente, el sexto capítulo concluye la tesis resumiendo los problemas encontrados durante el proceso, discutiendo los resultados en los escenarios probados y proponiendo trabajo futuro que no fue cubierto en la presente tesis.

CAPÍTULO 2 - ANTECEDENTES

Los orígenes de lo que ahora conocemos como aprendizaje por refuerzo dependieron de trabajos en muchas áreas del conocimiento, tales como: la informática, la estadística, la psicología, las neurociencias y las ciencias de la computación. Existen dos formas de generar agentes capaces de aprender del entorno: Aprender un controlador sin un modelo del sistema o aprender un modelo y, a partir de él, derivar un controlador (Kaelbling, Littman & Moore, 1996).

En 1989 se introdujo un método de aprendizaje por refuerzo llamado "Q-learning", en donde no es necesario contar con un modelo del sistema y en el que un agente aprende de manera óptima en un dominio de Markov a través de las consecuencias de sus actos (Watkins, 1989; Watkins & Dayan, 1992). En 1983 se muestra cómo un algoritmo resuelve el problema del péndulo invertido sin conocer las ecuaciones que rigen al sistema para realizar el control. La única retroalimentación con la que se cuenta, es el fracaso cuando el péndulo sobrepasa cierto ángulo, es decir, que el péndulo va a caerse. Se hace énfasis en diferenciar este algoritmo de uno basado en neuronas artificiales (Barto, Sutton & Anderson, 1983).

Por otra parte, también se han desarrollado algoritmos dependientes de un modelo, como Dyna-PI y algunas variantes del mismo (Dyna-Q) (Sutton, 1990) que consisten en tomar como entrada una descripción actual del entorno y producir una salida a él. El entorno genera también una salida de recompensa y la idea es maximizar esta recompensa a largo plazo. Esta arquitectura requiere de la existencia de un modelo del entorno.

El aprendizaje por refuerzo ha logrado ser combinado y complementado con otras técnicas de inteligencia artificial. Tal es el caso de algoritmos como GAFRL, que combina algoritmos genéticos con aprendizaje por refuerzo y lógica difusa

(Zhou, 2002), o aprendizaje por refuerzo profundo, que combina redes neuronales profundas con el aprendizaje por refuerzo. Éste último ha adquirido mucha relevancia en los últimos años ya que existen dos grandes casos de éxito los cuales son el aprender a jugar Atari 2600 a un nivel superhumano y el vencer al campeón mundial de Go en una partida oficial (Arulkumaran, Deisenroth, Brundage & Bharath, 2017). En el celebrado suceso de la partida de Go, también se utilizó la búsqueda avanzada en árboles de decisión (Silver et al., 2016).

Una de las áreas donde el aprendizaje por refuerzo puede ser aplicado es la robótica, en concreto, en los robots móviles. Existen distintas clasificaciones que se pueden utilizar para describir a un robot, algunas de ellas son: por el tipo de control; nivel de capacidad; configuración (sistema coordenado); y móviles o fijos. Hasta aproximadamente 1985 se hablaba generalmente de robots fijos y se comenzaba a introducir el concepto de robot móvil. Para entonces se podía distinguir entre los siguientes tipos de robots móviles (Critchlow, 1985):

Vehículo montado sobre llantas
Vehículo montado sobre una pista o riel
De cuatro a seis patas
Con dos patas

Entre 1980 y 1983 se argumentaba que el movimiento de robots de tres o más patas presenta un mejor desplazamiento sobre terrenos irregulares en comparación con otro tipo de robots móviles. En esos años se desarrolló un robot de seis patas (hexápodo) en Moscú, donde se expone la complejidad de la locomoción de un robot de este tipo desde el punto de vista del control (Devjanin et al., 1983). En 1983 se desarrolló el Hexápodo de la Universidad Estatal de Ohio, en donde su enfoque se centró en la movilidad en terreno irregular a través del uso de sensores piezoeléctricos, galgas extensiométricas y giroscopios (Klein, Olson & Pugh, 1983).

Es natural pensar que los robots hexápodos están inspirados, en gran medida, al estudio de insectos. El campo de la biología contribuyó de manera importante en el desarrollo de algunos robots. Tal es el caso de la "Máquina caminante TUM" (Pfeiffer, Eltze, & Weidemann, 1995), que se inspiró en el insecto *Carausius Morosus*, cuyo patrón de movimiento fue analizado por Graham en 1972. Otro ejemplo es el "Biobot", inspirado en la cucaracha americana *Periplaneta Americana* seleccionada por su velocidad y agilidad. (Delcomyn & Nelson, 2000).

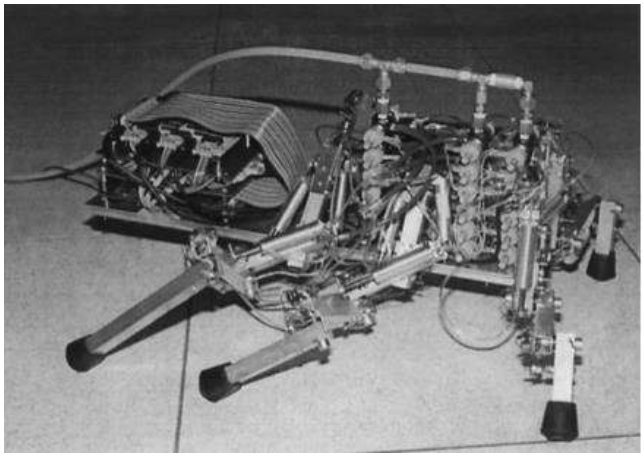

Figura 2.1 - Biobot (Delcomyn & Nelson, 2000)

En 2003 se desarrolló el COMET-III, mostrado en la figura 2.2, un robot hexápodo especializado en detección de minas que pesa alrededor de 1 000kg y mide 4 metros de largo, 2.5 metros de ancho y 0.8 metros de alto, que utilizó un sistema de control de supervisión de jerarquía y un sistema de control cooperativo multi-tarea (Nonami et al., 2003). Algunos desarrollos más recientes son los robots "Comet IV" (2009-2011), "CR200" (2013) y "Mantis" (2013), (Tedeschi & Carbone, 2014).

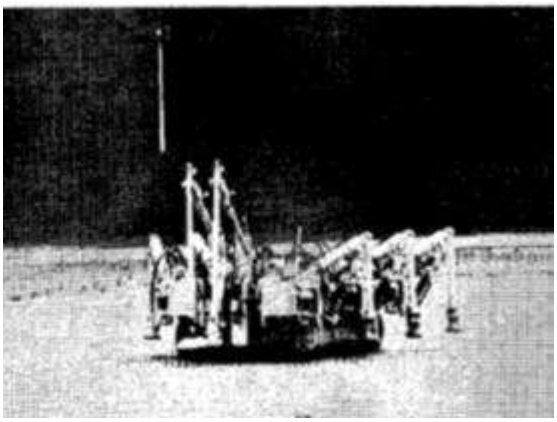

Figura 2.2 - Comet III (Nonami et al., 2003)

Se puede consultar una tabla comparativa de algunos robots hexápodos desarrollados a lo largo de la historia hasta el 2014 presentada por Tedeschi & Carbone en su artículo titulado *Design Issues for Hexapod Walking Robots*. En ella, se enlistan algunas de las características de los hexápodos como sus dimensiones, peso y aplicaciones.

Un hexápodo creado recientemente es el robot Octopus III, desarrollado en 2018. Pesa 240kg y tiene seis grados de libertad: es capaz de planear trayectorias utilizando métodos de optimización para evadir obstáculos conociendo su posición actual y la posición deseada (Zhao, Chai, Gao & Qi, 2018).

Existen algunas variantes del aprendizaje por refuerzo implementadas para robots hexápodos. En 1998 se utilizó una variante de "Q-learning", llamada "HQL", para aprender movimientos elementales de cada pata de manera individual, después controladores más complejos que aprendieran a mover el robot hacia adelante (Kirchner, 1998). Otra variante del algoritmo "Q-learning" es el "Q-learning distribuido", que se aplicó a un robot hexápodo debido a que un hexápodo necesita realizar movimientos coordinados para alcanzar la estabilidad. Por esto, el problema de control se descompone en un sistema distribuido, así que se tuvo que desarrollar un algoritmo basado en Q-learning para tratar este problema (Youcef & Pierre, 2004).

Se ha podido hacer una comparación entre un algoritmo genético contra la variante "Q-learning cooperativo" del aprendizaje por refuerzo en un mismo robot hexápodo llamado Kafka. Los resultados muestran que el algoritmo "Q-learning cooperativo" fue más rápido produciendo controladores distribuidos en una hora en contraste con las ocho horas que le tomó al algoritmo genético. Sin embargo, requirió considerablemente de más información ya que recibía datos en un intervalo de tiempo más corto (Barfoot, Earon & D'Eleuterio, 2006). El ejemplo más reciente es el diseño de algunos módulos de un robot hexápodo cuya aplicación sería rescates en incendios. En este desarrollo la locomoción del robot se realizó a través del algoritmo "Q-learning" (Yu, Chieh & Samani, 2018).

Trabajos más actuales muestran la incorporación de lógica difusa en el proceso del aprendizaje por refuerzo, ya sea a través de un sistema de recompensas difuso (Shahriari & Khayyat, 2013) o mediante "Q-learning difuso", utilizado en un robot hexápodo con sensores ultrasónicos para evitar obstáculos (Hong, Tang & Chen, 2017).

El uso del aprendizaje por refuerzo profundo aplicado a los hexápodos es de los más recientes avances que se han realizado en este campo. Existe un análisis del aprendizaje de las habilidades de locomoción simétricas y de baja energía de diferentes robots, incluyendo los hexápodos, utilizando el aprendizaje por refuerzo profundo (Yu, Turk & Liu, 2018). Con el fin de tener una mejor adaptabilidad a cambios bruscos en el ambiente, se ha propuesto también utilizar el aprendizaje por refuerzo profundo en combinación con el entrenamiento y almacenamiento de distintas políticas (Kume, Matsumoto, Takahashi, Ko & Tan, 2017).

Una investigación reciente muestra una configuración que utiliza el aprendizaje por refuerzo profundo jerárquico, donde el problema de control se separa en problemas individuales, similar al control distribuido. El primer nivel se encarga de

mover las juntas de manera individual y el segundo nivel selecciona el comportamiento adecuado utilizando redes neuronales profundas (Konen, Korthals, Melnik & Schilling, 2019).

Para entender el aprendizaje por refuerzo profundo se requiere abordar la clasificación de los métodos del aprendizaje por refuerzo. Esta clasificación ha evolucionado con el tiempo pero converge en dos grandes categorías: Métodos basados en función de valor y métodos basados en calcular el gradiente de la política. Este último grupo de algoritmos utiliza redes neuronales profundas como estimadores de la función de recompensa e incluso de las políticas y ha demostrado buenos resultados para tareas de control en tiempo continuo. (Schulman, Levine, Abbeel, Jordan, & Moritz, 2015)

Las pruebas para los algoritmos basados en el gradiente de política como *Trust Region Policy Optimization,* propuesto por Schulman, Levine, Abbeel, Jordan & Moritz, en 2015, o *Proximal Policy Optimization,* propuesto por Schulman, Wolski, Dhariwal, Radford, & Klimov en 2017, se han conducido en bípedos y cuadrúpedos con un grado alto de números de libertad. Ambos se han integrado en el proyecto OpenAI (OpenAI, 2017). Esto facilita replicar los trabajos de ambos artículos.

La tabla 1.1 condensa y resalta los trabajos descritos anteriormente que guardan mayor relación con el proyecto a desarrollar junto con el tema central de cada uno de ellos. Asimismo la tabla 1.2 muestra la relación entre algoritmos de aprendizaje por refuerzo que se han aplicado a robots hexápodos y el problema de locomoción que intentan resolver.

Tabla 1.1 – Trabajos previos

Autores	Año	Método utilizado	Enfoque central
Youcef & Pierre	2004	Q-Learning distribuido	Control de trayectorias y coordinación entre agentes
Barfoot, Earon & D'Eleuterio	2006	Q-Learning cooperativo	Acoplamiento y discretización de las extremidades. Solo dos acciones son permitidas para cada pata.
Shahriari & Khayyat	2013	Q-Learning difuso	Propuesta de estrategia básica para la generación de pasos de un robot hexápodo.
Schulman, Levine, Abbeel, Jordan & Moritz	2016	Deep Reinforcement Learning	Control en espacios continuos de varias dimensiones utilizando el estimador de ventaja generalizada. Locomoción de un bípedo.
Schulman, Wolski, Dhariwal, Radford & Klimov	2017	Deep Reinforcement Learning	Algoritmos de optimización próxima de políticas. Control continuo de locomoción de un bípedo y cuadrúpedo.
Hong, Tang & Cheng	2017	Q-Learning difuso	Evasión de obstáculos utilizando sensores ultrasónicos
Kume, Matsumoto, Takahashi, Ko & Tan	2017	Deep Reinforcement Learning	Entrenamiento y almacenamiento de distintas políticas
Yu, Chieh & Samani	2018	Q-Learning	Control de pasos para un robot de rescate en incendios
Yu, Turk & Liu	2018	Deep Reinforcement Learning	Estudio del aprendizaje de locomoción simétrica de baja energía en bípedos, cuadrúpedos, hexápodos y humanoides
Konen, Korthals, Melnik & Schilling	2019	Deep Reinforcement Learning	Entrenamiento de redes neuronales profundas en combinación con aprendizaje por refuerzo en un sistema de locomoción distribuido

Tabla 1.2 – Enfoque de los antecedentes

Algoritmo	Movimiento sobre un eje	Corrección de orientación	Movimiento de una pata	Evasión de obstáculos	Sobrepasar obstáculos
HQL (Kirchner, 1998)			X		
Q-Learning distribuído (Youcef & Pierre, 2004)		X			
Q-Learning cooperativo (Barfoot, Earon & D'Eleuterio, 2006)	X				
Q-Learning con señal de recompensa difusa (Shahriari & Khayyat, 2013)	X				
Q-Learning difuso (Hong, Tang & Chen, 2017)				X	
MMPRL (Kume, Matsumoto, Takahashi, Ko & Tan, 2017)	X				X
Modular DDPG (Konen, Korthals, Melnik & Schilling, 2019)	X		X		

CAPÍTULO 3 - FUNDAMENTACIÓN TEÓRICA

Las siguientes secciones introducen los conceptos técnicos clásicos y modernos del aprendizaje por refuerzo, modelado y locomoción que son la base o se aplicaron en el capítulo 4.

3.1 Aprendizaje por refuerzo clásico

El aprendizaje por refuerzo consiste en aprender a tomar acciones basadas en situaciones con el objetivo de maximizar una recompensa. No se le provee información al agente sobre las acciones que debe tomar, sino que él mismo explora las opciones en un esquema de prueba y error, midiendo el nivel de recompensa que cada acción genera. En algunos casos, la recompensa no se obtiene inmediatamente, sino que se manifiesta tiempo después. Este último tipo de sistemas es uno de los grandes retos que se presentan al resolver el problema del aprendizaje por refuerzo.

El ente que aprende y toma decisiones es llamado agente, y todo con lo que éste interactúa es llamado ambiente o entorno. El agente provee acciones y el entorno responde presentando nuevas situaciones al agente y un valor numérico al que se le conoce como señal de recompensa (Sutton & Barto, 1998), ver figura 3.1.

A diferencia de otras técnicas de aprendizaje máquina, el aprendizaje por refuerzo se enfrenta con el dilema de explotación contra exploración. La explotación es utilizar acciones que ya se han utilizado en el pasado y que han comprobado su éxito; mientras que la exploración es buscar nuevas acciones que devuelvan buenas recompensas. Es importante encontrar un balance entre estos dos conceptos ya que

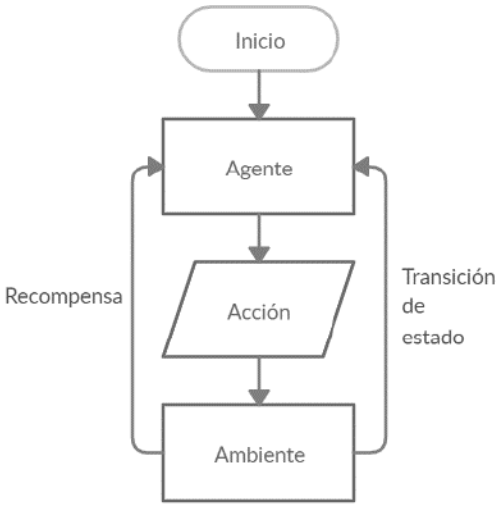

Figura 3.1 - Proceso del aprendizaje por refuerzo (Sutton & Barto, 1998)

para poder explotar una acción primero debió ser explorada. En la robótica un agente puede ser un robot móvil, el ambiente es todo lo que lo rodea, incluyendo sus sensores. Si un robot está buscando donde recargarse, entonces explora y explota acciones que lo acerquen a la estación de carga (Sutton & Barto, 1998).

Los elementos que componen al aprendizaje por refuerzo son:

- Una política que define el comportamiento del agente, esto es, a partir de los estados del entorno genera una acción correspondiente. Puede verse como una función $\pi(s, a)$ donde "s" representa un estado y "a" una acción.

- Una función de recompensa que provee un número real en cada instante de tiempo indicando qué tan buena fue la decisión tomada en ese momento.

- Una función de valor que indica qué tan buena es una decisión a largo plazo, es decir, es el total de recompensa que un agente espera acumular en el futuro partiendo de un estado determinado.

- Un modelo del sistema que imita el comportamiento que tendría el sistema en la realidad, este es un parámetro opcional, es posible aplicar métodos de aprendizaje por refuerzo sin la necesidad de contar con él.

Existen tareas que se pueden partir en secuencias llamadas episodios, por ejemplo, cualquier juego de mesa sigue una secuencia hasta llegar a un momento final, después de eso el juego se reinicia. A esta secuencia completa se le conoce como episodio. Sin embargo, hay tareas continuas que no pueden partirse en episodios como tareas de control. El objetivo del aprendizaje por refuerzo es maximizar la recompensa que el agente recibe, no obstante, la idea es maximizar la recompensa a largo plazo, no la recompensa instantánea, esto implica que si r_t es la recompensa que recibe el agente en el instante t entonces $R_t = r_{t+1} + r_{t+2} + \cdots + r_T$ donde T es el tiempo final de un episodio. Si la tarea no es episódica, T podrá llegar hasta infinito. En ese caso se necesita introducir factor de penalización "γ" que modifica el modelo de recompensa. Ver ecuación 3.1.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.1}$$

Donde $0 \leq \gamma \leq 1$

Nótese que al hacer $\gamma = 0$ se intentará maximizar solamente la recompensa inmediata, es decir, intentar que la acción a_t maximice solamente r_{t+1}. Entre más aproximemos el factor de penalización a uno se tomarán más en cuenta las recompensas futuras (Sutton & Barto, 1998).

Un agente toma decisiones en función del estado del entorno. El estado del entorno es toda información que esté disponible para el agente. La señal de estado cumple la propiedad de Markov si depende únicamente del estado y la acción

anterior. La ecuación 3.2 describe esta propiedad, siendo s_t, a_t y r_t el estado del proceso, la acción tomada y la recompensa obtenida respectivamente en el tiempo *t*. Es decir, la probabilidad de medir una recompensa s' en un tiempo $t+1$ no dependen de las acciones, estados ni recompensas anteriores.

$$P(s_{t+1} = s', r_{t+1} = r | s_t, a_t) = P(s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \dots, s_0, a_0, r_0) \quad (3.2)$$

De manera informal, la señal de estado contiene la información pasada de manera resumida y compacta. En el aprendizaje por refuerzo generalmente se asume que las señales de estado obedecen esta propiedad. Si una tarea satisface esta propiedad también se le llama proceso de decisión de Markov y si tanto el número de estados como el número de acciones son finitos entonces se le denomina Proceso de Markov Finito. De este tipo de procesos se define una probabilidad de transición. Ver ecuación 3.3.

$$P^a_{ss'} = P(s_{t+1} = s', | s_t = s, a_t = a) \quad (3.3)$$

Dicho de otra forma, es la probabilidad de pasar del estado s al estado s' a través de la acción a. De manera análoga el valor esperado de la siguiente recompensa está dado por la ecuación 3.4.

$$R^a_{ss'} = P(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s') \quad (3.4)$$

Estas dos cantidades definen los aspectos más significativos de la dinámica de un proceso de Markov (Ross, 2014). Algoritmos basados en programación dinámica requieren del conocimiento del modelo del sistema $P^a_{ss'}$ y el modelo de recompensa $R^a_{ss'}$ para calcular políticas y funciones de valor óptimas a través de algoritmos conocidos como *Policy Iteration* y *Value Iteration*. Sin embargo, otros métodos de

aprendizaje por refuerzo puedes estimar las políticas y funciones de valor sin la necesidad de un modelo del sistema (Sutton & Barto, 1998).

La mayoría de los algoritmos de aprendizaje por refuerzo clásicos se centran en estimar una función de valor, que indica que tan bueno es el estar en determinado estado. Es decir, es el valor esperado de la recompensa dado un estado y siguiendo la política π. En 1998, Sutton y Barto definen la función de valor dada una política π como se muestra en la ecuación 3.5. Donde E_π es el valor esperado bajo la política π, s es el estado del sistema, R corresponde a la suma acumulada de recompensas futuras, r es la recompensa inmediata y γ es un factor de penalización que se le da a las recompensas futuras.

$$V^\pi(s) = E_\pi\{R_t|s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k\, r_{t+k+1}|s_t = s\right\} \tag{3.5}$$

De forma análoga es posible definir una función de acción-valor para la misma política como el valor esperado de la recompensa dados un estado y una acción. La ecuación 3.5 se extiende introduciendo la variable acción como se muestra en la ecuación 3.6.

$$Q^\pi(s) = E_\pi\{R_t|s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k\, r_{t+k+1}|s_t = s, a_t = a\right\} \tag{3.6}$$

Estas dos funciones de valor pueden ser estimadas a partir de muestreos. Los métodos Monte Carlo y de Diferencias Temporales son ejemplos de familias de algoritmos utilizadas para estimar estas funciones (Sutton & Barto, 1998).

Al final, resolver una tarea de aprendizaje por refuerzo se resume a encontrar una política óptima π^* y para hacer esto es necesario encontrar una función de valor

óptima $V^*(s) = \max(V^\pi(s))$ para toda "s". Una política óptima también implica una función acción-valor óptima $Q^*(s) = \max\left(Q^\pi(s,a)\right)$ para toda "s" y "a". Algunos métodos que nos permiten encontrar o aproximar estas soluciones son la programación dinámica, los métodos de Monte Carlo y los métodos de diferencias temporales. La programación dinámica permite encontrar las políticas óptimas, pero requiere un modelo perfecto del sistema, lo cual es muy poco realista (Sutton & Barto, 1998). Los métodos Monte Carlo requieren que la tarea sea episódica, ya que la actualización de la función de valor se realiza al final del episodio. Y, los métodos de diferencias temporales combinan los dos anteriores al utilizar *bootstrapping* de tal forma que la función de valor se actualiza por cada muestra que se genere en el proceso (Sutton & Barto, 1998).

El aprendizaje por refuerzo profundo combina al aprendizaje por refuerzo con redes neuronales profundas. Éstos métodos se consideran el estado del arte y consisten en entrenar redes neuronales profundas que parametrizan y estiman políticas o bien funciones de valor (Arulkumaran, Deisenroth, Brundage & Bharath, 2017).

3.2 Aprendizaje por refuerzo basado en gradiente de políticas

Existen métodos dentro del aprendizaje por refuerzo que no requieren de mantener una función de valor, sino que se busca la política directamente en el espacio de las posibles políticas. Esto se realiza con un estimador del gradiente del valor esperado de la recompensa obtenido a través de varias trayectorias, donde una trayectoria se puede ver como la secuencia de cuatro elementos: estado actual, acción, recompensa y estado siguiente.

En la robótica generalmente se trabaja con espacios de estados y acciones continuos. Igualmente, estos sistemas tienen un número elevado de dimensiones.

Por estas dos razones los métodos basados en funciones de valor no son comúnmente utilizados en ésta área. Ya que si bien, en un sistema digital podemos asumir un tiempo discreto, los métodos basados en funciones de valor no escalan bien en altas dimensiones (Peters & Schaal, 2006).

Para buscar directamente las políticas en el espacio de todas las políticas posibles se parametriza la política como una función dependiente de varios parámetros normalmente llamados theta. De tal forma que el problema puede escribirse como: Encontrar los parámetros theta tal que la esperanza de la suma total de recompensas sea máxima. Es decir, se requiere optimizar la función 3.7 (Peters & Schaal, 2008).

$$J(\theta) = \frac{1}{a_\epsilon} E\left[\sum_{t_k} a_k r_k\right] \tag{3.7}$$

Donde E es el valor esperado, a_k es una constante que penaliza la recompensa, como se mostró en la ecuación 3.1, y a_ϵ una constante de normalización.

Para optimizar la función objetivo es posible calcular el gradiente y actualizar los parámetros theta en dirección del gradiente.

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta J(\theta)|_{\theta=\theta_k} \tag{3.8}$$

Donde α es conocido como el factor de aprendizaje.

Es importante mencionar que los cambios en la parametrización deben ser suaves ya que transiciones bruscas pueden hacer que el robot opere en condiciones peligrosas. Para garantizar esto junto con una mejora en la política por cada iteración se introducen métodos llamados actor-crítico, donde un nuevo elemento llamado

crítico se encarga de evaluar la política actual y proveer una base para mejorar las políticas estimadas (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017).

El problema para encontrar un estimador del gradiente de J se puede abordar utilizando una técnica llamada *likelihood ratio* (Aggarwal, 2018). Asumiendo que τ trayectorias son generadas por el sistema, J se puede escribir como se muestra en la ecuación 3.9.

$$J(\theta) = \sum_{\tau} p(\tau|\theta) r(\tau) \tag{3.9}$$

Es decir, el valor esperado de la recompensa total de una política se puede calcular en términos de todas las posibles trayectorias. De aquí

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta p(\tau|\theta) r(\tau)$$

$$\nabla_\theta J(\theta) = \sum_{\tau} \frac{p(\tau|\theta)}{p(\tau|\theta)} \nabla_\theta p(\tau|\theta) r(\tau)$$

$$\nabla_\theta J(\theta) = \sum_{\tau} p(\tau|\theta) \nabla_\theta \log p(\tau|\theta)\, r(\tau) \tag{3.10}$$

Esta última expresión se puede ver como una esperanza matemática y se puede expresar como en 3.11

$$\nabla_\theta J(\theta) = E[\nabla_\theta \log p(\tau|\theta) r(\tau)] \tag{3.11}$$

Y la esperanza matemática se puede estimar con muestras. Ver ecuación 3.12.

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=0}^{m} \nabla_\theta \log p\left(\tau^{(i)}|\theta\right) r\left(\tau^{(i)}\right) \tag{3.12}$$

Finalmente, para calcular la probabilidad de una trayectoria dada una parametrización se reescribe la expresión como se muestra en 3.13.

$$\nabla_\theta \log p\left(\tau^{(i)}|\theta\right) = \nabla_\theta log \left[\prod_{t=0}^{H} P\left(s_{t+1}^{(i)}\middle|s_t^{(i)}, a_t^{(i)}\right)\pi_\theta\left(a_t^{(i)}\middle|s_t^{(i)}\right)\right] \tag{3.13}$$

Al aplicar el gradiente respecto a theta y descomponiendo la multiplicación por una sumatoria a través de las propiedades del logaritmo se llega a la expresión 3.14.

$$\nabla_\theta \log p\left(\tau^{(i)}|\theta\right) = \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta\left(a_t^{(i)}\middle|s_t^{(i)}\right) \tag{3.14}$$

Esto demuestra que no es necesario conocer el modelo del sistema para estimar el gradiente (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). Solamente se requieren muestras.

$$\nabla_\theta J(\theta)|_{\theta=\theta_{anterior}} \tag{3.15}$$

Es posible evaluar la expresión 3.15 para obtener una estimación de qué tan buena fue la actualización de theta. A esta estimación se le conoce como crítico en una arquitectura llamada *actor-critic* donde existe un estimador para la política y un estimador para la función de valor o el crítico (Peters & Schaal, 2008).

La mayoría de los métodos de gradiente de política utilizan un estimador de esta naturaleza para calcular el gradiente del valor esperado de la recompensa y aplicarlo al algoritmo de gradiente ascendente estocástico. Incluyendo algoritmos como TRPO (Schulman, Levine, Abbeel, Jordan & Moritz, 2015) y PPO (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017).

3.3 Redes Neuronales Artificiales

Las redes neuronas artificiales se inspiran en las redes neuronas biológicas e intentan imitar el proceso de aprendizaje que ocurre en un organismo a través de un modelo de la neurona biológica. Este modelo consta de una neurona, una serie de dendritas con peso sináptico y un axón como se muestra en la figura 3.2.

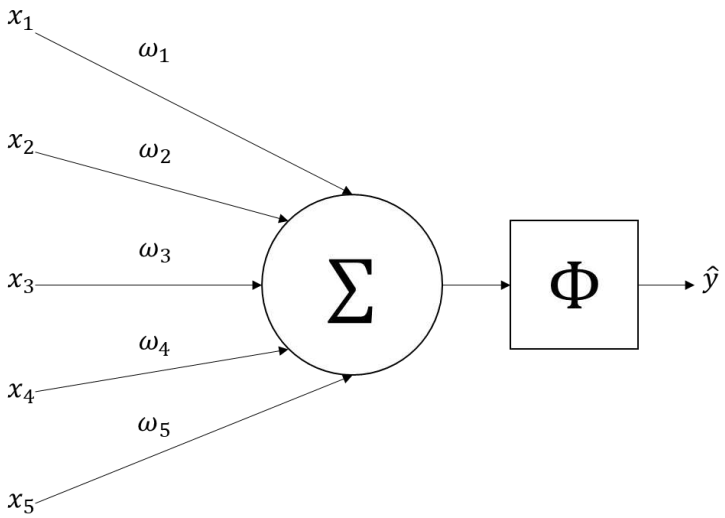

Figura 3.2 – Neurona Artificial (Aggarwal, 2018). Donde $\boldsymbol{x_1}, .., \boldsymbol{x_5}$ representan un valor de entrada a la neurona, $\boldsymbol{\omega_1}, ..., \boldsymbol{\omega_5}$ los pesos, $\hat{\boldsymbol{y}}$ la salida y $\boldsymbol{\Phi}$ la función de activación

Estas redes transforman las entradas en salidas propagando las entradas a través de la red desde las neuronas de entrada hasta las neuronas de salida, como se muestra en la ecuación 3.16. Los pesos ω son parámetros intermedios que modifican las entradas a cada neurona y el aprendizaje sucede ajustando estos pesos o parámetros hasta que el par de entrada y salida sea el esperado (Aggarwal, 2018). Desde esta perspectiva se puede ver a las redes neuronales artificiales como aproximadores de funciones que deben ajustar sus pesos a través de entrenamiento para representar la función deseada.

$$\hat{y} = \sum \omega_j x_j \tag{3.16}$$

La ecuación 3.16 es la representación de una neurona que recibe entradas x_i, con pesos ω_i produciendo una salida $\hat{y}$. Cada neurona cuenta con una función de activación de la forma que se muestra en la ecuación 3.17. Esta función de activación debe ser seleccionada cuidadosamente en función de lo que la neurona intenta predecir. Por ejemplo, si la salida de la neurona es una probabilidad, una buena función de activación podría ser la función sigmoide. Esta función de activación puede introducir no-linealidades que en arquitecturas de varias capas ayudan a modelar funciones que tienen no-linealidades (Aggarwal, 2018). E

$$\hat{y} = \Phi\left(\sum \omega_j x_j\right) \tag{3.17}$$

El entrenamiento se realiza utilizando recolectando un conjunto de datos de entradas con su salida deseada correspondiente, es decir, etiquetas. El proceso de entrenamiento consiste en introducir una entrada, a la red neuronal y comparar la salida de la red con la salida esperada. Esta comparativa provee retroalimentación que se puede utilizar para ajustar los pesos de la red.

La comparativa está matemáticamente justificada y asociada a una función llamada función de pérdida. Ésta función se utiliza como la función objetivo a optimizar. Para optimizar esta función se utilizan algoritmos como gradiente descendiente, sin embargo, en redes neuronales con más de una capa, calcular el gradiente es complicado (Aggarwal, 2018). Para ello, se utiliza el algoritmo llamado *backpropagation*.

La característica más importante de las redes neuronales artificiales es su capacidad de generalizar. Si bien se requiere una cantidad considerablemente alta de datos etiquetados para el entrenamiento, la exactitud de predecir salidas correctas

a partir de entradas nunca vistas en el entrenamiento es elevada en comparación con otros métodos de aprendizaje máquina (LeCun, Bengio & Hinton, 2015).

3.3.1 Redes Neuronales Profundas

Se dice que una red es multi-capa cuando existen neuronas ocultas entre la capa de entrada y capa de salida. Y, una red neuronal es profunda cuando tiene varias capas ocultas, lo que genera una cantidad muy grande de parámetros que ajustar de la red. Esto también implica la necesidad de un número de muestras etiquetadas más elevado (Aggarwal, 2018).

Las redes neuronales se caracterizan por su arquitectura, nodos y reglas de aprendizaje. La arquitectura de una red neuronal está dada por una matriz de pesos. Cada componente de la matriz indica el peso de la conexión entre dos nodos. Es posible que algún componente de la matriz sea cero, es decir, que un nodo no está conectado con otro. Esto permite que se puedan realizar distintas topologías. Un ejemplo de una arquitectura multicapa se muestra en la figura 3.3.

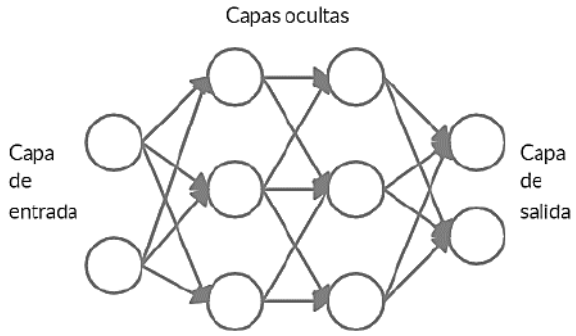

Figura 3.3 - Representación de una red neuronal multi-capa *fully connected* (Du & Swamy, 2014)

Cuando hablamos de aprendizaje profundo o redes neuronales profundas puede haber cientos de millones de pesos ajustables y cientos de millones de etiquetas para entrenar a la red. Técnicas como "gradiente descendiente estocástico" son utilizadas como parte del proceso para entrenar este tipo de redes.

Es posible incluir, después del proceso de entrenamiento, otro conjunto de ejemplos llamado el conjunto de pruebas. Este conjunto provee indicios de la capacidad de generalización que ha adquirido la red neuronal (LeCun, Bengio & Hinton, 2015).

3.4 Aprendizaje por refuerzo profundo

Recientemente se ha probado que es posible integrar redes neuronales profundas a los métodos clásicos del aprendizaje por refuerzo. Esta combinación ha sido muy exitosa para resolver problemas de toma de decisiones y son el estado del arte para resolver problemas de locomoción de robots. Las redes neuronales profundas escalan algunos métodos de aprendizaje por refuerzo al ser entrenadas estimar funciones como las funciones de valor o incluso las políticas directamente. Por ejemplo, el algoritmo Q-Learning se extiende a Deep Q-Learning (Van Hasselt, Guez, & Silver, 2016) al utilizar una red neuronal que parametriza y estima la función $Q(s, a; \theta_i)$ donde los parámetros theta son los pesos de la red neuronal.

Los métodos de gradiente de política no son la excepción. *Deterministic Policy Gradient* se ha escalado a *Deep Deterministic Policy Gradient*. Y métodos como TRPO (Schulman, Levine, Abbeel, Jordan & Moritz, 2015) o PPO (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017), que son dos de los métodos más utilizados para locomoción, utilizan redes neuronales profundas como estimadores de las políticas y funciones de valor.

El problema de locomoción se divide en varias etapas, como se muestra en la sección 3.5. Los algoritmos de aprendizaje por refuerzo clásico se adaptan bien a alguna de estas etapas como la coordinación de pasos. Pero el aprendizaje por refuerzo profundo se ha aplicado bien a resolver todas las capas de locomoción

desde la generación de trayectorias de las patas hasta la coordinación de los movimientos.

A continuación se hace énfasis y se detalla uno de los algoritmos de aprendizaje por refuerzo profundo que más éxito ha tenido resolviendo de manera comprehensiva todas las capas de la locomoción en bípedos y cuadrúpedos. Y que es utilizado en el desarrollo de esta tesis sobre un robot hexápodo.

3.4.1 Proximal Policy Optimization

La familia de algoritmos *Proximal Policy Optimization* PPO (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) se propone como una alternativa al algoritmo *Trust Region Policy Optimization* (Schulman, Levine, Abbeel, Jordan & Moritz, 2015) menos compleja e igual de robusta. Parte de encontrar un estimador para el gradiente de la suma total de recompensas que tiene la forma mostrada en la ecuación 3.18.

$$\hat{g} = \hat{E}_t\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\,\widehat{A_t}\right] \tag{3.18}$$

Donde $\hat{E}_t$ es el valor esperado estimado en el tiempo t, $\widehat{A_t}$ es conocido como el estimador de la función de ventaja en el tiempo t, π_θ es una política parametrizada por θ, a_t y s_t son la acción y estado respectivamente en el tiempo t.

El estimador $\hat{g}$ se obtiene aplicando el gradiente a la ecuación 3.19

$$L^{PG}(\theta) = \hat{E}_t\left[\log \pi_\theta(a_t|s_t)\,\widehat{A_t}\right] \tag{3.19}$$

Sin embargo, una política nueva se calcula utilizando muestras generadas por una política anterior. Tomando esto en cuenta, se puede escribir una forma equivalente a la ecuación 3.19 resultando en la ecuación 3.20.

$$L^{IS}_{\theta_{anterior}}(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{anterior}}(a_t|s_t)}\widehat{A_t}\right] \tag{3.20}$$

Si se intenta optimizar cualquiera de estas dos funciones directamente es posible que los cambios en theta no sean suaves. Para resolver este problema se define una región de confianza (*trust region*). Mientras se esté dentro de esta región es seguro optimizar la función de costo, esto garantiza incrementos suaves en theta. Existen muchas formas para definir esta región, puede ser a través de la distancia euclidiana, pero la más común es utilizar la divergencia KL que mide la diferencia entre dos distribuciones de probabilidad (Schulman, Levine, Abbeel, Jordan & Moritz, 2015). Para definir esta región se maximiza la expresión anterior sujeta a una región, como se muestra en la ecuación 3.21. Donde la restricción está compuesta por la divergencia KL de la política anterior $\pi_{\theta_{anterior}}$ respecto de la actual π_θ, esta divergencia se restringe por un valor δ.

$$\max_\theta \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{anterior}}(a_t|s_t)}\widehat{A_t}\right] \tag{3.21}$$
$$sujeto\ a\ \hat{E}_t\left[KL\left[\pi_{\theta_{anterior}}(\cdot\,|s_t), \pi_\theta(\cdot\,|s_t)\right]\right] \leq \delta$$

Este algoritmo es complicado de usar en arquitecturas MIMO. Se necesita calcular un gradiente conjugado que hace la implementación más compleja (Schulman, Levine, Abbeel, Jordan & Moritz, 2015).

A diferencia de los métodos TRPO, PPO analiza el factor $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{anterior}}(a_t|s_t)}$ y lo limita a cierto intervalo definido por un hiperparámetro ε, como se muestra en la ecuación 3.22.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{anterior}}(a_t|s_t)}$$
$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\widehat{A_t}, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\widehat{A_t}\right)\right] \quad (3.22)$$

En esencia se intenta que la tasa de cambio entre ambas probabilidades esté siempre cercana a uno. Sin embargo, empíricamente se introduce el operador *min* entre la tasa de cambio con y sin restricción para obtener un límite inferior (pesimista) sobre la tasa de cambio sin restricciones (Schulman, Levine, Abbeel, Jordan & Moritz, 2015).

Muchos algoritmos utilizan una función de valor V aprendida en el proceso y utilizada como crítico. Por ejemplo, GAE (*Generalized Advantage Estimation*) (Schulman, Moritz, Levine, Jordan & Abbeel, 2015). Es posible modificar la función de costo introduciendo esta función de valor y un factor de entropía S, como se muestra en la ecuación 3.23.

$$L_t^{CLIP+VF+S}(\theta) = \hat{E}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF} - c_2 S] \quad (3.23)$$

Donde c_1 y c_2 son constantes, S es el bono de entropía y L_t^{VF} es la función de pérdida para estimar la función de valor $\left(V_\theta(s_t) - V_t^{targ}\right)^2$.

Usualmente se generan muestras con una política durante T pasos. Y con la información obtenida se puede calcular el estimador GAE utilizando la ecuación 3.24.

$$\widehat{A_t} = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{3.24}$$

Algoritmo 3.1

Proximal Policy Optimization (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017)

Desde j=1, 2, …
    Desde i=1 hasta N
        Ejecutar política $\pi_{\theta_{anterior}}$ *para por T unidades de tiempo*
        Calcular las ventajas $\widehat{A_1}, \widehat{A_2}, \ldots, \widehat{A_T}$
    Fin
    Optimizar L con respecto a θ
    $\theta_{anterior} = \theta$
Fin

3.5 Locomoción

El problema de locomoción para un robot hexápodo se puede descomponer en tres capas (Tedeschi & Carbone, 2015):

- Planeación de trayectorias del robot
- Coordinación de pasos
- Planeación del movimiento de la pata.

La primera capa recibe la información de más alto nivel como la posición inicial y final del robot, velocidad y aceleración deseados. Esta capa se encarga de producir la dirección del vector que el robot debe seguir dado el estado actual para moverse al punto final deseado. Esta información se recibe en la segunda capa y es esta capa la que se encarga de coordinar las seis patas para producir el movimiento en la dirección deseada. La última capa recibe la información de la segunda capa para producir la trayectoria de la pata (Tedeschi & Carbone, 2015). La tabla 3.1 muestra la capa donde trabajan algunos robots hexápodos y algoritmos de aprendizaje estudiados en el capítulo 2.

La estabilidad del robot es un factor importante que tiene que estar bajo control para una locomoción exitosa. Existen dos tipos de estabilidad, dinámica y estática. La dinámica sucede cuando la proyección vertical del centro de gravedad esta fuera del polígono de soporte (Tedeschi & Carbone, 2015). Esto sucede cuando el robot está corriendo o brincando. Por otro lado, la estabilidad estática sucede cuando la proyección vertical del centro de gravedad está dentro del polígono de soporte (Gorrostieta, & Vargas Soto, 2008).

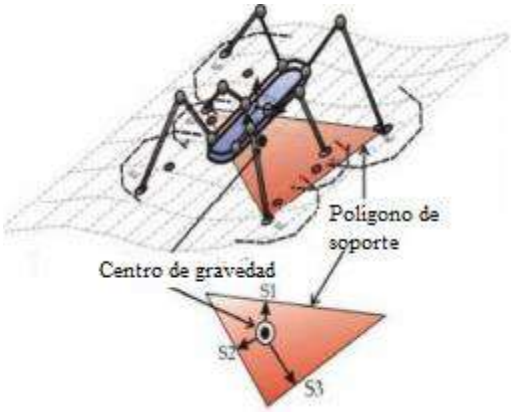

Figura 3.4 – Polígono de soporte en un hexápodo (Gorrostieta, & Vargas Soto, 2008)

Cuando la proyección vertical del centro de gravedad cae fuera del polígono de soporte el robot caerá a menos que se apliquen fuerzas para contrarrestar la caída. Este polígono se determina geométricamente por el número de patas que están en contacto con el suelo.

3.5.1 Planeación de pasos

Existen varios patrones de caminado para robots hexápodos. La selección de uno sobre otro depende del terreno y velocidad que se requiera alcanzar. Un patrón de caminado queda definido por la fase ipsolateral, fase contra lateral y el factor de trabajo. En un patrón regular cada pata tiene el mismo factor de trabajo lo que se traduce en que cada pata pasa en aire o en tierra el mismo lapso de tiempo. Por

ejemplo, un paso ondulado tiene un factor mayor que 0.5 para todas sus patas. Este patrón se encuentra comúnmente en insectos y garantiza un movimiento estáticamente estable con un amplio rango de velocidad (Tedeschi & Carbone, 2015).

Sin embargo, el paso más famoso es el trípode. Consiste en mover las patas delantera y trasera de un lado y la pata de en medio del lado contrario al mismo tiempo. Este paso es estáticamente estable y funciona para altas velocidades en tierra plana.

Existen otra clase conocida como locomoción libre (Gorrostieta, & Vargas Soto, 2008). Este paso no está atado a ninguna regla.

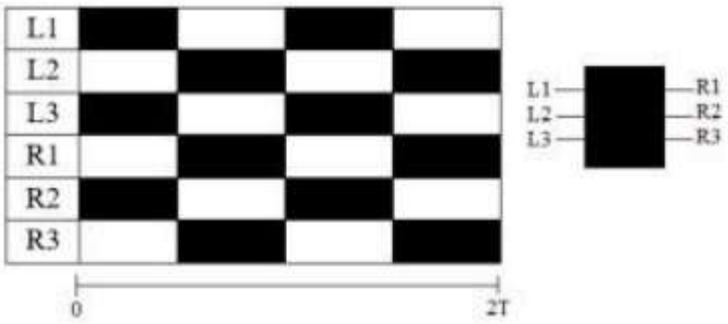

Figura 3.5 – Paso trípode (Tedeschi & Carbone, 2015)

3.5.2 Generación de trayectorias de la pata

La última capa de la locomoción requiere generar trayectorias moviendo las juntas de las patas de un estado inicial a un estado deseado dentro del espacio de trabajo del robot. Este paso requiere de un buen modelo cinemático del robot para diseñar un controlador capaz de garantizar que la punta de la pata se moverá a la posición deseada (Tedeschi & Carbone, 2015). Como se mencionó anteriormente. Existen algoritmos de aprendizaje por refuerzo independientes de un modelo del

sistema y es mediante la generación de múltiples trayectorias que estos algoritmos pueden aprender los movimientos deseados sin la necesidad del modelo cinemático.

La trayectoria más común generada por la punta de la pata es la trayectoria parabólica. Sin embargo existen estudios basados en el movimiento de insectos que muestran más tipos de trayectorias.

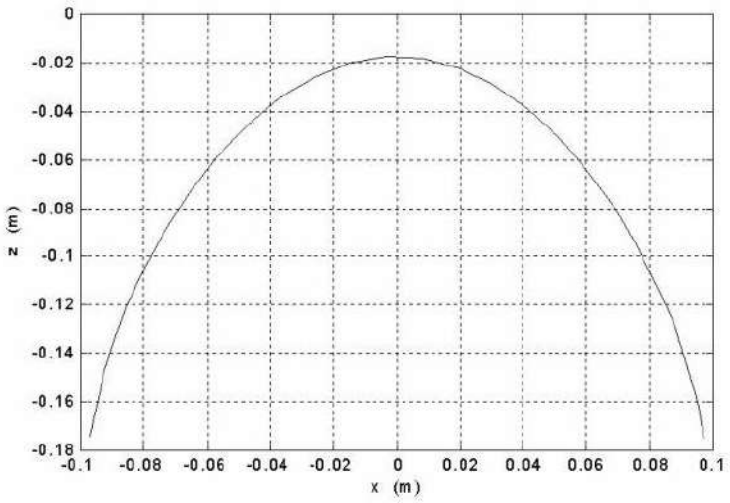

Figura 3.6 – Generación de trayectoria tipo parábola para una pata (Gorrostieta, & Vargas Soto, 2008)

Tabla 3.1 – Aplicación de algoritmos de aprendizaje por refuerzo en las capas de locomoción

Algoritmo	Planeación de trayectorias	Coordinación de pasos	Planeación del movimiento de la pata	Año
HQL (Kirchner, 1998)		X	X	1998
Q-Learning distribuído (Youcef & Pierre, 2004)		X		2004
Q-Learning cooperativo (Barfoot, Earon & D'Eleuterio, 2006)		X		2006
Q-Learning con señal de recompensa difusa (Shahriari & Khayyat, 2013)		X		2006
Q-Learning difuso (Hong, Tang & Chen, 2017)	X			2017
MMPRL (Kume, Matsumoto, Takahashi, Ko & Tan, 2017)		X	X	2017
Modular DDPG (Konen, Korthals, Melnik & Schilling, 2019)		X	X	2019

3.6 Modelado del robot

Es posible generar diferentes configuraciones para un robot hexápodo donde cada pata tenga cierto número de grados de libertad. Este trabajo se centrará en modelar un robot con seis patas adjuntas a un cuerpo rígido donde cada pata tendrá tres grados de libertad como se muestra en la figura 3.7.

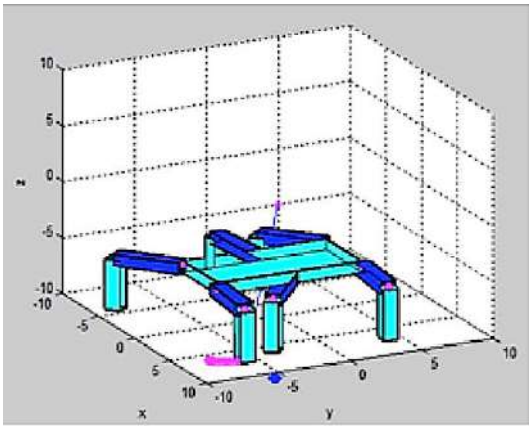

Figura 3.7 – Modelo de un robot hexápodo

Existen herramientas matemáticas que nos permiten realizar el modelado en la robótica. La representación de sólidos en el espacio es fundamental para describir la posición de un robot. Las coordenadas homogéneas resuelven este problema condensado la información de un vector en cuatro componentes. La matriz de transformación homogénea permite la transformación de un vector de coordenadas homogéneas de un sistema coordenado a otro, como se muestra en la ecuación 3.24.

$$T = \begin{pmatrix} \boldsymbol{R_{3x3}} & \boldsymbol{P_{3x1}} \\ 0 & 1 \end{pmatrix} \tag{3.24}$$

Donde R representa una matriz de rotación y P un vector de traslación. Por lo tanto, la transformación homogénea es una matriz de cuatro por cuatro.

El análisis de la cinemática de un robot se centra en tres partes importantes: la cinemática directa, que trata de determinar la posición de un robot dado un sistema de coordenadas, la cinemática inversa, que resuelve la configuración que debe tomar el robot dada la posición y orientación, y la resolución de las velocidades de las articulaciones y el extremo del robot a través de la matriz Jacobiana (Barrientos, Peñin, Balaguer & Aracil, 1997).

Para una pata de un robot hexápodo con tres grados de libertad la posición de la punta se puede determinar como se muestra en la ecuación 3.25 (García-López, Gorrostieta-Hurtado, Vargas-Soto, Ramos-Arreguín, Sotomayor-Olmedo & Morales, 2012).

$$q = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} l_2 \cos\theta_1 \cos\theta_2 + l_3 \cos\theta_1 \cos(\theta_2 + \theta_3) \\ l_2 \cos\theta_1 \sin\theta_2 + l_3 \sin\theta_1 \cos(\theta_2 + \theta_3) \\ l_1 + l_2 \sin\theta_2 + \cos(\theta_2 + \theta_3) \end{bmatrix} \tag{3.25}$$

Donde l_1, l_2 y l_3 son las longitudes de los tres eslabones y θ_1, θ_2 y θ_3 los ángulos de las juntas.

Por otro lado, el modelo dinámico explica los movimientos que ocurren cuando momentos y fuerzas son aplicados a cuerpos rígidos. Para el caso de una pata con tres grados de libertad se busca encontrar los torques de cada junta. Esto se puede hacer mediante el método de Newton-Euler. La ecuación 3.26 muestra la dinámica para una sola pata.

$$\tau = M(\Theta)\ddot{\Theta} + B(\Theta)\left[\dot{\Theta}\dot{\Theta}\right] + C(\Theta)\left[\dot{\Theta}^2\right] + G(\Theta) \tag{3.26}$$

Donde M es la matriz de masas de la pata, B son los coeficientes de Coriolis, C son los coeficientes de las componentes centrífugas y G el vector de gravedad.

Beaber, Zaghloul, Kamel & Hussein en 2018 muestran que todas las constantes están determinadas por las inercias, longitudes y masas de los tres eslabones. Para facilitar el cálculo de las inercias se asume que las formas de las patas y cuerpo del robot tienen la forma de un prisma rectangular o cilindros.

Tabla 3.4 – Fórmulas para calcular las matrices de inercia dependiendo del sólido

Sólido	Matriz de inercia	Términos
Prisma Rectangular	$I = \begin{bmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+h^2) \end{bmatrix}$	m: masa h: altura d: profundidad w: anchura
Cilindro	$I = \begin{bmatrix} \frac{1}{12}m(3r^2+h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2+h^2) & 0 \\ 0 & 0 & \frac{1}{2}mr^2 \end{bmatrix}$	m: masa r: radio h: altura

CAPÍTULO 4 - METODOLOGÍA

Para cumplir los objetivos, se debe entender primero el sistema que se va a trabajar y posteriormente se necesitará un entorno suficientemente completo como para realizar la experimentación. Para resolver el primer punto, es preciso un modelado simple del robot. Para abordar el segundo punto, será necesario el diseño u obtención de un simulador que permita emular la locomoción del robot y de esta forma probar las distintas estrategias de movimiento. Ambas tareas están estrechamente relacionadas dado que el modelo puede considerarse como una entrada al simulador.

Se iniciará el modelado cuyo producto se introducirá en el simulador. Se realizarán pruebas al mismo para garantizar que el modelo es una representación suficientemente buena de la realidad. En caso de no obtener buenos resultados se volverá a realizar el modelado en un proceso iterativo. Al final de este ciclo se deberá contar con el modelo y la configuración adecuada del simulador. Este proceso se representa en la figura 4.1.

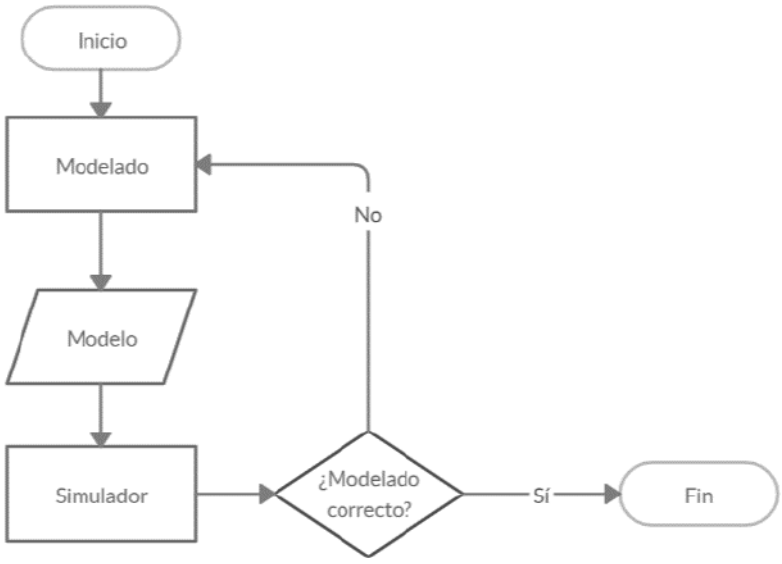

Figura 4.1 - Proceso de modelado

La generación de algoritmos para la estrategia de locomoción estará basada en una metodología iterativa de tal forma que se pueda mejorar de forma incremental el algoritmo en cada iteración. Primero, se realizará una investigación sobre los algoritmos basados en aprendizaje por refuerzo ya existentes; se analizarán las ventajas y desventajas de cada uno, y su fundamentación matemática. Después se procederá a realizar la implementación en el lenguaje de programación que mejor se adapte. Se obtendrá una primera versión de un algoritmo basado en aprendizaje por refuerzo, se realizarán pruebas y, en función de los resultados, ajustes hasta obtener un algoritmo de locomoción exitoso. Este proceso se muestra en la figura 4.2.

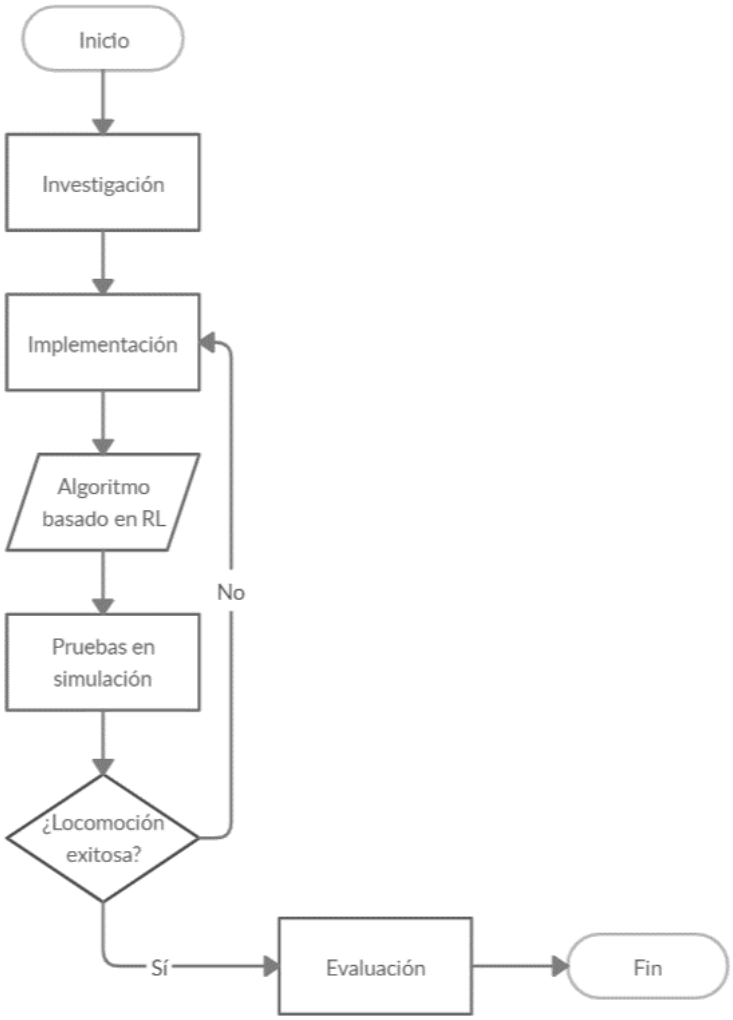

Figura 4.2 - Proceso iterativo de la implementación de un algoritmo basado en RL

Se evaluará el desempeño del algoritmo a través del diseño de casos de prueba. En estos casos de prueba se pretende analizar el comportamiento del sistema de locomoción bajo diferentes escenarios, por ejemplo, el mal funcionamiento de uno de los actuadores. El proceso de pruebas consistirá en realizar el diseño de los casos de prueba, adecuar el modelo o simulador de acuerdo con las precondiciones necesarias, aplicar los algoritmos de locomoción y contrastar los resultados entre los casos de prueba. Asimismo, obtener un indicador de desempeño por ejemplo, la curva de aprendizaje.

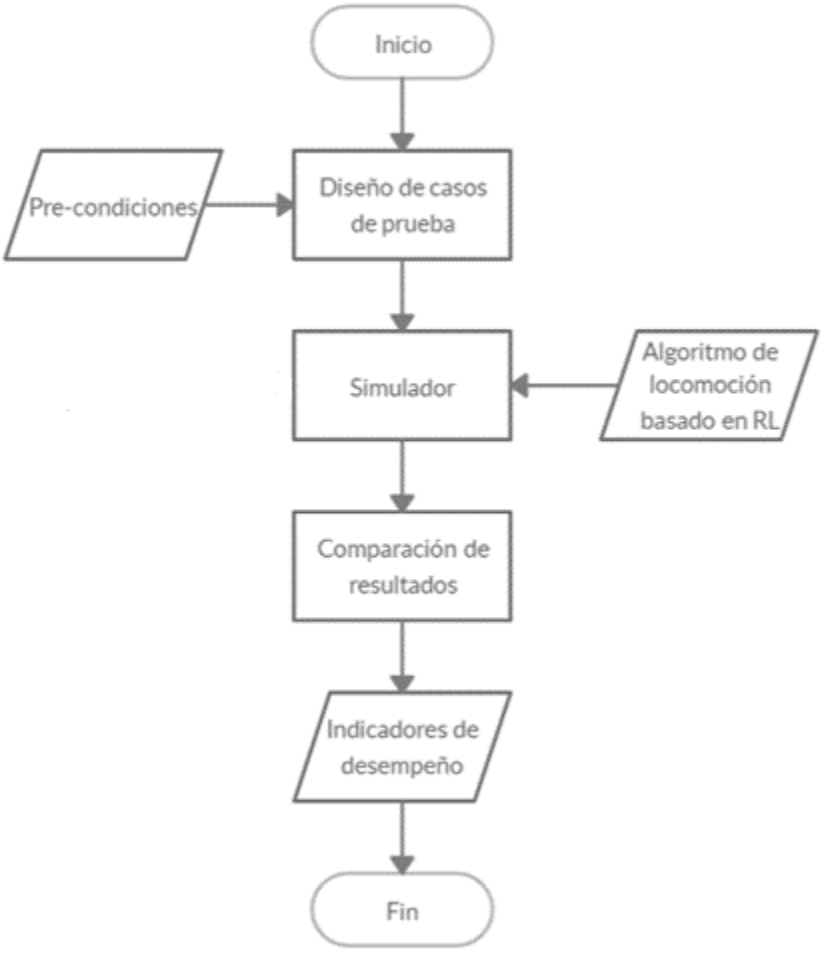

Figura 4.3 - Etapa de pruebas

CAPÍTULO 5 - RESULTADOS

5.1 Herramientas de software

Los algoritmos de aprendizaje por refuerzo se entrenarán y probarán en un entorno simulado. Por definición, el aprendizaje por refuerzo requiere de un agente que explore acciones desconocidas. Si se utiliza un robot real en la etapa de entrenamiento, es probable que se exploren acciones que dañen al robot. Es posible programar restricciones en la lógica del robot para minimizar los daños que pueda sufrir. En el presente trabajo, el entrenamiento y la experimentación se realizarán en un entorno de simulación.

Para simular el robot se utilizarán las siguientes herramientas:

- Gazebo: Ofrece un entorno de simulación para uno o varios robots en entornos cerrados y abiertos. Cuenta con un motor de física para resolver las ecuaciones dinámicas de los modelos, gráficos de alta calidad e interfaces programables (ROS).

- Robot Operative System (ROS): Es un conjunto de bibliotecas que facilitan el desarrollo de robots. Estas bibliotecas son tan variadas que pueden ir desde controladores de hardware hasta algoritmos de aplicación. ROS se utilizará como la interfaz programable con Gazebo. Es decir, desde ROS se enviarán las acciones que deberá ejecutar el robot y se recibirá la lectura de los sensores del robot.

La figura 5.1 muestra una lista de paquetes y plugins que pueden ser utilizados para comunicar ROS con Gazebo.

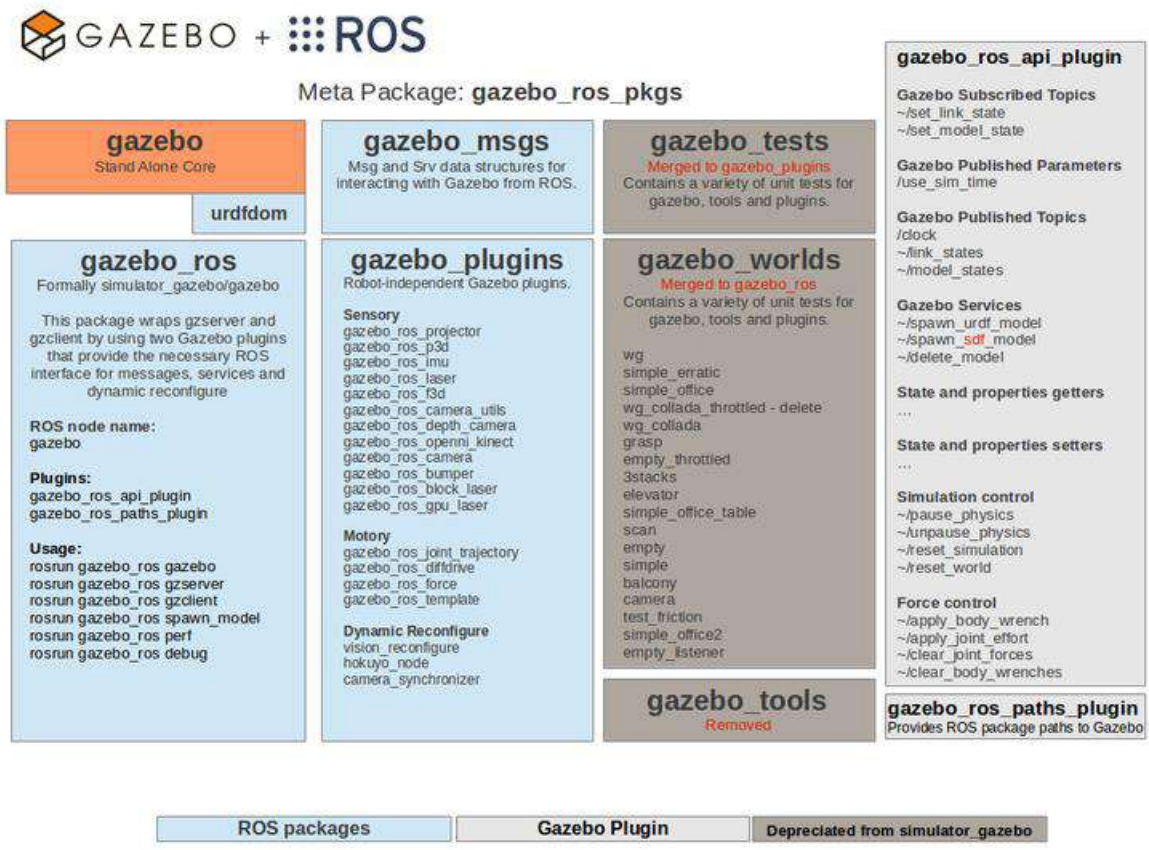

Figura 5.1 – Paquetes y plugins disponibles entre Gazebo y ROS

El simulador es independiente al algoritmo, estos interactúan a través de tópicos que se publican a través de ROS y son leídos por Gazebo, principalmente a través del plugin gazebo_ros_control cuya arquitectura se muestra en la figura 5.2. Este proyecto no se implementó en hardware real. Por lo tanto sólo la parte de simulación está implementada en el proyecto. Sin embargo, dada la arquitectura de ROS, es posible experimentar en hardware real.

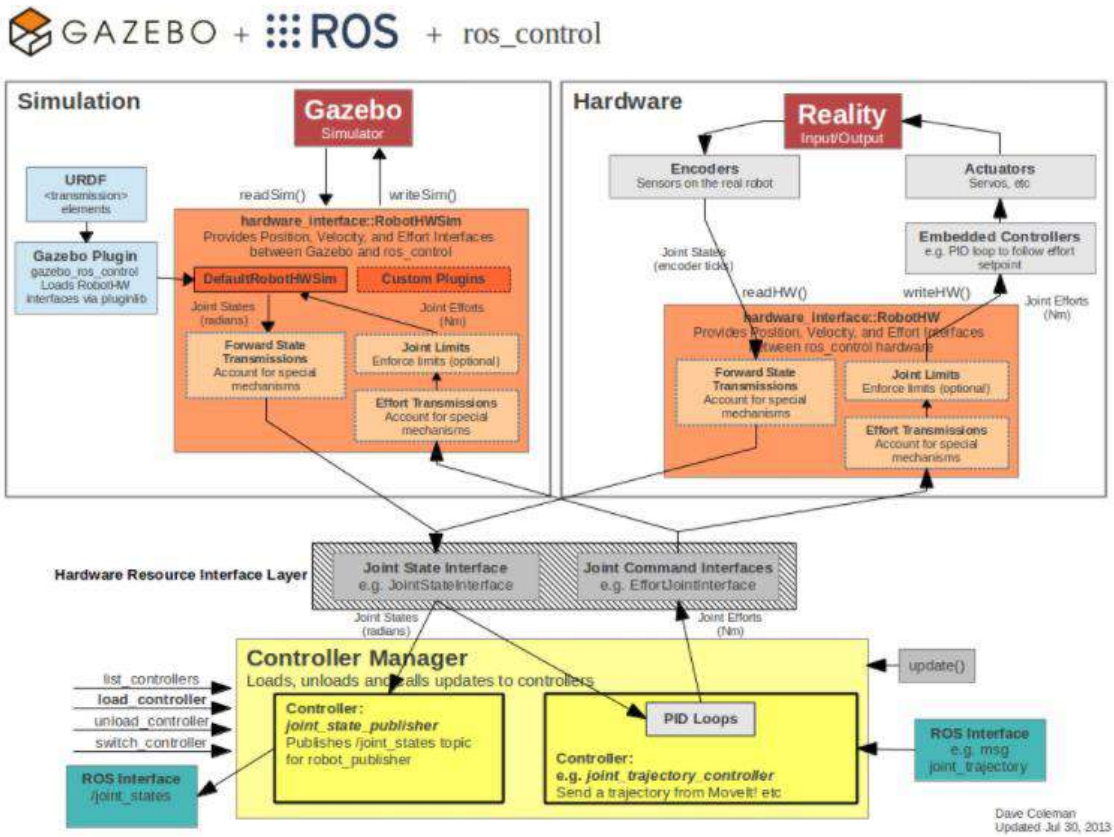

Figura 5.2 – Interfaces entre Gazebo y ROS utilizando el plugin gazebo_ros_control

5.2 Modelado

5.2.1 Diseño e inercias

Para realizar el modelado del robot se utilizaron sólidos regulares cuya masa está uniformemente distribuida para simplificar el análisis y cálculo de inercias. Para el cuerpo del robot se seleccionó un prisma rectangular y cada una de las patas se formó utilizando cilindros como se muestra en la figura 5.3.

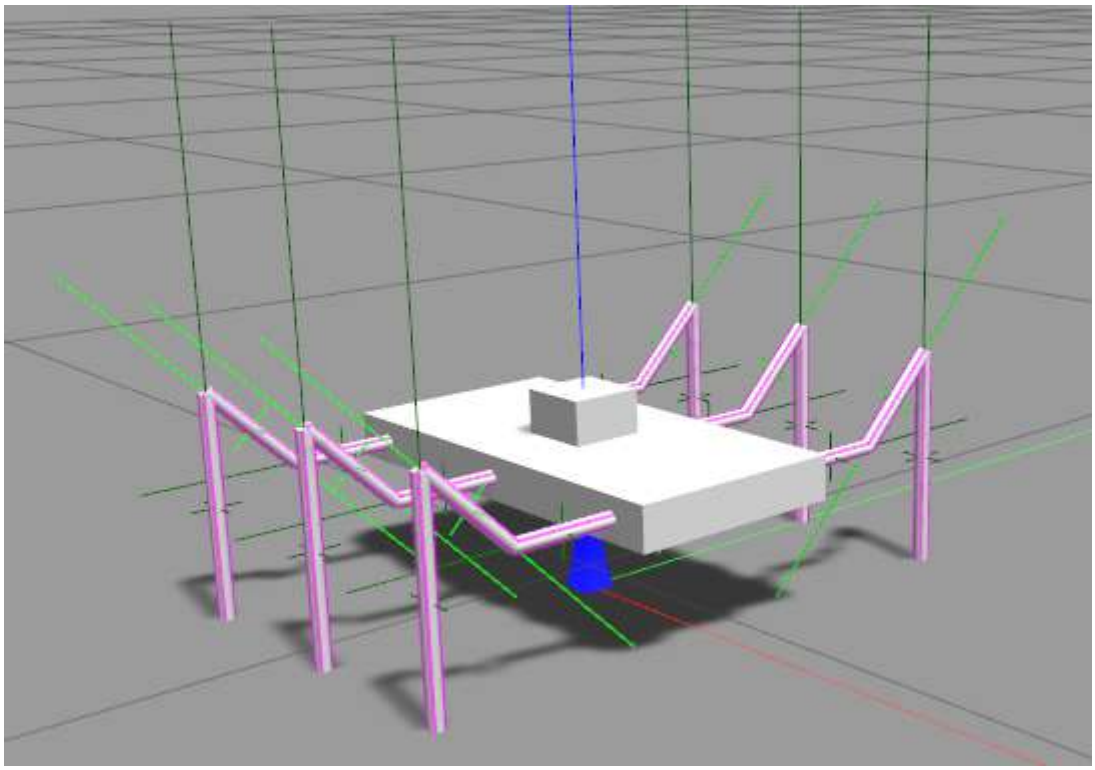

Figura 5.3 – Validación visual del cálculo de inercias en cada pata.

El robot tiene tres grados de libertad por cada pata; éstas se distribuyeron en los extremos más largos del cuerpo. La distribución se encuentra en la figura 5.4. Las características de una de las patas se muestran en la tabla 5.1. Cada eslabón está unido por una junta con otro eslabón pero éstos son invisibles en el modelado 3D, sin embargo, tienen un controlador asociado que permite ejercer torques en las juntas a pesar de no visualizar un motor en las juntas.

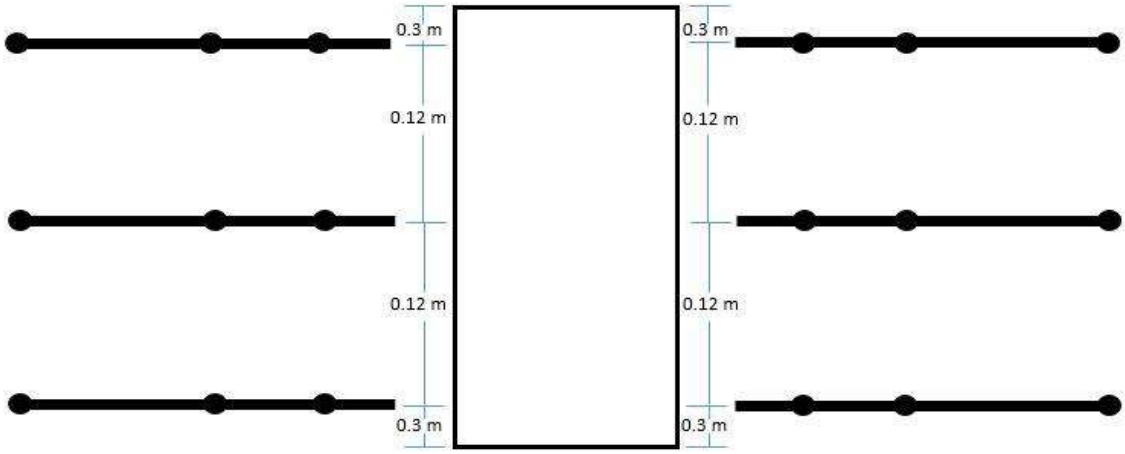

Figura 5.4 – Distribución de las patas a lo largo del cuerpo.

Tabla 5.1 – Parámetros de cada eslabón utilizados el motor de física integrado en Gazebo

Eslabón	Longitud $[m]$	Radio $[m]$	Matriz de inercia $[kg \cdot m^2]$	PID
1	0.07	0.005	$I_{xx} = 0.0000414584$ $I_{xy} = 0$ $I_{xz} = 0$ $I_{yy} = 0.0000414584$ $I_{yz} = 0$ $I_{zz} = 0.000000125$	$P = 100$ $I = 0$ $D = 0$
2	0.08	0.005	$I_{xx} = 0.0000539584$ $I_{xy} = 0$ $I_{xz} = 0$ $I_{yy} = 0.0000539584$ $I_{yz} = 0$ $I_{zz} = 0.000000125$	$P = 100$ $I = 0$ $D = 0$
3	0.15	0.005	$I_{xx} = 0.000188125$ $I_{xy} = 0$ $I_{xz} = 0$ $I_{yy} = 0.000188125$ $I_{yz} = 0$ $I_{zz} = 0.000000125$	$P = 100$ $I = 0$ $D = 0$

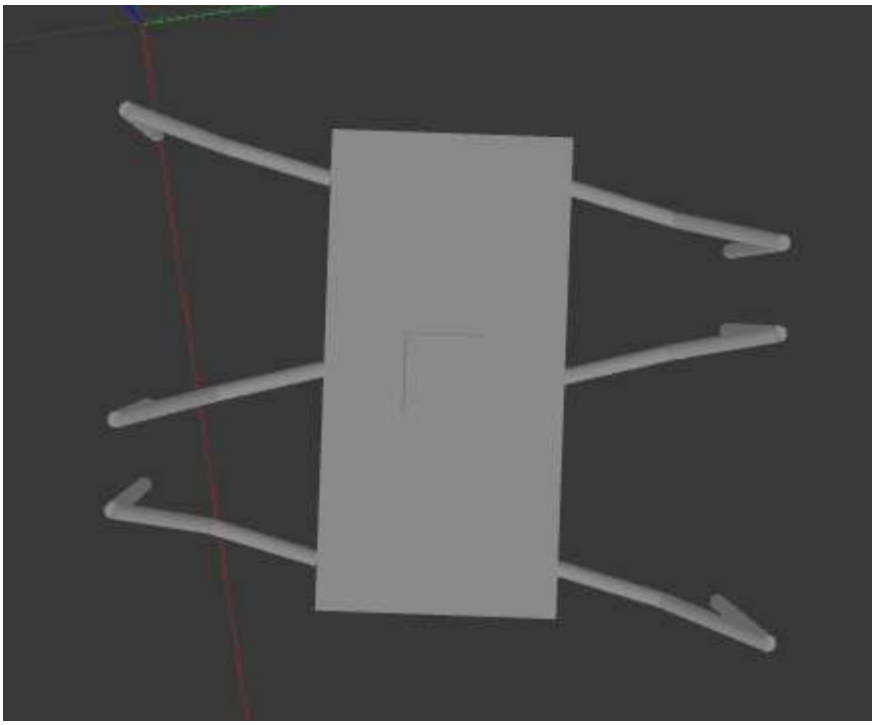

Figura 5.5 – Modelo del robot vista superior.

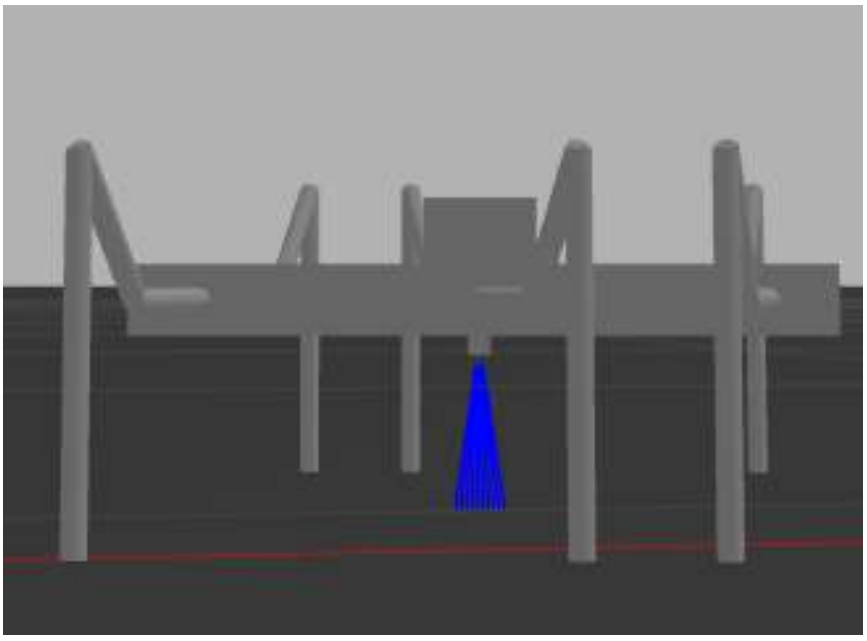

Figura 5.6 – Modelo del robot vista lateral.

5.2.2 Sensores y actuadores

El robot cuenta con un sensor IMU y un láser de proximidad. El sensor IMU entrega en tiempo real la orientación del robot y ayuda a determinar cuándo un episodio ha terminado debido a que el robot se ha caído o el simulador no ha podido resolver algún movimiento y ha detectado alguna singularidad. De manera similar, el sensor de proximidad está debajo del cuerpo del robot apuntando hacia el suelo y mide en tiempo real la distancia que hay entre el cuerpo y el piso.

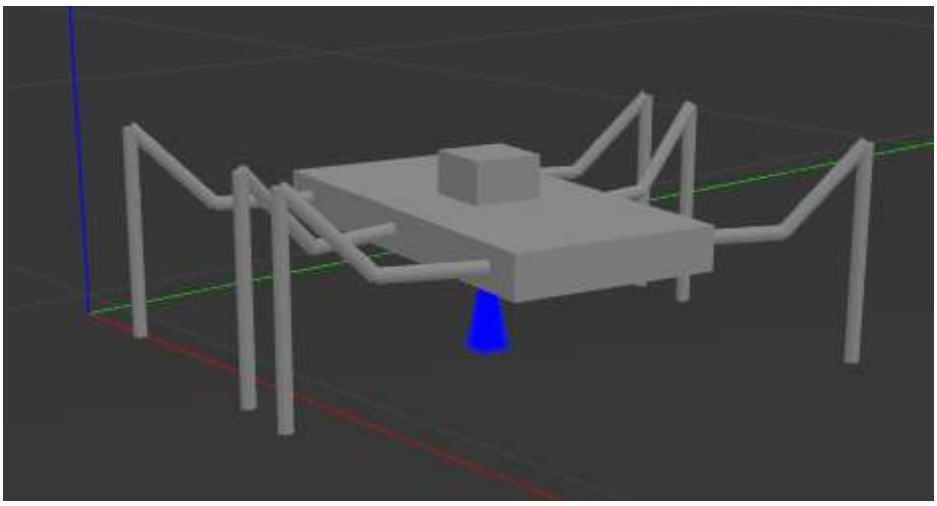

Figura 5.7 – Modelo del robot vista libre. El cubo sobre el cuerpo representa el sensor IMU. La luz azul debajo del cuerpo representa el sensor de proximidad láser.

El robot cuenta con tres grados de libertad por pata, lo que implica que existen dieciocho juntas. Cada junta tiene un controlador de posición asociado e independiente a los demás. Cuenta con un controlador PID como se muestra en la tabla 5.1 y el movimiento está limitado dependiendo la junta para reducir el espacio de trabajo del robot y evitar llevarlo a operar en zonas peligrosas. La tabla 5.2 muestra los límites de cada junta para una pata. Estos límites son idénticos para cada pata.

Tabla 5.2 – Espacio de trabajo de cada junta 6

Junta	Rango de movimiento [rad]
1	(-0.7854, 0.7854)
2	(-1.5708, 1.5708)
3	(0, 2.5)

5.3 *Proximal Policy Optimization*

Las políticas están representadas por una red neuronal perceptrón multi-capa con 56 neuronas en la capa de entrada:

- 18 neuronas representan las posiciones de las juntas
- 18 neuronas representan las velocidades de las juntas
- 18 neuronas representan el torque de las juntas
- Una neurona representa la altura del robot respecto al suelo
- Una neurona representa la distancia recorrida sobre el eje longitudinal

Posteriormente se encuentran cuatro capas ocultas de 512 neuronas cada una. Y finalmente, una capa de salida con 18 neuronas representando las nuevas 18 posiciones de las juntas. Todas las neuronas tienen una función de activación tangente hiperbólica y están totalmente interconectadas.

A diferencia de los experimentos conducidos por Schulman, Wolski, Dhariwal, Radford & Klimov en 2017, en este experimento se utilizaron controladores de posición en lugar de torque. Por lo tanto, se añadió una interpolación lineal a la salida de la red neuronal para convertir la capa de salida a posiciones válidas para los controladores como se muestra en la figura 5.9.

Otra diferencia importante respecto al mismo artículo es el hecho de que la interacción entre el algoritmo principal y el simulador sucede únicamente a través de tópicos. Es decir, el algoritmo no tiene completo control del tiempo de simulación y la simulación opera en un proceso completamente independiente. Por lo tanto se añadió una etapa de sincronización entre el tiempo de simulación y el algoritmo principal, como se muestra en la figura 5.8. De tal forma que el algoritmo sea capaz de informar al simulador que ha terminado su procesamiento para que el simulador espere y actúe sobre la salida del algoritmo y el algoritmo espere a que el simulador ejecute las acciones correspondientes para llevar al robot al siguiente estado.

Para estimar el crítico se utilizó una arquitectura idéntica con la única diferencia en la capa de salida donde se colocó solo una neurona para obtener el escalar que representa el valor del estado. Ambas redes se optimizan de forma separada utilizando *Adam*. Los parámetros adicionales se pueden ver en la tabla 5.3.

Tabla 5.3 – Parámetros del algoritmo PPO

Parámetro	Valor
ε	0.2
λ	0.99
γ	0.95
S (entropía)	0

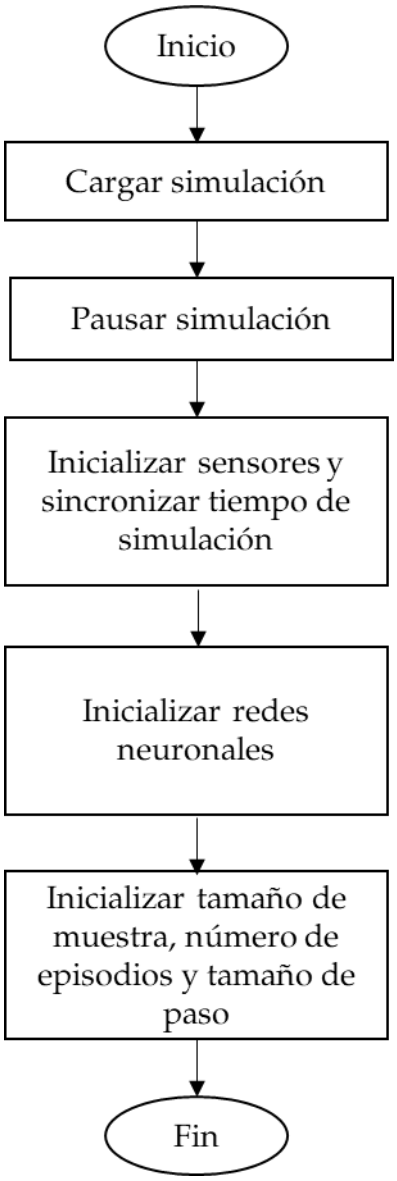

Figura 5.8 – Proceso de inicialización del algoritmo

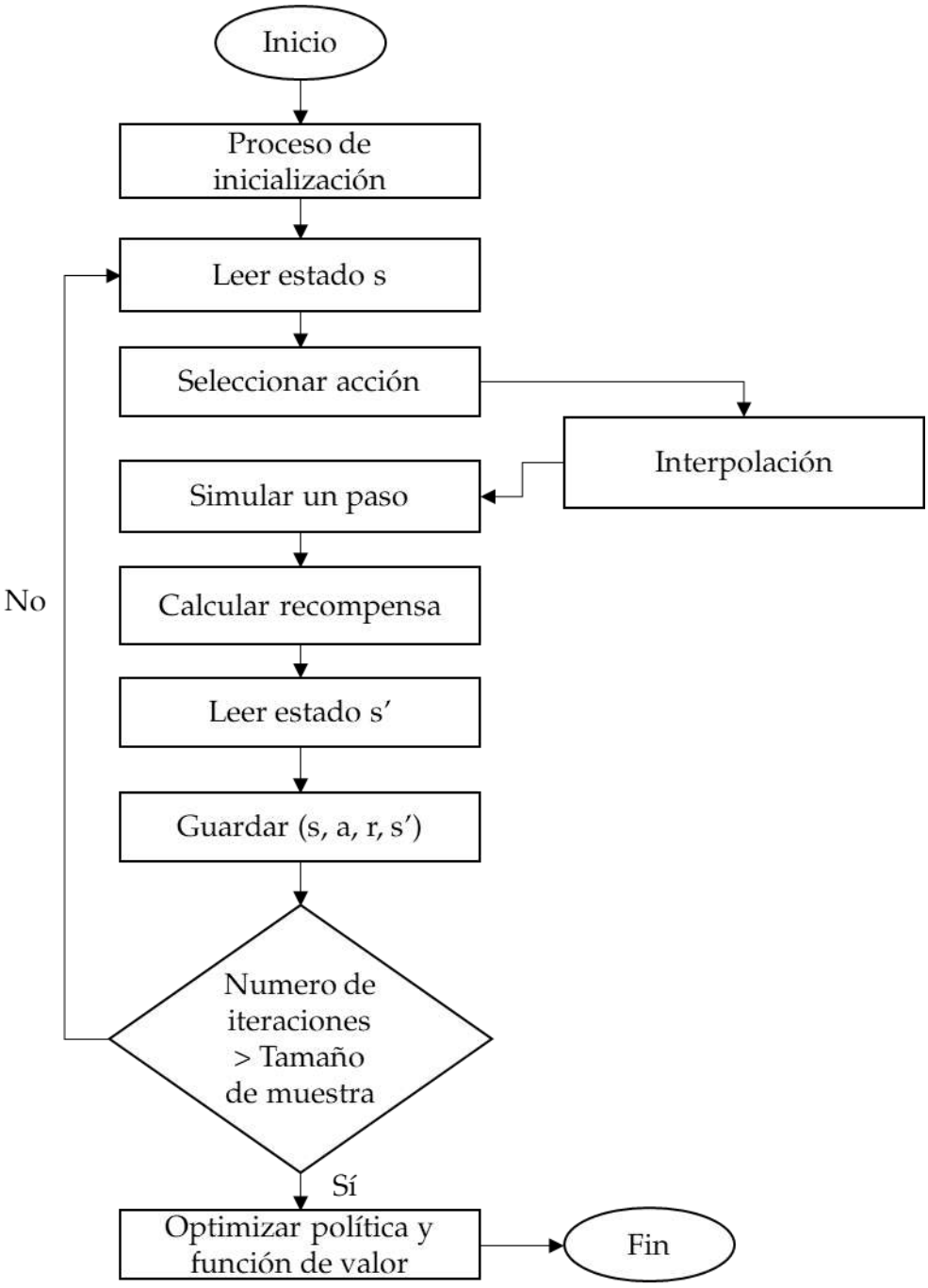

Figura 5.9 – Flujo principal del algoritmo

5.4 Escenarios

Una vez definido el conjunto de herramientas para la experimentación y sus configuraciones se describen a continuación el conjunto de escenarios evaluados. Cada escenario utiliza episodios con 4096 pasos y cada paso es igual a 0.1 segundos.

Los experimentos se condujeron en una computadora con procesador Intel® Core i5 de octava generación y un GPU NVIDIA® GEFORCE 930MX. Con estas características Gazebo es capaz de simular el robot a aproximadamente a 1.5 veces la velocidad real. Para obtener 250 episodios el entrenamiento se tiene que realizar por un aproximado de 24 horas.

Cuando el robot queda en un estado inoperativo, es decir, se cae o el simulador por alguna razón no puede resolver la dinámica, el robot y la simulación son reiniciados a una posición inicial y estable. Lo mismo sucede al término de cada episodio.

5.4.1 Movimiento hacia adelante

El primer escenario consiste en inicializar el robot a partir de una posición estable con el cuerpo separado del piso, sin obstáculos alrededor. El objetivo es ejecutar el algoritmo PPO para obtener la curva de aprendizaje y determinar si existe movimiento a lo largo del eje longitudinal.

En la ecuación 5.1 se definió la señal de recompensa.

$$r(t) = v(t) = \frac{\Delta x}{\Delta t} \tag{5.1}$$

Donde $\Delta t = 0.1s$

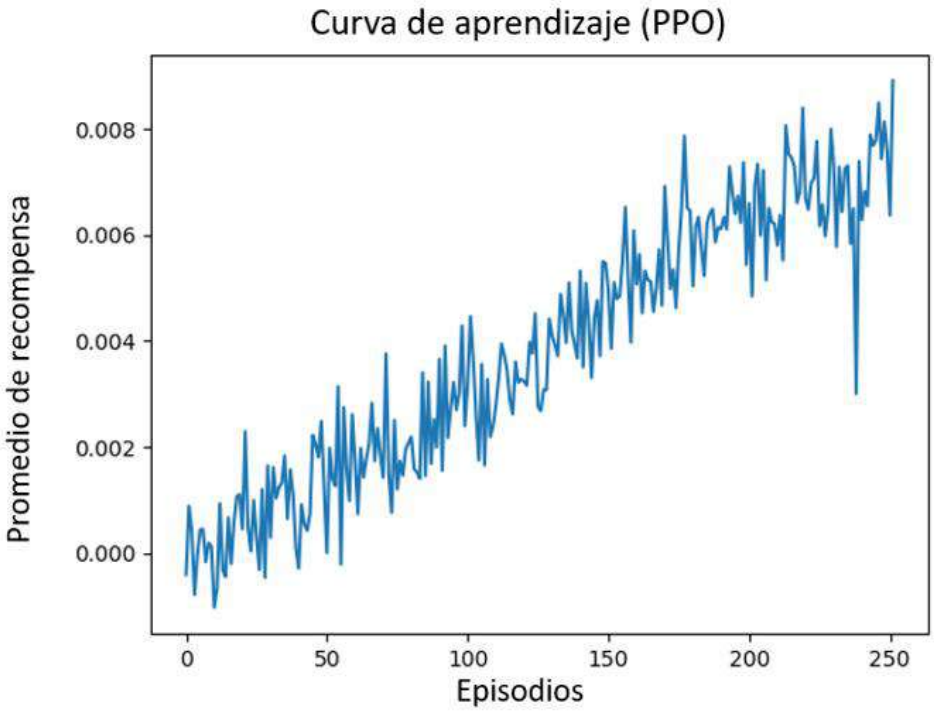

Figura 5.10 – Curva de aprendizaje para movimiento longitudinal con el robot completamente operativo. La tendencia creciente muestra un factor de aprendizaje positivo.

La 5.10 muestra una tendencia clara en el promedio de recompensa acumulada por episodio indicando que las políticas obtenidas cada vez son mejores. Sin embargo, no es un crecimiento monótono. Es claro que se generó en un punto una política mala cerca del episodio 250. La razón detrás puede ser que una acción llevó al robot a un estado difícil de recuperar y por ello la recompensa fue considerablemente más baja que en su vecindad.

Si bien el robot lograba desplazamiento longitudinal, también existía en algunas políticas un ligero desplazamiento lateral, debido a que la señal de recompensa no incluye una penalización por desplazamientos laterales.

5.4.2 Movimiento hacia adelante con una pata dañada

Consiste en evaluar el desempeño de la locomoción en situaciones donde se dañe una de las patas durante la operación. El estado inicial del robot para este escenario es estable con el cuerpo separado del piso y una pata completamente levantada sin hacer contacto con el suelo. Las pruebas se realizaron con la pata izquierda central.

La señal de recompensa es idéntica que en la sección 5.4.1.

Se observa que el patrón es similar a cuando todas las patas están operativas. El agente aún es capaz de encontrar políticas que generen recompensas cada vez mayores impulsando el robot hacia adelante. Nótese que la arquitectura de las redes neuronales no se modificó y por lo tanto el agente sigue produciendo una salida para la pata dañada sin embargo no tiene efecto en la simulación.

En este escenario también se notó el efecto del desplazamiento lateral debido a la falta de penalización por movimientos en otros ejes distintos al longitudinal.

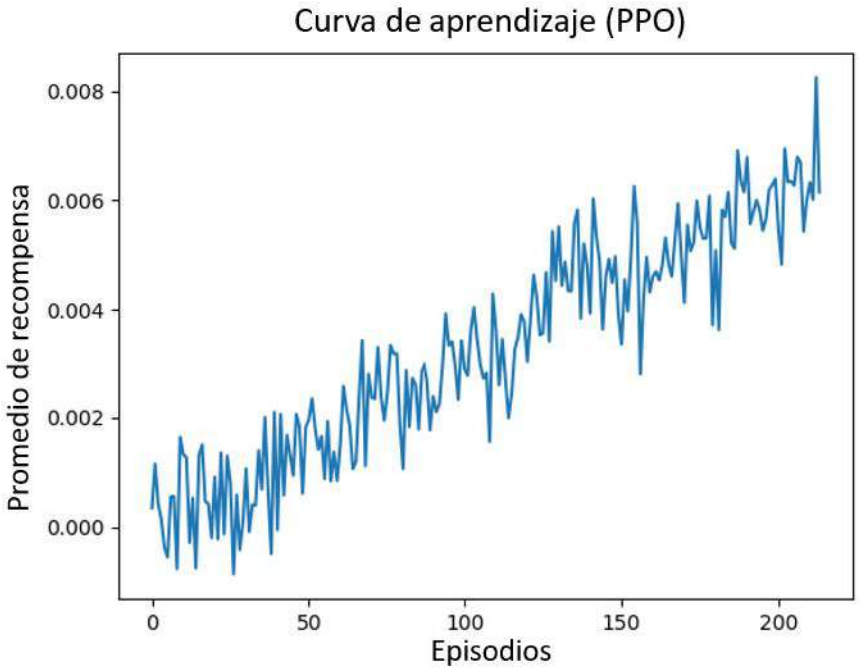

Figura 5.11 – Curva de aprendizaje para movimiento longitudinal con el robot parcialmente operativo. Pata central izquierda deshabilitada.

5.4.3 Rotación del cuerpo

Este caso de uso evaluará el aprendizaje de movimientos donde el cuerpo del robot necesite girar sobre su propio eje, sin la necesidad de trasladar el cuerpo. Se parte de condiciones similares a los escenarios anteriores. Una posición estable con el cuerpo despegado del suelo.

La señal de recompensa para este escenario se modifica para utilizar la velocidad angular, como se muestra en la ecuación 5.2

$$r(t) = \omega(t) = \frac{\Delta\phi}{\Delta t} \tag{5.2}$$

Donde $\Delta t = 0.1s$

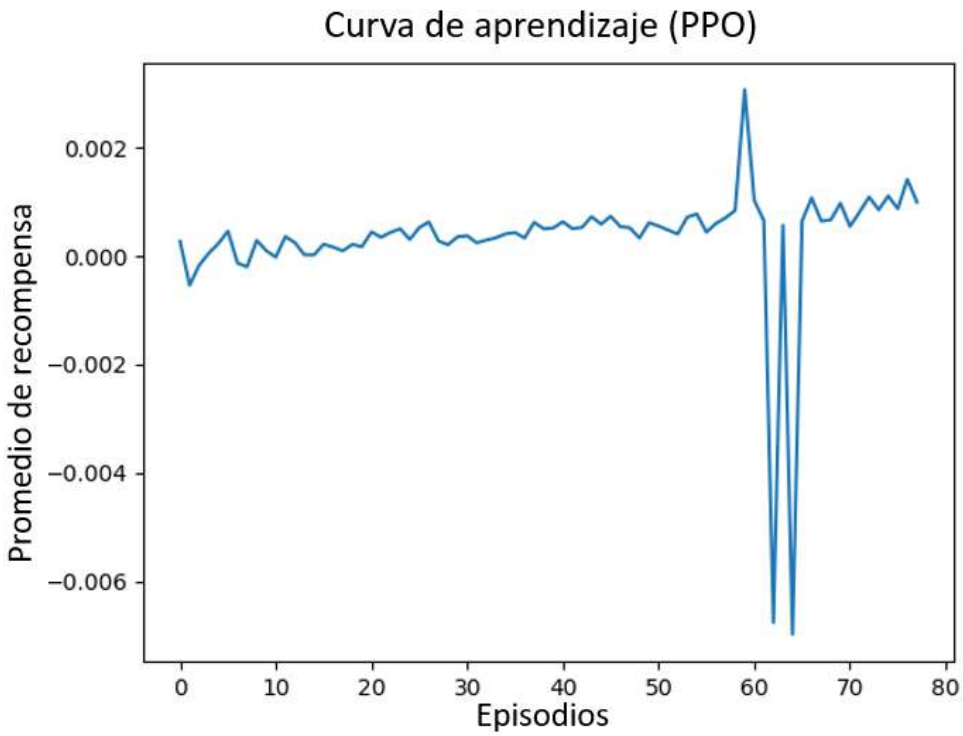

Figura 5.12 – Curva de aprendizaje para rotación sobre el eje z. Robot completamente operativo. Tendencia positiva con dos políticas catastróficas.

Los resultados para este experimento son considerablemente peores a los escenarios anteriores. Se notaron dos políticas catastróficas que llevaron al robot a obtener recompensas negativas, es decir, el robot giraba en la dirección contraria. También se notó como en los casos anteriores que existía desplazamiento lateral para todas las políticas además del giro esperado. A pesar de las dos malas políticas el algoritmo fue capaz de restablecer la tendencia de crecimiento en la función de recompensa. Logrando en las últimas políticas girar en el sentido deseado.

CAPÍTULO 6 - CONCLUSIONES Y TRABAJO FUTURO

Se cumplieron los tres objetivos. Primero, se realizó un modelado de un robot hexápodo. El modelo se construyó a través de un formato estándar conocido como URDF. Este formato es un archivo XML que describe los eslabones, juntas y conexiones entre ambos. Se especifican las características geométricas y físicas de cada eslabón, incluyendo su matriz de inercias para poder integrar el modelo a un motor de física. Fue posible especificar sensores como IMU y de proximidad que son interpretados por el simulador.

Segundo, se utilizó el simulador Gazebo. A través del modelo URDF, fue posible integrar el modelo del robot al motor de física ODE que utiliza el simulador. Se construyeron las interfaces necesarias para integrar Gazebo con ROS utilizando el plugin gazebo_ros_control. Se publicaron los mensajes que permiten comunicar los algoritmos de aprendizaje por refuerzo con el simulador y se escribieron decodificadores para recibir y entender los mensajes del simulador hacia el algoritmo.

Tercero, se utilizó el algoritmo de aprendizaje por refuerzo *Proximal Policy Optimization*. Este algoritmo es uno de los más cercanos al estado del arte que ha probado mejores resultados en varios trabajos de locomoción y ha sido poco probado en robots hexápodos. Este trabajo presentó tres escenarios en los que se aplica este algoritmo con resultados positivos. Se muestran curvas de aprendizaje con tendencia a generar buenas políticas. Además de estar implementado en un ambiente con potencial a escalar a un sistema robótico real.

Como trabajo futuro se proponen los siguientes puntos:

1. Penalizar el modelo de la señal de recompensa para evitar desplazamientos en direcciones no deseadas durante el entrenamiento.

2. Integrar al espacio de estados las patas que están en contacto con el suelo. Basados en el estado del arte, se cree que pueden aportar información que acelere el proceso de aprendizaje.

3. Implementar distintas arquitecturas de redes neuronales para el actor y el crítico y hacer un estudio comparativo de los resultados.

4. Implementar cómputo paralelo para acelerar el aprendizaje.

5. Implementar el algoritmo en un robot físico. La arquitectura propuesta funciona a través de la red local y sockets gracias al diseño de ROS. Es posible construir un robot físico y conectar el algoritmo con el robot a través de ROS sin cambios mayores al algoritmo.

REFERENCIAS

Aggarwal, C. C. (2018). Neural networks and deep learning. Springer, 10, 978-3.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6), 26-38.

Barfoot, T. D., Earon, E. J., & D'Eleuterio, G. M. (2006). Experiments in learning distributed control for a hexapod robot. Robotics and Autonomous Systems, 54(10), 864-872.

Barrientos, A., Peñin, L. F., Balaguer, C., & Aracil, R. (2007). Fundamentos de robótica (2nd ed., pp. 93-126). Madrid: McGraw-Hill.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. IEEE transactions on systems, man, and cybernetics, (5), 834-846.

Beaber, S. I., Zaghloul, A. S., Kamel, M. A., & Hussein, W. M. (2018, November). Dynamic modeling and control of the hexapod robot using matlab simmechanics. In *ASME International Mechanical Engineering Congress and Exposition* (Vol. 52033, p. V04AT06A036). American Society of Mechanical Engineers.

Critchlow, A. J. (1985). Introduction to robotics. New York, NY: Macmillan Pub Co.

Delcomyn, F., & Nelson, M. E. (2000). Architectures for a biomimetic hexapod robot. Robotics and Autonomous Systems, 30(1-2), 5-15.

Devjanin, E. A., Gurfinkel, V. S., Gurfinkel, E. V., Kartashev, V. A., Lensky, A. V., Shneider, A. Y., & Shtilman, L. G. (1983). The six-legged walking robot capable of terrain adaptation. Mechanism and Machine Theory, 18(4), 257-260.

Du, K. L., & Swamy, M. N. (2014). Neural networks and statistical learning (1st ed., pp. 1-12). London: Springer

García-López, M., Gorrostieta-Hurtado, E., Vargas-Soto, E., Ramos-Arreguín, J., Sotomayor-Olmedo, A., & Morales, J. M. (2012). Kinematic analysis for trajectory generation in one leg of a hexapod robot. *Procedia Technology*, *3*, 342-350.

Gorrostieta, E., & Vargas Soto, E. (2008). Algoritmo difuso de locomoción libre para un robot caminante de seis patas. *Computación y Sistemas*, *11*(3), 260-287.

Graham, D. (1972). A behavioural analysis of the temporal organisation of walking movements in the 1st instar and adult stick insect (Carausius morosus). Journal of Comparative Physiology, 81(1), 23-52.

Hong, J., Tang, K., & Chen, C. (2017). Obstacle avoidance of hexapod robots using fuzzy Q-learning. In 2017 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 1-6). IEEE.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of artificial intelligence research, 4, 237-285.

Kirchner, F. (1998). Q-learning of complex behaviours on a six-legged walking machine. Robotics and Autonomous Systems, 25(3-4), 253-262.

Klein, C. A., Olson, K. W., & Pugh, D. R. (1983). Use of force and attitude sensors for locomotion of a legged vehicle over irregular terrain. The International Journal of Robotics Research, 2(2), 3-17.

Konen, K., Korthals, T., Melnik, A., & Schilling, M. (2019). Biologically-Inspired Deep Reinforcement Learning of Modular Control for a Six-Legged Robot. In 2019 IEEE International Conference on Robotics and Automation Workshop on Learning Legged Locomotion Workshop. Montreal.

Kume, A., Matsumoto, E., Takahashi, K., Ko, W., & Tan, J. (2017). Map-based multi-policy reinforcement learning: enhancing adaptability of robots by deep reinforcement learning. arXiv preprint arXiv:1710.06117.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

Nonami, K., Huang, Q., Komizo, D., Fukao, Y., Asai, Y., Shiraishi, Y., ... & Ikedo, Y. (2003). Development and control of mine detection robot COMET-II and COMET-III. JSME International Journal Series C Mechanical Systems, Machine Elements and Manufacturing, 46(3), 881-890.

Peters, J., & Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. Neural networks, 21(4), 682-697.

Peters, J., & Schaal, S. (2006, October). Policy gradient methods for robotics. In 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 2219-2225). IEEE.

Pfeiffer, F., Eltze, J., & Weidemann, H. J. (1995). The TUM-walking machine. Intelligent Automation & Soft Computing, 1(3), 307-323.

Proximal Policy Optimization (openai.com)

Ross, S. M. (2014). Introduction to probability models. Academic press.

Shahriari, M., & Khayyat, A. A. (2013, August). Gait analysis of a six-legged walking robot using fuzzy reward reinforcement learning. In 2013 13th Iranian Conference on Fuzzy Systems (IFSC) (pp. 1-4). IEEE.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). Trust region policy optimization. In *International conference on machine learning* (pp. 1889-1897). PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484.

Sutton, R. S. (1990, June). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Machine Learning Proceedings 1990. https://doi.org/10.1016/B978-1-55860-141-3.50030-4

Sutton, R. S., & Barto, A. G. (1998). Introduction to reinforcement learning (Vol. 2, No. 4). Cambridge: MIT press.

Tedeschi, F., & Carbone, G. (2015). Hexapod walking robot locomotion. In *Motion and Operation Planning of Robotic Systems* (pp. 439-468). Springer, Cham.

Tedeschi, F., & Carbone, G. (2014). Design issues for hexapod walking robots. Robotics, 3(2), 181-206.

Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 30, No. 1).

Watkins, C. J. C. H. (1989). Learning from delayed rewards. (Doctoral dissertation, King's College, Cambridge, UK). Retrieved from http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.

Youcef, Z., & Pierre, C. (2004, May). Control of the trajectory of a hexapod robot based on distributed Q-learning. In 2004 IEEE International Symposium on Industrial Electronics (Vol. 1, pp. 277-282). IEEE.

Yu, T. K., Chieh, Y. M., & Samani, H. (2018). Reinforcement learning and convolutional neural network system for firefighting rescue robot. In MATEC Web of Conferences (Vol. 161, p. 03028). EDP Sciences.

Yu, W., Turk, G., & Liu, C. K. (2018). Learning symmetric and low-energy locomotion. ACM Transactions on Graphics (TOG), 37(4), 144.

Zhao, Y., Chai, X., Gao, F., & Qi, C. (2018). Obstacle avoidance and motion planning scheme for a hexapod robot Octopus-III. Robotics and Autonomous Systems, 103, 199-212.

Zhou, C. (2002). Robot learning with GA-based fuzzy reinforcement learning agents. Information Sciences, 145(1-2), 45-68.