Procesos de Decisión de Markov Aplicados a la Locomoción de Robots Hexápodos

Por

L. C. C. German Cuaya Simbro

Tesis sometida como requisito parcial para obtener el grado de

Maestría en Ciencias Computacionales

en el Instituto Nacional de Astrofísica, Óptica y Electrónica.

Supervisada por

Dra. Angélica Muñoz Meléndez
Coordinación de Ciencias Computacionales INAOE

Tonantzintla, Puebla
2007

Resumen

En este trabajo se aborda el problema de locomoción de robots caminantes o polípedos. El diseño de modelos de locomoción para este tipo de robots es un problema abierto en robótica móvil, debido a la complejidad para controlar robots móviles con al menos una decena de grados de libertad. En este trabajo se presenta un modelo para el control de un robot hexápodo hexagonal o circular, i.e. un robot hexápodo cuya base tiene la forma de un hexágono regular o de un círculo con extremidades rodeando la base.

Se buscó diseñar modelos descriptivos para el control de un robot hexápodo de base hexagonal aplicando Procesos de Decisión de Markov (MDPs) Descentralizados, que subdividen un problema en subproblemas tratados por MDPs especializados organizados en capas, según prioridades establecidas por el diseñador; así como MDPs organizados en dos capas que no dependen de jerarquías rígidas ni de supervisores para operar.

Se proponen dos modelos, el modelo DCAD (Decisión Centralizada y Acción Descentralizada) y el modelo DDAD (Decisión Descentralizada y Acción Descentralizada), dichos modelos permitieron definir un control eficiente para el robot hexápodo mencionado previamente. Los modelos de control permiten al robot desplazarse y mantenerse estable en diversas condiciones dentro de un ambiente semi-estructurado. Se presentan resultados de la aplicación de dichos modelos de control en un robot simulado en el ambiente de simulación Webots® y también se presentan los resultados de la aplicación de uno de los modelos para controlar un robot hexápodo hexagonal físico.

Se realiza una comparativa de desempeño y robustez de ambos modelos en simulación. El modelo DCAD tuvo un mejor comportamiento en las pruebas de desempeño mientras que el modelo DDAD mostró un mejor desempeño en las pruebas de robustez. Esto significa que ningún modelo es en definitiva mejor que el otro. Puede afirmarse que para ambientes semi o estructurados, en donde la integridad del robot no está en juego, el modelo DCAD es adecuado. En cambio, para ambientes poco estructurados en donde la operación del robot se vea comprometida, esto es, que existan perturbaciones físicas, el modelo DDAD puede ser adecuado.

Abstract

This work concerns with the locomotion of legged robots. The design of locomotion models for legged robots is an open problem in mobile robotics, due to the complexity in controlling mobile robots with at least 10 DOF. In this work, a model for controlling a hexagonal or circular hexapod robot is presented. This is a hexapod robot whose legs are located on the six faces of its regular hexagonal or circular body.

We are interested in descriptive models to control such a hexapod robot applying Decentralized Markov Decision Processes, Markov Decision Processes (MDPs) in which a problem is divided into several subproblems that are modeled as specialized MDPs organized according to priorities established by the designer, as well as MDPs comprised of two levels that do not rely on rigid hierarchies nor supervisors to operate.

Two control models are proposed, a model that depends on centralized decision and decentralized action (DCAD from its Spanish name) and second model that depends on decentralized decision and decentralized action (DDAD from its Spanish name). The proposed models enabled us to define an efficient control for the hexapod robot previously mentioned. Control models enable the robot to walk and remain stable in a semi-structured environment. Various results of the application of both models in a simulated robot in the environment Webots® are presented. Results of the application of one model to control a physical hexagonal hexapod robot are also presented.

A comparison regarding performance and robustness of both control models is presented. The DCAD model performs better than the DDAD model in

performance tests, whereas the DDAD model is better than the DCAD model in the robustness tests. This means that there is not a definitive best model. For structured or semi-structured environments, where the physical integrity of the robot is not compromised, the DCAD model is best suited. However, for non structured environments, where the robot operation is at risk and there are physical perturbations that can affect its operation, the DDAD model is adequate.

Agradecimientos

El autor le agradece al CONACyT el apoyo otorgado a través de la Beca para estudios de Maestría #201848. También al INAOE por proporcionar los espacios e infraestructura necesarios para la realización de los estudios de Maestría.

Agradezco a mis sinodales por la comprensión, paciencia y atención prestada para la culminación de mi tesis, así como a mi asesora de tesis por todo el apoyo y confianza que me ha brindado y a todas las personas que me ayudaron de cualquier manera, ya que sin esa ayuda hubiese sido muy complicado concluir mis estudios de maestría.

Índice general

Índice de figuras

Índice de tablas

Notación y abreviaturas

Termino	Significado	Traducción al español
DTP	**D**ecision-**T**heoretic **P**lanning	Planificación basada en Teoría de Decisiones
RL	Reinforcement Learning	Aprendizaje por Refuerzo
MDP	**M**arkov **D**ecision **P**rocess	Proceso de Decisión de Markov
HMDP	Hierarchical Markov Decision Proceses	Proceso de Decisión de Markov Jerárquico
π	Policy	Política
π^*	Optimal policy	Política óptima
S **/** *s*	State set / individual state	Conjunto de estados / estado individual
A **/** *a*	Action set / individual action	Conjunto de acciones / acción individual
ϕ	Transition function (Transition model)	Función de transición (Modelo de transición)
R	Reward function (Reward model)	Función de recompensa (Modelo de recompensa)
p(a|b)	Conditional probability of *a* given *b*	Probabilidad condicional de *a* si *b*
p(a|b,c)	Conditional probability of *a* given *b* and *c*	Probabilidad condicional de *a* si *b* y *c*
V_π	Value function of policy π	Función de valor para la política π
*V**	Optimal value function	Función de valor óptimo
r(s,a)	reward-value of action *a* at state *s*	Valor *recompensa* de la acción *a* en el estado *s*
$\max_a(f(a))$	Maximum value of *f(a)*	Valor máximo de *f(a)* para cualquier *a*
$\arg\max_a(f(a))$	Value of a that maximizes *f(a)*	Valor de a que maximiza *f(a)*
λ	Discount factor	Factor de descuento
[URL_ref]	Referencia en línea ver sección correspondiente	

CAPÍTULO I: Introducción

1.1 Motivación y problemática

Los sistemas robóticos modernos son cada vez más poderosos en términos del número de sensores utilizados y de movilidad. El progreso tecnológico actual hace posible que diversos tipos de robots realicen tareas en que apoyen o sustituyan a humanos. Robots con poca o nula autonomía han ganado terreno y son utilizados en la realización de tareas en ambientes controlados, por ejemplo en fábricas e industrias. Por su parte, robots móviles y autónomos realizan cada vez más tareas por nosotros en entornos complejos, como los que se encuentran frecuentemente en la exploración de zonas accidentadas o colapsadas y en la robótica de servicio.

Sin embargo, aún cuando hay muchas aplicaciones que se pueden solucionar utilizando robots adecuadamente diseñados y programados, equipados con las herramientas necesarias y con la destreza de un operador, persisten tareas que no pueden ser resueltas por robots debido a la dificultad de acceso a la zona en donde debe resolverse el problema. Estas dificultades conciernen, por ejemplo, pendientes, obstáculos a sobrepasar o evitar, terrenos irregulares, etc.

Estas dificultades y la forma de enfrentarlas son estudiadas en el área de robótica, en la cual se investiga la forma de diseñar sistemas de locomoción en un nivel bajo, y de localización y planeación en un alto nivel. Hay trabajos en los que se muestran soluciones interesantes a este problema, por ejemplo el uso de manipuladores remotos (Armada et al. 2005). Algunas otras soluciones, no menos interesantes, consisten en proveer de un sistema de transporte a los manipuladores tele-operados, resultando en vehículos con

ruedas o vehículos con orugas, y recientemente robots polípedos o caminantes.

Así, los roboticistas se han interesado en el diseño y control de robots polípedos, robots con patas, para solucionar la problemática de acceso y movilidad en ambientes peligrosos o desconocidos. En particular, los robots hexápodos autónomos pueden ser útiles en situaciones donde el ambiente es hostil, e. g. en la exploración de volcanes (Bares 1999), en las profundidades del océano, en zonas de desastre y de cuerpos extraterrestres; al igual que en tareas militares tales como la búsqueda de puntos de interés en el campo de batalla o la búsqueda de minas explosivas.

Un reto importante en el diseño de robots polípedos es lograr la autonomía completa de las capacidades del robot para adaptarse a los cambios del ambiente y al aumento de sus propias capacidades con la incorporación de nuevos sensores. Esto motiva a los diseñadores a hacer los sistemas de control de robots tan flexibles como sea posible utilizando algoritmos de aprendizaje automático (c. f. sección 3.4). Además de ayudar a desarrollar robots autónomos y flexibles, el aprendizaje reduce la ingeniería humana requerida para controlar robots en general.

1.1.1 Estabilidad estática y dinámica

En robótica existen dos formas básicas de usar los efectores de un robot, la primera es usar efectores para mover al robot mismo y la segunda forma es usar efectores para mover otros objetos. El primer caso se estudia en el área de locomoción, mientras el segundo caso en el área de manipulación. De esta forma se divide el estudio en robótica en dos grandes categorías, robótica móvil y robótica de manipuladores [URL_Historia].

Muchos tipos de efectores y actuadores* pueden ser usados para mover un robot, entre las categorías más conocidas que podemos encontrar tenemos: patas para caminar, reptar, subir, brincar; llantas para rodar; brazos para pivotear, reptar, subir; aletas para nadar; alas para volar o planear.

Mientras que la mayoría de los animales terrestres utilizan patas para desplazarse, en robótica la locomoción usando patas es un problema muy difícil de resolver, especialmente cuando se compara con la locomoción con ruedas. Esta dificultad estriba en que todo robot móvil necesita ser estable, es decir, no debe desequilibrarse ni caerse fácilmente. Existen dos clases de estabilidad, *estabilidad estática* y *estabilidad dinámica*. La estabilidad estática es la capacidad que tiene un objeto de volver a su posición original. La estabilidad dinámica se refiere a cuánto tiempo se tarda un objeto en regresar a su posición original [URL_Estabilidad].

Un robot estáticamente estable puede permanecer erguido sin caerse cuando no se encuentra en movimiento. Ésta es una característica útil y deseable, pero difícil de alcanzar, debido a que se requiere que exista un cierto número de patas o ruedas en el robot para proporcionar suficientes puntos de soporte. Las personas no son estáticamente estables, este equilibrio es en gran parte inconsciente y es aprendido durante los primeros años de vida y se logra gracias a los nervios, músculos y tendones del cuerpo humano.

Podría pensarse que con más patas, la estabilidad estática puede alcanzarse de manera más simple. Lo anterior no es totalmente cierto, debido a que para que un robot se encuentre estable, el centro de gravedad (CDG) del robot debe ser soportado por un polígono. Este *polígono* o *geometría de apoyo*, es

*La palabra actuadores se usará en esta tesis como sinónimo de activadores o accionadores. Los tres términos son usados indistintamente como traducción del término en inglés *actuators*.

básicamente la unión de todos los puntos de soporte sobre la superficie en que los que el robot se apoya. Así, en un robot de dos patas, el polígono es realmente una línea, y como se mencionará más adelante en esta misma sección, este tipo de robots no es estáticamente estable. Ahora bien, si consideramos robots con tres o múltiplos de tres patas, sus patas se organizan de forma alternada en una configuración de trípode y su cuerpo queda distribuido sobre tal configuración. Tales robots producen un polígono estable de soporte. Así, estos robots son estáticamente estables.

Ahora bien, una vez que el robot es estáticamente estable el problema no está totalmente resuelto. Las preguntas que habrá que responder a continuación son: ¿qué sucede cuando un robot estáticamente estable levanta una pata e intenta moverse?, ¿el centro de gravedad del robot permanece dentro del polígono de soporte antes mencionado?. Para responder a la primera cuestión se debe realizar un análisis más profundo, debido que este aspecto está completamente ligado al problema de locomoción en robótica, la última pregunta puede responderse considerando si el centro de gravedad permanece o no dentro de ese polígono dependiendo de la geometría del robot. Ciertas *geometrías de apoyo* robóticas hacen posible que un robot polípedo permanezca siempre estáticamente estable, e. g. triángulos, rectángulos, hexágonos y en general polígonos.

Una aspecto importante a considerar para que un robot mantenga una posición estáticamente estable, es que el peso de una pata es insignificante comparado con el peso del cuerpo, e. g. 10% del peso total del cuerpo, de modo que el CDG del robot no será afectado por la oscilación de la pata. De acuerdo con esta suposición, el paso estático convencional es diseñado para mantener el CDG del robot dentro del polígono de soporte. La figura 1.1 ilustra la posición del CDG en el caminar de un robot de cuatro patas.

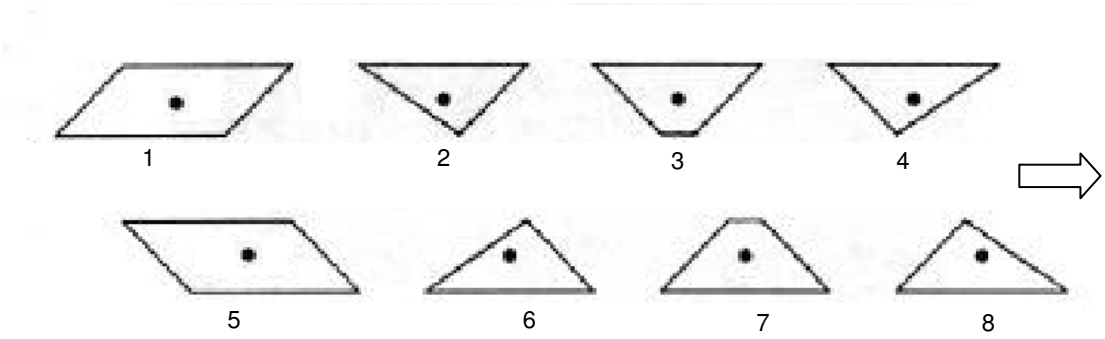


Figura 1.1 Posición del CGD durante el movimiento de un robot con cuatro patas

Una alternativa a la estabilidad estática es la estabilidad dinámica que permite que un robot o animal sea estable mientras se mueve. Por ejemplo, los robots brincadores con una sola pata son dinámicamente estables, i. e. saltan en un mismo lugar o en distintas posiciones para no caerse, pero no pueden parar y permanecer de pie. Esto es el problema de balance del péndulo invertido.

Para determinar cómo lograr que un robot polípedo sea estable al desplazarse, es importante preguntarse cuántas patas se encuentran en el aire durante el movimiento del robot, es decir, mientras genera el paso. Seis patas es un número conocido que permite una locomoción estable, si las mismas tres patas no adyacentes se mueven a la vez, dicho paso se conoce como el paso de trípode alternado, si las patas varían, se llama paso ondulatorio, dichos pasos se ilustran en la figura 1.2.

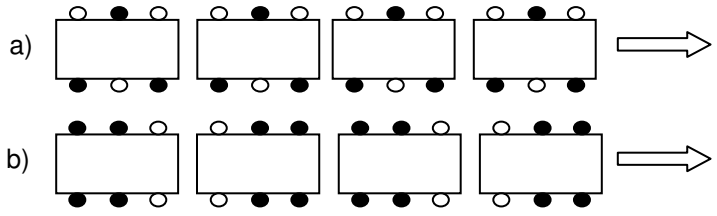


Figura 1.2 a) Paso de trípode alternado, b) paso ondulatorio, donde los círculos negros indican que la pata se encuentra posada en la superficie de apoyo y los círculos blancos indican que la pata se encuentra despegada de la superficie de apoyo

Un robot con seis patas puede levantar tres patas no adyacentes a la vez para moverse hacia adelante, y todavía conservará una estabilidad estática. Este tipo de robot puede lograr un movimiento dinámicamente estable debido a que utiliza un paso en forma de trípode alternado, el cual es un patrón común en la naturaleza para insectos con seis o más patas. Las tres patas fijas en tierra mantienen al robot estáticamente estable.

De manera general, en la naturaleza encontramos animales que tienen un sistema de locomoción usando seis patas con un paso en forma de trípode, pero estos animales son ligeros y pequeños, por ejemplo, las cucarachas mueven sus patas en forma de trípode y pueden hacerlo muy rápidamente. Los insectos con más de seis patas, e. g. los ciempiés y los milpiés, utilizan el paso ondulatorio. Sin embargo, cuando estos insectos se mueven rápidamente, cambian dicho paso para llegar a ser prácticamente aerotransportados, de esta forma durante breves períodos de tiempo no se encuentran estáticamente estables.

Los robots insecto biológicamente inspirados como los hexápodos son potencialmente muy estables dado que el movimiento de una pata no afecta generalmente la postura del robot, además de que con un algoritmo de control adecuado pueden contender con una amplia cantidad de superficies.

Para sintetizar:

- La estabilidad dinámica requiere un control más complejo que los controles usados para mantener una estabilidad estática.
- La estabilidad de un robot polípedo no es un problema trivial de resolver.

1.1.2 Locomoción

El desplazamiento o locomoción de un robot polípedo es el resultado de un movimiento coordinado de sus patas. Este movimiento está definido por cierto paso que refleja determinadas especificaciones tales como velocidad, dirección, etc.

Hay dos fases claramente distinguibles que contribuyen al movimiento de cada pata del robot: la *fase de soporte* y la *fase de transferencia*. Durante la *fase de soporte* o *apoyo*, cada pata debe ser capaz de ejercer cierta fuerza sobre la superficie en la que se encuentra el robot, de forma ordenada para proporcionar la fuerza necesaria al cuerpo del robot para así permitirle moverse según una trayectoria predeterminada. Después, durante la *fase de transferencia* o *transición*, la pata debe desplazarse de forma ordenada hacia el siguiente punto de soporte para reiniciar la secuencia de movimiento. Dichas fases se ilustran en la figura 1.3. Cada fase necesita un conjunto de requerimientos para la operación de la pata. Durante la fase de soporte, la pata debe ser capaz de mantenerse en equilibrio, mientras que en la fase de la transición el requisito principal es la velocidad de desplazamiento.

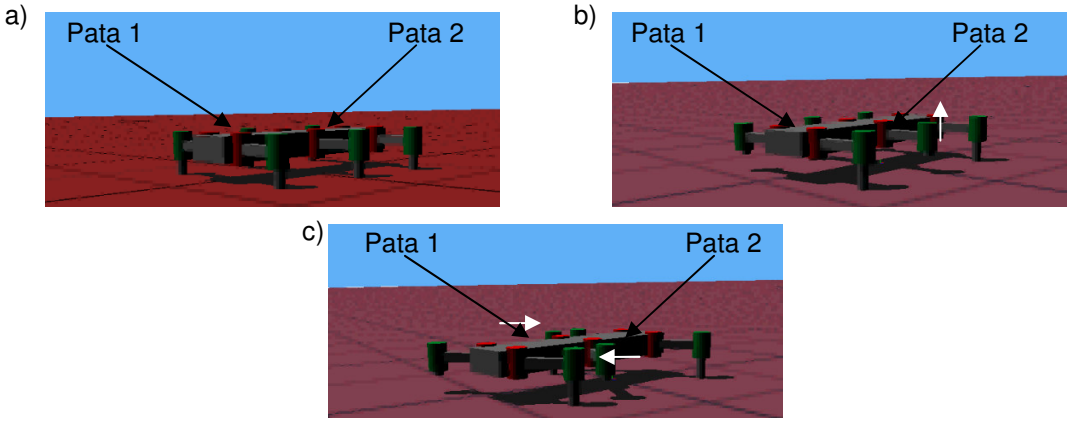


Figura 1.3 Secuencia de movimientos en donde la pata 2 ilustra la fase de transición y la pata 1 ilustra la fase de transferencia. a) posición inicial, b) levantamiento de la pata 2 y c) movimientos de cada pata

Una vez definida una tarea para el robot, la trayectoria se establece y debe ser seguida por el robot en su espacio de trabajo. El paso del robot definirá las transiciones de estado para cada pata. Sin embargo, la trayectoria de la pata durante la fase de soporte es determinada por la trayectoria del cuerpo del robot. Así, existe un número infinito de trayectorias que pueden ser utilizadas para obtener el movimiento deseado del robot. La definición de la trayectoria se basa únicamente en la movilidad de las patas y es funcional para un robot que camina en un terreno regular, debido a que no existen obstáculos en el movimiento de sus patas. Esta manera de definir la trayectoria de un robot no es aplicable a un robot que camina en terrenos irregulares, e. g. superficie con escalones o pendientes.

Para los robots que se desempeñan en ambientes irregulares, existe una gran incertidumbre en el resultado de la acción de cada pata. Así, es razonable seleccionar la trayectoria de cada pata y revisar continuamente el resultado de la acción de cada pata. Esto sugiere que además de todos los estados posibles considerados que representan las posiciones de cada pata, se consideren tantas variables como posibles lecturas de sensores que indican el resultado de cada acción de las patas. A mayor número de variables a controlar se incrementa la complejidad del sistema y en específico el control de cada pata.

Por ejemplo, supongamos que se tiene un robot hexápodo, cuyas patas poseen 2 grados de libertad o articulaciones y que cada articulación está controlada por un servomotor. Suponiendo que dicho servomotor tiene una precisión de paso de 2°, cada articulación tendría 180 posibles posiciones, dado que son 2 articulaciones se tendría *180x180 = 32400* posibles posiciones o configuraciones en las que puede estar una pata del robot. Como se tienen en total seis patas, entonces se tendría un total de $32{,}400^6$ posibles configuraciones de las patas del robot. Como se mencionó,

deseamos tener información acerca del resultado de cada acción de cada pata. Supongamos, para este ejemplo, que lo único que nos interesa es saber si la pata está posada o no en la superficie de apoyo, esta información puede obtenerse usando un sensor de contacto por cada pata. Así, el número anterior de configuraciones aumentaría considerando las lecturas de los sensores de contacto, con lo cual tendríamos $32{,}400^6 \times 2^6$ posibles estados del sistema.

Otro problema aunado a lo antes descrito es la optimización de trayectoria. Generalmente, se conocen los puntos inicial y final de la trayectoria del robot, y el problema es determinar la trayectoria óptima, de acuerdo a cierto criterio, que una ambos puntos. En este trabajo la trayectoria es conocida y es una línea recta, lo que debe ser decidido entonces es: (i) la localización del robot en un ambiente no estructurado, i. e. una superficie irregular, con pendientes, baches y obstáculos, y (ii) las posiciones óptimas de cada pata para realizar la trayectoria de forma estable.

1.1.3 Control

La generación de pasos estable es una parte integral del control de robots polípedos. Los pasos para hexápodos requieren la coordinación simultánea de los movimientos de las seis patas en un ciclo de activaciones. Además, el movimiento hacia delante sostenido requiere que la secuencia de acciones individuales de las patas que impulsan al robot sea repetida continuamente y que cada estado del robot, en el ciclo de movimiento, pueda ser fácilmente accesible. El problema se complica debido a las diferencias entre los robots hexápodos, e. g. morfología, altura, etc., así como las diferentes capacidades de cada pata. Hay patas con dos, tres o más grados de libertad, además de sensores adicionales para medir por ejemplo, posición de cada articulación y contacto con la superficie, entre otros.

Por ello, es importante desarrollar algoritmos de control que permitan responder de manera adecuada a las situaciones en las que llegue a encontrarse un robot, y más aún, que dicho algoritmo permita que el robot sea lo más autónomo posible, esto es, que sea capaz de responder ante situaciones tanto comunes como inesperadas con su propias capacidades y medios, sin la intervención del diseñador.

El control de robots hexápodos es un problema abierto en robótica. Se han propuesto algoritmos de control para robots hexápodos, los cuales han ido evolucionando y mejorándose con el tiempo. Existen varias propuestas de control las cuales se estudiarán en el capítulo II, pero todos tienen ciertas carencias. Algunos trabajos, por ejemplo (Jindrich et al. 1999), asumen varias hipótesis para simplificar el control del robot, e. g. que las acciones son completadas sin ningún tipo de desviación, lo que hace que el robot sea poco autónomo y robusto. Otros trabajos utilizan enfoques reactivos (Maes & Brooks 1990), lo cual hace que tales robots sólo puedan ser capaces de realizar tareas relativamente sencillas, e. g. avanzar de manera estable. Otros trabajos (Nakada et al 2003) usan redes neuronales, para que un robot polípedo sea más autónomo, aunque la forma en que se obtiene el control no permite realizar un estudio más detallado de las situaciones que existen en el mundo del robot para tomar una determinada decisión de acción. Es necesario entonces buscar alternativas de desarrollo de algoritmos de control que permitan al robot ser autónomo y robusto, en una diversidad de ambientes no estructurados y que además permitan explicar las situaciones por las que un robot se ve obligado a tomar cierta decisión, así como estudiar los patrones de movimiento del robot, su generalidad en el desplazamiento de robots polípedos y su eventual equivalencia con los patrones de movimiento de animales hexápodos.

1.2 Descripción de la tesis

1.2.1 Objetivos

El objetivo general de esta tesis es **diseñar e implementar un esquema de control descentralizado eficiente basado en Procesos de Decisión de Markov** (MDP por sus siglas en inglés, *Markov Decision Process*) **para un robot hexápodo**.

Por eficiente se entiende que el tiempo de respuesta sea del orden de segundos, tanto para el robot simulado como para un prototipo físico operando con tarjetas y procesadores comerciales actuales.

Como objetivos específicos se han establecido los siguientes:

- Diseño de un protocolo de coordinación para un sistema de control basado en descentralizaciones de un MDP, MDP jerárquico y distribuido. No existe trabajo reportado en este sentido.
- Análisis de la forma de caminar de hexápodos con una distribución circular de patas alrededor de su cuerpo.
- Diseño de un modelo de control que permita a un robot hexápodo desplazarse de manera eficiente, i. e. de forma estable y responder de forma adecuada ante situaciones conocidas como inesperadas del ambiente.
- Diseño de un modelo de control de un robot hexápodo más descriptivo de los que existen hasta el momento, i. e. un modelo que permita identificar las condiciones del ambiente que ocasionan que el robot genere una configuración determinada.
- Proposición de principios de organización de MDPs en problemas de locomoción robótica.

En esta tesis se aplicaron Procesos de Decisión de Markov (MDPs) para resolver un problema de locomoción de robots hexápodos, específicamente robots hexápodos con morfología hexagonal o circular.

Se realizaron varios modelos usando MDPs de los cuales finalmente se obtuvieron dos modelos descentralizados principales: un modelo con decisión centralizada y actuación descentralizada (DCAD) y un modelo con decisión y actuación descentralizadas (DDAD). Ambos modelos permiten al robot hexápodo desplazarse de manera estable en ambientes semi-estructurados. Se realizaron diversas pruebas para evaluar su desempeño del robot y para establecer las ventajas y desventajas de cada modelo. Finalmente se valoraron algunos elementos del modelo DDAD en un robot físico.

1.2.2 Organización de la tesis

El resto de la tesis se encuentra dividido en seis capítulos. En el segundo capítulo se describen los fundamentos de en esta tesis. Se hace también un recapitulativo acerca de los diferentes enfoques que existen y que han existido para el diseño de sistemas de control para robots hexápodos, destacando las ventajas y desventajas de cada enfoque, con la finalidad de determinar en dónde se encuentra ubicado nuestro trabajo. En el tercer capítulo se da una descripción formal de MDPs y HMDPs, los diferentes métodos de resolución de estos modelos y las diferencias que existen para el diseño de estos mismos. En los capítulos cuarto y quinto se describen, respectivamente, los modelos propuestos en esta tesis y los resultados obtenidos. Finalmente, en el sexto capítulo se exponen las conclusiones de este trabajo y sus perspectivas. Adicionalmente, en los apéndices se proporcionan aspectos técnicos de las herramientas utilizadas para la realización de esta tesis.

CAPÍTULO II: Historia y Estado del Arte de los Robots Hexápodos

2.1 Conceptos

Existen varias definiciones del término robot, entre las más populares pueden mencionarse las siguientes. Un *robot* se define informalmente como una máquina con capacidad de movimiento y acción, o computadora con "músculos" [URL_Robotica1]. También suele definirse el término *robot* como dispositivo electrónico, generalmente mecánico, que desempeña tareas automáticamente, ya sea bajo supervisión humana directa o a través de un programa predefinido o siguiendo un conjunto de reglas generales, utilizando técnicas de inteligencia artificial. Generalmente estas tareas reemplazan, asemejan o extienden el trabajo humano, como ensamble en líneas de manufactura, manipulación de objetos pesados o peligrosos, trabajo en el espacio, etc. [URL_Robotica2].

Dentro de las diversas clases de robots, nos interesaremos en tres familias: robots móviles, autónomos y polípedos. Un *robot móvil* es aquel capaz de desplazarse completamente en su ambiente de trabajo. Un *robot autónomo* es aquel capaz de tomar decisiones por sí mismo, y está orientado a reducir a lo mínimo la intervención del diseñador, en su operación. Un *robot polípedo* o con patas es aquel que físicamente tiene patas o extremidades, las cuales utiliza para desplazarse en su ambiente de trabajo. A su vez, los robots polípedos pueden ser divididos de acuerdo al *número de extremidades* que tienen, según se describe a continuación.

Robots monópodos. Este tipo de robots sólo cuenta con una extremidad o pata, el ejemplo más representativo es un robot desarrollado por el Instituto

de Robótica y el Departamento de Informática de la universidad americana Carnegie-Mellon. Este dispositivo salta sobre su única extremidad, buscando continuamente la estabilidad dinámica, i. e. con el robot en movimiento, ya que para este robot es difícil alcanzar una estabilidad estática i. e. con el robot inmóvil, debido a que lograr una estabilidad estática es complicado usando menos de cuatro patas. Este tipo de robots requiere en general un gran esfuerzo del diseñador para desarrollar su sistema de control.

Robots bípedos. Este tipo de robots posee dos patas y puede caminar lateralmente, avanzar y retroceder, simulando más o menos el modo de andar humano. Entre los robots bípedos más destacados podemos mencionar el presentado en (Ki et al. 2003).

Robots cuadrúpedos. Este tipo de robots posee cuatro extremidades o patas y es el más representativo de los robots con patas. En el Instituto Tecnológico de Tokyo fue construido un vehículo de cuatro patas, equipado con sensores de contacto y sensores de rotación. Cada pata tiene 3 DOF. El control se realiza desde un microprocesador que asegura la formación de un triángulo de apoyo con tres de sus patas de forma continua. Por otra parte, se han desarrollado nuevos algoritmos para conseguir que este tipo de robots conserve la estabilidad mientras camina, así como algoritmos para la detección de inestabilidades y la planificación de modos de andar.

Robots hexápodos. *Son robots que cuentan con seis patas. En la Universidad Estatal de Ohio se desarrolló un robot con seis patas que permite al operador tomar decisiones estratégicas, independientemente de la posición particular de cada extremidad. Está basado en un sistema de control consistente en 13 procesadores Intel 86/30. En el Instituto de Robótica de la Universidad de Carnegie-Mellon de EEUU se desarrolló el AMBLER, acrónimo de Autonomous MoBiLe Exploratory Robot, un robot hexápodo*

dirigido por programas específicamente diseñados para optimizar el movimiento de avance sobre terrenos abruptos.

Por ser de interés para esta tesis, a continuación se abundará sobre este último tipo de robots.

2.2 Robots hexápodos

Los robots insectos están atrayendo la atención de personas que trabajan en robótica móvil, debido a sus amplias capacidades de locomoción. Los investigadores de robots tipo insecto están explorando la naturaleza como una manera alternativa para crear nuevos robots y modelos de control para esos robots.

Los robots hexápodos autónomos pueden ser útiles en ambientes hostiles, e. g. exploración de volcanes, de las profundidades del océano, del escombro de desastres y de los cuerpos extraterrestres; tareas militares tales como la inspección del campo enemigo y la búsqueda de minas explosivas. Un factor importante para lograr la autonomía completa es la capacidad del robot para adaptarse a los cambios en su ambiente. Esto motiva a los diseñadores para hacer que el sistema de control sea flexible y se adapte tanto como sea posible a los cambios imprevistos del ambiente. Además, la adaptación a través del aprendizaje reduce la intervención humana requerida para que el sistema de control responda a los nuevos cambios.

La generación de pasos que permitan a un robot desplazarse de manera estable es una función integral de un control para todo robot con patas, debido a que estos pasos deben permitir aprovechar correctamente el número de extremidades del robot. Los pasos para hexápodos requieren la coordinación del movimiento simultáneo de sus seis patas. Además, el movimiento hacia adelante requiere que la secuencia de las acciones

individuales que mueven al robot sea repetida continuamente sin perder la estabilidad del robot, considerando que un grupo de extremidades del robot se mueve mientras otro grupo permanece apoyado en la superficie.

Otra fase importante en el desarrollo del control para un robot es evaluar que dicho control permita la estabilidad y el desplazamiento de un robot real. Dado que las simulaciones por sí solas no pueden predecir totalmente el comportamiento de robots físicos reales, debido a que, por definición, las simulaciones son aproximaciones de sus contrapartes reales.

A continuación se hace una revisión de los diferentes enfoques para diseñar e implementar un sistema de control eficiente para robots con patas, en particular, para robots hexápodos.

2.3 Tipos de robots hexápodos

Existen 2 tipos de morfología para robots hexápodos. La más estudiada, como se verá a lo largo de este capítulo, para desarrollo de sistemas de control eficientes para robots hexápodos es la morfología la cual hemos denominado *morfología rectangular* y la cual es presentada en la figura 2.1.

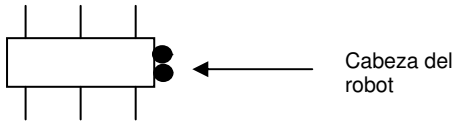
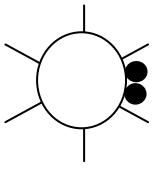


Figura 2.1. Morfología rectangular

En este trabajo, la morfología del robot hexápodo en la que se probará el sistema de control es diferente, le hemos denominado como *morfología circular* y es mostrada en la figura 2.2.



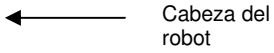


Figura 2.2. Morfología circular

Al hablar de morfología rectangular o circular, nos referimos a la forma del cuerpo del robot hexápodo y la distribución de las patas alrededor de éste. Además, cabe mencionar que cada pata tiene el mismo número de DOF en ambas morfologías.

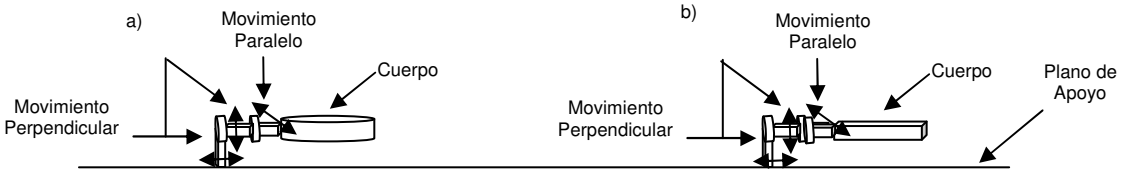


Figura 2.3. Vista lateral de una pata con 3 DOF, los movimientos perpendicular y paralelo son relativos al plano de apoyo. a) Morfología Circular b) Morfología Rectangular

De acuerdo a su morfología, un robot hexápodo tendrá diferentes capacidades de movimiento. Por ejemplo, con una morfología rectangular, el hacer que un robot cambie de dirección es un proceso que requiere un análisis puntual y quizás la construcción de un sistema de control específico. Además, hacer que el robot cambie de dirección de movimiento sin que éste gire total o parcialmente su cuerpo no es un problema sencillo. Esta situación limita la capacidad de este tipo de robots, debido a que si estos robots se encuentran inmersos en ambientes los cuales tienen espacios muy reducidos, puede ser imposible o al menos muy difícil para uno de estos robots salir de una determinada situación, como se muestra en la figura 2.4.

En el escenario anterior es complicado que el robot pase por el pasillo, sin que tenga que mover todo su cuerpo. Este problema también se presenta en robots con llantas, pero en este tipo de robots el problema es resuelto

usando llantas omnidireccionales, las cuales permiten al robot cambiar de dirección de movimiento sin necesidad de cambiar la orientación de su cuerpo. Lo que se ha empezado a hacer para resolver la problemática anterior, para robots con patas, es crear morfologías físicas las cuales permitan holomonicidad, i. e. la capacidad de movimiento en cualquier dirección sin necesidad de girar el cuerpo.

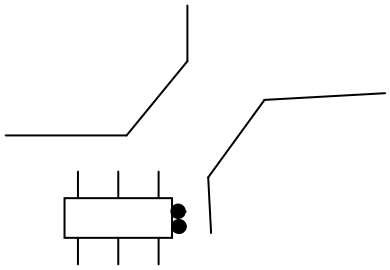


Figura 2.4. Posible situación que requiere diversas maniobras para ser superada por un robot hexápodo con morfología rectangular, e. g. retroceder y girar varias veces

Un robot hexápodo con una morfología circular en la misma situación que el robot anterior, podría cambiar la dirección de su desplazamiento sin necesidad de cambiar la orientación de su cuerpo, esto es, podría desenvolverse en ambientes en los cuales un robot hexápodo con morfología cuadrada no podría, como lo muestra la figura 2.5.

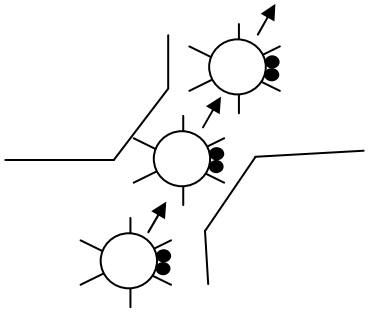


Figura 2.5. Posible situación que solamente requiere maniobras de desplazamiento lateral para ser solucionado por un robot hexápodo con morfología circular

Ahora bien, a pesar de que la morfología circular proporciona ventajas sobre la morfología rectangular, hasta el momento no se han reportado trabajos

relacionados con el desarrollo de sistemas de control eficientes para robots hexápodos con este tipo de morfología, así que uno de los objetivos y una contribución de esta tesis es el proponer e implantar un control eficiente para robots con este tipo de morfología el cual le permita aprovechar las características de la morfología circular.

Con aprovechar las características de dicha morfología, nos referimos a usar de manera adecuada la configuración de las patas alrededor del robot para realizar cambios de dirección de movimiento de una manera que no implique realizar varias maniobras, sino únicamente cambiar la funcionalidad de las patas. Esto será posible ya que teniendo una morfología circular, cada pata se moverá de alguna manera específica al desplazarse en una determinada dirección, cada movimiento de cada una de las patas tratará de conservar la distribución de éstas alrededor del cuerpo del robot. De esta manera, si en algún instante determinado el robot necesitara cambiar de dirección de desplazamiento, será suficiente cambiar la funcionalidad de las patas del robot. Esta redundancia o capacidad de intercambiar el funcionamiento de las patas no es posible de realizar en robots con morfología rectangular, debido a que la manera en la que se encuentran distribuidas las patas alrededor del robot no guarda la misma simetría de los robots con morfología circular.

La morfología rectangular ha sido ampliamente estudiada, probablemente debido a que los primeros modelos para crear robots hexápodos se inspiraron principalmente en insectos como la cucaracha, el palillo, los mosquitos y algunas arañas.

Los sistemas de control que se presentan en los trabajos revisados en esta tesis llegan a ser insuficientes e inadecuados para robots hexápodos con morfología circular. Esto se debe a que las configuraciones que le permiten

desplazarse a un robot con una morfología rectangular son especialmente aplicables a ese tipo de robots, pero no funcionales para robots con morfología circular. Con este trabajo se dará una propuesta para el diseño de sistemas de control para robots hexápodos con morfología circular.

Diseñar un sistema de control eficiente tanto para robots con morfología rectangular como para aquellos con morfología circular tiene cierto grado de dificultad. Con respecto al control de robots hexápodos con morfología circular, difiere y se vuelve más complicado que el control que normalmente se utiliza para robots con morfología rectangular. El trípode, movimiento en conjuntos de tres patas, permite que los robots hexápodos con morfología rectangular puedan desplazarse de una manera estable pues se mantiene el centro de gravedad del robot sin mucha alteración. Cabe señalar que dicho movimiento en trípode ha sido ampliamente estudiado e implementado. En contraste, para controlar robots hexápodos con morfología circular, la sincronización y funcionalidad de cada una de las patas es diferente debido a que se busca que el paso que le permita al robot desplazarse conserve la distribución de las patas alrededor del robot para explotar dicha característica como se discute más adelante. Así, el uso de trípode en los robots hexápodos de morfología circular puede llegar a ser no funcional, por ejemplo los movimientos y configuraciones de patas que son utilizados para hacer avanzar un robot con morfología rectangular no tendrán el mismo resultado al aplicarlos a un robot con morfología circular, debido a la manera en la que se encuentran distribuidas las pastas en ambos tipos de robots.

2.4 Tipos de control para robots hexápodos

Dado que estamos interesados en el diseño del sistema de control para un robot hexápodo es necesario hacer una revisión acerca de los trabajos relacionados con el diseño de control para locomoción de este tipo de robots, para así ubicar nuestro trabajo y su contribución al estado del arte.

Al paso del tiempo han existido diversas maneras en las que se ha diseñado el control para robots hexápodos, entre los enfoques más conocidos podemos mencionar los siguientes:

- Control manual
- Control orientado a metas y objetivos
- Control biológicamente inspirado
- Control difuso
- Arquitectura subsumción
- Control usando aprendizaje por refuerzo
- Control basado en Procesos de Decisión de Markov (MDPs)

A continuación se resumen las características y se revisan trabajos representativos de cada uno de los enfoques anteriores.

2.4.1 Control manual

Las primeras máquinas con patas desarrolladas eran de dimensiones y peso considerables, básicamente adaptaciones de camiones de carga, las cuales eran operadas manualmente por una persona, por eso era prácticamente inalcanzable dotar de cierta autonomía a este tipo de máquinas. Así, no se le puede dar el término de robot a este tipo de artefactos.

La primera máquina con patas que caminaba fue la *Walking Truck* (Raibert 1986), la cual contaba con cuatro extremidades, pertenecía a General

Electric y fue diseñada por R. Mosher. El control de esta máquina era totalmente manual debido a las limitantes tecnológicas de esos años, la figura 2.6 muestra una imagen de dicha máquina.

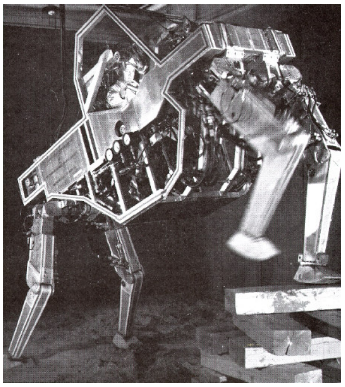


Figura 2.6. Walking Truck. Tomada de (Raibert 1986)

Como podemos ver en la figura 2.6 el objetivo de crear este tipo de máquinas era desarrollar una máquina capaz de moverse en terrenos en los cuales una máquina con llantas no pudiese moverse y esto se consiguió parcialmente usando patas en lugar de llantas.

2.4.2 Control orientado a metas y objetivos

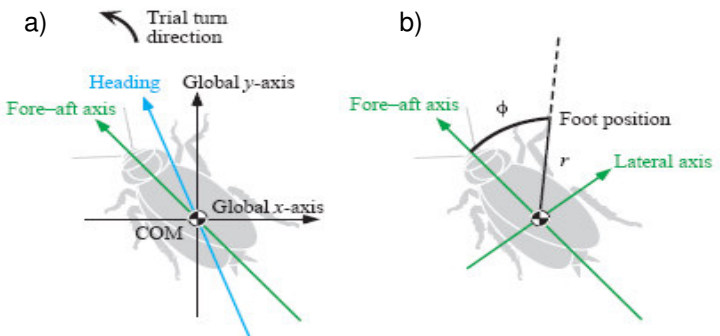
Una vez desarrolladas las primeras máquinas con patas, los diseñadores se dieron cuenta de que era necesario dotar de cierta autonomía a este tipo de máquinas, para evitar de esta forma que el diseñador interviniera de manera continua durante el funcionamiento de la máquina.

De esta forma se empezaron a diseñar sistemas de control que permitieran a la máquina, en principio, moverse y mantenerse estable. Como una primera forma de resolver este problema surgió un enfoque el cual básicamente consiste en dotar a la máquina de una tabla gigantesca que contempla todas las situaciones a las que la máquina puede enfrentarse. A este tipo de

máquinas ya podemos asignarles el término de robots, pues pueden alcanzar una meta u objetivo específico, como puede ser, mantenerse de pie o desplazarse. Es aquí en donde podemos empezar a hablar de robots caminantes semi-autónomos y autónomos.

Entre los trabajos en los cuales se usa un control para hexápodos orientado a metas y objetivos podemos mencionar los de Jindrich et al. (1999) y Chen et al. (2003). En este tipo de control se realiza una caracterización amplia y exhaustiva de todas las posibles situaciones en las que puede llegar a encontrarse el robot, y se programan todas las acciones de acuerdo al estado en el que se encuentre el robot. Por lo general se utilizan modelos matemáticos complejos y se usa el paradigma SPA, (Maes & Brooks 1990).

En el trabajo de Jindrich et al. (1999) se utiliza cinemática de dos dimensiones para estimar las fuerzas totales y de troqué (fuerza aplicada a una palanca que hace rotar alguna cosa), dicho modelo se ilustra en la figura 2.7. Además, se usa una técnica fotoelástica para estimar las fuerzas de *reacción – tierra*, fuerzas que se presentan al momento en que una pata hace contacto con la superficie de apoyo, de una pata durante un giro. Para interpretar la fuerza de una pata que se observa durante un giro, los autores desarrollaron un modelo general que relaciona la fuerza producida de una pata y su posición cuando se está llevando a cabo un giro. Este modelo predice que todas las patas podrían girar al cuerpo. Los autores realizan un estudio acerca de las consideraciones anteriores para tomarlas en cuenta en el desarrollo de un control para un robot hexápodo pero no llegan a implementar algún control. Así, este trabajo tiene un carácter esencialmente teórico.



c) d)

Figura 2.7. Modelo de cinemático de Jindrich. a) Sistema de tres coordenadas, b) media de la posición de una pata en coordenadas polares, c) sistema de coordenadas simplificado y d) variables cinemática. Tomadas de (Jindrich et al. 1999)

En el trabajo de Che et al. (2003) se prueba en un robot cuadrúpedo (*Planar Walker*) un control basado en un mecanismo de circuito cerrado, dicho robot se muestra en la figura 2.8. Este robot puede producir movimientos en dos direcciones sobre el plano horizontal (derecha-izquierda, adelante-atrás) y es capaz de rotar sobre él mismo. Los pasos que permiten al robot desplazarse en forma horizontal, los pasos de giro del robot con sus respectivas longitudes finitas y ángulos finitos de rotación son obtenidos con el sensado de los cilindros y de los "agarradores" (dispositivos que producen fuerza de agarre a la superficie) que forman las patas del robot. Las secuencias de actuación de los cilindros y de los "agarradores" para generar diversos tipos de pasos son modeladas usando autómatas finitos. La cinemática de los pasos se describe usando mecánica de movimiento de cuerpo rígido, la cual estudia el movimiento y equilibrio de los cuerpos sólidos ignorando sus deformaciones. Cuando una serie de pasos se ejecuta, el robot sigue una trayectoria segmentada no-lisa, esto es una trayectoria no lineal como lo haría un vehículo con llantas. De acuerdo con estas

características, tres métodos de navegación se desarrollaron para ser probados en un robot cuadrúpedo en varios panoramas: algoritmo de Simple Línea de Vistas (*SLS, Simple Line of Sight*), Planificación Exacta basado en Recocido Simulado (*SAAP, Simulated Annealing based Accurate Planning*) y el algoritmo Híbrido de Localización de Planeamiento Exacto (LHAP, Localized Hybrid Accurate Planning).

Los autores concluyen a partir de experimentos en simulación que el algoritmo SAAP produce secuencias exactas de pasos y el algoritmo LHAP ahorra tiempo y recursos de cómputo para lograr metas de largo alcance, estas metas pueden ser consultadas en [URL_Metas]. Sin embargo, los resultados experimentales demuestran que los errores posicionales inherentes al movimiento (incertidumbre de resultado de las acciones) individuales del paso pueden ser sustanciales y cuando estás se acumulan pueden hacer que un algoritmo particular resulte menos eficaz.

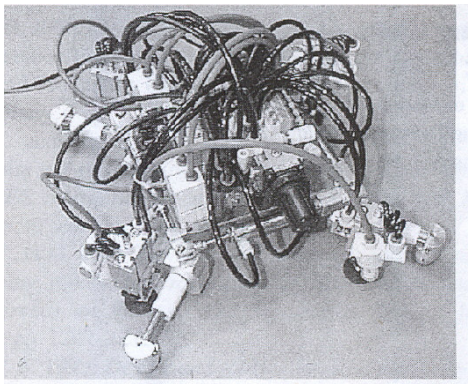


Figura 2.8. Robot de Chen. Tomadas de (Chen et al. 2003)

Kindermann (2002) presenta un sistema cinemático complejo de un robot hexápodo con un total de 18 DOF. Este sistema muestra una variedad de comportamientos los cuales son controlados por un sistema de control no trivial. Fue necesario realizar un estudio detallado cuantitativo de cada uno de los comportamientos del robot para comprender y abstraer las

propiedades de dichos comportamientos. Esta “etología artificial” es aplicada para controladores con una estructura descentralizada que utiliza un modelo base, en este caso el modelo del insecto palillo. El sistema aquí mostrado es capaz de adaptarse a condiciones externas imprevistas sin necesidad de reprogramación, debido a que el sistema considera y utiliza las configuraciones recurrentes que se presentan en un ciclo de movimiento a través del ambiente. El sistema de control le permite al robot sobrepasar obstáculos, que recuperarse de tropiezos o caminar con una pérdida total o parcial de una de sus patas. La figura 2.9 muestra imágenes del modelo y del robot de Kindermann.

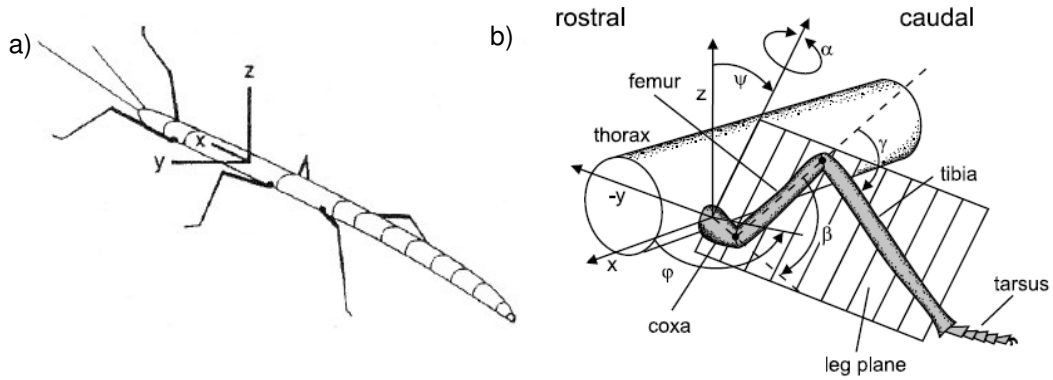


Figura 2.9. Modelo de Kindermann (2002). a) Esquema del insecto palillo, b) modelo de una pata del insecto. Tomadas de (Kindermann 2002)

Como se ha discutido previamente, en este tipo de enfoques se invierte una gran cantidad de trabajo y esfuerzo en el modelado matemático del problema. Además, generalmente no se considera la incertidumbre que existe en la ejecución de una acción del robot pues en estos enfoques se idealizan tanto la percepción como el resultado de las acciones ejecutadas por el robot.

2.4.3 Control biológicamente inspirado

Debido a las carencias de los enfoques antes discutidos, los investigadores buscaron nuevas formas de diseñar sistemas de control eficientes para robots con patas, uno de los enfoques que ha presentado buenos resultados es el conocido como biológicamente inspirado. Este enfoque nace del estudio del comportamiento de los insectos reales y básicamente trata de imitar en menor o mayor proporción las características biológicas de los insectos (Kerscher et al. 2004). En el trabajo de Kerscher et al. (2004) se describen los diferentes niveles de inspiración biológica en el diseño de robots con patas. Así, la categoría de los robots con patas biológicamente inspirados contempla cualquier robot en el cual alguno de los componentes de cuyo sistema, e. g. diseño mecánico, sistema de actuadores, arquitectura de control o control adaptativo; está ampliamente influido por un diseño con principios de funcionamiento encontrados en la naturaleza. Es por eso que en esta sección discutiremos algunos de los robots con patas los cuales están biológicamente inspirados en algunos de los niveles que menciona Kerscher et al. (2004).

2.4.3.1 Hardware

En esta sección consideramos aquellos robots con patas cuyos diseños mecánicos y su sistemas de actuadores están biológicamente inspirados.

En el trabajo de Kingsley (2006) se presenta la construcción de un robot hexápodo inspirado directamente de la fisonomía de las cucarachas que intenta aprovechar la ventaja de la neuromecánica, i e. sistemas que combinan controladores basados en redes neuronales y dispositivos mecánicos, y de los actuadores con un sistema enfocado a reproducir la funcionalidad de los músculos reales de tal insecto, con el fin de demostrar una las ventajas que tienen con respecto al uso de actuadores convencionales. Los actuadores convencionales son poco factibles para

capturar con precisión los rangos de movimiento de un robot hexápodo. Este trabajo muestra que dotando de 24 DOF a un robot hexápodo podemos obtener una muy buena aproximación de un robot capaz de caminar y escalar. Este trabajo citado describe el diseño físico del robot y muestra que es capaz de desplazarse en circuitos abiertos y accidentados. Utiliza un controlador de movimiento hacia adelante sin alguna retroalimentación y el robot es capaz de producir una locomoción hacia adelante aceptable. La figura 2.10 muestra el robot hexápodo construido en este trabajo.

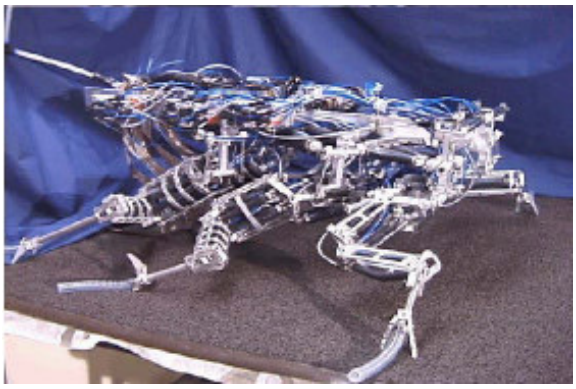


Figura 2.10. Robot de Kingsley. Tomada de (Kingsley 2006)

En Kerscher (2004) se describe la estructura mecánica de un robot hexápodo biológicamente inspirado llamado *Airinsect*, el cual se basó en el modelo natural del insecto *palillo*. El enfoque usado para la construcción de Airinsect, a diferencia de otros robots hexápodos construidos inspirados en dicho insecto, es que el arreglo de patas y el diseño de las empalmes fueron construidos como una exacta imitación de cómo se encuentran en el insecto verdadero. Para reducir el peso de este robot se utilizaron materiales ligeros. Para hacer uso de las ventajas de este robot se diseñó un sistema de control basado en un modelo de retroalimentación a nivel del microcontrolador. Dicho control calcula y ajusta los ángulos deseados de las empalmes del robot. Para el control de alto nivel son utilizados los algoritmos de control del robot LAURON. La figura 2.11 ilustra el robot Airinsect.

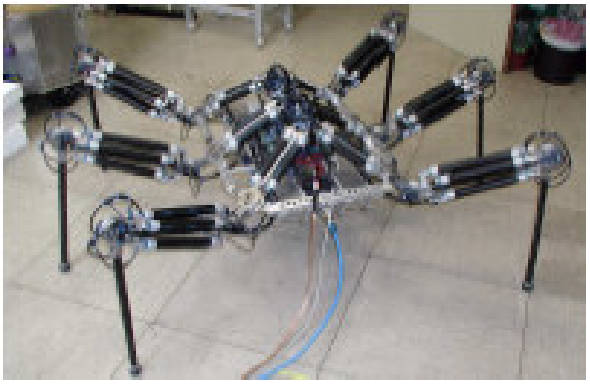


Figura 2.11. Robot Airinsect. Tomada de (Kerscher 2004)

En Weidemann (1994) se presenta una prueba de un robot hexápodo diseñado y construido en el Instituto de Mecánica de la Universidad Técnica en Munich. Este robot fue diseñado basándose en principios inspirados en la forma natural de caminar de los hexápodos. En este trabajo se describe al insecto investigado y sus características principales a pesar de que no se da una descripción del hardware mecánico y electrónico usados en la construcción. El trabajo está más apegado a un enfoque biológicamente inspirado a un nivel de hardware, aunque no se detalle el sistema de control utilizado.

En este tipo de trabajos se argumenta que, a partir de los resultados obtenidos de las investigaciones concernientes a la morfología y neuroetología de los animales, para obtener un robot con patas biológicamente inspirado se debe diseñar un sistema mecatrónico, i. e. sistemas mecánicos extendidos e integrados con sensores, microprocesadores y controladores, y un sistema de sensores capaces de obtener información altamente confiable acerca del estado interno del robot, así como de la situación del ambiente en el que se encuentran inmersos. De esta manera se descarga la complejidad del sistema de control para el robot.

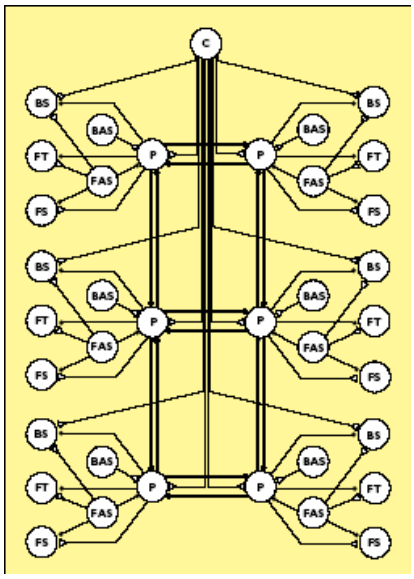
Debido a que los actuadores que comúnmente son usados en los robots móviles, y por ende en los robots con patas, no son capaces de proporcionar la velocidad, precisión y el torque deseados para realizar el movimiento de un robot con patas, existe incertidumbre acerca de cuál es el verdadero resultado de ejecutar cierta acción para hacer que un robot con patas se desplace. La tarea de realizar un sistema de control eficiente para un robot con patas el cual cubra el impacto de la incertidumbre antes mencionada se vuelve una labor complicada. Es por eso que este tipo de trabajos invierten gran cantidad de esfuerzo y recursos para definir un buen sistema mecatrónico y un sistema de sensores adecuado para el robot.

Otros trabajos que también podemos situar en este nivel de los robots hexápodos biológicamente inspirados son los presentados en (Cruse 1976) y (Berns 2002).

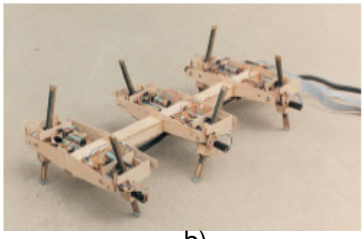
2.4.3.2 Sistema de control

En esta sección consideramos aquellos robots con patas cuya arquitectura de control o control adaptativo están biológicamente inspirados.

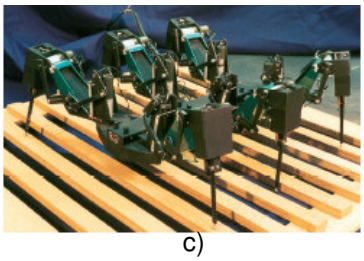
Beer et al. (1997) estudiaron la manera de diseñar el control de un robot con patas basado directamente en estudios de insectos reales y su sistema nervioso. Estos autores se inspiraron en un enfoque biológico para definir el diseño y control de un vehículo de exploración robusto. Su aportación fue el uso de redes neuronales para crear un modelo distribuido y robusto del control de un robot hexápodo.



a)



b)



c)

Figura 2.12. a) Esquema de la red neuronal utilizada por Beer b) Robot I de Beer c) Robot II de Beer. Tomadas de (Beer et al. 1997)

En el trabajo de Schneider (2006) se realiza un estudio acerca de la generación de movimientos en cadenas cinemáticas cerradas argumentando que la dificultad de éstas se debe a que todos los empalmes que participan en dichas cadenas tienen que ser movidos de una manera altamente coordinada para evitar tensiones destructivas en la pata de un robot. Schneider presentó un controlador descentralizado para los empalmes el cual utiliza las interacciones de bajo nivel que existen entre un empalme en movimiento y la consistencia de sus empalmes vecinos, el cuerpo y el entorno en el que se encuentra el robot. Este controlador está basado en un mecanismo (LPVF, *Local Positive Velocity Feedback*) el cual explota las características elásticas del empalme. La estrategia de control está inspirada por el sistema de locomoción de los insectos palillo. Schneider demuestra que una cadena cinemática cerrada que consiste de varios controles LPVF, a pesar de carecer de unidad central de control, puede solucionar tareas sin necesidad de una coordinación de alto nivel entre los empalmes. Su trabajo

fue probado sobre un manipulador que da vuelta a una manivela en una pata de un robot con 3 DOF.

En el trabajo de Berns (1994) se presenta una manera de construir una arquitectura de control basada exclusivamente en redes neuronales. Las redes neuronales son adecuadas para controlar tareas especializadas debido a su capacidad de procesamiento en tiempo real, tolerancia a fallos y adaptabilidad (Berns 1994). Para probar dicha arquitectura Berns utiliza un robot hexápodo. Finalmente, Berns da una descripción de los requerimientos de hardware necesarios cuando se utiliza una arquitectura de control basada en redes neuronales.

Otro rama de robots con patas biológicamente inspirados a nivel del sistema de control, comprendiendo aquellos trabajos que estudian la manera de generar un CPG (C*entral Pattern Generator*). El CPG es una red neuronal biológica la cual se encarga de controlar los movimientos rítmicos de un ser vivo, los cuales le permiten tener una locomoción estable al caminar, correr, nadar y volar. El CPG genera un patrón rítmico de actividad nerviosa de manera inconsciente y automática Dicho patrón rítmico activa neuronas motor, de esta manera los movimientos rítmicos de los animales son generados. Un sistema sensorial de retroalimentación regula la frecuencia y la fase del patrón rítmico generado por el CPG. Entre los trabajos más representativos podemos listar los siguientes.

En el trabajo de Nakada (2003) se propone un controlador CMOS analógico denominado por su autor como “neuromorfológico” (*nueromorphic*) para la coordinación de las patas en un cuadrúpedo. El autor utiliza un CPG como controlador de la locomoción de un robot. Muchos de los controladores que usan un CPG han sido desarrollados con un controlador digital y por esta razón presentan varias dificultades para su uso, una de las dificultades más

importantes es su alto consumo de energía. Para resolver ese problema, Nakada propone el uso de un controlador CMOS analógico. Además, usando tal controlador se espera reducir el costo de producción y obtener una buena miniaturización del dispositivo. Nakada prueba su controlador sólo a nivel de simulación, el circuito obtenido es capaz de generar varios patrones de movimiento rítmicos y de cambiar de manera rápida entre patrones. Así, el control básicamente consiste en construir un CPG y para alcanzar tal objetivo se utilizan redes neuronales, las neuronas que se utilizan en este trabajo son de tipo Amari – Hopfield, ver figura 2.13.

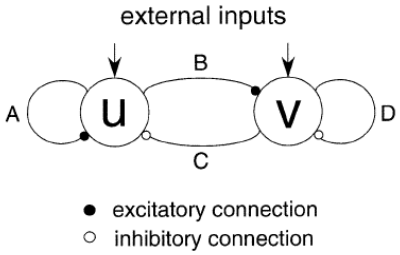


Figura 2.13. Modelo de las neuronas de Amari – Hopfield. Dicho modelo consiste de una neurona exitatoria y una neurona inhibidora, con una conexión inhibidora - exitatoria. Tomada de (Nakada 2003)

Otros trabajos relacionados en controles de robots con patas usando CPG son (Still & Tilden 1998), (Patel et al. 1998) y (Lewis et al. 2001) de Nakada (2003).

Lewis (1994) describe el estado de evolución para crear un CPG complejo para el control de los movimientos de las patas de un robot hexápodo. Como ya se mencionó, el CPG está compuesto por una red neuronal. En este trabajo se utiliza un algoritmo genético el cual determina la manera de interconectar las neuronas en una red neuronal en lugar de utilizar algoritmos de aprendizaje que es la manera típica de determinar las conexiones en una red neuronal. El uso de un algoritmo genético es para incrementar la velocidad a la que convergen los pesos de la red neuronal usada, de modo

que de esta manera se obtiene de una manera más rápida un comportamiento dirigido a alcanzar una meta. Primero se diseña un oscilador para los movimientos individuales de una sola pata, posteriormente una red de esos osciladores es diseñada para coordinar los movimientos de las diferentes patas.

Debido a que los robots biológicamente inspirados tienen un gran número de sensores y actuadores se necesita reducir la complejidad del problema usando arquitecturas de control descentralizadas (Albiez 2004).

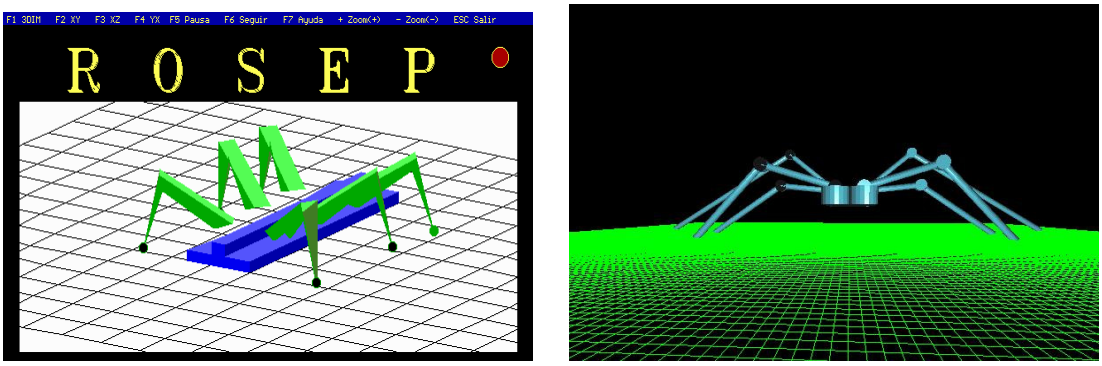
Estos últimos trabajos están orientados esencialmente a realizar controles para robots cuya riqueza en cantidad de sensores y actuadores es muy poderosa. Además, para garantizar un control eficiente este enfoque requiere arquitecturas de control muy detalladas, además de requerir una caracterización muy fina de las lecturas de los sensores y actuadores. Además Nakada afirma, con respecto a la coordinación de patas, que a partir de que el número de grados de libertad por pata en un robot polípedo suele ser grande (≥3) se requiere una coordinación eficiente de las patas para tener un desplazamiento estable. Por ello, Nakada propone que el uso de un CPG es muy apropiado para llevar a cabo dicha coordinación y afirma también que el CPG juega un papel fundamental en la locomoción de robots con patas (Nakada 2003).

2.4.4 Control difuso

Existe la alternativa de usar lógica difusa para diseñar un sistema de control para un robot hexápodo.

En el trabajo de Gorrostieta y Vargas (2004) se presentan diversos algoritmos, los cuales le permiten a un robot hexápodo realizar un

desplazamiento mediante un conjunto de acciones no establecidas ni periódicas, lo cual se conoce como locomoción libre. Los algoritmos presentados en este trabajo utilizan técnicas de lógica difusa para tomar la decisión sobre qué pata será movida de acuerdo a la información que se tenga en un instante dado. La valoración de los algoritmos se realizó mediante la simulación del robot. En este trabajo también se describe una medida cuantitativa de estabilidad del robot, la cual es aplicable para robots con una morfología rectangular. En la figura 2.14 se muestran imágenes del prototipo del robot hexápodo en el simulador.



La figura 2.14. Imágenes del robot hexápodo en simulación. Tomadas de (Gorrostieta y Vargas 2004)

Berardi et al. (1996) describe a DANIELA, un sistema neuro-difuso (*neuro-fuzzy*) para controlar un robot hexápodo. El sistema está basado en un dispositivo neuronal hecho a la medida, el cual puede implementar perceptrones multicapa, funciones de base radial o lógica difusa. El sistema implementa algoritmos de control inteligente mezclados con algoritmos neuro-difusos con autómatas de estados finitos, para controlar un robot hexápodo. Para el control del robot hexápodo se utiliza un control jerárquico el cual consiste en un controlador neuro-difuso y un autómata de estados finitos. Dicho control se organiza en tres capas: control de coordinación de movimiento, control de patas y control de las posiciones de los empalmes. El

controlador neuro-difuso se utiliza en el control de la coordinación de movimiento. La figura 2.15 muestra imágenes de dicho robot.

La figura 2.15. Robot de Berardi. Tomada de (Berardi et al. 1996)

Ding (1996) argumenta que usando lógica difusa el usuario es capaz de definir reglas lingüísticas para especificar un mapeo no lineal entre las señales de entrada de los sensores y las señales de salida de los actuadores, para así proporcionar un marco para programar un sistema embebido. Usando un robot hexápodo como cama de prueba, implementaron lógica difusa sobre un microcontrolador el cual controla dicho robot. En resumen, el trabajo de Ding muestra la implementación de un sistema de control basado en lógica difusa en un sistema embebido, probado en un robot hexápodo.

Al usar técnicas de lógica difusa para crear un sistema de control, se tienen dos desventajas principales. La primera de ellas es la dificultad de estudiar la estabilidad del sistema y la segunda es la necesidad de que un experto suministre su conocimiento. La segunda desventaja conlleva ciertos problemas como son: la dificultad de hacer explícito el conocimiento del experto, la dificultad de representación de dicho conocimiento sin llegar a empobrecerlo, y la posible incoherencia de la información proporcionada por el experto. Además, al usar lógica difusa se tiene un elevado número de

parámetros relacionados y no existen métodos sistemáticos y generales para el ajuste de los mismos.

2.4.5 Arquitectura subsumción

El enfoque orientado a metas invierte una gran cantidad de esfuerzo de cómputo en el procesamiento de los datos necesarios para tomar una decisión, es por eso se desarrollaron nuevas formas de realizar sistemas de control basados en el paradigma SA, (Maes & Brooks 1990), con la finalidad de tener sistemas que pudieran responder rápidamente.

Maes & Brooks (1990) fueron de los primeros investigadores que trataron de resolver el problema de diseñar un control para un robot hexápodo (ver figura 2.16) más adecuado a los diseñados hasta ese entonces. Ellos definieron un algoritmo que permitía a un robot hexápodo basado en comportamiento aprender a partir de la retroalimentación, positiva y negativa, cuando se activaban acciones individuales de un conjunto predeterminado. De acuerdo con la filosofía de los robots basados en comportamiento, el algoritmo de control se distribuye totalmente: cada una de las acciones individuales intenta independientemente descubrir (i) su relevancia, i. e. si está totalmente correlacionada con la retroalimentación positiva, y (ii) las condiciones bajo las cuales llega a ser confiable, i. e. las condiciones en las cuales se maximiza la probabilidad de recibir la retroalimentación positiva y al mismo tiempo se reduce al mínimo la probabilidad de recibir la retroalimentación negativa.

La principal carencia que tiene este enfoque es que el robot tiene que invertir tiempo en una fase de aprendizaje para probar acciones repetidamente y reconocer su efecto en el ambiente, así que cuando su ambiente sufre un cambio significativo, i. e. introducción de elementos nuevos al ambiente, el

robot necesita volver a invertir tiempo en la fase de aprendizaje para desplazarse de manera estable en el nuevo ambiente.

Figura 2.16. Prototipo del robot de Brooks. Tomada de [URL_Brooks_Robot]

En (Ferrell 1995), se exponen las dificultades a las que se enfrentan los diseñadores de sistemas de control para robots con un gran número de sensores, actuadores y comportamientos. Además, explora la factibilidad de usar la cooperación entre controladores locales para alcanzar a realizar un comportamiento global correcto. Ferrell diseñó e implementó un sistema de control para un robot hexápodo llamado HANNIBAL. Ferrell descompuso una tarea de control global de un robot hexápodo, e. g. caminar, en un conjunto de tareas de control más simples que la tarea global, para diseñar el sistema de control. HANNIBAL es un robot hexápodo compuesto por una gran variedad de sensores y actuadores sofisticados, con la finalidad de tener mayor fiabilidad de las lecturas de los sensores y de los resultados de las acciones. El sistema de control obtenido por Ferrell está basado en la arquitectura de control subsumción de Brooks, y es capaz de ejecutar una diversidad de comportamientos, e. g. caminar y evadir de obstáculos.

En (Celaya & Albarral 2003) se presenta una estructura de control para un robot hexápodo el cual fue previamente desarrollado y simulado en IRI (*Institut de Robòtica i Informàtica Industrial*). Posteriormente se probó sobre un robot hexápodo con 2 DOF en cada pata. Más tarde esta misma

estructura se ha implementado en un robot con tres grados de libertad por cada pata (LAURON III, de FZI, *Forschungs Zentrum Informatik*). Los principales aspectos que requirieron una atención especial fueron la implementación de los módulos de subsumción (Brooks 1984), la arquitectura de control nativa del robot, y las dificultades para llevar a cabo la coordinación de los movimientos de las patas con los actuadores disponibles. La estructura de control que se utilizó en IRI es centralizada. Celaya probó dicha estructura de control pero con una arquitectura descentralizada en un robot hexápodo con 2 grados de libertad por pata (Genghis II). Finalmente este nuevo sistema de control fue probado en simulación en un robot hexápodo con 3 DOF. Los autores hicieron una descomposición de tareas en varios espectros de habilidades como se propone en la arquitectura subsumción de Brooks. Ambos robots se muestran en la figura 2.17.

Figura 2.17. a) Lauron III, b) Genghis II. Tomadas de [URL_Subsumción_Robots]

2.4.6 Control usando aprendizaje por refuerzo

La idea de dotar a un robot de una cierta capacidad de aprendizaje surge de la motivación de tener robots lo suficientemente autónomos para reducir a lo mínimo la intervención del diseñador, esto es, dotar al robot de la capacidad de adaptarse a su medio a pesar de que éste cambie. Como ya se ha mencionado, por sus características físicas los robots con patas son deseables debido a la potencia que tienen de poder desenvolverse en

ambientes parcialmente desconocidos, pero para explotar tales características se debe dotarlos de un mecanismo de control que les permita adaptarse a cambios inesperados de su ambiente.

En todos los trabajos anteriores se busca encontrar ese mecanismo de control. Un nuevo enfoque para lograr crear ese mecanismo es el uso de aprendizaje por refuerzo, el cual ha demostrado ser adecuado para dotar de autonomía a un robot. Este enfoque se ha utilizado para el diseño de sistemas de control para robots con patas, debido a que trabaja de una manera más sencilla de como lo hacen los controles biológicamente inspirados, además de que no necesita tener un sistema mecatrónico y de sensores muy sofisticados, de nuevo como los robots biológicamente inspirados. En contraste con los sistemas de control orientados a objetivos y metas, el diseño de sistemas de control usando aprendizaje por refuerzo es relativamente más sencillo.

Lee et al. (2006) reportan un uso acertado del aprendizaje por refuerzo en problemas de evasión de obstáculos con un robot cuadrúpedo (ver figura 2.18). El algoritmo que presentan estos autores se basa en una descomposición jerárquica de dos niveles de la tarea, en la cual el control de alto nivel selecciona una configuración específica de las patas y el control de bajo nivel genera los movimientos continuos para mover cada pata del robot a una posición específica. El control de alto nivel usa un estimado de la función de valor para guiar su búsqueda, en este caso orientada a encontrar una configuración de las patas del robot que le permita mantenerse estable; el valor estimado se obtiene usando un algoritmo de aprendizaje semi-supervisado. El controlador de bajo nivel se obtiene por medio de la búsqueda de política sobre campos potenciales. Finalmente, los autores demostraron que su robot lograba contender con una variedad de obstáculos que no fueron considerados en el tiempo de entrenamiento.

Figura 2.18. Robot de Lee. Tomada de (Lee et al. 2006)

La desventaja de este enfoque es que puede tomar un tiempo considerable para converger a una política buena, probablemente no la óptima. Esto es debido a que el robot no posee información alguna acerca del modelo del ambiente y debido a eso tiene que pasar por una fase de entrenamiento para adquirir un modelo del ambiente.

2.4.7 Procesos de Decisión de Markov (MDPs)

En los últimos años, los MDP han sido aplicados en problemas de navegación robótica obteniendo buenos resultados. Sucar (2004) propuso una estructura para resolver problemas de decisión complejos basándose en la división del problema en problemas más pequeños que pueden resolverse independientemente, y en la combinación de estas soluciones para obtener una solución global óptima. Cada parte del problema es representada como un MDP que se resuelve independientemente. Cada MDP envía un valor para cada posible acción a un "árbitro", el cual se encarga de seleccionar las mejores decisiones que en conjunto maximicen la ganancia conjunta. Es aquí en donde se introduce el concepto de MDPs jerárquicos. En este trabajo se propone diseñar un control eficiente que le permita a un robot hexápodo desplazarse de manera estable usando MDPs jerárquicos, en cuyos fundamentos se abundará en el capítulo siguiente.

Pineau et al. (2003) presentan un algoritmo de control escalable que permite a un robot móvil desenvolverse, tomando decisiones de control de alto nivel, bajo consideraciones de creencia probabilística. Los autores dan información acerca de la estructura de control de un robot y de los MDPs jerárquicos para proponer un algoritmo de control probabilístico jerárquico. Con dicho algoritmo, el robot aprende conjuntos de subtareas específicas y de políticas. El control desarrollado fue exitoso al ser probado en un robot móvil que asistía a una enfermera.

2.5 Discusión

En este capítulo se hizo una revisión de la historia y del estado del arte de los robots polípedos, principalmente hexápodos.

El enfoque de control orientado a metas y objetivos requiere un gran esfuerzo en la definición del modelo matemático. Para obtener un buen modelo se requiere que se haga una caracterización muy fina de todos los factores que afectan al ambiente y al robot, como son sensores, actuadores, incertidumbre de acciones, cambios en el ambiente, etc. Por ello, los sistemas de control bajo este enfoque pueden llegar a ser complejos e insuficientes.

Los sistemas de control biológicamente inspirados requieren sistemas mecatrónicos y sistemas de sensores muy sofisticados, con el fin de minimizar el impacto de la incertidumbre en el resultado de las acciones de un robot. Como se mencionó, los sistemas de control biológicamente inspirados parten de la idea de tener sistemas de sensado y actuación muy precisos. Teniendo en cuenta lo anterior, este enfoque se concentra principalmente en desarrollar sistemas de control a un nivel muy detallado para aprovechar al máximo las características físicas, y los sistemas de

sensado y actuación del robot. Para alcanzar tal objetivo con frecuencia se utilizan redes neuronales, las cuales, al igual que los sistemas de control basados en lógica difusa, proporcionan buenos resultados pero tienen la desventaja, al no ser descriptivos, de no facilitar el estudio de su estabilidad pues no existen procedimientos sistemáticos para el ajuste de sus parámetros y de carecer de una metodología general de diseño. Además, en el caso de los sistemas basados en lógica difusa se requiere que un experto suministre conocimiento sobre la locomoción y estabilidad del sistema de un robot hexápodo, en este caso.

Por otro lado, tenemos el enfoque de aprendizaje por refuerzo para obtener un sistema de control, pero como ya se mencionó, puede llegar a ser muy tardado el tiempo que le toma al robot encontrar las relaciones entre percepciones y acciones en dicho control, sin tener la certeza que el sistema de control aprendido por el robot es el control óptimo. Esto es debido a que en este tipo de enfoque no se utiliza ningún modelo de referencia de donde el robot pueda partir para obtener un sistema de control óptimo.

Debido a que el sistema de control involucra la especificación de una relación entre las señales de entrada y las señales de salida para los actuadores, es necesario tener un método el cual permita obtener dicha relación de una manera más descriptiva y clara. Aún más, necesitamos saber si el sistema de control bajo ciertas consideraciones iníciales o modelo inicial es el mejor. Ahora bien, para evitar caer en los problemas de los enfoques orientados a metas y objetivos, este modelo inicial no debe de ser tan sofisticado, i. e. no usar modelos que consideren leyes físicas, matemáticas, de conservación, etc. y además debería permitirnos considerar de manera transparente la incertidumbre que existe en el resultado de las acciones del robot, i. e. no tener que caracterizar explícitamente esa incertidumbre, para así evitar la fuerte inversión en sistemas mecatrónicos y sistemas de sensado

sofisticados. Por estas razones, el trabajo de esta tesis se enfoca en obtener un sistema de control que cubra los aspectos anteriores.

Después de revisar los trabajos de Sucar y Pineau, la idea de utilizar MDPs para obtener un sistema de control óptimo es una buena alternativa para cubrir con los requerimientos anteriores, debido a que los MDPs proporcionan modelos más descriptivos, los cuales son útiles para analizar la respuesta del robot a situaciones específicas del mundo, así como las relaciones entre los actuadores del robot para generar configuraciones estables que permitan el desplazamiento. Los MDPs trabajan con un modelo, el cual incorpora información mínima acerca del entorno y del robot mismo. Dicho modelo nos permite considerar la incertidumbre que existe en los resultados de las acciones de un robot. Los MDPs son modelos probabilistas sustentados con una matemática formal, quiere esto decir que tienen metodología general de diseño y debido a los métodos con los cuales se resuelven los MDPs tenemos garantía de que el resultado es el óptimo. Esto es, considerando la información proporcionada al robot y la forma en que se caracterizaron los estados y acciones del robot, este último siempre ejecutará la mejor acción orientada al logro de una meta específica.

En los trabajos sobre MDPs revisados no se encontró reporte sobre el uso de MDPs para construir sistemas de control para robots con patas. Nuestro trabajo es pionero, hasta donde sabemos, en la aplicación de MDPs para el control de este tipo de robots.

CAPÍTULO III: Procesos de Decisión de Markov y Procesos de Decisión de Markov Jerárquicos

3.1 La necesidad de adaptación

Las capacidades de aprendizaje de los robots primitivos (Raibert 1986) eran nulas. El principal interés de Raibert (1986) era programar a sus robots con ciertos comportamientos que ellos mismos elegían basados en su propia experiencia. Esto acarreó tres problemas fundamentales, que se discuten a continuación.

En primer lugar, sucede muy a menudo que los **comportamientos** que los diseñadores de robots consideran adecuados, demuestran ser **poco útiles o incluso contraproducentes** al llevarlos a la práctica. Esto se debe a que en algunos casos, la única manera de definir estos comportamientos es a partir de simulaciones y de posteriores refinamientos de tales simulaciones. En otros casos, los comportamientos del robot se definen a partir de modelos matemáticos, aplicando métodos de ingeniería de control, con los que ocurre algo parecido al llevarlos a la práctica: el modelo matemático no se ajusta totalmente a lo esperado debido a factores no considerados. En el primer caso se termina dotando al robot con una serie de reglas y parámetros *ad-hoc*, que no surgen realmente de un conocimiento profundo del propio robot y de su interacción con su entorno, sino que se obtienen a partir de múltiples pruebas de ensayo y error. Y en el segundo la construcción del modelo matemático puede convertirse en una tarea no trivial.

El segundo problema en el control de robots es **la falta de capacidad de adaptación de un robot** cuyo comportamiento fue diseñado en las formas

anteriormente descritas. Un robot sin capacidad de aprendizaje puede funcionar adecuadamente en un entorno controlado, pero al enfrentarse a un entorno distinto o aún el mismo entorno modificado, las acciones que antes eran adecuadas pueden volverse inútiles.

El tercer problema consiste en la **dificultad de optimizar el consumo de energía y los movimientos del robot**. La optimización del consumo de energía y de los movimientos del robot pueden calcularse de manera teórica, empleando simulación [URL_Fleifel]. Sin embargo, por las mismas razones expuestas en los párrafos anteriores, es muy probable que los cálculos teóricos no se ajusten a la perfección a las características particulares del robot y del entorno. Si el robot tuviese la capacidad de calcular su propio consumo y aprender cuáles son las acciones que consiguen reducir dicho consumo al mínimo para cumplir un objetivo, esto podría suponer grandes ahorros en tiempo y energía en la vida útil del robot.

Los problemas descritos previamente podrían solucionarse al menos parcialmente proporcionando a los robots capacidades de aprendizaje. En concreto, las técnicas de aprendizaje por refuerzo pueden ayudar a solucionar cada uno de estos tres problemas [URL_Fleifel].

Los paradigmas tradicionales para el control de robots, Jindrich et al. (1999), (Maes & Brooks 1990), por sí solos adolecen de la capacidad de dotar al robot con herramientas para sobrevivir en un entorno abierto y dinámico. Es por ello que se considera de tanta importancia proporcionar a los robots con mecanismos de aprendizaje y adaptación.

Como lo que se desea en esta tesis es brindar a un robot la suficiente autonomía para que se desempeñe en un ambiente dinámico, en este capítulo describimos las formas con las que se puede dotar a un robot de autonomía.

3.2 La consideración de incertidumbre

La resolución de problemas de razonamiento, planificación y aprendizaje bajo condiciones de incertidumbre ha recibido una considerable atención en la comunidad científica durante los últimos años. De hecho, la mayor parte de los problemas planteados en el campo de la robótica o de las interfaces hombre-máquina se caracterizan por estar sometidos a múltiples fuentes de incertidumbre que deben ser consideradas a la hora de diseñar sistemas de control y planificación robustos.

Considérese a modo de ejemplo el robot hexápodo descrito en esta tesis, el cual debe desplazarse de forma autónoma y estable en un entorno no estructurado. Para ello, el robot puede ejecutar diversas acciones, como levantar cierta pata, apoyarla, desplazarla. En la práctica, el efecto real de estas acciones no es del todo fiable, puesto que al tratar de realizar cualquiera de las acciones descritas anteriormente puede suceder que no se alcance una posición o la realización precisa de alguna acción. Lo mismo sucede con las observaciones realizadas por los sensores del robot para percibir el entorno, debido a que las lecturas de los sensores no son cien por ciento confiables. A pesar de todos estos errores e incertidumbres, el robot debe utilizar la información sobre las acciones ejecutadas y las observaciones recibidas para seleccionar las próximas acciones a ejecutar que le permitan conseguir su objetivo final de navegación.

En definitiva, se plantea un problema en que un determinado agente, un robot en nuestro caso, debe tomar decisiones sobre las acciones a realizar para modificar el estado del mundo y su misma condición, e. g. las posiciones de sus patas, hasta conseguir un cierto objetivo. En última instancia, se trata de un problema de planificación bajo condiciones de incertidumbre que afectan tanto el resultado de las acciones como la percepción del entorno.

La Planificación basada en Teoría de Decisiones (DTP, *Decisión-Theoretic Planning*) (Feldman & Sproull 1977, Boutilier et al. 1999) es una extensión de la planificación clásica que permite resolver problemas de toma secuencial de decisiones en sistemas en los que el efecto de las acciones es incierto y la información sobre el entorno incompleta. Para ello, se introduce el concepto de utilidad de una acción, que mide hasta qué punto el resultado de esa acción puede beneficiar o contribuir a la consecución del objetivo global de planificación.

De entre los modelos probabilistas o probabilísticos más utilizados en la DTP cabe destacar los Procesos de Decisión de Markov (MDP, *Markov Decision Processes*), aplicables a aquellos sistemas que cumplen la conocida propiedad de Markov; toda la historia pasada del sistema puede resumirse en su estado actual. Un MDP contempla el resultado estocástico de las acciones, describiéndolo mediante funciones probabilísticas de transición de estados. El resultado de la planificación en un MDP no es una secuencia de acciones a ejecutar como sucede en la planificación clásica (Fikes & Nilsson 1971), sino una política que determina la acción a seleccionar en función del estado actual que va tomando el sistema a lo largo de la ejecución de la tarea. Una parte sustancial de este capítulo se dedica a revisar los fundamentos de los MDPs.

3.3 Procesos de Decisión de Markov

En general, en los problemas de decisión las acciones adoptadas por el agente determinan no sólo la recompensa inmediata sino el siguiente estado del entorno, al menos probabilísticamente. Por lo tanto, el agente toma en cuenta el siguiente estado y la recompensa cuando decide tomar una acción determinada. En estos casos, el modelo óptimo considerado para toda la ejecución determinará cómo tomar en cuenta los valores obtenidos en el futuro. El agente debe ser capaz de aprender a partir de recompensas

demoradas: o sea, puede obtener una secuencia de pequeñas recompensas inmediatas primero, para finalmente llegar a un estado donde se obtiene un valor de recompensa alto. Es decir, el agente debe aprender cuál de las acciones es deseable tomando en cuenta la o las recompensas que pueden obtenerse en un futuro arbitrariamente lejano.

Los modelos de Markov se han aplicado con éxito en la resolución de problemas de navegación en el campo de la robótica, como ejemplos tenemos los trabajos de Simmons & Koenig (1995), Cassandra et al. (1996), Kaelbling et al. (1998), Koenig & Simmons (1998), Nourbakhsh et al. (1995) y Thrun (2000).

3.3.1 Formalización

Un MDP es un modelo matemático de un problema el cual explícitamente considera la incertidumbre en las acciones del sistema. La dinámica del sistema está determinada por una función de transición de probabilidad.

Por otra parte, para cualquier MDP siempre hay una política $\pi : S \to A$ óptima, que permite decidir en cada estado qué acción tomar de manera que se maximice la utilidad esperada. Esta política π es *estacionaria,* i. e. no cambia en función del tiempo, y *determinista,* i. e. siempre se elige la misma acción cuando se está en el mismo estado.

Formalmente, un MDP M es una tupla $M = \langle S, A, \Phi, R \rangle$, (Ocaña M., 2005), donde:

- S es un conjunto finito de estados del sistema.
- A es un conjunto finito de acciones, que se ejecutan en cada estado.
- $\Phi : AxS \to \Pi(S)$ *:* es la función de transición de estados dada como una distribución de probabilidades y la cual asocia un conjunto de posibles

estados resultantes de un conjunto de acciones en el estado actual. La probabilidad de alcanzar un estado s' realizando la acción a en el estado s se escribe $\Phi(a, s, s')$.

- $R: S \text{ x } A \rightarrow R$ es una función de recompensa. $R(s, a)$ es la recompensa que el sistema recibe si lleva a cabo la acción a en el estado s. Esta función define la meta que se quiere alcanzar.

Cabe mencionar que en este trabajo se ha adoptado la notación de Ocaña M. (2005), para la definición formal de un MDP. Sin embargo la función de transición y la función de recompensa pueden representarse como $\Phi : SxAxS \rightarrow \Pi(S)$ y $R: S\text{x}AxS \rightarrow R$ respectivamente debido a que en ambos casos el resultado de cada función está asociada a un estado al que se llega con la acción realizada.

Una política para un MDP es una asociación $\pi : S \rightarrow A$ que selecciona una acción por cada estado, es decir, define cómo se comporta el sistema en un determinado estado, y puede verse como un mapeo de los estados a las acciones.

La función de valor V: indica lo que es "bueno" a largo plazo. Es la recompensa total que un agente puede esperar acumular empezando en ese estado, de alguna manera representa las predicciones de recompensas. Se buscan hacer acciones que den los valores más altos, no la recompensa inmediata mayor.

Las recompensas están dadas por el ambiente, pero los valores se deben estimar o aprender con base en las observaciones. Aplicando aprendizaje por refuerzo se aprenden las funciones de valor mientras el agente interactúa con el ambiente.

La solución a un MDP es una política que maximiza su valor esperado. Dos métodos comunes para resolver MDPs y determinar la política óptima son iteración de valores e iteración de política, (Ocaña M., 2005).

La propiedad de Markov dice: “No importa qué acciones se hayan llevado a cabo para alcanzar el estado actual, porque el estado actual es suficiente para decidir cuál debe de ser la acción futura”.

3.3.2 Políticas

Puesto que en un MDP el estado actual del entorno se considera completamente observable, i. e. que el agente sabe con exactitud en qué estado se encuentra, el único problema a resolver es determinar la acción a ejecutar en función de dicho estado, para conseguir el objetivo final. No se trata de un problema trivial, puesto que el efecto de las acciones no es determinístico. La única información de la cual se dispone para determinar la acción óptima a ejecutar es: la función de transición de estados T que caracteriza la incertidumbre en el efecto de las acciones, y la función de recompensa R que está relacionada con la utilidad de las acciones en función del objetivo final.

Por otro lado, un MDP es un proceso secuencial de toma de decisiones, y en este sentido es posible trabajar en dos contextos distintos. En el primero de ellos, conocido como de horizonte-finito, el agente sólo actúa durante un número finito y conocido de pasos k. Es obvio que en tal caso el objetivo final a la hora de seleccionar las acciones será maximizar la suma total de las recompensas obtenidas durante dichos pasos, tal como se representa en la ecuación (3.1),

$$E\left[\sum_{t=0}^{k-1} r_t\right] \qquad (3.1)$$

en donde r_t es la recompensa obtenida en el paso t.

En otras ocasiones, el agente actúa durante un número infinito o indefinido de pasos, recurriéndose en estos casos al modelo de horizonte-infinito. El objetivo también es maximizar la recompensa obtenida a largo plazo, utilizándose generalmente para ello un modelo con factor de descuento γ, $0 < \gamma < 1$, en el que el agente debe maximizar la ecuación (3.2).

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \qquad (3.2)$$

El factor de descuento permite tener en cuenta las recompensas futuras asegurando que la sumatoria tiende a un valor finito. Además, las recompensas recibidas en un futuro inmediato tienen más valor que las recibidas en el futuro a largo plazo. Cuanto mayor sea el factor de descuento (más próximo a 1), mayor es el peso que tienen las recompensas futuras sobre la decisión actual.

Existen dos tipos de políticas: las estacionarias y las no estacionarias. Una política estacionaria $(a = \pi(s))$ especifica directamente, para cada estado, la acción a realizar independientemente del momento o tiempo en que se encuentra el proceso. Este tipo de políticas se utiliza principalmente con procesos de horizonte-infinito, puesto que al no existir un límite de pasos, no existe ningún motivo para que el agente cambie su estrategia al elegir las acciones. Una política no estacionaria $(a = \pi_t(s))$, sin embargo, asigna una acción u otra al mismo estado en función del momento en que se encuentra

el sistema. Este tipo de políticas se utiliza en procesos de horizonte finito, puesto que es conveniente modificar la estrategia a seguir en función del número de pasos que quedan para finalizar el proceso. Así, la política $a = \pi_t(s)$ permite seleccionar la acción a ejecutar en el estado *s* cuando quedan *t* pasos para finalizar el proceso.

Para un mismo MDP pueden definirse múltiples políticas. Sin embargo, una política será tanto mejor cuanto mayor sea la recompensa que obtiene a largo plazo, siendo éste el criterio que permite seleccionar entre todas ellas, una política óptima.

Desafortunadamente, las ecuaciones (3.1) y (3.2), que representan la suma de todas las recompensas futuras obtenidas por el agente, son meramente conceptuales, puesto que resulta imposible predecir las recompensas futuras en un sistema cuyo resultado de las acciones es estocástico, y en el que por lo tanto la evolución de su estado y las recompensas recibidas en el futuro no pueden conocerse *a priori*.

En lugar de ello, y como criterio para comparar diferentes políticas, se utiliza una función auxiliar conocida como función de valor $(V_\pi(s))$, que para una determinada política asigna a cada estado, un valor numérico. Una política es tanto mejor cuanto mayor sea su función de valor sobre los estados.

3.3.3. Función de valor

La función de valor $V_\pi(s)$ determina la utilidad de cada estado *s* suponiendo que las acciones se escogen según la política π. Se trata de un concepto distinto al de recompensa. La función de recompensa asigna, para cada una de las acciones que pueden ejecutarse en cada estado, un valor numérico

que representa la utilidad inmediata de dicha acción. Sin embargo, la función de valor asigna a cada estado un valor numérico que representa la utilidad de dicho estado a largo plazo. Este valor numérico no es la suma exacta de recompensas futuras, que no se puede predecir, sino una aproximación probabilística que se calcula a partir de las funciones de transición ϕ y de recompensa R. Por ejemplo, un agente puede realizar en cierto estado una acción con recompensa positiva, que sin embargo le lleve a un estado que tenga un valor de utilidad negativo. Desde este punto de vista, es mucho más benéfico realizar la acción que prometa una mejor recompensa a largo plazo.

Supóngase una política π en un contexto de horizonte finito con k pasos. La función de valor asociada a esta política, $V_{\pi,k}(s)$ es la recompensa total "esperada" (no real) al ejecutar la política π empezando en el estado s durante k pasos. Obviamente, si *k=1*, $V_{\pi,1}(s) = r(s,\pi_1(s))$; es decir, si sólo se dispone de un paso o el proceso se encuentra en el último paso, la función de valor coincide con la recompensa obtenida al ejecutar la acción dada por la política. De esta manera, para cualquier otro horizonte, la función de valor puede calcularse recursivamente en retroceso según la ecuación (3.3), (Ocaña M., 2005):

$$V_{\pi,t}(s) = r(s,\pi_t(s)) + \gamma \sum_{s' \in S} p(s' \mid s,\pi_t(s)) \cdot V_{\pi,t-1}(s') \qquad (3.3)$$

Por consiguiente, el valor de horizonte t (es decir, cuando quedan t pasos para finalizar el proceso) para el estado s se calcula sumando a la recompensa inmediata $r(s,\pi_t(s))$ el valor esperado en los restantes *t-1* pasos del proceso ponderado por el factor de descuento. Este valor esperado en los pasos restantes se calcula sumando, para todos los posibles estados de

destino a los que puede pasar el proceso en el siguiente paso, el producto de su valor de horizonte *t-1* por la función de transición de estados.

En el contexto de horizonte-infinito con factor de descuento, la función de valor de cada estado sólo depende de la política y no del número de pasos futuros, y viene dada por la ecuación (3.4):

$$V_\pi(s) = r(s,\pi(s)) + \gamma \sum_{s' \in S} p(s' | s,\pi(s)) \cdot V_\pi(s') \qquad (3.4)$$

En este caso, planteando la ecuación (3.4) para cada uno de los estados que componen el MDP, se obtiene un sistema lineal de ecuaciones, siendo la función de valor V_π la solución del mismo.

3.3.4 Políticas óptimas

Resolver un MDP consiste en encontrar la política óptima, una directiva de control que maximiza la función de valor sobre los estados. Teóricamente, es posible obtener todas las posibles políticas para un MDP y a continuación escoger entre ellas, aquella que maximiza la función de valor. Sin embargo, este método es computacionalmente costoso, puesto que el número de políticas crece exponencialmente con el número de estados. Existen, sin embargo, otros métodos que permiten seleccionar la política óptima aprovechando la propiedad de que ésta será también localmente óptima para cada estado individual.

Howard (1960) demostró que, para el caso más general de horizonte-infinito, existe una política estacionaria $\pi*$ que es óptima para cualquier estado inicial del proceso. La función de valor para esta política viene dada por la solución de la ecuación de Bellman (3.5):

$$V^*(s) = \max_a \left(r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) \cdot V^*(s') \right), \forall s \in S \qquad (3.5)$$

Dada la función de valor óptima, la política óptima responde a la ecuación (6), donde $\arg\max_a(f(a))$ significa el valor de *a* que maximiza *f(a)*.

$$\pi^*(s) = \arg\max_a \left(r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) \cdot V^*(s') \right) \qquad (3.6)$$

El método tradicional para obtener políticas óptimas es la programación dinámica (Bellman 1957, Bertsekas 1995). Dos de los métodos más utilizados son el de Iteración de Valor y el de Iteración de Política. Ambos se basan en modificar las utilidades de los estados vecinos de manera que satisfagan las ecuaciones de Bellman. La repetición de este proceso de modificación local en cada uno de los estados durante un número suficiente de iteraciones hace converger las utilidades de los estados individuales hacia sus valores correctos. Estos métodos permiten obtener la política óptima fuera de línea, esto es, ayudan a que el agente no invierta tiempo en aprender la política probándola en tiempo de ejecución.

Los algoritmos 1 y 2 resumen respectivamente, los métodos de iteración de política e iteración de valor. El primero consiste en calcular la utilidad de cada uno de los estados y con base en estas utilidades, seleccionar una acción óptima para cada uno de ellos. El segundo funciona escogiendo una política, y luego calculando la utilidad de cada estado con base en dicha política. Luego actualiza la política correspondiente a cada estado utilizando las utilidades de los estados sucesores, lo que se repite hasta que se estabiliza la política.

Algoritmo 1. Iteración de política

1. Datos de entrada

   pol_estable – tabla para la política óptima

   b – tabla de una política inicial

   θ _ número positivo pequeño

2. Inicialización:

   $V(s) \in \Re$ y $\pi(s) \in A(s)$ arbitrariamente $\forall s \in S$

3. Evaluación de política:

   Repetir

   $\Delta \leftarrow 0$

   Para cada $s \in S$

   $v \leftarrow V(s)$

   $V(s) \leftarrow r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s' \mid s, \pi(s)) \cdot V(s')$

   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   Hasta que $\Delta < \theta$

4. Mejora de política

   $pol\_estable \leftarrow verdadero$

   Para cada $s \in S$

   $b \leftarrow \pi(s)$

   $\pi(s) \leftarrow \arg\max_a \left(r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s, a) \cdot V(s') \right)$

   Si $b \neq \pi$ entonces $pol\_estable \leftarrow falso$

   Si $pol\_estable$ entonces para, sino ir a 2

Algoritmo 2. Iteración de valor

1. Datos de entrada

   pol_estable – tabla para la política óptima

   b – tabla de una política inicial

   θ _ número positivo pequeño

2. Inicialización:

   $V(s) = 0 \quad \forall s \in S$

3. Repetir

   $\Delta \leftarrow 0$

   Para cada $s \in S$

   $v \leftarrow V(s)$

   $V(s) \leftarrow r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s' | s, \pi(s)) \cdot V(s')$

   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   Hasta que $\Delta < \theta$

4. Regresar una política determinística que satisfaga:

$$\pi(s) \leftarrow \arg\max_a \left(r(s,a) + \gamma \sum_{s' \in S} p(s' | s, a) \cdot V(s') \right)$$

Hasta aquí hemos discutido todo lo concerniente a MDPs, para dar un panorama amplio acerca del formalismo que existe detrás de los mismos.

3.4 Aprendizaje por refuerzo

El aprendizaje automático propone métodos específicos que permiten a los robots autónomos aprender de su interacción con el entorno, y además, aprender mientras se encuentran inmersos en dicho entorno.

En algunos ambientes, muchas veces se puede obtener sólo cierta retroalimentación, recompensa o refuerzo, e. g. valores de ganancia o perdida. El refuerzo puede darse en un estado terminal y/o en estados intermedios. Los refuerzos pueden ser componentes o sugerencias de la utilidad actual a maximizar, e. g. buen movimiento. En aprendizaje por refuerzo (RL, *Reinforcement Learning*) el objetivo es aprender cómo mapear situaciones a acciones para maximizar una cierta señal de recompensa.

El aprendizaje por refuerzo es el problema de conseguir que un agente actúe en un entorno de manera que maximice la recompensa que obtiene por sus acciones [URL_Fleifel].

En nuestro trabajo, el *agente* es un robot hexápodo, al hablar de *entorno* nos referimos al ambiente o espacio en el cual estará funcionando nuestro robot y la *recompensa* es un valor escalar que indicará lo deseable que es una situación para el robot. La recompensa puede tomar valores tanto positivos como negativos. Fisiológicamente, podría compararse un valor de recompensa negativo con el dolor y un valor positivo con el placer.

Cada vez que el robot ejecuta una acción, recibe un valor de recompensa. Estas recompensas no tienen por qué estar asociadas directamente con la última acción ejecutada, sino que pueden ser consecuencia de acciones anteriores llevadas a cabo por el robot.

No es sencillo establecer una correspondencia directa entre acciones y recompensas obtenidas. Es posible que la misma acción llevada a cabo en dos momentos diferentes devuelva recompensas distintas debido a la secuencia previa de acciones.

La promesa del RL es que se refina el programa de control de los agentes mediante premio y castigo sin necesidad de especificar cómo realizar una

tarea. A diferencia a otro tipo de aprendizaje, e. g. aprendizaje supervisado, en RL:

- No se le presentan al agente pares entrada - salida.
- El agente tiene que obtener experiencia útil acerca de los estados, acciones, transiciones y recompensas de manera activa para poder actuar de manera óptima.
- La evaluación del sistema ocurre en forma concurrente con el aprendizaje.

En RL un agente trata de aprender un comportamiento mediante interacciones de prueba y error en un ambiente dinámico e incierto. En general, al sistema no se le dice qué acción debe tomar, sino que él debe de descubrir qué acciones producen el máximo beneficio.

El robot y el entorno interactúan en una secuencia de instantes de tiempo t = *0, 1, 2, 3, …* En cada instante de tiempo t, el robot identifica una representación del estado del entorno $s \in S$, donde S es el conjunto de posibles estados. Con base en esto, el robot selecciona una acción $a_t \in A(s_t)$, donde $A(s_t)$ es el conjunto de acciones disponibles en el estado s_t. En el instante de tiempo posterior, y en parte como consecuencia de la acción llevada a cabo, el robot recibe una recompensa numérica, $r_{t+1} \in \Re$, y pasa a un nuevo estado, s_{t+1}.

En cada momento de tiempo, el robot lleva a cabo un mapeo entre las representaciones de los estados y las probabilidades de seleccionar cada una de las acciones posibles. Llamamos a este mapeo la ***política*** del robot, y la denotamos por π_t, donde $\pi_t(s,a)$ es la probabilidad de que $a_t = a$ si $s_t = s$. Los distintos métodos de aprendizaje por refuerzo especifican de qué manera

cambia el robot su política como resultado de la experiencia que va adquiriendo de su interacción con el entorno. Como ya se mencionó, el objetivo del robot es maximizar a largo plazo la suma de las recompensas obtenidas. Este proceso se ilustra en la figura 3.1.

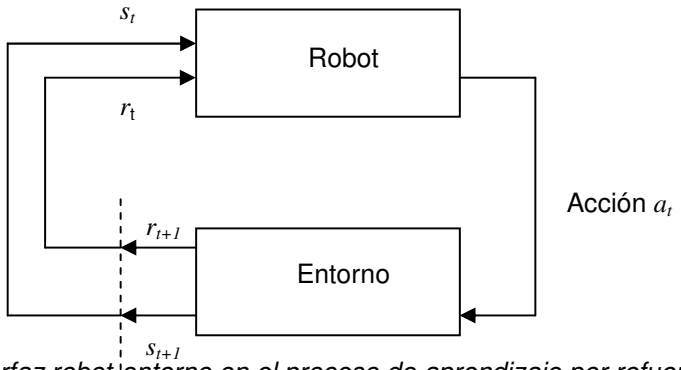


Figura 3.1 Interfaz robot-entorno en el proceso de aprendizaje por refuerzo

Como menciona uno de los proponentes del RL, Sutton & Barto (1998), lo mejor del marco de trabajo antes descrito es que es extremadamente flexible, lo que permite que sea aplicado a muchos problemas diferentes de maneras muy distintas. Hay que decir que dentro de este marco de aprendizaje las acciones pueden ser tanto controles directos, e. g. los voltajes aplicados a los motores del brazo de un robot, como decisiones de mayor nivel, e. g. escoger entre la ruta 1 y la ruta 2. De manera similar, los estados pueden obtenerse de una gran variedad de formas. Pueden estar completamente determinados por percepciones de bajo nivel, como las lecturas directas sobre los sensores, o pueden ser más abstractas y de mayor nivel, como las descripciones simbólicas de los objetos de una habitación, tal y como se pretendía en los inicios de la robótica [URL_Shakey].

También hay gran flexibilidad en la manera de diseñar los estados. Un estado puede consistir en las percepciones directas del robot recibidas en un momento dado, o puede estar compuesto en parte por datos almacenados

de percepciones anteriores. Incluso pueden existir estados completamente abstractos.

En general, el ambiente es no-determinístico, esto es, tomar la misma acción en el mismo estado puede dar resultados diferentes. Sin embargo, en los MDPs se asume que el ambiente es estacionario, esto es, las probabilidades de cambio de estado no cambian o cambian muy lentamente. Algunos aspectos importantes relacionados con lo anterior son:

- Se sigue un proceso de prueba y error, y
- la recompensa puede estar diferida en el tiempo

Otro aspecto importante es el balance entre exploración y explotación. El agente debe balancear la **exploración** de nuevos estados y acciones para obtener nueva información que le permita evitar óptimos locales, y la **explotación** de estados y acciones ya aprendidos y con una alta recompensa inmediata que le garantice una recompensa acumulada aceptable.

La caracterización de esta problemática está dada por Procesos de Decisión de Markov o MDP. Esto es, usando MDPs en donde se tiene un modelo, la exploración se realiza fuera de línea, i. e. no en tiempo de ejecución del robot, y exhaustivamente para obtener las mejores acciones.

Los métodos de aprendizaje aplicados en robótica pueden verse como una forma de aprender una correspondencia entre las entradas y salidas para maximizar ciertas medidas de desempeño, mientras el sistema se encuentra en operación. El aprendizaje por refuerzo no resulta tan eficaz en ambientes altamente dinámicos. En realidad esto es un problema para cualquier método de aprendizaje en línea y no supervisado. En el área de la robótica este problema es considerable debido a que un cambio en el entorno que los

seres humanos podemos considerar no significativo, puede ser un cambio significativo para un robot. Debe tenerse en cuenta que la capacidad de procesamiento de los robots actuales se encuentra muy limitada, y que sus capacidades perceptivas son mínimas comparadas con las de los seres vivos. Por ello, un ligero cambio en las condiciones del entorno puede desconcertar e impactar a los robots de forma importante.

Algunos de los diferentes problemas en robótica para los cuales se está empleando el aprendizaje por refuerzo son los siguientes:

- Aprendizaje de secuencias de movimientos y optimización de recursos: (Vargas et al. 2002), Kimura (1997), Kimura (1999), DeJong (1994), Boone (1997), Tham (1994).
- Coordinación de sistemas multi - agente: Uchibe (1995), Uchibe (1996a), Uchibe (1996b).
- Navegación: Mataric (1994).
- Aprendizaje distribuido y colectivo, e. g. aprendizaje por imitación, aprendizaje de la comunicación: Billard (1997), Hayes (1994), Demiris (1996).
- Aprendizaje de comportamientos: Asada (1996).
- Tareas más complejas / aprendizaje por refuerzo deliberativo: Dietterich (2000).
- Control: Schaal (1994), Mahadevan (1991).

3.5 Procesos de Decisión de Markov Jerárquicos

Los algoritmos clásicos para modelar y solucionar MDPs requieren la representación explícita de estados y de acciones. Muchos de los algoritmos clásicos de solución, como iteración de valor e iteración de política, necesitan explorar el espacio entero de estados durante cada iteración. Sin embargo, en problemas de aprendizaje que implican un enorme número de estados

explícitos y quizá de acciones también, esta exploración se vuelve exponencial cuando el número de variables en el sistema aumenta. Por ejemplo, supongamos que un robot es requerido para entregar el correo y café para los profesores en un laboratorio. Este sistema puede incluir características del estado, por ejemplo, si un profesor desea o no el café, si hay o no correo para él, qué tan lejos se encuentra la oficina de cada profesor del cuarto de café y de correo, y así sucesivamente. Si deseamos enumerar todos los estados explícitos, crecerá exponencialmente el número de estados de acuerdo al número de características que se deseen incluir.

Así, si utilizamos un enfoque clásico para tratar MDPs grandes, podemos tener dificultades de procesamiento debido al tamaño del espacio de estados. Es por esto que utilizamos un enfoque jerárquico para resolver MDPs grandes, algunos trabajos relacionados podemos mencionar (Parr & Russell 1997), (Gu 2003), (Laroche 2000), (Bakker et al. 2005).

Regresando al ejemplo anterior, entregar el café a ciertas oficinas o la recolección del café en cierta sala requiere lapsos diversos de tiempo, y es natural que uno planee o aprenda cómo resolver dichas situaciones en diferentes instantes de tiempo, lo cual se conoce como abstracción temporal. Existen varios enfoques ya establecidos para realizar la abstracción temporal y descomposición de la tarea, los cuales discutiremos en las siguientes secciones.

La problemática que se plantea con el ejemplo anterior, es factible de ser resuelta con un solo MDP, pero como se discutió, se tendría un modelo con un gran número de estados y probablemente de acciones, así que como también se adelantó, es preferible hacer una abstracción temporal del modelo original del MDP. A esta abstracción se le conoce como *Proceso de*

Decisión de Markov Jerárquico (*HMDP*, *Hierarchical Markov Decision Process*).

En este trabajo se aplicará un HMDP para simplificar la complejidad con la que se resolvería el problema usando solamente un MDP.

Es conveniente señalar que la concepción de un HMDP no consiste únicamente en agrupar partes de un MDP muy grande, sino que implica también un análisis sobre la manera de descentralizar el problema. Además, no existen guías de diseño de HMDPs, la concepción de diseño de un HMDP es un problema abierto.

A continuación describimos algunas de las formas de concepción de un HMDP.

3.5.1 Opciones

Sutton et al. (1999) extendieron la noción usual que se tiene de las acciones que componen un MDP, creando así una estructura modificada de MDP que incluye *opciones,* las cuales son políticas de ciclo cerrado para tomar determinadas acciones en un periodo de tiempo determinado.

Formalmente, dado un MDP $M = \langle S, A, \Phi, R \rangle$, una opción consiste de tres componentes: una política no determinista $\pi : SxA \to [0,1]$, una condición de paro $\beta : S \to [0,1]$ y un conjunto de iniciación $I \subseteq S$.

Una opción está disponible en un estado s_t en el tiempo t si y solo si $s_t \in I$. El conjunto de iniciación y la condición de paro de una opción restringen conjuntamente el rango de aplicación de dicha opción. Si la opción

$Op =< I, \pi, \beta >$ es tomada, entonces las acciones son seleccionadas de acuerdo a la política π hasta que la opción termine estocásticamente de acuerdo a β. En particular, una *Opción de Markov* se ejecuta como sigue. Primero, en el tiempo t, la acción a_t es seleccionada de acuerdo a una distribución de probabilidades $\pi(s_t, \cdot)$. Después de ejecutar la acción seleccionada el ambiente cambia, a esto se le conoce como la transición al estado s_{t+1}, donde la opción termina con probabilidad $\beta(s_{t+1})$, o también el agente puede continuar tomando la acción a_{t+1} de acuerdo a la política $\pi(s_{t+1}, \cdot)$ y posiblemente termine en el siguiente paso s_{t+2} de acuerdo a $\beta(s_{t+2})$ y así sucesivamente. El agente puede seleccionar otra opción cuando termine de ejecutar la acción que está ejecutando.

Sutton et al. (1999) también proponen que un MDP con un conjunto de opciones es un *Proceso de Decisión Semi-Markoviano (SMDP, Semi-Markov Decision Process),* cuyas transiciones de estados son estocásticas y la duración de cuyas opciones es estocástica pero no depende del tiempo.

Resultados empíricos en (Sutton et al. 1999) muestran que problemas de planeación y aprendizaje pueden ser resueltos mucho más rápidamente introduciendo y rehusando opciones apropiadamente. Esto se debe a que teniendo opciones adecuadas, las cuales pueden ser ya óptimas en algunos estados, se puede converger a una política óptima global más rápidamente. Además, Sutton et al. (1999) mostró que la estructura de las opciones puede ser aprendida, usada e interpretada mecánicamente.

3.5.2 Macro-acciones

Las macro-acciones son descritas y usadas para resolver MDPs de manera jerárquica en (Hauskrecht et al. 1998). En esencia, las macro-acciones son lo

mismo que las opciones. Sin embargo, este enfoque de macro-acciones es ligeramente diferente de las opciones a partir de la descripción dada por Sutton et al. (1999). En el trabajo de Sutton, $p(s,o,s')$ y $r(s,o)$ son definidas para todos los estados en el espacio de estados. Resolver el problema en este contexto requiere del uso explícito de la programación dinámica sobre todo el espacio de estados, lo cual no reduce el tamaño del espacio de estados.

Una macro-acción es vista como una política local sobre una región específica del espacio de estados, la cual termina de aplicarse cuando la región es dejada. Un *MDP abstracto* es construido a partir sólo del estado que se encuentre en los límites de regiones adyacentes y su espacio de acciones sólo consiste de macro-acciones. Un MDP jerárquico con macro-acciones generalmente tiene un espacio de estados y de acciones mucho más pequeños que los del MDP original. De esta manera, se simplifica la complejidad de resolver un MDP con espacio de acciones y estados grandes.

Hauskrecht et al. (1998) basó su modelo en una *descomposición basada en regiones* de un MDP determinado $M = \langle S, A, \Phi, R \rangle$. Como se define en (Dean & Lin 1995), S es dividido en regiones S_1, S_2 , ... S_N . Los estados que se encuentran en los límites de las regiones adyacentes incluyen dos tipos de estados para cada región S_i, *estados periféricos de salida de* S_i y *estados periféricos de entrada de* S_i.

Una macro-acción para una región S_i es una política $\pi_i : S_i \to A$. La terminación de una macro-acción es mucho más específica que la de una opción: la condición de inicio para π_i en el tiempo *t* es $s_t \in S_i$ y la condición de terminación en el tiempo *t* es $s_t \notin S_i$.

Hauskrecht et al. (1998) argumenta que la solución de un *MDP aumentado*, un MDP original extendido con un conjunto de macro-acciones, tiene garantía de ser una solución óptima, aunque no se puede asegurar un beneficio en términos computacionales. Realmente Hauskrecht et al. (1998) está interesado en reducir la abstracción del MDP obtenida reemplazando el conjunto de acciones primitivas con un conjunto de macro-acciones y reconstruyendo el modelo en los estados marginales.

Ventajas computacionales significativas pueden obtenerse debido a que un MDP simplificado o abstracto normalmente tiene un espacio de estados sustancialmente más pequeño que el del MDP original. Sin embargo, la política del MDP abstracto puede corresponder únicamente a una política sub-óptima del MDP original (Hauskrecht et al. 1998).

Varios trabajos proponen métodos para asegurar que las políticas para un MDP aumentado sean lo más aproximadas posible a la política óptima del MDP original (Parr 1998).

3.5.3 Conjuntos Markovianos de tareas

El enfoque de opciones descrito anteriormente puede ser considerado como una técnica de la familia de enfoques de descomposición para resolver MDPs grandes. Descomponer un MDP significa que un MDP es definido en términos de un conjunto de subprocesos o tareas "pseudo-independientes" (Singh & Cohn 1998) o automáticamente descompuesto en tales subprocesos. Esto es, se usa un algoritmo para realizar dicha descomposición (Boutilier et al. 1998). Las soluciones de esos sub-MDPs son usadas para construir una solución global aproximada. En el enfoque de opciones, el espacio de estados del MDP original es dividido en regiones

para formar sub-MDPs. Sin embargo, hay otra clase de técnicas las cuales tratan a los sub-MDPs como procesos concurrentes (Meuleau et al. 1998).

Meuleau et al. (1998) asume que los MDPs aplicados en problemas de asignación de recursos secuenciales estocásticos están ampliamente acoplados, i. e. los problemas están compuestos de múltiples tareas cuyos valores de utilidad son independientemente aditivos, de modo que las acciones tomadas con respecto a una tarea realizada no afectan el estado de alguna otra tarea. Estos problemas de asignación son modelados con una forma especial de MDPs, *conjuntos Markovianos de Tareas* (MTS, *Markov Sets of Tasks*) para n tareas. Un conjunto Markoviano de tareas es una tupla $\langle S, A, \Phi, R, c, M_g, M_l \rangle$. *S, A, R,* y Φ están definidos como en un MDP normal, c es el costo de solo una unidad del recurso a asignar, M_g la restricción de recurso global sobre la cantidad del recurso total y M_l la restricción de recurso local sobre la cantidad del recurso que puede ser usado en un solo paso. Bajo esta estructura, una política óptima es un vector de políticas locales que maximiza la suma de las recompensas asociadas con cada tarea. Estos MDPs son muy grandes e involucran cientos de tareas, y son útiles en problemas de asignación de recursos secuenciales estocásticos.

La estrategia de aproximación, llamada descomposición de tareas de Markov, se divide en dos fases. Una primera fase “fuera de línea” en la cual las funciones de valor y soluciones óptimas son calculadas para las tareas individualmente, usando programación dinámica. En la segunda fase “en línea”, esos valores son usados para calcular un gradiente para una búsqueda heurística, para calcular la próxima acción como una función del estado actual de todos los procesos. Resultados experimentales demuestran que esta técnica puede resolver MDPs con un número de estados y acciones grande y reducir el tiempo computacional de su proceso.

3.5.4 Máquinas abstractas jerárquicas

El aprendizaje basado en Máquinas Abstractas Jerárquicas (HAMs, *Hierarchical Abstract Machines*), fue propuesto por Parr & Russell (1997). Es otro enfoque de abstracción temporal usado con MDPs con un gran número de estados. De manera similar a las opciones, el uso de HAMs reduce el espacio de búsqueda y proporciona una estructura en la cual el conocimiento puede ser transferido a través de los subproblemas y los componentes de solución pueden ser recombinados para resolver el problema original. En contraste con las opciones, las cuales como ya se ha mencionado pueden ser vistas como políticas locales, las HAMs usan controladores de estado finito no determinísticos para expresar conocimiento *a priori* y así restringir las posibles políticas que el agente puede escoger.

Como se define en (Parr & Russell 1997) una HAM está compuesta de un conjunto de máquinas de estados, una función de transición y una función de inicio que determina el estado inicial de la máquina. Las máquinas de estado son de cuatro tipos: un *estado acción* ejecuta una acción, un *estado llamada* ejecuta otra máquina como subrutina, un *estado selección* selecciona de forma no determinista la siguiente máquina de estados y un *estado de paro* de la ejecución regresa el control al estado de llamada previo. La función de transición determina estocásticamente el resultado de la siguiente máquina de estados, de acuerdo al estado de la máquina de estados actual del tipo de acción o llamada y del ambiente resultante. Una HAM es en efecto un programa el cual, cuando es ejecutado por un agente en un ambiente, restringe las acciones que el agente puede tomar en cada estado.

Resultados experimentales de solución de un problema ilustrativo con 3600 estados usando una HAM, (Parr & Russell 1997), demostraron mejoras

drásticas sobre el algoritmo tradicional de iteración de política aplicado al MDP original sin la HAM. Parr y Russell también aplicaron este enfoque en aprendizaje por refuerzo, llamado HAMQ-learning, modificando la función *Q-learning* con el modelo MDP inducido por la HAM, y éste *HAMQ-learning* parece aprender mucho más rápido que el tradicional *Q-learning*.

Andre y Russell propusieron una lenguaje de programación para HAM (PHAM) (Andre & Russell 2001) el cual es un lenguaje expresivo de diseño del agente que extiende el lenguaje básico del lenguaje de programación HAM de Parr & Russell (1997) con las características del LISP.

3.5.5 Método MAXQ

Dietterich (1998, 2000a, 2000b) propuso un enfoque llamado MAXQ para tratar con problemas de toma de decisiones y aprendizaje por refuerzo grandes, basado en los trabajos de Singh (1992), Kaelbling (1993), Dayan & Hinton (1993) y Dean & Lin (1995). Basado en la hipótesis de que el programador puede identificar submetas y subtareas que ayuden a alcanzar esas metas, MAXQ descompone un MDP original en una jerarquía de MDPs más pequeños y al mismo tiempo descompone la función de valor del MDP original en una combinación aditiva de funciones de valor de los MDPs más pequeños.

Comparado con opciones y macro-acciones, en las cuales las tareas son definidas en términos de políticas locales fijas, y con las HAMs en las cuales las subtareas son definidas usando controladores basados en máquinas de estados no deterministas, MAXQ está orientado a metas y cada subtarea es definida usando predicados de terminación y una función de recompensa local. Dado un MDP $M = \langle S, A, \Phi, R \rangle$ y suponiendo que las políticas son mapeadas desde el espacio de estados al espacio de acciones, MAXQ

descompone a M en un conjunto finito de subtareas $\{M_0, M_1,..., M_n\}$ con la convención de que M_0 es la tarea raíz y resolver M_0 resuelve el MDP completo M.

Una política jerárquica π para $\{M_0, M_1,..., M_n\}$ es un conjunto el cual contiene una política para cada una de las subtareas $\pi = \{\pi_0, \pi_1,..., \pi_n\}$. Cada política de la subtarea π_i selecciona un estado y regresa una acción primitiva para ejecutarla en el estado en el que se encuentra, o el índice de una subrutina para poder invocarla.

Al igual que las opciones y las HAMs, MAXQ también usa una versión resumida del método iteración de valor para optimizar la aproximación de la selección de tarea.

Dietterich (2000) argumenta que dada una descomposición MAXQ, al igual que las opciones o las HAMs, hay un algoritmo para obtener la política óptima jerárquica que alcanza la recompensa prevista más alta entre todas las políticas que conforman la descomposición.

Finalmente, Dietterich (2000) afirma que usando técnicas de abstracción de estados, el método MAXQ no sólo representará un MDP grande de manera compacta, también obtendrá ventajas computacionales en procesar la función de valor descompuesta.

3.5.6 Aprendizaje por refuerzo jerárquico

El aprendizaje por refuerzo jerárquico (*HRL, Hierarchical Reinforcement Learning*) es un enfoque emergente en el cual los métodos de aprendizaje por refuerzo son aumentados con conocimiento previo con respecto a la estructura de comportamiento de alto nivel. La idea fundamental de este

enfoque es que el conocimiento previo debe acelerar considerablemente la búsqueda de una política óptima. En general, todos los formalismos de HRL son vistos como la adición de restricciones al aprendizaje por refuerzo, dichas restricciones limitan o fijan el comportamiento del agente, es decir, hacen que el agente siga directivas previamente establecidas en lugar de aprenderlas. Algunos trabajos en los cuales se aplica este enfoque los encontramos en (Sherstov & Stone 2005), (Marthi et al. 2005).

Lo antes descrito es aplicado principalmente en aprendizaje por refuerzo pero la idea básica puede ser aplicada en la construcción de un MDP. Así, usando este enfoque se crearía un modelo de dos niveles, uno de alto nivel y otro de bajo nivel, el MDP o MDPs de bajo nivel son pequeños en número de estados y se encargan de resolver problemas relativamente simples. El MDP o MDPs de alto nivel es básicamente una abstracción de la tarea deseada, esto es, una especie de secuencia de pasos que debe seguir un agente para alcanzar un objetivo. A esto se le puede considerar como el conocimiento previo que se le agrega a la construcción del modelo. Como se mencionó, este conocimiento restringe el comportamiento de alto nivel del agente. De esta manera, al restringir explícitamente el comportamiento que debe tener el agente para alcanzar una meta, se acota el espacio de búsqueda lo cual redunda en un problema computacionalmente más sencillo, pero no permite que el agente aprenda comportamientos distintos de los que están establecidos.

Cabe mencionar que no solamente se existe el enfoque de MDPs jerárquicos para resolver MDPs grandes, existen otros enfoques entre los cuales podemos mencionar factorización de MDPs (Degris T. et al., 2006), (Reyes A. et al., 2006).

3.6 Discusión

Una tarea de aprendizaje por refuerzo que satisface la propiedad de Markov, es llamada Proceso de Decisión de Markov (MDP).

Como se ha discutido en este capítulo, realmente no existe una definición formal de lo que es un HMDP, pero podemos definirlo de la siguiente manera: Un MDP jerárquico o HMDP es una abstracción temporal o espacial de un MDP complejo. Un HMDP sintetiza una tarea compleja dividida en pequeñas subtareas, las cuales son resueltas de manera independiente. La solución de estas subtareas contribuye a obtener la solución global. Esta división sirve para simplificar los cálculos y acotar el espacio de estados de un MDP muy complejo. Como se ha discutido, existen varias técnicas para tratar con MDPs grandes. Y como se ha descrito, dicha descomposición básicamente consiste en dividir el MDP original en subtareas.

Después de presentar los métodos más conocidos para descomponer MDPs, podemos comparar las ventajas y desventajas de cada uno. La principal ventaja de usar opciones y macro-acciones es que los modelos y propiedades de las opciones o macro-acciones pueden ser fácilmente establecidos en regiones locales, las cuales generalmente tienen un espacio de estados pequeño con respecto al del MDP original y que dichas opciones pueden ser rehusadas posteriormente. La desventaja de usar opciones es que esas políticas locales son fijas. Así, para obtener una política aproximada a la óptima aceptable para el MDP original, tenemos que definir conjuntos convenientes de macros u opciones para cada región, lo cual puede ser difícil, dado que no existe una guía a seguir de la manera adecuada de realizar esa definición de conjuntos.

Por otro lado, HAMs y MAXQ no son fijadas *a priori*. En MAXQ se definen subtareas en términos de predicados de terminación, lo cual requiere que el programador "adivine" el beneficio relativo de los diferentes estados en los cuales la subtarea puede terminar. Esto también puede resultar complicado de realizar, dado que dependerá en gran medida de la experiencia del diseñador. La ventaja de MAXQ es que este método descompone jerárquicamente el espacio de estados del MDP original en subtareas de múltiples capas, y las políticas aprendidas en los niveles inferiores de las subtareas pueden ser compartidas por múltiples tareas padre o del nivel superior. Sin embargo, en MAXQ se usa una técnica de abstracción de estados para eliminar aspectos irrelevantes en el espacio de estados de la subtarea y se descompone la función de valor del problema original en un conjunto de sub-funciones para los subproblemas, de esta manera este método puede resultar menos costoso, computacionalmente hablando, con respecto a los otros. Las HAMs restringen el gran número de posibles políticas a ser consideradas para obtener la óptima o las políticas cercanas a la óptima usando controladores no deterministas, para que así el método pueda alcanzar ventajas computacionales. Pero para calcular una política óptima que es HAM-consistente, necesitamos construir un MDP el cual tiene un enorme espacio de estados. Así que el Método MAXQ puede ser preferible que las HAMS debido a la forma de procesar las políticas.

Finalmente, podemos aplicar la idea de agregar conocimiento previo en la construcción del modelo, pero como ya se mencionó esto restringe al agente para encontrar comportamientos distintos a los que previamente se le han dado a conocer. La ventaja de este enfoque es que la obtención de la política es rápida, y que el diseño del modelo es sencillo.

Los métodos anteriores están enfocados a descomposiciones de MDPs para obtener HMDPs, y por lo general han sido aplicados en problemas de

navegación robótica. Cada una de estas técnicas hace divisiones jerárquicas en capas del MDP original. En los trabajos que usan HMDP, la descomposición de las tareas se lleva en un nivel de abstracción alto, por ejemplo, evitar obstáculos, caminar, reconocer rostros, preparar café, entregar correo, etc. (Gu 2003). Cada una de esas tareas por separado no tiene nada que ver con las otras, pues cada una de ellas tiene un propósito específico, es por eso que se hace una jerarquización de las tareas dando prioridades a aquellas que tienen más importancia e inhibiendo las de menor importancia. Por otro lado, no existen trabajos, hasta donde sabemos, en los cuales se definan y se usen MDPs de manera distribuida y no jerárquica, la principal diferencia entre los MDPs jerárquicos y los descentralizados, es que el caso de los jerárquicos generalmente se tiene un MDP para cumplir una tarea específica, en cambio en los descentralizados como veremos existen más de un MDP que tratan, de manera independiente, de cumplir con la tarea.

CAPÍTULO IV: Modelos Propuestos

Después de revisar la teoría de MDPs y HMDPs podemos afirmar que el uso de MDPs usando un enfoque descentralizado es factible cuando el objetivo total del sistema es uno y sólo uno, o a lo más dos como se discutirá en capítulos posteriores, ya que los subsistemas que conformen al sistema total trabajarán independientemente buscando el mismo objetivo. Por otro lado, cuando se tienen problemas los cuales plantean la realización de múltiples objetivos y para los cuales quiere usarse un MDP para su solución el mejor enfoque es el jerárquico.

En este capítulo se describen los modelos propuestos que más adelante se aplican en experimentos tanto en simulación como en experimentos llevados a cabo con un robot hexápodo físico comercial.

4.1 Modelo 1: MDP con Decisión Centralizada y Acción Descentralizada (DCAD)

4.1.1 Análisis y justificación

Como ya se ha mencionado, este trabajo se enfoca en crear un sistema de control que le permita a un hexápodo con una morfología circular desplazarse de manera estable. Para esto se realizó un análisis del paso que utilizan ciertas arañas con una distribución circular de patas alrededor de su cuerpo, como las que se muestran en la figura 4.1.

Este tipo de arañas utiliza un paso distinto al paso de trípode ya ampliamente estudiado y aplicado a robots hexápodos con morfología rectangular (c. f sección 2.3).

Figura 4.1 Arañas con distribución circular de patas, donde los círculos muestran dicha distribución

El paso que usa este tipo de arañas conserva la distribución circular que tienen sus patas después de completar un determinado ciclo, esto lo consiguen moviendo determinados segmentos de sus patas, esto es, manipulan determinados segmentos de una pata dependiendo de la posición que ocupa ésta alrededor del cuerpo de la araña. La figura 4.2 muestra la distribución y división de las patas de una araña con una distribución de patas circular alrededor de su cuerpo.

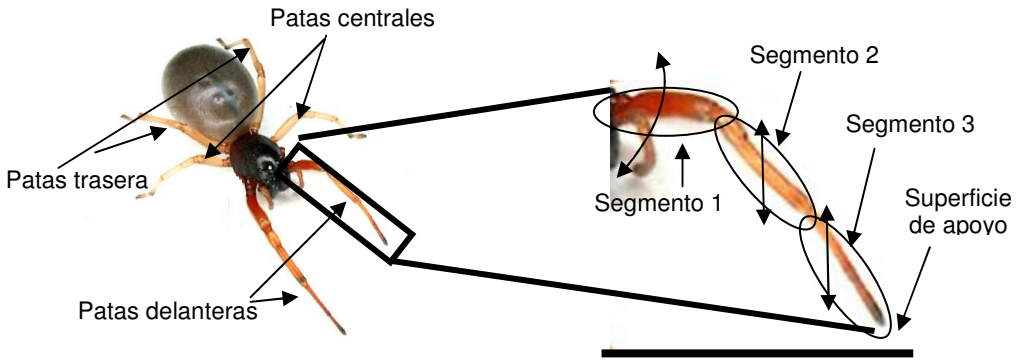


Figura 4.2 Características de las patas de las arañas con configuración circular de patas

En esa figura, puede apreciarse que el segmento 1 realiza movimientos horizontales y los segmentos 2 y 3 realizan movimientos verticales con respecto a la superficie de apoyo de la araña.

En el paso que utilizan estas arañas, las patas delanteras y traseras únicamente mueven los segmentos 2 y 3 de manera adecuada para generar

un movimiento hacia la dirección en que se dirige la araña, mientras que las patas centrales mueven los tres segmentos, pero en especial el segmento 1 para generar un movimiento hacia la dirección en que se dirige la araña. Tomando en cuenta la información anterior, la cual fue obtenida de una observación de la forma de avanzar del tipo de arañas de la figura 4.1, se diseñaron los modelos propuestos en esta tesis.

Como ya se mencionó, en contraste con otros trabajos, en el nuestro no se desea utilizar más sensores y actuadores de los indispensables, ni tampoco se dispone de sensores con gran sensibilidad para el equipamiento del robot. Se espera que el uso de HMDPs nos permitirá manejar de una manera adecuada la incertidumbre que existe en las lecturas de los sensores sin necesidad de tener más sensores o sensores más precisos.

Al decir que el robot será capaz de desplazarse de manera estable nos referimos a que el robot será capaz de moverse de manera autónoma y sin que el robot llegue a caerse dentro de un determinado ambiente. Además, se espera que pueda contender con cambios inesperados en el ambiente, e. g. pendientes y obstáculos, entre otros.

4.1.2 Definición del modelo

Para el diseño de este modelo se utilizó un enfoque jerárquico usando conocimiento previo, esto es, se creó un HMDP usando el enfoque de adición de conocimiento previo.

El conocimiento *a priori* que se le agrega a este modelo, es el paso que debe de seguir el robot para desplazarse de manera estable en el ambiente, lo cual es considerado como una restricción explícita. Dicho paso está inspirado directamente de la manera en que se desplazan las arañas que tienen una distribución circular de patas alrededor de su cuerpo, ilustradas en la figura 4.1.

De acuerdo a la definición que dimos de HMDP (c. f. sección 3.5), para la definición de nuestro primer modelo, hicimos una descomposición del problema original que consiste en controlar las seis patas con un solo MDP, lo cual representa tener un MDP con un gran número de acciones y de estados. Se realizó una descomposición del problema definiendo únicamente los MDPs necesarios para el control adecuado de una sola pata del robot, cada uno de estos MDPs es reutilizado para las otras patas, de modo que los MDPs definidos tienen un menor número de estados y acciones que los que tendría un solo MDP que resolviera el mismo problema.

En este primer modelo definimos una manera de coordinar los MDPs que controlan las patas del robot, para generar un desplazamiento hacia adelante. De modo que, para crear un sistema de control para el robot hexápodo, con este primer modelo utilizamos un HMDP de dos capas. La primera capa o *capa de bajo nivel*, se encarga de realizar acciones básicas de las patas, e. g. levantar una pata o apoyar una pata sobre el plano de apoyo. Dicha capa está compuesta de MDPs los cuales ayudan a que una pata alcance una determinada configuración. La segunda capa o *capa coordinadora*, se encarga de propiciar secuencias de movimiento de las patas del robot. En esta capa se ha agregado el conocimiento *a priori* acerca del paso utilizado por arañas con distribución circular de patas alrededor de su cuerpo. Así, esta segunda capa tiene definida la secuencia de movimientos necesarios de cada una de las patas para generar un movimiento hacia adelante, esto es, esta capa se encarga de aplicar un MDP de la capa de bajo nivel en alguna de las patas, de acuerdo a la secuencia predefinida.

4.1.2.1 Consideraciones

El robot hexápodo físico que se quiere controlar tiene 3 DOF por cada una de las patas. Cada uno de esos DOF corresponde a un segmento de la pata. La figura 4.3 ilustra de manera gráfica la estructura de una pata del robot.

Se realizó una simplificación de las posibles posiciones en las que puede estar un segmento de la pata del robot. Se considera que cada segmento de una pata sólo puede estar en una de tres posiciones, arriba, enmedio o abajo en el caso de los segmentos 2 y 3. Y adelante, atrás y enmedio para el segmento 1, como se muestra en la figura 4.3.

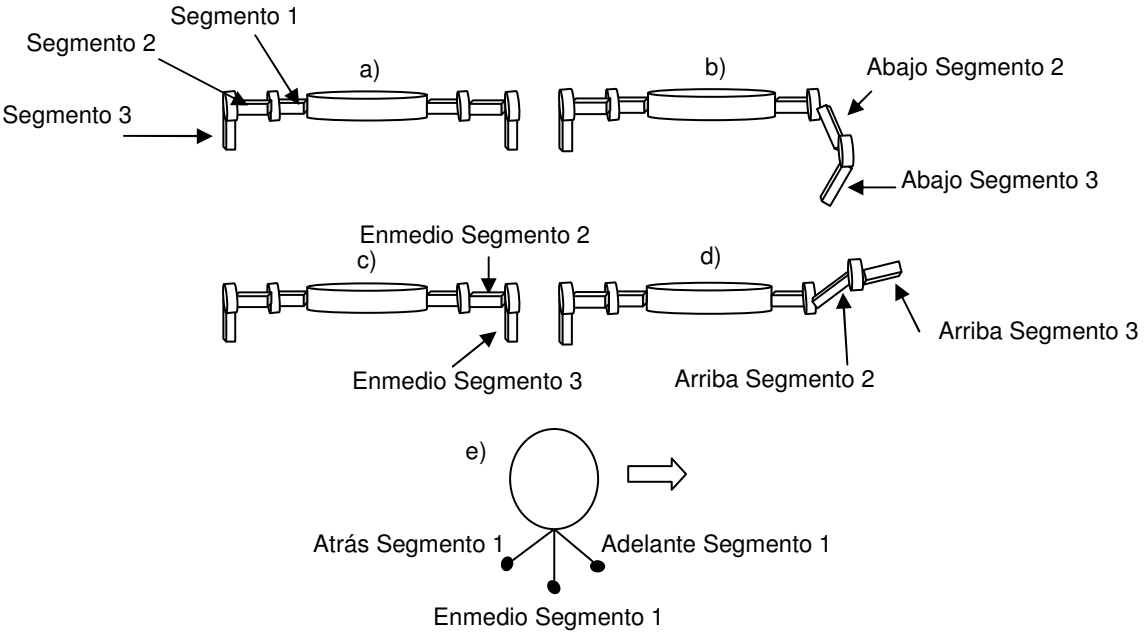


Figura 4.3. a) Segmentos de una pata, b) – d) posibles posiciones de los segmentos 2 y 3 mostrados en la pata derecha, e) posibles posiciones del segmento 1, vista superior

Nótese que si quisiéramos usar sólo un MDP, para la caracterización de los estados de éste, aún tomando en cuenta las consideraciones anteriores, necesitaríamos considerar todas las posibles configuraciones de las seis patas en conjunto. Esto sería equivalente a $(3x3x3)^6$ = 387,420,489

configuraciones posibles. Por otro lado, considerando únicamente dos acciones por segmento se tendrían $(2x2x2)^6$ = 262,144 posibles acciones.

Retomando el análisis de la sección 4.2.1, las posibles configuraciones de una pata de una araña del tipo mostrado en la figura 4.1, de acuerdo a una de tres posiciones ocupada alrededor del cuerpo: trasera, enmedio o delantera, son mostradas en la figura 4.4.Como se observa en la figura 4.4, existen básicamente tres *meta*-configuraciones, no importando qué posición ocupe una pata: trasera, enmedio o delantera. Estas meta-configuraciones se muestran en la figura 4.5.

Una meta-configuración *es* una posición objetivo que cualquier pata, no importando su posición, puede alcanzar. Y dicha posición será la meta a alcanzar de un MDP.

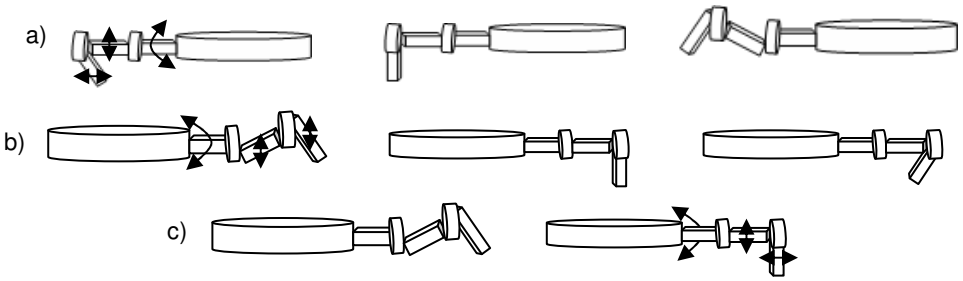


Figura 4.4. Configuraciones posibles de una pata. a) pata trasera, b) pata delantera, c) pata central. Donde ↕ y ↔ indica movimiento vertical y ↻ indica movimiento horizontal, con respecto al plano de apoyo.

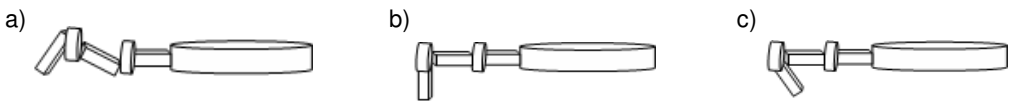


Figura 4.5. Meta-configuraciones, a) meta-configuración que levanta una pata, b) y c) meta-configuraciones que posan una pata

4.1.2.2 Definición de la capa de bajo nivel

Recordemos qué un MDP es una abstracción de un problema general el cual engloba características y consideraciones necesarias para resolver el

problema. Tal abstracción debe reflejar situaciones deseables o metas. Al solucionar el MDP se obtiene la mejor política o directivas de control.

Cuando se usa un enfoque probabilista basado únicamente en el uso de MDP, no es necesaria la incorporación de nueva información para actualizar la política, debido a que toda la información relevante se ha incorporado desde la definición del MDP.

Como ya se ha mencionado, la capa de bajo nivel está conformada por los MDPs que controlan una pata. Para definir estos MDPs analizamos las configuraciones que puede tener una pata del robot. Nótese que las meta-configuraciones de la figura 4.5 sólo involucran posiciones específicas de los segmentos 2 y 3, en estas configuraciones el segmento 1 no es movido, de hecho este segmento será manipulado en la capa de alto nivel.

Partiendo de las meta-configuraciones, necesitamos sólo tres MDPs, uno por cada meta-configuración, pues si utilizáramos un MDP con tres metas alcanzaríamos alguna de las metas pero sería difícil salir de esta una vez llegando a ella. A los MDPs les llamaremos MDP_UP, MDP_DW1 y MDP_DW2. Las metas de MDP_UP, MDP_DW1 y MDP_DW2 son respectivamente, ilustradas, en las figuras 4.5 a), 4.5 b) y 4.5 c). De acuerdo a las configuraciones definidas, estos MDPs se encargan de levantar una pata (MDP_UP) o de posarla en una de dos formas (MDP_DW1 o MDP_DW2).

Una vez identificados los MDPs, hay que definir a cada uno de ellos. Los tres MDPs utilizan en mismo conjunto de estados y de acciones, lo único que variará en sus definiciones es su función de recompensa. El conjunto de estados comprende todas las posibles configuraciones en las que puede estar una pata del robot. Debe recordarse la simplificación de posiciones en las que puede estar un segmento (ver figura 4.3). Considerando que estos

MDPs sólo controlarán dos segmentos de cada pata (ver figura 4.5), tenemos un total de 3x3 = 9 posibles configuraciones, las cuales se muestran en la figura 4.6.

El conjunto de acciones comprende los posibles movimientos que puede realizar una pata, considerando sólo dos de los segmentos que la conforman. La tabla 4.1 muestra el conjunto de acciones de este MDP.

Tabla 4.1. Conjunto de acciones

	Acciones
1	Mover hacia arriba segmento 1
2	Mover hacia abajo segmento 1
3	Mover hacia arriba segmento 2
4	Mover hacia abajo segmento 2
5	No hacer nada

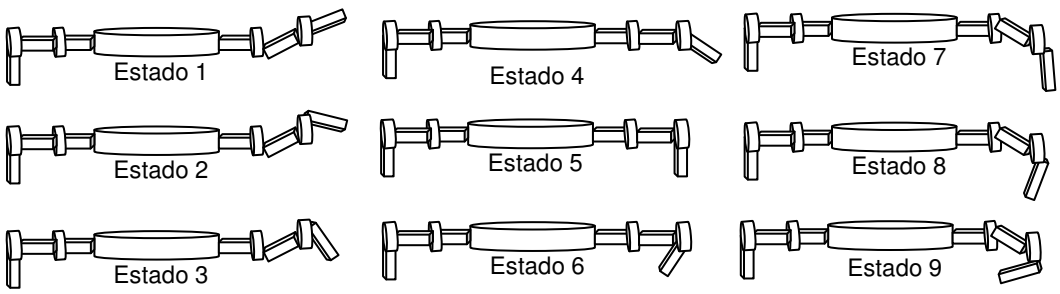


Figura 4.6 Conjunto de estados, ilustrados en la pata derecha

Así, definimos un conjunto de acciones con cinco elementos. Nótese que para estos MDPs utilizamos una acción de no movimiento, la decisión de usar esta acción será explicada posteriormente (c. f. sección 4.2.2.2).

Dado que se tienen nueve estados y cinco acciones, la función de transición será una matriz de 9x9x5. Dado que esta matriz refleja las probabilidades de llegar de un estado a otro ejecutando una determinada acción, esta matriz

será la misma para cada uno de los MDPs. Al definir esta matriz se considera la incertidumbre del resultado de ejecución de una acción. Dicha matriz es mostrada en el apéndice B y la cual fue construida a mano.

Finalmente, las matrices de recompensa las cuales indican la ganancia obtenida de realizar una determinada acción en un estado dado y en las cuales debe verse reflejada la meta que debe alcanzar un MDP se muestran en la tabla 4.2. Como se observa, cada matriz de recompensa refleja la meta que quiere alcanzar cada uno de los MDPs. Esto se logra asignando un valor positivo en el par (estado, acción), lo cual significa, para el robot que estando en ese determinado estado y ejecutando esa determinada acción se puede alcanzar la meta del MDP.

En la tabla 4.2 se muestran las matrices de recompensa de los MDPs MDP_UP, MDP_DW1 y MDP_DW2, donde se ha puesto el valor de 10 en los pares (estado, acción) debido a que estando en ese estado y ejecutando dicha acción es posible alcanzar la meta del MDP.

Tabla 4.2. Matrices de recompensa para los MDPs de la capa de bajo nivel

	Acciones				
Estados	**1**	**2**	**3**	**4**	**5**
1	-1	-1	-1	-1	-1
2	-1	-1	-1	10	-1
3	-1	-1	-1	-1	10
4	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	-1
6	10	-1	-1	-1	-1
7	-1	-1	-1	-1	-1
8	-1	-1	-1	-1	-1
9	-1	-1	-1	-1	-1

a) Matriz de recompensa de MDP_UP

	Acciones				
Estados	**1**	**2**	**3**	**4**	**5**
1	-1	-1	-1	-1	-1
2	-1	10	-1	-1	-1
3	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	10
6	-1	-1	10	-1	-1
7	-1	-1	-1	-1	-1
8	-1	-1	-1	-1	-1
9	-1	-1	-1	-1	-1

b) Matriz de recompensa de MDP_DW1

	Acciones				
Estados	**1**	**2**	**3**	**4**	**5**
1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1
3	-1	10	-1	-1	-1
4	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	10
7	-1	-1	-1	-1	-1
8	-1	-1	-1	-1	-1
9	10	-1	-1	-1	-1

c) Matriz de recompensa de MDP_DW2

4.1.2.3 Definición de la capa coordinadora

Esta capa es la encargada de coordinar los MDPs de la capa de bajo nivel, es decir, es la encargada de indicarles a dichos MDPs el orden y momento en que deben generar un desplazamiento hacia adelante estable. Para definir esta capa, usamos el conocimiento *a priori*, el cual, como ya se ha

mencionado, es el paso que utilizan las arañas con una distribución circular de patas alrededor de su cuerpo. Dicho paso lo hemos caracterizado en ocho movimientos coordinados de las seis patas, los cuales son (ver figura 4.7):

1. Llevar a la meta-configuración 1 las patas 1, 3 y 5
2. Mover hacia adelante la pata 5 y mover hacia atrás la pata 2
3. Llevar a la meta-configuración 2 las patas 1 y 5 y llevar a la meta-configuración 3 la pata 3
4. Llevar a la meta-configuración 1 las patas 2, 4 y 6
5. Llevar a la meta-configuración 3 pata 1 y 5, mover hacia atrás la pata 5 y mover hacia adelante la pata 2
6. Llevar a la meta-configuración 3 la pata 6 y llevar a la meta-configuración 2 las patas 2 y 4
7. Llevar a la meta-configuración 1 las patas 1, 3 y 5
8. Llevar a la meta-configuración 2 la pata 6, llevar a la meta-configuración 3 las patas 4, mover hacia adelante pata 5 y mover hacia atrás pata 2

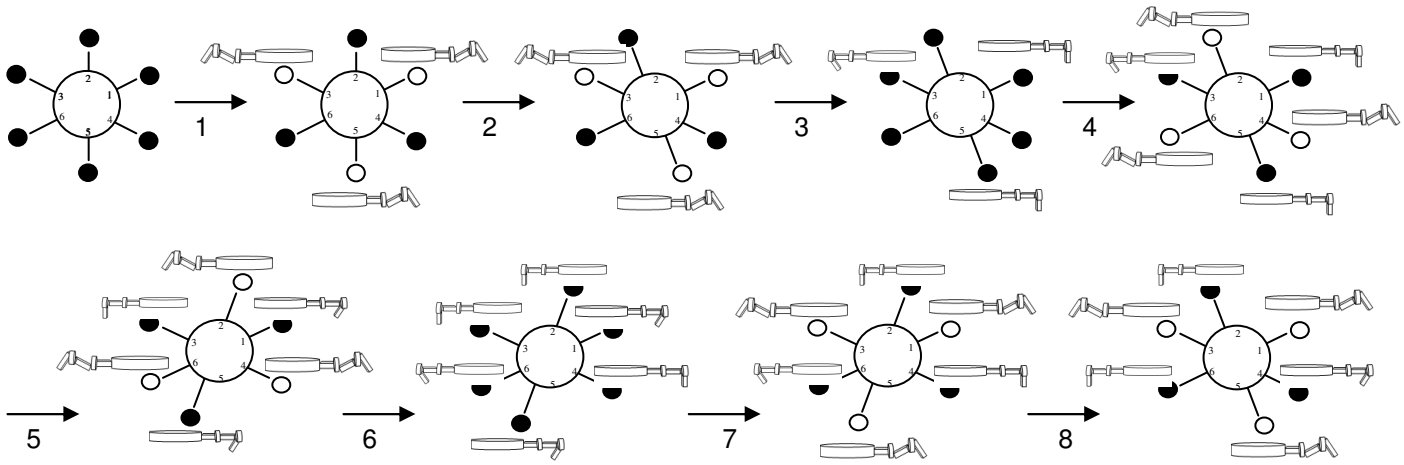


Figura 4.7 Caracterización del paso utilizado por arañas con una distribución circular de patas alrededor de su cuerpo

La indicación llevar una pata a una determinada configuración implica la ejecución de la política del MDP que hace que una pata alcance dicha configuración, e. g. al decir llevar a la meta-configuración 1 las patas 1, 3 y 5 significa que la política del MDP_UP será aplicada tanto en la pata 1, 3 y 5 para que dichas patas alcancen la configuración indicada.

Dicho paso, se basa en el conocimiento *a priori* suministrado al modelo. Usando este conocimiento el robot logra coordinar sus patas. Sin embargo, esta coordinación está impuesta por el diseñador, no fue aprendida por el robot haciendo una exploración de su espacio de estados y de sus acciones.

Finalmente, este primer modelo puede ser visto de manera gráfica como se muestra en la figura 4.8.

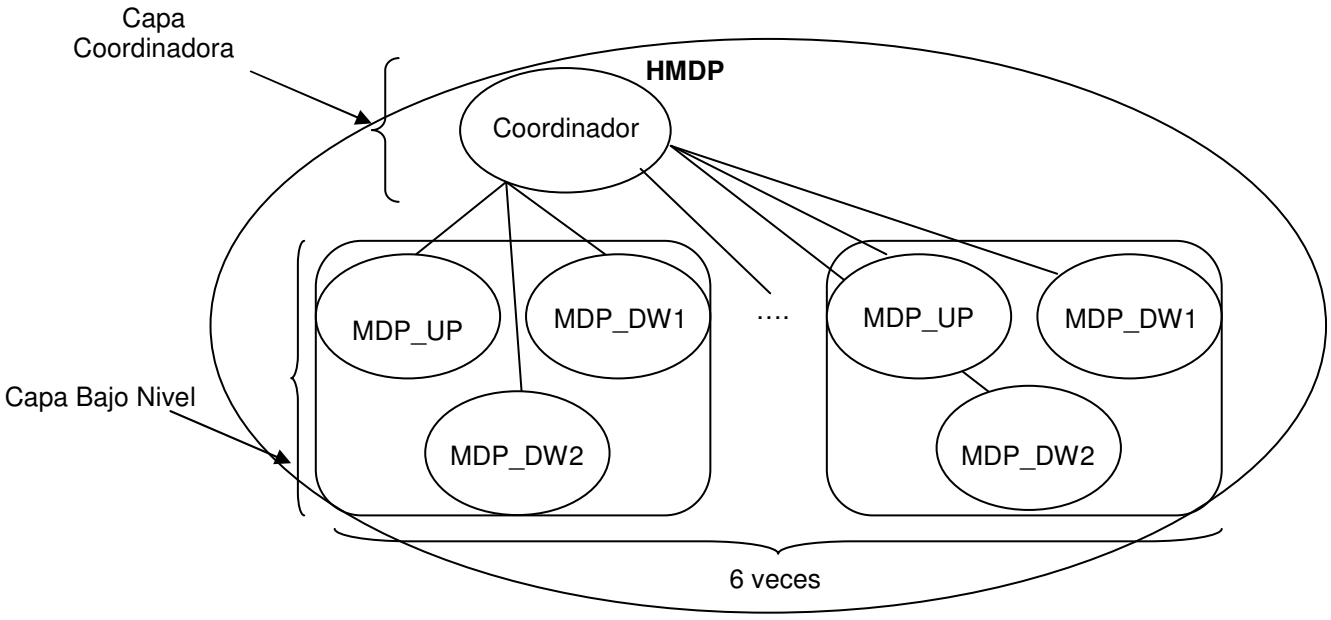


Figura 4.8. Estructura del HMDP del modelo 1

4.2 Modelo 2: MDP con Decisión Descentralizada y Acción Descentralizada (DDAD)

4.2.1 Análisis y justificación

El motivo por el cual se diseñó este segundo modelo, fue buscar una alternativa de descentralización (e. g. Parr & Russell (1997), Sucar (2004)) del MDP original para obtener un nuevo HMDP, de modo que, al igual que en el modelo previo, se obtienen múltiples MDPs más pequeños que el original, pero con la diferencia de que estos MDPs interactúan sin seguir un esquema jerárquico como los del primer modelo, sino siguiendo un esquema de control descentralizado. Además, en este segundo modelo no se le da al HMDP conocimiento *a priori* tan detallado como en el modelo anterior. Al igual que el modelo DCAD, este modelo usa la información obtenida del análisis del paso que utilizan las arañas con una distribución circular de patas alrededor de su cuerpo.

Cuando se quiere segmentar y simplificar un problema global, generalmente muy grande, para ser resuelto de manera distribuida, es necesario crear divisiones lo más independientes posible. Además, es importante que dichas divisiones sean capaces, por sí solas, de resolver parte del problema global sin depender o requerir la intervención de las otras divisiones.

El problema global de esta tesis es hacer que un robot hexápodo sea capaz de desplazarse de manera estable en un ambiente semiestructurado. Dicho problema debe ser segmentado por su complejidad. Una primera división del problema fue la propuesta en el modelo 1, en la cual, en lugar de controlar de manera conjunta y centralizada las seis patas del robot, se realizó una división del problema donde se controlan las patas de forma independiente. Sin embargo, usando esta división es necesaria la intervención de un coordinador que dé turnos de operación a las divisiones para generar un

movimiento estable hacia adelante. Esto es debido a que controlar de manera individual las patas no es suficiente para lograr que el robot avance y al mismo tiempo mantenga su cuerpo despegado de la superficie de apoyo. Es importante señalar que las configuraciones de las patas que permiten generar un desplazamiento hacia adelante pueden no ser útiles para mantener el cuerpo del robot despegado de la superficie de apoyo. Además, debido a la forma en la que están distribuidas las patas del robot y al torque de los servomotores de una pata se requieren al menos dos patas, estó es para robots hexápodos con una distribución circular alrededor de su cuerpo, para lograr estabilidad estática.

Después de un análisis más detallado acerca de la manera de caminar de las arañas con distribución circular de patas alrededor de su cuerpo, se observó que existen ciertos pares de patas, los cuales trabajan de manera independiente a otros pares de patas. Cada par de patas tiene además un paso de movimiento independiente de los otros pares, el cual permite que la araña se mueva hacia adelante y que al mismo tiempo mantenga el cuerpo despegado de la superficie de apoyo. La figura 4.9 muestra tales pares de patas.

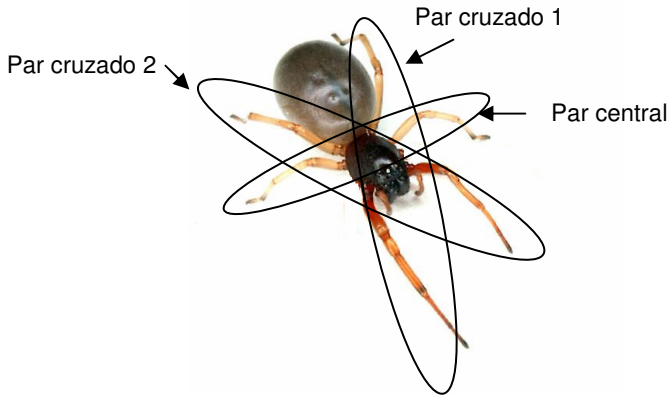


Figura 4.9. Pares de patas en una araña identificadas de acuerdo a su función es el desplazamiento

4.2.2 Definición del modelo 2

En este nuevo modelo se definió un HMDP usando un enfoque descentralizado. Esto es, se rompe el esquema jerárquico y se crean múltiples MDPs que interactúan sin la ayuda de un coordinador para alcanzar un objetivo en común.

Para crear este modelo se consideran los pares mostrados en la figura 4.9 para realizar la nueva división del problema global. Nuestra división del problema consistirá en usar un MDP para controlar cada uno de los pares de patas mostrados en la figura 4.9. Nótese que el *par cruzado 2* puede ser controlado con el mismo MDP que controle el *par cruzado 1,* debido a que su comportamiento es simétrico. De esta forma, nuestro nuevo modelo consistirá de dos MDPs, a los cuales hemos nombrado MDP_C y MDP_M, donde MDP_C controlara el par de patas 1 y 6 y el par 3 y 4 y el MDP_M controla el par 3 y 4 como se ilustra en la figura 4.10.

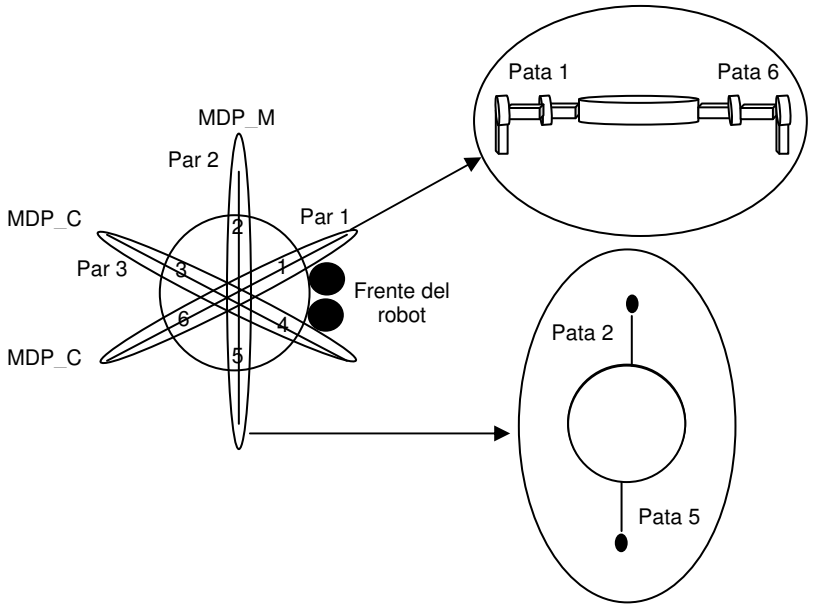


Figura 4.10 Pares de patas a controlar por los MDPs MDP_C y MDP_M

De esta manera, tenemos un HMDP el cual consiste de tres MDPs, cada uno de los cuales con un número menor de estados y de acciones comparados

con el MDP original. Estos MDPs trabajarán de manera independiente, sin la interacción un coordinador, para lograr la meta global.

4.2.2.1 Consideraciones

Las consideraciones acerca de los movimientos válidos de los segmentos de una pata y de las meta-configuraciones expuestas en la sección 4.1.2.1 son usadas para la definición de este nuevo modelo. Ver figuras 4.4 y 4.5.

Cuando un MDP es usado para resolver problemas de locomoción de robots con patas, generalmente la meta de dicho MDP consiste en alcanzar una determinada configuración de patas. Así el objetivo del MDP es aprender secuencias de configuraciones las cuales permitan el desplazamiento de manera estable hacia adelante, no importando en qué configuración se encuentren las patas del robot, estas secuencias serían equivalentes, en un problema de navegación robótica, a aprender las rutas a seguir desde cualquier posición del ambiente para alcanzar una posición meta en el mismo. En problemas de navegación de robots, las rutas dadas por el MDP son rutas que cumplen con dos objetivos principales, llegar a la meta y evitar determinados puntos del ambiente. Similarmente, en problemas de locomoción de robots con patas, es importante que las configuraciones obtenidas puedan alcanzar la configuración objetivo o meta-configuración y eviten ciertas configuraciones no deseadas.

Así, la manera de definir los estados para un MDP que está enfocado a resolver un problema de navegación es distinta a la que se utiliza para un MDPs enfocado a resolver un problema de locomoción de robots con patas. Debido a que lo que se busca en problemas de navegación es que el robot alcance un lugar en su mundo, pero para problemas de locomoción con patas lo que se busca es que el robot genere configuraciones de sus patas lo cuales le permitan caminar. Considerando lo anterior, las aseveraciones que

se hagan en adelante van exclusivamente considerando que estamos trabajando con un problema de locomoción robótica y no de navegación.

Cabe mencionar que para trabajos donde se utilizan MDPs jerárquicos existen dos conceptos, metas y submetas, una meta es un objetivo único, las submetas son subobjetivos. Como se muestra en la figura 4.11 donde la capa superior del HMDP tiene una meta y en la capa inferior existen submetas que son alcanzadas por MDPs, lo cuales dependen de la capa superior.

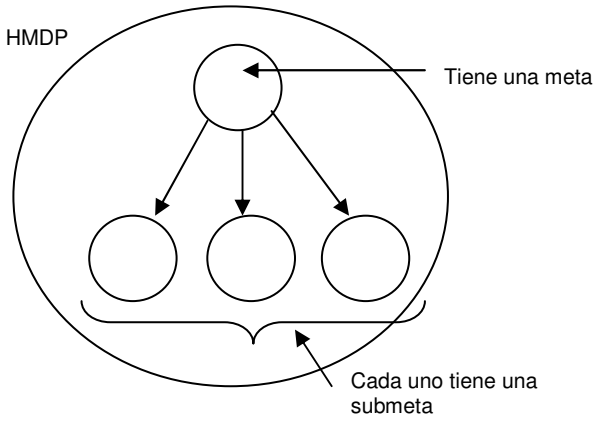


Figura 4.11 Metas en un HMDP

Para el modelo que estamos definiendo, MDP DDAD, definimos lo un nuevo concepto *metas simultáneas.* Las metas simultáneas son metas que no son coordinadas, es decir, que no están supeditadas por una capa superior, como se muestra en la figura 4.12, donde el MDP DDAD tiene una meta, pero en su composición contiene MDPs los cuales tienen metas simultaneas.

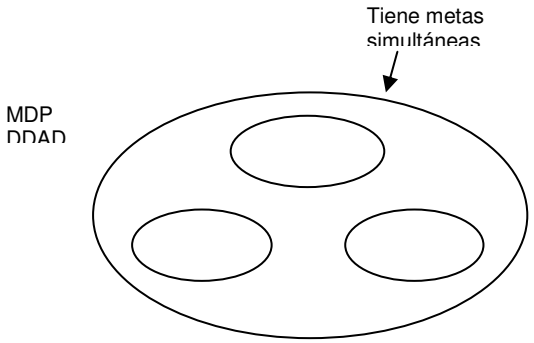


Figura 4.12 Metas en un MDP DDAD

La teoría de MDPs nos dice que un MDP puede tener más de una meta dada explícitamente en la definición de éste, es decir, reflejado en la matriz de recompensa del MDP. En problemas de locomoción, la división del problema en MDPs independientes no es trivial ¿qué pasa cuando se desea alcanzar varias metas simultáneamente?, ¿cómo puede ser construido un MDP que alcance más de una meta, por ejemplo, levantarse y avanzar, o bien, avanzar y mantenerse estable? La división en submetas no parece en estos casos suficiente. Estas reflexiones nos hicieron separar en principio los tipos de metas en exclusivas y simultaneas, para más adelante definir el nuevo modelo.

4.2.2.2 Definición MDP_C

Como ya se ha mencionado, este MDP controlará un par de patas del robot, nuestro espacio de estados estará compuesto por todas las configuraciones válidas de acuerdo a las consideraciones expuestas en la sección anterior.

Este MDP controlará uno de los pares cruzados de la figura 4.9, este par de par de patas tiene la característica de que para generar un movimiento estable hacia adelante sólo manipula los segmentos 2 y 3 de ambas patas y el segmento 1 no es movido. De esta forma, las configuraciones posibles de este par de patas consistirá únicamente en las configuraciones en las que pueda estar este par considerando sólo los segmentos 2 y 3. Así, tenemos un total de (3x3)x(3x3) = 81 configuraciones válidas, las cuales son mostradas en la figura 4.13.

El conjunto de acciones estará compuesto por conjuntos de cuatro acciones básicas. Una acción básica es mover hacia arriba o hacia abajo un segmento determinado de alguna de las dos patas. Los conjuntos de acciones básicas fueron definidos para alcanzar una configuración de un solo movimiento, en lugar de realizar cuatro movimientos para alcanzar una configuración. Ahora

bien, dado que hay que mover dos segmentos por cada pata y cada segmento puede ser movido de dos maneras diferentes, tenemos un total de $(2x2)^2 = 16$ acciones. El conjunto de acciones se muestra en la tabla 4.3.

Debido a que se tienen 81 estados y 16 acciones, la función de transición será una matriz de 81x81x16, esta matriz refleja las probabilidades de llegar de un estado a otro ejecutando una determinada acción. En la definición de dicha matriz se considera la incertidumbre del resultado de ejecución de una acción (dicha matriz no es mostrada en esta tesis debido a sus dimensiones).

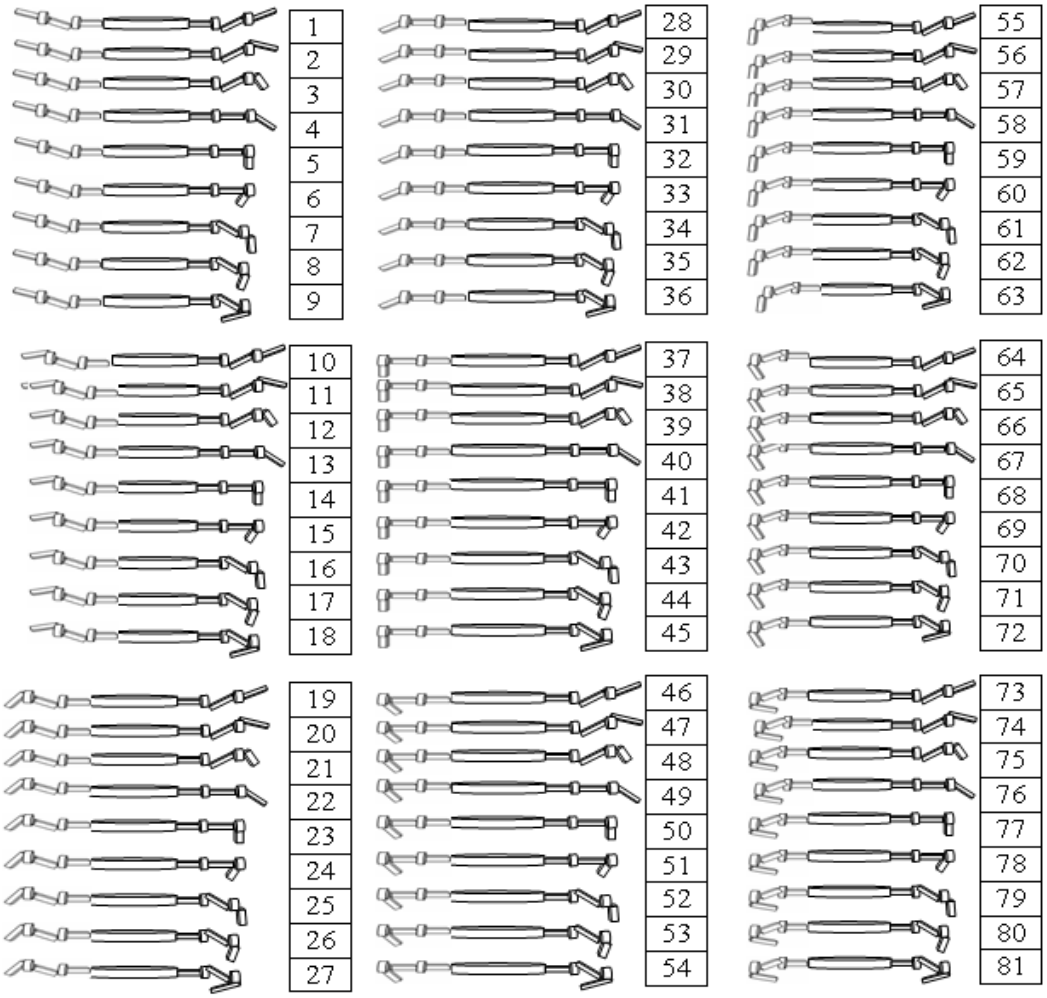


Figura 4.13. Espacio de estados de un par cruzado de patas, donde el número a la derecha de la configuración indica el número de estado con el cual es identificada dicha configuración

Antes de definir la matriz de recompensa del MDP_C, la cual refleja la meta que se quiere alcanzar, recordemos cuál es la meta que quiere alcanzar el MDP_C. La meta de este MDP es lograr que el par de patas que controla genere un paso que permita al robot moverse hacia adelante y además, que dicho par de patas logre mantener levantado de la superficie de apoyo el cuerpo del robot.

Tabla 4.3. Conjunto de acciones, las flechas indican el sentido del movimiento de cada segmento

No. de Acción	Conjunto de acciones básicas			
	Pata 1		Pata 2	
	Segmento 1	Segmento 2	Segmento 1	Segmento 2
1	↑	↑	↑	↑
2	↑	↑	↑	↓
3	↑	↑	↓	↑
4	↑	↑	↓	↓
5	↑	↓	↑	↑
6	↑	↓	↑	↓
7	↑	↓	↓	↑
8	↑	↓	↓	↓
9	↓	↑	↑	↑
10	↓	↑	↑	↓
11	↓	↑	↓	↑
12	↓	↑	↓	↓
13	↓	↓	↑	↑
14	↓	↓	↑	↓
15	↓	↓	↓	↑
16	↓	↓	↓	↓

Lo que se quiere es que este MDP alcance dos metas al mismo tiempo, avanzar y no caerse.

Para resolver el problema anterior hemos definido dos nuevos conceptos, *meta simultánea explícita* y *meta simultánea implícita*. Una meta simultánea

explícita no es otra cosa que una meta tradicional de un MDP, generalmente expresada en la matriz de recompensa con un valor positivo más grande en aquellos pares (estado, acción) que permiten alcanzar el estado meta. La meta simultánea implícita, es una noción propuesta en este trabajo y consiste en una restricción a las acciones que pueda tomar el robot, la cual es expresada en la matriz de recompensa como un valor negativo en los pares (estado, acción) que llevan a un estado no deseable al robot. De modo que al resolverse el MDP, se obtendrá una política la cual evitará que el robot alcance o llegue a pasar por ciertos estados, la meta simultánea implícita, siempre persiguiendo la meta simultánea explícita.

Cabe señalar que la definición de metas a través de valores de valores de recompensa positiva, y de restricción a través de valores de recompensa negativos se han utilizado tradicionalmente en el área, su aplicación en el contexto de submetas y metas simultáneas es distinto en nuestro trabajo.

Considerando lo anterior, es claro que la definición de la matriz de recompensa es fundamental para hacer que el robot alcance o evite determinados estados. Ahora bien, las metas simultáneas de este MDP son alcanzar una configuración la cual permita generar un movimiento hacia adelante y mantener el cuerpo del robot despegado de la superficie de apoyo. Tomando en cuenta las consideraciones acerca de la matriz de recompensa, podemos reflejar ambas metas en dicha matriz y así garantizar que alcance un estado meta y se evite pasar por aquellos estados que hagan que el cuerpo del robot toque o caiga en la superficie de apoyo.

Otro aspecto que ilustra la dificultad de alcanzar metas simultáneas con un MDP, es la consideración sobre lo que el robot o agente debe realizar una vez alcanzado el estado meta. Generalmente, la política obtenida del MDP tiene una acción definida cuando se alcance el estado meta, pero dicha acción llega a ser innecesaria debido a que si el robot se encuentra en un

estado meta, considera que ya ha logrado el objetivo del MDP y no ejecuta ninguna otra acción lo aleje de ese estado absorbente, como se observa en la figura 4.14. Específicamente en aplicaciones de navegación robótica llega a ser suficiente debido a que el objetivo es que el robot aprenda la mejor manera de llegar a un punto específico en un ambiente conocido, partiendo de cualquier punto de dicho ambiente. Es por eso que en el modelo anterior, DCAD, se incluyó una acción la cual era no hacer nada, con el fin de que una vez alcanzada la meta se terminara de ejecutar el MDP correspondiente.

Ahora bien, en problemas de locomoción de robots como el que estamos atacando, el objetivo es que el robot aprenda la manera de avanzar hacia adelante y no se caiga. Pero, el tener una meta absorbente por MDP para la locomoción de un robot polípedo llega hacer insuficiente, dado que se necesitará alcanzar y alternar más de un estado para generar un paso estable hacia adelante. Esto es fundamental si consideramos que únicamente un HMDP será el encargado de encontrar dicho paso, sin la intervención de un coordinador, como ocurre en el modelo DCAD.

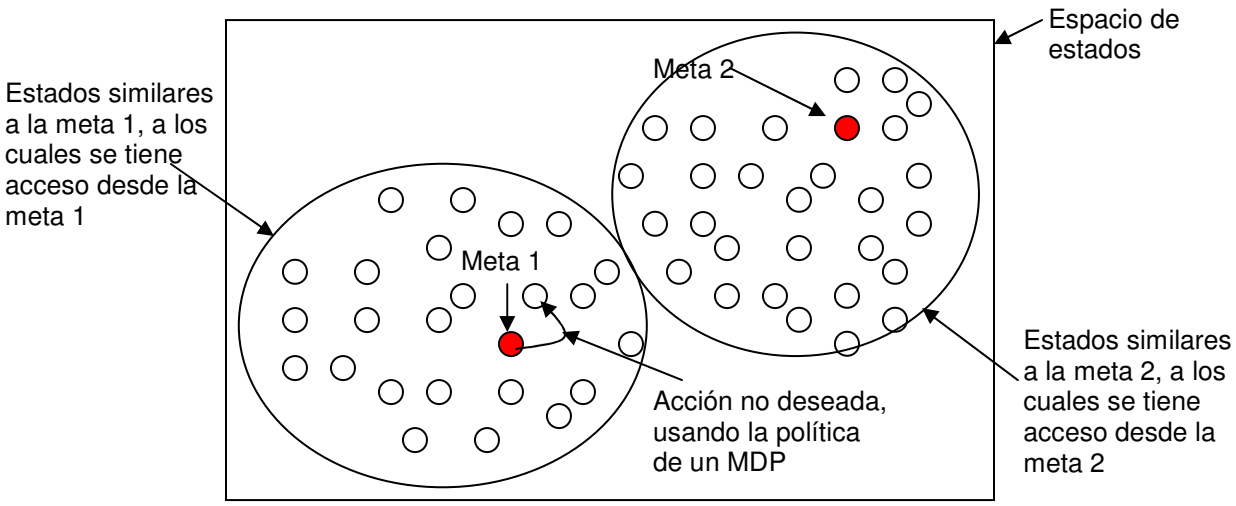


Figura 4.14 Esquema de funcionamiento de un MDP donde se ilustra la dificultad de alcanzar una segunda meta y el resultado de ejecutar una acción estando en una meta

Una posible solución para el primer problema es considerar una acción en la que el robot no haga nada, como en el primer modelo. Esto implicaría que una vez alcanzado el estado meta ya no habría manera de salir de él, a menos que existiera algún coordinador en una capa superior, el cual ejecutaría algún otro MDP, lo cual no es deseado para este modelo. De modo que hemos propuesto un criterio de llamada entre MDPs, el cual consiste en que una vez que un MDP, *MDP1,* haya alcanzado su meta, es decir, cuando se encuentre en el estado meta, en lugar de ejecutar la acción dada por la política envíe una señal a otro MDP, *MDP2*. Dicha señal le indicará al *MDP2* que debe ejecutarse. El *MDP2* tendrá el mismo objetivo, hacer que el robot aprenda a avanzar hacia adelante de manera estable, pero usando alguna otra configuración distinta a la del *MDP1*, que le ayude a conseguirlo. Así, el *MDP2* tendrá el mismo espacio de estados y de acciones pero con una meta diferente, es decir, una matriz de recompensa diferente. Gráficamente podríamos ver al MDP_C como se muestra en la figura 5.15.

La política obtenida después de resolver el MDP MDP_C, tendrá acciones que guiarán al par de patas a alcanzar una meta-configuración. Aunado a esto se presentan dos problemas, primero, dicha política tendrá, como se ha discutido, una acción a ejecutar cuando el robot se encuentre en el estado meta, la cual sacará momentáneamente del estado meta al robot llevándolo a algún otro estado. A pesar de que regresaremos nuevamente al estado meta, dado que la política es aplicada continuamente, la acción tomada en el estado meta puede generar un movimiento no deseado pues lo que se quiere es que una vez en el estado meta el robot permanezca ahí. El segundo problema se presenta cuando al tener sólo una meta a alcanzar, se limitan los movimientos del robot, haciendo que éste sólo use configuraciones similares a la meta-configuración, esto es, se limita a encontrar configuraciones lo más cercanas posibles a la meta-configuración, sin

explorar el posible uso de configuraciones distintas a la meta-configuración. Ver la figura 4.14.

Por último, para terminar de definir el *MDP_C* habrá que definir las metas y modo de interactuar entre los MDPs *MDP1* y *MDP2*.

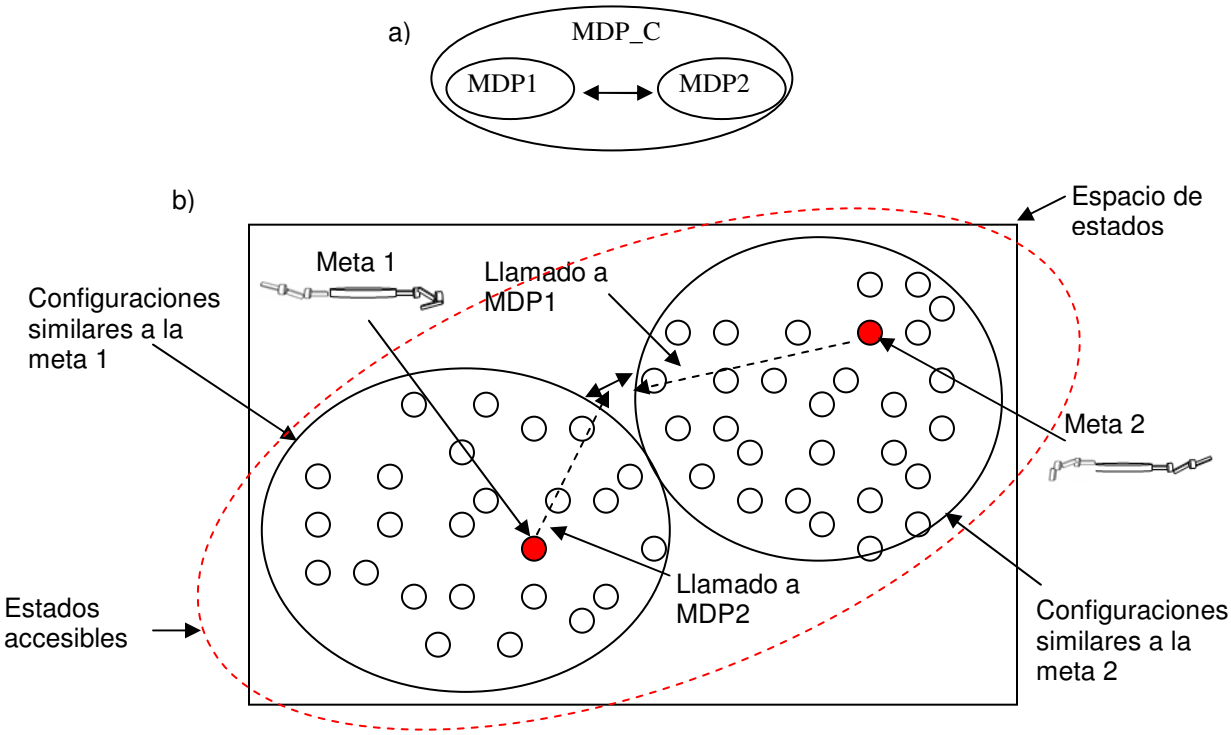


Figura 4.15 a) Estructura del MDP_C, b) esquema de funcionamiento del MDP_C donde se ilustra cómo pueden ser alcanzadas dos metas realizando llamados entre dos MDPs cada uno con una meta distinta

Como deseamos que la búsqueda de configuraciones para obtener un paso estable se realice en todo el espacio de configuraciones, la meta del MDP *MDP1* será el estado 9 ver figura 4.15. Esta meta permitirá explorar en las primeras configuraciones del espacio de estados, dado que las configuraciones más similares a esta configuración se encuentran en las primeras configuraciones del espacio de estados y como se discutió previamente, la búsqueda se realiza entre las configuraciones más similares

a la meta-configuración. Por las mismas razones, la meta del MDP *MDP2* será el estado 55 ver figura 4.15 la cual permitirá explorar en las últimas configuraciones del espacio de estados. Se han elegido estas configuraciones debido a que ambas contribuirán a mantener el cuerpo del robot desplegado de la superficie de apoyo y permitirán realizar un avance hacia adelante. Las matrices de recompensa de ambos MDPs, en las cuales se ven reflejadas dichas metas, se muestran en el apéndice C.

4.3.2.3 Definición MDP_M

Al igual que con el MDP_C, se utiliza la misma estrategia, uso de metas implícita y explícita, uso de dos MDPs, *MDP1* y *MDP2* para la definición del *MDP_M*. La estructura del *MDP_M* se muestra en la figura 4.16.

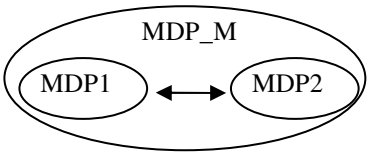


Figura 4.16 Estructura del MDP_M

Ahora bien, como se había mencionado el par de patas centrales se enfoca esencialmente en el control del movimiento del segmento 1 sin importarle cómo se mueven los segmento 2 y 3 (véase figura 4.3). Al MDP_M únicamente le interesa en qué posición está el segmento 1 y si la pata está posada o levantada de la superficie de apoyo. Así, el espacio de estados para este MDP comprende todas las posibles configuraciones en las que sólo importan las consideraciones anteriores. Así, tendremos un total de (3x2)x(3x2) = 36 estados, los cuales son ilustrados en la figura 4.17.

El conjunto de acciones, al igual que para MDP_C, estará compuesto por conjuntos de cuatro acciones básicas. En este caso las acciones básicas son: mover hacia adelante o atrás el segmento 1 de alguna de las patas

involucradas en el par, y levantar o bajar alguna de las patas. Para llevar a cabo las acciones levantar o bajar utilizamos respectivamente los MDPs MDP_UP y MDP_DW1 con el fin de lograr tener un mejor control de ambas patas. El conjunto de acciones se muestra en la tabla 4.4.

Dado que se tienen 36 estados y 16 acciones, la función de transición será una matriz de 36x36x16 (dicha matriz no es mostrada en esta tesis debido a sus dimensiones).

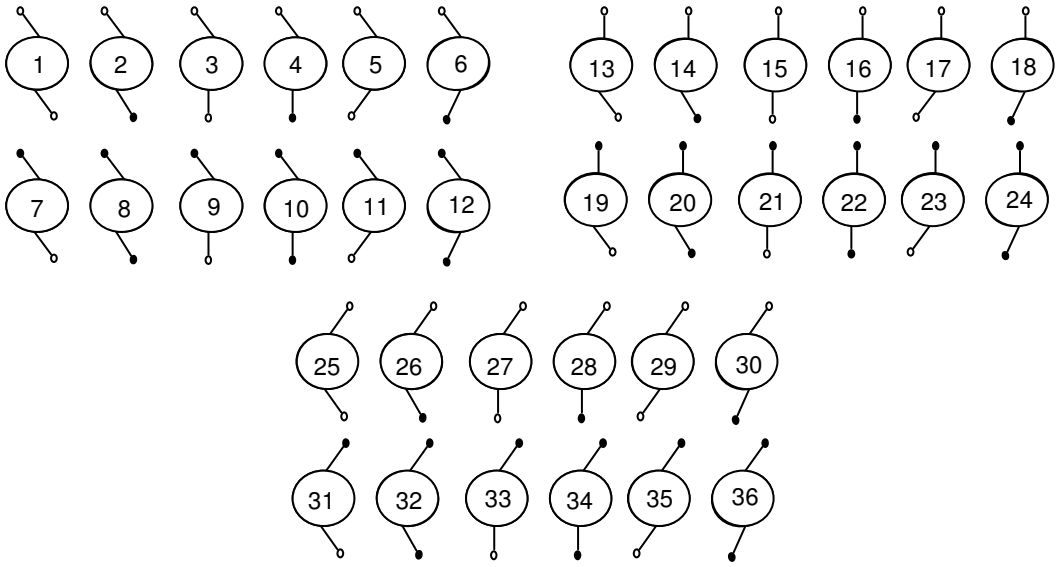


Figura 4.17 Espacio de estados. Los círculos negros indican que la pata está posada y los blancos indican que la pata está levantada de la superficie de apoyo. Por claridad, el robot se muestra desde una vista panorámica

Finalmente, la meta para el *MDP1* y *MDP2* de *MDP_M* son los estados 2 y 35 respectivamente, ver figura 4.17, para que al igual que en el MDP *MDP_C* se realice una exploración completa de todas las configuraciones del espacio de estados. Las matrices de recompensa de *MDP1* y *MDP2* se muestran el apéndice D.

Como se mencionó, el MDP *MDP_C* controla los pares de patas (1,6) y (3,4), de esta manera el HMDP de este modelo es visualizado gráficamente como se muestra en la figura 4.18.

Tabla 4.4 Conjunto de acciones, las flechas con dirección vertical indican si una pata es levanta o apoyada y las flechas con dirección horizontal indican el sentido del movimiento del segmento 1

No. de Acción	Conjunto de acciones básicas			
	Pata 1		Pata 2	
	Mov. Vertical	Mov. Horizontal	Mov. Vertical	Mov. Horizontal
1	↑	→	↑	→
2	↑	→	↑	←
3	↑	→	↓	→
4	↑	→	↓	←
5	↑	←	↑	→
6	↑	←	↑	←
7	↑	←	↓	→
8	↑	←	↓	←
9	↓	→	↑	→
10	↓	→	↑	←
11	↓	→	↓	→
12	↓	→	↓	←
13	↓	←	↑	→
14	↓	←	↑	←
15	↓	←	↓	→
16	↓	←	↓	←

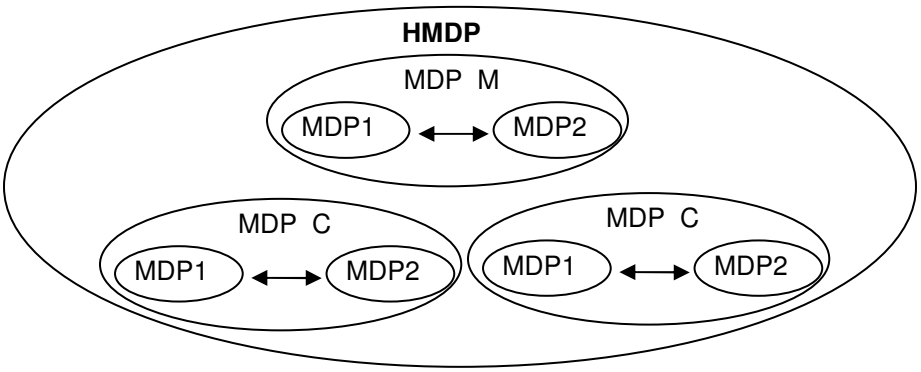


Figura 4.18 Estructura del HMDP, modelo DDAD

Nótese que los MDPs que conforman a este HMDP no tienen comunicación alguna ni dependen para su ejecución de señales enviadas entre ellos. Las señales enviadas entre MDPs ocurren estrictamente al interior de MDP_M y MDP_C, por ello hemos catalogado como *HMDP descentralizado* al HMDP de este modelo.

4.3 Discusión

En este capítulo se han expuesto dos modelos usando HMDPs para el control de un robot hexápodo, cabe mencionar que no se llegó a estos modelos de manera automática, sino que existieron modelos intermedios los cuales no se describen aquí, que fueron la base para la creación de estos modelos finales: DCAD y DDAD. Un MDP jerárquico puede verse como un MDP centralizado, por su parte un MDP DDAD puede verse como un MDP descentralizado. Cabe mencionar que para algunos autores del área de inteligencia artificial distribuida (Durfe, 1999) y robótica (Cao et al., 1997) los modelos DCAD y DDAD se conocen respectivamente como centralizado o jerárquico y distribuido. Recordemos que las aseveraciones sobre MDPs son aplicables a aquellos MDPs que son usados para resolver problemas de locomoción.

Se han descrito conceptos necesarios para el entendimiento de las diferencias que existen en los HMDPs usados para controlar robots con patas y para controlar robots con un arreglo diferencial de llantas, y aún más, las diferencias que existen en aplicar HMDPs para solucionar problemas de navegación robótica con respecto a problemas de locomoción de robots con patas. Para ello, hemos propuesto e implantado cuatro conceptos: meta simultánea, meta exclusiva, meta simultánea implícita y meta simultánea explícita.

En este último modelo, modelo DDAD, se ha descrito la manera en la que se obtuvo dicho modelo desde un punto de vista roboticista. Esta manera de dividir el problema consideramos constituye el inicio de una discusión de mayor importancia, la cual rebasa el ámbito de esta tesis, la definición de *principios de organización de MDPs*. Esto es, estamos proponiendo la generación de MDPs multi-objetivo vistos como un solo MDP, el cual está constituido de varios sub MDPs, donde el MDP principal o TopMDP puede tener múltiples objetivos o metas simultáneas y los sub MDPs que lo conforman únicamente pueden tener un objetivo o meta exclusiva.

Afirmamos que nuestra propuesta puede contribuir a establecer principios de organización de MDPs pues, con las debidas extensiones, nuestro modelo DDAD podría servir para atacar problemas de dos o más objetivos, para el mismo u otro de los muchos problemas en robótica de robots polípedos. No sería necesario redefinir nuestro MDP DDAD, si por ejemplo, deseáramos que nuestro robot hexápodo además de caminar pudiera brincar, pues al esquema actual se le incorporaría una nueva capa al HMDP la cual tendría por objetivo hacer que el robot pudiese brincar.

Este tipo de trabajos, que se alejan de la inevitable pero no siempre capitalizable y reproducible experimentación robótica, son necesarios en el área y deben converger hacia el establecimiento de principios o esquemas de organización más o menos genéricos, que sean aplicables a familias de problemas y no sólo a casos específicos de estudio. Hasta el momento podemos comentar una de las principales diferencias entre el modelo DCAD y el modelo DDAD, la cual es el numero de estados y de acciones por MDP, esta diferencia es debida a la manera en la que son definidos ambos modelos, para el modelo DDAD se ha realizado un análisis más a detalle de cómo alcanzar la meta global usando MDPs que trabajen de manera independiente. Más ventajas y desventajas de ambos modelos serán discutidas después de las pruebas realizadas a cada uno de ellos.

CAPÍTULO I: Introducción

1.1 Motivación y problemática

Los sistemas robóticos modernos son cada vez más poderosos en términos del número de sensores utilizados y de movilidad. El progreso tecnológico actual hace posible que diversos tipos de robots realicen tareas en que apoyen o sustituyan a humanos. Robots con poca o nula autonomía han ganado terreno y son utilizados en la realización de tareas en ambientes controlados, por ejemplo en fábricas e industrias. Por su parte, robots móviles y autónomos realizan cada vez más tareas por nosotros en entornos complejos, como los que se encuentran frecuentemente en la exploración de zonas accidentadas o colapsadas y en la robótica de servicio.

Sin embargo, aún cuando hay muchas aplicaciones que se pueden solucionar utilizando robots adecuadamente diseñados y programados, equipados con las herramientas necesarias y con la destreza de un operador, persisten tareas que no pueden ser resueltas por robots debido a la dificultad de acceso a la zona en donde debe resolverse el problema. Estas dificultades conciernen, por ejemplo, pendientes, obstáculos a sobrepasar o evitar, terrenos irregulares, etc.

Estas dificultades y la forma de enfrentarlas son estudiadas en el área de robótica, en la cual se investiga la forma de diseñar sistemas de locomoción en un nivel bajo, y de localización y planeación en un alto nivel. Hay trabajos en los que se muestran soluciones interesantes a este problema, por ejemplo el uso de manipuladores remotos (Armada et al. 2005). Algunas otras soluciones, no menos interesantes, consisten en proveer de un sistema de transporte a los manipuladores tele-operados, resultando en vehículos con

ruedas o vehículos con orugas, y recientemente robots polípedos o caminantes.

Así, los roboticistas se han interesado en el diseño y control de robots polípedos, robots con patas, para solucionar la problemática de acceso y movilidad en ambientes peligrosos o desconocidos. En particular, los robots hexápodos autónomos pueden ser útiles en situaciones donde el ambiente es hostil, e. g. en la exploración de volcanes (Bares 1999), en las profundidades del océano, en zonas de desastre y de cuerpos extraterrestres; al igual que en tareas militares tales como la búsqueda de puntos de interés en el campo de batalla o la búsqueda de minas explosivas.

Un reto importante en el diseño de robots polípedos es lograr la autonomía completa de las capacidades del robot para adaptarse a los cambios del ambiente y al aumento de sus propias capacidades con la incorporación de nuevos sensores. Esto motiva a los diseñadores a hacer los sistemas de control de robots tan flexibles como sea posible utilizando algoritmos de aprendizaje automático (c. f. sección 3.4). Además de ayudar a desarrollar robots autónomos y flexibles, el aprendizaje reduce la ingeniería humana requerida para controlar robots en general.

1.1.1 Estabilidad estática y dinámica

En robótica existen dos formas básicas de usar los efectores de un robot, la primera es usar efectores para mover al robot mismo y la segunda forma es usar efectores para mover otros objetos. El primer caso se estudia en el área de locomoción, mientras el segundo caso en el área de manipulación. De esta forma se divide el estudio en robótica en dos grandes categorías, robótica móvil y robótica de manipuladores [URL_Historia].

Muchos tipos de efectores y actuadores* pueden ser usados para mover un robot, entre las categorías más conocidas que podemos encontrar tenemos: patas para caminar, reptar, subir, brincar; llantas para rodar; brazos para pivotear, reptar, subir; aletas para nadar; alas para volar o planear.

Mientras que la mayoría de los animales terrestres utilizan patas para desplazarse, en robótica la locomoción usando patas es un problema muy difícil de resolver, especialmente cuando se compara con la locomoción con ruedas. Esta dificultad estriba en que todo robot móvil necesita ser estable, es decir, no debe desequilibrarse ni caerse fácilmente. Existen dos clases de estabilidad, *estabilidad estática* y *estabilidad dinámica*. La estabilidad estática es la capacidad que tiene un objeto de volver a su posición original. La estabilidad dinámica se refiere a cuánto tiempo se tarda un objeto en regresar a su posición original [URL_Estabilidad].

Un robot estáticamente estable puede permanecer erguido sin caerse cuando no se encuentra en movimiento. Ésta es una característica útil y deseable, pero difícil de alcanzar, debido a que se requiere que exista un cierto número de patas o ruedas en el robot para proporcionar suficientes puntos de soporte. Las personas no son estáticamente estables, este equilibrio es en gran parte inconsciente y es aprendido durante los primeros años de vida y se logra gracias a los nervios, músculos y tendones del cuerpo humano.

Podría pensarse que con más patas, la estabilidad estática puede alcanzarse de manera más simple. Lo anterior no es totalmente cierto, debido a que para que un robot se encuentre estable, el centro de gravedad (CDG) del robot debe ser soportado por un polígono. Este *polígono* o *geometría de apoyo*, es

*La palabra actuadores se usará en esta tesis como sinónimo de activadores o accionadores. Los tres términos son usados indistintamente como traducción del término en inglés *actuators*.

básicamente la unión de todos los puntos de soporte sobre la superficie en que los que el robot se apoya. Así, en un robot de dos patas, el polígono es realmente una línea, y como se mencionará más adelante en esta misma sección, este tipo de robots no es estáticamente estable. Ahora bien, si consideramos robots con tres o múltiplos de tres patas, sus patas se organizan de forma alternada en una configuración de trípode y su cuerpo queda distribuido sobre tal configuración. Tales robots producen un polígono estable de soporte. Así, estos robots son estáticamente estables.

Ahora bien, una vez que el robot es estáticamente estable el problema no está totalmente resuelto. Las preguntas que habrá que responder a continuación son: ¿qué sucede cuando un robot estáticamente estable levanta una pata e intenta moverse?, ¿el centro de gravedad del robot permanece dentro del polígono de soporte antes mencionado?. Para responder a la primera cuestión se debe realizar un análisis más profundo, debido que este aspecto está completamente ligado al problema de locomoción en robótica, la última pregunta puede responderse considerando si el centro de gravedad permanece o no dentro de ese polígono dependiendo de la geometría del robot. Ciertas *geometrías de apoyo* robóticas hacen posible que un robot polípedo permanezca siempre estáticamente estable, e. g. triángulos, rectángulos, hexágonos y en general polígonos.

Una aspecto importante a considerar para que un robot mantenga una posición estáticamente estable, es que el peso de una pata es insignificante comparado con el peso del cuerpo, e. g. 10% del peso total del cuerpo, de modo que el CDG del robot no será afectado por la oscilación de la pata. De acuerdo con esta suposición, el paso estático convencional es diseñado para mantener el CDG del robot dentro del polígono de soporte. La figura 1.1 ilustra la posición del CDG en el caminar de un robot de cuatro patas.

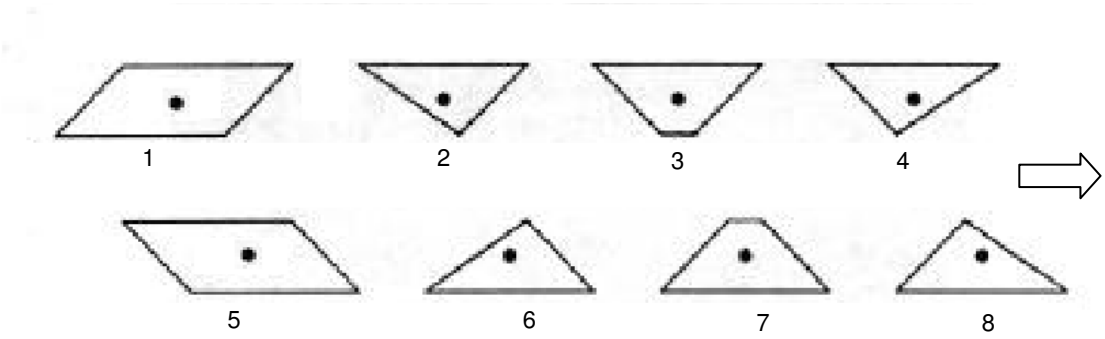


Figura 1.1 Posición del CGD durante el movimiento de un robot con cuatro patas

Una alternativa a la estabilidad estática es la estabilidad dinámica que permite que un robot o animal sea estable mientras se mueve. Por ejemplo, los robots brincadores con una sola pata son dinámicamente estables, i. e. saltan en un mismo lugar o en distintas posiciones para no caerse, pero no pueden parar y permanecer de pie. Esto es el problema de balance del péndulo invertido.

Para determinar cómo lograr que un robot polípedo sea estable al desplazarse, es importante preguntarse cuántas patas se encuentran en el aire durante el movimiento del robot, es decir, mientras genera el paso. Seis patas es un número conocido que permite una locomoción estable, si las mismas tres patas no adyacentes se mueven a la vez, dicho paso se conoce como el paso de trípode alternado, si las patas varían, se llama paso ondulatorio, dichos pasos se ilustran en la figura 1.2.

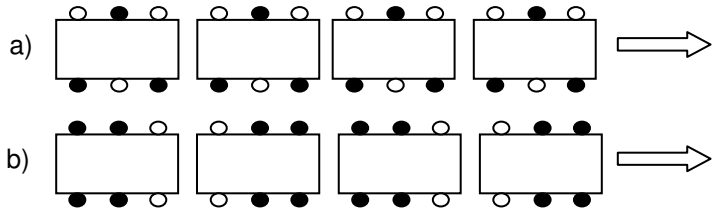


Figura 1.2 a) Paso de trípode alternado, b) paso ondulatorio, donde los círculos negros indican que la pata se encuentra posada en la superficie de apoyo y los círculos blancos indican que la pata se encuentra despegada de la superficie de apoyo

Un robot con seis patas puede levantar tres patas no adyacentes a la vez para moverse hacia adelante, y todavía conservará una estabilidad estática. Este tipo de robot puede lograr un movimiento dinámicamente estable debido a que utiliza un paso en forma de trípode alternado, el cual es un patrón común en la naturaleza para insectos con seis o más patas. Las tres patas fijas en tierra mantienen al robot estáticamente estable.

De manera general, en la naturaleza encontramos animales que tienen un sistema de locomoción usando seis patas con un paso en forma de trípode, pero estos animales son ligeros y pequeños, por ejemplo, las cucarachas mueven sus patas en forma de trípode y pueden hacerlo muy rápidamente. Los insectos con más de seis patas, e. g. los ciempiés y los milpiés, utilizan el paso ondulatorio. Sin embargo, cuando estos insectos se mueven rápidamente, cambian dicho paso para llegar a ser prácticamente aerotransportados, de esta forma durante breves períodos de tiempo no se encuentran estáticamente estables.

Los robots insecto biológicamente inspirados como los hexápodos son potencialmente muy estables dado que el movimiento de una pata no afecta generalmente la postura del robot, además de que con un algoritmo de control adecuado pueden contender con una amplia cantidad de superficies.

Para sintetizar:

- La estabilidad dinámica requiere un control más complejo que los controles usados para mantener una estabilidad estática.
- La estabilidad de un robot polípedo no es un problema trivial de resolver.

1.1.2 Locomoción

El desplazamiento o locomoción de un robot polípedo es el resultado de un movimiento coordinado de sus patas. Este movimiento está definido por cierto paso que refleja determinadas especificaciones tales como velocidad, dirección, etc.

Hay dos fases claramente distinguibles que contribuyen al movimiento de cada pata del robot: la *fase de soporte* y la *fase de transferencia*. Durante la *fase de soporte* o *apoyo*, cada pata debe ser capaz de ejercer cierta fuerza sobre la superficie en la que se encuentra el robot, de forma ordenada para proporcionar la fuerza necesaria al cuerpo del robot para así permitirle moverse según una trayectoria predeterminada. Después, durante la *fase de transferencia* o *transición*, la pata debe desplazarse de forma ordenada hacia el siguiente punto de soporte para reiniciar la secuencia de movimiento. Dichas fases se ilustran en la figura 1.3. Cada fase necesita un conjunto de requerimientos para la operación de la pata. Durante la fase de soporte, la pata debe ser capaz de mantenerse en equilibrio, mientras que en la fase de la transición el requisito principal es la velocidad de desplazamiento.

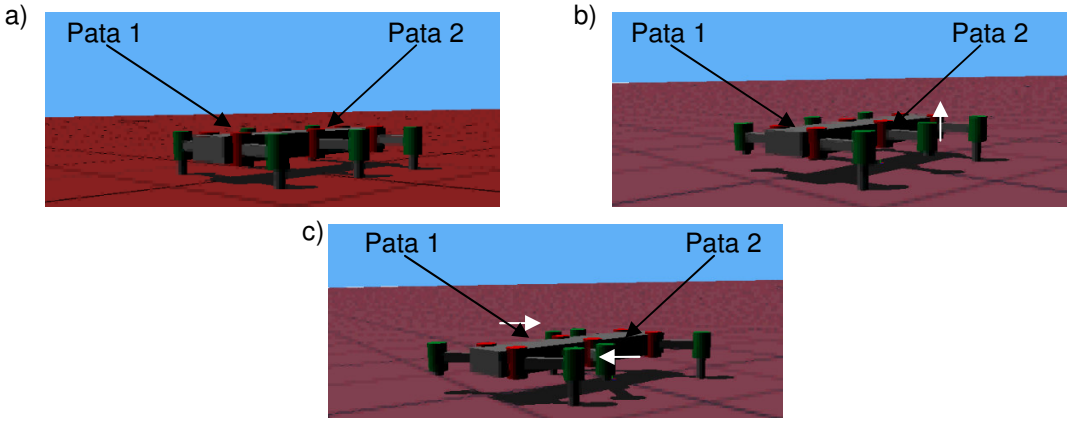


Figura 1.3 Secuencia de movimientos en donde la pata 2 ilustra la fase de transición y la pata 1 ilustra la fase de transferencia. a) posición inicial, b) levantamiento de la pata 2 y c) movimientos de cada pata

Una vez definida una tarea para el robot, la trayectoria se establece y debe ser seguida por el robot en su espacio de trabajo. El paso del robot definirá las transiciones de estado para cada pata. Sin embargo, la trayectoria de la pata durante la fase de soporte es determinada por la trayectoria del cuerpo del robot. Así, existe un número infinito de trayectorias que pueden ser utilizadas para obtener el movimiento deseado del robot. La definición de la trayectoria se basa únicamente en la movilidad de las patas y es funcional para un robot que camina en un terreno regular, debido a que no existen obstáculos en el movimiento de sus patas. Esta manera de definir la trayectoria de un robot no es aplicable a un robot que camina en terrenos irregulares, e. g. superficie con escalones o pendientes.

Para los robots que se desempeñan en ambientes irregulares, existe una gran incertidumbre en el resultado de la acción de cada pata. Así, es razonable seleccionar la trayectoria de cada pata y revisar continuamente el resultado de la acción de cada pata. Esto sugiere que además de todos los estados posibles considerados que representan las posiciones de cada pata, se consideren tantas variables como posibles lecturas de sensores que indican el resultado de cada acción de las patas. A mayor número de variables a controlar se incrementa la complejidad del sistema y en específico el control de cada pata.

Por ejemplo, supongamos que se tiene un robot hexápodo, cuyas patas poseen 2 grados de libertad o articulaciones y que cada articulación está controlada por un servomotor. Suponiendo que dicho servomotor tiene una precisión de paso de 2°, cada articulación tendría 180 posibles posiciones, dado que son 2 articulaciones se tendría *180x180 = 32400* posibles posiciones o configuraciones en las que puede estar una pata del robot. Como se tienen en total seis patas, entonces se tendría un total de $32{,}400^6$ posibles configuraciones de las patas del robot. Como se mencionó,

deseamos tener información acerca del resultado de cada acción de cada pata. Supongamos, para este ejemplo, que lo único que nos interesa es saber si la pata está posada o no en la superficie de apoyo, esta información puede obtenerse usando un sensor de contacto por cada pata. Así, el número anterior de configuraciones aumentaría considerando las lecturas de los sensores de contacto, con lo cual tendríamos $32{,}400^6 \times 2^6$ posibles estados del sistema.

Otro problema aunado a lo antes descrito es la optimización de trayectoria. Generalmente, se conocen los puntos inicial y final de la trayectoria del robot, y el problema es determinar la trayectoria óptima, de acuerdo a cierto criterio, que una ambos puntos. En este trabajo la trayectoria es conocida y es una línea recta, lo que debe ser decidido entonces es: (i) la localización del robot en un ambiente no estructurado, i. e. una superficie irregular, con pendientes, baches y obstáculos, y (ii) las posiciones óptimas de cada pata para realizar la trayectoria de forma estable.

1.1.3 Control

La generación de pasos estable es una parte integral del control de robots polípedos. Los pasos para hexápodos requieren la coordinación simultánea de los movimientos de las seis patas en un ciclo de activaciones. Además, el movimiento hacia delante sostenido requiere que la secuencia de acciones individuales de las patas que impulsan al robot sea repetida continuamente y que cada estado del robot, en el ciclo de movimiento, pueda ser fácilmente accesible. El problema se complica debido a las diferencias entre los robots hexápodos, e. g. morfología, altura, etc., así como las diferentes capacidades de cada pata. Hay patas con dos, tres o más grados de libertad, además de sensores adicionales para medir por ejemplo, posición de cada articulación y contacto con la superficie, entre otros.

Por ello, es importante desarrollar algoritmos de control que permitan responder de manera adecuada a las situaciones en las que llegue a encontrarse un robot, y más aún, que dicho algoritmo permita que el robot sea lo más autónomo posible, esto es, que sea capaz de responder ante situaciones tanto comunes como inesperadas con su propias capacidades y medios, sin la intervención del diseñador.

El control de robots hexápodos es un problema abierto en robótica. Se han propuesto algoritmos de control para robots hexápodos, los cuales han ido evolucionando y mejorándose con el tiempo. Existen varias propuestas de control las cuales se estudiarán en el capítulo II, pero todos tienen ciertas carencias. Algunos trabajos, por ejemplo (Jindrich et al. 1999), asumen varias hipótesis para simplificar el control del robot, e. g. que las acciones son completadas sin ningún tipo de desviación, lo que hace que el robot sea poco autónomo y robusto. Otros trabajos utilizan enfoques reactivos (Maes & Brooks 1990), lo cual hace que tales robots sólo puedan ser capaces de realizar tareas relativamente sencillas, e. g. avanzar de manera estable. Otros trabajos (Nakada et al 2003) usan redes neuronales, para que un robot polípedo sea más autónomo, aunque la forma en que se obtiene el control no permite realizar un estudio más detallado de las situaciones que existen en el mundo del robot para tomar una determinada decisión de acción. Es necesario entonces buscar alternativas de desarrollo de algoritmos de control que permitan al robot ser autónomo y robusto, en una diversidad de ambientes no estructurados y que además permitan explicar las situaciones por las que un robot se ve obligado a tomar cierta decisión, así como estudiar los patrones de movimiento del robot, su generalidad en el desplazamiento de robots polípedos y su eventual equivalencia con los patrones de movimiento de animales hexápodos.

1.3 Descripción de la tesis

1.2.1 Objetivos

El objetivo general de esta tesis es **diseñar e implementar un esquema de control descentralizado eficiente basado en Procesos de Decisión de Markov** (MDP por sus siglas en inglés, *Markov Decision Process*) **para un robot hexápodo**.

Por eficiente se entiende que el tiempo de respuesta sea del orden de segundos, tanto para el robot simulado como para un prototipo físico operando con tarjetas y procesadores comerciales actuales.

Como objetivos específicos se han establecido los siguientes:

- Diseño de un protocolo de coordinación para un sistema de control basado en descentralizaciones de un MDP, MDP jerárquico y distribuido. No existe trabajo reportado en este sentido.
- Análisis de la forma de caminar de hexápodos con una distribución circular de patas alrededor de su cuerpo.
- Diseño de un modelo de control que permita a un robot hexápodo desplazarse de manera eficiente, i. e. de forma estable y responder de forma adecuada ante situaciones conocidas como inesperadas del ambiente.
- Diseño de un modelo de control de un robot hexápodo más descriptivo de los que existen hasta el momento, i. e. un modelo que permita identificar las condiciones del ambiente que ocasionan que el robot genere una configuración determinada.
- Proposición de principios de organización de MDPs en problemas de locomoción robótica.

En esta tesis se aplicaron Procesos de Decisión de Markov (MDPs) para resolver un problema de locomoción de robots hexápodos, específicamente robots hexápodos con morfología hexagonal o circular.

Se realizaron varios modelos usando MDPs de los cuales finalmente se obtuvieron dos modelos descentralizados principales: un modelo con decisión centralizada y actuación descentralizada (DCAD) y un modelo con decisión y actuación descentralizadas (DDAD). Ambos modelos permiten al robot hexápodo desplazarse de manera estable en ambientes semi-estructurados. Se realizaron diversas pruebas para evaluar su desempeño del robot y para establecer las ventajas y desventajas de cada modelo. Finalmente se valoraron algunos elementos del modelo DDAD en un robot físico.

1.2.3 Organización de la tesis

El resto de la tesis se encuentra dividido en seis capítulos. En el segundo capítulo se describen los fundamentos de en esta tesis. Se hace también un recapitulativo acerca de los diferentes enfoques que existen y que han existido para el diseño de sistemas de control para robots hexápodos, destacando las ventajas y desventajas de cada enfoque, con la finalidad de determinar en dónde se encuentra ubicado nuestro trabajo. En el tercer capítulo se da una descripción formal de MDPs y HMDPs, los diferentes métodos de resolución de estos modelos y las diferencias que existen para el diseño de estos mismos. En los capítulos cuarto y quinto se describen, respectivamente, los modelos propuestos en esta tesis y los resultados obtenidos. Finalmente, en el sexto capítulo se exponen las conclusiones de este trabajo y sus perspectivas. Adicionalmente, en los apéndices se proporcionan aspectos técnicos de las herramientas utilizadas para la realización de esta tesis.

CAPÍTULO II: Historia y Estado del Arte de los Robots Hexápodos

2.1 Conceptos

Existen varias definiciones del término robot, entre las más populares pueden mencionarse las siguientes. Un *robot* se define informalmente como una máquina con capacidad de movimiento y acción, o computadora con "músculos" [URL_Robotica1]. También suele definirse el término *robot* como dispositivo electrónico, generalmente mecánico, que desempeña tareas automáticamente, ya sea bajo supervisión humana directa o a través de un programa predefinido o siguiendo un conjunto de reglas generales, utilizando técnicas de inteligencia artificial. Generalmente estas tareas reemplazan, asemejan o extienden el trabajo humano, como ensamble en líneas de manufactura, manipulación de objetos pesados o peligrosos, trabajo en el espacio, etc. [URL_Robotica2].

Dentro de las diversas clases de robots, nos interesaremos en tres familias: robots móviles, autónomos y polípedos. Un *robot móvil* es aquel capaz de desplazarse completamente en su ambiente de trabajo. Un *robot autónomo* es aquel capaz de tomar decisiones por sí mismo, y está orientado a reducir a lo mínimo la intervención del diseñador, en su operación. Un *robot polípedo* o con patas es aquel que físicamente tiene patas o extremidades, las cuales utiliza para desplazarse en su ambiente de trabajo. A su vez, los robots polípedos pueden ser divididos de acuerdo al *número de extremidades* que tienen, según se describe a continuación.

Robots monópodos. Este tipo de robots sólo cuenta con una extremidad o pata, el ejemplo más representativo es un robot desarrollado por el Instituto

de Robótica y el Departamento de Informática de la universidad americana Carnegie-Mellon. Este dispositivo salta sobre su única extremidad, buscando continuamente la estabilidad dinámica, i. e. con el robot en movimiento, ya que para este robot es difícil alcanzar una estabilidad estática i. e. con el robot inmóvil, debido a que lograr una estabilidad estática es complicado usando menos de cuatro patas. Este tipo de robots requiere en general un gran esfuerzo del diseñador para desarrollar su sistema de control.

Robots bípedos. Este tipo de robots posee dos patas y puede caminar lateralmente, avanzar y retroceder, simulando más o menos el modo de andar humano. Entre los robots bípedos más destacados podemos mencionar el presentado en (Ki et al. 2003).

Robots cuadrúpedos. Este tipo de robots posee cuatro extremidades o patas y es el más representativo de los robots con patas. En el Instituto Tecnológico de Tokyo fue construido un vehículo de cuatro patas, equipado con sensores de contacto y sensores de rotación. Cada pata tiene 3 DOF. El control se realiza desde un microprocesador que asegura la formación de un triángulo de apoyo con tres de sus patas de forma continua. Por otra parte, se han desarrollado nuevos algoritmos para conseguir que este tipo de robots conserve la estabilidad mientras camina, así como algoritmos para la detección de inestabilidades y la planificación de modos de andar.

Robots hexápodos. *Son robots que cuentan con seis patas. En la Universidad Estatal de Ohio se desarrolló un robot con seis patas que permite al operador tomar decisiones estratégicas, independientemente de la posición particular de cada extremidad. Está basado en un sistema de control consistente en 13 procesadores Intel 86/30. En el Instituto de Robótica de la Universidad de Carnegie-Mellon de EEUU se desarrolló el AMBLER, acrónimo de Autonomous MoBiLe Exploratory Robot, un robot hexápodo*

dirigido por programas específicamente diseñados para optimizar el movimiento de avance sobre terrenos abruptos.

Por ser de interés para esta tesis, a continuación se abundará sobre este último tipo de robots.

2.2 Robots hexápodos

Los robots insectos están atrayendo la atención de personas que trabajan en robótica móvil, debido a sus amplias capacidades de locomoción. Los investigadores de robots tipo insecto están explorando la naturaleza como una manera alternativa para crear nuevos robots y modelos de control para esos robots.

Los robots hexápodos autónomos pueden ser útiles en ambientes hostiles, e. g. exploración de volcanes, de las profundidades del océano, del escombro de desastres y de los cuerpos extraterrestres; tareas militares tales como la inspección del campo enemigo y la búsqueda de minas explosivas. Un factor importante para lograr la autonomía completa es la capacidad del robot para adaptarse a los cambios en su ambiente. Esto motiva a los diseñadores para hacer que el sistema de control sea flexible y se adapte tanto como sea posible a los cambios imprevistos del ambiente. Además, la adaptación a través del aprendizaje reduce la intervención humana requerida para que el sistema de control responda a los nuevos cambios.

La generación de pasos que permitan a un robot desplazarse de manera estable es una función integral de un control para todo robot con patas, debido a que estos pasos deben permitir aprovechar correctamente el número de extremidades del robot. Los pasos para hexápodos requieren la coordinación del movimiento simultáneo de sus seis patas. Además, el movimiento hacia adelante requiere que la secuencia de las acciones

individuales que mueven al robot sea repetida continuamente sin perder la estabilidad del robot, considerando que un grupo de extremidades del robot se mueve mientras otro grupo permanece apoyado en la superficie.

Otra fase importante en el desarrollo del control para un robot es evaluar que dicho control permita la estabilidad y el desplazamiento de un robot real. Dado que las simulaciones por sí solas no pueden predecir totalmente el comportamiento de robots físicos reales, debido a que, por definición, las simulaciones son aproximaciones de sus contrapartes reales.

A continuación se hace una revisión de los diferentes enfoques para diseñar e implementar un sistema de control eficiente para robots con patas, en particular, para robots hexápodos.

2.3 Tipos de robots hexápodos

Existen 2 tipos de morfología para robots hexápodos. La más estudiada, como se verá a lo largo de este capítulo, para desarrollo de sistemas de control eficientes para robots hexápodos es la morfología la cual hemos denominado *morfología rectangular* y la cual es presentada en la figura 2.1.

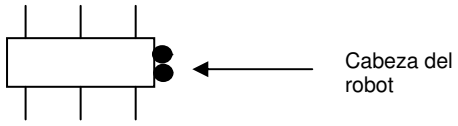
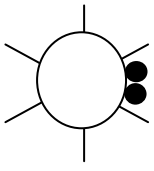


Figura 2.1. Morfología rectangular

En este trabajo, la morfología del robot hexápodo en la que se probará el sistema de control es diferente, le hemos denominado como *morfología circular* y es mostrada en la figura 2.2.



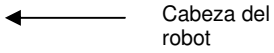


Figura 2.2. Morfología circular

Al hablar de morfología rectangular o circular, nos referimos a la forma del cuerpo del robot hexápodo y la distribución de las patas alrededor de éste. Además, cabe mencionar que cada pata tiene el mismo número de DOF en ambas morfologías.

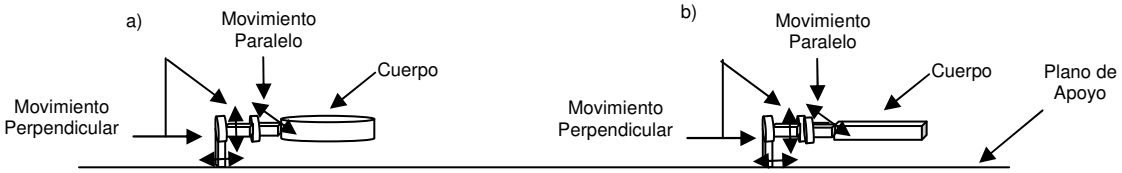


Figura 2.3. Vista lateral de una pata con 3 DOF, los movimientos perpendicular y paralelo son relativos al plano de apoyo. a) Morfología Circular b) Morfología Rectangular

De acuerdo a su morfología, un robot hexápodo tendrá diferentes capacidades de movimiento. Por ejemplo, con una morfología rectangular, el hacer que un robot cambie de dirección es un proceso que requiere un análisis puntual y quizás la construcción de un sistema de control específico. Además, hacer que el robot cambie de dirección de movimiento sin que éste gire total o parcialmente su cuerpo no es un problema sencillo. Esta situación limita la capacidad de este tipo de robots, debido a que si estos robots se encuentran inmersos en ambientes los cuales tienen espacios muy reducidos, puede ser imposible o al menos muy difícil para uno de estos robots salir de una determinada situación, como se muestra en la figura 2.4.

En el escenario anterior es complicado que el robot pase por el pasillo, sin que tenga que mover todo su cuerpo. Este problema también se presenta en robots con llantas, pero en este tipo de robots el problema es resuelto

usando llantas omnidireccionales, las cuales permiten al robot cambiar de dirección de movimiento sin necesidad de cambiar la orientación de su cuerpo. Lo que se ha empezado a hacer para resolver la problemática anterior, para robots con patas, es crear morfologías físicas las cuales permitan holomonicidad, i. e. la capacidad de movimiento en cualquier dirección sin necesidad de girar el cuerpo.

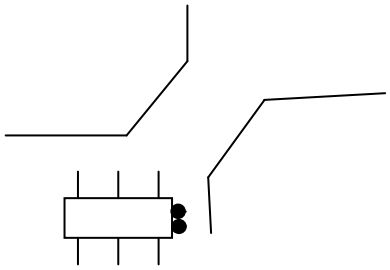


Figura 2.4. Posible situación que requiere diversas maniobras para ser superada por un robot hexápodo con morfología rectangular, e. g. retroceder y girar varias veces

Un robot hexápodo con una morfología circular en la misma situación que el robot anterior, podría cambiar la dirección de su desplazamiento sin necesidad de cambiar la orientación de su cuerpo, esto es, podría desenvolverse en ambientes en los cuales un robot hexápodo con morfología cuadrada no podría, como lo muestra la figura 2.5.

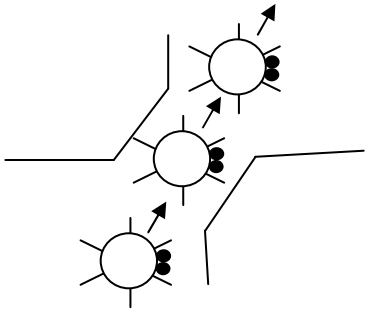


Figura 2.5. Posible situación que solamente requiere maniobras de desplazamiento lateral para ser solucionado por un robot hexápodo con morfología circular

Ahora bien, a pesar de que la morfología circular proporciona ventajas sobre la morfología rectangular, hasta el momento no se han reportado trabajos

relacionados con el desarrollo de sistemas de control eficientes para robots hexápodos con este tipo de morfología, así que uno de los objetivos y una contribución de esta tesis es el proponer e implantar un control eficiente para robots con este tipo de morfología el cual le permita aprovechar las características de la morfología circular.

Con aprovechar las características de dicha morfología, nos referimos a usar de manera adecuada la configuración de las patas alrededor del robot para realizar cambios de dirección de movimiento de una manera que no implique realizar varias maniobras, sino únicamente cambiar la funcionalidad de las patas. Esto será posible ya que teniendo una morfología circular, cada pata se moverá de alguna manera específica al desplazarse en una determinada dirección, cada movimiento de cada una de las patas tratará de conservar la distribución de éstas alrededor del cuerpo del robot. De esta manera, si en algún instante determinado el robot necesitara cambiar de dirección de desplazamiento, será suficiente cambiar la funcionalidad de las patas del robot. Esta redundancia o capacidad de intercambiar el funcionamiento de las patas no es posible de realizar en robots con morfología rectangular, debido a que la manera en la que se encuentran distribuidas las patas alrededor del robot no guarda la misma simetría de los robots con morfología circular.

La morfología rectangular ha sido ampliamente estudiada, probablemente debido a que los primeros modelos para crear robots hexápodos se inspiraron principalmente en insectos como la cucaracha, el palillo, los mosquitos y algunas arañas.

Los sistemas de control que se presentan en los trabajos revisados en esta tesis llegan a ser insuficientes e inadecuados para robots hexápodos con morfología circular. Esto se debe a que las configuraciones que le permiten

desplazarse a un robot con una morfología rectangular son especialmente aplicables a ese tipo de robots, pero no funcionales para robots con morfología circular. Con este trabajo se dará una propuesta para el diseño de sistemas de control para robots hexápodos con morfología circular.

Diseñar un sistema de control eficiente tanto para robots con morfología rectangular como para aquellos con morfología circular tiene cierto grado de dificultad. Con respecto al control de robots hexápodos con morfología circular, difiere y se vuelve más complicado que el control que normalmente se utiliza para robots con morfología rectangular. El trípode, movimiento en conjuntos de tres patas, permite que los robots hexápodos con morfología rectangular puedan desplazarse de una manera estable pues se mantiene el centro de gravedad del robot sin mucha alteración. Cabe señalar que dicho movimiento en trípode ha sido ampliamente estudiado e implementado. En contraste, para controlar robots hexápodos con morfología circular, la sincronización y funcionalidad de cada una de las patas es diferente debido a que se busca que el paso que le permita al robot desplazarse conserve la distribución de las patas alrededor del robot para explotar dicha característica como se discute más adelante. Así, el uso de trípode en los robots hexápodos de morfología circular puede llegar a ser no funcional, por ejemplo los movimientos y configuraciones de patas que son utilizados para hacer avanzar un robot con morfología rectangular no tendrán el mismo resultado al aplicarlos a un robot con morfología circular, debido a la manera en la que se encuentran distribuidas las pastas en ambos tipos de robots.

2.4 Tipos de control para robots hexápodos

Dado que estamos interesados en el diseño del sistema de control para un robot hexápodo es necesario hacer una revisión acerca de los trabajos relacionados con el diseño de control para locomoción de este tipo de robots, para así ubicar nuestro trabajo y su contribución al estado del arte.

Al paso del tiempo han existido diversas maneras en las que se ha diseñado el control para robots hexápodos, entre los enfoques más conocidos podemos mencionar los siguientes:

- Control manual
- Control orientado a metas y objetivos
- Control biológicamente inspirado
- Control difuso
- Arquitectura subsumción
- Control usando aprendizaje por refuerzo
- Control basado en Procesos de Decisión de Markov (MDPs)

A continuación se resumen las características y se revisan trabajos representativos de cada uno de los enfoques anteriores.

2.4.1 Control manual

Las primeras máquinas con patas desarrolladas eran de dimensiones y peso considerables, básicamente adaptaciones de camiones de carga, las cuales eran operadas manualmente por una persona, por eso era prácticamente inalcanzable dotar de cierta autonomía a este tipo de máquinas. Así, no se le puede dar el término de robot a este tipo de artefactos.

La primera máquina con patas que caminaba fue la *Walking Truck* (Raibert 1986), la cual contaba con cuatro extremidades, pertenecía a General

Electric y fue diseñada por R. Mosher. El control de esta máquina era totalmente manual debido a las limitantes tecnológicas de esos años, la figura 2.6 muestra una imagen de dicha máquina.

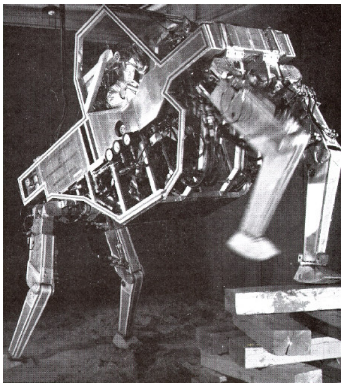


Figura 2.6. Walking Truck. Tomada de (Raibert 1986)

Como podemos ver en la figura 2.6 el objetivo de crear este tipo de máquinas era desarrollar una máquina capaz de moverse en terrenos en los cuales una máquina con llantas no pudiese moverse y esto se consiguió parcialmente usando patas en lugar de llantas.

2.4.2 Control orientado a metas y objetivos

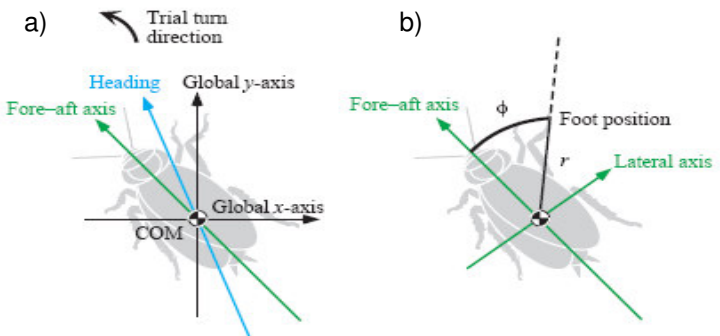
Una vez desarrolladas las primeras máquinas con patas, los diseñadores se dieron cuenta de que era necesario dotar de cierta autonomía a este tipo de máquinas, para evitar de esta forma que el diseñador interviniera de manera continua durante el funcionamiento de la máquina.

De esta forma se empezaron a diseñar sistemas de control que permitieran a la máquina, en principio, moverse y mantenerse estable. Como una primera forma de resolver este problema surgió un enfoque el cual básicamente consiste en dotar a la máquina de una tabla gigantesca que contempla todas las situaciones a las que la máquina puede enfrentarse. A este tipo de

máquinas ya podemos asignarles el término de robots, pues pueden alcanzar una meta u objetivo específico, como puede ser, mantenerse de pie o desplazarse. Es aquí en donde podemos empezar a hablar de robots caminantes semi-autónomos y autónomos.

Entre los trabajos en los cuales se usa un control para hexápodos orientado a metas y objetivos podemos mencionar los de Jindrich et al. (1999) y Chen et al. (2003). En este tipo de control se realiza una caracterización amplia y exhaustiva de todas las posibles situaciones en las que puede llegar a encontrarse el robot, y se programan todas las acciones de acuerdo al estado en el que se encuentre el robot. Por lo general se utilizan modelos matemáticos complejos y se usa el paradigma SPA, (Maes & Brooks 1990).

En el trabajo de Jindrich et al. (1999) se utiliza cinemática de dos dimensiones para estimar las fuerzas totales y de troqué (fuerza aplicada a una palanca que hace rotar alguna cosa), dicho modelo se ilustra en la figura 2.7. Además, se usa una técnica fotoelástica para estimar las fuerzas de *reacción – tierra*, fuerzas que se presentan al momento en que una pata hace contacto con la superficie de apoyo, de una pata durante un giro. Para interpretar la fuerza de una pata que se observa durante un giro, los autores desarrollaron un modelo general que relaciona la fuerza producida de una pata y su posición cuando se está llevando a cabo un giro. Este modelo predice que todas las patas podrían girar al cuerpo. Los autores realizan un estudio acerca de las consideraciones anteriores para tomarlas en cuenta en el desarrollo de un control para un robot hexápodo pero no llegan a implementar algún control. Así, este trabajo tiene un carácter esencialmente teórico.



c) d)

Figura 2.7. Modelo de cinemático de Jindrich. a) Sistema de tres coordenadas, b) media de la posición de una pata en coordenadas polares, c) sistema de coordenadas simplificado y d) variables cinemática. Tomadas de (Jindrich et al. 1999)

En el trabajo de Che et al. (2003) se prueba en un robot cuadrúpedo (*Planar Walker*) un control basado en un mecanismo de circuito cerrado, dicho robot se muestra en la figura 2.8. Este robot puede producir movimientos en dos direcciones sobre el plano horizontal (derecha-izquierda, adelante-atrás) y es capaz de rotar sobre él mismo. Los pasos que permiten al robot desplazarse en forma horizontal, los pasos de giro del robot con sus respectivas longitudes finitas y ángulos finitos de rotación son obtenidos con el sensado de los cilindros y de los "agarradores" (dispositivos que producen fuerza de agarre a la superficie) que forman las patas del robot. Las secuencias de actuación de los cilindros y de los "agarradores" para generar diversos tipos de pasos son modeladas usando autómatas finitos. La cinemática de los pasos se describe usando mecánica de movimiento de cuerpo rígido, la cual estudia el movimiento y equilibrio de los cuerpos sólidos ignorando sus deformaciones. Cuando una serie de pasos se ejecuta, el robot sigue una trayectoria segmentada no-lisa, esto es una trayectoria no lineal como lo haría un vehículo con llantas. De acuerdo con estas

características, tres métodos de navegación se desarrollaron para ser probados en un robot cuadrúpedo en varios panoramas: algoritmo de Simple Línea de Vistas (*SLS, Simple Line of Sight*), Planificación Exacta basado en Recocido Simulado (*SAAP, Simulated Annealing based Accurate Planning*) y el algoritmo Híbrido de Localización de Planeamiento Exacto (LHAP, Localized Hybrid Accurate Planning).

Los autores concluyen a partir de experimentos en simulación que el algoritmo SAAP produce secuencias exactas de pasos y el algoritmo LHAP ahorra tiempo y recursos de cómputo para lograr metas de largo alcance, estas metas pueden ser consultadas en [URL_Metas]. Sin embargo, los resultados experimentales demuestran que los errores posicionales inherentes al movimiento (incertidumbre de resultado de las acciones) individuales del paso pueden ser sustanciales y cuando estás se acumulan pueden hacer que un algoritmo particular resulte menos eficaz.

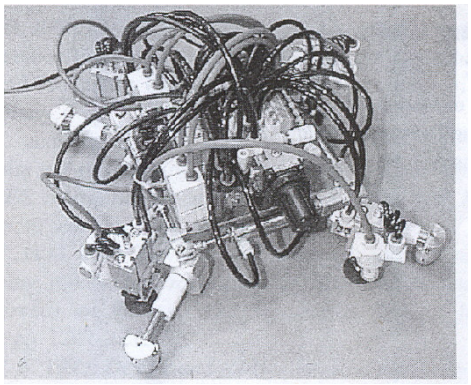


Figura 2.8. Robot de Chen. Tomadas de (Chen et al. 2003)

Kindermann (2002) presenta un sistema cinemático complejo de un robot hexápodo con un total de 18 DOF. Este sistema muestra una variedad de comportamientos los cuales son controlados por un sistema de control no trivial. Fue necesario realizar un estudio detallado cuantitativo de cada uno de los comportamientos del robot para comprender y abstraer las

propiedades de dichos comportamientos. Esta “etología artificial” es aplicada para controladores con una estructura descentralizada que utiliza un modelo base, en este caso el modelo del insecto palillo. El sistema aquí mostrado es capaz de adaptarse a condiciones externas imprevistas sin necesidad de reprogramación, debido a que el sistema considera y utiliza las configuraciones recurrentes que se presentan en un ciclo de movimiento a través del ambiente. El sistema de control le permite al robot sobrepasar obstáculos, que recuperarse de tropiezos o caminar con una pérdida total o parcial de una de sus patas. La figura 2.9 muestra imágenes del modelo y del robot de Kindermann.

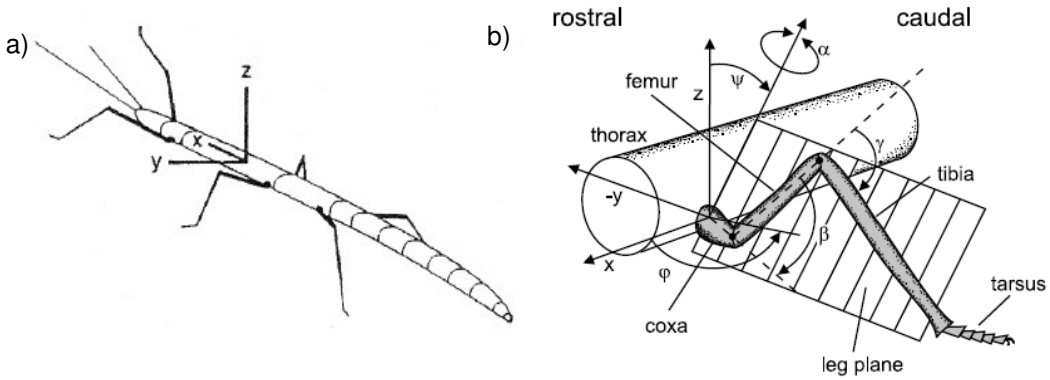


Figura 2.9. Modelo de Kindermann (2002). a) Esquema del insecto palillo, b) modelo de una pata del insecto. Tomadas de (Kindermann 2002)

Como se ha discutido previamente, en este tipo de enfoques se invierte una gran cantidad de trabajo y esfuerzo en el modelado matemático del problema. Además, generalmente no se considera la incertidumbre que existe en la ejecución de una acción del robot pues en estos enfoques se idealizan tanto la percepción como el resultado de las acciones ejecutadas por el robot.

2.4.3 Control biológicamente inspirado

Debido a las carencias de los enfoques antes discutidos, los investigadores buscaron nuevas formas de diseñar sistemas de control eficientes para robots con patas, uno de los enfoques que ha presentado buenos resultados es el conocido como biológicamente inspirado. Este enfoque nace del estudio del comportamiento de los insectos reales y básicamente trata de imitar en menor o mayor proporción las características biológicas de los insectos (Kerscher et al. 2004). En el trabajo de Kerscher et al. (2004) se describen los diferentes niveles de inspiración biológica en el diseño de robots con patas. Así, la categoría de los robots con patas biológicamente inspirados contempla cualquier robot en el cual alguno de los componentes de cuyo sistema, e. g. diseño mecánico, sistema de actuadores, arquitectura de control o control adaptativo; está ampliamente influido por un diseño con principios de funcionamiento encontrados en la naturaleza. Es por eso que en esta sección discutiremos algunos de los robots con patas los cuales están biológicamente inspirados en algunos de los niveles que menciona Kerscher et al. (2004).

2.4.3.1 Hardware

En esta sección consideramos aquellos robots con patas cuyos diseños mecánicos y su sistemas de actuadores están biológicamente inspirados.

En el trabajo de Kingsley (2006) se presenta la construcción de un robot hexápodo inspirado directamente de la fisonomía de las cucarachas que intenta aprovechar la ventaja de la neuromecánica, i e. sistemas que combinan controladores basados en redes neuronales y dispositivos mecánicos, y de los actuadores con un sistema enfocado a reproducir la funcionalidad de los músculos reales de tal insecto, con el fin de demostrar una las ventajas que tienen con respecto al uso de actuadores convencionales. Los actuadores convencionales son poco factibles para

capturar con precisión los rangos de movimiento de un robot hexápodo. Este trabajo muestra que dotando de 24 DOF a un robot hexápodo podemos obtener una muy buena aproximación de un robot capaz de caminar y escalar. Este trabajo citado describe el diseño físico del robot y muestra que es capaz de desplazarse en circuitos abiertos y accidentados. Utiliza un controlador de movimiento hacia adelante sin alguna retroalimentación y el robot es capaz de producir una locomoción hacia adelante aceptable. La figura 2.10 muestra el robot hexápodo construido en este trabajo.

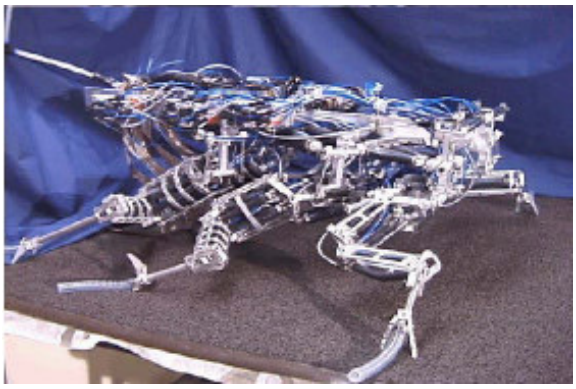


Figura 2.10. Robot de Kingsley. Tomada de (Kingsley 2006)

En Kerscher (2004) se describe la estructura mecánica de un robot hexápodo biológicamente inspirado llamado *Airinsect*, el cual se basó en el modelo natural del insecto *palillo*. El enfoque usado para la construcción de Airinsect, a diferencia de otros robots hexápodos construidos inspirados en dicho insecto, es que el arreglo de patas y el diseño de las empalmes fueron construidos como una exacta imitación de cómo se encuentran en el insecto verdadero. Para reducir el peso de este robot se utilizaron materiales ligeros. Para hacer uso de las ventajas de este robot se diseñó un sistema de control basado en un modelo de retroalimentación a nivel del microcontrolador. Dicho control calcula y ajusta los ángulos deseados de las empalmes del robot. Para el control de alto nivel son utilizados los algoritmos de control del robot LAURON. La figura 2.11 ilustra el robot Airinsect.

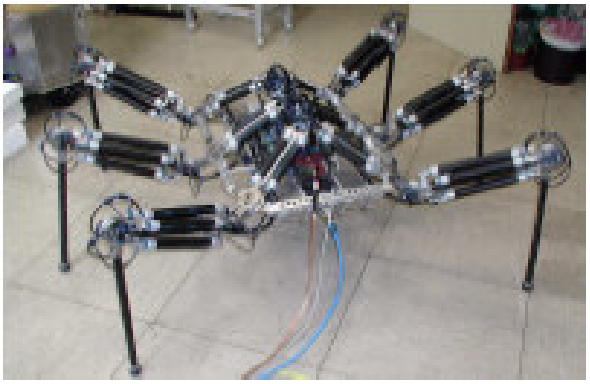


Figura 2.11. Robot Airinsect. Tomada de (Kerscher 2004)

En Weidemann (1994) se presenta una prueba de un robot hexápodo diseñado y construido en el Instituto de Mecánica de la Universidad Técnica en Munich. Este robot fue diseñado basándose en principios inspirados en la forma natural de caminar de los hexápodos. En este trabajo se describe al insecto investigado y sus características principales a pesar de que no se da una descripción del hardware mecánico y electrónico usados en la construcción. El trabajo está más apegado a un enfoque biológicamente inspirado a un nivel de hardware, aunque no se detalle el sistema de control utilizado.

En este tipo de trabajos se argumenta que, a partir de los resultados obtenidos de las investigaciones concernientes a la morfología y neuroetología de los animales, para obtener un robot con patas biológicamente inspirado se debe diseñar un sistema mecatrónico, i. e. sistemas mecánicos extendidos e integrados con sensores, microprocesadores y controladores, y un sistema de sensores capaces de obtener información altamente confiable acerca del estado interno del robot, así como de la situación del ambiente en el que se encuentran inmersos. De esta manera se descarga la complejidad del sistema de control para el robot.

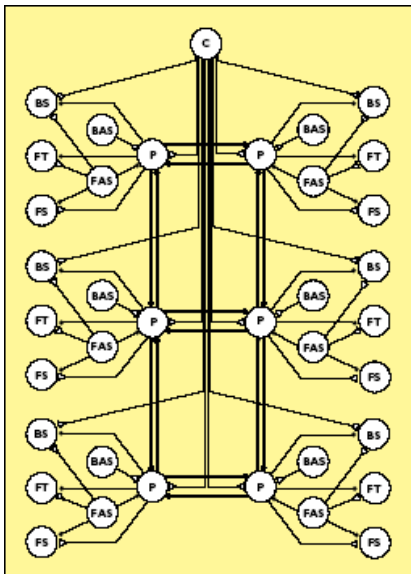
Debido a que los actuadores que comúnmente son usados en los robots móviles, y por ende en los robots con patas, no son capaces de proporcionar la velocidad, precisión y el torque deseados para realizar el movimiento de un robot con patas, existe incertidumbre acerca de cuál es el verdadero resultado de ejecutar cierta acción para hacer que un robot con patas se desplace. La tarea de realizar un sistema de control eficiente para un robot con patas el cual cubra el impacto de la incertidumbre antes mencionada se vuelve una labor complicada. Es por eso que este tipo de trabajos invierten gran cantidad de esfuerzo y recursos para definir un buen sistema mecatrónico y un sistema de sensores adecuado para el robot.

Otros trabajos que también podemos situar en este nivel de los robots hexápodos biológicamente inspirados son los presentados en (Cruse 1976) y (Berns 2002).

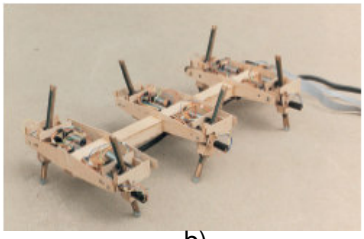
2.4.3.2 Sistema de control

En esta sección consideramos aquellos robots con patas cuya arquitectura de control o control adaptativo están biológicamente inspirados.

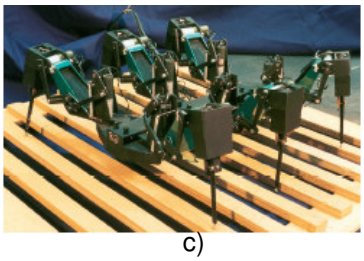
Beer et al. (1997) estudiaron la manera de diseñar el control de un robot con patas basado directamente en estudios de insectos reales y su sistema nervioso. Estos autores se inspiraron en un enfoque biológico para definir el diseño y control de un vehículo de exploración robusto. Su aportación fue el uso de redes neuronales para crear un modelo distribuido y robusto del control de un robot hexápodo.



a)



b)



c)

Figura 2.12. a) Esquema de la red neuronal utilizada por Beer b) Robot I de Beer c) Robot II de Beer. Tomadas de (Beer et al. 1997)

En el trabajo de Schneider (2006) se realiza un estudio acerca de la generación de movimientos en cadenas cinemáticas cerradas argumentando que la dificultad de éstas se debe a que todos los empalmes que participan en dichas cadenas tienen que ser movidos de una manera altamente coordinada para evitar tensiones destructivas en la pata de un robot. Schneider presentó un controlador descentralizado para los empalmes el cual utiliza las interacciones de bajo nivel que existen entre un empalme en movimiento y la consistencia de sus empalmes vecinos, el cuerpo y el entorno en el que se encuentra el robot. Este controlador está basado en un mecanismo (LPVF, *Local Positive Velocity Feedback*) el cual explota las características elásticas del empalme. La estrategia de control está inspirada por el sistema de locomoción de los insectos palillo. Schneider demuestra que una cadena cinemática cerrada que consiste de varios controles LPVF, a pesar de carecer de unidad central de control, puede solucionar tareas sin necesidad de una coordinación de alto nivel entre los empalmes. Su trabajo

fue probado sobre un manipulador que da vuelta a una manivela en una pata de un robot con 3 DOF.

En el trabajo de Berns (1994) se presenta una manera de construir una arquitectura de control basada exclusivamente en redes neuronales. Las redes neuronales son adecuadas para controlar tareas especializadas debido a su capacidad de procesamiento en tiempo real, tolerancia a fallos y adaptabilidad (Berns 1994). Para probar dicha arquitectura Berns utiliza un robot hexápodo. Finalmente, Berns da una descripción de los requerimientos de hardware necesarios cuando se utiliza una arquitectura de control basada en redes neuronales.

Otro rama de robots con patas biológicamente inspirados a nivel del sistema de control, comprendiendo aquellos trabajos que estudian la manera de generar un CPG (C*entral Pattern Generator*). El CPG es una red neuronal biológica la cual se encarga de controlar los movimientos rítmicos de un ser vivo, los cuales le permiten tener una locomoción estable al caminar, correr, nadar y volar. El CPG genera un patrón rítmico de actividad nerviosa de manera inconsciente y automática Dicho patrón rítmico activa neuronas motor, de esta manera los movimientos rítmicos de los animales son generados. Un sistema sensorial de retroalimentación regula la frecuencia y la fase del patrón rítmico generado por el CPG. Entre los trabajos más representativos podemos listar los siguientes.

En el trabajo de Nakada (2003) se propone un controlador CMOS analógico denominado por su autor como “neuromorfológico” (*nueromorphic*) para la coordinación de las patas en un cuadrúpedo. El autor utiliza un CPG como controlador de la locomoción de un robot. Muchos de los controladores que usan un CPG han sido desarrollados con un controlador digital y por esta razón presentan varias dificultades para su uso, una de las dificultades más

importantes es su alto consumo de energía. Para resolver ese problema, Nakada propone el uso de un controlador CMOS analógico. Además, usando tal controlador se espera reducir el costo de producción y obtener una buena miniaturización del dispositivo. Nakada prueba su controlador sólo a nivel de simulación, el circuito obtenido es capaz de generar varios patrones de movimiento rítmicos y de cambiar de manera rápida entre patrones. Así, el control básicamente consiste en construir un CPG y para alcanzar tal objetivo se utilizan redes neuronales, las neuronas que se utilizan en este trabajo son de tipo Amari – Hopfield, ver figura 2.13.

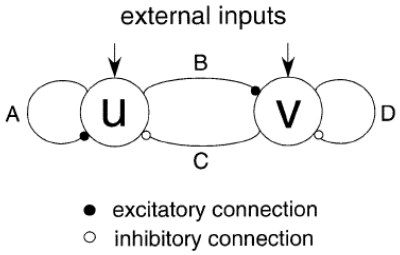


Figura 2.13. Modelo de las neuronas de Amari – Hopfield. Dicho modelo consiste de una neurona exitatoria y una neurona inhibidora, con una conexión inhibidora - exitatoria. Tomada de (Nakada 2003)

Otros trabajos relacionados en controles de robots con patas usando CPG son (Still & Tilden 1998), (Patel et al. 1998) y (Lewis et al. 2001) de Nakada (2003).

Lewis (1994) describe el estado de evolución para crear un CPG complejo para el control de los movimientos de las patas de un robot hexápodo. Como ya se mencionó, el CPG está compuesto por una red neuronal. En este trabajo se utiliza un algoritmo genético el cual determina la manera de interconectar las neuronas en una red neuronal en lugar de utilizar algoritmos de aprendizaje que es la manera típica de determinar las conexiones en una red neuronal. El uso de un algoritmo genético es para incrementar la velocidad a la que convergen los pesos de la red neuronal usada, de modo

que de esta manera se obtiene de una manera más rápida un comportamiento dirigido a alcanzar una meta. Primero se diseña un oscilador para los movimientos individuales de una sola pata, posteriormente una red de esos osciladores es diseñada para coordinar los movimientos de las diferentes patas.

Debido a que los robots biológicamente inspirados tienen un gran número de sensores y actuadores se necesita reducir la complejidad del problema usando arquitecturas de control descentralizadas (Albiez 2004).

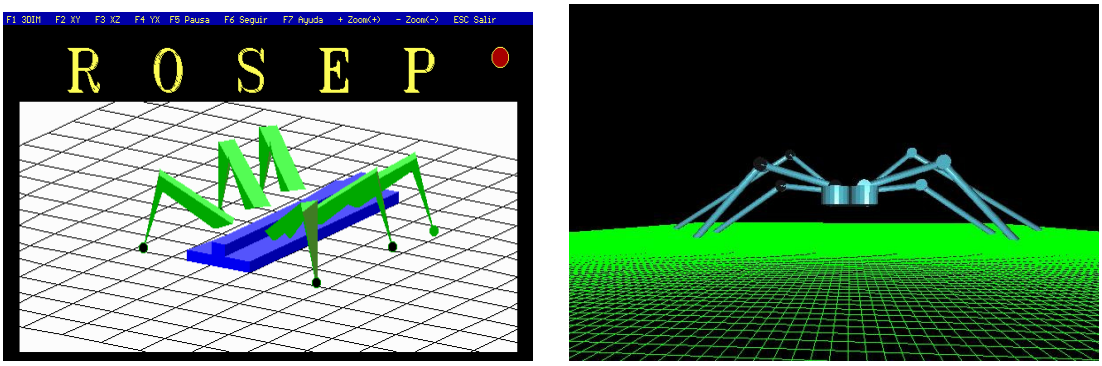
Estos últimos trabajos están orientados esencialmente a realizar controles para robots cuya riqueza en cantidad de sensores y actuadores es muy poderosa. Además, para garantizar un control eficiente este enfoque requiere arquitecturas de control muy detalladas, además de requerir una caracterización muy fina de las lecturas de los sensores y actuadores. Además Nakada afirma, con respecto a la coordinación de patas, que a partir de que el número de grados de libertad por pata en un robot polípedo suele ser grande (≥3) se requiere una coordinación eficiente de las patas para tener un desplazamiento estable. Por ello, Nakada propone que el uso de un CPG es muy apropiado para llevar a cabo dicha coordinación y afirma también que el CPG juega un papel fundamental en la locomoción de robots con patas (Nakada 2003).

2.4.4 Control difuso

Existe la alternativa de usar lógica difusa para diseñar un sistema de control para un robot hexápodo.

En el trabajo de Gorrostieta y Vargas (2004) se presentan diversos algoritmos, los cuales le permiten a un robot hexápodo realizar un

desplazamiento mediante un conjunto de acciones no establecidas ni periódicas, lo cual se conoce como locomoción libre. Los algoritmos presentados en este trabajo utilizan técnicas de lógica difusa para tomar la decisión sobre qué pata será movida de acuerdo a la información que se tenga en un instante dado. La valoración de los algoritmos se realizó mediante la simulación del robot. En este trabajo también se describe una medida cuantitativa de estabilidad del robot, la cual es aplicable para robots con una morfología rectangular. En la figura 2.14 se muestran imágenes del prototipo del robot hexápodo en el simulador.



La figura 2.14. Imágenes del robot hexápodo en simulación. Tomadas de (Gorrostieta y Vargas 2004)

Berardi et al. (1996) describe a DANIELA, un sistema neuro-difuso (*neuro-fuzzy*) para controlar un robot hexápodo. El sistema está basado en un dispositivo neuronal hecho a la medida, el cual puede implementar perceptrones multicapa, funciones de base radial o lógica difusa. El sistema implementa algoritmos de control inteligente mezclados con algoritmos neuro-difusos con autómatas de estados finitos, para controlar un robot hexápodo. Para el control del robot hexápodo se utiliza un control jerárquico el cual consiste en un controlador neuro-difuso y un autómata de estados finitos. Dicho control se organiza en tres capas: control de coordinación de movimiento, control de patas y control de las posiciones de los empalmes. El

controlador neuro-difuso se utiliza en el control de la coordinación de movimiento. La figura 2.15 muestra imágenes de dicho robot.

La figura 2.15. Robot de Berardi. Tomada de (Berardi et al. 1996)

Ding (1996) argumenta que usando lógica difusa el usuario es capaz de definir reglas lingüísticas para especificar un mapeo no lineal entre las señales de entrada de los sensores y las señales de salida de los actuadores, para así proporcionar un marco para programar un sistema embebido. Usando un robot hexápodo como cama de prueba, implementaron lógica difusa sobre un microcontrolador el cual controla dicho robot. En resumen, el trabajo de Ding muestra la implementación de un sistema de control basado en lógica difusa en un sistema embebido, probado en un robot hexápodo.

Al usar técnicas de lógica difusa para crear un sistema de control, se tienen dos desventajas principales. La primera de ellas es la dificultad de estudiar la estabilidad del sistema y la segunda es la necesidad de que un experto suministre su conocimiento. La segunda desventaja conlleva ciertos problemas como son: la dificultad de hacer explícito el conocimiento del experto, la dificultad de representación de dicho conocimiento sin llegar a empobrecerlo, y la posible incoherencia de la información proporcionada por el experto. Además, al usar lógica difusa se tiene un elevado número de

parámetros relacionados y no existen métodos sistemáticos y generales para el ajuste de los mismos.

2.4.5 Arquitectura subsumción

El enfoque orientado a metas invierte una gran cantidad de esfuerzo de cómputo en el procesamiento de los datos necesarios para tomar una decisión, es por eso se desarrollaron nuevas formas de realizar sistemas de control basados en el paradigma SA, (Maes & Brooks 1990), con la finalidad de tener sistemas que pudieran responder rápidamente.

Maes & Brooks (1990) fueron de los primeros investigadores que trataron de resolver el problema de diseñar un control para un robot hexápodo (ver figura 2.16) más adecuado a los diseñados hasta ese entonces. Ellos definieron un algoritmo que permitía a un robot hexápodo basado en comportamiento aprender a partir de la retroalimentación, positiva y negativa, cuando se activaban acciones individuales de un conjunto predeterminado. De acuerdo con la filosofía de los robots basados en comportamiento, el algoritmo de control se distribuye totalmente: cada una de las acciones individuales intenta independientemente descubrir (i) su relevancia, i. e. si está totalmente correlacionada con la retroalimentación positiva, y (ii) las condiciones bajo las cuales llega a ser confiable, i. e. las condiciones en las cuales se maximiza la probabilidad de recibir la retroalimentación positiva y al mismo tiempo se reduce al mínimo la probabilidad de recibir la retroalimentación negativa.

La principal carencia que tiene este enfoque es que el robot tiene que invertir tiempo en una fase de aprendizaje para probar acciones repetidamente y reconocer su efecto en el ambiente, así que cuando su ambiente sufre un cambio significativo, i. e. introducción de elementos nuevos al ambiente, el

robot necesita volver a invertir tiempo en la fase de aprendizaje para desplazarse de manera estable en el nuevo ambiente.

Figura 2.16. Prototipo del robot de Brooks. Tomada de [URL_Brooks_Robot]

En (Ferrell 1995), se exponen las dificultades a las que se enfrentan los diseñadores de sistemas de control para robots con un gran número de sensores, actuadores y comportamientos. Además, explora la factibilidad de usar la cooperación entre controladores locales para alcanzar a realizar un comportamiento global correcto. Ferrell diseñó e implementó un sistema de control para un robot hexápodo llamado HANNIBAL. Ferrell descompuso una tarea de control global de un robot hexápodo, e. g. caminar, en un conjunto de tareas de control más simples que la tarea global, para diseñar el sistema de control. HANNIBAL es un robot hexápodo compuesto por una gran variedad de sensores y actuadores sofisticados, con la finalidad de tener mayor fiabilidad de las lecturas de los sensores y de los resultados de las acciones. El sistema de control obtenido por Ferrell está basado en la arquitectura de control subsumción de Brooks, y es capaz de ejecutar una diversidad de comportamientos, e. g. caminar y evadir de obstáculos.

En (Celaya & Albarral 2003) se presenta una estructura de control para un robot hexápodo el cual fue previamente desarrollado y simulado en IRI (*Institut de Robòtica i Informàtica Industrial*). Posteriormente se probó sobre un robot hexápodo con 2 DOF en cada pata. Más tarde esta misma

estructura se ha implementado en un robot con tres grados de libertad por cada pata (LAURON III, de FZI, *Forschungs Zentrum Informatik*). Los principales aspectos que requirieron una atención especial fueron la implementación de los módulos de subsumción (Brooks 1984), la arquitectura de control nativa del robot, y las dificultades para llevar a cabo la coordinación de los movimientos de las patas con los actuadores disponibles. La estructura de control que se utilizó en IRI es centralizada. Celaya probó dicha estructura de control pero con una arquitectura descentralizada en un robot hexápodo con 2 grados de libertad por pata (Genghis II). Finalmente este nuevo sistema de control fue probado en simulación en un robot hexápodo con 3 DOF. Los autores hicieron una descomposición de tareas en varios espectros de habilidades como se propone en la arquitectura subsumción de Brooks. Ambos robots se muestran en la figura 2.17.

Figura 2.17. a) Lauron III, b) Genghis II. Tomadas de [URL_Subsumción_Robots]

2.4.6 Control usando aprendizaje por refuerzo

La idea de dotar a un robot de una cierta capacidad de aprendizaje surge de la motivación de tener robots lo suficientemente autónomos para reducir a lo mínimo la intervención del diseñador, esto es, dotar al robot de la capacidad de adaptarse a su medio a pesar de que éste cambie. Como ya se ha mencionado, por sus características físicas los robots con patas son deseables debido a la potencia que tienen de poder desenvolverse en

ambientes parcialmente desconocidos, pero para explotar tales características se debe dotarlos de un mecanismo de control que les permita adaptarse a cambios inesperados de su ambiente.

En todos los trabajos anteriores se busca encontrar ese mecanismo de control. Un nuevo enfoque para lograr crear ese mecanismo es el uso de aprendizaje por refuerzo, el cual ha demostrado ser adecuado para dotar de autonomía a un robot. Este enfoque se ha utilizado para el diseño de sistemas de control para robots con patas, debido a que trabaja de una manera más sencilla de como lo hacen los controles biológicamente inspirados, además de que no necesita tener un sistema mecatrónico y de sensores muy sofisticados, de nuevo como los robots biológicamente inspirados. En contraste con los sistemas de control orientados a objetivos y metas, el diseño de sistemas de control usando aprendizaje por refuerzo es relativamente más sencillo.

Lee et al. (2006) reportan un uso acertado del aprendizaje por refuerzo en problemas de evasión de obstáculos con un robot cuadrúpedo (ver figura 2.18). El algoritmo que presentan estos autores se basa en una descomposición jerárquica de dos niveles de la tarea, en la cual el control de alto nivel selecciona una configuración específica de las patas y el control de bajo nivel genera los movimientos continuos para mover cada pata del robot a una posición específica. El control de alto nivel usa un estimado de la función de valor para guiar su búsqueda, en este caso orientada a encontrar una configuración de las patas del robot que le permita mantenerse estable; el valor estimado se obtiene usando un algoritmo de aprendizaje semi-supervisado. El controlador de bajo nivel se obtiene por medio de la búsqueda de política sobre campos potenciales. Finalmente, los autores demostraron que su robot lograba contender con una variedad de obstáculos que no fueron considerados en el tiempo de entrenamiento.

Figura 2.18. Robot de Lee. Tomada de (Lee et al. 2006)

La desventaja de este enfoque es que puede tomar un tiempo considerable para converger a una política buena, probablemente no la óptima. Esto es debido a que el robot no posee información alguna acerca del modelo del ambiente y debido a eso tiene que pasar por una fase de entrenamiento para adquirir un modelo del ambiente.

2.4.7 Procesos de Decisión de Markov (MDPs)

En los últimos años, los MDP han sido aplicados en problemas de navegación robótica obteniendo buenos resultados. Sucar (2004) propuso una estructura para resolver problemas de decisión complejos basándose en la división del problema en problemas más pequeños que pueden resolverse independientemente, y en la combinación de estas soluciones para obtener una solución global óptima. Cada parte del problema es representada como un MDP que se resuelve independientemente. Cada MDP envía un valor para cada posible acción a un "árbitro", el cual se encarga de seleccionar las mejores decisiones que en conjunto maximicen la ganancia conjunta. Es aquí en donde se introduce el concepto de MDPs jerárquicos. En este trabajo se propone diseñar un control eficiente que le permita a un robot hexápodo desplazarse de manera estable usando MDPs jerárquicos, en cuyos fundamentos se abundará en el capítulo siguiente.

Pineau et al. (2003) presentan un algoritmo de control escalable que permite a un robot móvil desenvolverse, tomando decisiones de control de alto nivel, bajo consideraciones de creencia probabilística. Los autores dan información acerca de la estructura de control de un robot y de los MDPs jerárquicos para proponer un algoritmo de control probabilístico jerárquico. Con dicho algoritmo, el robot aprende conjuntos de subtareas específicas y de políticas. El control desarrollado fue exitoso al ser probado en un robot móvil que asistía a una enfermera.

2.5 Discusión

En este capítulo se hizo una revisión de la historia y del estado del arte de los robots polípedos, principalmente hexápodos.

El enfoque de control orientado a metas y objetivos requiere un gran esfuerzo en la definición del modelo matemático. Para obtener un buen modelo se requiere que se haga una caracterización muy fina de todos los factores que afectan al ambiente y al robot, como son sensores, actuadores, incertidumbre de acciones, cambios en el ambiente, etc. Por ello, los sistemas de control bajo este enfoque pueden llegar a ser complejos e insuficientes.

Los sistemas de control biológicamente inspirados requieren sistemas mecatrónicos y sistemas de sensores muy sofisticados, con el fin de minimizar el impacto de la incertidumbre en el resultado de las acciones de un robot. Como se mencionó, los sistemas de control biológicamente inspirados parten de la idea de tener sistemas de sensado y actuación muy precisos. Teniendo en cuenta lo anterior, este enfoque se concentra principalmente en desarrollar sistemas de control a un nivel muy detallado para aprovechar al máximo las características físicas, y los sistemas de

sensado y actuación del robot. Para alcanzar tal objetivo con frecuencia se utilizan redes neuronales, las cuales, al igual que los sistemas de control basados en lógica difusa, proporcionan buenos resultados pero tienen la desventaja, al no ser descriptivos, de no facilitar el estudio de su estabilidad pues no existen procedimientos sistemáticos para el ajuste de sus parámetros y de carecer de una metodología general de diseño. Además, en el caso de los sistemas basados en lógica difusa se requiere que un experto suministre conocimiento sobre la locomoción y estabilidad del sistema de un robot hexápodo, en este caso.

Por otro lado, tenemos el enfoque de aprendizaje por refuerzo para obtener un sistema de control, pero como ya se mencionó, puede llegar a ser muy tardado el tiempo que le toma al robot encontrar las relaciones entre percepciones y acciones en dicho control, sin tener la certeza que el sistema de control aprendido por el robot es el control óptimo. Esto es debido a que en este tipo de enfoque no se utiliza ningún modelo de referencia de donde el robot pueda partir para obtener un sistema de control óptimo.

Debido a que el sistema de control involucra la especificación de una relación entre las señales de entrada y las señales de salida para los actuadores, es necesario tener un método el cual permita obtener dicha relación de una manera más descriptiva y clara. Aún más, necesitamos saber si el sistema de control bajo ciertas consideraciones iníciales o modelo inicial es el mejor. Ahora bien, para evitar caer en los problemas de los enfoques orientados a metas y objetivos, este modelo inicial no debe de ser tan sofisticado, i. e. no usar modelos que consideren leyes físicas, matemáticas, de conservación, etc. y además debería permitirnos considerar de manera transparente la incertidumbre que existe en el resultado de las acciones del robot, i. e. no tener que caracterizar explícitamente esa incertidumbre, para así evitar la fuerte inversión en sistemas mecatrónicos y sistemas de sensado

sofisticados. Por estas razones, el trabajo de esta tesis se enfoca en obtener un sistema de control que cubra los aspectos anteriores.

Después de revisar los trabajos de Sucar y Pineau, la idea de utilizar MDPs para obtener un sistema de control óptimo es una buena alternativa para cubrir con los requerimientos anteriores, debido a que los MDPs proporcionan modelos más descriptivos, los cuales son útiles para analizar la respuesta del robot a situaciones específicas del mundo, así como las relaciones entre los actuadores del robot para generar configuraciones estables que permitan el desplazamiento. Los MDPs trabajan con un modelo, el cual incorpora información mínima acerca del entorno y del robot mismo. Dicho modelo nos permite considerar la incertidumbre que existe en los resultados de las acciones de un robot. Los MDPs son modelos probabilistas sustentados con una matemática formal, quiere esto decir que tienen metodología general de diseño y debido a los métodos con los cuales se resuelven los MDPs tenemos garantía de que el resultado es el óptimo. Esto es, considerando la información proporcionada al robot y la forma en que se caracterizaron los estados y acciones del robot, este último siempre ejecutará la mejor acción orientada al logro de una meta específica.

En los trabajos sobre MDPs revisados no se encontró reporte sobre el uso de MDPs para construir sistemas de control para robots con patas. Nuestro trabajo es pionero, hasta donde sabemos, en la aplicación de MDPs para el control de este tipo de robots.

CAPÍTULO III: Procesos de Decisión de Markov y Procesos de Decisión de Markov Jerárquicos

3.2 La necesidad de adaptación

Las capacidades de aprendizaje de los robots primitivos (Raibert 1986) eran nulas. El principal interés de Raibert (1986) era programar a sus robots con ciertos comportamientos que ellos mismos elegían basados en su propia experiencia. Esto acarreó tres problemas fundamentales, que se discuten a continuación.

En primer lugar, sucede muy a menudo que los **comportamientos** que los diseñadores de robots consideran adecuados, demuestran ser **poco útiles o incluso contraproducentes** al llevarlos a la práctica. Esto se debe a que en algunos casos, la única manera de definir estos comportamientos es a partir de simulaciones y de posteriores refinamientos de tales simulaciones. En otros casos, los comportamientos del robot se definen a partir de modelos matemáticos, aplicando métodos de ingeniería de control, con los que ocurre algo parecido al llevarlos a la práctica: el modelo matemático no se ajusta totalmente a lo esperado debido a factores no considerados. En el primer caso se termina dotando al robot con una serie de reglas y parámetros *ad-hoc*, que no surgen realmente de un conocimiento profundo del propio robot y de su interacción con su entorno, sino que se obtienen a partir de múltiples pruebas de ensayo y error. Y en el segundo la construcción del modelo matemático puede convertirse en una tarea no trivial.

El segundo problema en el control de robots es **la falta de capacidad de adaptación de un robot** cuyo comportamiento fue diseñado en las formas

anteriormente descritas. Un robot sin capacidad de aprendizaje puede funcionar adecuadamente en un entorno controlado, pero al enfrentarse a un entorno distinto o aún el mismo entorno modificado, las acciones que antes eran adecuadas pueden volverse inútiles.

El tercer problema consiste en la **dificultad de optimizar el consumo de energía y los movimientos del robot**. La optimización del consumo de energía y de los movimientos del robot pueden calcularse de manera teórica, empleando simulación [URL_Fleifel]. Sin embargo, por las mismas razones expuestas en los párrafos anteriores, es muy probable que los cálculos teóricos no se ajusten a la perfección a las características particulares del robot y del entorno. Si el robot tuviese la capacidad de calcular su propio consumo y aprender cuáles son las acciones que consiguen reducir dicho consumo al mínimo para cumplir un objetivo, esto podría suponer grandes ahorros en tiempo y energía en la vida útil del robot.

Los problemas descritos previamente podrían solucionarse al menos parcialmente proporcionando a los robots capacidades de aprendizaje. En concreto, las técnicas de aprendizaje por refuerzo pueden ayudar a solucionar cada uno de estos tres problemas [URL_Fleifel].

Los paradigmas tradicionales para el control de robots, Jindrich et al. (1999), (Maes & Brooks 1990), por sí solos adolecen de la capacidad de dotar al robot con herramientas para sobrevivir en un entorno abierto y dinámico. Es por ello que se considera de tanta importancia proporcionar a los robots con mecanismos de aprendizaje y adaptación.

Como lo que se desea en esta tesis es brindar a un robot la suficiente autonomía para que se desempeñe en un ambiente dinámico, en este capítulo describimos las formas con las que se puede dotar a un robot de autonomía.

3.2 La consideración de incertidumbre

La resolución de problemas de razonamiento, planificación y aprendizaje bajo condiciones de incertidumbre ha recibido una considerable atención en la comunidad científica durante los últimos años. De hecho, la mayor parte de los problemas planteados en el campo de la robótica o de las interfaces hombre-máquina se caracterizan por estar sometidos a múltiples fuentes de incertidumbre que deben ser consideradas a la hora de diseñar sistemas de control y planificación robustos.

Considérese a modo de ejemplo el robot hexápodo descrito en esta tesis, el cual debe desplazarse de forma autónoma y estable en un entorno no estructurado. Para ello, el robot puede ejecutar diversas acciones, como levantar cierta pata, apoyarla, desplazarla. En la práctica, el efecto real de estas acciones no es del todo fiable, puesto que al tratar de realizar cualquiera de las acciones descritas anteriormente puede suceder que no se alcance una posición o la realización precisa de alguna acción. Lo mismo sucede con las observaciones realizadas por los sensores del robot para percibir el entorno, debido a que las lecturas de los sensores no son cien por ciento confiables. A pesar de todos estos errores e incertidumbres, el robot debe utilizar la información sobre las acciones ejecutadas y las observaciones recibidas para seleccionar las próximas acciones a ejecutar que le permitan conseguir su objetivo final de navegación.

En definitiva, se plantea un problema en que un determinado agente, un robot en nuestro caso, debe tomar decisiones sobre las acciones a realizar para modificar el estado del mundo y su misma condición, e. g. las posiciones de sus patas, hasta conseguir un cierto objetivo. En última instancia, se trata de un problema de planificación bajo condiciones de incertidumbre que afectan tanto el resultado de las acciones como la percepción del entorno.

La Planificación basada en Teoría de Decisiones (DTP, *Decisión-Theoretic Planning*) (Feldman & Sproull 1977, Boutilier et al. 1999) es una extensión de la planificación clásica que permite resolver problemas de toma secuencial de decisiones en sistemas en los que el efecto de las acciones es incierto y la información sobre el entorno incompleta. Para ello, se introduce el concepto de utilidad de una acción, que mide hasta qué punto el resultado de esa acción puede beneficiar o contribuir a la consecución del objetivo global de planificación.

De entre los modelos probabilistas o probabilísticos más utilizados en la DTP cabe destacar los Procesos de Decisión de Markov (MDP, *Markov Decision Processes*), aplicables a aquellos sistemas que cumplen la conocida propiedad de Markov; toda la historia pasada del sistema puede resumirse en su estado actual. Un MDP contempla el resultado estocástico de las acciones, describiéndolo mediante funciones probabilísticas de transición de estados. El resultado de la planificación en un MDP no es una secuencia de acciones a ejecutar como sucede en la planificación clásica (Fikes & Nilsson 1971), sino una política que determina la acción a seleccionar en función del estado actual que va tomando el sistema a lo largo de la ejecución de la tarea. Una parte sustancial de este capítulo se dedica a revisar los fundamentos de los MDPs.

3.3 Procesos de Decisión de Markov

En general, en los problemas de decisión las acciones adoptadas por el agente determinan no sólo la recompensa inmediata sino el siguiente estado del entorno, al menos probabilísticamente. Por lo tanto, el agente toma en cuenta el siguiente estado y la recompensa cuando decide tomar una acción determinada. En estos casos, el modelo óptimo considerado para toda la ejecución determinará cómo tomar en cuenta los valores obtenidos en el futuro. El agente debe ser capaz de aprender a partir de recompensas

demoradas: o sea, puede obtener una secuencia de pequeñas recompensas inmediatas primero, para finalmente llegar a un estado donde se obtiene un valor de recompensa alto. Es decir, el agente debe aprender cuál de las acciones es deseable tomando en cuenta la o las recompensas que pueden obtenerse en un futuro arbitrariamente lejano.

Los modelos de Markov se han aplicado con éxito en la resolución de problemas de navegación en el campo de la robótica, como ejemplos tenemos los trabajos de Simmons & Koenig (1995), Cassandra et al. (1996), Kaelbling et al. (1998), Koenig & Simmons (1998), Nourbakhsh et al. (1995) y Thrun (2000).

3.3.1 Formalización

Un MDP es un modelo matemático de un problema el cual explícitamente considera la incertidumbre en las acciones del sistema. La dinámica del sistema está determinada por una función de transición de probabilidad.

Por otra parte, para cualquier MDP siempre hay una política $\pi : S \to A$ óptima, que permite decidir en cada estado qué acción tomar de manera que se maximice la utilidad esperada. Esta política π es *estacionaria,* i. e. no cambia en función del tiempo, y *determinista,* i. e. siempre se elige la misma acción cuando se está en el mismo estado.

Formalmente, un MDP M es una tupla $M = \langle S, A, \Phi, R \rangle$, (Ocaña M., 2005), donde:

- S es un conjunto finito de estados del sistema.
- A es un conjunto finito de acciones, que se ejecutan en cada estado.
- $\Phi : AxS \to \Pi(S)$ *:* es la función de transición de estados dada como una distribución de probabilidades y la cual asocia un conjunto de posibles

estados resultantes de un conjunto de acciones en el estado actual. La probabilidad de alcanzar un estado s' realizando la acción a en el estado s se escribe $\Phi(a, s, s')$.

- $R: S \text{ x } A \rightarrow R$ es una función de recompensa. $R(s, a)$ es la recompensa que el sistema recibe si lleva a cabo la acción a en el estado s. Esta función define la meta que se quiere alcanzar.

Cabe mencionar que en este trabajo se ha adoptado la notación de Ocaña M. (2005), para la definición formal de un MDP. Sin embargo la función de transición y la función de recompensa pueden representarse como $\Phi : SxAxS \rightarrow \Pi(S)$ y $R: S\text{x}AxS \rightarrow R$ respectivamente debido a que en ambos casos el resultado de cada función está asociada a un estado al que se llega con la acción realizada.

Una política para un MDP es una asociación $\pi : S \rightarrow A$ que selecciona una acción por cada estado, es decir, define cómo se comporta el sistema en un determinado estado, y puede verse como un mapeo de los estados a las acciones.

La función de valor V: indica lo que es "bueno" a largo plazo. Es la recompensa total que un agente puede esperar acumular empezando en ese estado, de alguna manera representa las predicciones de recompensas. Se buscan hacer acciones que den los valores más altos, no la recompensa inmediata mayor.

Las recompensas están dadas por el ambiente, pero los valores se deben estimar o aprender con base en las observaciones. Aplicando aprendizaje por refuerzo se aprenden las funciones de valor mientras el agente interactúa con el ambiente.

La solución a un MDP es una política que maximiza su valor esperado. Dos métodos comunes para resolver MDPs y determinar la política óptima son iteración de valores e iteración de política, (Ocaña M., 2005).

La propiedad de Markov dice: “No importa qué acciones se hayan llevado a cabo para alcanzar el estado actual, porque el estado actual es suficiente para decidir cuál debe de ser la acción futura”.

3.3.2 Políticas

Puesto que en un MDP el estado actual del entorno se considera completamente observable, i. e. que el agente sabe con exactitud en qué estado se encuentra, el único problema a resolver es determinar la acción a ejecutar en función de dicho estado, para conseguir el objetivo final. No se trata de un problema trivial, puesto que el efecto de las acciones no es determinístico. La única información de la cual se dispone para determinar la acción óptima a ejecutar es: la función de transición de estados T que caracteriza la incertidumbre en el efecto de las acciones, y la función de recompensa R que está relacionada con la utilidad de las acciones en función del objetivo final.

Por otro lado, un MDP es un proceso secuencial de toma de decisiones, y en este sentido es posible trabajar en dos contextos distintos. En el primero de ellos, conocido como de horizonte-finito, el agente sólo actúa durante un número finito y conocido de pasos k. Es obvio que en tal caso el objetivo final a la hora de seleccionar las acciones será maximizar la suma total de las recompensas obtenidas durante dichos pasos, tal como se representa en la ecuación (3.1),

$$E\left[\sum_{t=0}^{k-1} r_t\right] \qquad (3.1)$$

en donde r_t es la recompensa obtenida en el paso t.

En otras ocasiones, el agente actúa durante un número infinito o indefinido de pasos, recurriéndose en estos casos al modelo de horizonte-infinito. El objetivo también es maximizar la recompensa obtenida a largo plazo, utilizándose generalmente para ello un modelo con factor de descuento γ, $0 < \gamma < 1$, en el que el agente debe maximizar la ecuación (3.2).

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \qquad (3.2)$$

El factor de descuento permite tener en cuenta las recompensas futuras asegurando que la sumatoria tiende a un valor finito. Además, las recompensas recibidas en un futuro inmediato tienen más valor que las recibidas en el futuro a largo plazo. Cuanto mayor sea el factor de descuento (más próximo a 1), mayor es el peso que tienen las recompensas futuras sobre la decisión actual.

Existen dos tipos de políticas: las estacionarias y las no estacionarias. Una política estacionaria $(a = \pi(s))$ especifica directamente, para cada estado, la acción a realizar independientemente del momento o tiempo en que se encuentra el proceso. Este tipo de políticas se utiliza principalmente con procesos de horizonte-infinito, puesto que al no existir un límite de pasos, no existe ningún motivo para que el agente cambie su estrategia al elegir las acciones. Una política no estacionaria $(a = \pi_t(s))$, sin embargo, asigna una acción u otra al mismo estado en función del momento en que se encuentra

el sistema. Este tipo de políticas se utiliza en procesos de horizonte finito, puesto que es conveniente modificar la estrategia a seguir en función del número de pasos que quedan para finalizar el proceso. Así, la política $a = \pi_t(s)$ permite seleccionar la acción a ejecutar en el estado *s* cuando quedan *t* pasos para finalizar el proceso.

Para un mismo MDP pueden definirse múltiples políticas. Sin embargo, una política será tanto mejor cuanto mayor sea la recompensa que obtiene a largo plazo, siendo éste el criterio que permite seleccionar entre todas ellas, una política óptima.

Desafortunadamente, las ecuaciones (3.1) y (3.2), que representan la suma de todas las recompensas futuras obtenidas por el agente, son meramente conceptuales, puesto que resulta imposible predecir las recompensas futuras en un sistema cuyo resultado de las acciones es estocástico, y en el que por lo tanto la evolución de su estado y las recompensas recibidas en el futuro no pueden conocerse *a priori*.

En lugar de ello, y como criterio para comparar diferentes políticas, se utiliza una función auxiliar conocida como función de valor $(V_\pi(s))$, que para una determinada política asigna a cada estado, un valor numérico. Una política es tanto mejor cuanto mayor sea su función de valor sobre los estados.

3.3.3. Función de valor

La función de valor $V_\pi(s)$ determina la utilidad de cada estado *s* suponiendo que las acciones se escogen según la política π. Se trata de un concepto distinto al de recompensa. La función de recompensa asigna, para cada una de las acciones que pueden ejecutarse en cada estado, un valor numérico

que representa la utilidad inmediata de dicha acción. Sin embargo, la función de valor asigna a cada estado un valor numérico que representa la utilidad de dicho estado a largo plazo. Este valor numérico no es la suma exacta de recompensas futuras, que no se puede predecir, sino una aproximación probabilística que se calcula a partir de las funciones de transición ϕ y de recompensa R. Por ejemplo, un agente puede realizar en cierto estado una acción con recompensa positiva, que sin embargo le lleve a un estado que tenga un valor de utilidad negativo. Desde este punto de vista, es mucho más benéfico realizar la acción que prometa una mejor recompensa a largo plazo.

Supóngase una política π en un contexto de horizonte finito con k pasos. La función de valor asociada a esta política, $V_{\pi,k}(s)$ es la recompensa total "esperada" (no real) al ejecutar la política π empezando en el estado s durante k pasos. Obviamente, si *k=1*, $V_{\pi,1}(s) = r(s,\pi_1(s))$; es decir, si sólo se dispone de un paso o el proceso se encuentra en el último paso, la función de valor coincide con la recompensa obtenida al ejecutar la acción dada por la política. De esta manera, para cualquier otro horizonte, la función de valor puede calcularse recursivamente en retroceso según la ecuación (3.3), (Ocaña M., 2005):

$$V_{\pi,t}(s) = r(s,\pi_t(s)) + \gamma \sum_{s' \in S} p(s' \mid s,\pi_t(s)) \cdot V_{\pi,t-1}(s') \qquad (3.3)$$

Por consiguiente, el valor de horizonte t (es decir, cuando quedan t pasos para finalizar el proceso) para el estado s se calcula sumando a la recompensa inmediata $r(s,\pi_t(s))$ el valor esperado en los restantes *t-1* pasos del proceso ponderado por el factor de descuento. Este valor esperado en los pasos restantes se calcula sumando, para todos los posibles estados de

destino a los que puede pasar el proceso en el siguiente paso, el producto de su valor de horizonte *t-1* por la función de transición de estados.

En el contexto de horizonte-infinito con factor de descuento, la función de valor de cada estado sólo depende de la política y no del número de pasos futuros, y viene dada por la ecuación (3.4):

$$V_\pi(s) = r(s,\pi(s)) + \gamma \sum_{s' \in S} p(s' | s,\pi(s)) \cdot V_\pi(s') \qquad (3.4)$$

En este caso, planteando la ecuación (3.4) para cada uno de los estados que componen el MDP, se obtiene un sistema lineal de ecuaciones, siendo la función de valor V_π la solución del mismo.

3.3.4 Políticas óptimas

Resolver un MDP consiste en encontrar la política óptima, una directiva de control que maximiza la función de valor sobre los estados. Teóricamente, es posible obtener todas las posibles políticas para un MDP y a continuación escoger entre ellas, aquella que maximiza la función de valor. Sin embargo, este método es computacionalmente costoso, puesto que el número de políticas crece exponencialmente con el número de estados. Existen, sin embargo, otros métodos que permiten seleccionar la política óptima aprovechando la propiedad de que ésta será también localmente óptima para cada estado individual.

Howard (1960) demostró que, para el caso más general de horizonte-infinito, existe una política estacionaria $\pi*$ que es óptima para cualquier estado inicial del proceso. La función de valor para esta política viene dada por la solución de la ecuación de Bellman (3.5):

$$V^*(s) = \max_a \left(r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) \cdot V^*(s') \right), \forall s \in S \qquad (3.5)$$

Dada la función de valor óptima, la política óptima responde a la ecuación (6), donde $\arg\max_a(f(a))$ significa el valor de *a* que maximiza *f(a)*.

$$\pi^*(s) = \arg\max_a \left(r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) \cdot V^*(s') \right) \qquad (3.6)$$

El método tradicional para obtener políticas óptimas es la programación dinámica (Bellman 1957, Bertsekas 1995). Dos de los métodos más utilizados son el de Iteración de Valor y el de Iteración de Política. Ambos se basan en modificar las utilidades de los estados vecinos de manera que satisfagan las ecuaciones de Bellman. La repetición de este proceso de modificación local en cada uno de los estados durante un número suficiente de iteraciones hace converger las utilidades de los estados individuales hacia sus valores correctos. Estos métodos permiten obtener la política óptima fuera de línea, esto es, ayudan a que el agente no invierta tiempo en aprender la política probándola en tiempo de ejecución.

Los algoritmos 1 y 2 resumen respectivamente, los métodos de iteración de política e iteración de valor. El primero consiste en calcular la utilidad de cada uno de los estados y con base en estas utilidades, seleccionar una acción óptima para cada uno de ellos. El segundo funciona escogiendo una política, y luego calculando la utilidad de cada estado con base en dicha política. Luego actualiza la política correspondiente a cada estado utilizando las utilidades de los estados sucesores, lo que se repite hasta que se estabiliza la política.

Algoritmo 1. Iteración de política

5. Datos de entrada

   pol_estable – tabla para la política óptima

   b – tabla de una política inicial

   θ _ número positivo pequeño

6. Inicialización:

   $V(s) \in \Re$ y $\pi(s) \in A(s)$ arbitrariamente $\forall s \in S$

7. Evaluación de política:

   Repetir

   $\Delta \leftarrow 0$

   Para cada $s \in S$

   $v \leftarrow V(s)$

   $V(s) \leftarrow r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s' \mid s, \pi(s)) \cdot V(s')$

   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   Hasta que $\Delta < \theta$

8. Mejora de política

   $pol\_estable \leftarrow verdadero$

   Para cada $s \in S$

   $b \leftarrow \pi(s)$

   $\pi(s) \leftarrow \arg\max_a \left(r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s, a) \cdot V(s') \right)$

   Si $b \neq \pi$ entonces $pol\_estable \leftarrow falso$

   Si $pol\_estable$ entonces para, sino ir a 2

Algoritmo 2. Iteración de valor

5. Datos de entrada

   pol_estable – tabla para la política óptima

   b – tabla de una política inicial

   θ _ número positivo pequeño

6. Inicialización:

   $V(s) = 0 \quad \forall s \in S$

7. Repetir

   $\Delta \leftarrow 0$

   Para cada $s \in S$

   $v \leftarrow V(s)$

   $V(s) \leftarrow r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s' | s, \pi(s)) \cdot V(s')$

   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   Hasta que $\Delta < \theta$

8. Regresar una política determinística que satisfaga:

$$\pi(s) \leftarrow \arg\max_a \left(r(s,a) + \gamma \sum_{s' \in S} p(s' | s, a) \cdot V(s') \right)$$

Hasta aquí hemos discutido todo lo concerniente a MDPs, para dar un panorama amplio acerca del formalismo que existe detrás de los mismos.

3.4 Aprendizaje por refuerzo

El aprendizaje automático propone métodos específicos que permiten a los robots autónomos aprender de su interacción con el entorno, y además, aprender mientras se encuentran inmersos en dicho entorno.

En algunos ambientes, muchas veces se puede obtener sólo cierta retroalimentación, recompensa o refuerzo, e. g. valores de ganancia o perdida. El refuerzo puede darse en un estado terminal y/o en estados intermedios. Los refuerzos pueden ser componentes o sugerencias de la utilidad actual a maximizar, e. g. buen movimiento. En aprendizaje por refuerzo (RL, *Reinforcement Learning*) el objetivo es aprender cómo mapear situaciones a acciones para maximizar una cierta señal de recompensa.

El aprendizaje por refuerzo es el problema de conseguir que un agente actúe en un entorno de manera que maximice la recompensa que obtiene por sus acciones [URL_Fleifel].

En nuestro trabajo, el *agente* es un robot hexápodo, al hablar de *entorno* nos referimos al ambiente o espacio en el cual estará funcionando nuestro robot y la *recompensa* es un valor escalar que indicará lo deseable que es una situación para el robot. La recompensa puede tomar valores tanto positivos como negativos. Fisiológicamente, podría compararse un valor de recompensa negativo con el dolor y un valor positivo con el placer.

Cada vez que el robot ejecuta una acción, recibe un valor de recompensa. Estas recompensas no tienen por qué estar asociadas directamente con la última acción ejecutada, sino que pueden ser consecuencia de acciones anteriores llevadas a cabo por el robot.

No es sencillo establecer una correspondencia directa entre acciones y recompensas obtenidas. Es posible que la misma acción llevada a cabo en dos momentos diferentes devuelva recompensas distintas debido a la secuencia previa de acciones.

La promesa del RL es que se refina el programa de control de los agentes mediante premio y castigo sin necesidad de especificar cómo realizar una

tarea. A diferencia a otro tipo de aprendizaje, e. g. aprendizaje supervisado, en RL:

- No se le presentan al agente pares entrada - salida.
- El agente tiene que obtener experiencia útil acerca de los estados, acciones, transiciones y recompensas de manera activa para poder actuar de manera óptima.
- La evaluación del sistema ocurre en forma concurrente con el aprendizaje.

En RL un agente trata de aprender un comportamiento mediante interacciones de prueba y error en un ambiente dinámico e incierto. En general, al sistema no se le dice qué acción debe tomar, sino que él debe de descubrir qué acciones producen el máximo beneficio.

El robot y el entorno interactúan en una secuencia de instantes de tiempo $t = 0, 1, 2, 3, \ldots$ En cada instante de tiempo t, el robot identifica una representación del estado del entorno $s \in S$, donde S es el conjunto de posibles estados. Con base en esto, el robot selecciona una acción $a_t \in A(s_t)$, donde $A(s_t)$ es el conjunto de acciones disponibles en el estado s_t. En el instante de tiempo posterior, y en parte como consecuencia de la acción llevada a cabo, el robot recibe una recompensa numérica, $r_{t+1} \in \Re$, y pasa a un nuevo estado, s_{t+1}.

En cada momento de tiempo, el robot lleva a cabo un mapeo entre las representaciones de los estados y las probabilidades de seleccionar cada una de las acciones posibles. Llamamos a este mapeo la ***política*** del robot, y la denotamos por π_t, donde $\pi_t(s,a)$ es la probabilidad de que $a_t = a$ si $s_t = s$. Los distintos métodos de aprendizaje por refuerzo especifican de qué manera

cambia el robot su política como resultado de la experiencia que va adquiriendo de su interacción con el entorno. Como ya se mencionó, el objetivo del robot es maximizar a largo plazo la suma de las recompensas obtenidas. Este proceso se ilustra en la figura 3.1.

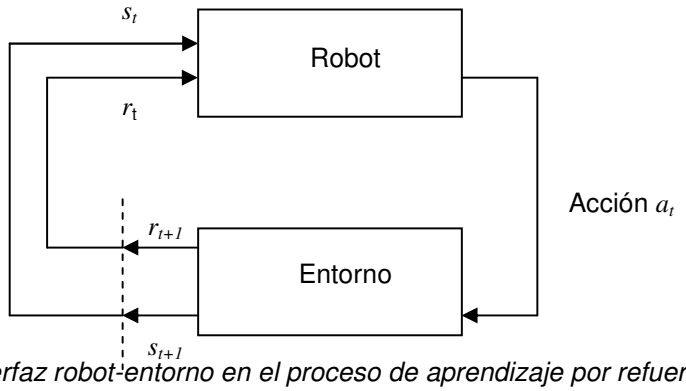


Figura 3.1 Interfaz robot-entorno en el proceso de aprendizaje por refuerzo

Como menciona uno de los proponentes del RL, Sutton & Barto (1998), lo mejor del marco de trabajo antes descrito es que es extremadamente flexible, lo que permite que sea aplicado a muchos problemas diferentes de maneras muy distintas. Hay que decir que dentro de este marco de aprendizaje las acciones pueden ser tanto controles directos, e. g. los voltajes aplicados a los motores del brazo de un robot, como decisiones de mayor nivel, e. g. escoger entre la ruta 1 y la ruta 2. De manera similar, los estados pueden obtenerse de una gran variedad de formas. Pueden estar completamente determinados por percepciones de bajo nivel, como las lecturas directas sobre los sensores, o pueden ser más abstractas y de mayor nivel, como las descripciones simbólicas de los objetos de una habitación, tal y como se pretendía en los inicios de la robótica [URL_Shakey].

También hay gran flexibilidad en la manera de diseñar los estados. Un estado puede consistir en las percepciones directas del robot recibidas en un momento dado, o puede estar compuesto en parte por datos almacenados

de percepciones anteriores. Incluso pueden existir estados completamente abstractos.

En general, el ambiente es no-determinístico, esto es, tomar la misma acción en el mismo estado puede dar resultados diferentes. Sin embargo, en los MDPs se asume que el ambiente es estacionario, esto es, las probabilidades de cambio de estado no cambian o cambian muy lentamente. Algunos aspectos importantes relacionados con lo anterior son:

- Se sigue un proceso de prueba y error, y
- la recompensa puede estar diferida en el tiempo

Otro aspecto importante es el balance entre exploración y explotación. El agente debe balancear la **exploración** de nuevos estados y acciones para obtener nueva información que le permita evitar óptimos locales, y la **explotación** de estados y acciones ya aprendidos y con una alta recompensa inmediata que le garantice una recompensa acumulada aceptable.

La caracterización de esta problemática está dada por Procesos de Decisión de Markov o MDP. Esto es, usando MDPs en donde se tiene un modelo, la exploración se realiza fuera de línea, i. e. no en tiempo de ejecución del robot, y exhaustivamente para obtener las mejores acciones.

Los métodos de aprendizaje aplicados en robótica pueden verse como una forma de aprender una correspondencia entre las entradas y salidas para maximizar ciertas medidas de desempeño, mientras el sistema se encuentra en operación. El aprendizaje por refuerzo no resulta tan eficaz en ambientes altamente dinámicos. En realidad esto es un problema para cualquier método de aprendizaje en línea y no supervisado. En el área de la robótica este problema es considerable debido a que un cambio en el entorno que los

seres humanos podemos considerar no significativo, puede ser un cambio significativo para un robot. Debe tenerse en cuenta que la capacidad de procesamiento de los robots actuales se encuentra muy limitada, y que sus capacidades perceptivas son mínimas comparadas con las de los seres vivos. Por ello, un ligero cambio en las condiciones del entorno puede desconcertar e impactar a los robots de forma importante.

Algunos de los diferentes problemas en robótica para los cuales se está empleando el aprendizaje por refuerzo son los siguientes:

- Aprendizaje de secuencias de movimientos y optimización de recursos: (Vargas et al. 2002), Kimura (1997), Kimura (1999), DeJong (1994), Boone (1997), Tham (1994).
- Coordinación de sistemas multi - agente: Uchibe (1995), Uchibe (1996a), Uchibe (1996b).
- Navegación: Mataric (1994).
- Aprendizaje distribuido y colectivo, e. g. aprendizaje por imitación, aprendizaje de la comunicación: Billard (1997), Hayes (1994), Demiris (1996).
- Aprendizaje de comportamientos: Asada (1996).
- Tareas más complejas / aprendizaje por refuerzo deliberativo: Dietterich (2000).
- Control: Schaal (1994), Mahadevan (1991).

3.5 Procesos de Decisión de Markov Jerárquicos

Los algoritmos clásicos para modelar y solucionar MDPs requieren la representación explícita de estados y de acciones. Muchos de los algoritmos clásicos de solución, como iteración de valor e iteración de política, necesitan explorar el espacio entero de estados durante cada iteración. Sin embargo, en problemas de aprendizaje que implican un enorme número de estados

explícitos y quizá de acciones también, esta exploración se vuelve exponencial cuando el número de variables en el sistema aumenta. Por ejemplo, supongamos que un robot es requerido para entregar el correo y café para los profesores en un laboratorio. Este sistema puede incluir características del estado, por ejemplo, si un profesor desea o no el café, si hay o no correo para él, qué tan lejos se encuentra la oficina de cada profesor del cuarto de café y de correo, y así sucesivamente. Si deseamos enumerar todos los estados explícitos, crecerá exponencialmente el número de estados de acuerdo al número de características que se deseen incluir.

Así, si utilizamos un enfoque clásico para tratar MDPs grandes, podemos tener dificultades de procesamiento debido al tamaño del espacio de estados. Es por esto que utilizamos un enfoque jerárquico para resolver MDPs grandes, algunos trabajos relacionados podemos mencionar (Parr & Russell 1997), (Gu 2003), (Laroche 2000), (Bakker et al. 2005).

Regresando al ejemplo anterior, entregar el café a ciertas oficinas o la recolección del café en cierta sala requiere lapsos diversos de tiempo, y es natural que uno planee o aprenda cómo resolver dichas situaciones en diferentes instantes de tiempo, lo cual se conoce como abstracción temporal. Existen varios enfoques ya establecidos para realizar la abstracción temporal y descomposición de la tarea, los cuales discutiremos en las siguientes secciones.

La problemática que se plantea con el ejemplo anterior, es factible de ser resuelta con un solo MDP, pero como se discutió, se tendría un modelo con un gran número de estados y probablemente de acciones, así que como también se adelantó, es preferible hacer una abstracción temporal del modelo original del MDP. A esta abstracción se le conoce como *Proceso de*

Decisión de Markov Jerárquico (*HMDP*, *Hierarchical Markov Decision Process*).

En este trabajo se aplicará un HMDP para simplificar la complejidad con la que se resolvería el problema usando solamente un MDP.

Es conveniente señalar que la concepción de un HMDP no consiste únicamente en agrupar partes de un MDP muy grande, sino que implica también un análisis sobre la manera de descentralizar el problema. Además, no existen guías de diseño de HMDPs, la concepción de diseño de un HMDP es un problema abierto.

A continuación describimos algunas de las formas de concepción de un HMDP.

3.5.1 Opciones

Sutton et al. (1999) extendieron la noción usual que se tiene de las acciones que componen un MDP, creando así una estructura modificada de MDP que incluye *opciones,* las cuales son políticas de ciclo cerrado para tomar determinadas acciones en un periodo de tiempo determinado.

Formalmente, dado un MDP $M = \langle S, A, \Phi, R \rangle$, una opción consiste de tres componentes: una política no determinista $\pi : SxA \to [0,1]$, una condición de paro $\beta : S \to [0,1]$ y un conjunto de iniciación $I \subseteq S$.

Una opción está disponible en un estado s_t en el tiempo t si y solo si $s_t \in I$. El conjunto de iniciación y la condición de paro de una opción restringen conjuntamente el rango de aplicación de dicha opción. Si la opción

$Op =< I, \pi, \beta >$ es tomada, entonces las acciones son seleccionadas de acuerdo a la política π hasta que la opción termine estocásticamente de acuerdo a β. En particular, una *Opción de Markov* se ejecuta como sigue. Primero, en el tiempo t, la acción a_t es seleccionada de acuerdo a una distribución de probabilidades $\pi(s_t, \cdot)$. Después de ejecutar la acción seleccionada el ambiente cambia, a esto se le conoce como la transición al estado s_{t+1}, donde la opción termina con probabilidad $\beta(s_{t+1})$, o también el agente puede continuar tomando la acción a_{t+1} de acuerdo a la política $\pi(s_{t+1}, \cdot)$ y posiblemente termine en el siguiente paso s_{t+2} de acuerdo a $\beta(s_{t+2})$ y así sucesivamente. El agente puede seleccionar otra opción cuando termine de ejecutar la acción que está ejecutando.

Sutton et al. (1999) también proponen que un MDP con un conjunto de opciones es un *Proceso de Decisión Semi-Markoviano (SMDP, Semi-Markov Decision Process),* cuyas transiciones de estados son estocásticas y la duración de cuyas opciones es estocástica pero no depende del tiempo.

Resultados empíricos en (Sutton et al. 1999) muestran que problemas de planeación y aprendizaje pueden ser resueltos mucho más rápidamente introduciendo y rehusando opciones apropiadamente. Esto se debe a que teniendo opciones adecuadas, las cuales pueden ser ya óptimas en algunos estados, se puede converger a una política óptima global más rápidamente. Además, Sutton et al. (1999) mostró que la estructura de las opciones puede ser aprendida, usada e interpretada mecánicamente.

3.5.2 Macro-acciones

Las macro-acciones son descritas y usadas para resolver MDPs de manera jerárquica en (Hauskrecht et al. 1998). En esencia, las macro-acciones son lo

mismo que las opciones. Sin embargo, este enfoque de macro-acciones es ligeramente diferente de las opciones a partir de la descripción dada por Sutton et al. (1999). En el trabajo de Sutton, $p(s,o,s')$ y $r(s,o)$ son definidas para todos los estados en el espacio de estados. Resolver el problema en este contexto requiere del uso explícito de la programación dinámica sobre todo el espacio de estados, lo cual no reduce el tamaño del espacio de estados.

Una macro-acción es vista como una política local sobre una región específica del espacio de estados, la cual termina de aplicarse cuando la región es dejada. Un *MDP abstracto* es construido a partir sólo del estado que se encuentre en los límites de regiones adyacentes y su espacio de acciones sólo consiste de macro-acciones. Un MDP jerárquico con macro-acciones generalmente tiene un espacio de estados y de acciones mucho más pequeños que los del MDP original. De esta manera, se simplifica la complejidad de resolver un MDP con espacio de acciones y estados grandes.

Hauskrecht et al. (1998) basó su modelo en una *descomposición basada en regiones* de un MDP determinado $M = \langle S, A, \Phi, R \rangle$. Como se define en (Dean & Lin 1995), S es dividido en regiones S_1, S_2 , ... S_N . Los estados que se encuentran en los límites de las regiones adyacentes incluyen dos tipos de estados para cada región S_i, *estados periféricos de salida de* S_i y *estados periféricos de entrada de* S_i.

Una macro-acción para una región S_i es una política $\pi_i : S_i \to A$. La terminación de una macro-acción es mucho más específica que la de una opción: la condición de inicio para π_i en el tiempo *t* es $s_t \in S_i$ y la condición de terminación en el tiempo *t* es $s_t \notin S_i$.

Hauskrecht et al. (1998) argumenta que la solución de un *MDP aumentado*, un MDP original extendido con un conjunto de macro-acciones, tiene garantía de ser una solución óptima, aunque no se puede asegurar un beneficio en términos computacionales. Realmente Hauskrecht et al. (1998) está interesado en reducir la abstracción del MDP obtenida reemplazando el conjunto de acciones primitivas con un conjunto de macro-acciones y reconstruyendo el modelo en los estados marginales.

Ventajas computacionales significativas pueden obtenerse debido a que un MDP simplificado o abstracto normalmente tiene un espacio de estados sustancialmente más pequeño que el del MDP original. Sin embargo, la política del MDP abstracto puede corresponder únicamente a una política sub-óptima del MDP original (Hauskrecht et al. 1998).

Varios trabajos proponen métodos para asegurar que las políticas para un MDP aumentado sean lo más aproximadas posible a la política óptima del MDP original (Parr 1998).

3.5.3 Conjuntos Markovianos de tareas

El enfoque de opciones descrito anteriormente puede ser considerado como una técnica de la familia de enfoques de descomposición para resolver MDPs grandes. Descomponer un MDP significa que un MDP es definido en términos de un conjunto de subprocesos o tareas “pseudo-independientes” (Singh & Cohn 1998) o automáticamente descompuesto en tales subprocesos. Esto es, se usa un algoritmo para realizar dicha descomposición (Boutilier et al. 1998). Las soluciones de esos sub-MDPs son usadas para construir una solución global aproximada. En el enfoque de opciones, el espacio de estados del MDP original es dividido en regiones

para formar sub-MDPs. Sin embargo, hay otra clase de técnicas las cuales tratan a los sub-MDPs como procesos concurrentes (Meuleau et al. 1998).

Meuleau et al. (1998) asume que los MDPs aplicados en problemas de asignación de recursos secuenciales estocásticos están ampliamente acoplados, i. e. los problemas están compuestos de múltiples tareas cuyos valores de utilidad son independientemente aditivos, de modo que las acciones tomadas con respecto a una tarea realizada no afectan el estado de alguna otra tarea. Estos problemas de asignación son modelados con una forma especial de MDPs, *conjuntos Markovianos de Tareas* (MTS, *Markov Sets of Tasks*) para n tareas. Un conjunto Markoviano de tareas es una tupla $\langle S, A, \Phi, R, c, M_g, M_l \rangle$. *S, A, R,* y Φ están definidos como en un MDP normal, c es el costo de solo una unidad del recurso a asignar, M_g la restricción de recurso global sobre la cantidad del recurso total y M_l la restricción de recurso local sobre la cantidad del recurso que puede ser usado en un solo paso. Bajo esta estructura, una política óptima es un vector de políticas locales que maximiza la suma de las recompensas asociadas con cada tarea. Estos MDPs son muy grandes e involucran cientos de tareas, y son útiles en problemas de asignación de recursos secuenciales estocásticos.

La estrategia de aproximación, llamada descomposición de tareas de Markov, se divide en dos fases. Una primera fase “fuera de línea” en la cual las funciones de valor y soluciones óptimas son calculadas para las tareas individualmente, usando programación dinámica. En la segunda fase “en línea”, esos valores son usados para calcular un gradiente para una búsqueda heurística, para calcular la próxima acción como una función del estado actual de todos los procesos. Resultados experimentales demuestran que esta técnica puede resolver MDPs con un número de estados y acciones grande y reducir el tiempo computacional de su proceso.

3.5.4 Máquinas abstractas jerárquicas

El aprendizaje basado en Máquinas Abstractas Jerárquicas (HAMs, *Hierarchical Abstract Machines*), fue propuesto por Parr & Russell (1997). Es otro enfoque de abstracción temporal usado con MDPs con un gran número de estados. De manera similar a las opciones, el uso de HAMs reduce el espacio de búsqueda y proporciona una estructura en la cual el conocimiento puede ser transferido a través de los subproblemas y los componentes de solución pueden ser recombinados para resolver el problema original. En contraste con las opciones, las cuales como ya se ha mencionado pueden ser vistas como políticas locales, las HAMs usan controladores de estado finito no determinísticos para expresar conocimiento *a priori* y así restringir las posibles políticas que el agente puede escoger.

Como se define en (Parr & Russell 1997) una HAM está compuesta de un conjunto de máquinas de estados, una función de transición y una función de inicio que determina el estado inicial de la máquina. Las máquinas de estado son de cuatro tipos: un *estado acción* ejecuta una acción, un *estado llamada* ejecuta otra máquina como subrutina, un *estado selección* selecciona de forma no determinista la siguiente máquina de estados y un *estado de paro* de la ejecución regresa el control al estado de llamada previo. La función de transición determina estocásticamente el resultado de la siguiente máquina de estados, de acuerdo al estado de la máquina de estados actual del tipo de acción o llamada y del ambiente resultante. Una HAM es en efecto un programa el cual, cuando es ejecutado por un agente en un ambiente, restringe las acciones que el agente puede tomar en cada estado.

Resultados experimentales de solución de un problema ilustrativo con 3600 estados usando una HAM, (Parr & Russell 1997), demostraron mejoras

drásticas sobre el algoritmo tradicional de iteración de política aplicado al MDP original sin la HAM. Parr y Russell también aplicaron este enfoque en aprendizaje por refuerzo, llamado HAMQ-learning, modificando la función *Q-learning* con el modelo MDP inducido por la HAM, y éste *HAMQ-learning* parece aprender mucho más rápido que el tradicional *Q-learning*.

Andre y Russell propusieron una lenguaje de programación para HAM (PHAM) (Andre & Russell 2001) el cual es un lenguaje expresivo de diseño del agente que extiende el lenguaje básico del lenguaje de programación HAM de Parr & Russell (1997) con las características del LISP.

3.5.5 Método MAXQ

Dietterich (1998, 2000a, 2000b) propuso un enfoque llamado MAXQ para tratar con problemas de toma de decisiones y aprendizaje por refuerzo grandes, basado en los trabajos de Singh (1992), Kaelbling (1993), Dayan & Hinton (1993) y Dean & Lin (1995). Basado en la hipótesis de que el programador puede identificar submetas y subtareas que ayuden a alcanzar esas metas, MAXQ descompone un MDP original en una jerarquía de MDPs más pequeños y al mismo tiempo descompone la función de valor del MDP original en una combinación aditiva de funciones de valor de los MDPs más pequeños.

Comparado con opciones y macro-acciones, en las cuales las tareas son definidas en términos de políticas locales fijas, y con las HAMs en las cuales las subtareas son definidas usando controladores basados en máquinas de estados no deterministas, MAXQ está orientado a metas y cada subtarea es definida usando predicados de terminación y una función de recompensa local. Dado un MDP $M = \langle S, A, \Phi, R \rangle$ y suponiendo que las políticas son mapeadas desde el espacio de estados al espacio de acciones, MAXQ

descompone a M en un conjunto finito de subtareas $\{M_0, M_1,..., M_n\}$ con la convención de que M_0 es la tarea raíz y resolver M_0 resuelve el MDP completo M.

Una política jerárquica π para $\{M_0, M_1,..., M_n\}$ es un conjunto el cual contiene una política para cada una de las subtareas $\pi = \{\pi_0, \pi_1,..., \pi_n\}$. Cada política de la subtarea π_i selecciona un estado y regresa una acción primitiva para ejecutarla en el estado en el que se encuentra, o el índice de una subrutina para poder invocarla.

Al igual que las opciones y las HAMs, MAXQ también usa una versión resumida del método iteración de valor para optimizar la aproximación de la selección de tarea.

Dietterich (2000) argumenta que dada una descomposición MAXQ, al igual que las opciones o las HAMs, hay un algoritmo para obtener la política óptima jerárquica que alcanza la recompensa prevista más alta entre todas las políticas que conforman la descomposición.

Finalmente, Dietterich (2000) afirma que usando técnicas de abstracción de estados, el método MAXQ no sólo representará un MDP grande de manera compacta, también obtendrá ventajas computacionales en procesar la función de valor descompuesta.

3.5.6 Aprendizaje por refuerzo jerárquico

El aprendizaje por refuerzo jerárquico (*HRL, Hierarchical Reinforcement Learning*) es un enfoque emergente en el cual los métodos de aprendizaje por refuerzo son aumentados con conocimiento previo con respecto a la estructura de comportamiento de alto nivel. La idea fundamental de este

enfoque es que el conocimiento previo debe acelerar considerablemente la búsqueda de una política óptima. En general, todos los formalismos de HRL son vistos como la adición de restricciones al aprendizaje por refuerzo, dichas restricciones limitan o fijan el comportamiento del agente, es decir, hacen que el agente siga directivas previamente establecidas en lugar de aprenderlas. Algunos trabajos en los cuales se aplica este enfoque los encontramos en (Sherstov & Stone 2005), (Marthi et al. 2005).

Lo antes descrito es aplicado principalmente en aprendizaje por refuerzo pero la idea básica puede ser aplicada en la construcción de un MDP. Así, usando este enfoque se crearía un modelo de dos niveles, uno de alto nivel y otro de bajo nivel, el MDP o MDPs de bajo nivel son pequeños en número de estados y se encargan de resolver problemas relativamente simples. El MDP o MDPs de alto nivel es básicamente una abstracción de la tarea deseada, esto es, una especie de secuencia de pasos que debe seguir un agente para alcanzar un objetivo. A esto se le puede considerar como el conocimiento previo que se le agrega a la construcción del modelo. Como se mencionó, este conocimiento restringe el comportamiento de alto nivel del agente. De esta manera, al restringir explícitamente el comportamiento que debe tener el agente para alcanzar una meta, se acota el espacio de búsqueda lo cual redunda en un problema computacionalmente más sencillo, pero no permite que el agente aprenda comportamientos distintos de los que están establecidos.

Cabe mencionar que no solamente se existe el enfoque de MDPs jerárquicos para resolver MDPs grandes, existen otros enfoques entre los cuales podemos mencionar factorización de MDPs (Degris T. et al., 2006), (Reyes A. et al., 2006).

3.6 Discusión

Una tarea de aprendizaje por refuerzo que satisface la propiedad de Markov, es llamada Proceso de Decisión de Markov (MDP).

Como se ha discutido en este capítulo, realmente no existe una definición formal de lo que es un HMDP, pero podemos definirlo de la siguiente manera: Un MDP jerárquico o HMDP es una abstracción temporal o espacial de un MDP complejo. Un HMDP sintetiza una tarea compleja dividida en pequeñas subtareas, las cuales son resueltas de manera independiente. La solución de estas subtareas contribuye a obtener la solución global. Esta división sirve para simplificar los cálculos y acotar el espacio de estados de un MDP muy complejo. Como se ha discutido, existen varias técnicas para tratar con MDPs grandes. Y como se ha descrito, dicha descomposición básicamente consiste en dividir el MDP original en subtareas.

Después de presentar los métodos más conocidos para descomponer MDPs, podemos comparar las ventajas y desventajas de cada uno. La principal ventaja de usar opciones y macro-acciones es que los modelos y propiedades de las opciones o macro-acciones pueden ser fácilmente establecidos en regiones locales, las cuales generalmente tienen un espacio de estados pequeño con respecto al del MDP original y que dichas opciones pueden ser rehusadas posteriormente. La desventaja de usar opciones es que esas políticas locales son fijas. Así, para obtener una política aproximada a la óptima aceptable para el MDP original, tenemos que definir conjuntos convenientes de macros u opciones para cada región, lo cual puede ser difícil, dado que no existe una guía a seguir de la manera adecuada de realizar esa definición de conjuntos.

Por otro lado, HAMs y MAXQ no son fijadas *a priori*. En MAXQ se definen subtareas en términos de predicados de terminación, lo cual requiere que el programador "adivine" el beneficio relativo de los diferentes estados en los cuales la subtarea puede terminar. Esto también puede resultar complicado de realizar, dado que dependerá en gran medida de la experiencia del diseñador. La ventaja de MAXQ es que este método descompone jerárquicamente el espacio de estados del MDP original en subtareas de múltiples capas, y las políticas aprendidas en los niveles inferiores de las subtareas pueden ser compartidas por múltiples tareas padre o del nivel superior. Sin embargo, en MAXQ se usa una técnica de abstracción de estados para eliminar aspectos irrelevantes en el espacio de estados de la subtarea y se descompone la función de valor del problema original en un conjunto de sub-funciones para los subproblemas, de esta manera este método puede resultar menos costoso, computacionalmente hablando, con respecto a los otros. Las HAMs restringen el gran número de posibles políticas a ser consideradas para obtener la óptima o las políticas cercanas a la óptima usando controladores no deterministas, para que así el método pueda alcanzar ventajas computacionales. Pero para calcular una política óptima que es HAM-consistente, necesitamos construir un MDP el cual tiene un enorme espacio de estados. Así que el Método MAXQ puede ser preferible que las HAMS debido a la forma de procesar las políticas.

Finalmente, podemos aplicar la idea de agregar conocimiento previo en la construcción del modelo, pero como ya se mencionó esto restringe al agente para encontrar comportamientos distintos a los que previamente se le han dado a conocer. La ventaja de este enfoque es que la obtención de la política es rápida, y que el diseño del modelo es sencillo.

Los métodos anteriores están enfocados a descomposiciones de MDPs para obtener HMDPs, y por lo general han sido aplicados en problemas de

navegación robótica. Cada una de estas técnicas hace divisiones jerárquicas en capas del MDP original. En los trabajos que usan HMDP, la descomposición de las tareas se lleva en un nivel de abstracción alto, por ejemplo, evitar obstáculos, caminar, reconocer rostros, preparar café, entregar correo, etc. (Gu 2003). Cada una de esas tareas por separado no tiene nada que ver con las otras, pues cada una de ellas tiene un propósito específico, es por eso que se hace una jerarquización de las tareas dando prioridades a aquellas que tienen más importancia e inhibiendo las de menor importancia. Por otro lado, no existen trabajos, hasta donde sabemos, en los cuales se definan y se usen MDPs de manera distribuida y no jerárquica, la principal diferencia entre los MDPs jerárquicos y los descentralizados, es que el caso de los jerárquicos generalmente se tiene un MDP para cumplir una tarea específica, en cambio en los descentralizados como veremos existen más de un MDP que tratan, de manera independiente, de cumplir con la tarea.

CAPÍTULO IV: Modelos Propuestos

Después de revisar la teoría de MDPs y HMDPs podemos afirmar que el uso de MDPs usando un enfoque descentralizado es factible cuando el objetivo total del sistema es uno y sólo uno, o a lo más dos como se discutirá en capítulos posteriores, ya que los subsistemas que conformen al sistema total trabajarán independientemente buscando el mismo objetivo. Por otro lado, cuando se tienen problemas los cuales plantean la realización de múltiples objetivos y para los cuales quiere usarse un MDP para su solución el mejor enfoque es el jerárquico.

En este capítulo se describen los modelos propuestos que más adelante se aplican en experimentos tanto en simulación como en experimentos llevados a cabo con un robot hexápodo físico comercial.

4.1 Modelo 1: MDP con Decisión Centralizada y Acción Descentralizada (DCAD)

4.1.1 Análisis y justificación

Como ya se ha mencionado, este trabajo se enfoca en crear un sistema de control que le permita a un hexápodo con una morfología circular desplazarse de manera estable. Para esto se realizó un análisis del paso que utilizan ciertas arañas con una distribución circular de patas alrededor de su cuerpo, como las que se muestran en la figura 4.1.

Este tipo de arañas utiliza un paso distinto al paso de trípode ya ampliamente estudiado y aplicado a robots hexápodos con morfología rectangular (c. f sección 2.3).

Figura 4.1 Arañas con distribución circular de patas, donde los círculos muestran dicha distribución

El paso que usa este tipo de arañas conserva la distribución circular que tienen sus patas después de completar un determinado ciclo, esto lo consiguen moviendo determinados segmentos de sus patas, esto es, manipulan determinados segmentos de una pata dependiendo de la posición que ocupa ésta alrededor del cuerpo de la araña. La figura 4.2 muestra la distribución y división de las patas de una araña con una distribución de patas circular alrededor de su cuerpo.

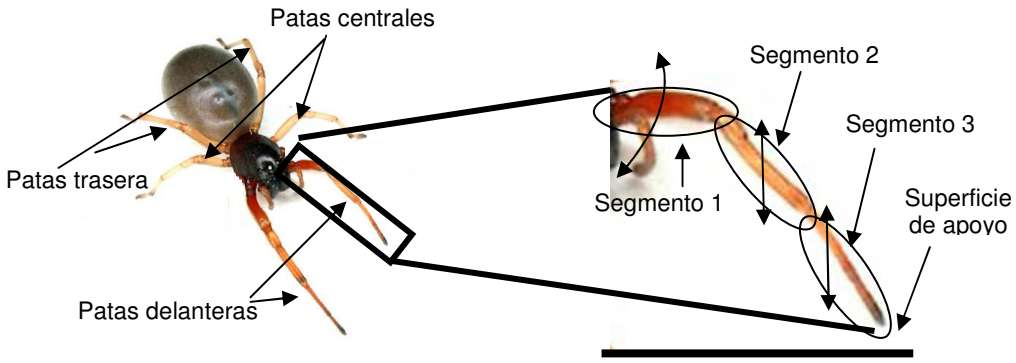


Figura 4.2 Características de las patas de las arañas con configuración circular de patas

En esa figura, puede apreciarse que el segmento 1 realiza movimientos horizontales y los segmentos 2 y 3 realizan movimientos verticales con respecto a la superficie de apoyo de la araña.

En el paso que utilizan estas arañas, las patas delanteras y traseras únicamente mueven los segmentos 2 y 3 de manera adecuada para generar

un movimiento hacia la dirección en que se dirige la araña, mientras que las patas centrales mueven los tres segmentos, pero en especial el segmento 1 para generar un movimiento hacia la dirección en que se dirige la araña. Tomando en cuenta la información anterior, la cual fue obtenida de una observación de la forma de avanzar del tipo de arañas de la figura 4.1, se diseñaron los modelos propuestos en esta tesis.

Como ya se mencionó, en contraste con otros trabajos, en el nuestro no se desea utilizar más sensores y actuadores de los indispensables, ni tampoco se dispone de sensores con gran sensibilidad para el equipamiento del robot. Se espera que el uso de HMDPs nos permitirá manejar de una manera adecuada la incertidumbre que existe en las lecturas de los sensores sin necesidad de tener más sensores o sensores más precisos.

Al decir que el robot será capaz de desplazarse de manera estable nos referimos a que el robot será capaz de moverse de manera autónoma y sin que el robot llegue a caerse dentro de un determinado ambiente. Además, se espera que pueda contender con cambios inesperados en el ambiente, e. g. pendientes y obstáculos, entre otros.

4.1.2 Definición del modelo

Para el diseño de este modelo se utilizó un enfoque jerárquico usando conocimiento previo, esto es, se creó un HMDP usando el enfoque de adición de conocimiento previo.

El conocimiento *a priori* que se le agrega a este modelo, es el paso que debe de seguir el robot para desplazarse de manera estable en el ambiente, lo cual es considerado como una restricción explícita. Dicho paso está inspirado directamente de la manera en que se desplazan las arañas que tienen una distribución circular de patas alrededor de su cuerpo, ilustradas en la figura 4.1.

De acuerdo a la definición que dimos de HMDP (c. f. sección 3.5), para la definición de nuestro primer modelo, hicimos una descomposición del problema original que consiste en controlar las seis patas con un solo MDP, lo cual representa tener un MDP con un gran número de acciones y de estados. Se realizó una descomposición del problema definiendo únicamente los MDPs necesarios para el control adecuado de una sola pata del robot, cada uno de estos MDPs es reutilizado para las otras patas, de modo que los MDPs definidos tienen un menor número de estados y acciones que los que tendría un solo MDP que resolviera el mismo problema.

En este primer modelo definimos una manera de coordinar los MDPs que controlan las patas del robot, para generar un desplazamiento hacia adelante. De modo que, para crear un sistema de control para el robot hexápodo, con este primer modelo utilizamos un HMDP de dos capas. La primera capa o *capa de bajo nivel*, se encarga de realizar acciones básicas de las patas, e. g. levantar una pata o apoyar una pata sobre el plano de apoyo. Dicha capa está compuesta de MDPs los cuales ayudan a que una pata alcance una determinada configuración. La segunda capa o *capa coordinadora*, se encarga de propiciar secuencias de movimiento de las patas del robot. En esta capa se ha agregado el conocimiento *a priori* acerca del paso utilizado por arañas con distribución circular de patas alrededor de su cuerpo. Así, esta segunda capa tiene definida la secuencia de movimientos necesarios de cada una de las patas para generar un movimiento hacia adelante, esto es, esta capa se encarga de aplicar un MDP de la capa de bajo nivel en alguna de las patas, de acuerdo a la secuencia predefinida.

4.1.2.1 Consideraciones

El robot hexápodo físico que se quiere controlar tiene 3 DOF por cada una de las patas. Cada uno de esos DOF corresponde a un segmento de la pata. La figura 4.3 ilustra de manera gráfica la estructura de una pata del robot.

Se realizó una simplificación de las posibles posiciones en las que puede estar un segmento de la pata del robot. Se considera que cada segmento de una pata sólo puede estar en una de tres posiciones, arriba, enmedio o abajo en el caso de los segmentos 2 y 3. Y adelante, atrás y enmedio para el segmento 1, como se muestra en la figura 4.3.

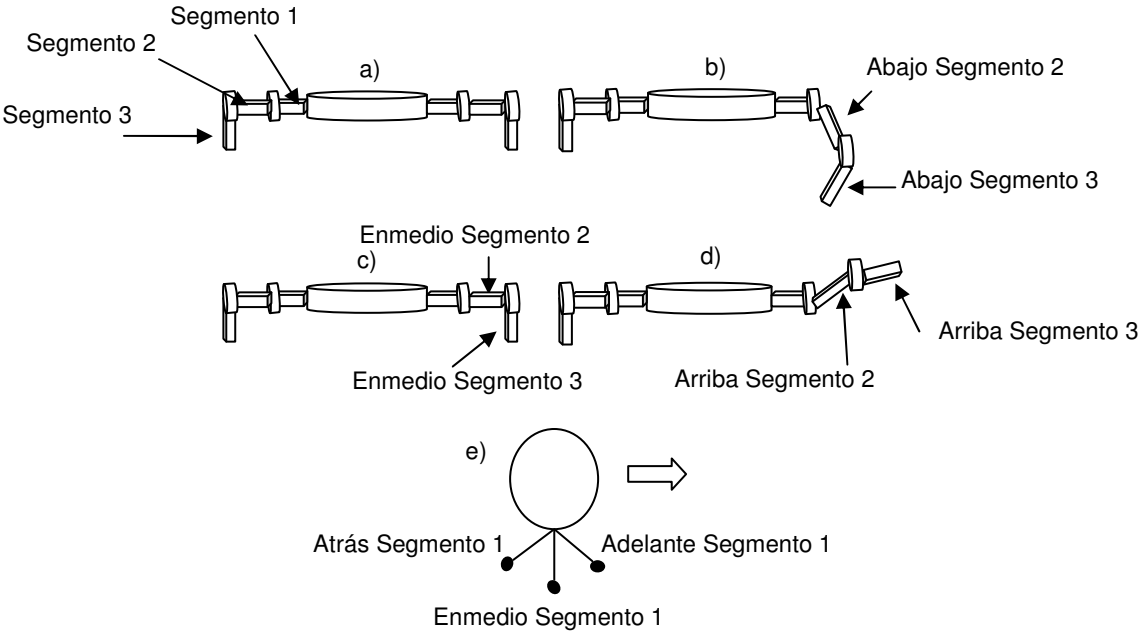


Figura 4.3. a) Segmentos de una pata, b) – d) posibles posiciones de los segmentos 2 y 3 mostrados en la pata derecha, e) posibles posiciones del segmento 1, vista superior

Nótese que si quisiéramos usar sólo un MDP, para la caracterización de los estados de éste, aún tomando en cuenta las consideraciones anteriores, necesitaríamos considerar todas las posibles configuraciones de las seis patas en conjunto. Esto sería equivalente a $(3x3x3)^6$ = 387,420,489

configuraciones posibles. Por otro lado, considerando únicamente dos acciones por segmento se tendrían $(2x2x2)^6$ = 262,144 posibles acciones.

Retomando el análisis de la sección 4.2.1, las posibles configuraciones de una pata de una araña del tipo mostrado en la figura 4.1, de acuerdo a una de tres posiciones ocupada alrededor del cuerpo: trasera, enmedio o delantera, son mostradas en la figura 4.4.Como se observa en la figura 4.4, existen básicamente tres *meta*-configuraciones, no importando qué posición ocupe una pata: trasera, enmedio o delantera. Estas meta-configuraciones se muestran en la figura 4.5.

Una meta-configuración *es* una posición objetivo que cualquier pata, no importando su posición, puede alcanzar. Y dicha posición será la meta a alcanzar de un MDP.

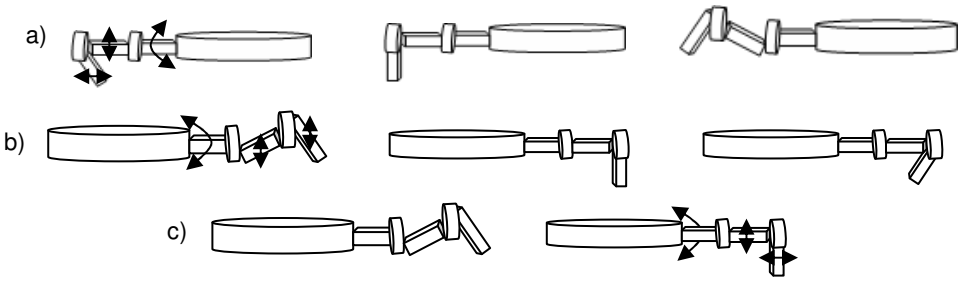


Figura 4.4. Configuraciones posibles de una pata. a) pata trasera, b) pata delantera, c) pata central. Donde ↕ y ↔ indica movimiento vertical y ⟳ indica movimiento horizontal, con respecto al plano de apoyo.

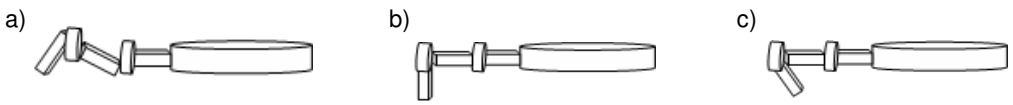


Figura 4.5. Meta-configuraciones, a) meta-configuración que levanta una pata, b) y c) meta-configuraciones que posan una pata

4.1.2.2 Definición de la capa de bajo nivel

Recordemos qué un MDP es una abstracción de un problema general el cual engloba características y consideraciones necesarias para resolver el

problema. Tal abstracción debe reflejar situaciones deseables o metas. Al solucionar el MDP se obtiene la mejor política o directivas de control.

Cuando se usa un enfoque probabilista basado únicamente en el uso de MDP, no es necesaria la incorporación de nueva información para actualizar la política, debido a que toda la información relevante se ha incorporado desde la definición del MDP.

Como ya se ha mencionado, la capa de bajo nivel está conformada por los MDPs que controlan una pata. Para definir estos MDPs analizamos las configuraciones que puede tener una pata del robot. Nótese que las meta-configuraciones de la figura 4.5 sólo involucran posiciones específicas de los segmentos 2 y 3, en estas configuraciones el segmento 1 no es movido, de hecho este segmento será manipulado en la capa de alto nivel.

Partiendo de las meta-configuraciones, necesitamos sólo tres MDPs, uno por cada meta-configuración, pues si utilizáramos un MDP con tres metas alcanzaríamos alguna de las metas pero sería difícil salir de esta una vez llegando a ella. A los MDPs les llamaremos MDP_UP, MDP_DW1 y MDP_DW2. Las metas de MDP_UP, MDP_DW1 y MDP_DW2 son respectivamente, ilustradas, en las figuras 4.5 a), 4.5 b) y 4.5 c). De acuerdo a las configuraciones definidas, estos MDPs se encargan de levantar una pata (MDP_UP) o de posarla en una de dos formas (MDP_DW1 o MDP_DW2).

Una vez identificados los MDPs, hay que definir a cada uno de ellos. Los tres MDPs utilizan en mismo conjunto de estados y de acciones, lo único que variará en sus definiciones es su función de recompensa. El conjunto de estados comprende todas las posibles configuraciones en las que puede estar una pata del robot. Debe recordarse la simplificación de posiciones en las que puede estar un segmento (ver figura 4.3). Considerando que estos

MDPs sólo controlarán dos segmentos de cada pata (ver figura 4.5), tenemos un total de 3x3 = 9 posibles configuraciones, las cuales se muestran en la figura 4.6.

El conjunto de acciones comprende los posibles movimientos que puede realizar una pata, considerando sólo dos de los segmentos que la conforman. La tabla 4.1 muestra el conjunto de acciones de este MDP.

Tabla 4.1. Conjunto de acciones

	Acciones
1	Mover hacia arriba segmento 1
2	Mover hacia abajo segmento 1
3	Mover hacia arriba segmento 2
4	Mover hacia abajo segmento 2
5	No hacer nada

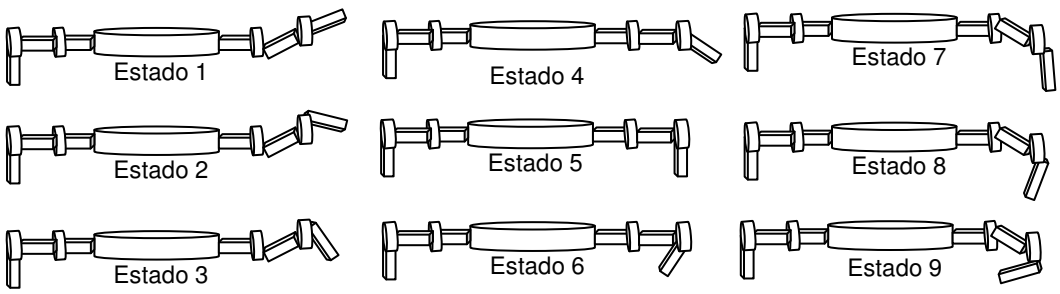


Figura 4.6 Conjunto de estados, ilustrados en la pata derecha

Así, definimos un conjunto de acciones con cinco elementos. Nótese que para estos MDPs utilizamos una acción de no movimiento, la decisión de usar esta acción será explicada posteriormente (c. f. sección 4.2.2.2).

Dado que se tienen nueve estados y cinco acciones, la función de transición será una matriz de 9x9x5. Dado que esta matriz refleja las probabilidades de llegar de un estado a otro ejecutando una determinada acción, esta matriz

será la misma para cada uno de los MDPs. Al definir esta matriz se considera la incertidumbre del resultado de ejecución de una acción. Dicha matriz es mostrada en el apéndice B y la cual fue construida a mano.

Finalmente, las matrices de recompensa las cuales indican la ganancia obtenida de realizar una determinada acción en un estado dado y en las cuales debe verse reflejada la meta que debe alcanzar un MDP se muestran en la tabla 4.2. Como se observa, cada matriz de recompensa refleja la meta que quiere alcanzar cada uno de los MDPs. Esto se logra asignando un valor positivo en el par (estado, acción), lo cual significa, para el robot que estando en ese determinado estado y ejecutando esa determinada acción se puede alcanzar la meta del MDP.

En la tabla 4.2 se muestran las matrices de recompensa de los MDPs MDP_UP, MDP_DW1 y MDP_DW2, donde se ha puesto el valor de 10 en los pares (estado, acción) debido a que estando en ese estado y ejecutando dicha acción es posible alcanzar la meta del MDP.

Tabla 4.2. Matrices de recompensa para los MDPs de la capa de bajo nivel

	Acciones				
Estados	**1**	**2**	**3**	**4**	**5**
1	-1	-1	-1	-1	-1
2	-1	-1	-1	10	-1
3	-1	-1	-1	-1	10
4	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	-1
6	10	-1	-1	-1	-1
7	-1	-1	-1	-1	-1
8	-1	-1	-1	-1	-1
9	-1	-1	-1	-1	-1

a) Matriz de recompensa de MDP_UP

	Acciones				
Estados	**1**	**2**	**3**	**4**	**5**
1	-1	-1	-1	-1	-1
2	-1	10	-1	-1	-1
3	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	10
6	-1	-1	10	-1	-1
7	-1	-1	-1	-1	-1
8	-1	-1	-1	-1	-1
9	-1	-1	-1	-1	-1

b) Matriz de recompensa de MDP_DW1

	Acciones				
Estados	**1**	**2**	**3**	**4**	**5**
1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1
3	-1	10	-1	-1	-1
4	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	10
7	-1	-1	-1	-1	-1
8	-1	-1	-1	-1	-1
9	10	-1	-1	-1	-1

c) Matriz de recompensa de MDP_DW2

4.1.2.3 Definición de la capa coordinadora

Esta capa es la encargada de coordinar los MDPs de la capa de bajo nivel, es decir, es la encargada de indicarles a dichos MDPs el orden y momento en que deben generar un desplazamiento hacia adelante estable. Para definir esta capa, usamos el conocimiento *a priori*, el cual, como ya se ha

mencionado, es el paso que utilizan las arañas con una distribución circular de patas alrededor de su cuerpo. Dicho paso lo hemos caracterizado en ocho movimientos coordinados de las seis patas, los cuales son (ver figura 4.7):

9. Llevar a la meta-configuración 1 las patas 1, 3 y 5
10. Mover hacia adelante la pata 5 y mover hacia atrás la pata 2
11. Llevar a la meta-configuración 2 las patas 1 y 5 y llevar a la meta-configuración 3 la pata 3
12. Llevar a la meta-configuración 1 las patas 2, 4 y 6
13. Llevar a la meta-configuración 3 pata 1 y 5, mover hacia atrás la pata 5 y mover hacia adelante la pata 2
14. Llevar a la meta-configuración 3 la pata 6 y llevar a la meta-configuración 2 las patas 2 y 4
15. Llevar a la meta-configuración 1 las patas 1, 3 y 5
16. Llevar a la meta-configuración 2 la pata 6, llevar a la meta-configuración 3 las patas 4, mover hacia adelante pata 5 y mover hacia atrás pata 2

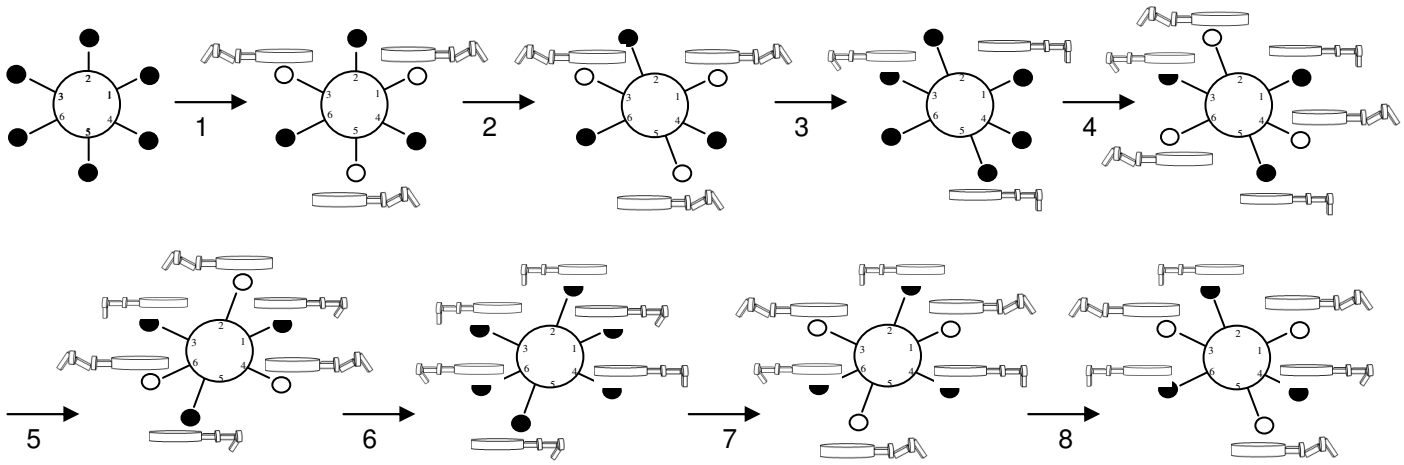


Figura 4.7 Caracterización del paso utilizado por arañas con una distribución circular de patas alrededor de su cuerpo

La indicación llevar una pata a una determinada configuración implica la ejecución de la política del MDP que hace que una pata alcance dicha configuración, e. g. al decir llevar a la meta-configuración 1 las patas 1, 3 y 5 significa que la política del MDP_UP será aplicada tanto en la pata 1, 3 y 5 para que dichas patas alcancen la configuración indicada.

Dicho paso, se basa en el conocimiento *a priori* suministrado al modelo. Usando este conocimiento el robot logra coordinar sus patas. Sin embargo, esta coordinación está impuesta por el diseñador, no fue aprendida por el robot haciendo una exploración de su espacio de estados y de sus acciones.

Finalmente, este primer modelo puede ser visto de manera gráfica como se muestra en la figura 4.8.

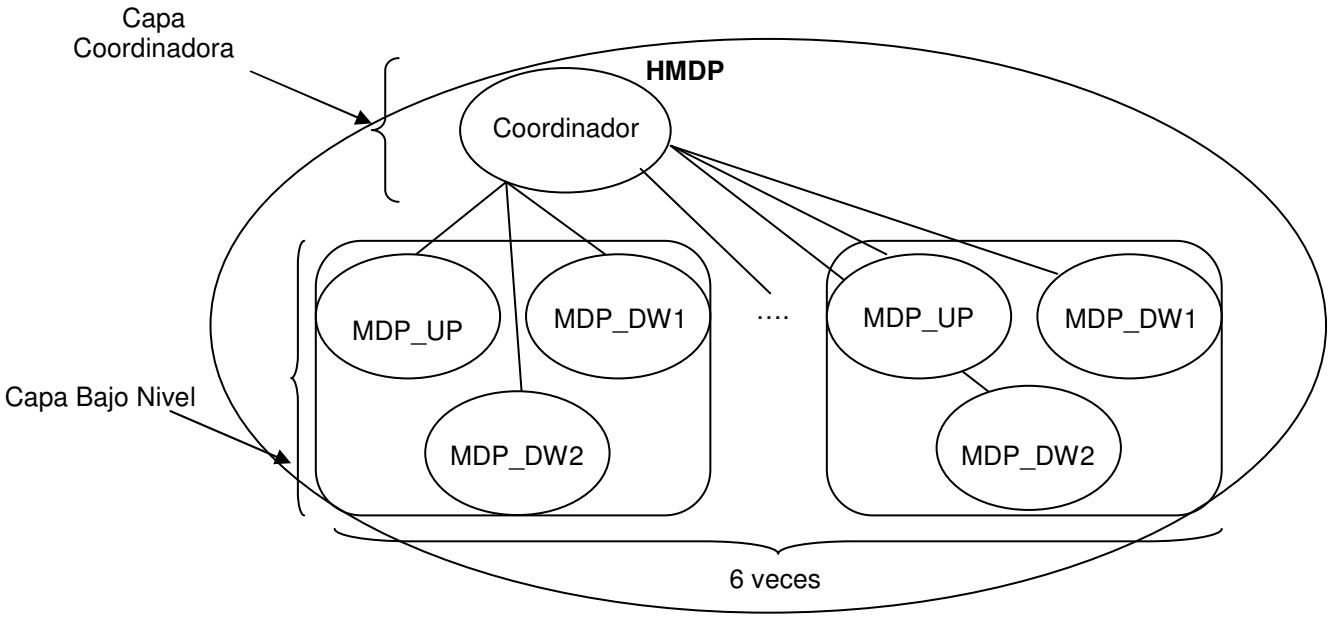


Figura 4.8. Estructura del HMDP del modelo 1

4.2 Modelo 2: MDP con Decisión Descentralizada y Acción Descentralizada (DDAD)

4.2.1 Análisis y justificación

El motivo por el cual se diseñó este segundo modelo, fue buscar una alternativa de descentralización (e. g. Parr & Russell (1997), Sucar (2004)) del MDP original para obtener un nuevo HMDP, de modo que, al igual que en el modelo previo, se obtienen múltiples MDPs más pequeños que el original, pero con la diferencia de que estos MDPs interactúan sin seguir un esquema jerárquico como los del primer modelo, sino siguiendo un esquema de control descentralizado. Además, en este segundo modelo no se le da al HMDP conocimiento *a priori* tan detallado como en el modelo anterior. Al igual que el modelo DCAD, este modelo usa la información obtenida del análisis del paso que utilizan las arañas con una distribución circular de patas alrededor de su cuerpo.

Cuando se quiere segmentar y simplificar un problema global, generalmente muy grande, para ser resuelto de manera distribuida, es necesario crear divisiones lo más independientes posible. Además, es importante que dichas divisiones sean capaces, por sí solas, de resolver parte del problema global sin depender o requerir la intervención de las otras divisiones.

El problema global de esta tesis es hacer que un robot hexápodo sea capaz de desplazarse de manera estable en un ambiente semiestructurado. Dicho problema debe ser segmentado por su complejidad. Una primera división del problema fue la propuesta en el modelo 1, en la cual, en lugar de controlar de manera conjunta y centralizada las seis patas del robot, se realizó una división del problema donde se controlan las patas de forma independiente. Sin embargo, usando esta división es necesaria la intervención de un coordinador que dé turnos de operación a las divisiones para generar un

movimiento estable hacia adelante. Esto es debido a que controlar de manera individual las patas no es suficiente para lograr que el robot avance y al mismo tiempo mantenga su cuerpo despegado de la superficie de apoyo. Es importante señalar que las configuraciones de las patas que permiten generar un desplazamiento hacia adelante pueden no ser útiles para mantener el cuerpo del robot despegado de la superficie de apoyo. Además, debido a la forma en la que están distribuidas las patas del robot y al torque de los servomotores de una pata se requieren al menos dos patas, estó es para robots hexápodos con una distribución circular alrededor de su cuerpo, para lograr estabilidad estática.

Después de un análisis más detallado acerca de la manera de caminar de las arañas con distribución circular de patas alrededor de su cuerpo, se observó que existen ciertos pares de patas, los cuales trabajan de manera independiente a otros pares de patas. Cada par de patas tiene además un paso de movimiento independiente de los otros pares, el cual permite que la araña se mueva hacia adelante y que al mismo tiempo mantenga el cuerpo despegado de la superficie de apoyo. La figura 4.9 muestra tales pares de patas.

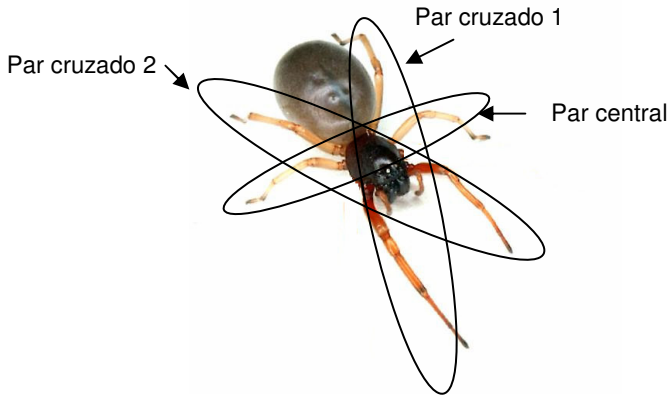


Figura 4.9. Pares de patas en una araña identificadas de acuerdo a su función es el desplazamiento

4.2.2 Definición del modelo 2

En este nuevo modelo se definió un HMDP usando un enfoque descentralizado. Esto es, se rompe el esquema jerárquico y se crean múltiples MDPs que interactúan sin la ayuda de un coordinador para alcanzar un objetivo en común.

Para crear este modelo se consideran los pares mostrados en la figura 4.9 para realizar la nueva división del problema global. Nuestra división del problema consistirá en usar un MDP para controlar cada uno de los pares de patas mostrados en la figura 4.9. Nótese que el *par cruzado 2* puede ser controlado con el mismo MDP que controle el *par cruzado 1,* debido a que su comportamiento es simétrico. De esta forma, nuestro nuevo modelo consistirá de dos MDPs, a los cuales hemos nombrado MDP_C y MDP_M, donde MDP_C controlara el par de patas 1 y 6 y el par 3 y 4 y el MDP_M controla el par 3 y 4 como se ilustra en la figura 4.10.

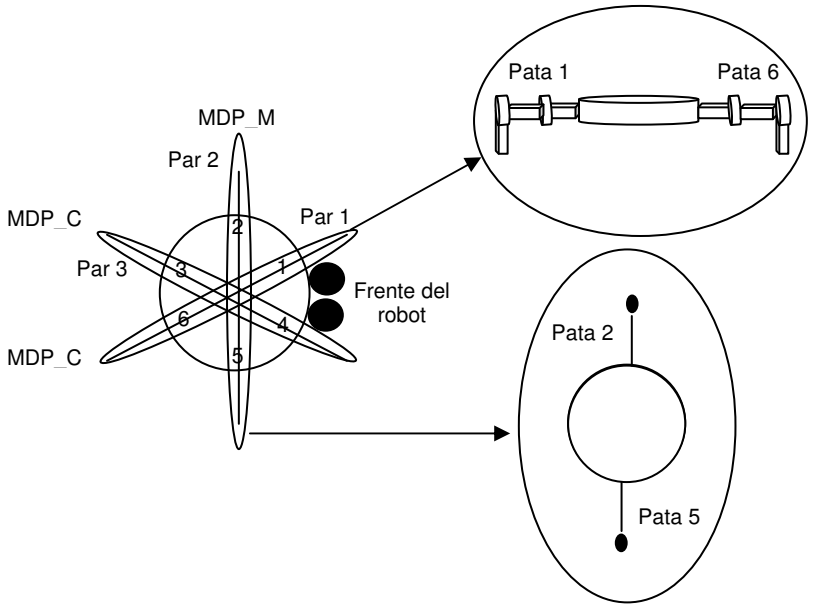


Figura 4.10 Pares de patas a controlar por los MDPs MDP_C y MDP_M

De esta manera, tenemos un HMDP el cual consiste de tres MDPs, cada uno de los cuales con un número menor de estados y de acciones comparados

con el MDP original. Estos MDPs trabajarán de manera independiente, sin la interacción un coordinador, para lograr la meta global.

4.2.2.1 Consideraciones

Las consideraciones acerca de los movimientos válidos de los segmentos de una pata y de las meta-configuraciones expuestas en la sección 4.1.2.1 son usadas para la definición de este nuevo modelo. Ver figuras 4.4 y 4.5.

Cuando un MDP es usado para resolver problemas de locomoción de robots con patas, generalmente la meta de dicho MDP consiste en alcanzar una determinada configuración de patas. Así el objetivo del MDP es aprender secuencias de configuraciones las cuales permitan el desplazamiento de manera estable hacia adelante, no importando en qué configuración se encuentren las patas del robot, estas secuencias serían equivalentes, en un problema de navegación robótica, a aprender las rutas a seguir desde cualquier posición del ambiente para alcanzar una posición meta en el mismo. En problemas de navegación de robots, las rutas dadas por el MDP son rutas que cumplen con dos objetivos principales, llegar a la meta y evitar determinados puntos del ambiente. Similarmente, en problemas de locomoción de robots con patas, es importante que las configuraciones obtenidas puedan alcanzar la configuración objetivo o meta-configuración y eviten ciertas configuraciones no deseadas.

Así, la manera de definir los estados para un MDP que está enfocado a resolver un problema de navegación es distinta a la que se utiliza para un MDPs enfocado a resolver un problema de locomoción de robots con patas. Debido a que lo que se busca en problemas de navegación es que el robot alcance un lugar en su mundo, pero para problemas de locomoción con patas lo que se busca es que el robot genere configuraciones de sus patas lo cuales le permitan caminar. Considerando lo anterior, las aseveraciones que

se hagan en adelante van exclusivamente considerando que estamos trabajando con un problema de locomoción robótica y no de navegación.

Cabe mencionar que para trabajos donde se utilizan MDPs jerárquicos existen dos conceptos, metas y submetas, una meta es un objetivo único, las submetas son subobjetivos. Como se muestra en la figura 4.11 donde la capa superior del HMDP tiene una meta y en la capa inferior existen submetas que son alcanzadas por MDPs, lo cuales dependen de la capa superior.

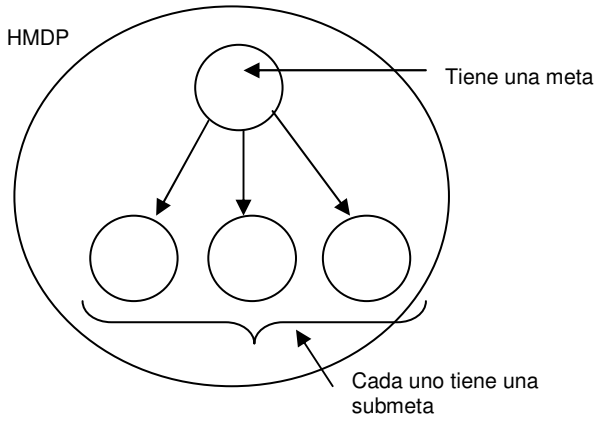


Figura 4.11 Metas en un HMDP

Para el modelo que estamos definiendo, MDP DDAD, definimos lo un nuevo concepto *metas simultáneas.* Las metas simultáneas son metas que no son coordinadas, es decir, que no están supeditadas por una capa superior, como se muestra en la figura 4.12, donde el MDP DDAD tiene una meta, pero en su composición contiene MDPs los cuales tienen metas simultaneas.

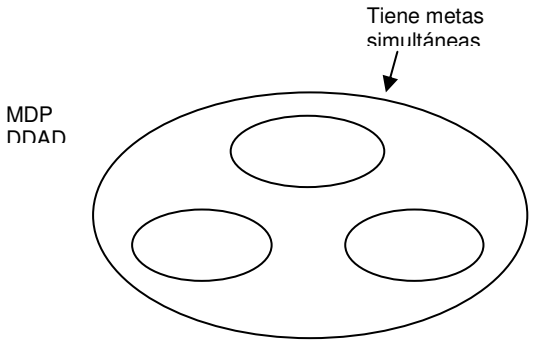


Figura 4.12 Metas en un MDP DDAD

La teoría de MDPs nos dice que un MDP puede tener más de una meta dada explícitamente en la definición de éste, es decir, reflejado en la matriz de recompensa del MDP. En problemas de locomoción, la división del problema en MDPs independientes no es trivial ¿qué pasa cuando se desea alcanzar varias metas simultáneamente?, ¿cómo puede ser construido un MDP que alcance más de una meta, por ejemplo, levantarse y avanzar, o bien, avanzar y mantenerse estable? La división en submetas no parece en estos casos suficiente. Estas reflexiones nos hicieron separar en principio los tipos de metas en exclusivas y simultaneas, para más adelante definir el nuevo modelo.

4.2.2.2 Definición MDP_C

Como ya se ha mencionado, este MDP controlará un par de patas del robot, nuestro espacio de estados estará compuesto por todas las configuraciones válidas de acuerdo a las consideraciones expuestas en la sección anterior.

Este MDP controlará uno de los pares cruzados de la figura 4.9, este par de par de patas tiene la característica de que para generar un movimiento estable hacia adelante sólo manipula los segmentos 2 y 3 de ambas patas y el segmento 1 no es movido. De esta forma, las configuraciones posibles de este par de patas consistirá únicamente en las configuraciones en las que pueda estar este par considerando sólo los segmentos 2 y 3. Así, tenemos un total de (3x3)x(3x3) = 81 configuraciones válidas, las cuales son mostradas en la figura 4.13.

El conjunto de acciones estará compuesto por conjuntos de cuatro acciones básicas. Una acción básica es mover hacia arriba o hacia abajo un segmento determinado de alguna de las dos patas. Los conjuntos de acciones básicas fueron definidos para alcanzar una configuración de un solo movimiento, en lugar de realizar cuatro movimientos para alcanzar una configuración. Ahora

bien, dado que hay que mover dos segmentos por cada pata y cada segmento puede ser movido de dos maneras diferentes, tenemos un total de $(2x2)^2 = 16$ acciones. El conjunto de acciones se muestra en la tabla 4.3.

Debido a que se tienen 81 estados y 16 acciones, la función de transición será una matriz de 81x81x16, esta matriz refleja las probabilidades de llegar de un estado a otro ejecutando una determinada acción. En la definición de dicha matriz se considera la incertidumbre del resultado de ejecución de una acción (dicha matriz no es mostrada en esta tesis debido a sus dimensiones).

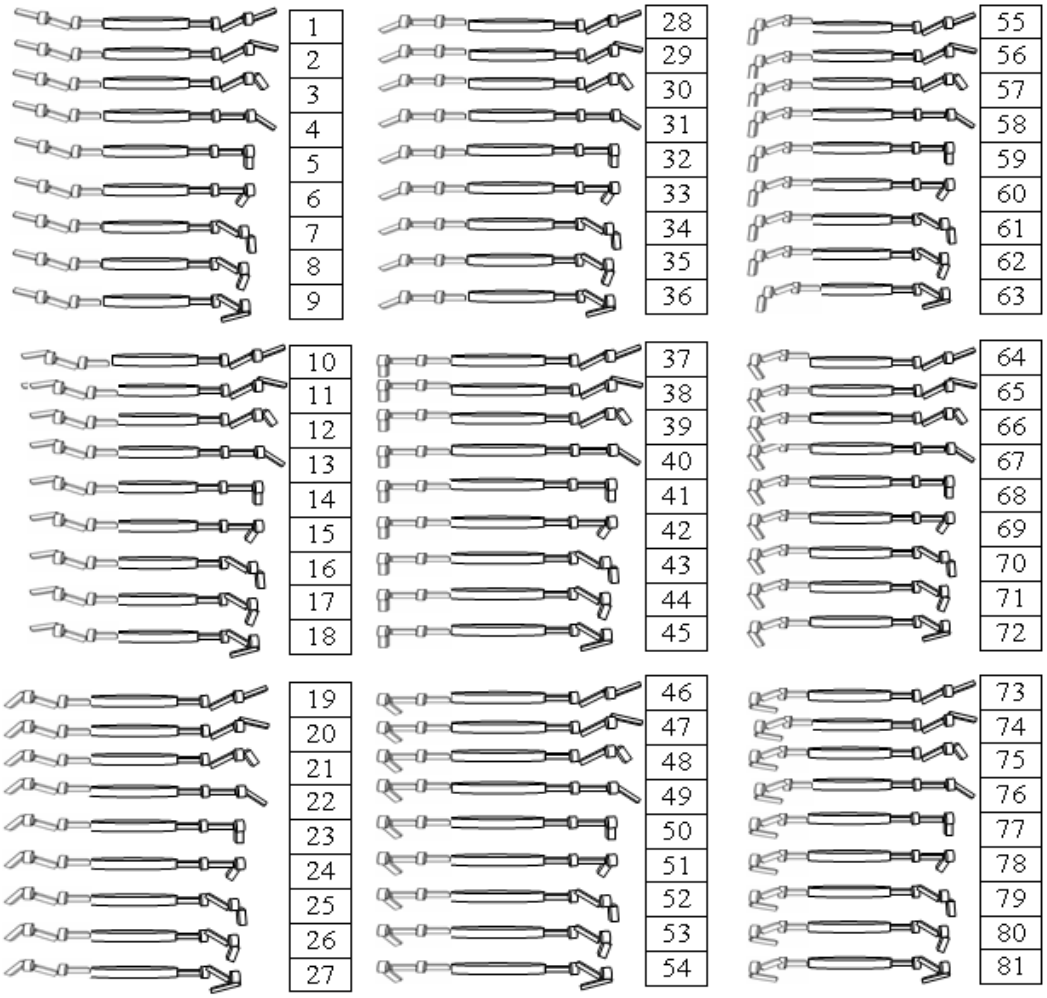


Figura 4.13. Espacio de estados de un par cruzado de patas, donde el número a la derecha de la configuración indica el número de estado con el cual es identificada dicha configuración

Antes de definir la matriz de recompensa del MDP_C, la cual refleja la meta que se quiere alcanzar, recordemos cuál es la meta que quiere alcanzar el MDP_C. La meta de este MDP es lograr que el par de patas que controla genere un paso que permita al robot moverse hacia adelante y además, que dicho par de patas logre mantener levantado de la superficie de apoyo el cuerpo del robot.

Tabla 4.3. Conjunto de acciones, las flechas indican el sentido del movimiento de cada segmento

No. de Acción	Conjunto de acciones básicas			
	Pata 1		Pata 2	
	Segmento 1	Segmento 2	Segmento 1	Segmento 2
1	↑	↑	↑	↑
2	↑	↑	↑	↓
3	↑	↑	↓	↑
4	↑	↑	↓	↓
5	↑	↓	↑	↑
6	↑	↓	↑	↓
7	↑	↓	↓	↑
8	↑	↓	↓	↓
9	↓	↑	↑	↑
10	↓	↑	↑	↓
11	↓	↑	↓	↑
12	↓	↑	↓	↓
13	↓	↓	↑	↑
14	↓	↓	↑	↓
15	↓	↓	↓	↑
16	↓	↓	↓	↓

Lo que se quiere es que este MDP alcance dos metas al mismo tiempo, avanzar y no caerse.

Para resolver el problema anterior hemos definido dos nuevos conceptos, *meta simultánea explícita* y *meta simultánea implícita*. Una meta simultánea

explícita no es otra cosa que una meta tradicional de un MDP, generalmente expresada en la matriz de recompensa con un valor positivo más grande en aquellos pares (estado, acción) que permiten alcanzar el estado meta. La meta simultánea implícita, es una noción propuesta en este trabajo y consiste en una restricción a las acciones que pueda tomar el robot, la cual es expresada en la matriz de recompensa como un valor negativo en los pares (estado, acción) que llevan a un estado no deseable al robot. De modo que al resolverse el MDP, se obtendrá una política la cual evitará que el robot alcance o llegue a pasar por ciertos estados, la meta simultánea implícita, siempre persiguiendo la meta simultánea explícita.

Cabe señalar que la definición de metas a través de valores de valores de recompensa positiva, y de restricción a través de valores de recompensa negativos se han utilizado tradicionalmente en el área, su aplicación en el contexto de submetas y metas simultáneas es distinto en nuestro trabajo.

Considerando lo anterior, es claro que la definición de la matriz de recompensa es fundamental para hacer que el robot alcance o evite determinados estados. Ahora bien, las metas simultáneas de este MDP son alcanzar una configuración la cual permita generar un movimiento hacia adelante y mantener el cuerpo del robot despegado de la superficie de apoyo. Tomando en cuenta las consideraciones acerca de la matriz de recompensa, podemos reflejar ambas metas en dicha matriz y así garantizar que alcance un estado meta y se evite pasar por aquellos estados que hagan que el cuerpo del robot toque o caiga en la superficie de apoyo.

Otro aspecto que ilustra la dificultad de alcanzar metas simultáneas con un MDP, es la consideración sobre lo que el robot o agente debe realizar una vez alcanzado el estado meta. Generalmente, la política obtenida del MDP tiene una acción definida cuando se alcance el estado meta, pero dicha acción llega a ser innecesaria debido a que si el robot se encuentra en un

estado meta, considera que ya ha logrado el objetivo del MDP y no ejecuta ninguna otra acción lo aleje de ese estado absorbente, como se observa en la figura 4.14. Específicamente en aplicaciones de navegación robótica llega a ser suficiente debido a que el objetivo es que el robot aprenda la mejor manera de llegar a un punto específico en un ambiente conocido, partiendo de cualquier punto de dicho ambiente. Es por eso que en el modelo anterior, DCAD, se incluyó una acción la cual era no hacer nada, con el fin de que una vez alcanzada la meta se terminara de ejecutar el MDP correspondiente.

Ahora bien, en problemas de locomoción de robots como el que estamos atacando, el objetivo es que el robot aprenda la manera de avanzar hacia adelante y no se caiga. Pero, el tener una meta absorbente por MDP para la locomoción de un robot polípedo llega hacer insuficiente, dado que se necesitará alcanzar y alternar más de un estado para generar un paso estable hacia adelante. Esto es fundamental si consideramos que únicamente un HMDP será el encargado de encontrar dicho paso, sin la intervención de un coordinador, como ocurre en el modelo DCAD.

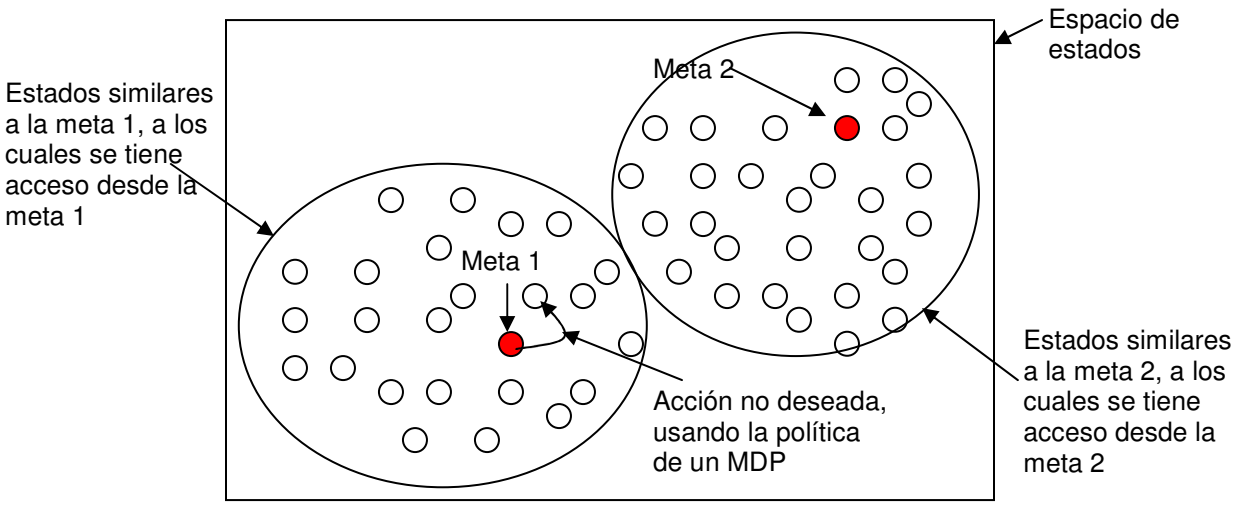


Figura 4.14 Esquema de funcionamiento de un MDP donde se ilustra la dificultad de alcanzar una segunda meta y el resultado de ejecutar una acción estando en una meta

Una posible solución para el primer problema es considerar una acción en la que el robot no haga nada, como en el primer modelo. Esto implicaría que una vez alcanzado el estado meta ya no habría manera de salir de él, a menos que existiera algún coordinador en una capa superior, el cual ejecutaría algún otro MDP, lo cual no es deseado para este modelo. De modo que hemos propuesto un criterio de llamada entre MDPs, el cual consiste en que una vez que un MDP, *MDP1,* haya alcanzado su meta, es decir, cuando se encuentre en el estado meta, en lugar de ejecutar la acción dada por la política envíe una señal a otro MDP, *MDP2*. Dicha señal le indicará al *MDP2* que debe ejecutarse. El *MDP2* tendrá el mismo objetivo, hacer que el robot aprenda a avanzar hacia adelante de manera estable, pero usando alguna otra configuración distinta a la del *MDP1*, que le ayude a conseguirlo. Así, el *MDP2* tendrá el mismo espacio de estados y de acciones pero con una meta diferente, es decir, una matriz de recompensa diferente. Gráficamente podríamos ver al MDP_C como se muestra en la figura 5.15.

La política obtenida después de resolver el MDP MDP_C, tendrá acciones que guiarán al par de patas a alcanzar una meta-configuración. Aunado a esto se presentan dos problemas, primero, dicha política tendrá, como se ha discutido, una acción a ejecutar cuando el robot se encuentre en el estado meta, la cual sacará momentáneamente del estado meta al robot llevándolo a algún otro estado. A pesar de que regresaremos nuevamente al estado meta, dado que la política es aplicada continuamente, la acción tomada en el estado meta puede generar un movimiento no deseado pues lo que se quiere es que una vez en el estado meta el robot permanezca ahí. El segundo problema se presenta cuando al tener sólo una meta a alcanzar, se limitan los movimientos del robot, haciendo que éste sólo use configuraciones similares a la meta-configuración, esto es, se limita a encontrar configuraciones lo más cercanas posibles a la meta-configuración, sin

explorar el posible uso de configuraciones distintas a la meta-configuración. Ver la figura 4.14.

Por último, para terminar de definir el *MDP_C* habrá que definir las metas y modo de interactuar entre los MDPs *MDP1* y *MDP2*.

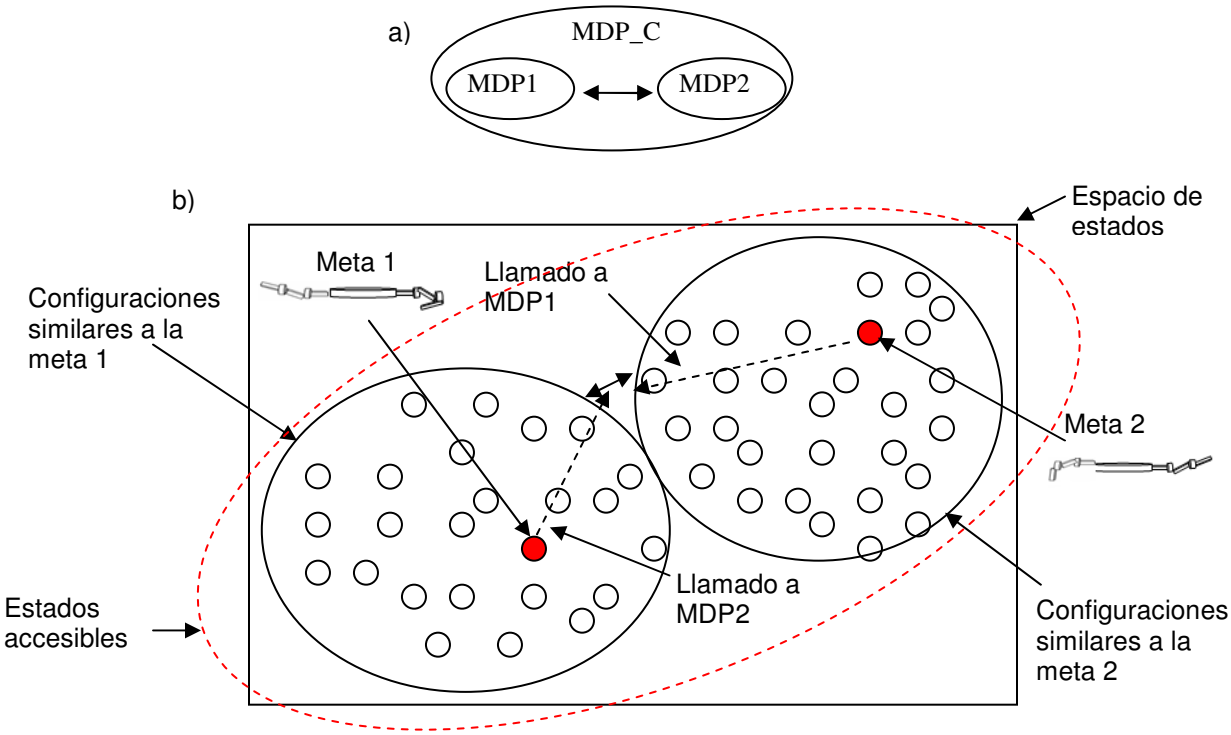


Figura 4.15 a) Estructura del MDP_C, b) esquema de funcionamiento del MDP_C donde se ilustra cómo pueden ser alcanzadas dos metas realizando llamados entre dos MDPs cada uno con una meta distinta

Como deseamos que la búsqueda de configuraciones para obtener un paso estable se realice en todo el espacio de configuraciones, la meta del MDP *MDP1* será el estado 9 ver figura 4.15. Esta meta permitirá explorar en las primeras configuraciones del espacio de estados, dado que las configuraciones más similares a esta configuración se encuentran en las primeras configuraciones del espacio de estados y como se discutió previamente, la búsqueda se realiza entre las configuraciones más similares

a la meta-configuración. Por las mismas razones, la meta del MDP *MDP2* será el estado 55 ver figura 4.15 la cual permitirá explorar en las últimas configuraciones del espacio de estados. Se han elegido estas configuraciones debido a que ambas contribuirán a mantener el cuerpo del robot desplegado de la superficie de apoyo y permitirán realizar un avance hacia adelante. Las matrices de recompensa de ambos MDPs, en las cuales se ven reflejadas dichas metas, se muestran en el apéndice C.

4.3.2.3 Definición MDP_M

Al igual que con el MDP_C, se utiliza la misma estrategia, uso de metas implícita y explícita, uso de dos MDPs, *MDP1* y *MDP2* para la definición del *MDP_M*. La estructura del *MDP_M* se muestra en la figura 4.16.

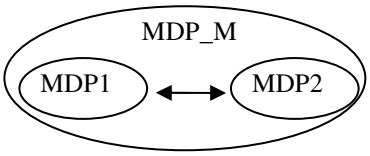


Figura 4.16 Estructura del MDP_M

Ahora bien, como se había mencionado el par de patas centrales se enfoca esencialmente en el control del movimiento del segmento 1 sin importarle cómo se mueven los segmento 2 y 3 (véase figura 4.3). Al MDP_M únicamente le interesa en qué posición está el segmento 1 y si la pata está posada o levantada de la superficie de apoyo. Así, el espacio de estados para este MDP comprende todas las posibles configuraciones en las que sólo importan las consideraciones anteriores. Así, tendremos un total de (3x2)x(3x2) = 36 estados, los cuales son ilustrados en la figura 4.17.

El conjunto de acciones, al igual que para MDP_C, estará compuesto por conjuntos de cuatro acciones básicas. En este caso las acciones básicas son: mover hacia adelante o atrás el segmento 1 de alguna de las patas

involucradas en el par, y levantar o bajar alguna de las patas. Para llevar a cabo las acciones levantar o bajar utilizamos respectivamente los MDPs MDP_UP y MDP_DW1 con el fin de lograr tener un mejor control de ambas patas. El conjunto de acciones se muestra en la tabla 4.4.

Dado que se tienen 36 estados y 16 acciones, la función de transición será una matriz de 36x36x16 (dicha matriz no es mostrada en esta tesis debido a sus dimensiones).

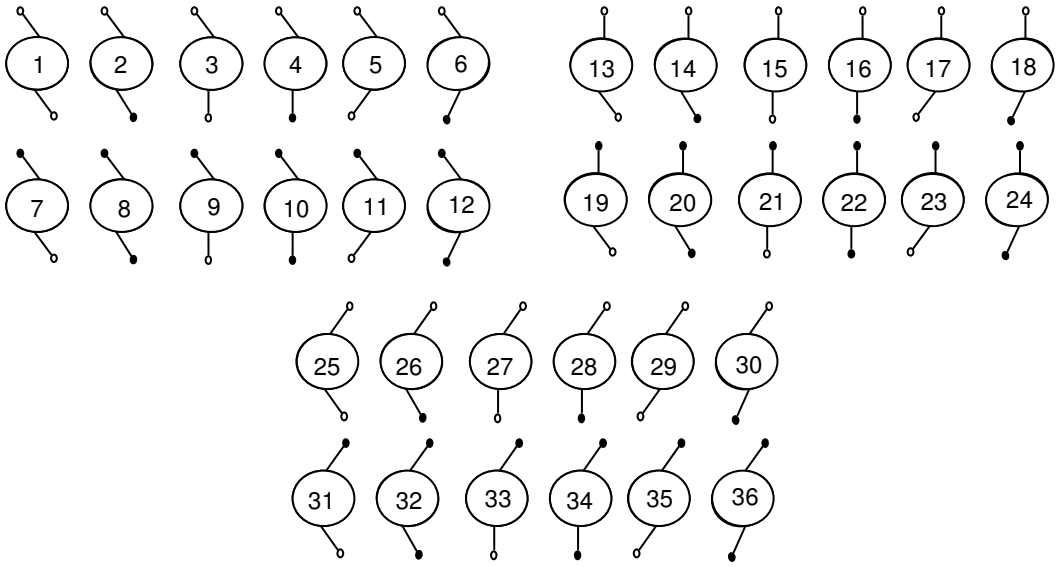


Figura 4.17 Espacio de estados. Los círculos negros indican que la pata está posada y los blancos indican que la pata está levantada de la superficie de apoyo. Por claridad, el robot se muestra desde una vista panorámica

Finalmente, la meta para el *MDP1* y *MDP2* de *MDP_M* son los estados 2 y 35 respectivamente, ver figura 4.17, para que al igual que en el MDP *MDP_C* se realice una exploración completa de todas las configuraciones del espacio de estados. Las matrices de recompensa de *MDP1* y *MDP2* se muestran el apéndice D.

Como se mencionó, el MDP *MDP_C* controla los pares de patas (1,6) y (3,4), de esta manera el HMDP de este modelo es visualizado gráficamente como se muestra en la figura 4.18.

Tabla 4.4 Conjunto de acciones, las flechas con dirección vertical indican si una pata es levanta o apoyada y las flechas con dirección horizontal indican el sentido del movimiento del segmento 1

No. de Acción	Conjunto de acciones básicas			
	Pata 1		Pata 2	
	Mov. Vertical	Mov. Horizontal	Mov. Vertical	Mov. Horizontal
1	↑	→	↑	→
2	↑	→	↑	←
3	↑	→	↓	→
4	↑	→	↓	←
5	↑	←	↑	→
6	↑	←	↑	←
7	↑	←	↓	→
8	↑	←	↓	←
9	↓	→	↑	→
10	↓	→	↑	←
11	↓	→	↓	→
12	↓	→	↓	←
13	↓	←	↑	→
14	↓	←	↑	←
15	↓	←	↓	→
16	↓	←	↓	←

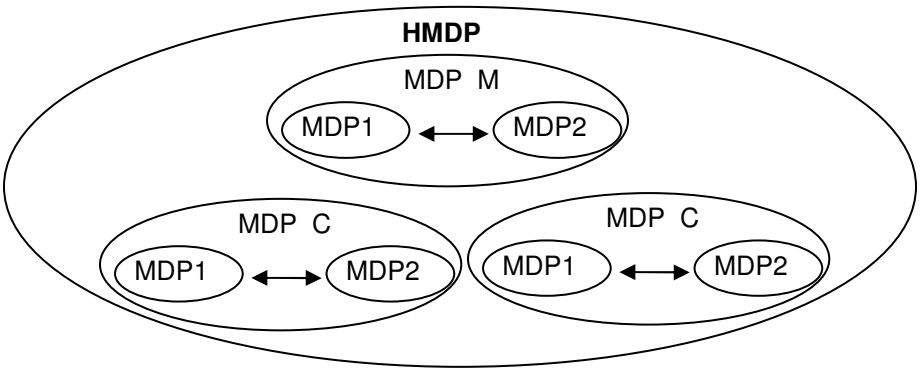


Figura 4.18 Estructura del HMDP, modelo DDAD

Nótese que los MDPs que conforman a este HMDP no tienen comunicación alguna ni dependen para su ejecución de señales enviadas entre ellos. Las señales enviadas entre MDPs ocurren estrictamente al interior de MDP_M y MDP_C, por ello hemos catalogado como *HMDP descentralizado* al HMDP de este modelo.

4.3 Discusión

En este capítulo se han expuesto dos modelos usando HMDPs para el control de un robot hexápodo, cabe mencionar que no se llegó a estos modelos de manera automática, sino que existieron modelos intermedios los cuales no se describen aquí, que fueron la base para la creación de estos modelos finales: DCAD y DDAD. Un MDP jerárquico puede verse como un MDP centralizado, por su parte un MDP DDAD puede verse como un MDP descentralizado. Cabe mencionar que para algunos autores del área de inteligencia artificial distribuida (Durfe, 1999) y robótica (Cao et al., 1997) los modelos DCAD y DDAD se conocen respectivamente como centralizado o jerárquico y distribuido. Recordemos que las aseveraciones sobre MDPs son aplicables a aquellos MDPs que son usados para resolver problemas de locomoción.

Se han descrito conceptos necesarios para el entendimiento de las diferencias que existen en los HMDPs usados para controlar robots con patas y para controlar robots con un arreglo diferencial de llantas, y aún más, las diferencias que existen en aplicar HMDPs para solucionar problemas de navegación robótica con respecto a problemas de locomoción de robots con patas. Para ello, hemos propuesto e implantado cuatro conceptos: meta simultánea, meta exclusiva, meta simultánea implícita y meta simultánea explícita.

En este último modelo, modelo DDAD, se ha descrito la manera en la que se obtuvo dicho modelo desde un punto de vista roboticista. Esta manera de dividir el problema consideramos constituye el inicio de una discusión de mayor importancia, la cual rebasa el ámbito de esta tesis, la definición de *principios de organización de MDPs*. Esto es, estamos proponiendo la generación de MDPs multi-objetivo vistos como un solo MDP, el cual está constituido de varios sub MDPs, donde el MDP principal o TopMDP puede tener múltiples objetivos o metas simultáneas y los sub MDPs que lo conforman únicamente pueden tener un objetivo o meta exclusiva.

Afirmamos que nuestra propuesta puede contribuir a establecer principios de organización de MDPs pues, con las debidas extensiones, nuestro modelo DDAD podría servir para atacar problemas de dos o más objetivos, para el mismo u otro de los muchos problemas en robótica de robots polípedos. No sería necesario redefinir nuestro MDP DDAD, si por ejemplo, deseáramos que nuestro robot hexápodo además de caminar pudiera brincar, pues al esquema actual se le incorporaría una nueva capa al HMDP la cual tendría por objetivo hacer que el robot pudiese brincar.

Este tipo de trabajos, que se alejan de la inevitable pero no siempre capitalizable y reproducible experimentación robótica, son necesarios en el área y deben converger hacia el establecimiento de principios o esquemas de organización más o menos genéricos, que sean aplicables a familias de problemas y no sólo a casos específicos de estudio. Hasta el momento podemos comentar una de las principales diferencias entre el modelo DCAD y el modelo DDAD, la cual es el numero de estados y de acciones por MDP, esta diferencia es debida a la manera en la que son definidos ambos modelos, para el modelo DDAD se ha realizado un análisis más a detalle de cómo alcanzar la meta global usando MDPs que trabajen de manera independiente. Más ventajas y desventajas de ambos modelos serán discutidas después de las pruebas realizadas a cada uno de ellos.

Bibliografía

1. Andre D. y Russell S. J. (2001). Programmable reinforcement learning agents. *Advances in Neural Information Processing Systems*, Leen T. K., Dietterich T. G., y Tresp V., 13, 1019-1025, MIT Press.

2. Armada M., Prieto M., Akinfiev T., Fernández R., González P., García E., Montes H., Nabulsi S., Ponticelli R., Sarria J, Estremera J., Ros S., Grieco J. y Fernandez G. (2005). On the design and development of climbing and walking robots for the maritime industries, *Journal of maritime research*, 2 (1): 9 – 32.

3. Asada M., Uchibe E. y Hosoda K. (1996). Behavior Coordination for a Mobile Robot Using Modular Reinforcement Learning. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems,* 1329 – 1336.

4. Bakker B., Zivkovic Z. y Krose B. (2005). Hierarchical dynamic programming for robot path planning. *International Conference on Intelligent Robots and Systems, 2005. (IROS 2005),* 2756 – 2761.

5. Bakker P. B. (2004). *The State of Mind, Reinforcement Learning with Recurrent Neural Networks*. Tesis doctoral. Universiteit Leiden.

6. Bares J. y Wettergreen D. (1999). Dante II: Technical Description, Results and Lessons Learned. *International Journal of Robotics Research*. 18 (7): 621 – 649.

7. Beer R. D., Chiel H. J. y Ritzmann R. E. (1997). Biologically Inspired Approaches to Robotics. *Communications Of The ACM*, 40(3): 30-38.

8. Bellman R. E. (1957a). *Dynamic Programming.* Princeton University Press. New Jersey.

9. Bellman, R. E. (1957b). A Markov decision process. *Journal of Mathematical Mech.,* 6:679 – 684.

10. Berardi F. Chiaberge M. Miranda E. y Reyneri, L. M. (1996). A neuro-fuzzy systems for control applications. *International Symposium on Neuro-Fuzzy Systems, 1996.* AT'96, Lausanne, Switzerland.

11. Berns K. (2002). Biological Inspired Walking Machines. *Intelligent Autonomous Systems,* 7, 118 – 126.

12. Berns K., Cordes S. y Ilg W. (1994). Adaptive, neural control architecture for the walking machine LAURON. *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems '94*. Advanced Robotic Systems and the Real World, IROS '94, 2, 1172 – 1177.

13. Bertsekas D. P. (2005). Dynamic Programming and Optimal Control, Athena Scientific, Belmont, Massachusetts, 1 y 2.

14. Billard, A. y Dautenhahn, K. (1997) Grounding communication in situated, social robots. *Proceedings of the Towards Intelligent Mobile Robots Conference*, Manchester, Technical Report Series, Department of Computer Science, Manchester University. 1361 - 6161.

15. Boone G. (1997). Minimum-time control of the acrobot. *International Conference on Robotics and Automation*, Albuquerque, NM.

16. Boutilier C., Brafman R. y Geib C. (1997). Prioritized goal decomposition of Markov decision processes: Towards a synthesis of classical and decision theoretic planning. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, Martha Pollack, 1156-1163, San Francisco, Morgan Kaufmann.

17. Boutilier C., Dean T. y Hanks S. (1999). Decision-theoretic planning: structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 27, 199 - 230.

18. Brooks, R. (1990). Elephants Don't Play Chess. *Robotics and Autonomous Systems, 6*, 3 – 15.

19. Cao Y. U., Fukunaga A. S. y Kahng A. B. (1997). Cooperative Mobile Robotics: Antecedents and Directions. *Autonomous Robots*, 4, 1 – 23.

20. Cassandra A. R., Kaelbling L. P. y Kurien J. A. (1996). Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 963 – 972.

21. Celaya E. y Albarral L. J. (2003). Implementation of a hierarchical walk controller for the LAURON III hexapod robot. *Proceedings of the 6th International Conference on Climbing and Walking Robots*, 409 – 416, Catania, Italy.

22. Chen, I. M. y Yeo, S. H. (2003). Locomotive Gait Generation for Inchworm-Like Robots Using Finite State Approach. *The International Journal Robotics Research*, 22(1): 21-40.

23. Cheol Ki C., Cheol M. y Jo S. (2003). Go Development of a biped robot with toes to improve gait pattern. *Proceedings IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2(20-24), 729 – 734.

24. Cruse R. (1976). Function of the Legs in the Free Walking Insect, Caurausius morus. *Journal of Comparative Physiology*, 112, 235 –262.

25. Dayan P. y Hinton G. E. (1993). Feudal reinforcement learning. *Advances in Neural Information Processing Systems 5, Proceedings of the IEEE Conference in Denver*, Giles C. L., Hanson S. J., y Cowan J. D., San Mateo, CA, . Morgan Kaufmann 13, 227 – 303.

26. Dean T. y Lin S. H. (1995). Decomposition techniques for planning in stochastic domains. *International Joint Conference on AI 1995*, 1121-1127.

27. Degris T., Sigaud O., Wuillemin P.-H. (2006). Learning the structure of factores markov decision processes in reinforcement learning problems. *Proceedings of the 23nd International Conference on Machine Learning*, Pittsburgh, USA.

28. DeJong G. y Spong M. W. (1994). Swinging up the acrobot: An example of intelligent control. *Proceedings of the American Control Conference*, 2158 – 2162.

29. Demiris J. y Gillian H. (1996) Imitative Learning Mechanisms in Robots and Humans. *Proceedings of the 5th European Workshop on Learning Robots*, Klingspor V., 9 – 16, Bari, Italy.

30. Dietterich T. G. (1998). The MAXQ method for hierarchical reinforcement learning. *Proceedings 15th International Conference on Machine Learning*, 118 –126, Morgan Kaufmann, San Francisco, CA.

31. Dietterich T. G. (2000a). Hierarchical reinforcement learning with the MAXQ value function decomposition. Journal of Artificial Intelligence Research, 13:227-303.

32. Dietterich T. G. (2000b). State abstraction in MAXQ hierarchical reinforcement learning. *Advances in Neural Information Processing Systems,* 12, 994 –1000.

33. Dietterich, T. G. (2000). An Overview of MAXQ Hierarchical Reinforcement Learning. B. Choueiry Y. y Walsh T. *Proceedings of the Symposium on Abstraction, Reformulation and Approximation SARA 2000,* 26 -44*,* Springer Verlag, New York.

34. Ding Z. (1996). Implementing fuzzy logic control with the XA. AN710. *Phillips Semiconductors Application*, 555.

35. Dorigo M., Trianni V., Sahin E., Gross R., Labella T. H., Baldassarre G., Nolfi S., Deneubourg J.-L., Mondada F., Floreano D. y Gambardella L. M. (2004). Evolving self-organizing behaviors for a swarmbot. *Autonomous Robots,* 17(2-3)*,* 223 – 245.

36. Durfe E. H. (1999). Distributed Problem Solving and Planning. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence,* 121 – 164, MIT Press.

37. Feldman J. A. y Sproull R. F.. (1977). Decision theory and artificial intelligence: the hungry monkey. *Cognitive Science*, 1, 158-192.

38. Ferrell C. L. (1995). Global behavior via cooperative local control. *Autonomous Robots*, 2(2), 105 –125, Springer Netherlands.

39. Fikes R. E. y Nilsson N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence*, 2, 189-208.

40. Gorrostieta E. y Vargas E. (2004). Free Locomotion for Six Legged Robot. *3th WSEAS International Conference on Signal Processing, Robotics and Automation, ISPRA 2004*, 13-15, Salzburg, Austria.

41. Gu Y. (2003). *Solving Large Markov Decision Processes*. Reporte técnico, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada.

42. Guestrin C., Koller D., Parr R. (2001) Multiagent planning with factored MDPs. *Proceedings of the 14th Neural Information Processing Systems*, 1523-1530.

43. Hauskrecht M., Meuleau N., Kaelbling L. P., Dean T. y Boutilier C. (1998). Hierarchical solution of Markov decision processes using macro-actions. *Proceedings of the Fourteenth Conference on*

Uncertainty In Artificial Intelligence, 220-229, San Francisco, Morgan Kaufmann.

44. Hayes, G.M. y Demiris J. (1994). A Robot Controller Using Learning by Imitation. *Proceedings of the 2nd International Symposium on Intelligent Robotic Systems*, Grenoble, France.

45. Howard R. A. (1960). *Dynamic Programming and markov Processes*. MIT Press, Cambridge, Massachusetts.

46. Jindrich D. L. y Full R. J. (1999). Many- legged maneuverability: dynamics of turning in hexapods. *Journal of Experimental Biology,* 202(12): 1603-1623.

47. Jonsson A., Baron A. (2005). A causal approach to hierarchical decomposition of factored MDPs. *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany.

48. Kaelbling L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. *International Conference on Machine Learning*, 167 – 173.

49. Kaelbling L. P., Littman M. L. y Cassandra A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 99-134.

50. Kerscher T., Albiez J., Zöllner J.M. y Dillmann R. (2004). AirInsect - A New Innovative Biological Inspired Six-Legged Walking Machine Driven by Fluidic Muscles . *Proceedings of IAS 8, The 8th Conference*

on Intelligent Autonomous Systems, 10-13. Amsterdam, The Netherlands.

51. Kimura H. y Kobayashi S. (1997). Reinforcement Learning for Locomotion of a Two-linked Robot Arm. *Proceedings of the 6th European Workshop on Learning Robots*, 144 –153.

52. Kimura H. y Kobayashi S. (1999). Efficient Non-Linear Control by Combining Q-learning with Local Linear Controllers. *16th International Conference on Machine Learning*, 210--219.

53. Kindermann T. (2002). Behavior and Adaptability of a Six-Legged Walking System with Highly Distributed Control. *International Society of Adaptive Behavior,* 9(1), 16 – 41.

54. Kingsley D. A., Quinn R. D. y Ritzmann R. E. (2003). A Cockroach Inspired Robot With Artificial Muscles. *International Symposium on Adaptive Motion of Animals and Machines (AMAM 2003)*. 1837-1842, Kyoto, Japan.

55. Koenig S. y Simmons R. (1998). Xavier: a robot navigation architecture based on partially observable Markov decision process models. *Artificial intelligence and mobile robots*, 91-122.

56. Laroche P. (2000). Building efficient partial plans using Markov decision processes. *Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence, 2000. ICTAI 2000*, 156 – 163, Vancouver, BC, Canada.

57. Lee H., Shen Y., Yu C., Singh G. y Ng A. Y. (2006). Quadruped robot obstacle negotiation via reinforcement learning. *IEEE International Conference on Robotics and Automation* (ICRA 2006).

58. Lewis M. A., Hartmann M. J., Etienne-Cummings R. y Cohen A. H. (2001). Control of a robot leg with an adaptive a VLSI CPG chip, *Neurocomputation*, 1409 – 1421.

59. Lizcano R., Puentes J. C. y Valenzuela C. A. (2005). *Control para un robot articulado con tres grados de libertad que simule el movimiento de pata.* Tesis de ingeniería electrónica, Universidad Javeriana Facultad de Ingeniería, Bogota D. C.

60. Maes P. y Brooks R. A. (1990). Learning to Coordinate Behaviors. *Proceedings of the IAAA 1990 Conference*, Morgan Kanfman, San Mateo, California: 224 – 230.

61. Mahadevan S. y Connell J. (1991). Automatic programming of behavior-based robots using reinforcement learning. *Proceedings of the Ninth National Conference on Artificial Intelligence*, 8 – 14, Anaheim, C.A.

62. Marthi B., Latham D., Russell S. y Guestrin C. (2005). Concurrent Hierarchical Reinforcement Learning. *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*.

63. Mataric M. J. (1994). Reward functions for accelerated learning. *Proceedings of the Eleventh International Conference on Machine Learning*, W. W. Cohen and H. Hirsh, Morgan Kaufmann.

64. Meuleau N., Hauskrecht M., Kim K. E., Peshkin L., Kaelbling L., Dean T. y Boutilier C. (1998). Solving very large weakly coupled Markov decision processes. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 165-172, Madison, WI.

65. Minsky M. (1985). *The Society of Mind*. Simon and Schuster, New York.

66. Mori T., Nakamura Y., Sato M. y Ishii S. (2004). Reinforcement Learning for CPG-Driven Biped Robot. *Proceedings of the IAAA 2004*: 623-630.

67. Nakada K., Asai T. y Amemiya Y. (2003). An analog central pattern generator for interlimb coordination in quadruped locomotion. *IEEE Transactions on Neural Networks*, 14(5), 1356 – 1365.

68. Nilsson N. J. (1984). Shakey the robot (Tech. Rep.). SRI A.I. *Technical Note 323*.

69. Nourbakhsh I., Powers R. y Birchfield S. (1995). DERVISH an office navigating robot. *AI Magazine*, 16(2), 53-60.

70. Ocaña M. (2005). *Sistema de localización global WiFi aplicado a la navegación de un robot semiautónomo.* Tesis doctoral, Universidad de Alcalá Escuela Politécnica Superior, Alcalá de Henares, España.

71. Parr R. (1998). Flexible decomposition algorithms for weakly coupled Markov decision problems. *Proceedings of the Fourteenth Annual Conference in Uncertainty in Artificial Intelligence (UAI-98),* 422-430, Morgan Kaufmann.

72. Parr R. y Russell S. (1997). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Systems*, 10, Jordan M. I., Kearns M. J. y Solla S. A., The MIT Press.

73. Patel G., Holleman J. y Deweerth S. (1998). Analog VLSI model of intersegmental coordination with nearest neighbor coupling. *Advanced Neural Information Processing Systems*, 10, 710–725.

74. Patel G., Holleman J. y Deweerth S. (1998). Analog VLSI model of intersegmental coordination with nearest neighbor coupling. *Advanced Neural Information Processing Systems,* 10, 710–725.

75. Pineau J., Gordon G. y Thrun S. (2003). Policy-contingent abstraction for robust robot control. *Conference on Uncertainty in Artificial Intelligence (UAI)*: 477 - 484.

76. Puterman M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming,* John Wiley & Sons.

77. Puterman M. L. (2005). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley-Interscience, 2 edición.

78. Raibert M. H. (1986). Legged robots. *Communications of the ACM*, 29(6), 499 – 514.

79. Reyes A., Ibarguengoytia P., Sucar L.E., Morales E. (2006) Abstraction and Refinement for Solving Continuous Markov Decision Processes. *Proceedings of the The third European Workshop on Probabilistic Graphical Models (PGM-06)*, 263-270.

80. Reyes A., Sucar L.E., Morales E.F., Ibarguengoytia P.H. (2006) Solving Hybrid Markov Decision Processes. *Lecture Notes in Computer Science (LNCS).* 4293, 227 – 236. Springer Berlin / Heidelberg. MICAI 2006: Advances in Artificial Intelligence.

81. Schaal S. y Atkeson C. (1994). Robot juggling: An implementation of memory-based learning. *Control Systems Magazine*, 14, 57 – 71.

82. Schneider A. (2006). Decentralized Control of Elastic Limbs in Closed Kinematic Chains. *The International Journal of Robotics Research,* 2006, 25(9), 913-930.

83. Sherstov A. A. y Stone P. (2005). Improving action selection in MDP's via knowledge transfer. *Proceedings 20th National Conference on Artificial Intelligence (AAAI-05)*, 9-13, Pittsburgh, USA.

84. Simmons R. y Koenig S. (1995). Probabilistic robot navigation in partially observable environments. *Proceedings of the International Joint Conference on Artificial Intelligence*, 1080-1087.

85. Singh S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8,323-339.

86. Singh S. y Cohn D. (1998). How to dynamically merge Markov decision processes. *Advances in Neural Information Systems,* 10, Jordan M. I., Kearns M. J. y Solla S. A., The MIT Press.

87. Stein G. y Barratt M. (1989). *Developments in control theory*, Scientific Honeyweller, Systems and Research Center.

88. Still S. y Tilden M. W. (1998). Controller for a four legged walking Machine. *Neuromorphic Systems: Engineering Silicon from Neurobiology*, Smith S. y Hamilton A., Singapure, World Scientific, 138–148.

89. Sucar L. E. (2004). Parallel Markov Decision Processes, *PGM'04, Proceedings of the Second European Workshop on Probalistic Graphical Models*, Lucas P., Lorentz Center, Leiden, the Netherlands: 201 – 208.

90. Sutton R., Barto S. y Andrew G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

91. Sutton R., Precup D. y Singh S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Journal of Artificial Intelligence*, 112, 181 – 211.

92. Tham C. K. y Prager R. W. (1994). A modular Q-Learning architecture for manipulation task decomposition. *Proceedings of the Eleventh International Conference*.

93. Thrun S. (2000). Probabilistic algorithms in robotics. *AI Magazine*, 21, 93-109.

94. Uchibe E., Asada M. y Hosoda K (1995). *Cooperative Behavior Acquisition in Multi Mobile Robots Environment by Reinforcement Learning Based on State Vector Estimation.* Tesis de ingeniería en sistemas, Dept. of Adaptive Machine Systems, Graduate School of Engineering, Osaka University, Japón.

95. Uchibe E., Asada M. y Hosoda K. (1996a). Behavior Coordination for a Mobile Robot Using Modular Reinforcement Learning. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1329-1336.

96. Uchibe E., Asada M., Noda S., Takahashi Y. y Hosoda K. (1996b). Vision-Based Reinforcement Learning for RoboCup: Towards Real Robot Competition. *Proc. of IROS 96 Workshop on RoboCup*.

97. Vargas B., Morales E., Morales R. (2002). Uso de aprendizaje computacional para extraer modelos del estudiante en el problema del péndulo invertido. *Procedente del XI Congreso Internacional de Computación, Avances en Ciencias de la Computación e Ingeniería de Cómputo* (CIC2000), 1, 307 – 318.

98. Weidemann H. J. Pfeiffer F. y Eltze J. (1994). The six-legged TUM walking robot. *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems '94. 'Advanced Robotic Systems and t he Real World'*, IROS '94. 2, 1026 –1033.

Referencias en línea

[URL_Historia] Antecedentes históricos: http://www.oni.escuelas.edu.ar/2001/bs-as/hombre-vs-maquina/histoori.htm. Fecha de visita: Octubre 2006.

[URL_Estabilidad] Estabilidad y control: http://wings.avkids.com/Libro/Controls/intermediate/stability-01.html. Fecha de visita: Octubre 2006.

[URL_Robotica1] Robótica: http://www.monografias.com/trabajos31/robotica/robotica.shtml. Fecha de visita: Noviembre 2006.

[URL_Robotica2] Robótica: www.blogger.com/feeds/10188383334870456O5/posts/default. Fecha de visita: Noviembre 2006.

[URL_Metas] Modular robotics and Robot locomotion: http://155.69.254.10/users/risc/www/loco-planar.html. Fecha de visita: Febrero 2007.

[URL_Brooks_Robot] Fast, Cheap, and Out of Control: Stills: http://www.sonypictures.com/classics/fastcheap/stillsclips/stills/robot-1.html. Fecha de visita: Febrero 2007.

[URL_Subsumción_Robots] Robots: http://haydn.upc.es/groups/arobots/legged_robots/robots.htm. Fecha de visita: Febrero 2007.

[URL_Fleifel] Robots Autónomos y Aprendizaje por Refuerzo: http://www.fleifel.net/ia/robotsyaprendizaje.php. Fecha de visita: Abril 2007.

[URL_Rust] Rust J. P. (1997). A Comparison of Policy Iteration Methods for Solving Continuous-State, Infinite-Horizon Markovian Decision Problems Using Random, Quasi-random, and Deterministic Discretizations: http://ssrn.com/abstract=37768. Fecha de visita: Febrero 2007.

[URL_Shakey] Shakey the Robot: http://en.wikipedia.org/wiki/Shakey_the_Robot. Fecha de visita: Abril 2007.

Apéndice A

Webots©

Esta herramienta de simulación fue desarrollada por la compañía suiza *Cyberbotics* fundada en 1998 por Oliver Michel. *Cyberbotics* desarrolla herramientas de simulación como el simulador del robot *Aibo* que fue realizado para la compañía *Sony©.* También desarrolla y comercializa *Webots*, el cual permite realizar un rápido "prototipado" y evaluar software para robots móviles simulados. Desde 1996, *Webots* se convirtió en un software de referencia usado por alrededor de 250 universidades y centros de investigación de todo el mundo.

El software de simulación de robótica móvil *Webots* proporciona un ambiente para modelar, programar y simular robots móviles. *Webots* permite programar los robots en C, C++ y Java, así como transferir controles a robots móviles físicos, incluyendo los robots *Aibo®, Lego, Mindstorms®, Khepera®, Koala®* y *Hemisson®*.

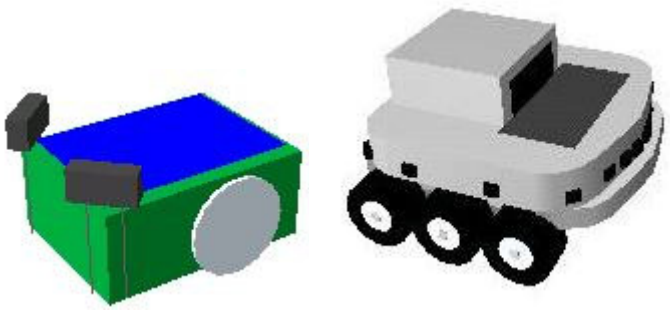


Figura A.1) Prototipos de robots con llantas construidos en el ambiente Webots

Webots es un poderoso simulador de robots en 2 y 3 dimensiones. Algunas de sus versiones permiten simular cualquier robot que utilice dos ruedas

además de una gran variedad de robots móviles terrestres (ver figura A.1). Los robots simulados son útiles para investigar el comportamiento de los robots móviles antes de construirlos físicamente, o incluso sin tener que construirlos realmente, garantizando así la realización de pruebas a bajo costo.

Este tipo de herramientas tiene un costo que oscila entre 2,400 y 10,000 dólares americanos.

Para simular un robot móvil utilizando Webots, se requiere construir un editor 3D en lenguaje VMRL. Este editor permite modelar cualquier robot que utilice dos ruedas y accesorios tales como sensores infra-rojo y ultrasónicos, cámaras de video, emisores y receptores infrarrojos y ondas de radio.

Además, es posible simular diversos escenarios, con una amplia variedad de características (ver figura A.2).

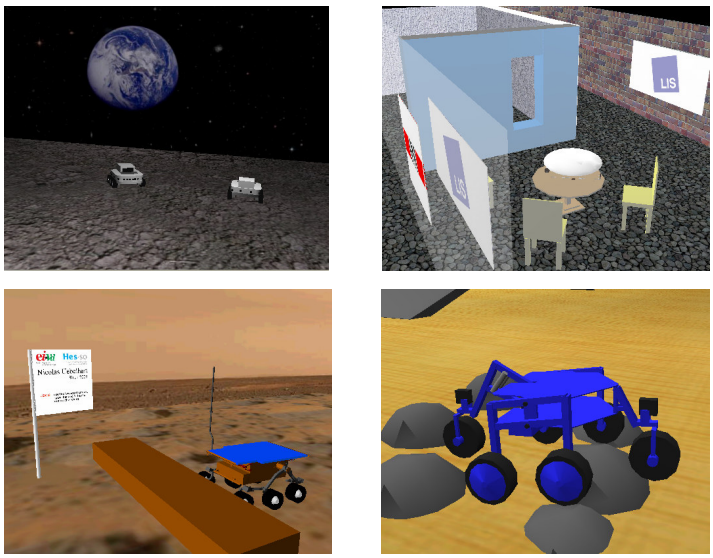


Figura A.2) Distintos escenarios en Webots

Si se desea conocer más acerca de las características del ambiente Webots, puede consultarse la siguiente liga:

http://www.cyberbotics.com/products/webots/

Apéndice B

Matriz de transición de los MDPs, MDP_UP, MDP_DOWN1 y MDP_DOWN2

Tabla B.1. Acción 1 = Arriba 1a Articulación

Edos/Edos	1	2	3	4	5	6	7	8	9
1	0.7	0	0	0.2	0	0	0.1	0	0
2	0	0.7	0	0	0.2	0	0	0.1	0
3	0	0	0.7	0	0	0.2	0	0	0.1
4	0.7	0	0	0.2	0	0	0.1	0	0
5	0	0.7	0	0	0.2	0	0	0.1	0
6	0	0	0.7	0	0	0.2	0	0	0.1
7	0.1	0	0	0.7	0	0	0.2	0	0
8	0	0.1	0	0	0.7	0	0	0.2	0
9	0	0	0.1	0	0	0.7	0	0	0.2

Tabla B.2. Acción 2 = Abajo 1a Articulación

Edos/Edos	1	2	3	4	5	6	7	8	9
1	0.2	0	0	0.7	0	0	0.1	0	0
2	0	0.2	0	0	0.7	0	0	0.1	0
3	0	0	0.2	0	0	0.7	0	0	0.1
4	0.1	0	0	0.2	0	0	0.7	0	0
5	0	0.1	0	0	0.2	0	0	0.7	0
6	0	0	0.1	0	0	0.2	0	0	0.7
7	0.1	0	0	0.2	0	0	0.7	0	0
8	0	0.1	0	0	0.2	0	0	0.7	0
9	0	0	0.1	0	0	0.2	0	0	0.7

Tabla B.3. Acción 3 = Arriba 2a Articulación

Edos/Edos	1	2	3	4	5	6	7	8	9
1	0.7	0.2	0.1	0	0	0	0	0	0
2	0.7	0.2	0.1	0	0	0	0	0	0
3	0.1	0.7	0.2	0	0	0	0	0	0
4	0	0	0	0.7	0.2	0.1	0	0	0
5	0	0	0	0.7	0.2	0.1	0	0	0
6	0	0	0	0.1	0.7	0.2	0	0	0
7	0	0	0	0	0	0	0.7	0.2	0.1
8	0	0	0	0	0	0	0.7	0.2	0.1
9	0	0	0	0	0	0	0.1	0.7	0.2

Tabla B.4. Acción 4 = Abajo 2a Articulacion

Edos/Edos	1	2	3	4	5	6	7	8	9
1	0.1	0.7	0.2	0	0	0	0	0	0
2	0.1	0.2	0.7	0	0	0	0	0	0
3	0.7	0.2	0.1	0	0	0	0	0	0
4	0	0	0	0.1	0.7	0.2	0	0	0
5	0	0	0	0.1	0.2	0.7	0	0	0
6	0	0	0	0.7	0.2	0.1	0	0	0
7	0	0	0	0	0	0	0.1	0.7	0.2
8	0	0	0	0	0	0	0.1	0.2	0.7
9	0	0	0	0	0	0	0.7	0.2	0.1

Tabla B.5. Acción5 = No hacer nada

Edos/Edos	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	1

Apéndice C

Matrices de recompensa del MDP MDP_C

Tabla C.1. Matriz de recompensa para la meta 9

	Acciones															
Estados	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0

37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
42	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
44	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
45	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
51	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
58	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
61	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
63	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
65	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
66	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
67	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
68	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
72	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
73	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
74	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
81	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla C.2. Matriz de recompensa para la meta 55

	Acciones															
Estados	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
51	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
55	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
56	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
58	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
59	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
61	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
63	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
65	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
66	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
67	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
68	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
72	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
73	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
74	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
81	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Apéndice D

Matrices de recompensa del MDP MDP_M

Tabla D.1. Matriz de recompensa para la meta 2

	Acciones															
Estados	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla D.2. Matriz de recompensa para la meta 35

	Acciones															
Estados	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0

Apéndice E

Políticas de los MDPs MDP_C y MDP MDP_M

Políticas para el MDP_C

Tabla E.1. Política

meta estado 9

Estado	Acción
1	16
2	16
3	15
4	16
5	4
6	4
7	16
8	4
9	4
10	4
11	4
12	3
13	4
14	4
15	4
16	2
17	4
18	4
19	12
20	12
21	11
22	12
23	15
24	11
25	12
26	10
27	11
28	4
29	4
30	3
31	4
32	4
33	4
34	2
35	4
36	4
37	4
38	4
39	3
40	4
41	4
42	4
43	2
44	4
45	4
46	4
47	4
48	3
49	4
50	10
51	3
52	2
53	2
54	1
55	8
56	8
57	7
58	8
59	14
60	7
61	8
62	6
63	7
64	4
65	4
66	3
67	4
68	7
69	3
70	4
71	2
72	3
73	4
74	4
75	3
76	4
77	10
78	3
79	4
80	2
81	3

Tabla E.2. Política

meta estado 55

Estado	Acción
1	9
2	9
3	15
4	9
5	9
6	13
7	14
8	9
9	13
10	9
11	9
12	9
13	9
14	9
15	9
16	9
17	9
18	9
19	12
20	9
21	11
22	9
23	9
24	9
25	10
26	9
27	9
28	9
29	9
30	9
31	9
32	9
33	9
34	13
35	13
36	9
37	9
38	9
39	9
40	9
41	9
42	9
43	9
44	9
45	9
46	11
47	11
48	9
49	9
50	9
51	9
52	9
53	9
54	9
55	9
56	9
57	7
58	9
59	9
60	5
61	6
62	1
63	5
64	9
65	9
66	9
67	9
68	9
69	9
70	1
71	1
72	1
73	4
74	1
75	3
76	1
77	1
78	1
79	2
80	1
81	1

Políticas para el MDP_M.

Tabla E.3. Política

meta estado 2

Estado	Acción
1	7
2	7
3	7
4	7
5	9
6	9
7	7
8	7
9	7
10	7
11	9
12	9
13	7
14	7
15	7
16	7
17	7
18	7
19	7
20	7
21	7
22	7
23	7
24	7
25	14
26	14
27	15
28	15
29	13
30	13
31	14
32	14
33	15
34	15
35	13
36	13

Tabla E.4. Política

meta estado 35

Estado	Acción
1	4
2	4
3	4
4	4
5	3
6	3
7	4
8	4
9	4
10	4
11	3
12	3
13	12
14	12
15	10
16	10
17	10
18	10
19	12
20	12
21	10
22	10
23	10
24	10
25	8
26	8
27	10
28	10
29	10
30	10
31	8
32	8
33	10
34	10
35	10
36	10

Apéndice F

Kit BH3-R Walking Robot

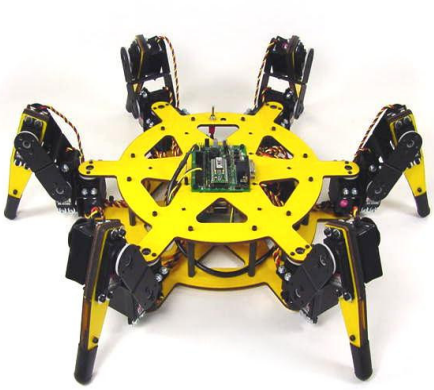


Figura F.1. BH3-R Walking Robot

El cuerpo circular simétrico de este robot lo hace ser un caso de estudio interesante y desafiante para la rama de robots caminantes. Cada pata que compone a este robot tiene 3 DOFs y aunado a su morfología proporciona una gran flexibilidad para caminar en cualquier dirección. El robot utiliza 18 servos HS-475 en total, 3 por cada pata. El *kit* incluye todo lo necesario para armar y hacer funcional al robot: electrónica, servos y piezas físicas de patas y cuerpo del robot.

El robot está hecho de componentes ultra-resistentes de Lexan, cortados con laser y hechos con moldes a la medida.

Características del robot

Control de movimiento de servo = manejo local de lazo cerrado
Movilidad = omnidireccional
Número de patas = 6
Grados de libertad por pata = 3
Velocidad de movimiento = 10"/s
Altura (cuerpo) = 2.00"
Altura (total) = hasta 6.25"
Anchura (cuerpo) = 8.50"
Anchura (total) = 18.00"
Peso (sin las baterías) = 3lb 13oz.
Separación de tierra = hasta 4.00"

Electrónica

El *kit* viene con la Atom Bot Board, y BASIC atom de 28 pines. Todo ello permite enviar información para la generación de pulsos hacia los servos y secuencias de movimiento. El lenguaje de programación (escrito por Laurent Gay) permite que el robot camine con velocidad variable, en cualquier dirección o rote o cualquier combinación de esos dos. La altura de la elevación y del tamaño de paso de la pata es ajustable, así como la velocidad del paso de caminado.

Existen ya algunos comportamientos preprogramados. PowerPod es el programa de Windows que automatiza totalmente la creación del programa para el procesador de la Atom Bot Board.

Las opciones del control son: autónomo, mando a distancia PS2, control vía remota, y control serial desde una PC, PDA, o desde cualquier otro microcontrolador compatible.

Tarjeta controladora Atom Bot Board

Figura F.2. Atom Bot Board

La tarjeta Atom Bot Board, es la tarjeta recomendada para operar con el BASIC Atom, microcontrolador de 24 o 28 pines. Esta tarjeta es adecuada para desarrollo de proyectos de robótica. Esta tarjeta tiene integrada una bocina, tres botones y 3 LEDs, un controlador de puerto serial Sony PS2, un botón para reiniciar la tarjeta, lógica y entradas para alimentar servo, un bus de entrada-salida con energía y tierra, y un regulador de 5Vdc de 250mA. Pueden ser conectados hasta 20 servos directamente. Además también tiene varios plug-play.

Microcontrolador BASIC Atom con 28 pines

El BASIC Atom tiene un gran alcance y es muy confiable para ser utilizado en proyectos de robótica móvil. Los conectores de la Bot Board completan el acceso total a todos los pines de entrada-salida. Este microcontrolador es programado usando una variante de lenguaje Basic. El software para programar este microcontrolador puede ser obtenido de manera gratuita. También puede usarse dicho software para depurar el funcionamiento de un programa cargado en el microcontrolador.

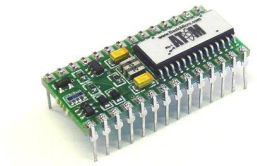


Figura F.3. Basic Atom

Tarjeta controladora de servomotores: SSC-32 Servo Controller

Éste es un controlador de servos el cual tiene 32 canales para control de servomotores con una velocidad de respuesta de 1 useg. Puede establecerse una comunicación bidireccional con comandos de una variante del lenguaje C. Pueden ser programados movimientos sincronizados independientes o en "grupo" de los servos que estén conectados a este controlador. Es el dispositivo recomendado para controlar los servos del hexápodo utilizado en esta tesis.

Figura F.4. SSC-32

Apéndice G

Servomotor HS-475 de HiTec

Figura G.1. Servomotor

Este es el tipo de servomotor utilizado con el BH3-R Walking Robot, el cual tiene las siguientes características.

Torque 4.8/6.0v :	4.3 / 5.5 kg.
Vel. 4.8/6.0v :	0.23 / 0.18 sg.
Tamaño :	41 x 20 x 37mm
Peso :	39 gr.

Micro-interruptor tipo 1: Push button switch

Figura G.2. Micro-interruptor tipo 1

Micro-interruptor utilizado en el segmento 1 de una pata del robot hexápodo, para obtener la posición de dicho segmento.

Características:

Temperatura: -20ºC - +70ºC

Humedad relativa: 40% - 95%

Voltaje: AC 250V (50Hz)

Resitencia al contacto: 200 MΩMax

Vida: 10,000 Veces

Micro-interruptor tipo 2: Tact Switch

Figura G.3. Micro-interruptor tipo 3

Micro-interruptor utilizado en el segmento 2 de una pata del robot hexápodo, para obtener la posición de dicho segmento.

Características:

Temperatura: -10℃ - +60℃

Voltaje: AC 250V (50Hz)

Fuerza de activación: 130 – 250 gf

Resitencia al contacto: 100 MΩMax

Vida: 100,000 veces

Micro-interruptor tipo 3: **Micro switch**

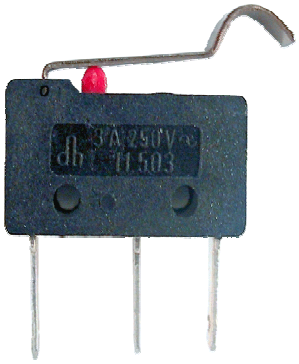


Figura G.4. Micro-interruptor tipo 2

Micro-interruptor utilizado en el segmento 3 de una pata del robot hexápodo, para obtener la posición de dicho segmento.

Características:

Temperatura: -40-70℃

Voltaje: AC 250V (50Hz)

Resitencia al contacto: 200 MΩMax

Fuerza de activación: 60-70 gf

Vida:100,000 veces

Apéndice H

Conexiones

Conexión de la tarjeta SSC-32 Servo Controller y la tarjeta Atom Bot Board, con ilustración de dos de los 12 micro-interruptores utilizados.

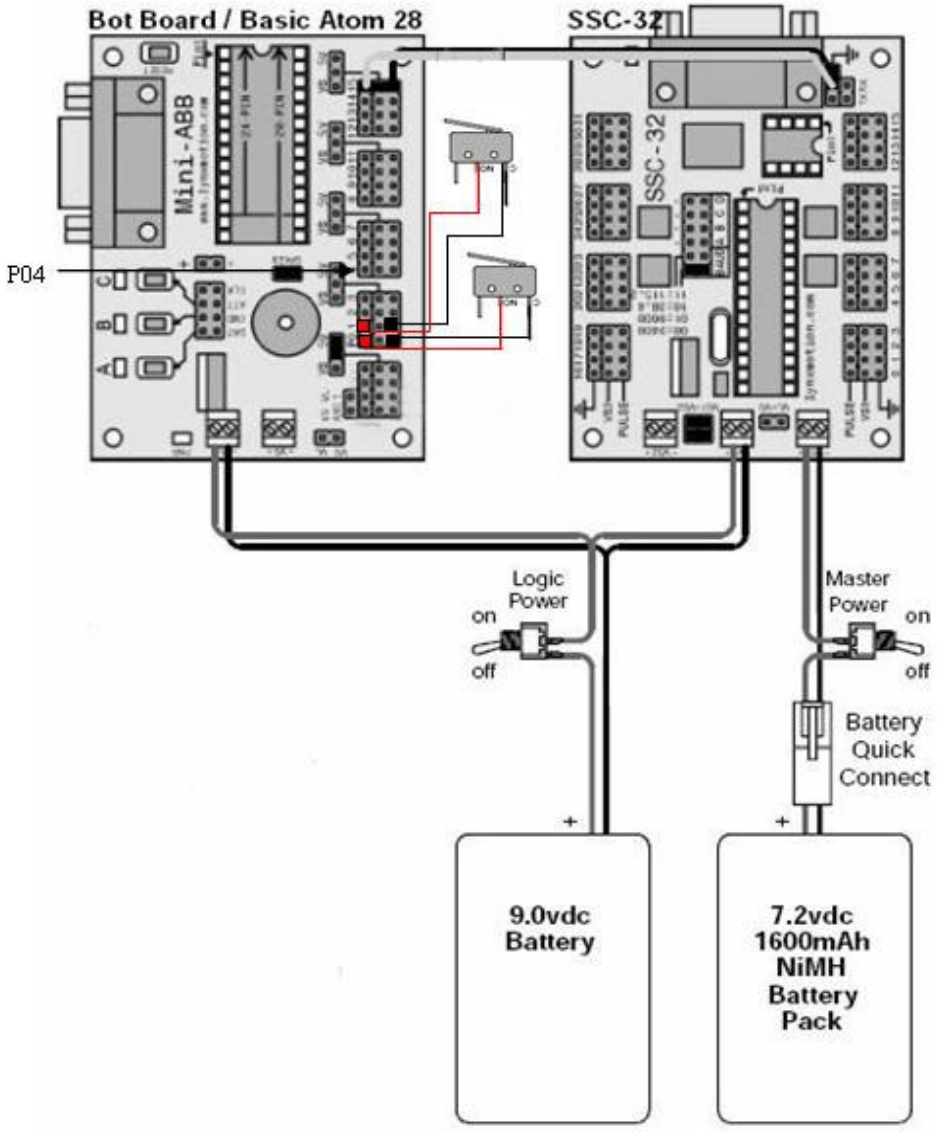


Figura H.1) Conexión de las tarjeta SSC-32 y la tarjeta Atom Bot Board

Un problema al que nos enfrentamos en esta tesis fue realizar una conexión adecuada para comunicar las tarjetas SSC-32 y la Atom Bot Board, para que funcionaran de manera autónoma. En la bibliografía consultada existe amplia información del modo de conectar dichas tarjetas para que funcionen de manera controlada por un Jostik. Sin embargo no hay mucha información disponible la manera adecuada de conectar dichas tarjetas para que funcionen de manera autónoma.

La figura H.1 muestra el diagrama de conexión entre la tarjeta electrónica de control Atom Bot Board y la tarjeta controladora de servomotores SSC-32. La figura H.2 muestra a detalle la conexión de dos micro-interruptores a la tarjeta Atom Bot Board, con la respectiva adaptación que se hizo para su buen funcionamiento. Esta adaptación incluye el cableado de una resistencia de 10k, y la colocación de un jumper para asegurar el paso de corriente a los micro-interruptores.

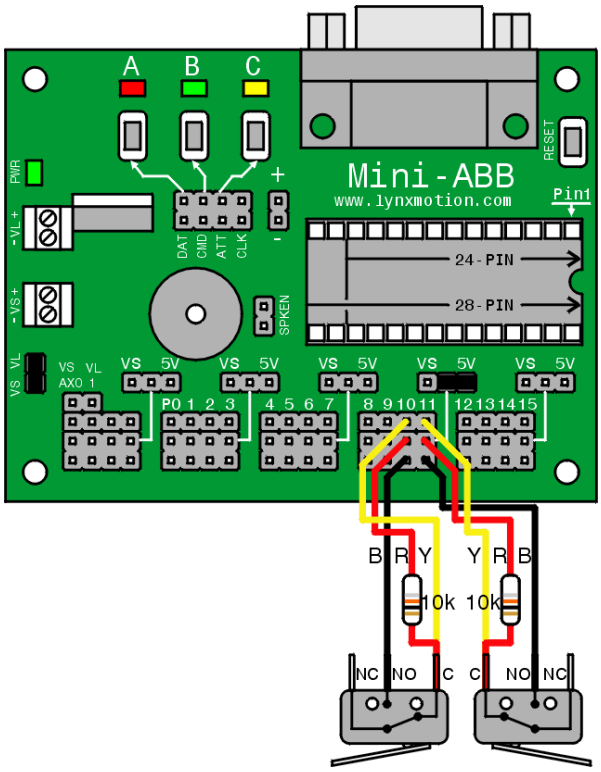


Figura H.2) Conexión de los micro-interruptores a la tarjeta Atom Bot Board