BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

Facultad de ciencia de la computación

TESINA

Para obtener el título de:

ING. EN CIENCIAS DE LA COMPUTACIÓN

Diseño y control de un robot escalador móvil articulado terrestre

Presenta: Jorge Ocampo Jiménez

Asesor : Dr. Manuel I. Martín Ortiz.

Mayo del 2010

Puebla, Puebla.

ÍNDICE

Resumen

Este documento se presenta el diseño, construcción y control de comportamiento para un robot multi-articulado de locomoción diferencial. El control fue desarrollado utilizando dos algoritmos de aprendizaje por refuerzo: las técnicas de programación dinámica y diferencia temporal. El sistema de control fue probado en un simulador y posteriormente implementado en el robot real propuesto. El desempeño entre los dos tipos diferentes de algoritmos, se probó usando un obstáculo vertical el cual el robot tuvo que sortear en el mínimo número de movimientos obtenidos por los algoritmos de aprendizaje por refuerzo.

1. Introducción

Este trabajo se realizó con el fin de proponer un diseño y control para robots ``tipo-serpiente". Estos robots son caracterizados por tener varias articulaciones, lo que le permite tener un espacio de trabajo aumentado, no solo a un espacio horizontal, si no también vertical, además de proporcionar en algunos casos, diferentes tipos de locomoción. En este trabajo se propuso un diseño que reduce el número de eslabones del robot.

El problema principal en los robots tipo serpiente, es su falta de autonomía en tareas de locomoción, la mayoría de estos robots son controlados por mas de un operador, debido a que poseen un gran número de grados de libertad y a la complejidad de la tarea, lo que repercute en su uso en situaciones donde el operador no tiene noción del estado del robot, así como al uso excesivo de recursos humanos.

En el diseño de un robot, siempre se tiene el compromiso entre funcionalidad y consumo de potencia. En este proyecto, se buscó poder sortear obstáculos que generalmente son difíciles o imposibles para un robot móvil del tipo diferencial, como lo son los obstáculos verticales.

El diseño del robot fue pensado para realizar un sistema de control de complejidad baja, este diseño se aplico con el fin de demostrar si realmente el diseño era viable.

Se pensaron en diversos métodos para el control del robot. El objetivo siempre fue que el robot fuera capaz de descubrir la estrategia para poder subir los obstáculos de manera autónoma. Debido al ambiente incierto se optó por un algoritmo de aprendizaje por refuerzo, el cual exploraría el ambiente y nos daría información acerca de las frecuencias consecuentes a las acciones del robot, con lo que podría funcionar como las observaciones para un algoritmo de aprendizaje por refuerzo.

El objetivo principal del desarrollo del control, fue lograr implementar un comportamiento en el robot real el cual le permitiera sortear obstáculos verticales automáticamente, y a su vez, proporcionarnos información acerca de las ventajas y desventajas de usar un algoritmo de aprendizaje por refuerzo. Se utilizaron las técnicas de programa-

ción dinámica y diferencia temporal. El sistema de control fue probado en un simulador, debido a las limitaciones del controlador utilizado en el robot real, esto con la finalidad de implementar la política resultante en el robot real. El desempeño entre los dos tipos diferentes de algoritmos se midió utilizando un obstáculo vertical el cual el robot tuvo que sortear en el mínimo número de movimientos alcanzados por los algoritmos de aprendizaje por refuerzo.
Una vez probado el control, se implemento la política obtenida en el robot real, con lo cual se logro el comportamiento deseado.

Finalmente, además de obtener un novedoso prototipo de robot “tipo serpiente” autónomo, se obtuvo un análisis de dos diferente algoritmos de aprendizaje por refuerzo ante la situación de estados desconocidos.

El contenido de este documento es el siguiente: en el capitulo 2 de este documento, presentamos la descripción detallada de los elementos básicos de aprendizaje por refuerzo. En el capitulo tres, se planteó la problemática resuelta en este trabajo. El capitulo cuatro, muestra la metodología seguida para la construcción y control del robot. En el capitulo 5, se encuentran las pruebas realizadas dirigidas a la tarea de locomoción del robot. Finalmente, el capitulo seis presenta las conclusiones de este trabajo.

2. Aprendizaje por refuerzo.

La idea de aprendizaje por refuerzo es la de adquirir conocimiento mediante la interacción con el ambiente [2]. Al ejercitar nuestros medios de interacción con el ambiente (extremidades, objetos, etc), producimos una abundancia de información acerca de causa y efecto, descubrimos las consecuencias de las acciones y como alcanzar metas. Aprendemos cómo nuestro ambiente reacciona a lo que hacemos, y buscamos influenciar lo que pasa a través del comportamiento. Aprender de estas interacciones es la idea fundamental de la mayoría de las teorías de aprendizaje e inteligencia.

2.1. Elementos del aprendizaje por refuerzo.

Al aprendiz y tomador de decisiones se le llama agente. A la interacción con cualquier objeto que no forma parte del agente se le llama ambiente. El agente y el ambiente interaccionan mutuamente de manera continua. El ambiente presenta recompensas, las cuales son valores numéricos especiales las cuales el agente trata de maximizar a través del tiempo. A la especificación completa de un agente en un ambiente se le llama tarea.

Más allá del ambiente del agente se pueden identificar los principales elementos de los sistemas de aprendizaje por refuerzo: una política, la función de recompensa, una función de valor y opcionalmente un modelo del ambiente.

Una política define el modo en que el aprendizaje del agente se realiza durante el tiempo. La política es el mapeo de los estados percibidos del ambiente a las acciones. En algunos casos la política puede ser una función o relación. La política es el núcleo del aprendizaje por refuerzo en un agente, en el sentido de que la política es suficiente para determinar el comportamiento. En general, las políticas son estocásticas.

La función de recompensa es la encargada de indicar que tan bueno es un sensado inmediato, y la función de valor especifica que tan bien se comporta a largo plazo. El valor de un estado es la cantidad total de recompensa que un agente puede esperar acumular al futuro comenzando desde ese estado. Mientras las recompensas determinan la

deseabilidad de los estados, los valores indican la deseabilidad a largo plazo después de tomar en cuenta los estados que probablemente seguirán y las recompensas permitidas en dichos estados.

Sin recompensas no podría haber valores, y el único propósito de estimar los valores es adquirir mas recompensa. La elección de acciones esta basada en el valor de deliberación. Desafortunadamente, es mucho más difícil determinar los valores que determinar las recompensas. Las recompensas son básicamente dadas por el ambiente, pero los valores deben de ser estimados y re-estimados de las secuencias de observaciones que los agentes hacen a lo largo de su tiempo de vida. El componente mas importante de los algoritmos de aprendizaje por refuerzo la estimación los valores.

La característica más distintiva del aprendizaje por refuerzo comparada con otros tipos es la de aprender usando información que evalúa las acciones más que las instrucciones dadas por correctas acciones. Esto es lo que crea la necesidad por exploración activa, búsquedas de prueba y error para lograr un buen comportamiento. La evaluación de la retroalimentación indica que tan buena es la acción tomada, pero no cual es la mejor o la peor acción posible. Evaluar la retroalimentación es la base de los métodos para funciones de optimización. La retroalimentación puramente instructiva, por otro lado, indica la acción correcta a tomar, independientemente de la acción tomada actualmente. Este tipo de retroalimentación es la base del aprendizaje supervisado, el cual incluye grandes partes de clasificación de patrones, redes neuronales, y sistemas de identificación. Estos dos tipos de retroalimentación son bastante distintos: evaluar la retroalimentación depende totalmente de la acción tomada, donde la retroalimentación instructiva es independiente de la acción tomada.

2.2. ¿Qué es aprendizaje por refuerzo?

Aprendizaje por refuerzo es aprender que hacer para maximizar una señal de recompensa numérica. El aprendiz no se le dice que acciones tomar, como en la mayoría de las formas de aprendizaje automático, en sustitución, se debe de descubrir cuales acciones dan paso a la mayor recompensa. Las acciones tienen efecto no solo en las recompensas inmediatas si no además en situaciones posteriores y en las subsecuentes recom-

pensas. Las características distintivas del aprendizaje por refuerzo son la búsqueda de basada en prueba - error y recompensa retardada [2].

El aprendizaje por refuerzo no está definido por la caracterización del método de aprendizaje, si no por la caracterización del problema de aprendizaje. Cualquier método es apropiado para resolver un problema. La idea básica es simplemente capturar los aspectos más importantes de un problema real para un agente que se enfrenta a la interacción con el ambiente para alcanzar una meta. Claramente, dicho agente debe de ser capaz de sensar el estado del ambiente hasta cierto punto y debe de ser capaz de tomar acciones que afecten el estado. El agente también debe de tener una meta o metas relacionadas al estado en que se encuentra.

Uno de los retos que surgen en el aprendizaje por refuerzo es el costo beneficio entre exploración y explotación. Para obtener una recompensa grande, un agente de aprendizaje por refuerzo debe preferir acciones que ha utilizado en el pasado y ha encontrado efectivas para producir una recompensa. Pero para descubrir dichas acciones, tiene que intentar acciones que no ha seleccionado anteriormente. El agente debe de explotar lo que aprendió con anterioridad, con el objetivo de obtener una recompensa, pero también debe explorar con el fin de hacer una mejor selección de acciones en el futuro. El dilema se encuentra en que ni la exploración ni la explotación pueden ser perseguidas exclusivamente sin fallar en la tarea. El agente debe de intentar una variedad de acciones y progresivamente favorecer las mejores. En una tarea estocástica, cada acción debe de intentarse varias veces para ganar una estimación confiable de la recompensa esperada.

Otra de las características clave en el aprendizaje por refuerzo es la de considerar explícitamente todo el problema coa mo el de un agente dirigido a metas interactuando con un ambiente incierto.

El aprendizaje por refuerzo comienza con un agente completamente interactivo orientado a metas. Todos los agentes de aprendizaje por refuerzo tienen metas explicitas, pueden sensar aspectos del ambiente, y pueden elegir acciones para influenciar estos ambientes. Es usualmente asumido desde el principio que el agente debe de operar a pesar de una significativa incertidumbre con el que se enfrenta en el ambiente. Cuando el aprendizaje por refuerzo requiere un plan, tiene que ser dirigido a la interacción entre la

planeación y la selección de acciones en tiempo real, así como a las preguntas de cómo los modelos del ambiente son adquiridos y mejorados.

2.3. Métodos de acción-valor

Se denota al valor real de la acción *a* como $Q_*(a)$, y el valor estimado en el tiempo *t* como $Q_t(a)$. El valor verdadero de la acción se refiere a la media del la recompensa recibida cuando la acción es seleccionada. Una manera de estimar esta recompensa es promediar las recompensas recibidas cuando la acción fue seleccionada. Si al tiempo *t*, la acción ha sido elegida k_a veces antes de *t*, siendo las recompensas $r_1, r_2, ..., r_{k_a}$, entonces el valor estimado es

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_a}}{k_a}. \qquad (2.1)$$

Si $k_a = 0$, definimos a $Q_t(a)$ con cualquier valor. Si k_a tiende a infinito, por la ley de largos números, $Q_t(a)$ converge a $Q_*(a)$. A este método se le llama método de promedio simple para estimar los valores de las acciones, ya que es el promedio simple de la muestra de recompensas relevantes.

La regla de selección de acción mas simple con valores altos de selección de acción es seleccionar al tiempo *t* acciones voraces, a^*, para la cual

$$Q_t(a^*) = \max_a Q_t(a). \qquad (2.2)$$

Este es un método que siempre explota el conocimiento actual para maximizar la recompensa inmediata. Una alternativa simple es comportarse voraz la mayoría del tiempo, pero de vez en cuando, con una pequeña probabilidad ε, seleccionar una acción aleatoria, uniformemente e independiente de los valores-acción determinados. A estos métodos se les llama ε-voraz. Una ventaja de estos métodos es que en el límite, mientras el número de juegos se incrementa, cada acción va a ser empleada un número infinito de veces, garantizando que k_a tiende a infinito para todas las *a*, y esto asegura que todas las $Q_t(a)$ convergen a $Q_*(a)$. La probabilidad de seleccionar la acción óptima es 1-ε.

La ventaja de los ε-voraz sobre los métodos voraces depende de la tarea. Si la varianza es cero, entonces el método voraz sabrá el verdadero valor de cada acción después de intentarlo la primera vez. Pero aun en el caso determinístico, hay una ventaja en explorar.

2.4. Evaluación contra instrucción.

La recompensa recibida después de cada acción da algo de información acerca de que tan buena fue la acción, pero no dice nada acerca de si la acción fue correcta o incorrecta, lo cual es, si fue la mejor acción o no. Se tiene que generar de alguna forma, métodos de generación y prueba donde se puedan intentar acciones, observar los resultados, y retener los que son más efectivos. Este aprendizaje por selección, en contraste a aprender por instrucción, es el que todos los aprendizajes por refuerzo tienen que utilizar de una forma u otra.

En el aprendizaje supervisado no hay una situación en que la acción es tomada, pero si hay un conjunto largo de diferentes situaciones, en las cuales se debe de responder correctamente. El mayor problema en aprendizaje supervisado es construir un mapa de situaciones a acciones que imiten las acciones correctas especificadas por el ambiente y generalice la nueva situación correcta. Un aprendizaje supervisado no puede aprender el control de su ambiente porque sigue, más que influenciar, la información instructiva. En lugar de tratar de hacer a su ambiente comportarse de cierta manera, intenta hacerse así mismo comportarse como lo instruyo el ambiente.

2.5. Implementación incremental.

El problema con la implementación directa son la memoria y que los requerimientos computacionales crecen durante el tiempo sin límite. Cada recompensa adicional seguida de una selección de acción a requiere más memoria para almacenarla y más tiempo computacional para calcular $Q_t(a)$.

Esto no es realmente necesario, se puede actualizar las formulas para promedios de computo pequeños, y un tiempo de computación constante es requerido para procesar la

nueva recompensa. Para cada acción Q_k denota el promedio de las primeras *k* recompensas. Dado el promedio de la recompensa *(k+1),* r_{k+1}

$$\begin{aligned} Q_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} r_i \\ &= \frac{1}{k+1}\left(r_{k+1} + \sum_{i=1}^{n} r_i\right) \\ &= \frac{1}{k+1}\Big(r_{k+1} + kQ_k + Q_k - Q_k\Big) \\ &= \frac{1}{k+1}\Big(r_{k+1} + (k+1)Q_k - Q_k\Big) \\ &= Q_k + \frac{1}{k+1}\Big[r_{k+1} - Q_k\Big], \end{aligned} \qquad (2.3)$$

la cual se mantiene aun cuando *k*=0, obteniendo que $Q_1 = r_k$ para un Q_0 arbitrario. Esta implementación requiere solo memoria para Q_k y *k*, y solo un pequeño cálculo para cada recompensa. La fórmula general es:

$$NuevoEstimado \leftarrow\!\!- ViejoEstimado + TamañodePaso[Objetivo - ViejoEstimado].$$

Donde $Objetivo - ViejoEstimado$ es el error de la estimación. Este es reducido tomando un paso al objetivo.

El tamaño del paso es utilizado para describir los cambios de paso a paso en el tiempo.

Los métodos de promedios son apropiados para ambientes estacionarios, en los cuales encontramos comúnmente problemas que son no-estacionarios. En dichos casos, tiene sentido sensar los pesos de las recompensas recientes mas estrictamente que las antiguas. Una de las maneras más populares es usar un parámetro de paso constante. Actualizando el promedio se obtiene:

$$Q_{k+1} = Q_k + \alpha\Big[r_{k+1} - Q_k\Big] \qquad (2.4)$$

Donde el peso del tamaño alfa que se encuentra entre 0 y 1 es constante, Q_k es el promedio pesado de las recompensas pasadas y Q_0 es el valor inicial estimado. Desarrollando Q_k:

$$\begin{aligned} Q_k &= Q_{k-1} + \alpha\left[r_k - Q_{k-1}\right] \\ &= \alpha r_k + (1-\alpha)Q_{k-1} \\ &= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 Q_{k-2} \\ &= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 \alpha r_{k-2} + \\ &\quad \cdots + (1-\alpha)^{k-1}\alpha r_1 + (1-\alpha)^k Q_0 \\ &= (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i. \end{aligned} \quad (2.5)$$

A este desarrollo se le llama promedio pesado, ya que se obtiene que

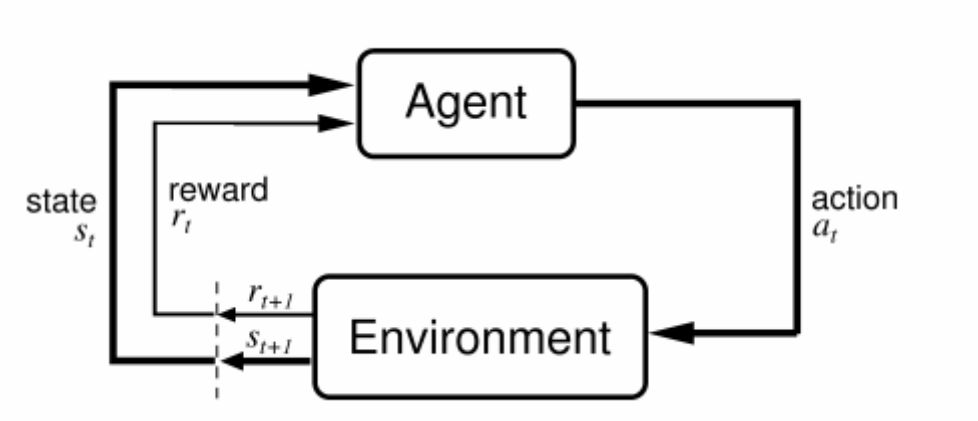
$(1-\alpha)^k + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} = 1$ *(2.6)*

Si se elige un tamaño variable de paso, se debe garantizar la convergencia con:

$$\sum_{k=1}^{\infty} \alpha_k(a) = \infty \qquad \text{and} \qquad \sum_{k=1}^{\infty} \alpha_k^2(a) < \infty. \quad (2.7)$$

2.6. El papel del agente en aprendizaje por refuerzo.

En computación, un agente interactúa con el ambiente en una secuencia de pasos *t* discretos, *t*=0,1,2,…. En cada paso *t*, el agente sensa una representación del ambiente s_t que pertenece a *S*, donde *S* es el conjunto de posibles estados, dependiendo del estado, el agente selecciona una acción a_t pertenece a $A(s_t)$, donde $A(s_t)$ es el conjunto de posibles acciones en el estado s_t. Cuando se realiza la transición de estado como consecuencia de una acción, el agente recibe la recompensa numérica r_{t+1} pertenece a *R*.



A cada paso que se realiza, el agente almacena las probabilidades de llegar a ese estado seleccionando con una acción. A este mapeo se le llama política, y se denota con π_t, donde π_t *(s,a)* es la probabilidad de que si se realiza la acción $a=a_t$ si el estado es igual a *s*. Los métodos de aprendizaje por refuerzo especifican como el agente cambia su política con la experiencia. La meta final del agente es maximizar la recompensa que recibe a largo plazo.

La recompensa se usa como señal de meta para el agente. Al diseñar soluciones por medio de aprendizaje por refuerzo, se deben de proveer las recompensas de manera que se maximicen cuando el agente se acerque a sus metas. La señal de recompensa es la manera de comunicar al agente que se quiere alcanzar y no del como se alcanza.

Las recompensas son dadas por el ambiente, y no por el agente. La razón de esto es que la meta final del agente debe de ser algo en lo que se tenga un control imperfecto.

2.6.1. Retorno.

Sean r_{t+1}, r_{t+2},... recompensas recibidas después de algún tiempo t, se define como regreso esperado R_t como

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T, \quad (2.8)$$

Donde *T* es el tiempo final, R_t es el valor que se busca maximizar.

La interacción con el ambiente puede dividirse en subsecuencias llamadas episodios. Cada episodio termina con un estado terminal, y es seguido por un reinicio de estados

iniciales. Tareas con episodios son llamadas tareas episódicas. Si la tarea no puede dividirse naturalmente en episodios identificables, es llamada tarea continua.

Para representar los intentos de acción de un agente se define un factor de descuento, donde obtenemos el regreso esperado con descuento se define

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \quad (2.9)$$

Donde γ es un parámetro entre 0 y 1 llamado radio de descuento, el cual determina el valor de las futuras recompensas.

2.6.2. Propiedad de Markov.

En aprendizaje por refuerzo, se toma como premisa que el problema cumple la propiedad markoviana.

Suponiendo que existe un numero finito de estados y de valores de recompensa, tomamos un ambiente *t* con la acción *a*, y queremos saber como reacciona en el tiempo *t+1*. En el caso más general, la respuesta en el tiempo *t+1* dependerá de todo lo que ha pasado hasta el momento. Donde podemos expresar la dinámica como una distribución completa de probabilidad

$$Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\} \quad (2.10)$$

Si la señal de estado, posee la propiedad markoviana, el ambiente en el tiempo *t+1* solo depende del estado y la acción en el tiempo *t*

$$Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}. \quad (2.11)$$

Iterando esta ecuación, se obtiene la predicción futura de recompensas para las acciones seleccionadas. Esto nos ayuda a tener un estado como una buena base para la predic-

ción, aunque se debe recordar que no todos los problemas satisfacen completamente la propiedad de Markov.

Una tarea de aprendizaje por refuerzo la cual cumple la propiedad markoviana se le llama proceso de decisión de Markov o MDP(por sus siglas en ingles). Si el espacio de estados es finito, se le llama MDP finito, los cuales se definen como

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1}=s' \mid s_t=s, a_t=a\}. \quad (2.12)$$

La probabilidad de pasar de *s* a *s'* con la acción *a*. llamada probabilidad de transición.

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} \mid s_t=s, a_t=a, s_{t+1}=s'\} \quad (2.13)$$

A $R^a_{ss'}$ se le llama recompensa esperada.

2.6.3. Funciones de valor.

Se encarga de estimar "que tan bien" es para un agente transitar por un estado. Esto es definido en términos de futuras recompensas esperadas.

El valor de un estado siguiendo una política π, denotada como V^π se define como

$$V^\pi(s) = E_\pi\{R_t|s_t=s\} = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \;\middle|\; s_t=s\right\} \quad (2.14)$$

Donde E_π denota el valor esperado del agente siguiendo una política π. A V^π se le llama a función de estado-valor para la política π.

Similarmente, se define el valor de tomar la acción a en el estado *s* como Q^π *(s,a)*, definida por

$$Q^\pi(s,a) = E_\pi\{R_t|s_t=s, a_t=a\} = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \;\middle|\; s_t=s, a_t=a\right\} \quad (2.15)$$

Donde se le llama a Q^π función acción-valor para la política π.

Desarrollando la función de valor

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\{R_t | s_t = s\} \\
&= E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s \right\} \\
&= E_{\pi}\left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_t = s \right\} \\
&= \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_{t+1} = s' \right\} \right] \\
&= \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right], \qquad \textit{(2.16)}
\end{aligned}
$$

Obtenemos la ecuación de Bellman para V^{π}, la cual expresa la relación entre valores de un estado y estados sucesores. Esta ecuación promedia todas las posibilidades, pesando cada una dado su probabilidad de ocurrencia, y enuncia que el valor del estado inicial debe de ser igual al valor esperado del siguiente estado más la recompensa esperada a lo largo del trayecto.

2.6.4. Funciones de valores optimas.

Resolver tareas de aprendizaje por refuerzo, es encontrar la política que proporcione la mayor recompensa.

Para MDPs finitos, se define como

$$V^{*}(s) = \max_{\pi} V^{\pi}(s). \qquad \textit{(2.17)}$$

Para funciones de valor de estado y

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \qquad \textit{(2.18)}$$

para funciones de valor de acción.

Se puede escribir *Q** en términos de *V** como

$$Q^{*}(s,a) = E\left\{ r_{t+1} + \gamma V^{*}(s_{t+1}) \mid s_t = s, a_t = a \right\}. \qquad \textit{(2.19)}$$

La ecuación de Bellman óptima se escribe como:

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^{\pi^*}(s,a) \\
&= \max_a E_{\pi^*}\left\{ R_t \;\middle|\; s_t = s, a_t = a \right\} \\
&= \max_a E_{\pi^*}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s, a_t = a \right\} \\
&= \max_a E_{\pi^*}\left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_t = s, a_t = a \right\} \\
&= \max_a E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \right\} \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s') \right]. \quad \textit{(2.20)}
\end{aligned}
$$

Y para *Q**

$$
\begin{aligned}
Q^*(s,a) &= E\left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \;\middle|\; s_t = s, a_t = a \right\} \\
&= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right]. \quad \textit{(2.21)}
\end{aligned}
$$

Esta ecuación tiene una única solución independiente de la política. La ecuación de optimización de Bellman es en realidad un sistema de ecuaciones, una para cada estado. Si la dinámica del ambiente es conocida, en principio se puede resolver un sistema de ecuaciones para *V**.

Los algoritmos de programación dinámica son obtenidos al transformar las ecuaciones de Bellman en asignaciones.

En un ambiente donde la dinámica es completamente conocida, se tiene un sistema de |*S*| ecuaciones lineales simultáneas con |*S*| incógnitas. En principio se tiene una solución directa, sin embargo es costosa computacionalmente. Considérese una secuencia de $V_0, V_1, \ldots$, cada una mapeada en *S* a *R*. La aproximación inicial V_0 es elegida arbitrariamente, y cada aproximación sucesiva es obtenida usando la ecuación de Bellman para V^{π} como una regla de actualización

$$V_{k+1}(s) = E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\}$$
$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right] \quad (2.22)$$

$V_k = V^\pi$ es un punto de actualización de la regla, lo cual lo asegura la ecuación de Bellman para V^π. En general, V_k converge a V^π cuando *k* tiende a infinito. A este algoritmo se le llama iteración de política.

Si

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s). \quad (2.23)$$

Entonce la política π' es tan o más buena que la política π. Lo cual se expresa

$$V^{\pi'}(s) \geq V^\pi(s). \quad (2.24)$$

Si tenemos una nueva política π', la cual no es mejor que la política anterior π, entonces $V^\pi = V^{\pi'}$, lo cual se escribe como

$$V^{\pi'}(s) = \max_a E\left\{r_{t+1} + \gamma V^{\pi'}(s_{t+1}) \mid s_t = s, a_t = a\right\}$$
$$= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi'}(s')\right]. \quad (2.25)$$

Una vez mejorada la política usando V^π a $V^{\pi'}$, podemos hacerlo nuevamente para obtener una mejor política π''

$$\pi_0 \xrightarrow{\text{E}} V^{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V^{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} V^*$$

donde E denota evaluación e I mejora.

Una desventaja de la iteración anterior, es que cada iteración trae consigo una evaluación de la política, lo cual requerirá múltiples ciclos por cada estado. La convergencia al V^π exacto sucede en el límite. Sin embargo, el paso de evaluación puede ser truncado de maneras diversas sin perder la garantía de convergencia de la iteración de política. Un caso especial es cuando la evaluación de la política es detenida luego de un solo ciclo. Este algoritmo es llamado iteración de valor. Puede ser escrito como una simple operación de respaldo que combina la mejora de política y la operación de avaluación truncada:

$$V_{k+1}(s) = \max_a E\{r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s, a_t = a\}$$
$$= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right], \quad (2.26)$$

Inicializa V arbitrariamente

Repite
$\Delta \leftarrow 0$
Para cada $s \in S$:
$v \leftarrow V(s)$
$V(s) \leftarrow \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
hasta $\Delta < \theta$ (un número positive pequeño)

Salida: una política determinística π, en la cual
$\pi(s) = \text{argmax}_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$

2.7. Predicción TD

Dada la experiencia siguiendo una política π, el método actualiza V de V^π. Si un estado no terminal es visitado en el tiempo t, se actualiza $V(s_t)$. Los métodos de Montecarlo esperan hasta regresar a un estado que conocen para actualizar $V(s_t)$. Lo cual se expresa como:

$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right] \quad (2.27)$$

A este método se le llama alfa MC.

Los métodos TD necesitan esperar solamente hasta el siguiente paso. Una vez en $t+1$, se forma un objetivo y se realiza una actualización usando la recompensa r_{t+1} y la estimación $V(s_{t+1})$. El método TD más simple es el TD(0), el cual se expresa con

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \quad (2.28)$$

Como los métodos TD se basan en actualizar una parte de la estimación existente, decimos que son métodos bootstrapping.

Los métodos de Montecarlo utilizan un estimado de

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} \quad (2.29)$$

Como objetivo, mientras que los métodos TD utilizan a

$$E_{\pi}\left\{ r_{t+1} + \gamma V^{\pi}(s_{t+1}) \,\middle|\, s_t = s \right\} \quad (2.30)$$

como objetivo. El objetivo de Montecarlo es estimado porque el valor esperado es conocido; una muestra del retorno es usada en lugar de la muestra del retorno esperada real. Los algoritmos de aprendizaje por refuerzo DP (programación dinámica)(como lo son los MDPs) no es un estimado de los valores esperados, los cuales se asume que son conocidos por el modelo del ambiente, en su lugar se estima $V^{\pi}(s_{t+1})$, que no es conocido en la estimación actual y se utiliza $V_t(s_{t+1})$ en su lugar. El objetivo de TD es una estimación donde se muestrea los valores esperados y los usa para estimar V_t en lugar del verdadero V^{π}. Así, los métodos TD combinan el muestreo de Montecarlo con *bootstrapping* de los DP.

Una ventaja de los TD sobre los DP es que no requieren un modelo del ambiente, el cual es la recompensa y las probabilidades del siguiente estado.

La ventaja de los TD sobre los Montecarlo es que son implementados naturalmente en línea, de manera totalmente incremental. Con los métodos Montecarlo uno debe esperar hasta el final de un episodio, porque solo entonces es el regreso conocido, y con los métodos TD solo se necesita esperar el siguiente paso. Esto es muy importante, ya que, si la aplicación tiene episodios muy largos se espera demasiado para el aprendizaje, además de que existen tareas continuas las cuales no tienen ningún episodio.

2.8. Sarsa: On policy TD control.

El primer paso es aprender una función de acción-valor en lugar de de estado-valor. Para un método on-policy debemos estimar $Q^{\pi}(s,a)$ para el comportamiento actual de la política π y para todos los estados *s* y acciones *a*. Esto puede ser hecho utilizando el método para estimar V^{π} de los algoritmos TD.

Consideramos la transición de un par estado-acción a otro par estado-acción, y aprendemos los valores de los pares estado acción. Formalmente estos casos son idénticos. Los teoremas de convergencia de los algoritmos TD también se aplican a los algoritmos de valores de acción

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big] \quad (2.31)$$

La actualización es realizada después de cada transición a un estado no terminal s_t. Si s_{t+1} es terminal, entonces $Q(s_{t+1}, a_{t+1})$ es definido como cero. Esta regla se utiliza para cada elemento de quíntupla de eventos $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$.

En todos los métodos on-policy se estima continuamente Q^π por el comportamiento de π, y al mismo tiempo cambia π respecto Q^π.

2.9. Trazas de elegibilidad.

Es un puente entre TD y Montecarlo. Cuando los métodos TD son aumentados con trazas de legibilidad, producen una familia de métodos de expansión de espectro igual a los Montecarlo, al final de un paso del TD. Esto produce un método intermedio entre TD y Montecarlo.

Existe otra manera de describir las trazas de elegibilidad, en la cual la traza es un registro temporal de la ocurrencia de un evento. La traza marca los parámetros de memoria asociados con el evento como elegible para aprendizaje de cambios. Cuando un error TD ocurre, solo los estados elegibles o acciones son asociados a estos errores. De esta manera, las trazas de elegibilidad son el puente entre los eventos y la información de aprendizaje.

2.9.1. Paso-n de la predicción TD.

Los métodos Montecarlo realizan un respaldo de cada estado basado en la secuencia entera de recompensas desde un estado hasta el final del episodio. El respaldo de un método simple TD esta basado solo en la siguiente recompensa, usando el valor del es-

tado un paso después como aproximación de las recompensas restantes. Un método intermedio, podría realizar un respaldo basado en un número intermedio de recompensas: más de uno, pero menos que todas hasta la terminación.

Los métodos que usan n-pasos como respaldo siguen siendo métodos TD debido a que siguen cambiando una estimación anterior basados en como se difiere con la estimación posterior. La estimación posterior no es un paso posterior, si no n pasos posteriores. Estos métodos son llamados n-pasos TD.

En los respaldos de Montecarlo se estima $V_t(s_t)$ de $V^{\pi}(s_t)$, y se actualiza en dirección al retorno completo:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T, \quad \textit{(2.32)}$$

donde *T* es el tiempo final del episodios.
En un respaldo de un paso, el objetivo es la primera recompensa más el valor estimado descontinuado del siguiente estado:

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}). \quad \textit{(2.33)}$$

En general, con un objetivo de n-pasos es

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}). \quad \textit{(2.34)}$$

Si el episodio termina en menos de n pasos, el truncamiento del retorno de n-pasos ocurre al final del episodio, resultando en el retorno completo convencional. Todos los regresos de n-pasos que duran más que el paso de terminación tienen el mismo valor que el retorno completo.

Un respaldo de n-pasos esta definido como el respaldo hacia el regreso de n pasos, el cual se escribe como:

$$\Delta V_t(s_t) = \alpha\left[R_t^{(n)} - V_t(s_t)\right], \quad (2.35)$$

donde α es un tamaño de paso positivo. Los incrementos de los valores estimados de otros estados son $\Delta V_t(s_t)$ =0, para todo *s* diferentes a s_t. En una actualización online, las actualizaciones son hechas durante el episodio, tan pronto como el incremento es calculado,

En la actualización off-line, los incrementos son realizados “aun lado” y no son usados para cambiar el valor de la estimación hasta el final del episodio.

El valor esperado del retorno de los n-pasos garantiza improvisar en cierta manera la función de valor como una aproximación del valor real de la función. Para cualquier *V*, el valor esperado del regreso del paso-n usando *V* garantiza ser mejor estimación de V^π que el actual *V*. El peor error bajo la nueva estimación es menor o igual que un γ^n veces el peor error bajo *V*:

$$\max_s \left| E_\pi\left\{ R_t^{(n)} \;\middle|\; s_t = s \right\} - V^\pi(s) \right| \;\leq\; \gamma^n \max_s |V(s) - V^\pi(s)|\,. \quad (2.36)$$

La fórmula anterior es llamada propiedad de reducción de error del regreso del paso-n. Gracias a esta propiedad se puede mostrar de manera formal que los métodos de predicción TD usando n-pasos de respaldo convergen a las predicciones correctas bajo las condiciones técnicas adecuadas.

Las recompensas pueden ser hechas no solo hacia n-pasos, si no también hacia el promedio de n-pasos. Un respaldo de promedios simples como componentes de los respaldos es llamado respaldo complejo.

El algoritmo TD(λ) puede ser entendido como la manera de promediar n-pasos promedio. Este promedio contiene todos los n-pasos, cada uno pesado proporcionalmente con

λ^{n-1} donde $0 \leq \lambda \leq 1$. Donde el factor normalizado 1-λ garantiza que los pesos sumen 1. El respaldo resultante hacia la recompensa es llamado retorno-λ, definido por

$$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}. \quad (2.37)$$

Los pesos se desvanecen en cada paso adicional. Después del estado terminal, todos los retornos subsecuentes son iguales a R_t. Si separamos los términos obtenemos:

$$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t. \quad (2.38)$$

Si la suma principal se vuelve cero, los términos restantes se reducen a un retorno convencional, R_t. En el caso de que λ sea igual a 1, la recompensa se comporta igual a Monte Carlo. Si λ es igual a cero, se reduce a un retorno de un paso.

El algoritmo de retorno λ emplea respaldos usando el retorno λ. En cada paso *t*, se calcula un incremento, $\lambda V_t(s_t)$ al valor del estado:

$$\Delta V_t(s_t) = \alpha\left[R_t^{\lambda} - V_t(s_t)\right] \quad (2.39)$$

Se puede anexar una memoria adicional a los algoritmos TD(λ), esta memoria es llamada traza de elegibilidad. Esta es denotada por $e_t(s_t)$. Encada paso, las trazas de elegibilidad para todos los pasos decaen λ^{γ} y la traza de elegibilidad para un estado visitado durante el paso es incrementado en 1

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t, \end{cases} \quad (2.40)$$

Esto es para todos los estados no terminales. Este tipo de traza es llamada acumulativa, ya que acumula en cada tiempo el estado visitado, y después decae gradualmente cuando el estado no es visitado.

Las traza indican cuando un estado es elegible para el aprendizaje, esto se logra con un evento de refuerzo. El evento de refuerzo que se considera es el error en un paso en un TD

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t) \quad (2.41)$$

Y el error global TD como

$$\Delta V_t(s) = \alpha \delta_t e_t(s), \qquad \text{for all } s \in \mathcal{S}. \quad (2.42)$$

2.10. Sarsa (λ)

La idea general es aprender los valores $Q_t(s,a)$ en lugar de $V_t(s)$. Se necesita una traza por cada par estado – acción. Sea $e_t(s,a)$ la traza por cada par *s,a*. Tenemos:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \delta_t e_t(s,a), \qquad \text{for all } s,a \quad (2.43)$$

Donde

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \quad (2.44)$$

Y

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise.} \end{cases} \qquad \text{for all } s,a \quad (2.45)$$

Sarsa(λ) es un algoritmo on-policy(se aproxima $Q^{\pi}(s,a)$), donde los valores para las acciones de la política son gradualmente mejorados basados en la aproximación de valores para la política actual.

3. Planteamiento y Análisis

3.1. Planteamiento del problema.

Las aplicaciones en las que los robots con comportamientos inteligentes pueden verse involucrados son varias: en aplicaciones donde el ser humano se encuentra en riego, entretenimiento, en problemas económicos o de minimización de recursos, transporte, etc.
En particular, los robots de exploración, tienen como objetivo primario cubrir áreas previamente desconocidas esquivando obstáculos simultáneamente.

Los robots inspirados en ápodos, que intentan imitar a animales sin ningún tipo de pata o miembro se caracterizan por:

- Se pueden mover en superficies escarpadas.
- Pueden adoptar diversas formas.
- Cada sección es pequeña comparada con su longitud total.
- Son uniformes.
- Pueden escalar.

En los diferentes trabajos reportados por la literatura, se han desarrollado líneas de investigación para este tipo de locomoción y diseño, las cuales resaltan:

- Búsqueda y rescate.
- Inspección tridimensional
- Movimiento

Una mejora al diseño de robots tipo serpiente, es la incorporación de llantas al diseño, con lo que se le permite tener un control de movimiento horizontal típico de un robot diferencial, proporcionándole gran velocidad de desplazamiento en superficies horizontales regulares, así como las ventajas que proporciona el movimiento de un robot tipo

ápodo. Aunque claro, se tiene que aumentar el número de efectores, con lo que se incrementa el consumo de energía.

Existen varios trabajos de robots tipo serpiente con locomoción diferencial, como lo son OT-4, ACM-R3, Soryu-I, Genbu, mostrados en la figura 1, MAKRO, entre otros. Todos estos trabajos presentan algo interesante, todos los robots son de control semi-autónomo, todos poseen un operador, y algunos poseen un sistema de control de apoyo para movimientos complejos para un solo operador. Tal es el caso de OT-4 mostrado en la figura 2, el cual posee una red neuronal, la cual calcula los pesos de las recompensas para un algoritmo de aprendizaje por refuerzo, los movimientos que ha aprendido son utilizados por el operador para realizar, por ejemplo, movimientos verticales.

Figura 1: (a) Robot ACM-R3. (b) Robot Soryu-I. (c) Robot Genbu.

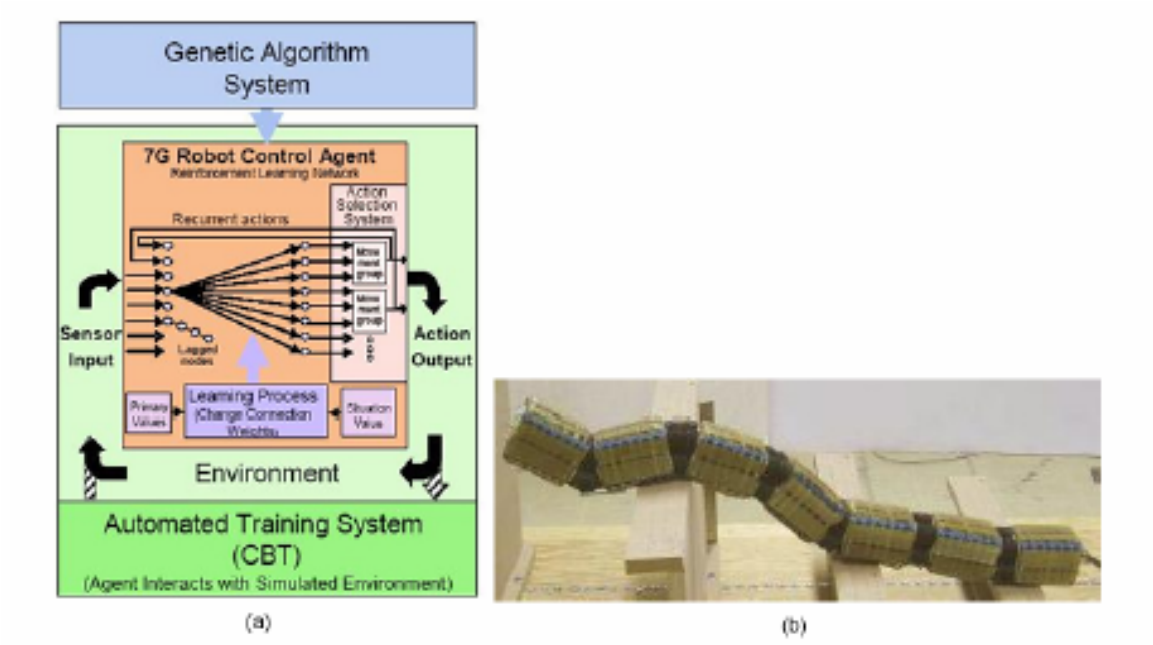


Figura 2: (a) Diagrama esquemático del sistema de control 7G. (b) Robot OT-4. Tomado de [1].

El problema principal en los robots tipo serpiente, es su falta de autonomía en tareas de locomoción, la mayoría de estos robots son controlados por mas de un operador, debido a que poseen un gran número de grados de libertad y a la complejidad de la tarea, lo que repercute en su uso en situaciones donde el operador no tiene noción del estado del robot, así como al uso excesivo de recursos humanos.

3.2. Análisis

Se diseñó, construyó y programó un prototipo de robot móvil articulado terrestre, el cual, a diferencia de los revisados en la literatura, cuenta con las características de:

- Autonomía: El robot debe de ser capaz de realizar sus tareas mediante sus propios medios (activadores, controladores y sensores) sin presencia de medios externos.
- Aprendizaje: El robot debe de aprender a realizar las tareas mediante el intento de acciones repetidas.
- Morfología: El número de eslabones del robot no debe de ser mayor a los reportados en la revisión literaria para robots tipo serpiente (4 eslabones).

Características morfológicas.

Las características que debe de cumplir como un robot tipo serpiente son:

- Cada segmento debe estar unido por una articulación flexible la cual ayuda a realizar el movimiento vertical del robot. Además, cada segmento debe contar con dos llantas, una de cada lado, ambas llantas controladas por un solo motor, en la figura 3 se muestra el diseño de un segmento del robot con llantas.

Figura 3: Diseño del obstáculo tipo escalón

- El robot debe contar con 5 grados de libertad.

- Movimiento rectilíneo sobre el piso, realizando movimientos hacia adelante y hacia atrás.
- Movimiento vertical, necesario para evitar obstáculos, cada articulación en el robot muestra un grado de libertad, la última articulación del robot debe contar con dos grados de libertad.

Características del ambiente de pruebas.

Con respecto a los obstáculos se deben cumplir las siguientes características:

- Deben ser obstáculos tipo escalón.
- El frente del obstáculo debe ser una superficie sólida.
- Deben tener una altura menor a la longitud aproximada del robot.
- Deben tener un largo de al menos 3 segmentos del robot.

Características de las tareas del robot.

El robot debe poder realizar las siguientes acciones o tareas:

- Desplazarse en línea recta en una superficie plana.
- Identificar obstáculos evitables dentro de su rango de acción.

Trepar los obstáculos evitables que se encuentren dentro de su rango de acción.

4. Diseño e implementación.

A continuación se dará una explicación acerca de los aspectos más relevantes en el diseño y construcción del robot, de los cuales destacan la morfología del robot, los materiales utilizados para la construcción del mismo, así como las unidades de control necesarias para alcanzar el objetivo planteado originalmente.

4.1. Morfología.

Este trabajo se realizó con el fin de proponer un diseño y control para robots ``tipo-serpiente", los cuales se caracterizan por tener varias articulaciones, lo que le permite tener un espacio de trabajo aumentado, no solo a un espacio horizontal, si no también vertical, además se decidió incorporar llantas al diseño con lo que se le permite tener un control de movimiento horizontal típico de un robot diferencial, proporcionándole gran velocidad de desplazamiento en superficies horizontales regulares.

Se diseñó un robot con cuatro segmentos con dos llantas cada uno y tres articulaciones, donde las dos primeras articulaciones cuentan con un servomotor lo que les permite mover la articulación hacia arriba, mientras que la tercera articulación cuenta con dos servomotores lo que le brinda un espacio de movimiento aumentado pudiendo moverse hacia arriba y hacia abajo, las llantas son impulsadas por un motor de corriente directa. En la figura 4 se ilustra lo descrito anteriormente.

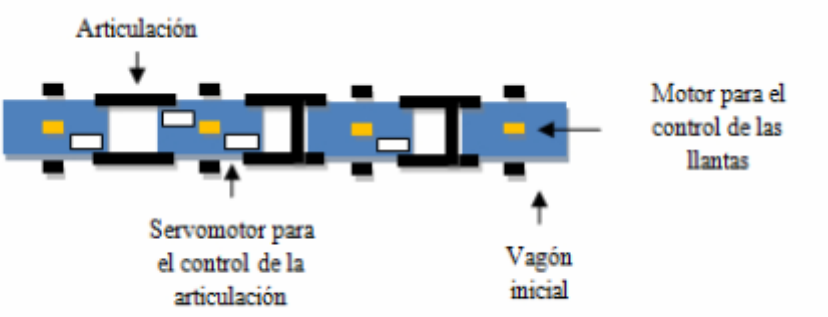


Figura 4: Morfología del robot.

4.2. Construcción y control

Para la construcción del robot se utilizaron los siguientes materiales, (tomando en cuenta el tamaño de los servomotores disponibles, su capacidad de torque y el peso de los materiales):

- Para la construcción de los vagones se utilizo un porta planos de cartón de 5 centímetros de diámetro, cada vagón consta de 16 centímetros de largo.
- Palos de madera, cortados de 11 centímetros de largo para la construcción de las articulaciones, cubiertos con masking-tape para hacerlos mas resistentes.
- Tornillos con tuerca, necesarios para sujetar las articulaciones al vagón.
- Bolas de unicel de 2.5 centímetros de diámetro, adaptadas como``caster-wheel'' en la parte de abajo y al inicio de cada vagón.
- Bolas de unicel de 5 centímetros de diámetro, adaptadas en la parte frontal del robot en el primer vagón.
- Llantas de aluminio de 5 centímetros de diámetro, dos para cada vagón.
- Taquetes de madera, para adaptar el eje de las llantas a las llantas.

Para el control del robot, se eligió una tarjeta controladora capaz de funcionar con baterías (debido a la autonomía del robot) y peso mínimo disponible, con lo que se utilizaron los siguientes materiales:

- Tarjeta de control brainstem, la cual cuenta con las siguientes características:
    - 5 entradas/salidas Digitales.
    - 5 entradas/salidas Analógicas.
    - Capacidad de manipulación de hasta 4 Servomotores.
    - Bus de comunicación IIC.

- Circuitos L293B (puente H) para el control de los motores de corriente directa de las llantas, cada circuito L293B controla las llantas de dos vagones.

En la figura 5 se muestran los materiales utilizados para la construcción y control del robot.

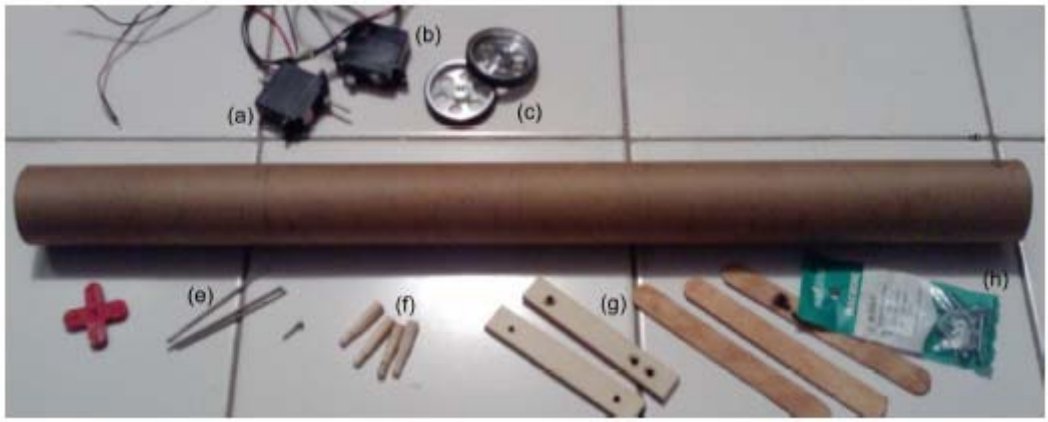


Figura 5: Material utilizado para la construcción del robot. (a) Servomotor modificado, con un eje de metal atravesado para la colocación de las llantas. (b) Servomotor, para el control de las articulaciones. (c) Llantas de aluminio. (d) Porta planos de cartón. (e) Eje de metal. (f) Taquetes de madera. (g) Palos de madera (h) Tornillos con tuerca.

4.3. Equipamiento.

Para el equipamiento del robot, se tomó en cuenta las limitaciones en la tarjeta de control a utilizar, por lo tanto para que el robot sea capaz de realizar las acciones mencionadas en los objetivos es necesario contar con los siguientes sensores y activadores.

- 4 Servomotores para el control de las articulaciones, nótese que la tercera articulación cuenta con dos servomotores.
- 4 Servomotores ``modificados", para el control de las llantas. Con servomotores modificados nos referimos a servomotores sin tope y convertidos a motores de corriente directa, esto con la finalidad de tener mas potencia en la tracción de las llantas ya que los motores de corriente directa convencionales no cuentan con la potencia suficiente para mover el robot.
- 4 Fotorresistencias, ubicadas en la parte de abajo y al frente de cada vagón, esto para que el robot tenga la capacidad de diferenciar si se encuentra abajo o arriba de un obstáculo.

- 4 Leds, ubicados junto a cada una de las fotorresistencias con la finalidad de obtener lectura mas precisas de las fotorresistencias.
- 1 Interruptor, ubicado en la parte frontal del tren necesario para que el robot identifique un posible obstáculo.

- 4 fotorresistencias, ubicadas en cada uno de los vagones, con las cuales el robot debe ser capaz de reconocer si el vagón se encuentra abajo o arriba del escalón, cada una de estas fotorresistencias esta acompañada de un *led* para ayudar a reconocer mejor los colores del piso.

4.4. Locomoción

Debido a que el robot construido en este trabajo es terrestre, se propuso un sistema de locomoción utilizando un arreglo de llantas donde cada vagón cuenta con dos llantas conectadas al mismo motor mediante un eje. Dando como resultado un arreglo de 8 llantas controladas por 4 servomotores modificados.

4.5. Caracterización de los sensores

Para obtener el estado actual del robot es necesario obtener los valores de las cuatro fotorresistencias y la posición de los cuatro servomotores de las articulaciones, debido a que la posición de los servomotores es conocida en todo momento no es necesario realizar una caracterización de la posición de los servomotores. La caracterización de las cuatro fotorresistencias se muestra en el cuadro 1.

Posisicion de la fotorresistencia	f1	f2	f3	f4
Sobre el piso negro	640 - 764	920 - 1024	690 - 800	840 - 960
Sobre el escalón blanco	440 - 520	710 - 835	400 - 510	690 - 790

Cuadro 1: Caracterización de las cuatro fotorresistencias del robot.

4.6. Mecanismo de control.

El proceso de planeación, es en si deliberativo y según la problemática planteada, el robot debería ser capaz de poder realizar una secuencia de acciones para poder sortear un obstáculo vertical.

Se decidió un enfoque de aprendizaje por refuerzo por las siguientes razones:

- Se comenzó a dudar si el diseño permitiría evadir obstáculos verticales.
- No se sabía cual era el menor número de acciones para realizar esta tarea.
- El controlador solo debía poseer la política óptima resultante de la aplicación de los algoritmos de aprendizaje, con lo que se ahorraría espacio en memoria y tiempo de procesamiento en el mismo.

El control se implementó en dos partes, la primera fue mediante una simulación, donde se planteó obtener una política mediante un algoritmo de diferencia temporal, además de obtener las frecuencias de transición de los estados para generar un modelo de programación dinámica con el fin de mejorar la política obtenida. La retroalimentación de los sensores al actuar con el ambiente se obtendría de la simulación. La segunda parte, en el prototipo real, se implementaría la mejor política (aquella que requiera el menor número de acciones) para implementarla en el prototipo real.

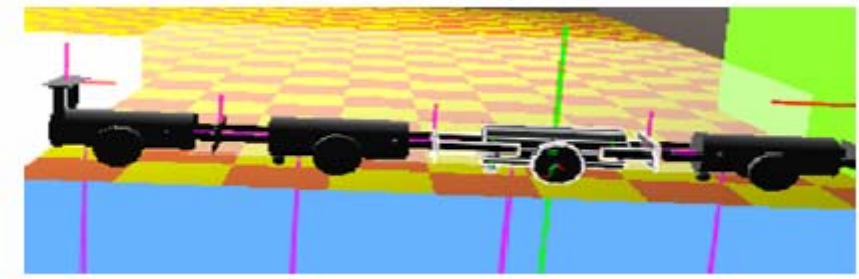


Figura 6: Imagen tomada de la simulación del robot.

La arquitectura de aprendizajes se muestra en la figura 7.

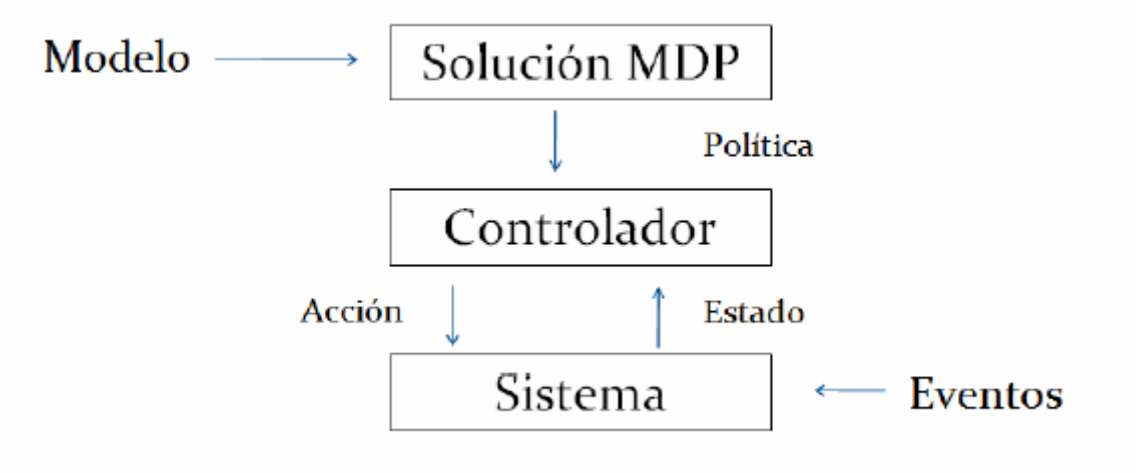


Figura 7: Metodología de desarrollo.

4.7. Modelo

Partiendo del sistema, se procedió paralelamente a definir el modelo, así como a elegir el controlador para el robot. El modelo, primeramente seria resuelto en el simulador Webots 6.1, debido a que la tarea de resolver un modelo de este estilo conlleva a hacer pruebas exhaustivas, el simulador permitiría acortar el tiempo de desarrollo, además de superar las limitaciones del controlador.

El diseño preliminar del prototipo, nos ofreció información acerca del tipo de actuadores que necesitábamos: 4 servomotores, y 4 motores con movimiento paralelo; traducido al modelo, los actuadores nos ofrecieron las acciones que teníamos que representar en el modelo, las cuales se pueden observar en el cuadro 2.

Acciones
Servo 1 arriba
Servo 2 arriba
Servo 3 arriba
Servo 4 arriba
Servo 1 alineado
Servo 2 alineado
Servo 3 alineado
Servo 4 alineado
Avanzar

Cuadro 2: Acciones del modelo del robot.

Uno de los primeros problemas que surgieron fue la definición de los estados del modelo, si bien se tenia el diseño, aun permanecía la pregunta de cuales serian los sensores que darían la información suficiente para representar un estado. La primera propuesta fue utilizar sensores de rango, lo cuales darían información acerca de que tanto el robot avanzaba, sin embargo, la simulación mostró que era impractico tratar de discretizar esta información, y además las limitaciones en el controlador no nos dejarían llevar a acabo aquella implementación. Se decidió utilizar fotorresistencias y cambiar el ambiente. Las fotorresistencias proporcionarían un valor para colores claros, y otro para colores fuertes, con lo que podemos representar, subir el escalón con un color claro, y no subirlo con un color fuerte, lo que nos deja una representación binaria de 0 y 1. Se coloco una fotorresistencia por segmento, lo que nos dio 4 bits (16 posibles estados) inicialmente. En la figura 8 se ilustra esta representación.

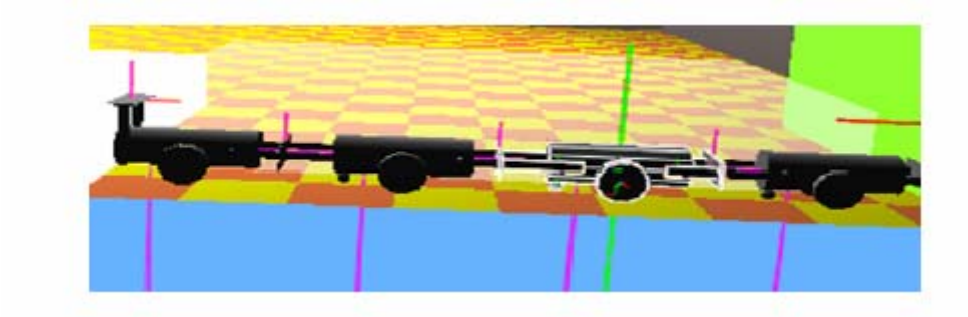


Figura 8: Las líneas rosas representan el rango de captura de las fotorresistencias.

Sin embargo esta representación también resultó impractica por la falta de expresividad, se tenia una ``representación horizontal" del estado del robot, pero no se contaba con una vertical, así que se asignaron 4 bits mas para la representación vertical, que es equivalente a decir si el servomotor esta en posición vertical u horizontal. Finalmente, la representación final fue de 8 bits (256 estados), donde los 4 bits menos significativos representan las posiciones verticales, y los 4 bits más significativos las posiciones horizontales en el ambiente. El bit en la posición *n*, representa al segmento *n* (visto del frente hacia el final del robot), así que los estados finales son representaciones decimales de números mayores a 127.

El siguiente paso fue resolver parte del problema, primero se utilizó un algoritmo de diferencia temporal, esto con el fin de encontrar una solución al control de movimientos

del robot para subir el obstáculo vertical, y la segunda, explorar el ambiente para obtener observaciones de las transiciones del robot para aplicarlo en el algoritmo de programación dinámica. Para esta tarea se eligió un algoritmo on-line con propagación, Sarsa(λ). Se eligió un algoritmo on-line ya que se obtendrían mas frecuencias de los estados *"más importantes"* para la solución que se obtuvieran en la fase de exploración, además de que la convergencia sería mas rápida por utilizar trazas de legibilidad. Se utilizo una exploración tipo ε-voraz con parámetro 0.6 con el fin de ayudar a obtener el mayor número de frecuencias para el modelo del MDP. Debido a que Sarsa(λ) disminuyó el valor del trayecto de la recompensa conforme se aleja de ella, se eligieron las funciones de recompensa como 0, para todos los estados menores a 128, con lo que se obtenían valores cercanos a cero para estados iniciales, y 10 para estados mayores o iguales a 128, que representan los estados finales, con el bit mas significativo (ultimo segmento) arriba del obstáculo.

Una vez obtenidas las frecuencias, se utilizó el algoritmo de programación dinámica, *value iteration*, para resolver el MDP. El teorema de *value iteration* asegura que se obtendrá la política óptima en el límite.

Segmento 4	Segmento 3	Segmento 2	Segmento 1	Servo 4	Servo 3	Servo 2	Servo 1
0 Abajo	0 Abajo	1 Arriba	1 Arriba	0 Abajo	0 Abajo	1 Arriba	0 Abajo

Cuadro 3. Representación de 8 bits de los estados del robot

5. Integración y pruebas.

A continuación se dará una explicación acerca de los experimentos realizados y los resultados obtenidos, así como también se muestra el diseño del ambiente de pruebas y los resultados obtenidos con los algoritmos de aprendizaje aplicados al robot..

5.1. Esquemas del ambiente de pruebas

El ambiente de pruebas se diseñó de la siguiente manera:

- El piso anterior al obstáculo se cubrió con material de color negro, esto con la finalidad de que el robot reconozca si se encuentra abajo del obstáculo.
- El obstáculo se cubrió con material de color blanco, esto con la finalidad de que el robot reconozca si se encuentra sobre el obstáculo.
- El obstáculo se construyo con un largo de 100 centímetros, una altura de 8 centímetros, la superficie anterior al obstáculo tiene una longitud de 100 centímetros, el inicio del obstáculo cuenta con una curvatura para facilitarle al robot el ascenso al mismo.

En la figura 9 se muestra el ambiente de pruebas.

Figura 9: (a) Obstáculo con 100 centímetros de largo y 8 centímetros de alto. (b) Curvatura del obstáculo para facilitar el ascenso del robot.

5.2. Experimentos.

Para probar el correcto funcionamiento del robot se realizaron dos pruebas distintas, para cada prueba se realizaron 10 intentos. A continuación se detallan cada una las pruebas.

Prueba A. Esta prueba se realizó sin la ayuda de los sensores (fotorresistencias), es decir el robot no era capaz de saber en que estado se encontraba. La idea principal de este experimento era la de comprobar si el robot era capaz de subir un obstáculo. Entonces al robot se le proporcionó una serie acciones a realizar las cuales debería ejecutar en forma secuencial. Los resultados de la prueba se muestran en el cuadro 4.

Número de intento	Resultado
Intento 1	El robot se cae
Intento 2	El robot sube 2 segmentos y se atora
Intento 3	El robot sube 3 segmentos y se atora con el ultimo vagón
Intento 4	El robot sube completamente
Intento 5	El robot sube completamente
Intento 6	El robot sube completamente
Intento 7	El robot sube completamente
Intento 8	El robot sube completamente
Intento 9	El robot sube completamente
Intento 10	El robot sube completamente

Cuadro 4: Resultados de la prueba A.

Prueba B. Esta prueba se realizó con la ayuda de los sensores (fotorresistencias), esto con el objetivo de que pueda saber en que estado se encuentra y a partir de esa información realizar una acción predefinida en la política óptima encontrada por el algoritmo de aprendizaje aplicado al modelo del robot. Entonces dadas las lecturas de los sensores el robot debe ser capaz de ejecutar una serie de acciones que lo lleven a alcanzar su objetivo. Los resultados de la prueba se muestran en el cuadro 5.

Número de intento	Resultado
Intento 1	El robot no lee correctamente los datos del sensor
Intento 2	El robot no lee correctamente los datos del sensor
Intento 3	El robot se cae y no logra recuperarse
Intento 4	El robot se cae y no logra recuperarse
Intento 5	El robot se cae y no logra recuperarse
Intento 6	El robot no sube completamente, pero los sensores marcan que llegó al estado final
Intento 7	El robot no sube completamente, pero los sensores marcan que llegó al estado final
Intento 8	El robot no sube completamente, pero los sensores marcan que llegó al estado final
Intento 9	El robot sube completamente
Intento 10	El robot no sube completamente, pero los sensores marcan que llegó al estado final

Cuadro 5: Resultados de la prueba B.

5.3. Trayectoria del robot en imágenes

En la figura 10 se muestra una trayectoria del robot en imágenes, las imágenes están ordenadas de arriba a abajo, siendo la imagen de arriba la posición inicial del robot y la imagen de abajo el estado final del robot cuando se encuentra sobre el obstáculo.

Figura 10: Secuencia de imágenes de la trayectoria del robot.

5.4. Resultados del aprendizaje

Se realizó la ejecución del algoritmo Sarsa(λ) con 50 episodios desde el estado inicial 0 y estado final mayor o igual a 10000000 (base 2). Se utilizó una exploración ε-voraz con ε igual a 0.6 este parámetro se eligió así para favorecer la exploración de mas estados dado la política. Se eligió un λ de 1, un α de 0.1 y un γ de 0.9 (α y γ se eligieron de tal manera que la recompensa no se "desvaneciera" después de varios pasos hacia adelante), después de 13 horas simuladas equivalente a 4 horas reales no se logro completar ningún episodio. Debido a el tiempo en el que procesa el simulador las variables físicas que intervienen en el modelo, combinado con el espacio de búsqueda hace intratable el numero de estados, por lo que se procedió a dividir el problema en subproblemas (expresados en el prototipo como la acción de subir cada uno de los segmentos), que si bien podría no ser la solución global optima, obtenemos esas soluciones más rápidamente en esos subespacios de estados, las cuales, para este problema en específico las podemos combinar para obtener una solución global.

Una vez dividido el problema se realizaron ejecuciones del algoritmo Sarsa(λ) con 20, 50, y 100 episodios, los resultados obtenidos se muestran en el cuadro 6.

Número de episodios	Acciones realizadas
20	19
50	17
100	17

Cuadro 6: Resultados de la solución global alcanzados mediante la fusión de las subsoluciones.

El siguiente paso, fue utilizar las frecuencias obtenidas por el algoritmo Sarsa(λ), con las cuales se calcularon las probabilidades, que fueron la frecuencia a la transición del estado *s'* al estado *s* con la acción *a*, entre la suma total de frecuencias de estados/acción para los estados *S*. Si bien se tenía una estimación de algunos estados, existían aun estados sin estimaciones de probabilidad, debido a que la exploración no recorrió todos los estados, con lo que se procedió a "rellenar" estas probabilidades. Se asignaron entonces a los estados/acciones no recorridos por el algoritmo probabilidades equiprobables. La función de recompensa elegida fue de -1 para todos los estados menorea a 128, y 10 para el caso contrario. El parámetro θ=1, y γ=0.9. El resultado observable para la si-

mulación fue que el robot caía en ciclos debido a la política, lo que implica que no llegaba nunca a la posición final, esto es atribuido, a que se le asignaron probabilidades arbitrariamente a transiciones que son imposible para el robot, por lo que a largo plazo, esas transiciones imposibles proporcionaban mayor recompensa que las registradas por las frecuencias.

El siguiente paso fue replantear las probabilidades faltantes, si bien, no se tenía certeza del siguiente estado, se podía suponer, que todas las probabilidades faltantes eran cero, y que todas las acciones llevadas en un estado desconocido, llevaban al mismo estado. Con lo que se aplico el algoritmo con estas suposiciones arrojando los siguientes resultados.

Número de episodios	Acciones realizadas
20	14
50	11
100	14

Cuadro 7: Resultados de la solución global alcanzados mediante la fusión de las subsoluciones.

Con lo que se muestra, una clara diferencia en el número de acciones alcanzadas con el algoritmo de programación dinámica y Sarsa(λ). El mínimo de acciones alcanzadas por las observaciones registradas se alcanzo con tan solo 20 episodios. Se atribuye que se haya alcanzado con 50 episodios un número de acciones menores al de 100, debido a que en 100 episodios, hubo más oportunidades de fallo con respecto a la política.

Debido a que se obtuvieron buenos resultados, se fue más allá en las suposiciones, y se planteó que la posición de los servomotores es totalmente determinista, si a un servomotor se le da la orden de subir o bajar, estos pasan indudablemente a ese estado. Así que se asignaron como uno dichas probabilidades, con lo que se obtuvo lo siguiente:

Número de episodios	Acciones realizadas
20	12
50	11
100	12

Cuadro 8: Resultados de la solución global alcanzados mediante la fusión de las subsoluciones.

Se estima que 11 es el numero óptimo de acciones para llegar a la meta (la estimación es basada en observaciones del experto). Lo cual esta cerca de alcanzar el algoritmo con una frecuencia obtenida de 20 episodios agregando conocimiento experto. En la figura 10 se muestra la comparación de los resultados mostrados en los cuadros anteriores.

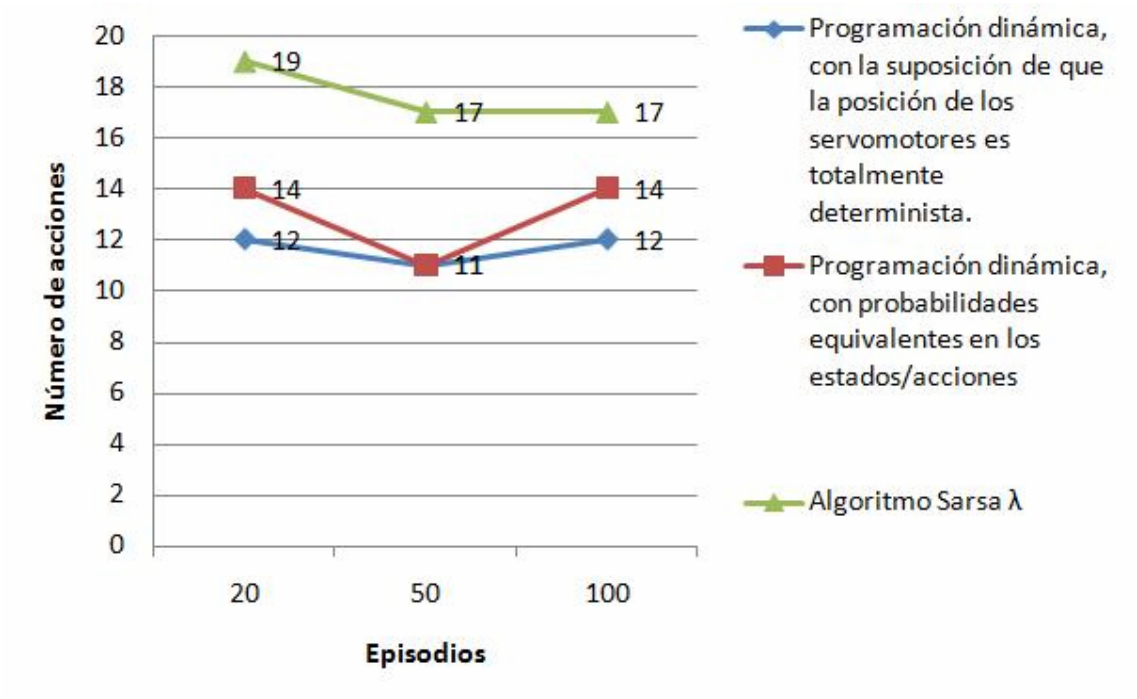


Figura 11: Comparación de resultados.

6. Conclusiones y trabajo a futuro.

- Se construyó un prototipo de robot “tipo-serpiente” con un comportamiento de locomoción autónomo.

- Se pudo obtener un modelo de las acciones y los estados a partir de un prototipo conceptual, estas acciones y estados se definieron también el desarrollo del prototipo real.

- Se obtuvo un comportamiento (política) para el prototipo dadas las interacciones con el ambiente (simulado).

- En esta clase de problemas es importante el modelado, ya que el número de estados afecta exponencialmente el tiempo que tarda en alcanzar la política óptima.

- Para problemas con espacio de búsqueda muy grande es recomendable implementar la solución en un simulador.

- Proceder a realizar más experimentos comparando el número de episodios para la convergencia de Sarsa(λ).

- Se debe definir un algoritmo de exploración lo suficientemente eficiente, para obtener el mayor número de frecuencias de los estados importantes y menor número de frecuencias de los estados que son menos frecuentes.

- Se necesita un gran número de probabilidades, lo cual repercute enormemente en el tiempo en que se obtendrá una solución del modelo.

- Las observaciones de los expertos, no pueden ser dadas a la ligera, fácilmente pueden llevar a modelos inconsistentes.

- Las observaciones correctas de los expertos pueden favorecer enormemente la eficiencia del modelo.

- La combinación de *model-learning* y *reinforcement-learning* arroja resultados bastante aceptables al utilizar estos dos tipos de enfoques por separado, sin embargo, el tiempo de procesamiento se ve incrementado, debido a esta mezcla de algoritmos. Se puede observar claramente el como se llego a la idea de algoritmos como Dyna-Q y Prioritized Sweeping.

- La simplificación del modelo es importante para la reducción de componentes en el hardware de control.

- Puede pensarse en extender el funcionamiento del robot creando nuevos comportamientos.

- Se puede tomar en consideración desechar estados imposibles en el modelo.

7. Bibliografía

[1] Hutchison, William R. and Constantine, Betsy J. and Borenstein, Johann and Pratt, Jerry. Development of Control for a Serpentine Robot. 7th IEEE International Symsium on Computational Intelligence in Robotics and Automation (CIRA2007). Junio 20-23, 2007. Jacksonville, USA

[2] Sutton, Richard y Barto, Andrew G. Reinforcement Learning: An Introduction. The MIT Press. Cambridge, Massachusetts.

[3] Hirose, Shigeo y Fukushima Edwardo F. Snakes and Strings: New Robotic Components for Rescue Operations Tokio Institute of Technology. 2001 Ookayama Meguro-ku, Japón.

[4] Granosik, Grzegorz and Borenstein Johann. Integrated Joint Actuator for Serpentine Robots. IEEE/ASME Transactions on Mechatronics, Vol 10, No 5. Octubre 2005, pp 473 – 481.

[5] Choset, H. et al. Design and motion planning for serpentine robots. Proceedings of the TITech COE/Super Mechano-Systems Symposium, pp. 167–172. 2000.

[6] Henningy, Wade. Hickmany, Frank. Chosety, Howie. Motion Planning for Serpentine Robots. www.cs.cmu.edu/~bio**robotics**/papers/**robot**98.ps.gz. 1998.

[7] Jörg Conradt y Paulina Varshavskaya. Distributed Central Pattern Generator Control for a Serpentine Robot. In ICANN. 2003.

[8] Auke Jan Ijspeert .Central pattern generators for locomotion control .in animals and robots: a review. Neural Networks Vol 21/4 pp 642-653, 2008.

[9] Shammas, E. Wolf, A. Choset, H. Three degrees-of-freedom joint for spatial hyper-redundant robots. Mechanism and Machine Theory, Volumen 41, número 2, pp 170-190.

[10] Lawrence S. Gyger, Jr., Brent W. Spranklin, Satyandra K. Gupta, and Hugh A. Bruck. Bio-inspired, Modular, and Multifunctional Thermal and Impact Protected (TI-Ped) Embedded Sensing Controls Actuation Power Element (ESCAPE) Structures. SEM Annual Conference and Exposition, St. Louis, Missouri, June 2006.