Procesos de decisión de Markov en un agente simulador de brazo robótico para búsqueda y traslación hacia un objeto

Luis Adrián León Alcalá.

National Institute of Astrophisics, Optics and Electronics (INAOE)
Luis Enrique Erro No. 1, Zip 72840, Tonantzintla, Puebla, México
Email: enthe@ccc.inaoep.mx

Resumen. Se presenta el desarrollo de un sistema simulador de brazo robótico basado en técnicas de aprendizaje por refuerzo con MDP que tiene como objetivo localizar objetos y determinar acciones para llegar a ellos en ambientes de dos y tres dimensiones. La idea, al desarrollar este agente simulador, es permitir el modelado y elección de parámetros antes de su implementación en un brazo físico. Se muestra cómo el uso de MDPs permiten una representación natural del problema y cómo, mediante la definición adecuada del modelo MDP, es posible abordar esta problemática. Se introduce la idea de conjunto de estados reconfigurable, que permite modelar el espacio de configuraciones del brazo mediante escenarios cuadriculados. Se muestra la evaluación del sistema en de acuerdo al porcentaje de aciertos y la eficiencia del mismo en términos de iteraciones y tiempo de ejecución de los algoritmos *value iteration* y *policy iteration* involucrados en el desarrollo.

Keywords: Aprendizaje por refuerzo, MDP, Simulador, Brazo Robótico.

1 Introducción

La primera impresión que se tiene al imaginar un brazo robótico es la capacidad de éste para tomar objetos, sin embargo, detrás de esta tarea hay un proceso a considerar. Cualquier objeto en el mundo real consiste de una posición espacial y es necesario, antes de tomarlo, ejecutar ciertas acciones que trasladen el brazo hasta esa posición.

La cuestión se torna más interesante cuando se requiere autonomía, por parte del brazo, para ejecutar esta tarea, de tal manera que el robot, por sí solo sea capaz de tomar decisiones y ejecutar acciones que lo lleven a la posición deseada.

Un área que afronta la necesidad anterior es aprendizaje automático (ML) cuyo objetivo desarrollar programas que mejoren su desempeño de manera automática mediante la experiencia. En esta área existen diferentes enfoques de acuerdo al tipo de aprendizaje y es, debido a las necesidades planteadas, el aprendizaje según el grado de retroalimentación al que se inclina el presente desarrollo. A este enfoque pertenecen el aprendizaje supervisado, el no supervisado y el aprendizaje por refuerzo; es importante recalcar que por las características que rigen a las primeras dos técnicas, es

posible ubicarlas en un paradigma de predicción, que se sirven de ejemplos para poder trabajar.

Aun que el aprendizaje supervisado y no supervisado sean técnicas muy fuertes, no siempre es posible su uso, pues en algunos casos se carece de conocimiento *a priori.*

Por su parte en aprendizaje por refuerzo, se tiene como objetivo programar agentes que sean capaces de lograr una solución, mapeando situaciones a acciones (mediante premio y castigo) de manera que se maximice una señal de recompensa sin necesidad de especificar la manera en que se llega a esta solución (S. Sutton & G. Barto, 1998).

Para el brazo robótico no se tiene un conjunto de ejemplos, además es necesario que el robot interactué con su ambiente de tal manera que éste le provea información para tomar de decisiones. De aquí que el uso de aprendizaje por refuerzo es innegable. Ahora bien, para el modelado certero de la toma de decisiones, los procesos de decisión de Markov (MDP) permiten apropiadamente caracterizar esta problemática.

Por último, la experimentación directa en un brazo físico puede ser costosa ya que es posible ocasionar errores que resulten en daños para el robot. Por tal, lo planteado hasta el momento, se ve reflejado en un agente simulador del brazo en cuestión que permita modelar este proceso de búsqueda y traslación. Esta simulación debe permitir la manipulación de parámetros para determinar, bajo una situación dada, los más adecuados que serán implementados en un modelo físico. Algo interesante del proyecto es la posibilidad de representar el espacio de configuraciones del robot mediante escenarios cuadriculados de tal forma que se permita la reconfiguración de éste con sólo ajustar la cantidad de cuadriculas.

El contenido del documento, queda estructurado de la siguiente manera: en la sección 2 brevemente se presentan el brazo utilizado. En el apartado 3 se aborda el trabajo relacionado. La sección 4 está dedicada a la metodología. Se sigue con la parte de experimentación en la 5 y Finaliza con un apartado de conclusiones y trabajo por desarrollar.

2 El brazo robótico

En esta sección se presenta el brazo robótico que fungió de modelo para el desarrollo del proyecto, se muestran sus partes y se define cuales de ellas son las más necesarias en el proceso de búsqueda, también se especifica un conjunto de consideraciones a tomar cuenta durante el resto del documento.

En la Figura 1 se puede apreciar el brazo en su totalidad del cual, las partes de mayor peso son la Base, que permite los movimientos de izquierda a derecha, el brazo superior (*Upper Arm*) que define los movimientos en profundidad (al frente y atrás) y el antebrazo (*Fore Arm*), parte que determina los movimientos hacia arriba y hacia abajo. Todas estas partes tienen asociado un factor numérico (ángulo) que especifica, a groso modo, la cantidad de movimiento a realizar. Esto parámetros nos permite definir una configuración inicial, de la cual parte el brazo a la búsqueda del objeto, y los límites de movimiento como se muestra en la Tabla 1:

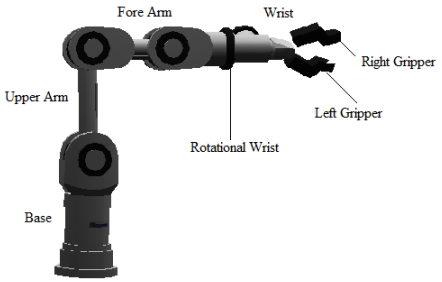

Figura 1: El brazo y sus partes

Tabla 1: Configuración inicial del brazo.[1]

Parte	Nombre corto	Valor inicial	Rango de movimiento permitido
Base	B	3.14	[2.14 , 4.14]
Upper Arm	U	-0.5	[-1 , 0]
Fore Arm	F	3.8	(3.0) [3.4, 4.2]
Wrist		-2.0	[-2.0 , -2.0]

Aun que *Wrist* no sea parte esencial en los movimientos se le asigna una configuración inicial que no se altera en ningún momento del proceso.

Por otro lado, se tienen escenarios cuadriculados para dos y tres dimensiones, que tienen dos intenciones. Una es presentar de manera visual, en la simulación, el conjunto estados del MDP y otra es representar el espacio de configuraciones correspondiente al brazo robótico. Una forma intuitiva de los escenarios pueden apreciarse en la Figura 2.

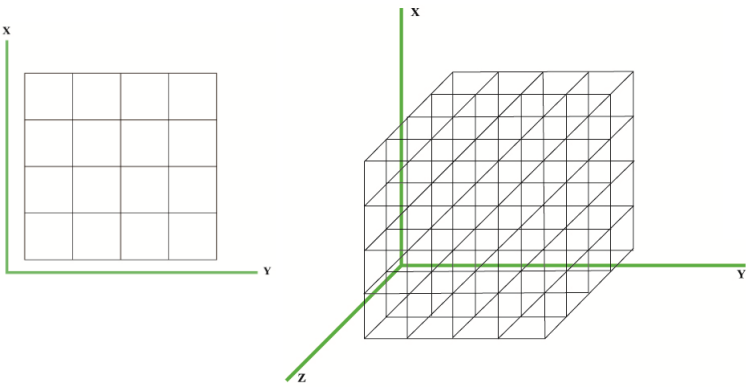

Figura 2: Escenarios cuadriculados que representan el conjunto de estados del MDP en 2D y 3D.

[1] El brazo y los parámetros presentados fueron obtenidos del simulador Webots®. El modelo del brazo real corresponde a: *Harminic Arm 6M 450* de la compañía neuronics.

Una última consideración, es tomar en cuenta que el brazo está inmerso en los escenarios. Para ejemplificar tómese en cuenta el escenario 3D (es importante recalcar que las pinzas son las que tocan el objeto), la configuración inicial del brazo está definida de tal forma que comienza la búsqueda partiendo de la posición inferior central (en los ejes x, y) y a mitad en profundidad (en el eje z) como se puede ver en la Figura 3. Esta consideración es necesaria porque se asegura con ella que el brazo alcance todos los puntos del escenario.

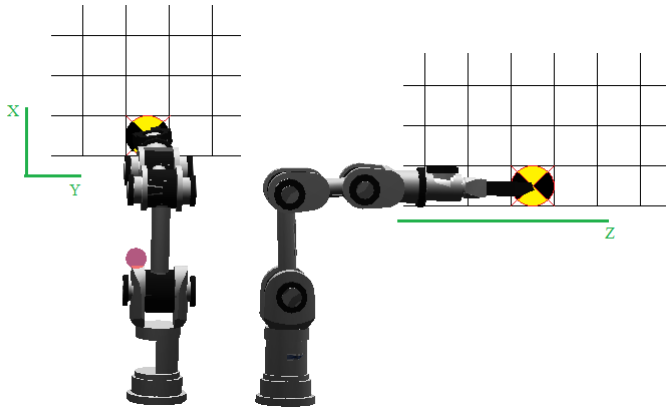

Figura 3: Configuración inicial; el brazo inmerso en el escenario 3D.

3 Trabajo relacionado

Dentro de los trabajos que comparten características con lo desarrollado se encuentra RL_Sim (Mehta & Kelkar) el cual, aun que no tenga por objetivo modelar un brazo robótico, ofrece un agradable ambiente de simulación de MDPs. Lo que se pudiera aprovechar de él es la capacidad de representar escenarios cuadriculados, sin embargo tiene la desventaja, de solo funcionar para escenarios 2D.

Otro par de herramientas que persigue el mismo fin de representar MDPs, son los *TolBox* de MatLab[2], poderosos posiblemente por la naturaleza de MatLab pero restringidos en el ambiente visual.

4 Metodología

En esta sección queda definida la manera en que se atacó el problema. Especificando primero la representación de la problemática mediante un MDP, seguido de la

[2] http://www.inra.fr/bia/T/MDPtoolbox/ y
http://www.ai.mit.edu/~murphyk/Software/MDP/mdp.html

implementación de los algoritmos *value iteration* y *policy iteration* en escenarios 2D y 3D y el desarrollo del ambiente grafico correspondiente a cada espacio (2D y 3D).

4.1 MDP

Formalmente un MDP se define como una tupla $M =< S, A, \Phi, R >$, donde S es un conjunto finito de estados del sistema $\{s_1, \dots, s_n\}$. A es un conjunto finito de acciones $\{a_1, \dots, a_m\}$. $\Phi: A \times S \to \Pi(S)$ es una función de transición que define las probabilidades de alcanzar un estado s' realizando la acción a estando en el estado s y $R: S \times A \to \Re$ la función de recompensa que define al premio o castigo que el sistema recibe si ejecuta la acción a estando en el estado s. (Ballesteros, 2006).

Ahora para el sistema en desarrollo:

4.1.1 Conjunto de Estados

Partimos recordando la Tabla 1. Esta contiene la configuración inicial del brazo y una columna de encabezado Nombre corto. Este nombre corto es utilizado para denotar una configuración, por ejemplo (B=3.14, U=-0.5, F=3.8) denota la configuración inicial. S debe ser un conjunto de configuraciones de este tipo por lo que a cada estado $s \in S$ le corresponde implícitamente una notación $s(B, U, F)$.

El papel de los escenarios es muy importante al momento de definir estados, ya que estos determinan la cantidad de configuraciones en el MDP y con eso, la cantidad de movimiento que debe realizarse en cada configuración. Esto es lo que llamaremos, conjunto de estados reconfigurable y queda directamente influenciado por los rangos de movimiento permitido.

Sean B_R, U_R y F_R factores numéricos de determinan el valor del rango de movimiento permitido:

$$B_R = 4.14 - 2.12 = 2$$
$$U_R = 0 - (-1) = 1$$
$$F_R = 4.2 - 3.4 = 0.8$$

sean X, Y y Z las longitudes en número de casillas de un escenario 3D en sus respectivos ejes y sean Δ_X, Δ_Y y Δ_Z porciones de movimiento tal que:

$$\Delta_X = \frac{X}{F_R}; \qquad \Delta_Y = \frac{Y}{B_R}; \qquad \Delta_Z = \frac{Z}{U_R}$$

se define al conjunto de estados partiendo de la configuración inicial en grupos de la siguiente forma:

Arriba: $s(B = B, U = U, F = F - \mathrm{f}_1\Delta_X)$ Abajo: $s(B = B, U = U, F = F + \mathrm{f}_1\Delta_X)$
Derecha: $s(B = B - \mathrm{f}_1\Delta_Y, U = U, F = F)$ Izquierda: $s(B = B + \mathrm{f}_1\Delta_Y, U = U, F = F)$

Frente $s(B = B, U = U+, \mathrm{f}_1\Delta_Z\ F = F - \mathrm{f}_2\Delta_X)$
Frente $s(B = B, U = U-, \mathrm{f}_1\Delta_Z\ F = F + \mathrm{f}_2\Delta_X)$

Donde f_i indica el número de veces que se descuenta o incrementa el Δ correspondiente. En la Figura 4 se puede ver de mejor manera lo mencionado. La ventaja de tener un conjunto de estados reconfigurable, es que se permite el modelado del espacio de configuraciones del brazo, para definir sin necesidad de experimentar en el brazo físico, el conjunto de éstas que mejor convenga.

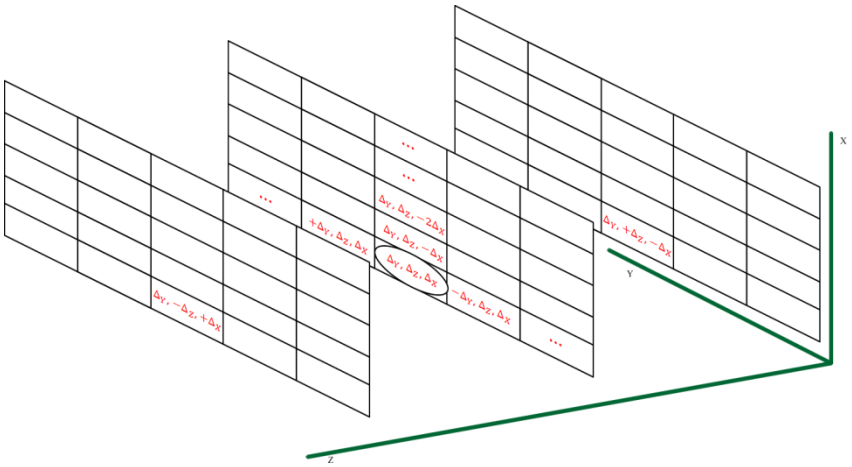

Figura 4: Conjunto de estados reconfigurable.

4.1.2 Conjunto de Acciones

A está regido por 4 movimientos para escenarios 2D y 6 para 3D. Los movimientos son:

Tabla 2: Conjunto de acciones

2D	Arriba	Derecha	Abajo	Izquierda		
3D	Arriba	Derecha	Abajo	Izquierda	Frente	Atrás

4.1.3 Función de Transición

La función de transición (Φ) queda determinada por un ε que indica la probabilidad de pasar a un estado s' partiendo de s con una acción deseada a. El resto de las probabilidades queda definida como $1 - \varepsilon / |A| - 1$ y se puede apreciar un ejemplo más descriptivo en la Figura 5.

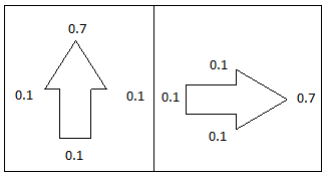

Figura 5: Función de transición con $\varepsilon = 0.7$

4.1.4 Función de Recompensa

Es necesario mencionar que los escenarios permiten tres tipos de estados, estados objetivo, obstáculo y normales, cada uno con una recompensa diferentes, y aun que esta puede ser ajustable desde la simulación, se considera necesario mantenerla fija (al menos para el presente reporte), ya que con los valores establecidos a partir de ahora se realizan las pruebas presentadas en secciones posteriores. La función de recompensa indica la ganancia o pérdida por ejecutar acciones que lleven a posicionarse en ese tipo de estado. Entonces R queda definida como:

$$r(estados\ objetivo) = 10{,}000$$
$$r(estados\ obstáculo) = -20$$
$$r(estados\ normales) = 1$$

4.1.5 Otros componentes del MDP

π: Una política (un conjunto de acciones asociadas a los estados) que define el comportamiento del sistema en cierto tiempo.

V: Una función de valor $V(s)$ que específica la utilidad que un agente puede acumular partiendo del estado $s \in S$.

La idea es maximizar los valores de la función $V(S)$ para cada $s \in S$ de tal forma que la política π defina un conjunto de acciones que determinen la traslación del brazo hacia un estado objetivo.

4.2 Algoritmos

En esta sección se muestran los algoritmos (a un alto nivel) involucrados que llevan por nombre *value iteration* y *policy iteration*. Ambos provienen de una colección de algoritmos que calculan políticas óptimas siempre y cuando se tenga un ambiente bien conocido. Los MDP cumplen esta condición por lo que estos algoritmos son naturalmente aplicables a este tipo de problemas. A la colección de algoritmos de donde provienen *value iteration* y *policy iteration* se le conoce como programación dinámica (S. Sutton & G. Barto, 1998). Se presenta primero el algoritmo de iteración de valor seguido de iteración de política y en la sección de anexos se muestran más a detalle.

4.2.1 Algoritmo iteración de valor (*Value Iteration*)

$Inicializar\ V(s)_t = V(s)_{t+1} = R$
$Repetir$
$\quad V(s)_t = V(s)_{t+1}$
$\quad V(s)_{t+1}(i) = R(i) + max_a \sum_j P(s_j \mid s_i, a)\, V(s)_t(j)$
$Hasta\ |V(s) - V(s)_{t+1}| < \theta$

Donde:

- $P(s_j \mid s_i, a)$ es la probabilidad de estar en el estado S_i y pasar a S_j ejecutando la acción a.

4.2.2 Algoritmo iteración de política (*Policy Iteration*)

Escoger una política inicial
Repetir hasta convergencia
Obtener el valor para V^{π} *a partir de iteración de valor basado en* π
Para cada accion calcular:
$Q_a = R + \gamma P_a V^{\pi}$
$Redefinir: \pi(s) = argmax_a \; Q_a(s)$

4.3 Implementación

Para este desarrollo fue necesario contar con un lenguaje y un ambiente de programación, necesidad que fue cubierta con el uso de Java 6.17 y NetBeans 6.8.

Esta sección tiene como fin mostrar la implementación mediante una breve vista del agente simulador. El sistema está dividido en 2 partes. La primera es la que corresponde a escenarios 2D y la segunda está dedicada para 3D. La idea principal al desarrollar un ambiente gráfico es que permita participar, en el proceso de aprendizaje, mediante la observación e intercambio de valores de los parámetros involucrados y sin más, se presentan en las figuras 7 y 8 las correspondientes interfaces de la aplicación.

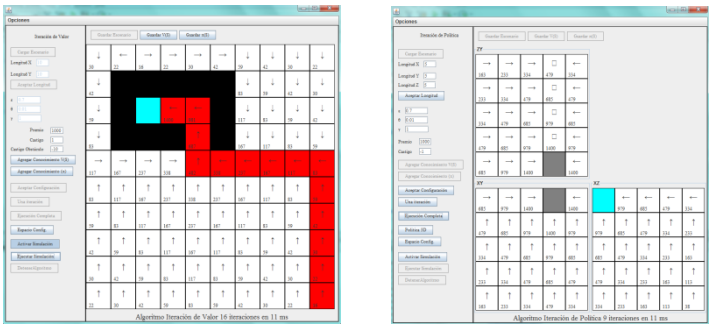

Figura 6: Ambientes gráficos de simulación para escenarios 2D y 3D

5 Pruebas y resultados

En esta sección se presentan los experimentos realizados, la configuración bajo la cual se llevaron a cabo y los resultados obtenidos de este proceso.

5.1 Configuración de los experimentos

La primer parte de la experimentación tiene por intención determinar el grado de certeza que permite cada algoritmo para posicionarse en un estado objetivo partiendo de cualquier otro involucrado. Para ello, es importante determinar la posición de un estado meta. Esta posición debe ser lo suficientemente crítica (alejada de algunos estados) para que permita describir este grado de exactitud. Es interesante ver como se comporta del sistema cuando se presentan obstáculos, por ello, la configuración involucra un conjunto de ellos. Para las pruebas se generan escenarios de $10 \times 10, 20 \times 20, \ldots, 100 \times 100$ en el caso de 2D, mientras que para 3D los escenarios son de $5 \times 5 \times 5, 8 \times 8 \times 8, \ldots 22 \times 22 \times 22$ de tal manera que en el escenario 3D participen aproximadamente el mismo número de estados que en 2D. Gráficamente estas configuraciones pueden apreciarse en las Figura 7 y 8 respectivamente.

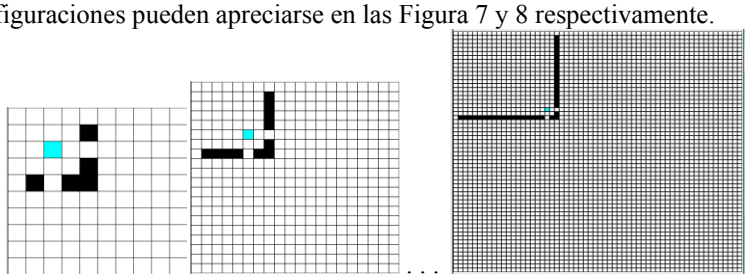

Figura 7: Configuración para la experimentación 2D (el estado marcado con azul es el objetivo, los negros son obstáculos y los blancos estados normales)

Ahora, los parámetros involucrados en estas pruebas quedan definidos de la siguiente manera:

- De la definición en la sección MDP las recompensas son: para estado objetivo = 10,000, para estado obstáculo = -20, para estado normal = 1.
- Para la función de transición, se define un $\varepsilon = 0.7$.
- El criterio de paro θ esta definido con 0.001.
- Y por último $\gamma = 1$. La idea al asignarle este valor es que no altere los valores de recompensa y que todos los estados adyacentes sean tratados de una manera similar.

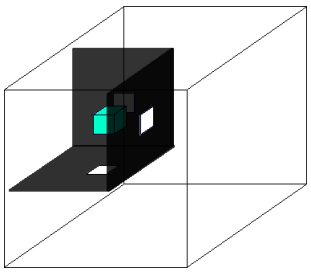

Figura 8: Configuración para la experimentación 3D

5.2 Prueba 1: porcentaje de estados que llegan al estado meta en escenarios 2D

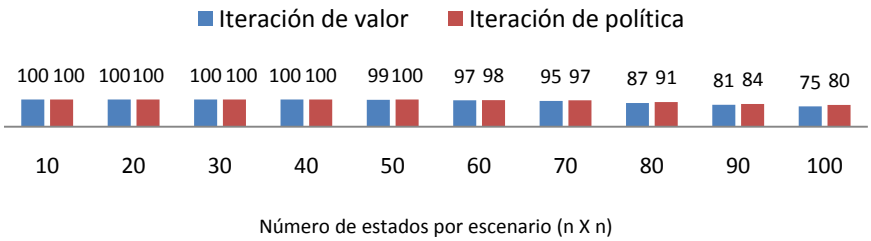

Figura 9: Porcentaje de estados que permiten, partiendo de ellos, llegar a la meta en escenarios 2D

5.3 Prueba 2: porcentaje de estados que llegan al estado meta en escenarios 3D

Para escenarios 3D bajo la configuración propuesta sorpresivamente es posible llegar desde cualquier estado participante hasta la meta, i.e. se cubre, para los dos algoritmos un 100% de aciertos.

Una segunda etapa de experimentación está dedicada a evaluar el rendimiento de cada uno de los algoritmos involucrados, tomando en cuenta el número de iteraciones que realiza cada uno y el tiempo que tardan independientemente en realizar su ejecución. Para esta parte, se toma la misma configuración anterior.

5.4 Prueba 3: numero de iteraciones por algoritmo en escenarios 2D

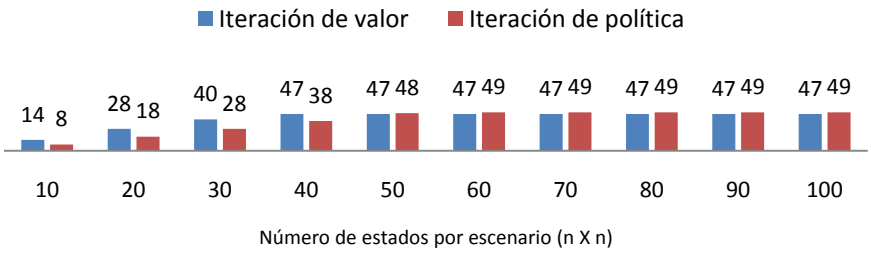

Figura 10: Número de iteraciones por algoritmo en escenarios 2D

5.5 Prueba 4: tiempo de ejecución (en ms) por cada algoritmo en escenarios 2D

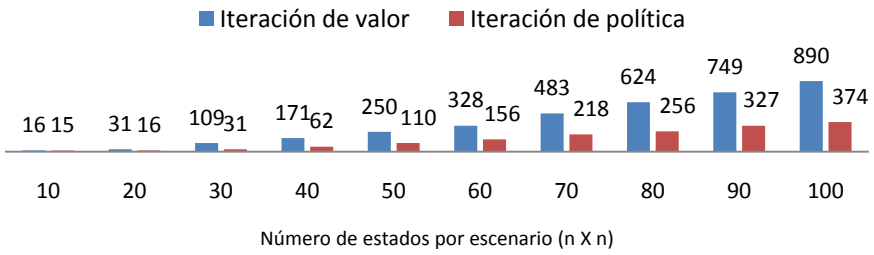

Figura 11: Tiempo de ejecución por algoritmo en escenarios 2D

5.6 Prueba 5: numero de iteraciones por algoritmo en escenarios 3D

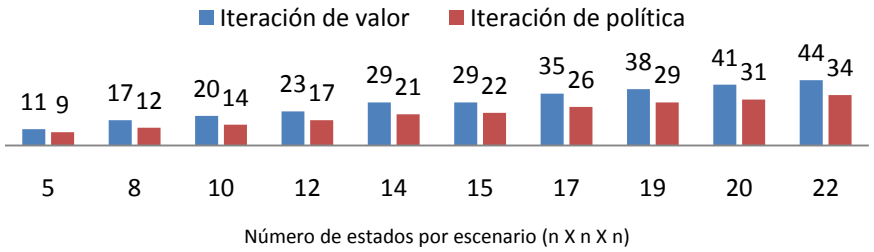

Figura 12: Número de iteraciones por algoritmo en escenarios 3D

5.7 Prueba 6: tiempo de ejecución (en ms) por cada algoritmo en escenarios 3D

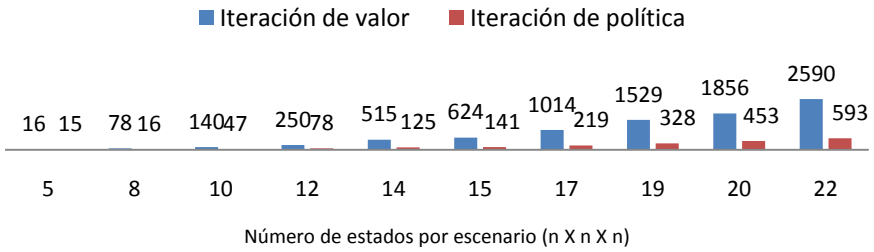

Figura 13: Tiempo de ejecución por algoritmo en escenarios 3D

6 Análisis de los resultados

En esta sección se describen los resultados obtenidos en la sección anterior y se intenta definir el porqué de estos resultados. Véase anexos 2.

7 Conclusiones y trabajo futuro.

Se ha presentado un sistema que hace uso de MDPs para la simulación de un brazo robótico en busca de un objeto. La idea, al desarrollar este sistema, es permitir el modelado y elección de parámetros antes de su implementación en un brazo físico. Se mostró como un MDP puede fácilmente abordar este tipo de problemas si se representa de una manera adecuada. Se presentó una idea para modelar el espacio de configuraciones de un brazo robótico como un conjunto de estados en el MDP. Se presentó la evaluación del sistema en porcentaje de certeza para llegar a la meta al partir de cada estado involucrado y la eficiencia de los algoritmos iteración de valor y de política en términos de iteraciones y tiempo.

El trabajo por realizar es permitir el ingreso de parámetros de brazo robótico, tanto de configuración inicial como de rango de movimiento permitido, con el objetivo de modelar otros brazos que compartan características con el presentado.

Bibliografía

[1] S. Sutton, R., & G. Barto, A. (1998). *Reinforcement Learning: An introduction.* The MIT Press.

[2] Ballesteros, A. R. (2006). *Representación y Aprendizaje de Procesos de Decisión de Markov Cualitativos.* Cuernavaca, Morelos: Tesis de Doctorado.

[3] Mehta, V., & Kelkar, R. *Simulation and Animation of Reinforcement Learning Algorithm:* Reporte de desarrollo RL_Sim. Pittsburg.

Anexos 1 (Algoritmos)

Iteración de valor

$Inicializa\ V(s) = 0\ para\ toda\ s \in S$
$Repite$
    $\Delta \leftarrow 0$
    $Para\ cada\ s \in S$
        $v \leftarrow V(s)$
        $V(s) \leftarrow max_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^*(s')]$
        $\Delta \leftarrow max(\Delta, |v - V(s)|)$
$Hasta\ que\ \Delta < \theta$
$\{Regresa\ una\ \pi\ determinista\}$
    $\pi \leftarrow argmax_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^*(s')]$

Donde

- $P^a_{ss'}$ es la probabilidad de estar en el estado s y pasar a s′ ejecutando la acción a.
- $R^a_{ss'}$ es la recompensa que se obtiene al estar en el estado s y pasar a s′ ejecutando la acción a.
- Δ es un valor que determina el criterio de convergencia junto con θ (para un valor pequeño positivo).
- γ es un factor de descuento. Tiene una mayor presencia en problemas que de horizonte infinito, donde es posible tener demasiados estados adyacentes y la utilidad acumulada en el estado del que se parte puede ser muy grande.

Iteración de política

1. *Inicialización*
$V(s) \in \Re\ y\ \pi(s) \in A(s)\ arbitrariamente\ \forall a\ \in S$

2. *Evaluación de política*
$Repite$
    $\Delta \leftarrow 0$
    $Para\ cada\ s \in S$
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{s'} P^{\pi(s)}_{ss'} [R^{\pi(s)}_{ss'} + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
$Hasta\ que\ \Delta < \theta$

3. *Mejora de política*
$pol\_estble \leftarrow true$
$Para\ cada\ s \in S$:
    $b \leftarrow \pi(s)$
    $\pi(s) \leftarrow argmax_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$
    $if(b \neq \pi)$
        $pol\_estable \leftarrow false$
$if(pol\_estable)$ $detener$
$else$ $go\ to\ 2.$

Donde

- $P^{\pi(s)}_{ss'}$ es la probabilidad de estar en el estado s y pasar a s' ejecutando la acción dictada por la política π en el estado s.

- $R_{s\,s'}^{\pi(s)}$ es la recompensa que se obtiene al estar en el estado s y pasar a s' ejecutando la acción dictada por la política π en el estado s.

A las acciones dictadas por la política $a\,y\,\pi(s)$ se les conoce como acciones deseadas. Las ecuaciones involucradas en los algoritmos del tipo $V(s) \leftarrow \sum_{s'} P_{s\,s'}^{a}[R_{s\,s'}^{a} + \gamma V(s')]$ son conocidas como ecuaciones de Bellman.

Anexos 2 (Análisis de los resultados)

Antes, es necesario hacer explicito que los valores arrojados en las pruebas presentadas están directamente relacionados con los valores definidos en los parámetros, sin embargo los de mayor peso en estos resultados son los premios y castigos.

En la prueba 1 (Figura 9) se puede ver como a mayor número de estados, a partir de escenarios de 50, el número de aciertos comienza a descender. Esto es porque el valor de recompensa por estado objetivo no es suficiente para cubrir a los estados más lejanos y prácticamente estos, al intentar cambiar su valor con el de estados adyacente, no tienen más de donde tomar. Esta situación hace que los algoritmos converjan y los estados más lejanos prácticamente no cambien su valor después unas cuantas iteraciones (tres en la observación para estados de las orillas).

Para la segunda prueba, en escenarios 3D, aun que se tuviera la misma cantidad de estados participantes por escenario, no ocurre lo mismo. Esto es, primero, porque la cantidad de estados adyacentes es mayor, (6 para 3D vs 4 para 2D) y por consiguiente es posible tomar una mayor utilidad, y el punto más importante, por el cual se tiene un 100% de acierto, es que en escenarios de 3D los estados entre sí comparten mayor cercanía debido a que están dispuestos de una manera más compacta. Para ejemplificar: supóngase 2 escenarios uno 10×10 para 2D y otro $5 \times 5 \times 5$ para 3D ($\approx$ en número de estados), supóngase un estado meta en el origen de ambos escenarios (0, 0 y 0, 0, 0 respectivamente) y determine la distancia de él hasta el último estado (10, 10 o 5, 5, 5). Utilizando la distancia euclidiana los valores correspondientes a cada diagonal son, 14.14 para 2D y 8.66 para 3D lo que demuestra que en escenarios 3D los estados comparten una menor distancia. Es por esta cercanía que el valor de recompensa es adecuado para calcular la utilidad de todos los estados.

De la Figura 10, referente al número de iteraciones por algoritmo 2D, se aprecia que para escenarios con cierta cantidad de estados el número de iteraciones prácticamente es el mismo. El hecho es que ambos algoritmos manejan un criterio de paro ($\theta = 0.001$ para este caso) el cual queda definido por una Δ que está determinado por la diferencia entre la función de valor de una iteración anterior y la nueva. Los valores de esta función de utilidad quedan limitados por los premios y castigos definidos al principio. Por lo cual para la configuración (premio = 10,000, obstáculo = -20 y estado norma = 1) 47 es el número máximo de iteraciones que se podrán alcanzar para Iteración de valor a partir de escenarios de 40×40 estados y 49 para Iteración de política a partir de 50×50 estados, no más.

En general, adelantándose a los demás análisis referentes a iteraciones (Figura 12), se pudo observar que Iteración de valor siempre ejecuta un mayor número de ellas debido a que, aun que ya se tenga una política óptima, el criterio de convergencia es

con referencia a la función de utilidad, cuya diferencia, entre una iteración y otra, puede continuar disminuyendo hasta alcanzar el umbral θ definido. Una excepción a esto es lo analizado anteriormente (iteración de valor se queda con 47 e iteración de política en 49).

Respecto al tiempo de ejecución, en todas las pruebas de esta índole se puede apreciar que, iteración de política, aun que involucre en cierta forma a iteración de valor, se desempeña mejor. Esto no es sólo porque realice un menor número de iteraciones, gran parte del aporte es porque al momento de traducir el algoritmo a código se tomó la mejor parte de iteración de valor ($max_a \sum_{s'} P^a_{s\,s'} [R^a_{s\,s'} + \gamma V^*(s')]$ que no se tiene contemplada en el algoritmo iteración de política original) y se decidió explotar la definición de acción deseada (sección 4.2); cuando se tiene una política π de alguna manera se tiene asociada una acción para cada estado. Esta $\pi(s)$ es la que se ejecuta y no es necesario, como en iteración de valor, probar cada $a \in A$.