SISTEMA AUTÓNOMO DE NAVEGACIÓN EN ROBOTS CON RECONOCIMIENTO DE PATRONES GEOMÉTRICOS REGULARES

■Danny Dos Santos y Rafael Peñalver
Tutor: Dr. Wilmer Pereira

Grupo de Investigación de Inteligencia Artificial (GIIA)
Universidad Católica Andrés Bello

Sinopsis

En este trabajo de grado se investigaron las diferentes técnicas de inteligencia artificial enfocadas al área del aprendizaje, específicamente al aprendizaje por refuerzo, que permitan a un ente autónomo o robot alcanzar un objetivo ubicado en un ambiente formado por cuadras de figuras geométricas regulares. Este ambiente no es conocido por el robot, por lo que el aprendizaje se hace en el momento en que éste entra en contacto con el ambiente. Lo único que se conoce previamente es la ubicación del objetivo, dado en términos de coordenadas (x, y).

Para la construcción del robot se utilizó el kit de Lego Mindstorm 2.0, el cual cuenta con tuna gran variedad de piezas, tales como motores, engranajes, sensores etc.; que facilitan la construcción de robots. Este kit posee una pieza denominada RCX, la cual está dotada de un microprocesador Hitachi H8/300, 3 puertos de entrada (sensores), 3 puertos de salida, una memoria para programas de usuario y un puerto infrarrojo utilizado para la comunicación entre RCX's y la computadora. Para este proyecto se utilizaron 2 sensores de luz y un sensor de rotación en el robot.

Con respecto al aprendizaje del robot o ente autónomo, el algoritmo de aprendizaje por refuerzo seleccionado fue Q-Learning, el cual consiste en recompensar o penalizar cada una de las acciones posibles que el robot ejecuta, logrando hallar así una política de movimiento apropiada que permita alcanzar el objetivo.

También se desarrolló una aplicación donde se simulan los algoritmos de aprendizaje por refuerzo Q-

Learning y Value Iteration en distintos tipos de ambientes ideales, lo cual permitió comparar el desempeño de ambos algoritmos bajo diferentes situaciones, ya sea cambiando las cuadras del ambiente, como la posición del objetivo.

Planteamiento del Problema

Resulta interesante poder realizar movimientos en diferentes tipos de ambientes compuestos por cuadras de figuras geométricas regulares que no sean previamente conocidas por el ente autónomo que se desplaza. En consecuencia, se desea diseñar un sistema que permita la navegación de un robot en un ambiente compuesto por figuras geométricas regulares (triángulos, cuadrados, pentágonos, etc.) sin mezclarlas, usando métodos o algoritmos de aprendizaje para desplazarse de un punto a otro independientemente del ambiente donde esté ubicado identificando diversos patrones geométricos regulares sin un conocimiento previo del mismo.

Objetivo General

Desarrollar y simular en una aplicación un software para el desplazamiento de un robot autónomo que permita el reconocimiento de patrones geométricos regulares ubicados en un ambiente de paredes geométricas regulares utilizando algoritmos de aprendizaje.

Objetivos Específicos

- Utilizar las lecturas obtenidas por los sensores, para generar una base de datos que sirva como entrada para el reconocimiento de patrones utilizando algoritmos de aprendizaje.
- Generar secuencias de movimiento en los robots con arquitecturas diversas para lograr el desplazamiento y programarlo con uno de los algoritmos de aprendizaje para lograr un mejor comportamiento dentro del medio ambiente.
- Realizar un estudio comparativo de los resultados arrojados en la implantación de los diversos algoritmos de aprendizaje y desarrollar una aplicación que simule el comportamiento de ellos en la computadora.
- Explorar caminos hacia una localidad; manteniendo en lo posible orientado el robot.

Justificación

El problema surge por la necesidad de ubicarse en un ambiente desconocido, por ejemplo: en Venezuela, la mayoría de las cuadras son rectangulares y los movimientos son dados en ángulos rectos mientras que en otros lugares, como Francia, se puede encontrar cuadras que son triangulares, por lo que es posible llegar a perder el sentido de la orientación. Si un ente no está acostumbrado a moverse en lugares donde las cuadras no son rectangulares, es necesario un nuevo proceso de aprendizaje que pueda hacer la tarea de reconocimiento de estos patrones y pueda lograr ir de un punto a otro sin perderse o llegar siempre al mismo lugar.

Marco Referencial

Aprendizaje por refuerzo

El objetivo en el aprendizaje por refuerzo es usar las recompensas en el aprendizaje con una satisfactoria función de agente. Lo anterior es difícil, ya que el agente nunca sabe ¿qué acciones tomar? para lograr ciertas recompensas.

El comportamiento aprendido usando aprendizaje por refuerzo contiene un modelo implícito del robot y su ambiente. Usando aprendizaje por refuerzo no se necesita tener ejemplos para construir y validar el comportamiento. El comportamiento es sintetizado usando como única fuente de información un escalar, llamado refuerzo, el cual evalúa las acciones del comportamiento donde el agente recibe refuerzos positivos, negativos o nulos de acuerdo a la utilidad de la situación a la que se entró como consecuencia de la acción. No hay separación entre la fase de aprendizaje y la fase de utilización. Además, usando el aprendizaje por refuerzo, sólo las asociaciones relevantes entre entradas y salidas son aprendidas.

En cierta forma, el aprendizaje por refuerzo es otra forma de plantear el problema de la IA. Un agente en un ambiente obtiene percepciones, las correlaciona con utilidades positivas o negativas y luego decide que acción emprender. A continuación se mencionan las diversas variaciones de la tarea del aprendizaje:

- El ambiente puede o no ser accesible. En un ambiente accesible los estados se identifican con determinadas percepciones; en el ambiente inaccesible el agente debe mantener cierto estado interno para tratar de llevar un registro

de lo que es el ambiente. El agente puede empezar con cierto conocimiento del ambiente y de los efectos de sus acciones o bien, deberá aprender éste modelo así como información de utilidad.

- Las recompensas pueden recibirse ya sea en estados terminales o en cualquier estado.
- Las recompensas pueden ser parte de la utilidad real (puntos para que un agente que juegue tenis de mesa, o dólares para un agente de apuestas) que el agente se esfuerza por maximizar, o también pueden ser sugerencias de la utilidad real ("buena jugada" o "perro malo").
- El agente puede ser un aprendiz pasivo o un aprendiz activo. El aprendiz pasivo se limita a observar como evoluciona el mundo y se esfuerza por aprender la utilidad que implica un estado determinado; el aprendiz activo también debe actuar de acuerdo a la información aprendida y puede recurrir a su generador de problemas para que le sugiera la exploración de áreas desconocidas del ambiente.

En el modelo estándar de aprendizaje por refuerzo un agente interactúa con su ambiente. Esta interacción toma la forma del agente sintiendo el ambiente, y basado en la entrada de los sensores, escoge una acción a ser desempeñada en el ambiente. La acción cambia el ambiente de alguna manera y este cambio es comunicado al agente a través de una señal de refuerzo. Según Harmon y Harmon (s.f.), la función de refuerzo y la función valor son partes fundamentales del aprendizaje por refuerzo.

La función refuerzo

Los sistemas de aprendizaje por refuerzo aprenden a asociar situaciones con acciones a través de interacciones de ensayo y error con un ambiente dinámico. El "objetivo" del sistema de aprendizaje por refuerzo es definido usando el concepto de una función de refuerzo, la cual es la función exacta de los futuros refuerzos que el agente busca maximizar. En otras palabras, existe una relación entre los pares estado-acción con los refuerzos; tras desempeñar una acción en un estado donde el agente recibirá algún refuerzo (recompensa) en forma de un valor escalar. El agente aprende a maximizar la suma de los refuerzos recibidos cuando comienza desde un estado inicial y procede a un estado terminal. Es el trabajo del diseñador del sistema de aprendizaje por refuerzo definir una función de refuerzo que describa propiamente los objetivos del agente.

La función valor

Anteriormente se discutieron el ambiente y la función de refuerzo, sin embargo, la manera de cómo el agente escoge "buenas" acciones o inclusive como podemos medir la utilidad de una acción no es explicado. Primero, se deben definir dos términos: a) una política determina cual acción debería ser ejecutada en cada estado, b) una política es una relación de los estados a las acciones. El valor de un estado es definido como la suma de los refuerzos recibidos cuando comienza en ese estado y siguiendo alguna política fija para alcanzar un estado terminal. Una política óptima sería entonces la relación de estados hacia las acciones que maximicen la suma de los refuerzos cuando comienza en un estado arbitrario y desempeña acciones hasta que un estado terminal es alcanzado. Bajo esta definición, el valor de un estado es dependiente de la política. La Función Valor es una relación de estados a valores de estado puede ser aproximado usando cualquier tipo de aproximador de funciones (perceptron de múltiples capas, tablas, sistemas basados en memoria, etc.).

Esto dirige a la pregunta fundamental de todas las investigaciones de aprendizaje por refuerzo: ¿Cómo se diseña un algoritmo que encuentre eficientemente la función valor óptima?.

Inicialmente la aproximación a la función valor óptima es muy pobre. En otras palabras, la aproximación de la relación de los estados a los valores de estado no es necesariamente válida. El objetivo primario del aprendizaje por refuerzo es encontrar la relación correcta. Una vez que esto es completado, la política óptima puede ser extraída fácilmente. En este punto, algunas notaciones necesitan ser introducida: V*(x) es la función de valor óptima, donde x_t es el vector de estados; V(x) es la aproximación de la función valor; es el factor de descuento en el rango de [0,1] que causa que el refuerzo inmediato tenga más importancia que el refuerzo futuro. En general, V(Xt) será inicializada con valores aleatorios y no contendrá información acerca de la función valor óptima V*(x,). Esto significa que la aproximación de la función valor óptima en un estado dado es igual al verdadero valor de ese estado V*(x,) más algún error en la aproximación, como se expresa en la ecuación (11•

$$V(x_t) = e(x_t) + V^*(x_t) \qquad (1)$$

Donde *e(x) es* el error en la aproximación del valor del estado ocupado en el tiempo *t,* de manera similar,

la aproximación en el tiempo $t+1$ es la siguiente:

$$V(x_{t+1}) = e(x_{t+1}) + V^*(x_{t+1}) \quad (2)$$

Como se mencionó previamente, el valor del estado xt para la política óptima es la suma de los refuerzos cuando comienza del estado x_t y desempeñando óptimas acciones hasta que un estado terminal es alcanzado. Por esta definición, una simple relación existe entre los valores de los sucesivos estados x_t y x_{t+1}. Esta relación es expresada por la ecuación Bellman (3). El factor de descuento es usado para decrementar exponencialmente el peso de los refuerzos recibidos en el futuro.

$$V^*(x_t) = r(x_t) + V^*(x_{t+1}) \quad (3)$$

La aproximación *V(xt)* además tiene la misma relación, como se muestra en la ecuación (4). Sustituyendo el lado derecho de las ecuaciones (1) y (2) en la ecuación (4) obtenemos la ecuación (5) y expandiéndola llegamos a la ecuación (6).

$$V(x_t) = r(x_t) + V(x_{t+1}) \quad (4)$$

$$e(x_t) + V^*(x_t) = r(x_t) + (e(x_{t+1}) + V^*(x_{t+1})) \quad (5)$$

$$e(x_t) + V^*(x_t) = r(x_t) + e(x_{t+1}) + V^*(x_{t+1}) \quad (6)$$

Usando la ecuación (3), V*(x) es sustraído de ambos lados de la ecuación (6) para revelar la relación en los errores de los estados sucesivos. Esta relación es expresada en la ecuación (7).

$$e(x_t) = e(x_{t+1}) \quad (7)$$

El proceso del aprendizaje es el proceso para encontrar una solución a la ecuación (4) para todos los estados *x,* (la cual es además la solución para la ecuación (7)). Muchos algoritmos han sido diseñados para esta tarea en particular.

Hay dos estrategias principales para resolver problemas de aprendizaje por refuerzo. El primero es buscar en el espacio de comportamientos para encontrar el que se desempeñe bien en el ambiente. Los algoritmos genéticos y la programación genética siguen esta corriente. La segunda es usar técnicas estadísticas y métodos programación dinámica para estimar la utilidad de ciertas acciones en el ambiente.

Q-Learning y Value Iteration son ciertamente los métodos más usados de programación dinámica.

Algoritmo Value Iteration

Según Harmon y Harmon (s.f.), si se asume que el aproximador de la función usado para representar *V** es una tabla (cada estado tiene su correspondiente elemento en la tabla cuya entrada es el valor de estado aproximado), entonces uno puede encontrar la función valor óptima haciendo barridos a través del espacio de estados, actualizando el valor de cada estado de acuerdo a la ecuación (8) hasta que un barrido a través del espacio de estados es ejecutado y no hay cambios en los valores de estado (los valores de estado han convergido).

$$(wt = \max u\ (r(x_t,u) + V(x_{t+1})) - V(x_t) \quad (8)$$

En la ecuación (8), *u* es la acción desempeñada en el estado *xty* causa una transición al estado xt+1y r(x,u) es el refuerzo recibido cuando desempeña la acción *u*en el estado x, La siguiente figura ilustra la actualización.

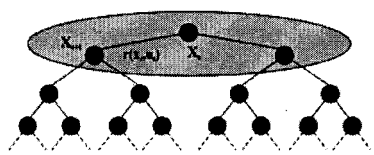

Figura i **Error! Argumento de modificador desconocido.**
Árbol de estados para Value Iteratio

La figura anterior describe el alcance de una simple actualización a la aproximación del valor x_t. Específicamente en este ejemplo, hay dos acciones posibles en el estado x_t, y cada una de estas acciones dirige a diferentes estados sucesores x_t+_1. En una actualización de Value Iteration, se debe encontrar primero la acción que devuelva el valor máximo. La única manera para lograr esto es ejecutar una acción y calcular la suma del refuerzo recibido y el valor aproximado del estado sucesor x_t, y no es posible sin un modelo de la dinámica del sistema.

Se debería notar que el lado derecho de la ecuación (8) es simplemente la diferencia en los dos lados de la ecuación de Bellman definida en la ecuación (4). Esta expresión es conocida como el residual Bellman, y está definida formalmente por la ecuación (9).

$$e(x_t) = \max u\ (r(x_t,u) + V(x_{t+1})) - V(xt) \quad (9)$$

e(x) es la función error definida por el residual de Bellman sobre todo el espacio de estados. Cada actualización (ecuación (8)) reduce el valor de e(x), y en el límite cuando el número de actualizaciones se acerca al infinito *e(x)* tiende a *O.* Cuando *e(x)=0,* la ecuación (4) es satisfecha y V(x)= V*(Xt) El aprendizaje es completado.

Algoritmo Q-Learning

Q-Learning (Watkins, 1989 y 1992) es otra extensión a la programación dinámica tradicional (Value Iteration) que resuelve el siguiente problema. Un proceso de decisión Markoviano (MDP según sus siglas en 'inglés) determinista es uno en el cual las transiciones de estado son deterministas (una acción desempeñada en un estado x siempre lleva al mismo estado sucesor x

En vez de encontrar una relación de estados a valores de estado (como en Value Iteration), Q-Learning encuentra una relación de pares situación-acción a valores (llamados Q-valores). En vez de tener una función valor asociada, Q-Learning hace uso de la función Q. En cada estado hay un Q-valor asociado con cada acción. La definición del Q-valor es la suma de los refuerzos recibidos cuando la acción es ejecutada siguiendo entonces la política dada. La definición de un Q-valor óptimo es la suma de los refuerzos recibidos cuando desempeñe la acción y siguiendo entonces la política óptima.

Según Touzet (1999) Q-Learning almacena la utilidad asociada a cada par situación-acción. Tres situaciones diferentes están involucradas: memorización, exploración y actualización. En respuesta a la presente situación, una acción es escogida por la función de evaluación con la ayuda de la memoria del robot. Esta acción es la que tiene la mayor probabilidad de dar la mayor recompensa. Tras la ejecución de la acción por el robot en el mundo real, una función de refuerzo proporciona un valor de refuerzo. Este valor, un simple criterio cualitativo (+1,-1,0), es usado por la función actualizadora para ajustar el valor de recompensa (Q) asociado al par situación—acción en la memoria del robot.

La función de evaluación

El algoritmo Q-Learning construye una función Q(10), que dado pares de situación—acción (i,a), devuelve los valores esperados *r: Q(a,i) es* el estimado del sistema del retorno que espera recibir dado el hecho de ejecutar una acción *a* en una situación j. El algoritmo usa una tabla para guardar la evaluación acumulativa.

$$Q(i, a)_{nuevo} = Q(i, a)_{viejo} + ß(r + g.Max(Q(i', a) - Q(i, a)_{viejo})) \quad (10)$$

Q-Learning difiere de Value Iteration en que no requiere que en un estado dado cada acción sea ejecutada y sean calculados los valores esperados de los estados sucesores. Mientras value Iteration ejecuta una actualización análoga a un barrido de un nivel, Q-Learning toma una simple muestra de un lance de dados Montecarlo. Este proceso es demostrado en la figura 6. ß representa la tasa de aprendizaje y debe ser mayor que 0; *g* es el factor de descuento usado para disminuir el impacto de la de la diferencia entre el *Q-valor* futuro con el *Q-valor* actual y debe estar entre O y 1.

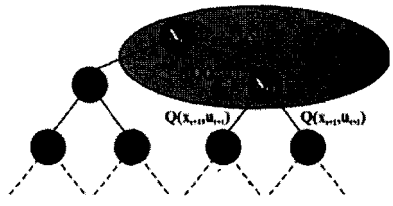

Figura ¡Error! Argumento de modificador desconocido.
Árbol de Estados vs. Acción

La ecuación de actualización en la figura 2 es válida cuando se usa una tabla para representar la función Q. El Q-valor es una predicción de la suma de los refuerzos que recibirá cuando ejecute la acción asociada y siguiendo entonces la política dada. Para actualizar esa predicción Q(Xt,u) se debe desempeñar la acción asociada ucausando una transición al próximo estado x y retornando un refuerzo escalar *r(x,u).* Entonces, se necesita encontrar sólo el máximo Q-valor en el nuevo estado para tener toda la información necesaria para revisar la predicción (Q-valor) asociado con la acción ejecutada. Q-Learning no requiere que se calcule la integral sobre todos los posibles estados sucesores en caso de que las transiciones no sean deterministas. La razón es que una simple muestra del estado sucesor para una acción dada es un estimado sin base del valor esperado del estado sucesor. En otras palabras, tras muchas actualizaciones el Q-valor asociado con una acción particular convergerá a la suma de todos los refuerzos recibidos cuando desempeñe esa acción y siguiendo después una política óptima.

Pasos básicos en el algoritmo Q-Learning

Touzet (1999) define los siguientes pasos básicos en el algoritmo Q-Learning:

1. **Inicialización** de la memoria del robot, para todos los pares de situación-acción, el valor Q asociado es cero (0). (Por ejemplo: Q(i,a)= O). También podría ser inicializada con valores aleatorios.

2. Repetir:

   a) Sea i una situación del mundo.

   b) La función de evaluación selecciona la acción a vara ejecutarse:

   $$a = \text{Max}(Q(i,a'))$$

   donde a'representa cualquier acción posible. El proceso de selección puede ser ligeramente diferente para que sea capaz de explorar nuevas opciones de exploración.

   c) El robot ejecuta la acción *a*en el mundo. Sea rla recompensa (rpuede ser O) asociada con la ejecución de *a* en el mundo.

   d) Actualiza la memoria del robot :

$$Q_{t+1}(i , a) = Q_t (i , a) + ß(r + g . \text{Max} (Q_t (i' , a')) - Q_t (i , a))$$

   donde ¡'es la nueva situación tras naoer sido ejecutada la acción *a* en la situación i, *a'* representa cualquier acción posible y O < , g <**1.**

Metodología

Las metodologías que se eligieron para la realización de esta investigación fueron el Modelo Lineal Secuencial y el Modelo Espiral. En la figura 3, se puede observar como' se combinaron ambas metodologías:

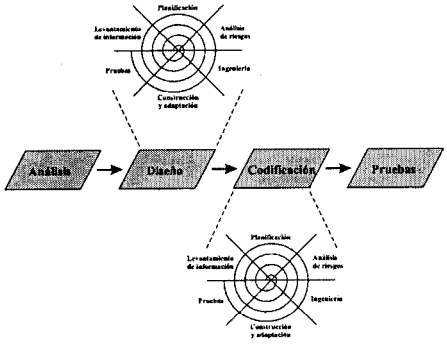

Figura ¡Error!Argumento de modificador desconocido.
Combinación de las metodologías Lineal Secuencial y Espiral

La metodología lineal exige seguridad en el análisis, la cual no está presente en ciertas fases de la elaboración de este trabajo. En varios puntos de la investigación (sobre todo en las fases de diseño y codificación) se tuvo que volver a la fase de levantamiento de información por lo que se optó por usar la metodología espiral en las fases de diseño y codificación. Solamente asegurando un diseño óptimo se podía garantizar un sistema autónomo completo en el tiempo estipulado para el trabajo especial de grado.

La metodología espiral fue dividida en espiras sucesivas y siempre se volvía a la fase **de** levantamiento de información de la espira, ya sea por pruebas fallidas, o por éxito en las mismas (en cuyo caso se procedía a la construcción ***o*** diseño de otro módulo). Cabe destacar que en cada espira no se diseñaba o construía completamente un módulo sino que se trabajaba sobre un prototipo el cual crecía conforme pasaba el tiempo. Sólo se procedía a la siguiente etapa del esquema secuencial una vez finalizadas todas las tareas previstas para esa fase del desarrollo del sistema. Es preciso señalar que este esquema fue utilizado tanto para hacer el prototipo del robot como la aplicación de simulación.

Desarrollo

FASE 1. ANÁLISIS

En primer lugar, se realizaron una serie de entrevistas con el Dr. Wilmer Pereira, Coordinador y Profesor del área de Redes y Sistemas Operativos de la Escuela de Ingeniería Informática de la UCAB, quien dirige el Grupo de Investigación de Inteligencia Artificial (GIIA) de la misma, para delimitar el alcance del proyecto, fijar objetivos y fechas para el trabajo especial de grado. Las primeras interrogantes que surgieron fueron:

¿Qué características debe tener el ambiente en el que se desenvuelve el robot? La idea que dio origen a este trabajo especial de grado fue el hecho de aprender a orientarse en un ambiente cuyas cuadras son desconocidas por el ente autónomo. 'Tomando como ejemplo al mundo real, específicamente el caso de las ciudades y sus cuadras, se llegó a la conclusión de que el ambiente debe conformarse por cuadras de figuras geométricas regulares, como lo muestra la figura 4.

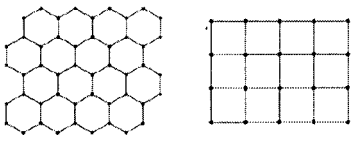

Figura ¡Error!Argumento de modificador desconocido.
Formas de las cuadras del ambiente

¿Qué algoritmos de aprendizaje por refuerzo se deberían implementarse en el robot y en la simulación?. Los requerimientos del sistema (ver las secciones Alcances y objetivos) son muy específicos en este aspecto, se debe programar al menos dos algoritmos de aprendizaje en la simulación de la computadora y uno de ellos en el robot, pero la interrogante es ¿Cuál de ellos?.

Q-Learning es un algoritmo de aprendizaje por refuerzo, basado en premios y castigos. Aplicado al problema de navegación, se premia al robot si éste se acerca al objetivo y se castiga si se aleja de él, hallando una política para encontrar el objetivo después de realizar varias corridas. Para hacer una analogía con el mundo real, si una persona va a un sitio muchas veces, esta acaba por aprender la ruta hacia su objetivo; las primeras veces puede que cometa errores hasta llegar pero tras un número determinado de viajes la persona acaba por aprender el camino hacia su destino.

Además de Q-Learning, se escogió otro algoritmo de aprendizaje para compararlos entre ellos y tras escuchar las recomendaciones hechas por el Profesor Pereira, se optó por escoger el algoritmo Value Iteration, aunque esta elección no fue tan directa como la anterior. Primero se propuso utilizar un algoritmo genético para el aprendizaje y se investigó acerca del mismo, pero tras la evaluación de los riesgos de la programación del algoritmo se llegó a la conclusión de que éste presentaba un alto riesgo, ya que era difícil aplicarlo al problema de navegación, y fue descartado.

Value Iteration es un algoritmo similar a Q-Learning, ambos provienen del principio de programación dinámica, pero aplican enfoques distintos para llegar al mismo resultado.

¿Cuál es el sistema operativo apropiado (Firmware) para programar el RCX?. Este es otro aspecto importante para el desarrollo del trabajo, se investigó acerca de cuatro posibles sistemas operativos para programar el RCX: BrickOS (conocido anteriormente como legOS), NQC, Robotic Invention System (RIS) y LejOS.

Se eligió LejOS por ser un lenguaje netamente orientado a objetos, cuyo mantenimiento a la larga resultaría más fácil que en un lenguaje estructurado como C, además está disponible para una amplia variedad de plataformas y resulta más fácil de instalar en la computadora.

¿Bajo cuál lenguaje programar la simulación de los algoritmos de aprendizaje? El lenguaje de programación elegido para programar la simulación de los algoritmos Value Iteration y Q-Learning fue Visual Basic, bajo el entorno de programación Microsoft Visual Studio 6.0.

FASE 2. DISEÑO

En la segunda fase de la metodología lineal se debe elaborar representaciones del software a desarrollar, así como también diseñar y probar diferentes tipos de estructuras físicas para el ente autónomo, que en este caso es un robot. En esta fase se aplica también la metodología en espiral para el diseño del robot y de la implementación del algoritmo Q-Learning en el RCX así como también la de los algoritmos Q-Learning y Value Iteration para la simulación.

Espira 1

Se construyó el primer prototipo del robot, usando dos orugas y dos motores, como se muestra en la figura 5. Pero tras sucesivas pruebas, se observó que el robot perdía exactitud en el movimiento. Esto es debido a que la velocidad de los motores es ligeramente diferente uno del otro, lo que ocasionaba que se desviara en algún momento si recorría un tramo en línea recta. Se tuvo que volver de nuevo a investigar para buscar alguna forma de evitar la inexactitud de los motores.

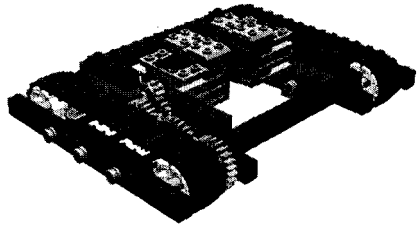

Figura ¡Error!Argumento de modificador desconocido.
Prototipo de robot construido con orugas

Espira 2

Debido a la inexactitud en el movimiento observado en la espira anterior, se procedió a investigar otros diseños de robots Lego que ofrecieran mayor precisión

en los movimientos. Una de las arquitecturas encontradas para el movimiento del robot fue la de diferenciales duales, tal y como lo muestra la figura 6.

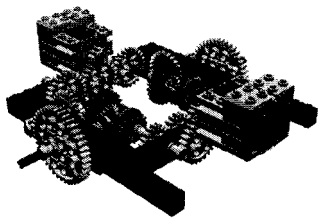

Figura ¡Error!Argumento de modificador desconocido.
Sistema de transmisión construido con diferenciales duales

En este sistema un motor se encarga de mover el robot en línea recta (hacia adelante o hacia atrás), mientras que el segundo motor se encarga de realizar los giros (hacia la izquierda o hacia la derecha) y sólo uno de los motores debe trabajar a la vez, no pueden trabajar los dos al mismo tiempo. La ventaja de utilizar este sistema radica en que ambas ruedas trabajan a la misma velocidad manteniendo en lo posible un movimiento optimo.

Espira 3

Una vez resuelto el problema de la exactitud de los movimientos, se escogió el tipo de ruedas para el robot. Inicialmente se pensó usar orugas, pero éstas producían un efecto indeseable en los giros, ya que éstas tienden a desplazarse hacia los lados y no sobre el mismo eje, lo cual es indispensable para mantener al robot orientado en todo momento. Tras probar con varios tipos de configuraciones (4 ruedas, 2 ruedas grandes, ruedas pequeñas, etc.) se tomó la decisión de utilizar dos ruedas.

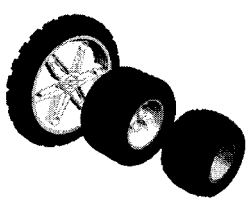

Figura **¡Error!Argumento de modificador desconocido.**. Tipos de ruedas

Para la parte delantera se consideraron diversas opciones, entre ellas una tercera rueda adelante, una rueda loca, etc., pero estas traían inconvenientes al movimiento, es decir, la rueda loca entorpecía la dirección cuando se movía en línea recta, y la opción de la tercera rueda producía tanto roce en los giros que ocasionaban problemas con los mismos. Finalmente se colocó un soporte que fijaba la parte delantera del robot, estabilizándolo. La parte inferior de la pieza actúa como un esquí (véase figura 8), minimizando el roce con el piso y logrando un movimiento estable con un nivel óptimo de exactitud tanto en los giros como en los movimientos rectos.

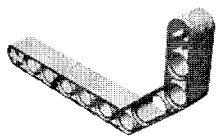

Figura ¡Error!Argumento de modificador desconocido.
Soporte en forma de esquí

Espira 4

Para lograr captar los eventos del ambiente, se hizo un estudio para definir qué sensores se utilizarían para tal fin. Inicialmente se pensó utilizar tres sensores ultrasónicos (véase figura 9), colocados uno en la parte delantera del robot, y los otros dos en los lados laterales del mismo, pero surgieron graves inconvenientes con los sensores ya que las lecturas, en muchos casos, no reflejaban la situación real.

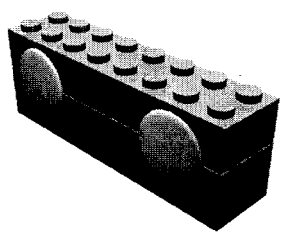

Figura ¡Error!Argumento de modificador desconocido.
Sensor ultrasónico

Para tratar este problema era necesario hacer un estudio probabilístico de las lecturas. Además, no había manera exacta de reconocer las intersecciones en el camino ya que sólo detectan superficies que estén al frente del sonar, así como su distancia; y tampoco a que ángulos se encuentran los caminos o cuanto ha girado el robot. Estos detalles complican excesivamente el problema y como se verá más adelante, hay formas más sencillas de reconocer los posibles caminos, por lo que fue descartada esta configuración.

Otra configuración propuesta fue la de colocar un sonar en la parte delantera del robot, un sensor de luz justo abajo del robot apuntando hacia el suelo y un sensor de rotación. Esto soluciona parte de los problemas mencionados anteriormente, como lo son el reconocimiento de caminos e intersecciones, ya que el sensor de luz mantiene ajustado el robot al camino, es decir, si el camino es de un color determinado, por ejemplo azul, y luego detecta otro color, por ejemplo blanco, éste se ajusta para seguir el camino. El sensor de rotación permite saber cuantos grados ha girado el robot, siendo un elemento importante para la navegación del robot. No obstante, el problema persiste con el sonar, es sumamente difícil lidiar con las lecturas del mismo y solamente se utilizaría en las intersecciones para detectar aberturas (caminos). En su lugar se colocó otro sensor de luz apuntando al suelo en la parte delantera del robot, el cual permite detectar de una manera mas rápida y sencilla cuantos caminos hay en cada intersección.

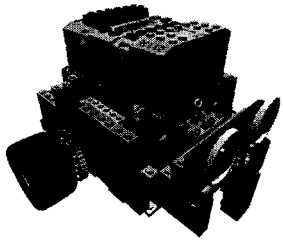

Figura ¡Error!Argumento de modificador desconocido.
Versión finalizada del robot

Espira 5

Una vez finalizada la estructura física del robot, se procedió a la construcción del ambiente. Con la eliminación del sonar, se decidió eliminar las paredes y se optó por colocar solamente la superficie, los caminos y las intersecciones, identificados con el color blanco, azul y negro, respectivamente.

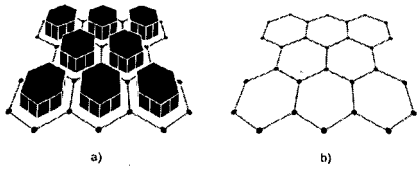

Figura ¡Error!Argumento de modificador desconocido.
Tipos de ambiente. a) Con paredes. b) Sin paredes

FASE 3. CODIFICACIÓN

Esta etapa marca el inicio de la programación.

Espira 1

Tras el levantamiento de requerimientos, además del diseño del robot y el ambiente, se procedió a la programación del algoritmo de aprendizaje por refuerzo Q-Learning en el RCX. Para ello, se utilizó la herramienta JCreator LE, el cual permite trabajar en el entorno Java de una manera fácil y cómoda.

Pero, ¿cómo aplicamos Q-Learning al problema de navegación?. El objetivo es aprender a llegar a un punto específico del mapa desde un punto de inicio. En cada intersección se debe decidir entre una serie de caminos, algunos alejan al robot de la meta, otros lo acercan y algunos hasta incluso mantienen la misma distancia. Si una acción en ese estado acerca al robot, esta acción debe ser recompensado, mientras que si el robot se aleja, la acción debe ser penalizada. El ente autónomo debe aprender a llegar a su destino escogiendo las acciones que ofrezcan una mayor recompensa.

Q-Learning requiere elaborar una matriz (llamada matriz Q) de acciones contra estados. Las acciones en este caso son: rotar un ángulo específico y moverse en línea recta. La figura 12 muestra algunas de las posibles acciones que puede tomar el robot en una intersección a tres posibilidades:

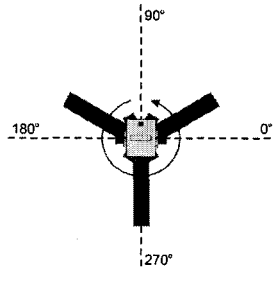

Figura ¡Error!Argumento de modificador desconocido.
Acciones que puede realizar el robot en un tipo de intersección

En este punto, el robot puede girar hacia el camino que está a 30°, 150° o 270° con respecto a la horizontal. En la matriz Q, estas acciones son denominadas por el valor del ángulo que se debe girar.

Los estados representan situaciones que pueden ocurrir en la interacción del robot con el ambiente y pueden ser valores arrojados directamente por los sensores, combinaciones o abstracciones derivadas de las lecturas de los sensores o representaciones internas del robot.

El diseño de estados propuesto fue uno donde el robot construyera un sistema de coordenadas donde el origen es el punto de partida (véase figura 13) y las coordenadas del objetivo ya son fijas y conocidas por el robot previamente, lo cual fue tomado como hipótesis del problema planteado. Usando estas referencias, y mediante cálculos trigonométricos, se calcula la distancia entre el robot y el objetivo.

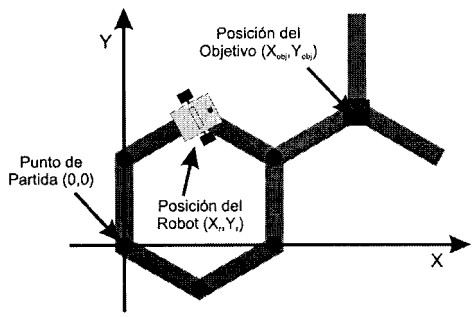

Figura ¡Error!Argumento de modificador desconocido.
Sistema de coordenadas utilizado por el robot

También es necesario saber como está el robot con respecto al objetivo, es decir, si está por arriba, por abajo, a la derecha ó a la izquierda del mismo. Para obtener esa información se restan las coordenadas del objetivo con las coordenadas del robot, por ejemplo, si la resta de las abscisas y de las ordenadas son negativas, el robot se encuentra por encima y a la izquierda del objetivo (cuadrante I de la figura 14).

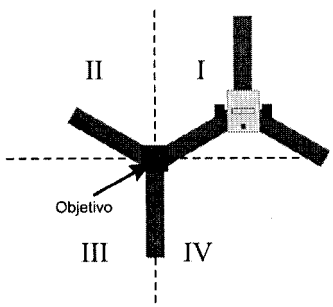

Figura ¡Error!Argumento de modificador desconocido.
Posición del robot con respecto al objetivo

Los estados vienen dados por las combinaciones entre los cuadrantes y la diferencias entre la distancia del robot al objetivo. En la tabla 1 se muestran todos los estados posibles que maneja la matriz Q donde dant indica la distancia del robot al objetivo cuando estaba en la intersección anterior y $_{dact}$ indica la distancia del robot al objetivo en la intersección actual.

Tabla ¡Error!Argumento de modificador desconocido..
Estados posibles para la matriz Q

Estado	Cuadrante	Distancias
0	1	$d_{ant} < d_{act}$
1	1	$d_{ant} > d_{act}$
2	2	$d_{ant} < d_{act}$
3	2	$d_{ant} > d_{act}$
4	3	$d_{ant} < d_{act}$
5	3	$d_{ant} > d_{act}$
6	4	$d_{ant} < d_{act}$
7	4	$d_{ant} > d_{act}$

Finalmente, la matriz Q (Tabla 2) fue construida usando los estados mencionados anteriormente, y se tomaron como acciones los ángulos desde 0° hasta 360° numerados de 10 en 10, es decir, 0°, 10°, 20°, etc. Las filas son los estados y las columnas son las acciones.

Tabla **i Error !Argumento de modificador desconocido.**
Matriz Q

0	0,48		1	0,93	0,23
1	0,70	0,60 ..		0,67	0,37
2	0,97	0,45 ..		0,23	0,87
3	0,65	0,74 ..		0,62	0,34
4	0,10	0,76 ..		0,34	0,73
5	0,29	0,56 ..		0,89	0,66
6	0,69	0,77 ..		0,83	0,65

Cada estado tiene asociado una serie de ángulos que representan las acciones posibles. Por ejemplo, si el robot llegó a una intersección que lo acercó al objetivo y está en el cuadrante 3, el estado que representa esta acción es el estado 5 (ver tabla 1 de estados); una vez identificado el estado, determina la acción que va a tomar buscando el mayor valor de la fila para ese estado. En la tabla 2 se puede apreciar que el valor máximo para el estado 5 corresponde al ángulo de 350° por lo que la acción que va a tomar el robot será la de girar hasta 350° para después moverse en línea recta hasta la siguiente intersección. Estos valores usados para determinar el máximo, son denominados Q-valores, y representan la utilidad de un par (estado-acción).

Una vez hecho esto y llegar a la siguiente intersección, el robot identifica nuevamente el estado al que llegó, calcula la recompensa de la acción tomada en el estado anterior y vuelve a realizar los mismos pasos ya explicados en el párrafo anterior. Una recompensa positiva toma lugar cuando el robot ejecuta una acción que lo acerca al objetivo, premiándola, mientras que una recompensa negativa toma lugar cuando en vez de acercarse al objetivo éste se aleja, por lo que la acción ejecutada es penalizada (véase figura 15). Con la recompensa calculada, se actualiza el Q valor (valor en la matriz Q) correspondiente al par (estado anterior, acción) modificando los valores que pueden afectar a decisiones futuras para ese estado.

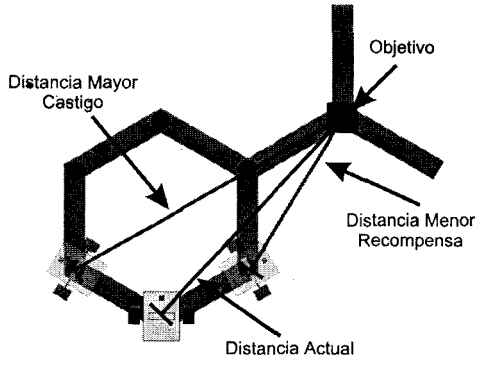

Figura ¡Error!Argumento de modificador desconocido. Recompensa de las acciones posibles que puede tomar el robot

Además del RCX, se realizó también una aplicación de Q-Learning para la simulación en la computadora. Para el RCX se crearon las siguientes clases:

- **RotationNav.java:** Se encarga de realizar los movimientos del robot, los giros, movimientos en línea recta, trasladarse de un punto a otro, etc. Utiliza el sensor de rotación para mantenerse orientado y saber a que ángulo se encuentra con respecto a la horizontal.
- **Qlearning.java:** Es la clase más importante, ya que es la encargada de calcular los estados del robot, recompensar las acciones tomadas en cada estado y de almacenar en la matriz Q, es decir, todó lo que tenga que ver con el aprendizaje del robot.
- **Huron.java:** Es la clase principal y es la encargada de manejar y coordinar todas las llamadas a las otras dos clases explicadas anteriormente.

Espira 2

Tras programar el algoritmo Q-learning en el RCX y en la simulación, fue necesario programar el algoritmo Value Iteration en la simulación para poder hacer las comparaciones entre ellos.

Es pertinente exponer primero como puede ser aplicado Value Iteration al problema de navegación. Value Iteration trabaja sobre un proceso de decisión Markoviano determinista (véase Capitulo II), es decir, la acción tomada en un estado siempre lleva a un mismo estado sucesor, mientras que en Q-Learning, la acción tomada en un estado podía ser cualquiera de los otros estados, que no siempre es el mismo. Por tal motivo, se tenía que cambiar el esquema de acciones y estados para que Value Iteration pudiera ser aplicado.

El esquema de acciones es el mismo que fue utilizado para Q-Learning, pero los estados se tuvieron que implementar de una manera distinta. Se optó por representar los estados como las intersecciones del mapa principal, identificadas por números enteros.

Una vez definidos los estados y acciones, hay que proceder a elaborar la base de conocimiento. El algoritmo sólo necesita conocer la utilidad (Valor V) por estado como se muestra en la tabla 3.

Tabla ¡Error!Argumento de modificador desconocido. Base de conocimiento utilizada

	1	2	3	....	49	50
1	V(1)	90	30	....	-1	-1
2	270	V(2)	-1	....	-1	-1
3	190	-1	V(3)	....	-1	0
....	....	....	....	....	....	....
49	-1	-1	-1	....	V(49)	0
50	-1	-1	-1	....	180	V(50)

El esquema de recompensas se mantiene igual a Q-Learning, es decir, si el robot se acerca a la meta éste es recompensado, pero, si se aleja éste es penalizado.

La aplicación de Value Iteration es relativamente sencilla. Cuando el ente autónomo llega a una nueva intersección evalúa los diferentes caminos que puede tomar en esa una intersección. Luego evalúa cómo es la recompensa que recibe por cada camino posible sumado con el valor Vdel estado al que llega si ejecuta esa acción. Una vez realizadas todas estas operaciones, se elige el camino cuya suma sea la

mayor, se sustituye el valor Vdel nodo actual con el del camino elegido y se recorre el camino elegido, para repetir todos los pasos mencionados anteriormente en este párrafo hasta que halla el objetivo.

Debido a que el aprendizaje se realiza en cada estado el robot considera que ha aprendido en esa situación cuando el valor V anterior no difiere del valor V actual, es decir, el error entre ellos es cero.

FASE 4. PRUEBAS

En esta última fase del método lineal secuencia) se realizaron pruebas sucesivas para la detección de errores, se observó el desempeño del robot en el medio ambiente y el comportamiento del algoritmo Q-learning aplicado para la navegación. Este proceso fue relativamente corto, ya que al aplicar la metodología espiral en las fases anteriores, estas pruebas se realizaron en espiras pasadas.

No obstante, esta fase se centró más que todo en establecer las comparaciones necesarias entre Value Iteration y Q-Learning usando la aplicación de la simulación para tal fin. Es importante mencionar que una iteración comienza con el inicio del robot en el punto de partida y finaliza cuando éste llega al objetivo. Cada corrida puede tener asociado un número finito de iteraciones.

Resultados

El principal objetivo de este proyecto fue el de desarrollar y simular un software que permitiese desplazar a un ente autónomo en un ambiente formado por cuadras de figuras geométricas regulares. Se construyó un robot capaz de moverse en distintos tipos de ambientes y de captar eventos externos al mismo, utilizando para ello sensores de luz y de rotación. Las secuencias de movimientos generadas y programadas en el robot se comportaron de manera exitosa y las entradas de los sensores permitieron identificar y obtener datos del medio ambiente, siendo ésta información importante para la base de conocimiento utilizada para el aprendizaje, tanto para el robot como para la simulación en la computadora.

El algoritmo de aprendizaje por refuerzo *Q-Learning* mostró ser una buena elección para los problemas de navegación. Se hicieron varias corridas con el robot construido con el kit de LEGO Mindstorm y se pudo observar que el algoritmo lograba alcanzar la meta la mayor parte de las veces, pero hubo momentos en los cuales el desempeño del mismo se vió afectado por eventos externos aleatorios y por los errores que se presentaban en las lecturas de los sensores de luz, debido a que no son del todo precisos.

Con respecto a la simulación, se programaron los algoritmos Q-Learning y Value Iteration, donde se pudo observar el comportamiento de cada uno ellos en un ambiente ideal, para así poder realizar comparaciones entre ambos algoritmos, evitando así los problemas con la imprecisión de los sensores. Una de las ventajas de utilizar ésta simulación es que es posible ver de una manera rápida y sencilla, como se realiza el aprendizaje, obviando las imperfecciones de los sensores. Además, se pueden hacer modificaciones en ciertas partes de los algoritmos, sin alterar la lógica de los mismos.

A continuación se muestran los resultados de la comparación entre Q-Learning y Value Iteration obtenidas de la simulación. Una corrida es un conjunto de iteraciones y una iteración abarca desde que el robot empieza la búsqueda hasta que alcanza la meta. La cantidad de iteraciones indica cuantas veces fue ejecutado cada uno de los algoritmos.

Prueba 1

Se realizó una corrida de 20 iteraciones, manteniendo el objetivo fijo, con cada uno de los dos algoritmos para ver como era la curva de aprendizaje en ambos casos. La gráfica obtenida fue. la siguiente:

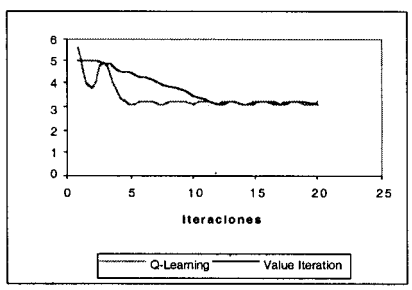

Figura ¡Error! Argumento de modificador desconocido.
Curva de aprendizaje Q-Learning y Value Iteration

Aquí se puede observar que Q-Learning converge de manera más rápida que Value Iteration, es decir, tarda menos tiempo en aprender a llegar al objetivo mientras que a Value Iteration le toma mas tiempo hacerlo, pero a la larga ambos hallan una política de movimiento eficiente y que en algunos casos puede ser la misma, pues los dos convergen.

Prueba 2

Dado que la matriz Q puede ser inicializada con valores aleatorios o con valores en cero, se realizaron dos corridas, en donde una de ellas fue inicializada con valores aleatorios y la otra con ceros. El resultado fue el siguiente:

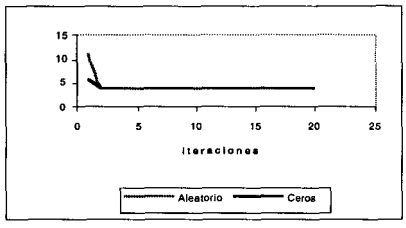

Figura20. Curva de aprendizaje Q-Learning

Se puede inferir de ésta gráfica que no importan los valores con que se inicializa la matriz Q, ya que siempre va a converger de manera similar, es decir, la forma en como el robot llega al objetivo. Lo único que podría cambiar es el tiempo que tarda en aprender a llegar a la meta en las primeras iteraciones.

Prueba 3

Se realizaron 4 corridas del algoritmo Q-Learning, de 20 iteraciones cada una, manteniendo al objetivo fijo e inicializando la matriz Q con valores aleatorios en cada corrida. Estos números aleatorios son generados por la aplicación y varían en cada iteración utilizando la sentencia Randomize de Visual basic, que es la encargada de calcular la semilla que se utilizará para la generación de números aleatorios. El resultado de la corrida se puede apreciar en la siguiente gráfica:

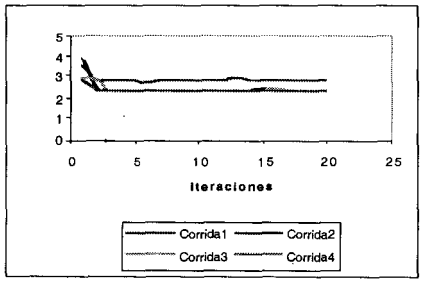

Figura **¡Error!Argumento de modificador desconocido.**. Curva de aprendizaje Matriz Q aleatoria

Se puede observar que Q-Learning mantiene un comportamiento parecido en cada corrida, es decir, los valores con que se inicializa la matriz Q no influyen de manera significativa en el desempeño del algoritmo. Cabe destacar que con el uso de valores aleatorios en la matriz Q, el algoritmo puede hallar diferentes políticas de movimiento que ocasionan que el robot tarde más o menos tiempo en llegar al objetivo. Por ejemplo, la corrida 1 encontró una política de movimiento que tardó más tiempo en alcanzar la meta que las encontradas en las otras corridas.

Prueba 4

Se realizaron 4 corridas del algoritmo Q-Learning, de 20 iteraciones cada una, manteniendo al objetivo fijo e inicializando la matriz Q con ceros en cada corrida. El resultado se puede apreciar en la siguiente gráfica:

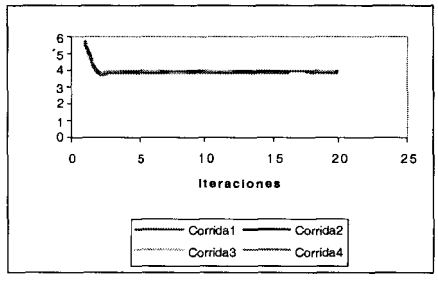

Figura **¡Error!Argumento de modificador desconocido.**. Curva de aprendizaje Matriz Q en cero

En este caso, cada una de las corridas convergieron a una misma política de movimiento, por lo que el tiempo que tomaba el robot en llegar a la meta es el mismo para todas las corridas, es decir, el algoritmo muestra un comportamiento uniforme utilizando la matriz inicializada con ceros.

Prueba 5

Se realizaron 2 corridas del algoritmo Q-Learning de 40 iteraciones cada una, cambiando la posición del objetivo después de cada 10 iteraciones y manteniéndolo fijo durante esas 10 iteraciones. Para la primera corrida se trabajó con la matriz inicializada con valores aleatorios y para la segunda corrida se inicializó la matriz con ceros. La gráfica resultante fue la siguiente:

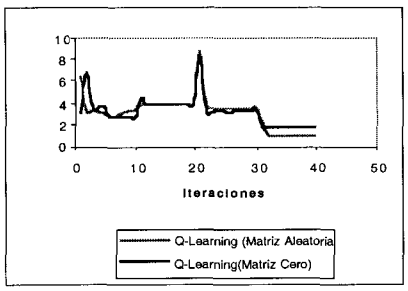

Figura ¡Error!Argumento de modificador desconocido. Curva de aprendizaje Q-Learning variando objetivo

Aquí se puede observar que el comportamiento de Q-Learning para ambos casos es semejante, es decir, logran establecer casi el mismo tiempo de llegada al objetivo. Los picos representan el punto en donde el objetivo fue cambiado de posición y debido a esto el algoritmo toma más tiempo para el aprendizaje al tratar de adaptarse al nuevo cambio.

Prueba 6

Se realizaron 2 corridas de 40 iteraciones cada una, cambiando la posición del objetivo cada 10 iteraciones. Para la primera corrida se trabajó con el algoritmo Q-Learning y para la segunda el algoritmo Value Iteration. La gráfica resultante fue la siguiente:

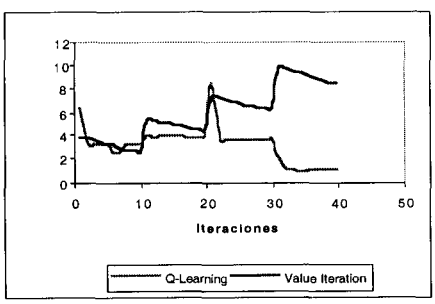

Figura ¡Error!Argumento de modificador desconocido. Curva de aprendizaje Q-Learninig y Value Iteration variando objetivo

Como se puede observar, Q-Learning se adapta a los cambios de una manera rápida, tal y como se vio en la gráfica anterior. Sin embargo, Value Iteration, una vez que dio por finalizado el aprendizaje de la ruta hasta el objetivo, al presentarse un cambio del mismo, el tiempo que tarda en llegar al objetivo es equivalente al tiempo que tardó en llegar a la posición anterior del objetivo más el tiempo que necesita para volver a aprender a llegar a la nueva posición de la meta, es decir, pasa por donde se encontraba el objetivo anterior y desde ahí empieza a buscar el nuevo objetivo.

Es importante notar que Value Iteration, por guardar la ruta en un grafo de estados, se ve afectado críticamente si se ve obligado a pasar por un nodo o intersección ya visitado, ya que cada uno de éstos tiene asignada una acción que fue resultado del aprendizaje anterior, lo que ocasiona se tome el mismo camino cada vez que se llegue a ese estado y no se realice el proceso de aprendizaje para esa intersección, evitando llegar a la nueva posición del objetivo.

Prueba 7

Debido al problema expuesto anteriormente para el algoritmo de Value Iteration y por el tiempo que toma en aprender a llegar al nuevo objetivo, se decidió hacer una modificación que consistió en eliminar la condición de fin de aprendizaje, es decir, que se realice el proceso de aprendizaje en cada nodo o intersección visitada. Para este caso también se realizaron 2 corridas cambiando el objetivo de posición, de 20 iteraciones cada una, utilizando para la primera el algoritmo Q-Learning y para la segunda el algoritmo Value Iteration. La gráfica resultante se muestra a continuación:

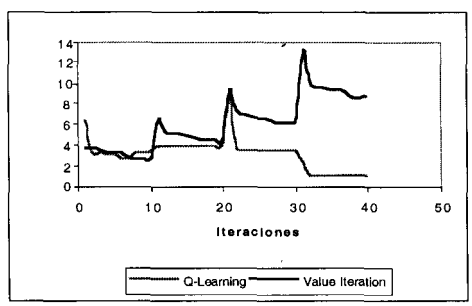

Figura ¡Error!Argumento de modificador desconocido. Curva de aprendizaje Q-Learninig y Value Iteration modificado variando objetivo

Lo que se observó fue que no hubo un cambio significativo en el comportamiento de Value Iteration, éste seguía los mismos patrones de comportamiento que el caso anterior, sólo que el aprendizaje se hacía en todo momento, es decir, en cada intersección visitada o no.

Prueba 8

Por último, se realizó una segunda modificación en el algoritmo Value Iteration, que consistía en reinicializar el grafo cada vez que se cambiaba el objetivo de posición, manteniendo la condición de fin de aprendizaje. También se realizaron 2 corridas de 20 iteraciones cada una, cambiando el objetivo de posición y utilizando para la primera corrida el algoritmo Q-Learning y para la segunda el algoritmo Value Iteración.

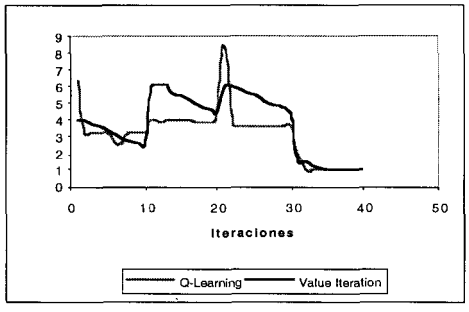

Figura ¡Error! Argumento de modificador desconocido. Curva de aprendizaje Q-Learninig y Value Iteration modificado variando objetivo

Para este caso, se logró que el algoritmo Value Iteration se adaptara mejor a los cambios de posición del objetivo, pero que en comparación con Q-Learning, sigue tomando más tiempo de aprendizaje que el utilizado por Q-Learning. La desventaja de esto es que ya no guarda su conocimiento si debe recorrer el ambiente de nuevo.

Cabe destacar que el algoritmo Value Iteration siempre busca la ruta mas corta, mientras que Q-learning sólo trata de llegar al objetivo según una política de movimientos producto del aprendizaje, por lo que hubo momentos en los cuales Value Iteration visitaba menos intersecciones que las visitadas por Q-Learning

Conclusiones

A lo largo de este trabajo se estudió a fondo diferentes aspectos de la inteligencia artificial aplicados al área del aprendizaje, específicamente, el aprendizaje por refuerzo. Los mismos permiten resolver una gran variedad de problemas en los que el aprendizaje es un factor importante para la búsqueda de una solución.

La utilización del kit LegoMindStorm resultó apropiada para la construcción del prototipo del robot, debido a que permite construir fácil y rápidamente una gran variedad de modelos de robots adaptables a diversos problemas. Este kit posee varios sensores, entre los cuales destacan los sensores de luz y de rotación, que fueron usados para el robot. **En** particular, hay que destacar que el sensor de rotación posee una notable exactitud en los valores medidos, en contraste con los sensores de luz, cuyas lecturas se vieron afectadas por factores externos como la luz ambiental.

Una desventaja del kit es la cantidad de entradas que tiene el RCX, ya que solo permite conectar tres sensores a la vez, limitando la funcionalidad de los robots y por ende la complejidad del programa que se desee implementar en el mismo. Además, posee una cantidad limitada de memoria (32Kb), lo que no permite implementar programas muy extensos. Además, los firmware o sistemas operativos, ocupan espacio en la memoria, disminuyendo aún más la capacidad para almacenar programas de usuario.

Con respecto al sistema operativo del RCX (firmware), la elección de utilizar lejOS fue conveniente por varias razones. En primer lugar, éste sólo ocupa 16 Kb de la memoria, dejando disponibles los restantes 16Kb de memoria para programas de usuario, mientras que BrickOS (legOS) ocupa 20 Kb, dejando 12 Kb de memoria para programas de usuario. En segundo lugar, lejOS es un firmware basado en el lenguaje de programación Java, heredando las potentes capacidades que éste lenguaje proporciona.

El problema de la navegación para encontrar un objetivo se logró resolver empleando un algoritmo de aprendizaje por refuerzo. Existen otros tipos de técnicas de aprendizaje, como el aprendizaje evolutivo, que involucra el uso de algoritmos genéticos y que pueden ser utilizados para resolver problemas de navegación. Éstos algoritmos son lentos y sólo permiten encontrar una solución cercana a la óptima, sin ninguna garantía de convergencia o de la calidad de la solución encontrada. Tardan mucho tiempo en procesar las soluciones, siendo esto un factor limitante para la implementación del mismo en el robot, ya que éste cuenta con una capacidad de memoria limitada.

No obstante, los algoritmos de aprendizaje por refuerzo son más rápidos y fáciles de implementar y no requieren una gran cantidad de recursos del sistema, facilitando su implementación en el RCX. Entre las diferentes técnicas investigadas con respecto a este tipo de aprendizaje se estudiaron en

profundidad los algoritmos Q-Learning y Value Iteration (simulados) y siendo Q-Learning el implementado físicamente en el robot,

Value Iteration al igual que Q-Learning, está basado en el principio de la programación dinámica, el cual permite realizar un aprendizaje efectivo a través de premios y castigos. Las ventajas de usar Value Iteration son las siguientes:

- No necesita probar todos los estados posibles en el sistema, sino que éste evalúa los estados sucesores inmediatos.
- La solución hallada tiende a ser en la mayoría de los casos la óptima.
- Si el ambiente cambia no necesita ser reprogramado.

Como desventajas del algoritmo Value Iteration se puede mencionar las siguientes:

- Sólo puede ser aplicado bajo esquemas estado-acción donde, la acción tomada en un estado dado siempre conduce al mismo estado sucesor.
- Evalúa todas las acciones posibles en un estado dado, lo que hace más lento el proceso de aprendizaje.
- Una vez finalizado el aprendizaje en un estado dado, el algoritmo considera que no se debe volver a realizar el aprendizaje en ese estado, en caso que regrese a ese punto.
- Si el objetivo cambia de posición, el algoritmo puede que nunca alcance el objetivo o que sea posible que el aprendizaje tome mucho más tiempo, ya que todos los caminos que hayan sido tomados y luego aprendidos por el robot, éstos serán tomados siempre aún cuando el objetivo este ubicado en algún otro punto, dicho de otra manera, las rutas aprendidas serán siempre recorridas por el robot para que luego desde ese punto, se aprenda otra ruta que lo lleve al nuevo objetivo.

Con respecto a Q-Learning, las principales ventajas de utilizar éste algoritmo son las siguientes:

- Puede actuar tanto en el esquema estado-acción mencionado anteriormente, como en uno donde las acciones que puedan ser tomadas en un estado dado, no siempre conduzcan a un mismo estado sucesor.
- Si el ambiente cambia no necesita ser reprogramado.
- Si el objetivo cambia de posición Q-Learning ajusta eficientemente su aprendizaje, logrando llegar siempre al objetivo.
- Si la matriz Q es inicializada con valores aleatorios el robot puede experimentar otros caminos que amplíen su aprendizaje. Esto debido a que el algoritmo busca el valor Q máximo para las acciones posibles de un estado, el cual puede cambiar habiéndose inicializado la matriz con valores aleatorios.
- La curva de aprendizaje tiende a converger de forma más rápida que Value Iteration, aunque de manera menos uniforme.
- El proceso de elegir una acción es casi inmediato.

Las desventajas de utilizar Q-learning son:

- La solución hallada no siempre es la óptima, aunque tiende a estar muy cerca de la misma.
- Para que el aprendizaje sea más efectivo, el algoritmo debe evaluar la mayor cantidad de estados posibles.

También se observó que al aplicarse ciertos cambios de condiciones en el algoritmo Value Iteration se mejoró el desempeño del mismo cuando se cambiaba el objetivo de posición (ver Capitulo VI), pero seguía siendo más lento que Q-Learning, aunque seguía hallando la solución óptima.

En este punto se podría preguntar; ¿Bajo qué condiciones se debería elegir uno de estos dos algoritmos de aprendizaje por refuerzo?, La respuesta viene dada dependiendo de la situación. Si el tiempo para llegar al objetivo no es problema se puede aplicar Value Iteration, en caso contrario Q-Learning es la mejor opción. Sin embargo, pára ambos casos, la ruta optima que encuentra Value Iteration es semejante a la ruta que encuentra Q-Learning, variando muy poco una con respecto a la otra.

Cuando se diseñan programas basados en algoritmos de aprendizaje por refuerzo es necesario definir y diseñar de manera detallada los estados, acciones y la política de recompensas, ya que éstos factores juegan un rol muy importante en el funcionamiento del mismo. Si alguno de estos factores falla el desempeño del algoritmos puede verse seriamente afectado, o peor aún, podría no llegar a ninguna solución.

Bibliografía

Bagnall, B. (2002). CORE Lego MindstormsTM Programming. Edit. Prentice Hall PTR. Nueva York, Estados Unidos de América.

Ferrari, G., Gombos, A. Hilmer, S., Stuber, J., Porter, M., Waldinger, J. y Laverde, D. (2002). Programming LEGO MindstormsTM with Java. Edit. Sysgress. Estados Unidos América.

Rich, E. y Knight, K. (1994). Inteligencia Artificial. 2da Edición. Edit. McGraw Hill. Madrid.

Russel, S. y Norvig, P. (1996). Inteligencia Artificial Un enfoque moderno. 1 ra Edición. Edit. Prentice Hall. Edo. De México, 1996.

Pressman, Roger S. (1998). Ingeniería del Software un enfoque práctico. 4ta Edición. Edit. McGraw Hill. Madrid, España.

Carrasquero Z., Oscar H. y McMaster F., Eduardo (2002). Diseño y construcción de un robot con el módulo RCX 1.0 para ambientes no predeterminados (Tesis de Ingeniero en Informático, Universidad Católica Andrés Bello).

Bagnall, Bryan (2001). Lejos: Java for the RCX. [página web en línea] Consultado en 10/5/2003 Disponible en http://lejos.sourceforge.net/.

(2000). BrickOS Home Page. [página web en línea] Consultado en 2/5/2003 Disponible en http:// brickos.sourceforge.net/.

Dartmouth College Computer Science Department (2001). Robo-Rats Locomotion: Dual Differential Drive. [página web en línea] Consultado en 20/ 5/2003 Disponible en http:// www.cs.dartmouth.edu/-robotlab/robotlab/ courses/cs54-2001s/dualdiff.html

Gross M., Stephan V. y Boehme J. (1996). Sensory based robot navigation using self-organizing networks and Q-Learning. [documento en línea] Consultado en 12/9/2003 Disponible en http://citeseer.nj.nec.com/ gross96sensorybased.html.

Mance E. Harmon y Stephanie S. Harmon (s.f.). Reinforcement Learning: A Tutoriál. [documento en línea] Consultado en 20/9/2003 Disponible en http://www.nada.kth.se/kurser/kth/2D1432/ 2003/rltutorial.pdf.

Touzet, Claude F. (1999). Neural Networks and Q-Learning for Robotics. [documento en línea] Consultado en 3/9/2003 Disponible en http:// avalon.epm.ornl.gov/-touz.eto/Publi/ Touzet_IJCNN_Tut.pdf.