UNIVERSIDAD CATÓLICA ANDRÉS BELLO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Sistema autónomo de navegación y búsqueda basado en enfoque multiagente utilizando algoritmos de aprendizaje en tiempo real

Este Jurado; una vez realizado el examen del presente trabajo ha evaluado su contenido con el resultado: 17 ptos.

J U R A D O E X A M I N A D O R

Hilmar Castro	Jorge Serebrinsky	Wilmer Pereira

REALIZADO POR	Claudio Jesús Rojas Astudillo Eliso Ignacio Silva Noriega
PROFESOR GUIA	Dr. Wilmer Pereira
FECHA	Caracas, Julio de 2006

INDICE DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

SINOPSIS

El presente trabajo especial de Grado, consistió en diseñar un sistema multiagente utilizando un algoritmo de aprendizaje en tiempo real. El objetivo de este sistema es que varios agentes autónomos (robots) naveguen por un ambiente con obstáculos, colaboren entre si para que uno de ellos llegue a un punto conocido, tratando en lo posible de realizar el recorrido en el menor numero de pasos, y detectando la menor cantidad de obstáculos.

Los algoritmos de aprendizaje en tiempo real o no supervisados son aquellos que permiten al sistema aprender en el instante que ocurre un suceso, es decir, se adaptan a entornos dinámicos. La mayoría de los algoritmos que existen en el área son para Sistemas Monoagente (un agente explorado el ambiente). Por esta razón se diseñó un modelo de aprendizaje con base Monoagente, adaptado al enfoque multiagente. También se desarrolló una aplicación que permita simular el comportamiento del sistema, pero con una mayor cantidad de robots.

El ambiente seleccionado para diseñar este sistema fue un laberinto. La modalidad de laberinto se adapta a las condiciones del ambiente que se desean lograr, ya que posee obstáculos y un punto conocido (salida). Para el diseño de los robots se utilizó el kit LEGO® MindStorms® *Robotics Invention System*, para luego ser programados en LeJOS, un firmware basado en Java para el manejo de la unidad de procesamiento de estos robots. La simulación de los robots se programó en Microsoft® Visual Basic, con base en lo programado para el modelo físico.

Se utilizó como metodología de desarrollo la conocida como espiral, ya que permite avanzar de manera incremental en el desarrollo del mismo. De acuerdo a esta metodología, se desarrollaron los elementos fundamentales para el funcionamiento de este sistema: navegación de los robots (seguimiento de línea, posicionamiento), comunicación (pase de mensajes, protocolo de comunicación) y aprendizaje (estrategia de aprendizaje, identificación de estados y selección de refuerzos).

CAPÍTULO I
INTRODUCCIÓN

PLANTEAMIENTO DEL PROBLEMA.

Uno de los problemas a resolver en las investigaciones de Inteligencia Artificial es cómo se puede lograr la colaboración entre un conjunto de agentes autónomos para alcanzar una meta propuesta. Existe gran variedad de algoritmos desarrollados en el área, pero se desea ver su desempeño en la navegación por un ambiente desconocido, y localizar un objetivo del que se conoce previamente su ubicación. Se desea diseñar e implantar un sistema que permita a un conjunto de robots autónomos, sin conocimiento previo del ambiente donde se ubiquen, colaborar entre ellos para así localizar un objetivo específico, utilizando para ello métodos de cooperación multiagente y algoritmos de aprendizaje en tiempo real. También se desea estudiar las modificaciones en tiempo de diseño que puedan realizarse a dichos algoritmos, de modo que puedan adaptarse para lograr un modo de orientación para los robots basado en la colaboración entre los mismos.

OBJETIVO GENERAL.

- Desarrollar un programa basado en algoritmos de aprendizaje en tiempo real que implantado en un conjunto de robots autónomos, permita a los mismos comunicarse y colaborar entre sí para orientarse en un ambiente con obstáculos desconocidos y localizar un objetivo ubicado en unas coordenadas específicas previamente conocidas.
- Diseñar una aplicación de software que permita simular el comportamiento de dicho conjunto de robots en un ambiente controlado.

OBJETIVOS ESPECÍFICOS.

- Evaluar diferentes algoritmos de aprendizaje en tiempo real.
- Estudiar los diferentes mecanismos de orientación que permitan a los robots desplazarse por un ambiente plano, con obstáculos desconocidos para el mismo.
- Evaluar diferentes medios de transmisión de datos y protocolos de comunicación que permitan el intercambio de información entre los robots, favoreciendo la colaboración entre los mismos para la localización de un objetivo.
- *Construir un prototipo funcional de un conjunto de robots que dado un* ambiente plano con obstáculos, puedan orientarse en el mismo y localizar un objetivo ubicado en unas coordenadas específicas previamente conocidas.
- Determinar las modificaciones necesarias que deben realizarse a los algoritmos seleccionados para adaptarse a la búsqueda en conjunto, así como examinar su comportamiento.
- Realizar un estudio comparativo de los resultados arrojados en la simulación del algoritmo y la posterior implantación del programa en el prototipo.

JUSTIFICACIÓN.

En un trabajo de investigación que involucra IA (Inteligencia Artificial) existen principalmente dos factores que limitan el desarrollo de algoritmos más eficientes, y por ende su implantación en la industria: coste de los equipos y tolerancia a fallas. Se desea desarrollar un programa que de ser implantado a futuro en un conjunto de robots autónomos, desarrollen acciones que resulten de alto riesgo para los humanos, como realizar acciones de rescate en zonas inaccesibles para un ser humano, como en las profundidades del océano, o en las laderas de un volcán en erupción, etc. También se prevé que su campo de acción no estaría limitado sólo al planeta tierra. En la exploración espacial, se podrían construir robots que puedan explorar planetas y dividir el trabajo de un sólo robot entre un conjunto de robots menos costosos pero con mayor efectividad, y capaces de tomar decisiones por sí mismos basados en las experiencias que les provee el medio ambiente donde se desenvuelven.

LIMITACIONES Y ALCANCES.

LIMITACIONES.

- El estudio abarca sólo las técnicas de aprendizaje en tiempo real como el *aprendizaje por refuerzo, que son las técnicas que abarcan la mayor parte de* la bibliografía referente al tema.
- El número de robots que comprenden el prototipo funcional a implantar consta de 2 a 3 robots, razón por la cual se realizará la simulación con un mayor número de agentes.

ALCANCES.

- Obtener un programa basado en algoritmos de aprendizaje en tiempo real, que permita adaptarse a la colaboración entre un conjunto de robots autónomos para localizar un objetivo conocido por un ambiente con obstáculos desconocido.
- Simular en una computadora el comportamiento del algoritmo en un conjunto de agentes autónomos en un ambiente controlado.
- Obtener una evaluación del comportamiento del programa sobre un prototipo funcional de un conjunto de robots construido para localizar un objetivo conocido en un ambiente con obstáculos desconocidos.

CAPÍTULO II
MARCO TEÓRICO

1) ROBÓTICA

La *robótica*[1] es una ciencia o rama de la tecnología, que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano o que requieren del uso de inteligencia. Las ciencias y tecnologías de las que deriva podrían ser: el álgebra, los autómatas programables, las máquinas de estados, la mecánica o la informática. Es un concepto de dominio público. La mayor parte de la gente tiene una idea de lo que es la robótica, sabe sus aplicaciones y el potencial que tiene; sin embargo, no conocen el origen de la palabra robot, ni tienen idea del origen de las aplicaciones útiles de la robótica como ciencia.

El *Diccionario de inglés de Oxford* le da crédito a Isaac Asimov por introducir la palabra *robótica* en el idioma inglés. Asimov fue un escritor y bioquímico ruso nacionalizado estadounidense, exitoso y excepcionalmente prolífico autor de obras de ciencia ficción y divulgación científica.

Un robot puede ser visto en diferentes niveles de sofisticación, depende de la perspectiva con que se mire. Un técnico en mantenimiento puede ver un robot como una colección de componentes mecánicos y electrónicos; por su parte un ingeniero en sistemas puede pensar que un robot es una colección de subsistemas interrelacionados; un programador en cambio, simplemente lo ve como una máquina que ha de ser programada; por otro lado para un ingeniero de manufactura es una máquina capaz de realizar un tarea específica. En contraste, un científico puede pensar que un robot es un mecanismo el cuál él construye para probar una hipótesis.

[1] El nombre procede del término checo ***robota*** (trabajador, siervo) utilizado por el escritor Karel Capek en su obra *"Los robots universales de Rossum"*, en 1920

La potencia del software en el controlador determina la utilidad y flexibilidad del robot dentro de las limitantes del diseño mecánico y la capacidad de los sensores.

Los robots han sido clasificados de acuerdo a su función, a su nivel de inteligencia, a su nivel de control, y a su nivel de programación. Estas clasificaciones reflejan la potencia del software en el controlador. La generación de un robot se determina por el orden histórico de desarrollos en la robótica. Cinco generaciones son normalmente asignadas a los robots industriales. La tercera generación es utilizada en la industria, la cuarta se desarrolla en los laboratorios de investigación, y la quinta generación es un gran sueño.

El desarrollo de la robótica se está orientando hacia la construcción de máquinas que sepan interactuar con el medio en el cual desarrollan su actividad (reconocimientos de formas, toma de decisiones, etc.). Estos robots reciben entradas del medio que les rodea, a través de sensores conectados al medio ambiente y producen salidas por medio de efectores, logrando así reconocer elementos que estén en el ambiente y tener la capacidad de elegir una acción para conseguir su objetivo. Este tipo de robots reciben el nombre de robots autónomos o *agentes autónomos*.

Siendo los robots un conjunto de componentes mecánicos y electrónicos, su uso a lo largo del tiempo puede producir fallas en su funcionamiento y ser afectado por los siguientes aspectos: lecturas erróneas de los sensores, el desempeño del robot puede verse afectado por factores ambientales (por ejemplo, que las ruedas derrapen, las baterías se agoten o se rompa alguna de las partes del robot) y por la continuidad del mundo real, es decir, los estados y acciones proceden de una serie de configuraciones físicas y de movimientos, lo cual impide numerar todas las acciones posibles y muchas veces será necesario modificar muchos de los algoritmos.

1.1) LEGO® MindStorms® y la Robótica

LEGO® fue una de las empresas pioneras en comercializar a gran escala kits para construir robots, su serie LEGO® MindStorms® alcanzó gran popularidad entre

los aficionados a la robótica con mayor poder adquisitivo y cubrió un nicho de mercado con reducida competencia. LEGO® MindStorms® es una línea que combina bloques programables, motores eléctricos, sensores, bloques Lego, y otras piezas para construir robots y otros sistemas automáticos interactivos. Comercialmente se publicita como Sistema de Invención Robotizado (*Robotics Invention System*, ó RIS). También se vende como herramienta educacional, lo que originalmente se pensó en una sociedad entre Lego y el MIT.

El RIS se vende como un kit completo formado por más de 700 piezas tradicionales de LEGO, dos motores, un sensor de contacto y uno de luz. El cerebro de estos robots es llamado RCX, como se muestra en la **figura 1**, y está constituido por los siguientes componentes: una Unidad Central de Procesamiento (Hitachi H8/300, que trabaja a una velocidad comprendida entre los 10 y 16 MHz), una Memoria ROM (contiene 18 KB de memoria), un Memoria RAM (32 KB de memoria), 3 Puertos de entrada (para sensores de choque, de luz, de temperatura y rotación), 3 Puertos de salida (utilizados para controlar motores), un Puerto de Comunicación Infrarrojo (permite la descarga en el RCX de las aplicaciones diseñadas y la transmisión-recepción de datos desde y hacia el PC) y una Torre Infrarroja (transmite los programas desde la PC hacia el RCX por medio de una señal infrarroja).

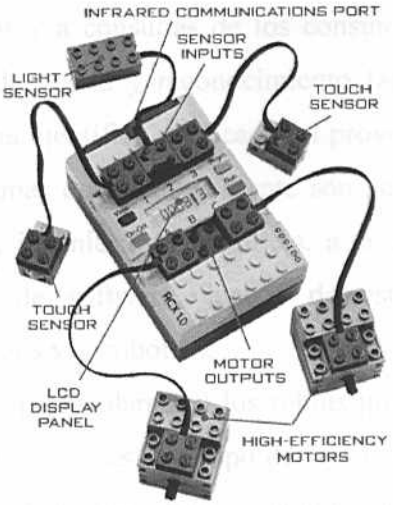


Figura 1. LEGO® MindStorms® RCX
Fuente: http://www.lewiston.k12.id.us/vocprograms/LEGOMindstorms/rcx.gif

2) INTELIGENCIA ARTIFICIAL

La *Inteligencia artificial* o *IA,* es definida como la inteligencia exhibida por una entidad artificial o maquina. Generalmente se asume que dicha entidad, maquina o sistema sea un computador. La *Inteligencia Artificial* estudia la capacidad de una maquina de realizar los mismos tipos de funciones que caracterizan al pensamiento humano. A pesar de que la IA tiene una fuerte connotación en la ciencia ficción, forma una rama vital en las ciencias de la computación, la cual lidia con el comportamiento inteligente, el aprendizaje y la adaptación en las máquinas.

El campo de la inteligencia artificial está pasando rápidamente de los laboratorios universitarios a la aplicación práctica en la industria, y se están desarrollando máquinas capaces de realizar comportamiento inteligente mediante la ejecución de tareas cognitivas como el aprendizaje por experiencia o en tiempo real. Indudablemente, resulta vital en el proceso de emular comportamientos inteligentes, que un sistema pueda mejorar su comportamiento sobre la base de la experiencia que colecte al efectuar la misma tarea en repetidas ocasiones, y que además desarrolle algún tipo de noción de lo que es un error, y cómo evitarlo. Algunos ejemplos se encuentran en el área de control de sistemas y planificación automática, la habilidad

de responder a diagnósticos y a consultas de los consumidores, reconocimiento de escritura, reconocimiento del habla y reconocimiento facial. De este modo, se ha convertido en una disciplina científica, enfocada en proveer soluciones a problemas de la vida diaria. Los sistemas de IA actualmente son parte de la rutina en campos como economía, medicina, ingeniería y la milicia, a si como se ha usado en gran variedad de aplicaciones de software, juegos de estrategia como ajedrez de computador, otros videojuegos y la robótica.

Los cambios más espectaculares en los robots provienen de su capacidad de razonamiento cada vez mayor gracias al campo de la *Inteligencia Artificial*. Diversos estudios en el campo de la AI se concentran en un enfoque en el que un solo agente o robot resuelve un problema en específico. En el mundo de hoy existen problemas cada vez más complejos por su dimensión, originando gran dificultad en la solución de estos por parte de un solo robot. Esta situación es uno de los problemas a resolver en el área de la Inteligencia Artificial, orientándose a un enfoque multiagente o grupo de varios robots que colaboren e interactúen entre sí para resolver determinado problema. Para solucionar estos problemas existen técnicas en el área de aprendizaje en tiempo real, que permiten simular el comportamiento y colaboración entre varios robots. El aprendizaje se puede clasificar en dos categorías generales:

a) **Aprendizaje supervisado:** el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un supervisor o maestro que determina la respuesta que debería generar el sistema a partir de una entrada determinada. Este tipo de aprendizaje, por lo general, es implementado mediante la técnica de Redes Neuronales. Las Redes Neuronales están constituidas por unidades que simulan las neuronas del cerebro humano y necesitan ser entrenadas mediante un conjunto de ejemplos donde el resultado de cada uno de ellos es conocido. Si la red no responde como se espera, se modifican sus pesos y/o estructura con el fin de que la misma pueda realizar exitosamente un porcentaje alto de los casos de entrenamiento.

b) **Aprendizaje en tiempo real o no supervisado:** en este caso no es necesario un supervisor o maestro, el sistema debe organizarse a si mismo y por si solo. El más importante representante de este tipo de aprendizaje es el aprendizaje por reforzamiento (AR). En este tipo de aprendizaje no se dispone de un ejemplo concreto para la salida o comportamiento deseado en específico, esto implica que no se conoce la salida exacta para cada entrada pero si como debería ser el comportamiento de manera general ante diferentes entradas[2]. Esto refleja la relación de entrada-salida a través de un proceso de éxito o fracaso, produciendo una señal (señal de refuerzo) que mide el funcionamiento del sistema.

3) SISTEMAS MULTIAGENTE (MAS) E IA

Los **Sistemas Multiagente** (MAS) en conjunto con la IA forman una ciencia y una técnica que trata con los sistemas de inteligencia artificial en red. La concepción moderna de Inteligencia Artificial está centrada sobre el concepto del agente racional. Un agente es cualquier cosa que percibe su ambiente a través de sensores e interactúa con el mismo a través de efectores (Russell y Norvig, 2003). Un agente que siempre intenta optimizar su rendimiento es llamado un agente racional. Desde esta perspectiva, IA puede ser definida como el estudio de los principios y diseños de los agentes racionales artificiales.

Los agentes son sistemas autónomos. En muchas situaciones, ellos coexisten e interactúan con otros agentes de diferentes maneras. Ejemplos de ello incluyen agentes en Internet, robots jugadores de fútbol (robo-soccer) y muchos otros más. Un sistema que consiste en un grupo de agentes que pueden potencialmente interactuar con otros es llamado un sistema multiagente (MAS, siglas en ingles).

[2] Este comportamiento general o patrón de conducta, es lo que se denomina como política, la cual se encuentra referida en la sección 1 del Marco Referencial.

3.1) CARACTERÍSTICAS DE LOS SISTEMAS MULTIAGENTE

Los aspectos fundamentales que caracterizan a los MAS y las diferencias con los Sistemas Monoagente[3], se puede pensar en las siguientes dimensiones:

a) Diseño del agente. Es común, el caso en que varios agentes que coexistan en un MAS sean diseñados de diferentes maneras. Ejemplo típico de esto son los agentes de software (soft-bots) que pueden ser implementados por diferentes personas. En general, las diferencias de diseño pueden envolver el hardware (por ejemplo los agentes de software que corren en diferentes sistemas operativos). A veces, se puede afirmar que en su mayoría los agentes son heterogéneos en contraste a los agentes homogéneos que son diseñados de una forma idéntica y tienen en común las mismas capacidades. Sin embargo, esta distinción no esta muy clara, los agentes que están basados en el mismo hardware/software, pero que implementan diferentes comportamientos también pueden ser llamados heterogéneos. Esta característica puede afectar todos los aspectos funcionales de un agente, desde su percepción hasta su toma de decisiones, mientras que en los Sistemas Monoagente, esta característica no es influyente.

b) Ambiente. Los agentes tienen que interactuar con ambientes que pueden ser estáticos (invariantes en el tiempo) o dinámicos (no estacionarios). La mayoría de las técnicas de IA existentes, son para Sistemas Monoagente; han sido diseñadas para ambientes estáticos porque son fáciles de manejar y permiten un tratamiento matemático más riguroso. En el MAS la mera presencia de múltiples agentes hace que el ambiente sea dinámico desde el punto de vista de cada agente. Esto puede ser a veces problemático, en el caso de agentes de aprendizaje concurrentes donde no pueden apreciarse comportamientos inestables.

c) Percepción. La información colectiva que recogen los sensores de los agentes en un MAS es típicamente distribuida: los agentes pueden observar datos que difieren espacialmente (aparecen en diferentes sectores), temporalmente (llegan en

[3] Se define como Sistema Monoagente, aquel en donde un solo agente (robot) interactúa con el ambiente

diferentes tiempos), o semánticamente (requieren interpretaciones diferentes). Estos elementos hacen que automáticamente el estado del mundo parcialmente observado por cada agente, afecta consecuentemente la toma de decisiones de los otros agentes.

d) Control. El control esta ligado a la coordinación entre los agentes, lo cual asegura que las decisiones individuales de cada agente resultan las mejores decisiones para el grupo.

e) Conocimiento. En los Sistemas Monoagente se asume típicamente que el agente conoce sus propias acciones, pero necesariamente el mundo es afectado por dichas acciones. En un MAS, los niveles de conocimiento de cada agente sobre el estado actual del mundo pueden diferir sustancialmente. Por ejemplo, en un MAS de equipo que envuelve dos agentes homogéneos, cada agente puede conocer la disponibilidad de un conjunto de acciones del otro agente, ambos agentes pueden conocer (por comunicación) sus propias percepciones, o pueden inferir las intenciones de los otros basados en algún conocimiento compartido. Por otra parte un agente que observa un equipo de agentes adversario, no haría disponible sus acciones posibles, y hasta podría interferir en los planes del adversario. En general, en un MAS cada agente debe considerar el conocimiento del otro agente en su toma de decisiones. Un concepto crucial en este punto es el conocimiento común, de acuerdo al conocimiento de cada agente, cada agente tiene conocimiento del conocimiento de los otros agentes.

f) Comunicación. La interacción es a veces asociada con algún tipo de comunicación. Típicamente vemos la comunicación en un MAS como un proceso de dos vías, donde todos los agentes pueden potencialmente enviar y recibir mensajes. La comunicación explora que protocolos de comunicación son usados para intercambiar información que sea transmitida de forma rápida y segura, y el lenguaje que deben hablar los agentes para poder entenderse (especialmente si son heterogéneos).

3.2) DESAFÍOS PARA LOS SISTEMAS MULTIAGENTE

La transición de Sistemas Monoagente a Multiagente tiene mucho potencial de desarrollo, pero enfrenta grandes desafíos. Algunos de ellos son:

- Como descomponer un problema, localizar las sub-tareas de los agentes, y sintetizar los resultados parciales.
- Como manejar la información de la percepción distribuida.
- Como permitir a los agentes mantener modelos compartidos del mundo que sean consistentes.
- Como diseñar algoritmos eficientes de planeamiento y aprendizaje multiagente.
- Como representar el conocimiento y permitir a los agentes razonar acerca de sus acciones, planes, y el conocimiento de los otros agentes.
- Como permitir la comunicación entre los agentes. Que lenguajes de comunicación y protocolos usar. Que, cuando y con quien los agentes deberían comunicarse.
- Como permitir a los agentes negociar y resolver conflictos.
- Como asegurar el comportamiento coherente y estable del sistema.

CAPÍTULO III
MARCO REFERENCIAL

1) APRENDIZAJE POR REFUERZO

El Aprendizaje por Refuerzo (AR) es una técnica de aprendizaje interesante, que ha tenido éxito en situaciones donde otras técnicas tales como aprendizaje supervisado no eran aplicables (por ejemplo Redes Neuronales). Sus aplicaciones han alcanzado áreas de considerable complejidad tales como robótica, manufacturación industrial.

El aprendizaje por refuerzo trata de resolver el problema de cómo un agente autónomo que recibe información sensorial y actúa en un entorno puede aprender a elegir acciones óptimas para alcanzar sus objetivos. Se trata, por tanto, de un sistema de aprendizaje supervisado que recibe información del entorno externo y de un "profesor" que produce refuerzos positivos o castigos según la calidad de los resultados a los que llega el agente. A diferencia del aprendizaje supervisado, el agente no posee experiencia previa acerca del ambiente con el que interactúa. Existen tres aspectos fundamentales a tomar en cuenta en el AR, ellos son: **el ambiente, la función de refuerzo, la política y la función valor.**

- **El ambiente**: Es todo aquello que no es controlado por el agente, genera estados y recibe acciones. Para evaluar el resultado de las acciones del agente en el ambiente, éste pasa al agente información acerca de su estado actual y del resultado de la ejecución de la acción. Si solo se cuenta con información parcial del estado actual del ambiente, se dice que se tiene una observación. Estas observaciones llegan al agente en forma de: lecturas de sensores, descripciones simbólicas y situaciones mentales.

- **La función de refuerzo (r):** es aquella que define la meta del agente en un problema de AR, proporcionando las acciones positivas que el agente busca maximizar. Asocia a cada par (estado, acción) un número escalar denominado recompensa o castigo de forma inmediata, e indica qué tan bueno fue el resultado de la acción con respecto a la meta global.
- **La función valor o de evaluación (V):** su objetivo es derivar una asociación de estados con acciones; esta asociación relaciona cada estado con la mejor acción que se debe tomar. Permite determinar una política.
- **La política (π):** es aquella que selecciona las acciones que deben ser ejecutadas para los estados percibidos del ambiente. Es el núcleo del aprendizaje ya que delimita la conducta del agente. Entre las políticas propuestas se encuentran:

  a) Random: elige de forma aleatoria la acción.

  b) Greedy: consiste en elegir siempre la acción cuyo valor estimado corriente[4] es el mayor.

El objetivo del aprendizaje por refuerzo es encontrar una política, π^*, que para cada estado en el que se encuentre el agente, indique cuál es la mejor acción que se puede ejecutar para alcanzar un objetivo dado; o traducido al aprendizaje por refuerzo, maximice alguna medida de refuerzo a largo plazo. Esto es lo que se conoce como *política óptima*. Es importante aclarar que el agente puede encontrar políticas que no sean óptimas, pero que generen un patrón de conducta convergente en el tiempo para el agente.

Existen diversos métodos para aplicar el AR, entre los métodos utilizados se encuentran: Q-Learning, Dyna-Q y Prioritized Sweeping. El más popular y utilizado es Q-Learning, debido a su sencillez a la hora de codificarlo y a la gran cantidad de información que existe acerca de él. En la **Tabla 1** se puede observar una comparación entre los siguientes métodos.

[4] Los valores estimados son asignados a cada una de las acciones posibles en todos los estados, y un valor estimado se actualiza cada vez que se ejecuta la acción correspondiente a dicha acción.

	Ventajas	Desventajas
Q-Learning	a) Garantiza un resultado óptimo. b) Baja dificultad de codificación. c) Alto porcentaje de exploración del ambiente, debido a su convergencia lenta.	a) Convergencia lenta hacia la política óptima.
Dyna-Q	a) Garantiza un resultado óptimo. b) Convergencia rápida hacia la política óptima.	a) Alta dificultad de codificación. b) Bajo porcentaje de exploración del ambiente, debido a su convergencia rápida.
Prioritized Sleeping	a) Garantiza un resultado óptimo. b) Convergencia rápida hacia la política óptima.	a) Alta dificultad de codificación. b) Bajo porcentaje de exploración del ambiente, debido a su convergencia rápida.

Tabla 1. Cuadro comparativo Métodos AR.
Fuente: Elaboración Propia.

1.1) ALGORITMO Q-LEARNING

El Algoritmo Q-Learning es uno de los métodos más sencillos y utilizados en la práctica. Este algoritmo establece una relación de pares estados-acciones a valores, llamados Q-valores. Generalmente son representados como una matriz, con las filas como los estados y las columnas como posibles acciones en los estados, como se muestra en la Figura 2.

	Acción 1	Acción 2	.....	Acción n
Estado 1	0.5540642182507065	0.9065059349059392	.....	0.2580756335009564
Estado 2	0.2567741597100365	0.8256158956417472	.....	0.4703028571322908
. . .	. . .	. . .	.....	. . .
Estado n	0.14712125790226305	0.31524575392194554	.....	0.2184643051634042

Figura 2. Ejemplo de Matriz de Q-Valores.
Fuente: Elaboración Propia.

En cada estado, hay un Q-valor asociado con cada acción. Inicialmente, estos Q-valores se pueden inicializar en cero o en valores aleatorios (ver **Figura 2**). Los Q-valores son los que producen la toma de decisiones, de acuerdo de la política seleccionada.

La relación estados-acciones es posible por medio de la función valor o de evaluación de este algoritmo, llamada la función Q. Por medio de esta función, obtenemos la acción a ejecutar. Cuando la acción es ejecutada, se suman o acumulan los refuerzos y castigos que actualizan los Q-valores. Un Q-valor óptimo es la suma de los refuerzos recibidos cuando desempeñe la acción y siguiendo entonces la política óptima. La actualización de los Q-Valores, es posible a la función de actualización o acumuladora.

La función de actualización básica en el algoritmo de Q-Learning es la siguiente:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma max\ Q(s',a') - Q(s,a))$$

En este caso, ***Q(s,a)*** representa el Q-valor anterior para la acción anterior *"a"* en el estado anterior *"s"*; ***Q(s',a')*** representa el Q-valor actual para la acción actual *"a"* en el estado actual *"s"* ,*"*α*"* representa la tasa de aprendizaje (α>0), *"*γ*"* representa el factor de descuento (0<=γ<=1), y *"r"* simboliza los castigos o recompensas devueltos por el par (situación, acción) al aplicarse la función Q (*"r"* puede tener cualquier valor).

1.2) PASOS BÁSICOS EN EL ALGORITMO Q-LEARNING.

De acuerdo a análisis recientes, se definen los siguientes pasos básicos en el algoritmo Q-Learning:

1. Inicialización de la memoria del robot: para todos los pares de estado *"s"* y acción *"a"*, se inicializa el valor Q asociado en un valor aleatorio o cero.
2. Observar estado actual *s*.
3. Repetir:
   3.1) La función de evaluación selecciona la acción para ejecutarse. El proceso de selección puede variar de acuerdo a la política π implementada.

3.2) El robot ejecuta la acción *a* en el mundo. Recibe la recompensa inmediata *r*. Sea *r* la recompensa (*r* puede ser 0) asociada con la ejecución de *a* en el mundo.

3.3) Observar el nuevo estado *s'*.

3.4) Actualiza la memoria del robot, por medio de la función actualizadora :

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma max\, Q(s',a') - Q(s,a))$$

Donde *s'* es el nuevo estado tras haber sido ejecutada la acción *a* en el estado *s*, *a'* representa cualquier acción posible y $\alpha>0$, $0<=\gamma<=1$.

3.5) Asignar *s'* a *s*.

Es importante aclarar que los valores asignados a α y a γ determinan como será el aprendizaje del agente y de que forma podrá maximizar sus recompensas hasta adquirir la política óptima.

CAPÍTULO IV
MARCO METODOLÓGICO

Para Pressman (1998), "Una metodología es un conjunto de pasos y procedimientos que deben seguirse para la creación y desarrollo de un proyecto, software o sistema determinado. Es una versión amplia y detallada de un ciclo de vida completo en el desarrollo de sistemas. Podemos decir que el ciclo de vida define las fases y tareas esenciales para el desarrollo de sistemas, sin importar el tipo o la envergadura del sistema que se intenta construir".

Además, Pressman (1998), explica que una metodología puede ser puntualizada de la siguiente manera:

- Tareas paso a paso para cada fase.
- Funciones individuales y en grupo desempeñadas en cada tarea.
- Productos resultantes y normas de calidad para cada tarea.
- Técnicas de desarrollo que se utilizan en cada tarea.

1) METODOLOGIA UTILIZADA

Para la realización de este trabajo de investigación, se debe disponer de una metodología que permita cubrir los objetivos propuestos de una forma eficiente, y poder adaptar la investigación a los lapsos propuestos por un cronograma de trabajo. Por esta razón se escoge como metodología el modelo de ***espiral***.

Es un modelo de proceso evolutivo que acompaña la naturaleza interactiva de construcción de prototipos con el proceso sistemático y lineal del desarrollo secuencial. El software para el robot se desarrolla en una serie de versiones incrementales, las cuales deben ser probadas en simulación para adaptarse luego al

prototipo. Si se observa fallas en su ejecución, se debe volver a una fase anterior para determinar los errores y hallar nuevas alternativas.

El modelo, se compone de las siguientes tareas:

- **Levantamiento de información:** son las tareas requeridas para establecer la comunicación entre el desarrollador y el cliente.
- **Planificación:** se definen recursos, el tiempo y otras informaciones relacionadas con el proyecto.
- **Análisis de riesgos:** son las tareas requeridas para la evaluación de riesgos técnicos y de gestión.
- **Ingeniería:** tareas requeridas para construir una o más representaciones de la aplicación.
- **Construcción y adaptación:** las tareas requeridas para construir, probar, instalar y proporcionar soporte para el usuario.
- **Evaluación:** tareas para obtener la reacción del cliente según la evaluación de las representaciones del software creadas durante las etapas anteriores.

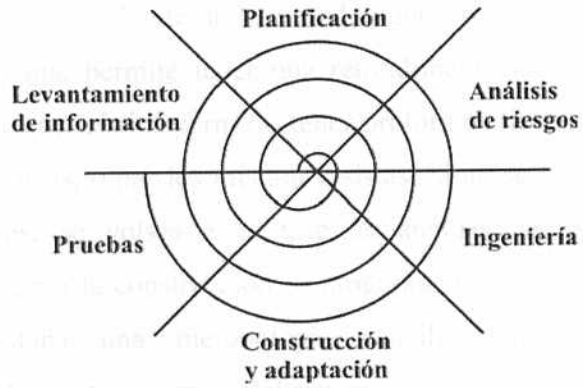


Figura 3. Fases del Modelo en espiral.
Fuente: Ingeniería del Software. Pressman, 1999.

Como lo muestra el gráfico anterior, se sigue en cada espira un modelo secuencial que recorre las fases antes mencionadas, de manera que si en las pruebas se detecta alguna anomalía, se puede regresar a levantamiento de información que permita determinar las causas de dicha anomalía y poder solventarla. También

permite desarrollar la investigación en pequeños avances, que permiten corregir errores antes que el proyecto culmine, reduciendo los riesgos al final de la investigación.

Esta metodología podría no seguirse completamente al pie de la letra, ya que se debe cumplir con un cronograma que finalizara con la presentación del trabajo de investigación.

2) JUSTIFICACION DEL USO DE LA METODOLOGIA UTILIZADA

La metodología en espiral es comúnmente utilizada para proyectos científicos de investigación, en los que existe cierto grado de incertidumbre acerca de los resultados definitivos o los productos que se puedan obtener. Es por esto que se necesita contar con una herramienta que permita mantener un ciclo constante, en el que cada elemento que se desarrolle pase por las pruebas que se requieren para poder pasar a la siguiente fase.

Una característica resaltante de la metodología espiral, es que en cada fase se obtiene un producto, que permite tener una retroalimentación antes de obtener el producto final. Esta característica permitió tener prototipos listos para ser evaluados, ya sea por el tutor de tesis, o por los mismos tesistas. Si no se alcanzaban los niveles de calidad requeridos, se volvía a la fase de análisis, si la prueba resultaba satisfactoria se procedía a la construcción (o programación) de módulos adicionales. También se necesitaba una metodología sencilla que permitiese trabajar simultáneamente en diferentes aspectos del sistema, los cuales se aportan información entre si. Ya se han obtenido éxitos en desarrollo de proyectos de este tipo con esta metodología (Dos Santos, D. y Peñalver, R, 2003)

CAPÍTULO V
DESARROLLO

De acuerdo a lo señalado en la metodología para la realización de este trabajo de investigación, la fase de desarrollo se divide básicamente en cinco secciones, cada una con sus actividades relacionadas. Estas se presentan en el siguiente resumen:

1) Espiral 1: Planificación y diseño del ambiente y los robots.

1.1) Planificación y diseño del ambiente.

1.2) Planificación y diseño de los robots.

1.3) Selección de las herramientas de codificación

2) Espiral 2: Manejo del sistema de navegación de los robots.

2.1) Manejo y control de los sensores.

2.2) Diseño de la estrategia de navegación del robot de acuerdo al ambiente construido.

3) Espiral 3: Manejo de la comunicación en el sistema.

3.1) Establecimiento de los elementos de comunicación en el sistema.

3.2) Selección y diseño del protocolo de comunicación.

4) Espiral 4: Manejo del aprendizaje en el sistema.

4.1) Selección, planificación y diseño del modelo de aprendizaje más adecuado para el sistema.

4.2) Selección de los estados.

4.3) Política de selección de las acciones.

4.4) Manejo de la función refuerzo.

4.5) Desarrollo del algoritmo de aprendizaje.

5) Espiral 5: Montaje del diseño definitivo del sistema.

5.1) Coordinación de los elementos de navegación, comunicación y aprendizaje.

5.2) Pruebas en conjunto del sistema.

Aun cuando las fases cumplen con un aspecto definido del desarrollo, todas están relacionadas con el objetivo común de este trabajo de investigación, y por tanto, se observará las contribuciones de una fase a la otra.

Es importante señalar que, tanto el diseño del sistema físico como la simulación se llevaron a cabo en paralelo, debido a la interdependencia existente entre estas dos actividades: por ejemplo, las pruebas obtenidas en la simulación (diseñada a partir del sistema físico), sirven para detectar errores en el sistema físico y corregirlos. Por esta razón, la gran mayoría de las actividades llevadas a cabo en cada una de las fases de desarrollo son compartidas entre el sistema físico y la simulación.

1) Espiral 1: Planificación y diseño del ambiente y los robots

1.1) Planificación y diseño del ambiente

Para llegar al diseño ideal del ambiente del sistema físico, se debe determinar lo que se quiere lograr con el mismo; en el caso de este trabajo de investigación, lo que se desea lograr es que un conjunto de robots autónomos puedan recorrer un ambiente con obstáculos desconocidos y localizar un objetivo ubicado en unas coordenadas específicas previamente conocidas.

De acuerdo a trabajos de investigación relacionados (Errecalde, 1999; Vlassis, 2003) el modelo mas común de ambiente para pruebas con robots es lo que se denomina *mundo de malla* o *Grid World,* en el cual el ambiente se divide en secciones formando una cuadricula. Dichas secciones son perfectamente identificables por los sensores del robot y facilitan la navegación por el ambiente.

Del concepto de *Grid World* se decidió diseñar un ambiente base sobre un plano horizontal, en forma de cuadricula. Cada línea entre intersecciones representaría un camino; y cada intersección representaría una coordenada del plano (ver **figura 4**), los cuales serian perfectamente identificables por los robots, ya que los robots seguirían los caminos de plano, y tendrían un mecanismo para distinguir entre coordenadas. Se planteo inicialmente, tanto para el sistema físico como para la

simulación que dicho ambiente contenga 5 rectas horizontales y 5 verticales, lo que da un total de 25 intersecciones y 40 caminos (20 horizontales y 20 verticales).

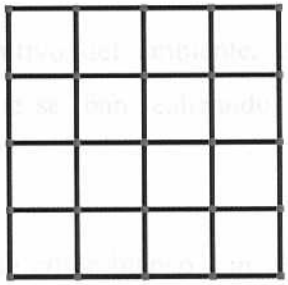
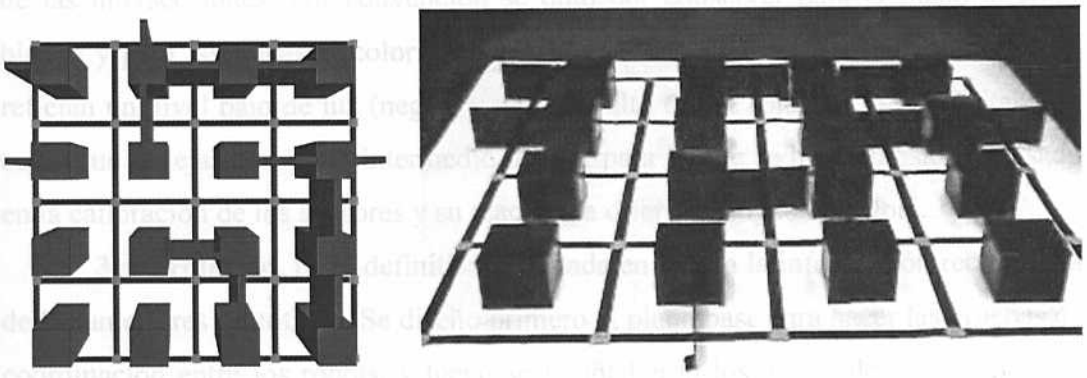


Figura 4. Modelo de Malla propuesto para el ambiente.
Fuente: Elaboración Propia.

Para la parte de los obstáculos, se decidió que estos se encontraran en una disposición tal que formaran un laberinto. Se planeo colocar láminas ortogonales al plano en forma de paredes colocadas aleatoriamente; esto haría que en ciertos caminos un robot no pueda ir de una intersección a otra, pero permite que pueda seguir la línea. Estos obstáculos se encontrarían empotrados en cubos colocados en la parte interna de cada cuadro, tal como lo muestra la **figura 5**.



(a) (b)

Figura 5. Modelo de obstáculos en forma de laberinto propuesto para el ambiente.
Imágenes del laberinto planificado (a) y desarrollado (b).
Fuente: Elaboración Propia.

Se definió como objetivo (o salida del laberinto) una intersección cualquiera de la cuadrícula con una ubicación de coordenadas especificas, al cual los robots deben llegar.

Antes del diseño definitivo del ambiente, éste pasó por diferentes fases, dependiendo de las pruebas que se iban realizando. Estos modelos o prototipos se listan a continuación:

1er Prototipo. Plano de color blanco con una línea de color negro. Dicha pista solo se utilizó para las pruebas del seguimiento de la línea.

2do Prototipo. Pista cerrada que consistía en un cuadrado, y los cuatro extremos del mismo (rellenos de diferente color) representaban las intersecciones que serian las coordenadas del modelo final. El objetivo de esta pista era determinar en que forma el robot haría los giros entre intersecciones.

Un particular detalle para diseñar y probar este prototipo era determinar el color que mejor se adaptara para las intersecciones, de forma que el sensor de luz pudiese diferenciar efectivamente entre los tres colores: el del plano, de las líneas y de las intersecciones. Por convención se opto por conservar para el plano el color blanco y para las líneas el color negro, ya que al ser tonos complementarios ambos reflejan un nivel bajo de luz (negro) y un nivel alto de luz (blanco). Se necesitaba un color que reflejase un nivel intermedio de luz, para lo que se hizo un estudio basado en la calibración de los sensores y su reacción a diferentes tipos de color[5].

3er Prototipo. Pista definitiva, diseñada en base a la información recolectada de los anteriores prototipos. Se diseño primero el plano base para hacer las pruebas de coordinación entre los robots, y luego se le añadieron los obstáculos para hacer las pruebas definitivas.

En cada una de las fases de construcción del ambiente, surgieron cuestiones referentes a la interacción del robot con el ambiente, que influyeron tanto en la

[5] En la sección de navegación se mostraran los detalles para seleccionar los colores definitivos de las intersecciones.

modificación tanto del ambiente como del robot. Estas cuestiones fueron resueltas en cada una de las fases posteriores de desarrollo.

1.2) Planificación y diseño de los robots

Antes de comenzar a diseñar el robot se analizó que clase de robot se necesitaba para el ejercicio a realizar. Se necesitaba un robot que pudiese recorrer un plano, pero también se necesitaba que pudiese detectar obstáculos. Este modelo debería ser capaz de reproducirse de la misma forma para los demás agentes que conformarían el sistema.

En la evaluación preliminar, con la colaboración del Dr. Wilmer Pereira, de la Escuela de Ingeniería de la Universidad Católica Andrés Bello, se determino que para construir el robot, la solución más conveniente era utilizar el kit de LEGO® MindStorms® *Robotics Invention System* 2.0, por varias razones:

- Ya se contaba con experiencia en su uso.
- Es muy sencillo para construir prototipos en poco tiempo.
- No requiere artefactos adicionales para su funcionamiento.
- Se cuenta con una gran documentación para su diseño y construcción.

Para el modelo definitivo del sistema se disponía de 2 kit de LEGO® MindStorms® propios, los cuales permitieron obtener el feedback para el sistema físico, sin embargo, las pruebas de carga se realizarían a partir de la simulación, ya que se podría trabajar con un número mayor de robots.

Al igual que el ambiente, el robot pasó por una serie de modelos o prototipos de acuerdo a las necesidades de cada fase.

1er Prototipo. El prototipo inicial para el desarrollo del robot se tomó del modelo *roverbot*, que se encuentra en la *Constructopedia*, manual de diseño y construcción que viene incluido en el kit de LEGO® MindStorms® *Robotics Invention System* 2.0, realizándole pequeñas modificaciones en la posición del sensor de luz y el sensor de contacto, colocando ambos en la parte delantera del robot. Se

construyó este prototipo inicial como base para tener una visión concreta de lo que se quería lograr con el modelo final.

2do Prototipo. En la siguiente etapa se determinaron los requerimientos del robot para interactuar con el ambiente. Primero, la interacción con el ambiente requería que el robot tuviese un sensor de luz que pudiese seguir una línea recta, detenerse cuando encuentre una intersección y proseguir su camino en esa dirección o girar en otra dirección.

La interacción con los obstáculos podría optimizarse colocando un sensor de luz horizontal en lugar del sensor de contacto, que permitiese detectar los obstáculos antes de chocar con ellos. Sin embargo, se determinó en las pruebas preliminares con el ambiente inicial que se necesitaba un segundo sensor para corregir el rumbo del robot si éste se desviaba del camino[6], lo que presentaba un problema respecto al numero de sensores de luz que se utilizarían. Actualmente sólo se contaba con un sensor que venia incluido en el kit de LEGO® MindStorms®, y sólo se dispondría de un sensor de luz adicional por robot, los cuales serian proveídos por el *Grupo de Inteligencia Artificial de La Universidad Católica Andrés Bello (GIIAR)*. Por esta razón, se decidió que la detección de obstáculos se realizara mediante un sensor de contacto colocado en la parte frontal del robot, tal como el modelo inicial.

Se determinó que el ancho del robot era muy grande como para recorrer un ambiente con las dimensiones planeadas, y podría crear un obstáculo con los otros robots. Por esto se descartó el modelo de oruga del *roverbot* y se sustituyeron dichas orugas por ruedas, que disminuyeron considerablemente el ancho del robot. También se modificó el modelo para que tanto los sensores de luz como el de contacto quedasen en la parte delantera del robot. La configuración de los motores se dejó de la misma forma que en el modelo inicial, dejando que el engranaje principal de ambos motores quedara unido a las dos pares de ruedas laterales, permitiendo el movimiento independiente de las dos ruedas laterales.

[6] Este punto se tocara en su profundidad en la parte de navegación.

Consultando la bibliografía respectiva (Ferrari, M., Ferrari y G. Hempel, R., 2002) se determino que la luz ambiental altera notablemente las lecturas de los sensores de luz. Debido a esto, se colocaron los sensores a una distancia muy cercana del piso. Adicionalmente se implementó un truco sugerido en el libro *"Building Robots with LEGO Mindstorms"* (Ferrari, 2002), y se modificó la entrada de luz de los LED's colocándoles una pequeña pieza de Lego (1 x 2 one-hole brick) justo enfrente al sensor de Luz (ver **Figura 6**). Esta pieza concentró notablemente la cantidad de luz emitida por el LED rojo, y a su vez permitía que el fotorreceptor pudiese tener una lectura mas precisa.

3er Prototipo. Para las posteriores pruebas el modelo volvió a sufrir modificaciones, esta vez por el posicionamiento del sensor IR del RCX, forzando a colocar el RCX de forma vertical[7], lo que influyó en un cambio radical de toda la forma del robot, tanto de los motores como los sensores y el chasis en general. Los motores, que originalmente se encontraban en la parte inferior del cuerpo del robot se colocaron adelante, así como los sensores de contacto y de luz (el sensor de contacto colocado en la parte superior, y los sensores de luz colocados en la parte inferior).

Este modelo también cambió completamente la configuración de los engranajes de las ruedas, ya que para ahorrar espacio y hacer que el modelo fuese mucho más pequeño, se colocó una sola rueda unida a cada engranaje de motor, y en la parte posterior se complementó con una tercera rueda, denominada *rueda loca*, debido a que su movimiento no es fijo, pero sirve de gran apoyo para la estabilidad del robot, tanto para el seguimiento de la línea como para los giros. El modelo definitivo adoptado se aprecia en la **figura 6**.

[7] Este punto se tocara en profundidad en la sección de comunicación.

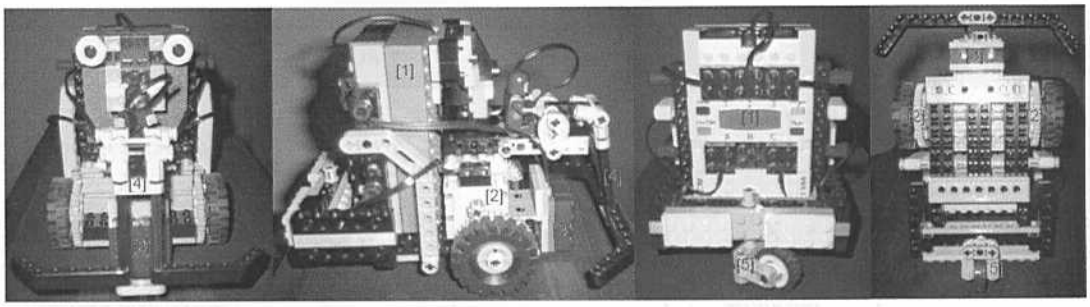


Figura 6. Robot definitivo (Prototipo 3).
En las diferentes vistas se pueden apreciar los elementos del robot: [1] RCX; [2] Motores y ruedas; [3] Sensores de Luz; [4] Sensor de Contacto; [5] Rueda trasera o "Rueda Loca".
Fuente: Elaboración Propia.

Este modelo se repetiría en los siguientes robots que se construirían (2 robots, propiedad de los autores, y un robot que seria prestado por el GIIAR).

1.3) Selección de las herramientas de codificación

Un punto importante en la fase preliminar consistió en determinar la mejor herramienta para codificar el sistema, tanto la aplicación del modelo físico como la simulación.

Luego de construir el prototipo inicial del robot, se comenzó a trabajar con la herramienta de programación para el RCX. La herramienta escogida fue LejOS. La razón de esta escogencia se basa en las siguientes razones:

- Contiene librerías para el manejo de los elementos del RCX: botones, pantalla, motores y sensores.
- Posee librerías para el manejo de la comunicación de la torre con el robot (como se apreciará en la siguiente fase)
- Posee funciones matemáticas, manejo de threads (hilos), estructuras dinámicas de datos, etc. Basados en el lenguaje de programación Java.

Esta herramienta de programación ofrecía ventajas respecto a otras herramientas de programación para el RCX, como se podrá observar en la **figura 7**.

	LejOS	BrickOS (legOS)	NQC	Robotic Invention System (RIS)
Lenguaje de programación	Java	C (BrickOS incluye soporte para C++)	Lenguaje con sintaxis similar a C	Lenguaje gráfico basado en iconos
Sistemas operativos soportados	Windows, Linux, Unix, Solaris, Mac OS X	Windows, Linux	Windows, Mac OS X	Windows
Comunicación vía puerto infrarrojo	Sí, soporta LNP	Sí, soporta LNP	Sí, posee su propio protocolo	N/A
Facilidad de programación	Fácil si se domina Java (también lo es si se domina C++)	Fácil si se domina C/C++	Fácil para todo usuario, pero se facilita aún más si conoce C	Fácil para todo tipo de usuario
Documentación	Sí, API completa en el sitio web	Sí, API completa en el sitio web	Sí, API completa en el sitio web	Aparte de la ayuda en RIS, no hay mucha documentación
Facilidad de instalación	Fácil	Difícil, (sobretodo si el sistema operativo es Windows)	Fácil	Fácil
Funciones matemáticas	Sí	Sí	No	No
Capacidad para programas multitarea (Threads)	Sí	Sí	No	No
Estructuras de datos	Sí	Sí	No	N/A

Figura 7. Comparación de Herramientas de programación para el RCX.
Fuente: Dos Santos, D. y Peñalver, R. (2003).

Para el diseño de la simulación, se debía recurrir a un lenguaje de programación fácil de usar y rápido en el diseño de interfaces graficas, por lo que se determinó codificar la simulación en Microsoft® Visual Basic 6, debido a que es un lenguaje fácil de usar, el entorno de trabajo es agradable y la depuración de los programas se hace de manera rápida y sencilla; permite construir interfaces gráficas rápidamente, y tiene todas las funciones matemáticas y estructuras de datos necesarias para la programación de los algoritmos.

Inicialmente sólo se desarrolló la parte del ensamblaje del ambiente, el cual consiste en una interfaz gráfica que permite editar la ubicación de los obstáculos y los

robots, generando un archivo de texto (*.txt) que sirve como información para el programa que genera el movimiento de los robots, tal como se muestra en la figura 8.

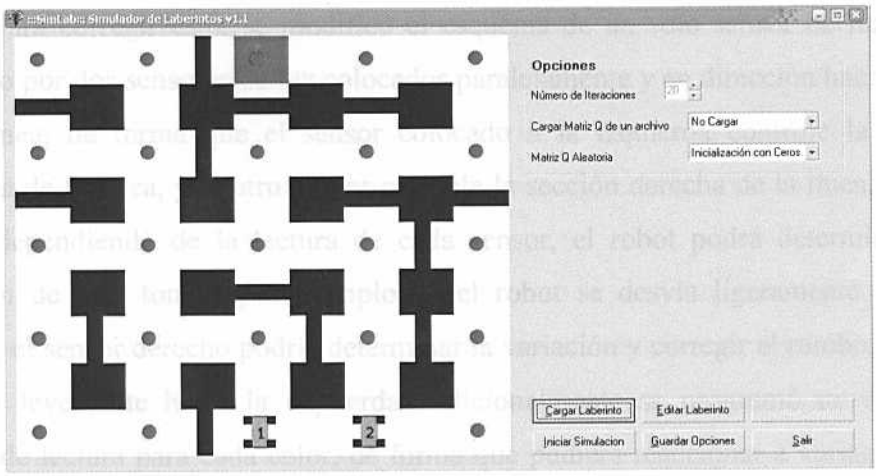


Figura 8. Simulador y editor de laberinto.
Fuente: Elaboración Propia.

2) Espiral 2: Manejo del sistema de navegación de los robots

2.1) Manejo y control de los sensores

En esta fase se realizaron las primeras pruebas de navegación, en las cuales se utilizaron los primeros prototipos del ambiente y del robot. Estas primeras pruebas consistieron en el simple seguimiento de una línea recta, para determinar la exactitud del movimiento del robot. Para ello se desarrolló el 1° prototipo de código en java para el RCX cuya función consistía en que el robot pudiese seguir una línea de color negro sobre fondo blanco. Se desarrolló una clase base `Robot`, con las funciones básicas para seguir la línea (en el **Apéndice A** se muestra el diagrama de clases para el robot).

Para la prueba realizada se observó en la mayoría de los casos que el robot no podía seguir la línea desde su extremo inicial hasta su extremo final. Esto fue debido a que la lectura del sensor para un determinado color difería de los valores fijados en el código, tanto en la parte interna de la línea como en los bordes de la línea. Adicionalmente, el robot debía tener un mecanismo de poder regresar a la línea si se

desviaba de la misma, y que pudiese determinar la dirección a la cual se estaba desviando, para determinar la dirección de la corrección.

Para corregir esto, se modificó el esquema de un solo sensor de luz del 1° Prototipo por dos sensores de luz colocados paralelamente y en dirección hacia dentro de la línea, de forma que el sensor colocado a la izquierda controle la sección izquierda de la línea, y el otro sensor controle la sección derecha de la línea. De esta forma, dependiendo de la lectura de cada sensor, el robot podrá determinar cual dirección de giro tomar: por ejemplo, si el robot se desvía ligeramente hacia la derecha, el sensor derecho podría determinar la variación y corregir el rumbo, girando el robot levemente hacia la izquierda. Adicionalmente se programó un rango de valores de lectura para cada color, de forma que pudiera reaccionar a variaciones de la degradación de color en los bordes de la línea.

En la siguiente actividad, se procedió a realizar las pruebas de giro en las intersecciones. Comenzando con estas pruebas se programó para el RCX una rutina que obtuviese el rango de intensidad de luz que recibían los sensores para un determinado color. Se realizaron varias pruebas con determinados colores. Tanto el gris como el verde ofrecían un rango de luz que no se solapaba con los colores negro y blanco, tomando al final como color para las intersecciones el color verde.

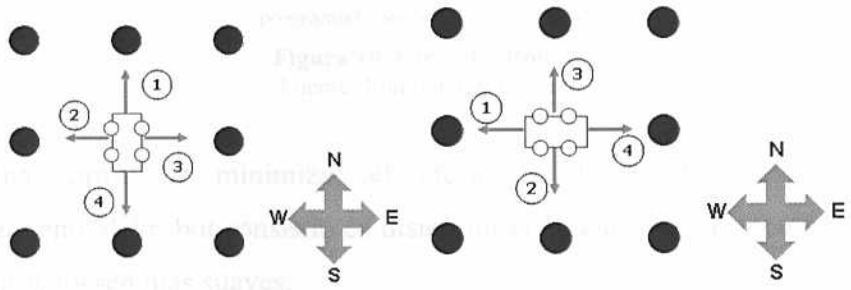
Para realizar el giro en las intersecciones, ya que no se disponía de un sensor de rotación, se programó una rutina basada en el seguimiento de la línea, el cual se compone de las siguientes actividades:

- Si el giro es hacia la derecha o la izquierda, el robot realiza un giro aproximado al ángulo deseado (90°), y luego corrige el movimiento hacia la línea.
- Si el giro es hacia atrás, el robot retrocede una pequeña distancia de la intersección, realiza un giro aproximado al ángulo deseado (180°), y luego corrige el movimiento hacia la línea. En este caso, el giro siempre es en una sola dirección.

2.2) Diseño de la estrategia de navegación del robot de acuerdo al ambiente construido.

La siguiente actividad de esta fase consistió en permitir al robot que este pudiese determinar la ubicación del mapa en la que se encuentra. Para ello se programó la rutina ***tomar_direccion()***, que a partir de unas coordenadas iniciales ya conocidas, realice un recorrido aleatorio del mapa y calcule sus coordenadas de posición a medida que se vaya desplazando.

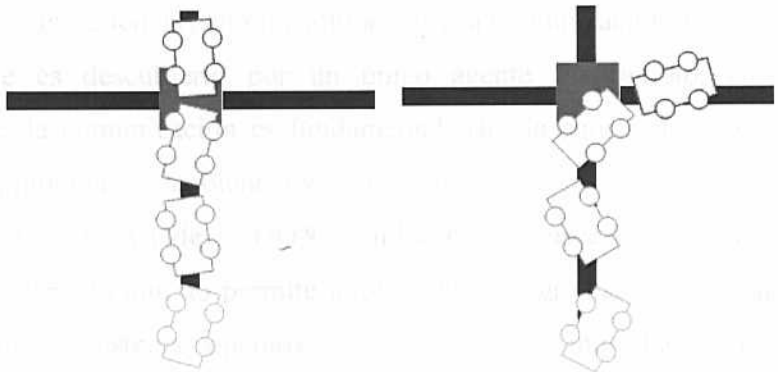
Para la construcción de esta rutina se utilizaron dos variables de orientación para el robot (***direccion*** y ***ruta_def***). La primera variable representa la posible dirección que va a tomar el robot luego de llegar a una intersección {ARRIBA, IZQUIERDA, DERECHA, ABAJO}; la segunda variable representa la posición absoluta del robot respecto del mapa del ambiente {NORTE, SUR, ESTE, OESTE. Ambos conjuntos tienen los mismos valores {0, 1, 2, 3}. La dirección que toma el robot modifica el valor de la posición absoluta del robot, sin embargo, esta posición absoluta no modifica los valores de la dirección, ya que son relativas al robot. Por ejemplo, un robot que inicia el recorrido con una posición absoluta NORTE, al girar a la derecha su posición absoluta respecto al mapa cambia a ESTE, pero si en la siguiente intersección gira a la derecha, lo hará de la misma forma que con posición absoluta NORTE. En la **figura 9** se muestra como se orienta el robot con el mapa del ambiente.



Las direcciones que puede tomar el robot son: (1) Arriba, (2) Izquierda, (3) Derecha, (4) Abajo. Observe que, aunque para el robot la posición absoluta haya cambiado, las direcciones en (a) son iguales que en (b) porque son relativas al robot.

Figura 9. Posicionamiento del robot.
Fuente: Elaboración Propia.

Un problema presentado en esta prueba fue la coordinación del robot en los giros. El robot sigue su trayectoria en línea recta hasta llegar a la intersección, realizando constantemente correcciones en su trayectoria; pero en determinados casos el robot corregía su trayectoria con giros demasiado bruscos. Esta acción daba como resultado un error de posicionamiento si el robot continuaba haciendo estos giros al llegar a la intersección (si el robot llegaba a la intersección un poco desviado a la derecha, a pesar que la dirección computada fuese hacia adelante, realizaría un giro hacia la derecha, "creyendo" que la dirección tomada fue hacia adelante). En la **figura 10**, se observa como es la dirección tomada sin giros bruscos (a) y con giros bruscos (b).



(a) (b)

Efectos de los giros bruscos en la localización del robot. En (a) el robot se dirige hacia "arriba" y efectivamente se mueve hacia arriba, pero en (b), el efecto de giro brusco se traduce en un "hacia la derecha" aunque este programado que vaya hacia "arriba"

Figura 10. Giros del Robot.
Fuente: Elaboración Propia.

Una forma de minimizar el efecto de los giros bruscos sobre el posicionamiento del robot consistió en disminuir la potencia de los motores, de forma que los giros fuesen más suaves.

3) Espiral 3: Manejo de la comunicación en el sistema

3.1) Establecimiento del la forma de comunicación en el sistema

Las siguientes preguntas se derivan de la parte de navegación:

¿Qué acción debería ejecutar el robot en las intersecciones límite del ambiente, donde una o dos direcciones de giro no se encuentran disponibles?

¿De que forma los robots podrían coordinar sus posiciones sin que existan colisiones entre los mismos?

En esta fase se trató de determinar como la comunicación puede ser efectiva en el entorno multiagente, y cuales fueron los medios y protocolos de comunicación empleados.

En las experiencias con un solo agente, la comunicación no es relevante, pues el ambiente es descubierto por un único agente En cambio, en los sistemas multiagente, la comunicación es fundamental para la coordinación entre los robots que están explorando el ambiente (Vlassis, 2003)

El RCX del kit de LEGO® MindStorms® posee una cantidad limitada de memoria (32Kb), lo que no permite implementar programas muy extensos. Además, el *firmware*[8] o sistema operativo, ocupa espacio importante en la memoria, disminuyendo aún más la capacidad para almacenar programas de usuario. El *firmware* de LeJOS ocupa 16 Kb de la memoria, dejando disponibles los restantes 16Kb de memoria para programas de usuario. La aplicación de navegación tenía ya un tamaño de 8 Kb en total, haciendo casi imposible cargar otra aplicación que tenga un tamaño superior a 8 Kb. Como conclusión, el modulo de aprendizaje debía residir en un lugar externo al RCX del robot y que el mismo tuviese una forma de acceder a la información que reside en el mismo.

Partiendo de esta idea, se determinó la creación de un ente intermediario entre los robots, que pudiese almacenar toda la información que no pudiesen almacenar los robots, y pudiese servir de moderador en la comunicación en los robots. Este ente

[8] El *firmware* es un software que es instalado vía Terminal infrarrojo hasta el RCX y permite ejecutar programas de la PC.

sería una computadora, que con su capacidad de cálculo podría coordinar a todos los robots participantes, comunicándose con cada uno de los RCX's por el Terminal infrarrojo USB (Terminal IR) incluido en el kit de LEGO® MindStorms®, como se aprecia en la **figura 11.** Este elemento se convertiría en el servidor del sistema.

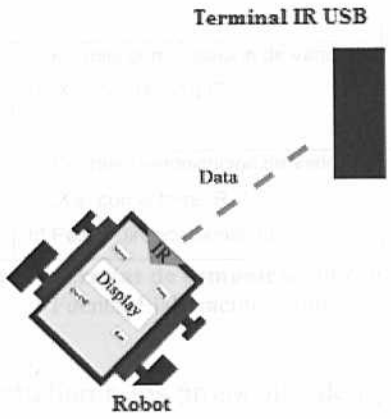


Figura 11. Comunicación del Robot con la torre IR.
Fuente: Elaboración Propia.

3.2) Selección y diseño del protocolo de comunicación

En la siguiente actividad se procedió a determinar los medios de comunicación que se utilizarán, dependiendo de lo que se desea comunicar. El primer planteamiento consistió en que la comunicación se llevará a cabo entre cada robot para conocer la ubicación del ambiente en la que se encontraba. LejOS posee diversidad de librerías que permiten la comunicación entre varios RCX's, en la **tabla 2** se aprecian las ventajas y desventajas de las más utilizadas.

Para la programación de la comunicación a bajo nivel en LejOS se seleccionó la librería ***RCXLNPAddressingPort***, que contiene funciones para el direccionamiento de mensajes entre la torre IR y los RCX's, así como entre RCX's. Sin embargo, tanto el robot como el medio de comunicación presentaron problemas: el robot no contaba con la suficiente memoria como para manejar simultáneamente la parte de aprendizaje y comunicación entre los robots; además, la comunicación entre los transmisores IR tanto de la torre infrarroja como del RCX no es full dúplex, es decir, el transmisor es incapaz de recibir y enviar mensajes de diferentes fuentes al mismo

tiempo; tampoco puede establecer comunicación simultanea entre dos o mas RCX's. Esta situación indujo a que se tuviese que explorar otra forma de comunicación entre los robots.

	Ventajas	Desventajas
RCXPort	a) Dificultad de codificación: baja.	a) No permite la comunicación entre varios RCX's con la torre IR.
LLC	a) Permite comunicación de varios RCX's, con la torre IR.	a) Alta dificultad de codificación, debido a que requiere programación a bajo nivel.
RCXLNPAddressingPort	a) Permite comunicación de varios RCX's, con la torre IR. b) Posee direccionamiento.	a) Dificultad de codificación: baja.

Tabla 2. Librerías de comunicación con Lejos.
Fuente: Elaboración Propia.

Para el análisis se estudiaron los protocolos de tipo amo-esclavo (donde a los esclavos se les concede el permiso de transmitir por una estación principal en la red) y protocolos de *token*[9] (al agente que posee el *token* se le permite enviar data y después cede el *token* al siguiente agente). En estos protocolos, el maestro inicia la comunicación, enviando datos. El esclavo se bloquea hasta recibir mensajes del maestro y sólo puede enviar data en la confirmación (*ack*) para los datos recibidos del maestro. Si no hay datos para enviar, cada lado sólo envía mensajes de confirmación planos, manteniendo sincronizados las dos partes. La secuencia, que en este caso es circular (cuando llega al último robot se reinicia la secuencia) es incrementada solo por el maestro; el esclavo sólo recibe los mensajes que sean dirigidos a él, esto significa que la dirección del remitente coincida con el numero de secuencia.

Después de este análisis, se adaptó el protocolo de comunicación basado en *token*, en el cual, de acuerdo a una secuencia ordenada, cada robot establece comunicación con la torre IR conectada a la computadora. El protocolo de comunicación diseñado funciona de la siguiente forma:

1. Cada agente que llega a una intersección se bloquea hasta que reciba mensajes del servidor.

[9] Un *token* es una variable, que es pasada de agente en agente, y controla el flujo de comunicación en la red.

2. El servidor inicia la transmisión con cada agente enviándole un valor de inicio de transmisión.
3. El robot acepta la transmisión enviándole las coordenadas de su ubicación.
4. El servidor verifica las coordenadas que le envía el robot y calcula la dirección en la que debe desplazase el agente.
5. El servidor le envía la dirección en la que debe desplazarse el robot, éste confirma la recepción del mensaje.
6. El servidor finaliza la comunicación con el agente y pasa la comunicación al siguiente agente en la secuencia de robots.

La sesión de comunicación, y por ende, la sesión de comunicación del servidor con los robots finaliza cuando uno de los robots llega al objetivo.

Es importante señalar que el protocolo no distingue prioridad entre los agentes; todos tienen la misma prioridad de comunicación. La política del servidor para el inicio de la comunicación entre cada agente es tener un ciclo de retransmisión si el robot no confirma el inicio de transmisión la primera vez. Si en diez intentos el servidor no ha podido establecer comunicación con el RCX, pasa la comunicación al siguiente robot en la lista.

Luego de diseñar el protocolo de comunicación, se comenzó a desarrollar la rutina que reside del lado de la computadora, el cual contiene una clase principal `Servidor`, (en el **Apéndice B** se muestra el diagrama de clases para el servidor) que maneja la comunicación de la torre IR con los RCX's.

Para sincronizar los movimientos de los robots, evitar que no existan colisiones entre los mismos, permitir que no realicen giros hacia posiciones fuera del mapa, y distribuir la exploración del ambiente por diferentes sectores, se creó un registro de los movimientos de los robots, a manera de una simulación, en la cual cada movimiento realizado por los robots es simulado por agentes de software. De esta forma, un robot que se encuentre en un punto determinado del mapa del ambiente puede obtener un *feedback* del servidor que le permita desplazarse a otro punto del mapa sin salirse del mismo, ni colisionar con otros robots.

Se diseñó una estructura centralizada de control, la cual se muestra en la **figura 12**, que contiene por cada coordenada del mapa (encerrada entre corchetes), una lista de los caminos adyacentes a la coordenada, los cuales serían marcados por cada robot que recorra dicho camino (con un numero identificador del robot); también marca aquellos caminos inaccesibles o bloqueados por obstáculos. Los caminos marcados en cero (0) se encuentran disponibles, los marcados con (-1) son inaccesibles, y los marcados en (-2) presentan obstáculos, los cuales se irán descubriendo a medida que los robots exploren el mapa. Los restantes valores distintos de cero (1...n) representan los caminos visitados por el robot 1...n.

MATRIZ DE POSICIONES

F=Arriba I=Izquierda D=Derecha B=Abajo

	(0)	(1)	(2)	(3)	(4)
	F I D B	F I D B	F I D B	F I D B	F I D B
(4)	[-1-1 0 0]	[-1 0 0 0]	[-1 0 1 0]	[-1 1 1 0]	[-1 1-1 1]
(3)	[0-1 0 0]	[0 0 0 0]	[0 0 1 1]	[0 1 1 0]	[1 1-1-2]
(2)	[0-1 1 1]	[0 1 0 1]	[1 0 1 0]	[0 1-2 1]	[-2-2-1 2]
(1)	[1-1-2 1]	[1-2 1 0]	[0 1 0 1]	[1 0-2 1]	[2-2-1 2]
(0)	[1-1 0-1]	[0 0 0-1]	[1 0 1-1]	[1 1 2-1]	[2 2-1-1]

Figura 12. Matriz de posiciones.
Fuente: Elaboración Propia.

Las funciones del mapa de posiciones se encuentran en la clase ***Mapa***, y las funciones de simulación de los movimientos de los robots en el servidor se encuentran en la clase ***NavRobot***. Se estableció como política de navegación, y con el fin de distribuir los robots en el mapa de navegación, **un robot no puede navegar por el camino recorrido por otro robot**. El procedimiento mediante el cual los robots interactúan con el mapa de localización se representa de la siguiente manera:

1. Cada robot marca en el mapa de posición con su ID los caminos que haya visitado.
2. Cada robot marca con su ID la coordenada donde se encuentra en un instante determinado

3. Cuando un robot va a tomar una acción, este chequea en la matriz de posiciones si dicho camino ha sido visitado por otro robot (valor del camino = 0) y si la coordenada de destino también se encuentra disponible,
4. Una vez que el robot se mueve hacia la siguiente coordenada, este desmarca la coordenada en la que se encontraba y marca la coordenada hacia la que se dirige

Sin embargo esto podría traer inconvenientes de sincronización entre los robots, debido a que un robot que marca un camino visitado, podría bloquear para los otros robots la única salida hacia el objetivo. Sin embargo esta situación no afecta en medida general el objetivo del estudio, ya que en efecto, si hubiese un único camino que llevarse al objetivo, y uno de los robots encontró primero ese camino, efectivamente habrá un robot que llegará al objetivo.

Es importante aclarar, que cuando el agente en el servidor "marca" un camino o una coordenada, significa que actualiza el valor en la matriz de posiciones con su identificador (***indRob***), y cuando el robot la "desmarca", este actualiza dicho valor a cero (0) para indicar que la coordenada se encuentra disponible, esto es para resolver la situación en la que una coordenada sirve de intersección a caminos que se encuentran disponibles con caminos que están bloqueados.

Otro aspecto de importancia para el robot como entidad individual y grupal se refiere a la decisión que debe tomar el robot cuando se encuentra en una coordenada en la cual todos los caminos de acceso se encuentran inaccesibles, bien porque se hayan detectado obstáculos, sean camino que se encuentren en la frontera del mapa, o sean caminos ya visitados por otro robot.

Para garantizar que el robot pueda encontrar el objetivo, este cuenta con un mecanismo de retorno (*backtracking*) basado en el recorrido de una estructura de datos en forma de pila, que se llena progresivamente con las coordenadas que se van visitando. A medida que una coordenada se detecta como "inaccesible", se desempila hasta llegar a la ultima posición donde se detecto un "camino bueno". Una vez que el robot ha desempilado todas las coordenadas hasta llegar al inicio de su recorrido,

dicho agente se declara como "inactivo", y cede su proceso de comunicación a los demás robots que están "activos" en el sistema.

Si un robot esta "inactivo" (esto lo detecta el servidor por una bandera con la que se marca el robot) en servidor no establece comunicación con el mismo hasta la siguiente sesión de aprendizaje.

En esta fase se tomó una decisión de diseño para optimizar el proceso de desarrollo de la aplicación: se utilizaría para la simulación el mismo código proveniente de sistema físico. La decisión se fundamenta en dos ideas concretas:

- Reutilización de código, lo cual optimiza el proceso de desarrollo
- La utilización en la simulación del código del sistema físico permite realizar pruebas que vayan más allá de la mera simulación del proceso: se está trabajando con el código que será implantado en el sistema real. Esto además permite detectar fallas antes de la fase de montaje definitivo de todos los elementos del sistema físico.

La decisión de conservar la interfaz grafica en Visual Basic se debe a que ya se había desarrollado completamente el modulo para generar el laberinto, y la idea principal era la reutilización de código.

La forma de comunicar la interfaz grafica desarrollada en Visual Basic con la aplicación desarrollada en Java seria mediante pipes, o archivos de texto, que alimentan la información del la simulación hacia el sistema físico y viceversa:

- En la aplicación de la simulación en Visual Basic se diseña la configuración del mapa del ambiente y la posición de los robots. Esto genera dos (2) archivos de texto (*.txt) con la configuración de los obstáculos del ambiente, y otro con la posición de los robots.
- La aplicación en Java es iniciada por un archivo batch (*.bat), y lee los archivos de texto, que se encuentran en una ubicación especifica (..\Mapas\Mapa.txt, y ...\Mapas\MapaObstaculos.txt). Estos archivos contienen la información de la posición de los robots y los obstáculos.

- La aplicación en Java genera la corrida de los robots y genera un archivo con el movimiento de los robots (...\Reportes\MapLog.txt)
- La aplicación en Visual Basic lee el archivo del movimiento de los robots y reproduce las acciones realizadas por los mismos.

La diferencia fundamental entre el desarrollo de la simulación y el sistema físico radica principalmente en la parte de la comunicación. En la simulación la comunicación se lleva a cabo directamente en la aplicación y no se necesita una clase para implementar la comunicación física. Sin embargo el resto de las funciones se llevan acabo de la misma forma en la simulación y el sistema físico. Por esta razón la comunicación se almacena en la clase ***ComunicacionRobot*** (que se omite en el lado de la simulación).

4) Espiral 4: Manejo del aprendizaje en el sistema

4.1) Selección, planificación y diseño del modelo de aprendizaje más adecuado para el sistema

Desde un principio se tuvo claro que el aprendizaje debía ser en tiempo real; esto se traduce en que se necesita una herramienta por el cual el conjunto de robots vayan aprendiendo en vivo acerca del ambiente y colaborando entre ellos para lograr tal efecto; por esto se seleccionó la técnica de aprendizaje por refuerzo.

Los puntos importantes a determinar en esta fase es la selección de estados, la escogencia de la función refuerzo, y el diseño del algoritmo para la actualización de los valores.

4.2) Selección de estados

Los estados se colocaron en una estructura de tabla hash o indexada, para así garantizar su acceso mediante la invocación de los elementos que conforman la percepción del agente: las coordenadas de posición $(x,y)_t$ en el tiempo t, el conocimiento del estados de los obstáculos en el punto $(x,y)_t$, los cuales ha

descubierto hasta el momento, y la posición absoluta del robot pA={NORTE, SUR, ESTE, OESTE}

La ubicación de los obstáculos descubiertos en el tiempo t se colocaron en formato binario (0 si no hay obstáculo, 1 si lo hay) en el espacio de la intersección pA={NORTE, SUR, ESTE, OESTE}. Esto da un resultado total de 1600 estados para el ejercicio (5x5x4x2x2x2x2 = 1600). La definición completa del estado quedaría como se muestra en la **figura 13**.

Coordenada Y	Coordenada X	Posición Absoluta	Obstáculo (NORTE)	Obstáculo (SUR)	Obstáculo (ESTE)	Obstáculo (OESTE)
5v	5v	4v	2v	2v	2v	2v

Figura 13. Definición estados.
Fuente: Elaboración Propia.

Ejemplo de estado

0130100 = Coordenadas (1,0), Posición absoluta SUR, solo un obstáculo al SUR

2301110 = Coordenadas (3,2), Posición absoluta NORTE, obstáculos al NORTE, OESTE y ESTE

Luego de realizar pruebas con el sistema físico, se observó que mantener a la ubicación de los obstáculos como una parte del estado iba a representar una complicación. El robot no cuenta con los sensores necesarios para detectar los obstáculos cuando llega a una coordenada, que es el momento en el cual el robot analiza su estado actual. Esto implica que el robot en una determinada sesión de aprendizaje puede encontrarse realmente en un determinado estado de acuerdo a la configuración de los obstáculos a su alrededor, pero al no poder detectarlos todos al mismo tiempo, determinara que se encuentra en otro estado, lo cual influye en el refuerzo y el consiguiente aprendizaje del recorrido.

Por esta razón se decidió eliminar de la definición del estado a la ubicación de los obstáculos, dejando solo como definición de estados a la coordenada de ubicación y la posición absoluta inicial, que efectivamente son elementos que puede detectar el

robot una vez que llega a una coordenada y determina con precisión el estado en el que se encuentra, como se muestra en la **figura 14**.

Coordenada Y	Coordenada X	Posición Absoluta
5v	5v	4v

Figura 14. Redefinición de estados propuesto.
Fuente: Elaboración Propia.

Ejemplo de estado

013 = Coordenadas (1,0), Posición absoluta SUR

231 = Coordenadas (2,3), Posición absoluta OESTE

Esto da un resultado total de 100 estados para el ejercicio (5x5x4 = 100). A pesar que se disminuye drásticamente la cantidad de estados, representa una cifra significativa para un ejercicio de esta magnitud.

4.3) Política de Selección de las acciones

Una vez que se han seleccionado los estados, se procede a modelar la estructura de aprendizaje, que en este caso seria una matriz, la cual contiene un índice en el estado que se encuentra actualmente el agente, y por cada estado un conjunto de acciones, que son las acciones que puede tomar el robot {ARRIBA, IZQUIERDA, DERECHA, ABAJO}. Cada acción tiene un valor inicializado con ceros o con una semilla aleatoria, tal como se muestra en la **figura 15** (en este caso inicializado con valores aleatorios).

Índice estado	Acción (ARRIBA)	Acción (IZQUIERDA)	Acción (DERECHA)	Acción (ABAJO)
1	0.5540642182507065	0.9065059349059392	0.678292594114425	0.2580756335009564
2	0.2567741597100365	0.8256158956417472	0.33266452283637926	0.4703028571322908
3	0.387784413412647	0.155339358330144	0.05049120029915122	0.9342368821149583
4	0.14712125790226305	0.31524575392194554	0.8437253042228029	0.2184643051634042

Figura 15. Estructura de Aprendizaje.
Fuente: Elaboración Propia.

Para determinar la política que mejor se aplicara al trabajo de investigación, se realizó un estudio de las técnicas implementadas (Russell, 1996; Bowling, 2003;

Errecalde, 1999) y luego se procedió a adaptar el modelo seleccionado a los requerimientos de la investigación. El modelo seleccionado utiliza la técnica ***greedy*** que se basa en la selección de la acción con el mayor valor estimado. La adaptación que se realizó permite que las acciones que tome un robot estén coordinadas con las acciones de los otros robots a fin de evitar colisiones, así como también evitar que el robot pueda salirse del mapa del ambiente.

La política implementada, a la cual denominamos ***m-greedy*** (*multiagent-greedy*), se compone de una técnica ***greedy*** con una verificación previa de las acciones, con el objetivo de evitar colisiones. Consta de las siguientes fases:

- Las posibles acciones para el estado en el cual se encuentra el agente, se ordenan de mayor a menor.
- Se selecciona la acción con el mayor valor y se verifica si dicho camino y coordenada de destino están disponibles (esto significa que el camino no haya sido visitado por otro robot, que el camino no conduzca fuera del mapa y que la coordenada de destino no esté siendo ocupada por otro robot)
- Si alguna de las condiciones anteriores no se cumplen, se selecciona la siguiente acción del conjunto ordenado, y se verifica su disponibilidad; esto se repite hasta que se hayan explorado todas las acciones o hasta que se haya conseguido una acción en la cual su camino y coordenada destino se encuentren disponibles

Un ejemplo de esto es: si la acción de mayor valor es *ir a la derecha* pero hacia la derecha hay fin de camino un obstáculo ya detectado o ya ha sido visitado por otro robot, este elige la acción con el siguiente valor y así sucesivamente hasta explorar todas las acciones posibles en dicho estado.

4.4) Manejo de la Función refuerzo

La función refuerzo utilizada consta de varias refuerzos independientes, que ayudan al robot a acercarse al objetivo. La función de refuerzo no enseña directamente al robot como llegar directamente al objetivo, solo entrena al robot

como acercarse al objetivo por el camino mas corto sin obstáculos. La función refuerzo presupone que el ambiente satisface la condición de que al menos hay un camino libre en el cual el robot puede llegar al objetivo sin chocar.

Partiendo de la idea expresada en el anterior párrafo, la función refuerzo se condicionó de acuerdo a si logra el objetivo principal (llegar al final del laberinto) o los objetivos inmediatos (encontrar un camino libre para avanzar al siguiente estado).

Se manejaron las siguientes recompensas y castigos para cada robot:

Comportamiento del robot	Recompensa (+) o Castigo (-)
Alcanzo las coordenadas del objetivo	+3
No consiguió camino disponible para avanzar	-0.5
Alcanzo efectivamente la siguiente posición	+1
Consiguió un obstáculo	-1
Distancia actual al objetivo < Distancia anterior al objetivo	+2
Distancia actual al objetivo > Distancia anterior al objetivo	-2

Tabla 3. Recompensas y Castigos fijados para el experimento.
Fuente: Elaboración Propia.

Es importante recalcar que los valores de **Distancia actual** y **Distancia anterior** se encuentran medidas con respecto al objetivo.

4.5) Diseño de función de actualización de estados

De entre los algoritmos de aprendizaje en tiempo real, Q-learning fue el escogido. Además de las razones expuestas en el Marco Referencial, Q-Learning tiene la capacidad de adaptarse a ambientes dinámicos, y con condiciones continuamente cambiantes (Errecalde, 1999)

En Q-learning, la función de actualización de estados se diseñó de la siguiente manera:

$$Q^t(s^{t-1}, a^{t-1}) = (Q^t(s^{t-1}, a^{t-1}) + \alpha[r(a^{t-1}, s^t) + \gamma * max_{a \in A}\, Q^t(s^t, a^t) - Q^t(s^{t-1}, a^{t-1})]$$

Donde Q^t es la matriz de aprendizaje, s^{t-1} es el estado anterior, a^{t-1} la acción en el estado anterior, s^t el estado actual, a^t es la acción en el estado actual, γ es el factor de descuento y α es la tasa de aprendizaje.

Sin embargo, dicha función, originalmente diseñada para el aprendizaje de un solo agente se adaptó para el aprendizaje multiagente. En este ejercicio el medio de enlace (servidor) sirve para ampliar el conocimiento de cada uno de los agentes, de forma que todo el conjunto se comporta como un gran agente que aprende en paralelo de cada una de las acciones que ejecutan los agentes. Por esta razón cada agente i actualiza la matriz de la forma siguiente:

$$Q^t(s^{t-1}_i, a^{t-1}_i) = (Q^t(s^{t-1}_i, a^{t-1}_i) + \alpha_t[r(a^{t-1}_i, s^t_i) + \gamma * max_{a \in A} Q^t(s^t_i, a^t_i) - Q^t(s^{t-1}_i, a^{t-1}_i)]$$

4.5) Desarrollo del algoritmo de aprendizaje.

La idea principal para el desarrollo del algoritmo de aprendizaje era cada vez que el robot llega al punto de cruce, obtenga *feedback* del servidor. Cuando el robot detecte un obstáculo, se devuelve a su posición original, para luego informarle de esto al servidor en la próxima transmisión, luego de esto se corrige la posición del robot en el servidor, para calcular la recompensa y actualizar la matriz de aprendizaje.

En la primera fase de desarrollo se construyó el algoritmo de forma que evaluara la posición actual del móvil y lo almacenara como el estado actual, luego determinar la acción y calcular el estado futuro para en ese momento actualizar la matriz; sin embargo ello acarreaba problemas cuando el robot tenia que devolverse cuando encontrase un obstáculo. Para ello se modifico el algoritmo para que en la primera iteración almacenara el estado actual, el cual se convertiría en *estado anterior* cuando llegase a la siguiente intersección, verificase realmente la posición a la que llego y almacenara esto como estado actual y actualizara la matriz. Ello se resolvió añadiendo las variables *estado_anterior y accion_anterior* a cada robot, para que en cada iteración tuviesen almacenados estos valores que se actualizan en la iteración anterior. Si para un estado anterior, el estado actual *no es alcanzable*, el robot deberá devolverse hasta la próxima posición conocida, no el estado anterior, en el cual volverá a explorar sus opciones. Si el próximo estado es alcanzable desde el estado actual se aplicara el refuerzo correspondiente para poder actualizar los Q-valores.

4.6) Definición de la variación de Q-learning multiagente implementado.

El algoritmo Q-Learning utilizado se muestra en la **figura 16.**

Inicializar t=0
Inicializar Q^t con valores aleatorios

Para cada episodio t
  Para cada agente i
    Si t>0
      Observar estado s^t_i
      Calcular recompensa $r(a^{t-1}_i, s^t_i)$
      Actualizar Q^t
        $Q^t(s^{t-1}_i, a^{t-1}_i) = (Q^t(s^{t-1}_i, a^{t-1}_i) + \alpha[r(a^{t-1}_i, s^t_i) + \gamma * max_{a \in A} Q^t(s^t_i, a^t_i) - Q^t(s^{t-1}_i, a^{t-1}_i)]$
      Almacenar s^t_i
      Seleccionar una acción $(a^t_i)_1, (a^t_i)_2 \ldots (a^t_i)_n$ de acuerdo a una política π
    Sino
      Almacenar s^t_i
      Seleccionar una acción $(a^t_i)_1, (a^t_i)_2 \ldots (a^t_i)_n$ de acuerdo a una política π
Hasta que t sea Terminal

Figura 16. Pseudocódigo del Algoritmo Q-Learning implementado.
Fuente: Elaboración Propia.

4.7) Problemas encontrados en la estructuración de las sesiones de aprendizaje

En el desarrollo de la simulación las secuencias de aprendizaje se pueden realizar simplemente con un ciclo ***for*** para un determinado numero de iteraciones. Sin embargo en el sistema físico, se debe invertir tiempo de corrida mientras se vuelven a colocar los robots en su posición inicial. Por esta razón se decidió implementar un método de escritura y carga de un archivo de texto que almacena el contenido de la matriz de valores Q (mQ.txt), de forma que al finalizar una sesión de aprendizaje, se llene el archivo con el contenido de la matriz, y luego mediante la carga del archivo, la información de la iteración anterior pueda ser utilizada. Las funciones para el aprendizaje de los agentes se encuentran en la clase ***AprendizajeRefuerzo*** (ver **Apéndice B**, donde se muestra el diagrama de clases del servidor).

5) Espiral 5: Montaje del diseño definitivo del sistema

5.1) Coordinación de los elementos de navegación, comunicación y aprendizaje

Una vez que han sido evaluados todos los aspectos del sistema, se integran en el sistema definitivo, en el cual se realizaran las respectivas pruebas.

Para cargar los datos iniciales de la corrida del sistema, se utilizó, de la misma forma que en la simulación, dos archivos de texto; Mapa.txt (contiene la ubicación inicial de los robots, y la posición del objetivo) y MapaObstaculos.txt (contiene la posición de los obstáculos en el mapa del laberinto). Esto se hace de forma que el servidor pueda detectar la situación en la que se encuentra el robot físicamente, y ofrecerle un *feedback* (como por ejemplo, notificarle cuando este haya llegado al objetivo).

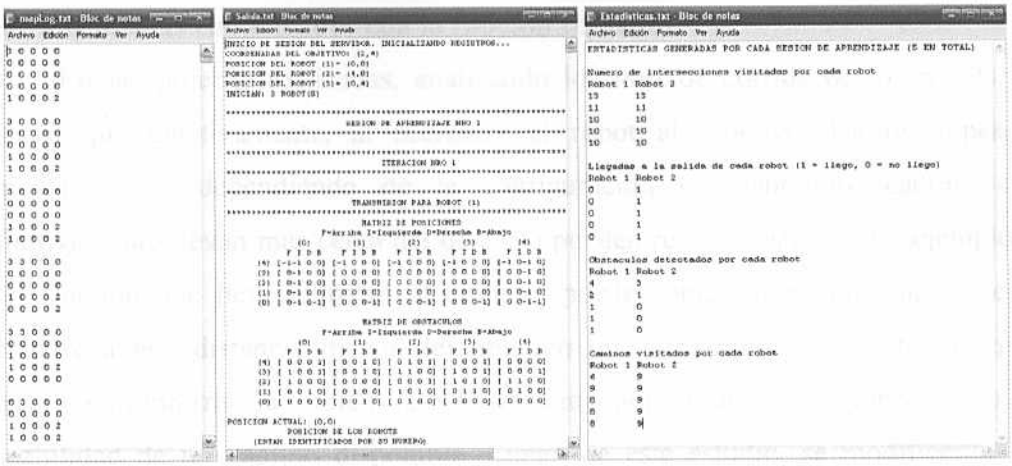
5.1.1) *Logs* generados en la aplicación

En la aplicación se generaron diferentes ***logs***, dependiendo del aspecto de la corrida del sistema que se quería observar:

a) Salida.txt (ver Figura 17): Archivo principal que muestra el comportamiento del sistema. Imprime por cada iteración de la corrida de la aplicación, los valores de los estados que se están recorriendo, las recompensas obtenidas, los caminos visitados, y los obstáculos detectados en el recorrido de cada robot.

b) mapLog.txt: Muestra la secuencia de pasos de la matriz de posiciones de los robots.

c) Estadisticas.txt: Muestra las estadísticas arrojadas en la corrida del sistema: número de pasos hasta encontrar el objetivo, obstáculos detectados y número de caminos explorados.



(a) (b) (c)

Figura 17. *Logs* generados por la aplicación.
Los tres *Logs* generados: *MapLog.txt* (a), *Salida.txt* (b) y *Estadisticas.txt* (c).
Fuente: Elaboración propia

5.1.2) Problemas encontrados en la localización de los robots

Un problema que se presenta en el sistema físico es que los errores provenientes del hardware inducen efectos no deseados en el aprendizaje del robot. Los giros que no están bien controlados pueden inducir fallas en la localización y posicionamiento del los robots, y por consecuencia impiden el alcance del objetivo.

El robot solo sabía los caminos que no estaban ocupados, no las intersecciones como tal, por lo que el robot debía dejar una bandera sobre el camino que estaba visitando actualmente y desmarcarlo cuando hubiese abandonado dicha posición.

Sin embargo el problema que se presentaba con esto, es que el robot podía volver a su posición original si encontraba un obstáculo en el camino, lo cual sólo podría hacerse del conocimiento de los demás robots al momento que este llegase a la intersección que es en el momento que recibe un feedback del servidor. Por ello se creó una marca de visita temporal, que se activa cuando el robot abandona la posición y que sólo se desmarca cuando se asegura que el robot llegó a su posición de destino.

5.1.3) Problemas encontrados con la función refuerzo

En las pruebas realizadas, analizando los ***logs*** de corrida de los robots, se observó que efectivamente, al acercarse el robot al objetivo, las recompensas aumentan, pero dependiendo de la configuración del laberinto, camino que definitivamente llevan más cerca del objetivo pueden recibir castigos. Por ejemplo el único camino que lleve al robot al objetivo puede comenzar en un punto que se encuentre a una distancia mayor del objetivo que otro punto. Por esta razón se determinó minimizar la acción de los castigos por la distancia respecto con la deseabilidad de un camino disponible. Luego de este estudio, se modificaron los pesos de las recompensas y castigos, quedando como se muestra en la **tabla 4**.

Comportamiento del robot	Recompensa (+) o Castigo (-)
Alcanzo las coordenadas de destino	+3
No consiguió camino disponible para avanzar	-0.5
Alcanzo efectivamente la siguiente posición	+1
Consiguió un obstáculo	-1
Distancia actual al objetivo < Distancia anterior al objetivo	+0.75
Distancia actual al objetivo > Distancia anterior al objetivo	-0.75

Tabla 4. Recompensas y castigos Modificados para el experimento.
Fuente: Elaboración Propia.

5.2) Pruebas en conjunto del sistema

En esta fase, las pruebas se realizaron en el sistema, uniendo todos los factores estudiados por separado. En las fases anteriores se realizaron las pruebas respectivas a cada uno de los elementos que conforman el sistema; en esta parte se puede evaluar el sistema como un todo, y analizar su desempeño.

Las pruebas realizadas permiten medir la efectividad del algoritmo de aprendizaje implementado frente a un conjunto de robots. Debido a que se cuenta con poca cantidad de agentes físicos (apenas dos robots), la simulación es la técnica de evaluación escogida para las pruebas de carga de trabajo. Se determinó que el estudio debería apoyarse más en los resultados arrojados por la simulación, y complementar los mismos con los resultados de las pruebas en el sistema real. La simulación es

también es una prueba representativa del sistema real (tomando en cuenta que la simulación se desarrolló con las clases del sistema físico).

Básicamente se quiere evaluar el desempeño del sistema, colocando varios robots, dispuestos en coordenadas específicas del mapa. También se quiere evaluar el desempeño del algoritmo de aprendizaje si se inicializa la matriz de Q valores en cero o si se inicializa con valores aleatorios.

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. Para cada prueba realizada se estableció un número de 100 iteraciones, y al final de la corrida del programa, se extrajeron estadísticas derivadas de cada prueba, para lo que se recurrió a la información obtenida por los ***logs*** de la aplicación, y a partir de dicha información se generaron las graficas que permiten realizar las evaluaciones definitivas.

Para todas las pruebas realizadas se fijó el valor de la tasa de aprendizaje (α) en 0,5, y el valor para el factor de descuento (γ) se fijo en 0,6. De acuerdo a la bibliografía consultada (Dos Santos, D. y Peñalver, R, 2003), estos valores son ideales para obtener resultados con una convergencia media en el tiempo. No se realizaron pruebas adicionales con estos valores ya que no es el objetivo principal de esta investigación.

La descripción completa de las pruebas realizadas y las graficas asociadas a los resultados de estas pruebas se presentan en la sección de resultados.

CAPÍTULO VI
RESULTADOS

Las diferentes fases de desarrollo del sistema arrojaron resultados que sirvieron para la consecución de las otras fases. Sin embargo, los resultados más relevantes, que están relacionados con el prototipo final, serán mostrados en detalle en esta sección.

1) PARÁMETROS Y MÉTRICAS DE LA EVALUACIÓN

De acuerdo a un análisis de la funcionalidad del sistema se determinaron los siguientes parámetros y métricas de evaluación tanto para el sistema físico como para la simulación:

a) Parámetros: Existen infinidad de parámetros que afectan el desempeño del sistema. Esta evaluación se concentró en cuatro parámetros, claves para determinar la eficiencia del sistema:

- Número y disposición de los robots que interactúan en el ambiente
- Cantidad y disposición de los obstáculos
- Posición del objetivo
- Configuración inicial de la matriz de aprendizaje

b) Métricas: Con los resultados de las pruebas se intentó determinar cuan efectivo es el sistema desarrollado. El sistema sería mas eficiente mientras trate de llegar al objetivo en el menor numero de pasos posible y con la menor cantidad de obstáculos detectados; por esta razón se plantearon tres métricas para la evaluación:

- **Número de pasos para llegar al objetivo**, específicamente del robot que llega al objetivo, debido a que sólo uno de los robots llega al objetivo. Este número de pasos representa una medida de tiempo para los robots en el sistema.

- **Cantidad de obstáculos detectados**, tanto del robot que llega al objetivo como de todos los robots en el sistema.
- **Porcentaje (%) de mapa del ambiente explorado**, tanto del robot que llega al objetivo como de todos los robots en el sistema. Para determinar este porcentaje, se calcula el número de caminos visitados por cada robot, y se divide entre el número de caminos posibles en el mapa (que serían el número total de caminos en el mapa menos el número de caminos bloqueados por obstáculos)

2) DESCRIPCIÓN DE LAS PRUEBAS REALIZADAS Y RESULTADOS OBTENIDOS

Para la realización de estas pruebas se estableció un modelo del ambiente tal como se muestra a continuación. En este mapa (ver **figura 18**) el número de caminos posibles es de 26 (40 caminos – 14 caminos bloqueados por obstáculos)

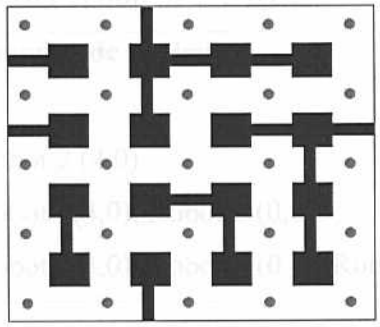


Figura 18. Modelo de mapa del ambiente propuesto.
Fuente: Elaboración propia.

Básicamente se plantearon 2 modalidades de pruebas: pruebas con objetivo central, y pruebas con cambio de objetivo.

2.1) PRUEBAS CON OBJETIVO CENTRAL.

En estas pruebas (1° y 2° conjunto de pruebas) se evaluó el desempeño del sistema colocando el objetivo en un punto central del mapa del ambiente de forma que exista la misma probabilidad de llegar al objetivo para todos los robots. En cada

prueba del conjunto se varió el número de robots en el sistema. Cada conjunto de pruebas consta de cuatro (4) pruebas, incluyendo 1, 2, 3 y 4 robots en el sistema, tal como se puede apreciar en la **figura 19**.

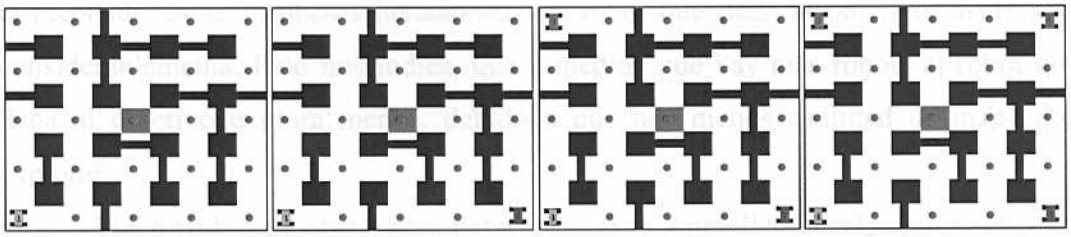


Figura 19. Posición de los robots y del objetivo en las diferentes pruebas de objetivo central.
Fuente: Elaboración propia.

1° Conjunto de pruebas. Objetivo central, matriz Q inicial aleatoria.

Valores iniciales de la matriz de aprendizaje Q: Aleatorio.
Coordenadas del objetivo: (2,2).
Número de Iteraciones o sesiones de aprendizaje: 100 (Sólo se graficaron las primeras 50 Iteraciones)
Número de pasos óptimo hasta el objetivo: 5 pasos.
Robots utilizados y coordenadas de posición: **1 Robot:** Robot 1 (0,0) **2 Robots:** Robot 1 (0,0), Robot 2 (4,0) **3 Robots:** Robot 1 (0,0), Robot2 (4,0), Robot 3 (0,4) **4 Robots:** Robot 1 (0,0), Robot2 (4,0), Robot 3 (0,4), Robot 4 (4,4)

Los resultados para las métricas seleccionadas, obtenidos en estas pruebas fueron los siguientes:

a) Número de pasos para llegar al objetivo (ver Apéndice C: Figura C-1a): se puede observar que independientemente de la prueba (con 1, 2, 3 o 4 robots), no impide que el robot que llegue al objetivo converja al número óptimo[10] de pasos,

[10] En los siguientes resultados, cuando se hace referencia a *número óptimo*, se basa en un resultado prefijado calculado mediante un análisis o conteo inicial, independiente del ejercicio. En ningún momento, el robot no posee la capacidad de determinar a cual es el valor óptimo para una determinada métrica.

que en este caso es 5. También se observa en la prueba con 1 robot, que en las primeras iteraciones el número de pasos o recorrido es extremadamente mayor. Y a medida que van ingresando más robots en el ambiente (pruebas con 2, 3 y 4 robots), el recorrido en las primeras iteraciones, del robot que llega al objetivo, disminuye considerablemente. Esto nos indica, que a medida que hay más robots el robot que llega al objetivo explora menos, debido a que hay menos cantidad de mapa que explorar.

b) Cantidad de obstáculos detectados (ver Apéndice C: Figuras C-1b y C-1c): En el caso del número de obstáculos detectados para el robot que llega al objetivo, se observa una tendencia similar, independientemente de la prueba (con 1, 2, 3 o 4 robots), se produce una convergencia a número óptimo de obstáculos detectados (cero). A pesar de observarse un comportamiento optimo para el robot que llega al objetivo, la detección de obstáculos en conjunto, permite determinar que los robots que no llegan al objetivo, detectan continuamente obstáculos. Esto nos indica que a medida que ingresan más robots al ambiente (pruebas con 2, 3 y 4 robots) se detectan más obstáculos. Para analizar con detenimiento este comportamiento, se revisaron los ***logs***, de la corrida, y se observó que en una iteración determinada los robots que intentan encontrar el camino óptimo se bloquean el uno al otro en una parte del recorrido. Cuando uno de los robots no tiene otro camino disponible, opta por dirigirse al camino aparentemente disponible, pero que contiene un obstáculo. En la siguiente iteración, si los demás robots han reforzado la ruta anterior, bloquearan nuevamente el camino, dejando nuevamente al robot sin otra opción que irse por el camino que contiene el obstáculo (aunque no "sabe" que esta allí, pues la ubicación de los obstáculos desaparece en la nueva iteración).

c) Porcentaje (%) de mapa del ambiente explorado (ver Apéndice C: Figuras C-1d y C-1e): Para el robot que llega al objetivo, se observa la clara convergencia hacia un número de caminos a explorar, que sin duda concuerdan con el número de pasos óptimo hasta el objetivo (5 pasos = 4 caminos explorados / 26 caminos disponibles = 15% de mapa explorado).

Para la exploración en conjunto realizada por los robots, se observa una tendencia creciente en la exploración del mapa. No obstante, el porcentaje de exploración en iteraciones iniciales para 1 robot es casi semejante al % explorado por 3 robots en la convergencia. La tendencia observada es creciente pero no lineal, la diferencia de exploración entre los robots va disminuyendo a medida que aumenta el número de robots en el mapa.

2° Conjunto de pruebas. Objetivo central, matriz Q inicial con ceros.

Valores iniciales de la matriz de aprendizaje Q: Ceros.
Coordenadas del objetivo: (2,2).
Numero de Iteraciones o sesiones de aprendizaje: 100 (Sólo se graficaron las primeras 50 Iteraciones)
Número de pasos óptimo hasta el objetivo: 5 pasos.
Robots utilizados y coordenadas de posición: **1 Robot:** Robot 1 (0,0) **2 Robots:** Robot 1 (0,0), Robot 2 (4,0) **3 Robots:** Robot 1 (0,0), Robot2 (4,0), Robot 3 (0,4) **4 Robots:** Robot 1 (0,0), Robot2 (4,0), Robot 3 (0,4), Robot 4 (4,4)

Los resultados para las métricas seleccionadas, obtenidos en estas pruebas fueron los siguientes:

a) Número de pasos para llegar al objetivo (ver Apéndice C: Figura C-2a): se observa en comparación con el modelo de matriz Q inicializada con ceros, que la convergencia hacia el número de pasos óptimo se realiza en un menor número de pasos y desde valores más pequeños. En el caso de la matriz con valores aleatorios, todos los valores convergen desde la sexta iteración, mientras que aquí todos los valores convergen desde la tercera iteración.

b) Cantidad de obstáculos detectados (ver Apéndice C: Figuras C-2b y C-2c): se observa que en las primeras iteraciones detectó obstáculos. La detección de

obstáculos para el robot que llega al objetivo, no se ve afectada en esta prueba por el número de robots en el ambiente. En conjunto los robots, detectan más obstáculos.

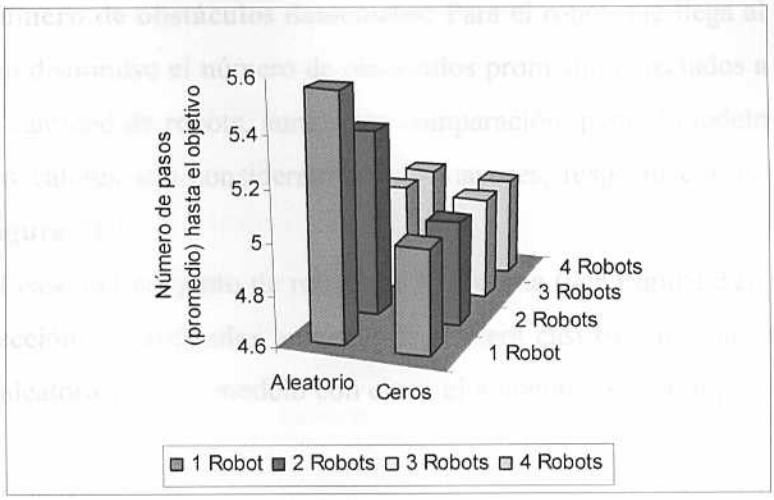
c) Porcentaje (%) de mapa del ambiente explorado (ver Apéndice C: Figuras C-2d y C-2e): en este caso la exploración para el robot que llega al objetivo, no se ve afectada en este modelo por número de robots en el sistema. La linealidad de la exploración con respecto al número de pasos hasta el objetivo se debe a que en las primeras iteraciones detectó obstáculos, que permitieron que explorara la misma cantidad de caminos en mayor número de iteraciones. Para el conjunto de robots se aprecia un aumento del porcentaje de mapa explorado en la convergencia para varios robots (2-4 robots), esto se explica en el hecho de la detección de obstáculos. En estos casos, debido a que la exploración es más directa que en el modelo con valores aleatorios, cuando los robots que no pueden llegar al objetivo detectan obstáculos en iteraciones iniciales, tratan de buscar caminos alternos sin obstáculos, lo que se traduce en una mayor cantidad de caminos explorados, ya que los mismos refuerzan ambas trayectorias. Esto hace que aumente el nivel de exploración total del sistema.

2.1.1) ANÁLISIS COMPARATIVO DE LOS RESULTADOS OBTENIDOS EN LAS PRUEBAS DE OBJETIVO CENTRAL.

En las graficas anteriores se puede observar como en ambos casos (matriz Q inicializada con ceros y matriz Q inicializada con valores aleatorios), las métricas llegan a su valor optimo, independientemente del número de robots en el sistema.

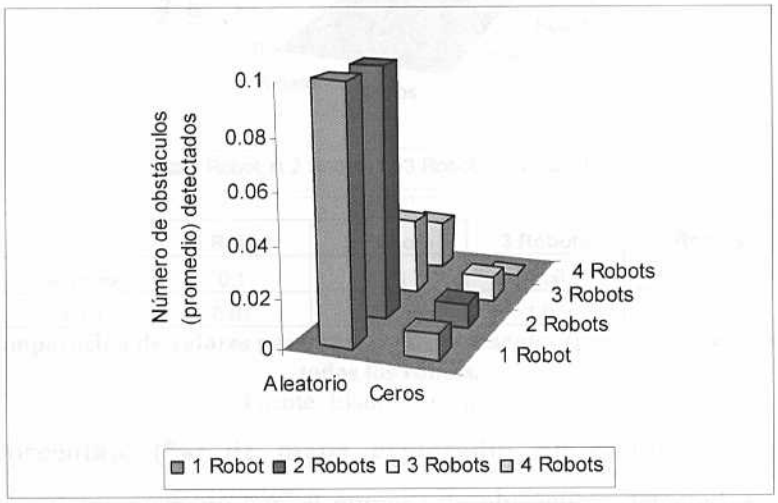
Se observaron ciertas tendencias en las métricas seleccionadas, con respecto a la cantidad de robots en el sistema. Para tener una visión mas clara de esta tendencia, se calculó el valor promedio de las métricas, para obtener una comparación.

a) Número de pasos hasta el objetivo (ver Figura 20): Se observa una tendencia de disminución respecto con el número de robots; a medida que aumenta el número de robots, disminuye el número de pasos promedio en llegar al objetivo. La disminución se aprecia más gradual para el modelo con matriz Q inicializada con valores aleatorios



	1 Robot	2 Robots	3 Robots	4 Robots
Aleatorio	5,56	5,34	5,03	5,02
Ceros	5,01	5,01	5,01	5

Figura 20. Comparación de valores para número de pasos (promedio) al objetivo. Robot que llega al objetivo.
Fuente: Elaboración propia.

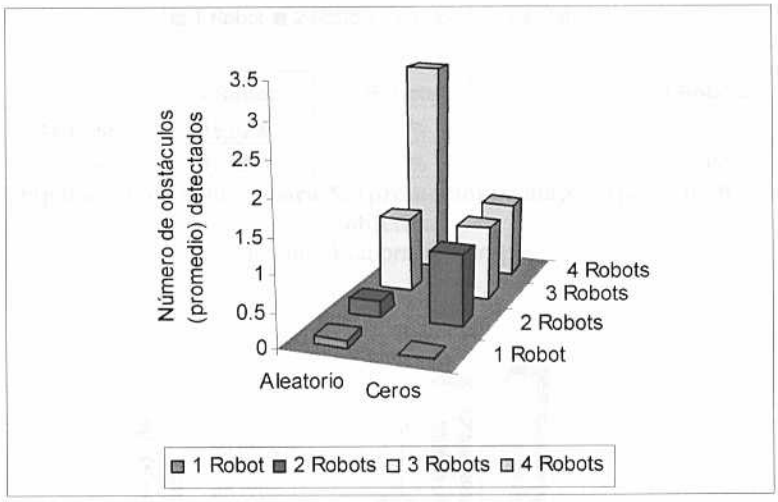


	1 Robot	2 Robots	3 Robots	4 Robots
Aleatorio	0,1	0,1	0,03	0,02
Ceros	0,01	0,01	0,01	0

Figura 21. Comparación de valores para número de obstáculos (promedio) detectados. Robot que llega al objetivo.
Fuente: Elaboración propia.

b) Número de obstáculos detectados: Para el robot que llega al objetivo, se observa como disminuye el número de obstáculos promedio detectados a medida que aumentan la cantidad de robots, aunque en comparación, para el modelo con valores aleatorios los valores son considerablemente mayores, respecto con el modelo con ceros (ver **Figura 21**).

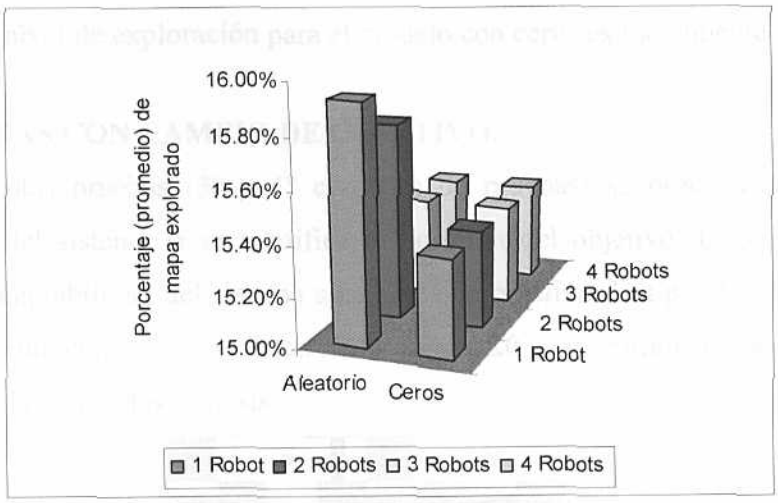
En el caso del conjunto de robots en el sistema (ver **Figura 22**), la tendencia hacia la detección de obstáculos aumenta de manera casi exponencial en el modelo con valores aleatorios. En el modelo con ceros el aumento es casi imperceptible, pero gradual.



	1 Robot	2 Robots	3 Robots	4 Robots
Aleatorio	0,1	0,21	1,06	3,05
Ceros	0,01	1,02	1,05	1,06

Figura 22. Comparación de valores para número de obstáculos (promedio) detectados. Suma de todos los robots.
Fuente: Elaboración propia.

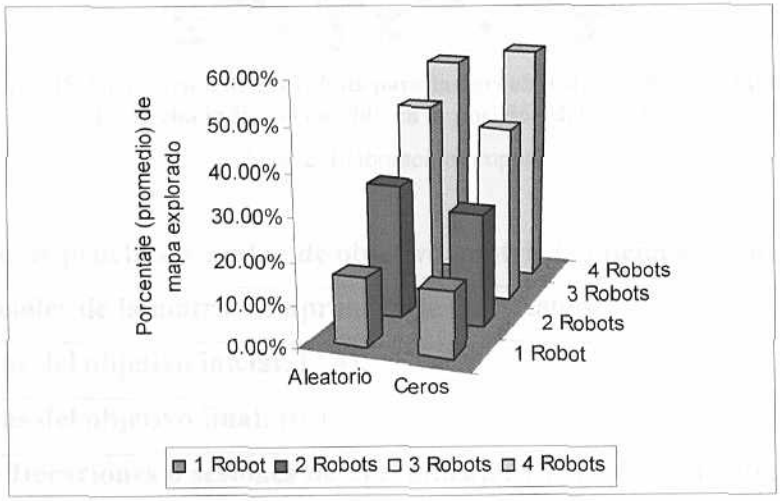
c) Porcentaje (%) de mapa explorado: En esta métrica se observan variaciones similares respecto con el número de obstáculos detectados: a media que aumenta el número de robots en el sistema, disminuye el porcentaje de mapa explorado para el robot que llega al objetivo (ver **Figura 23**), pero el total de los robots aumenta (ver **Figura 24**). Se pudiera decir que el robot que llega al objetivo encuentra una tendencia exploratoria cuando aumenta el número de robots, para los dos modelos.



	1 Robot	2 Robots	3 Robots	4 Robots
Aleatorio	15,92%	15,77%	15,38%	15,38%
Ceros	15,38%	15,38%	15,38%	15,38%

Figura 23. Comparación de valores para % (promedio) de mapa explorado. Robot que llega al objetivo.

Fuente: Elaboración propia.



	1 Robot	2 Robots	3 Robots	4 Robots
Aleatorio	15,92%	31,58%	46,23%	53,96%
Ceros	15,38%	26,88%	42,19%	57,54%

Figura 24. Comparación de valores para % (promedio) de mapa explorado. Suma de todos los robots.

Fuente: Elaboración propia.

La tendencia de exploración global se mantiene uniforme para los dos modelos, sin embargo, el nivel de exploración para el modelo con ceros es ligeramente menor.

2.2) PRUEBAS CON CAMBIO DE OBJETIVO.

En estas pruebas (3° y 4° conjunto de pruebas) se procedió a evaluar el desempeño del sistema si se modifica la posición del objetivo. Esto se hace para evaluar la adaptabilidad del sistema a cambios de objetivo. Las pruebas se realizaron individualmente con 1, 2 y 3 robots. En la **figura 20** se muestra la disposición de los robots en cada una de las pruebas.

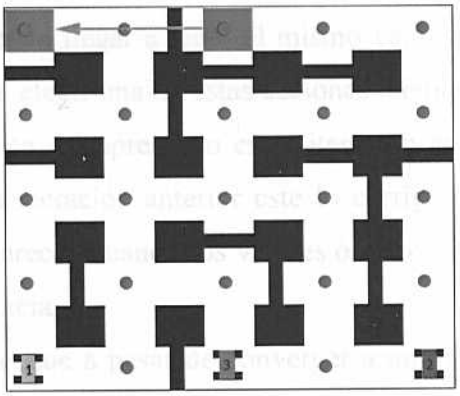


Figura 25. Disposición de los robots para las pruebas de cambio de objetivo. La flecha indica el cambio en la posición del objetivo.

Fuente: Elaboración propia.

3° Conjunto de pruebas. Cambio de objetivo, matriz Q inicial aleatoria

Valores iniciales de la matriz de aprendizaje Q: Aleatorio.
Coordenadas del objetivo inicial: (2,4).
Coordenadas del objetivo final: (0,4).
Numero de Iteraciones o sesiones de aprendizaje: 100 (50 con objetivo inicial y 50 con objetivo final)
Número de pasos (promedio) óptimo hasta el objetivo inicial: 11 pasos.
Número de pasos (promedio) óptimo hasta el objetivo final: 8 pasos.
Robots utilizados y coordenadas de posición: **1 Robot:** Robot 1 (0,0)

2 Robots: Robot 1 (0,0), Robot 2 (4,0)
3 Robots: Robot 1 (0,0), Robot2 (4,0), Robot 3 (2,0)

Los resultados para las métricas seleccionadas, obtenidos en estas pruebas fueron los siguientes (ver Apéndice D: **Figuras D-1a hasta D-1e**):

Un aspecto importante que se puede apreciar en estos resultados es la aparición de oscilaciones en los valores iniciales (especialmente para las pruebas de mas de 1 robot) antes de converger. El análisis de los ***logs*** arrojó que, debido a los castigos o premios que se realicen al agente en las iteraciones iniciales, dos acciones en un mismo estado pueden llegar a tener el mismo valor. Como el robot no puede distinguir entre ambas, al elegir una de estas acciones (siempre es la primera) lleva al robot a un camino distinto del aprendido en la iteración anterior. Si este camino es menos óptimo que en la iteración anterior este lo corrige en la iteración siguiente. Estas oscilaciones desaparecen cuando los valores óptimos se refuerzan de una forma tal que se da la convergencia.

Se puede apreciar que a pesar de converger a un mismo valor con el primer objetivo, convergen con diferentes valores cuando cambia el objetivo. Este punto es clave para el análisis del desempeño del sistema con el aumento de los robots. Esto se debe a que a partir de 2 robots los valores aprendidos se encuentran más cerca del valor óptimo que con 1 robot.

4° Conjunto de pruebas. Cambio de objetivo, matriz Q inicial con ceros.

Valores iniciales de la matriz de aprendizaje Q: Ceros.
Coordenadas del objetivo inicial: (2,4).
Coordenadas del objetivo final: (0,4).
Numero de Iteraciones o sesiones de aprendizaje: 100 (50 con objetivo inicial y 50 con objetivo final)
Número de pasos (promedio) óptimo hasta el objetivo inicial: 11 pasos.
Número de pasos (promedio) óptimo hasta el objetivo final: 8 pasos.

Robots utilizados y coordenadas de posición:

1 Robot: Robot 1 (0,0)

2 Robots: Robot 1 (0,0), Robot 2 (4,0)

3 Robots: Robot 1 (0,0), Robot2 (4,0), Robot 3 (2,0)

Los resultados para las métricas seleccionadas, obtenidos en estas pruebas fueron los siguientes (ver Apéndice B: **figuras D-2a hasta D-2e**):

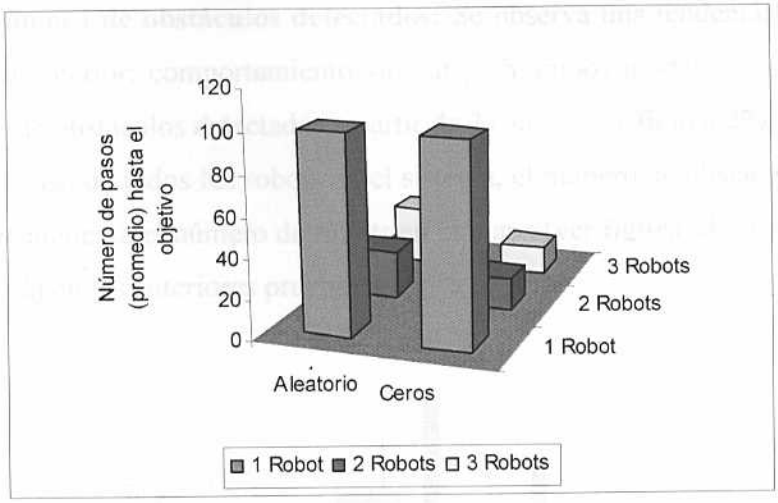
En las métricas obtenidas tanto con la matriz Q inicial en ceros, como la matriz Q inicial en valores aleatorios, se observo que aun cuando se alcanza el valor óptimo con el objetivo inicial, en la mayoría de los casos no llega a su valor óptimo con el cambio de objetivo, aunque los valores se llegan a estabilizar después de varias iteraciones.

Un aspecto importante con esta prueba es que se observa una notable disminución en las oscilaciones iniciales realizadas antes de converger; así mismo se aprecia una disminución leve en los valores de convergencia con respecto al modelo de matriz Q inicial con valores aleatorios.

2.2.1) ANÁLISIS COMPARATIVO DE LOS RESULTADOS OBTENIDOS EN *LA PRUEBAS DE CAMBIO DE OBJETIVO.*

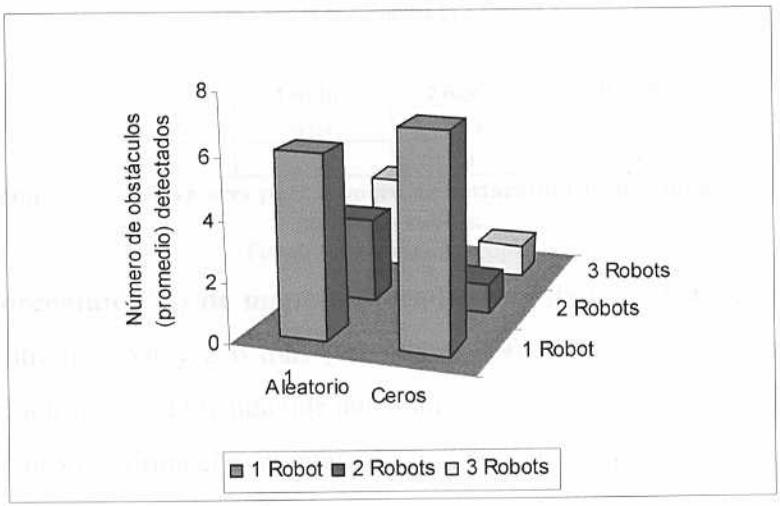
Para la realización de esta comparación se recurre de la misma forma que en las pruebas anteriores al análisis de los valores promedio de las métricas, específicamente para las iteraciones en las que es cambiado el objetivo. De esta forma se puede apreciar como es el comportamiento al adaptarse al cambio de objetivo.

a) Número de pasos hasta el objetivo (ver Figura 26): Se presenta una tendencia muy similar en los modelos con matriz Q inicial con ceros y matriz inicial Q con valores aleatorios. Por el contrario, se observa una disminución drástica del número de pasos de 2 robots en adelante, para ambos casos.



	1 Robot	2 Robots	3 Robots
Aleatorio	99.04	23.76	29.14
Ceros	100.14	16	14.24

Figura 26. Comparación de valores para número de pasos (promedio) al objetivo. Robot que llega al objetivo.
Fuente: Elaboración propia.

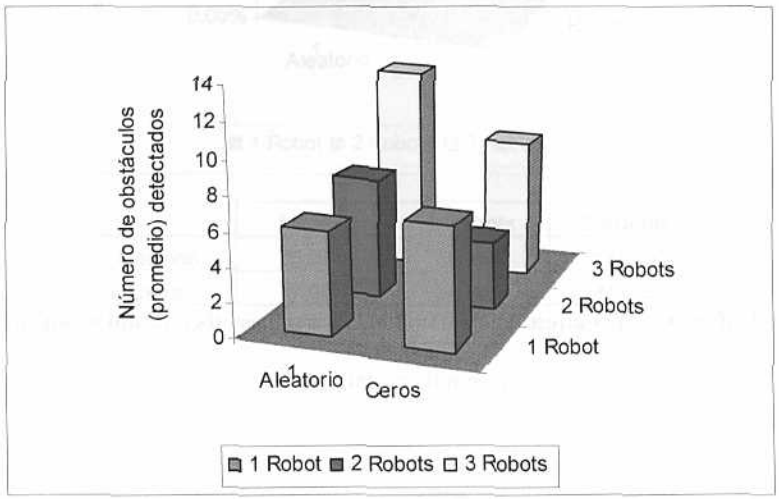


	1 Robot	2 Robots	3 Robots
Aleatorio	6.04	2.8	3.18
Ceros	7.06	1	1.12

Figura 27. Comparación de valores para número de obstáculos (promedio) detectados. Robot que llega al objetivo.
Fuente: Elaboración propia.

b) Número de obstáculos detectados: Se observa una tendencia similar que en la métrica anterior; comportamiento similar para ambos modelos, y disminución considerable de obstáculos detectados a partir de 2 robots (ver **figura 27**).

En el caso de todos los robots en el sistema, el número de obstáculos aumenta a medida que aumenta el número de robots en el mapa (ver **figura 28**). Esta tendencia ya fue reseñada en las anteriores pruebas.

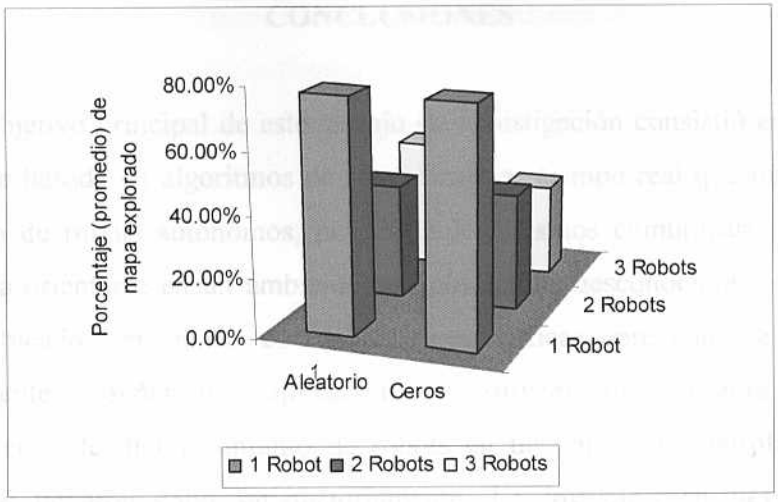


	1 Robot	2 Robots	3 Robots
Aleatorio	6.04	7.1	12.12
Ceros	7.06	4	8.24

Figura 28. Comparación de valores para número de obstáculos (promedio) detectados. Suma de todos los robots.
Fuente: Elaboración propia.

c) Porcentaje (%) de mapa explorado: En este caso disminuye un poco la diferencia entre 1 robot y 2 o más (ver **figura 29**). El robot que llega al objetivo tiende a explorar menos a medida que aumenta el número de robots en el sistema.

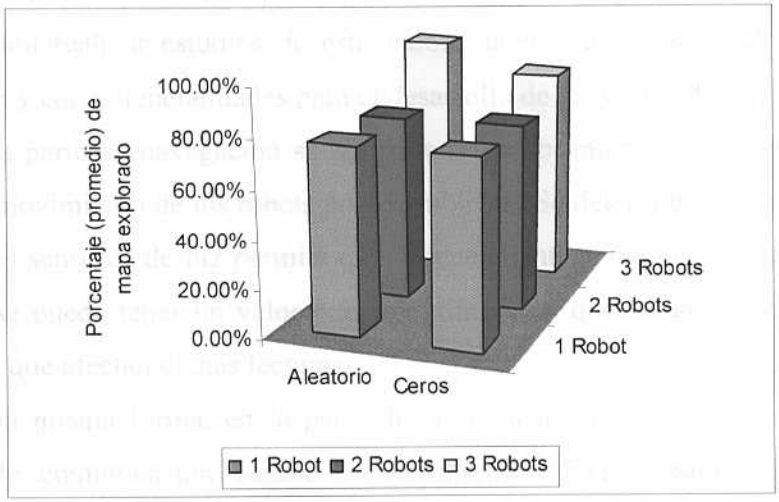
De la misma forma el porcentaje de exploración aumenta para todos los robots que se encuentran en el mapa. Sin embargo, estos valores se encuentran cercanos al 100% del mapa (en efecto, con 4 robots se llega a explorar el 100% del mapa) (ver **figura 30**).



	1 Robot	2 Robots	3 Robots
Aleatorio	76.92%	38.38%	43.23%
Ceros	77.08%	38.46%	31.00%

Figura 29. Comparación de valores para % (promedio) de mapa explorado. Robot que llega al objetivo.

Fuente: Elaboración propia.



	1 Robot	2 Robots	3 Robots
Aleatorio	76.92%	76.85%	99.92%
Ceros	77.08%	76.92%	88.62%

Figura 30. Comparación de valores para % (promedio) de mapa explorado. Suma de todos los robots.

Fuente: Elaboración propia.

CONCLUSIONES

El objetivo principal de este trabajo de investigación consistió en desarrollar un programa basado en algoritmos de aprendizaje en tiempo real que implantado en un conjunto de robots autónomos, permita a los mismos comunicarse y colaborar entre sí para orientarse en un ambiente con obstáculos desconocidos y localizar un objetivo ubicado en unas coordenadas específicas previamente conocidas. Adicionalmente, diseñar una aplicación de software que permita simular el comportamiento de dicho conjunto de robots en un ambiente controlado. Ambos objetivos se llevaron cabo satisfactoriamente. La investigación llevada a cabo permitió obtener información de gran utilidad, tanto para la utilización en otras áreas, como para profundización en trabajos posteriores.

El desarrollo de los robots con el kit de LEGO Mindstorms permitió concentrarse en aspectos mas relacionados con el aprendizaje de los agentes que con la construcción del robot en si, por esta razón es uno de los modelos robóticos mas utilizados para realizar estudios de esta índole, debido a su facilidad de diseño y construcción y con potencialidades para el desarrollo de proyectos de esta índole.

En la parte de navegación se realizaron descubrimientos relacionados con el control del movimiento de los robots por el ambiente. Se determinó que la calibración inicial de los sensores de luz permite que independientemente de los factores de luz ambiental, se pueda tener un valor más aproximado, y que se adapte a los factores ambientales que afectan dichas lecturas.

De la misma forma, en la parte de comunicación se logró implementar un protocolo de comunicación basado en la librería ***RCXLNPAddressingPort***, que maneja de una forma eficiente las limitaciones del robot en cuanto al modo de transmisión de datos. Un punto importante de discusión en esta actividad es analizar como esto afectaría la autonomía del robot. La conclusión a la que se llegó es que el robot efectivamente tiene autonomía, sólo que por motivos de espacio de almacenamiento, las funciones que ejercen dicha autonomía residen del lado del servidor, por esta razón es que se realiza la simulación de los agentes de software en

el servidor. Estos agentes representan el motor de decisión de cada robot, y el agente físico complementa la decisión realizando la acción.

Para evaluar el sistema definitivo, la utilización de la simulación fue de suma importancia. Además de ayudar a comprender elementos de diseño y desarrollo del sistema que no serían perfectamente observables en el entorno físico, permitió hacer un análisis de sensibilidad del sistema, consumiendo menos recursos.

En las pruebas con la simulación se pudo determinar la importancia de los siguientes elementos en el aprendizaje:

- **Una correcta selección de los estados:** Los estados a seleccionar deben representar la percepción actual del agente en un instante determinado. Por esta razón, las coordenadas del robot y la posición absoluta del mismo permitieron decir los estados que se utilizarían en el modelo final del sistema.
- **Una política de selección de acciones adaptada al entorno que se desea desarrollar:** existen casos en los que no se puede seleccionar una acción simplemente por una política fija. En el caso de esta investigación, la política diseñada (*m-greedy*) permite adaptarse al entorno multiagente, ya que la verificación previa de las acciones, permite minimizar las colisiones, y en lo posible, el choque con obstáculos. El *backtracking* en la selección de acciones, permite coordinar las acciones del robot, haciendo que independientemente de las condiciones del ambiente, exista un robot que encuentre la salida del laberinto, aunque no sea en el número optimo de pasos.
- **Una función refuerzo coherente con el objetivo que se desea alcanzar:** Este elemento garantiza la convergencia hacia el valor óptimo. Aunque la función refuerzo no se desarrolló sobre acciones conjuntas, junto con la política de selección de acciones permite que uno de los robots encuentre efectivamente el objetivo.

Los resultados arrojaron que la exploración usando varios agentes es efectiva, aun cuando no todos pueden llegar al objetivo, estos permiten que el robot que llegue

al objetivo lo realice con una disminución considerable de pasos, obstáculos a detectar y exploración del mapa.

Si bien no existe hasta el momento una respuesta universal sobre la mejor forma de explorar un ambiente, queda demostrado que con una bien planificada política de selección de acciones y una buen manejo de los refuerzos, esto aunado a la cantidad de robots, se pueden realizar dos cosas que un solo robot no puede realizar: explorar casi en su totalidad el ambiente y encontrar el camino optimo en el menor número de pasos. Ayuda a resolver la complejidad y el dinamismo propio de los sistemas multiagente, facilitando la interacción entre los agentes, bien sea de colaboración o negociación.

Sin embargo hay que destacar que siempre recuenta con el factor de "anarquía" cuando se trabaja en el entorno multiagente: a pesar de existir colaboración en el entorno multiagente hay acciones que pueden resultar beneficiosas para un agente y al mismo tiempo generar complicaciones en otro agente. Esto se traduce en una creciente detección de obstáculos a medida que aumenta el número de robots, ya que los robots que no pueden conseguir el objetivo tratan de explorar rutas alternas para alcanzarlo.

En el análisis de los resultados, la modificación del algoritmo de aprendizaje, inicializado la matriz de aprendizaje Q con ceros, demostró su efectividad en el entono multiagente, por tres razones fundamentales.

- Para varias corridas del algoritmo bajo las mismas condiciones se pueden obtener los mismos resultados
- Los valores aprendidos convergen a su valor óptimo en un menor número de pasos y desde valores más pequeños.
- Demuestra su adaptabilidad ante los cambios de objetivo.

Se observa que en el modelo de aprendizaje por refuerzo, el objetivo es, como su nombre lo indica "reforzar" un conjunto de acciones, que lleven al agente a un estado objetivo. Los resultados de las pruebas arrojaron que, aun cuando esa secuencia de acciones no sea la óptima, tratara de optimizarla hasta un máximo

posible y luego la reforzará, permitiendo que el algoritmo converja a esta política de selección de acciones.

Como conclusión final, es posible desarrollar un sistema multiagente que pueda aprender en tiempo real; en su diseño intervienen numerosos detalles (mencionados anteriormente), todos de suma importancia para lograr el producto final.

RECOMENDACIONES

Como resultado de este trabajo de investigación y con el objetivo de contribuir a futuras instigaciones se realizan las siguientes recomendaciones:

- El método de transmisión utilizando IR (infrarrojo) tal como lo presenta el RCX de LEGO® MindStorms® es muy limitada. Entre varios agentes resulta sumamente compleja y difícil de controlar. Se recomienda la adquisición de kits de robótica, por parte del Grupo de Inteligencia Artificial (GIA) de la UCAB, que permitan la utilización de otros medios de comunicación. Por ejemplo, en Julio de 2006 fue lanzado al mercado la nueva generación de LEGO® MindStorms®, llamada LEGO® NXT, cuyo kit de diseño permite comunicaciones utilizando *bluetooth*.
- En las investigaciones realizadas, se trabajó con una cantidad limitada de sensores; esta situación trajo como consecuencia que los robots no tuvieran una percepción más completa del ambiente que les rodea, y de esta forma poder tener una mayor cantidad de estados y así poder enriquecer el aprendizaje del mismo. Por esta razón, seria deseable dotar al laboratorio del Grupo de Robótica con dispositivos como sensores de proximidad, que permitirían al robot detectar los obstáculos antes de colisionar con ellos. De la misma forma, los sensores de rotación permitirían al robot realizar giros con mayor precisión, con la posibilidad de inducir menos errores en el aprendizaje de las acciones que maximicen su meta. Estos sensores vienen incluidos por defecto en el nuevo kit de LEGO NXT.
- Se podría desarrollar para trabajos futuros, modificaciones en la librería ***RCXLNPAddressingPort***, que permitiesen la comunicación *full duplex* entre los RCX's y la torre IR, y permitiendo el *broadcasting* de la información, lo cual es de gran utilidad cuando se desea que una información sea compartida simultáneamente por todos los robots que participen en el experimento.

- Seria deseable para trabajos futuros utilizar aprendizaje distribuido, utilizando robots con mayor capacidad de cálculo y almacenamiento, de forma que se puedan implementar en el mismo robot simultáneamente las rutinas de aprendizaje, comunicación y navegación del robot, sin afectar el rendimiento de las operaciones antes mencionadas.
- La política de selección de acciones es un aspecto muy importante para el aprendizaje por refuerzo. Sin embargo es muy complicado diseñar una política cuando se trata de un sistema multiagente. Se han formulado modelos de políticas de selección de acciones basadas en la teoría del *equilibrium* de John Nash; dichos modelos se enfocan en políticas que buscan el bienestar del grupo, en vez de políticas más orientadas hacia la competencia entre los robots. Se pudieran desarrollar técnicas para adaptar este tipo de algoritmos al entorno de cooperación entre los robots.
- Realizar sistemas multiagente que fueran un híbrido entre el aprendizaje en tiempo real o no supervisado y el supervisado.

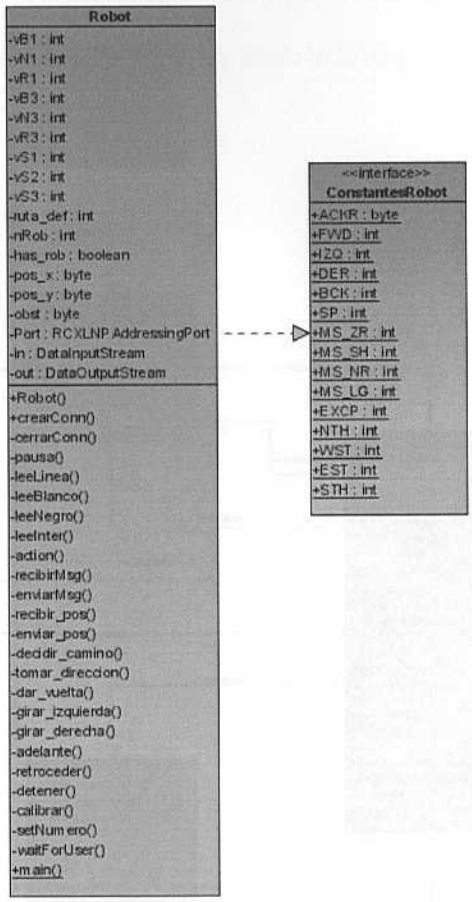
BIBLIOGRAFÍA

- Bagnall, B. (2002). *CORE Lego Mindstorms™ Programming*. Nueva York, NY: Prentice Hall.
- Bourg, D. (2004). *AI For Game Developers*. EEUU: O'Reilly.
- Buckland, M. (2005). *Programming Game AI by Example*. EEUU: Wordware Press.
- Ferrari, M., Ferrari y G. Hempel, R. (2002). *Building Robots with LEGO Mindstorms*. EEUU: Syngress.
- Ferrari, G., Gombos, A., Hilmer, S., Stuber, J., Porter, M., Waldinger, J. y Laverde, D. (2002). *Programming LEGO MindstormsTM with Java*. EEUU: Syngress.
- Pressman, Roger S. (1998). *Ingeniería del Software un enfoque práctico* (4a ed.). Madrid, España: McGraw Hill.
- Russel, S. y Norvig, P. (1996). *Inteligencia Artificial Un enfoque moderno* (1a ed.). Edo. De México: Prentice Hall.
- Sutton, R. y Barto, A. (1998). *Reinforcement Learning: An Introduction*. Massachusetts, EEUU: MIT Press.
- Bowling, M (2003). *Multiagent Learning in the Presence of Agents with Limitations* [documento en linea] Recuperado en 23/6/2006 de http://www.cs.cmu.edu/~mhb/papers/thesis.pdf
- Bowling, M. y Veloso, M. (2002). *Existence of Multiagent Equilibria with Limited Agents*. [documento en linea] Recuperado en 23/6/2006 de http://www.cs.cmu.edu/~mmv/papers/02TR-equilibria.pdf
- Conitzer, V. y Sandhlom, T. (2004). *Communication complexity as a lower bound for learning in games*. [documento en linea] Recuperado en 23/6/2006 de http://www.cs.cmu.edu/~conitzer/communicationICML04.pdf
- Draa, C (2001). *Agents et Systèmes Multiagents:Thèmes, Approches et Défis*. [documento en linea] Recuperado en 23/6/2006 de http://www.iro.umontreal.ca/~kropf/ift-6052/notes/agents-slides-bw.pdf
- Dresner, K. y Stone, P. (2005). *Multiagent Traffic Management: An Improved Intersection Control Mechanism*. [documento en línea] Recuperado en 23/6/2006 de http://www.cs.utexas.edu/~kdresner/pubs/files/2005aamas-dresner.pdf
- Errecalde, M., Crespo, M. y Montoya, C. (1999). *Aprendizaje por refuerzo: Un estudio de sus principales métodos*. En *II Encuentro Nacional de Computación (ENC'99)*. México: Sociedad Mexicana de Ciencia de la Computación.
- Errecalde, M. y Aguirre, G. (1999). *Una propuesta para integrar agentes de interfaz, aprendizaje y sistemas multi agentes*. En *II Encuentro Nacional de Computación (ENC'99)*. México: Sociedad Mexicana de Ciencia de la Computación.

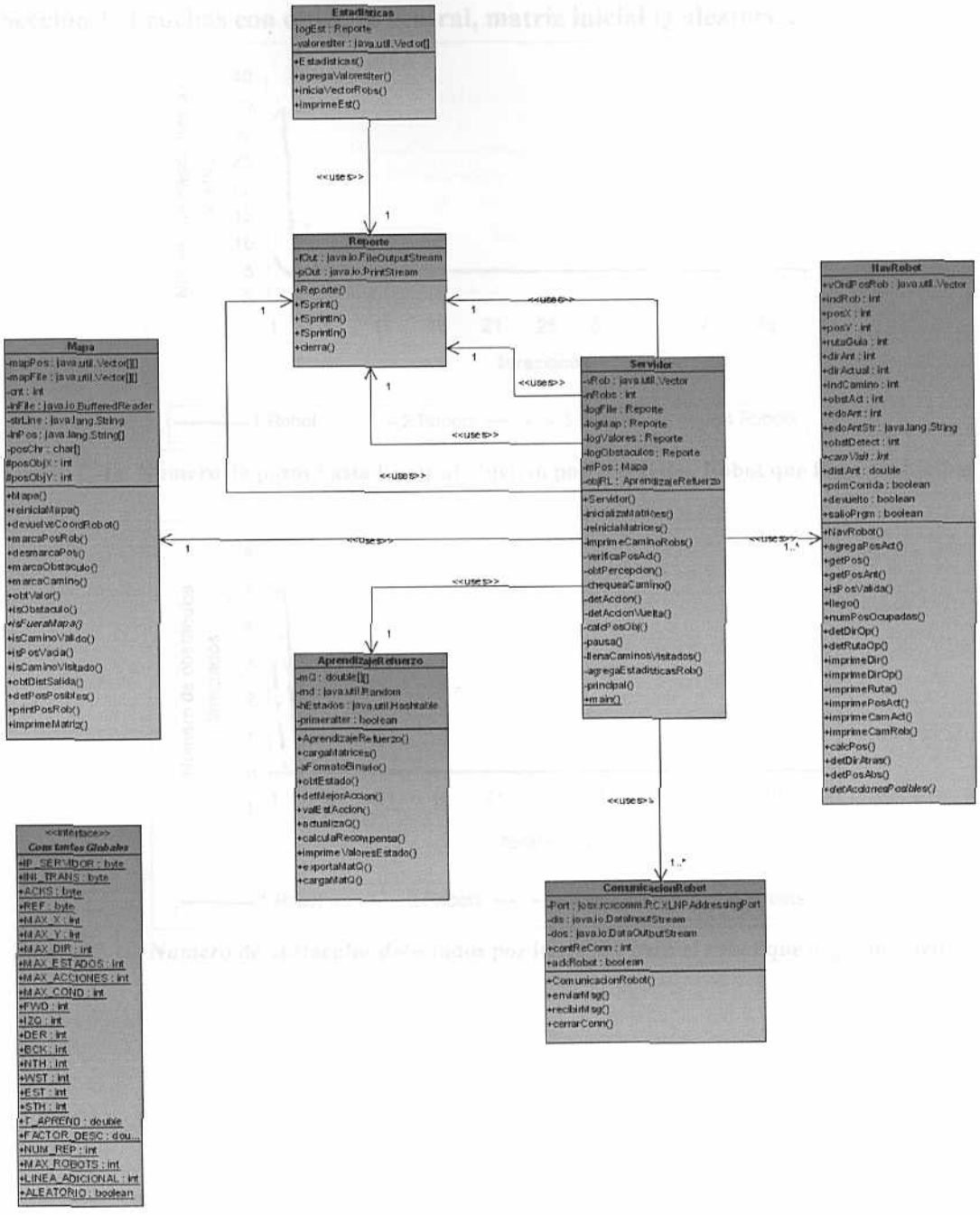
- Garcia Serrano y A., Ossowski, S. (1998). Inteligencia Artificial y distribuida en sistemas Multiagentes. *Revista Iberoamericana de Inteligencia Artificial. 6,2.*
- Greenwald, A. y Hall, K. (s.f.). *Correlated Q.Learning*. [documento en linea] Recuperado en 23/6/2006 de http://www.hpl.hp.com/conferences/icml2003/papers/231.pdf
- Hailu, G. y Sommer, G. (1999). On Amount and Quality of Bias in Reinforcement Learning. [documento en linea] Recuperado en 23/6/2006 de http://www.ks.informatik.uni-kiel.de/~vision/doc/Publications/gha/smc99.pdf
- Mance E. Harmon y Stephanie S. Harmon (s.f.). *Reinforcement Learning: A Tutorial*. [documento en línea] Recuperado en 23/6/2006 de http://www.nada.kth.se/kurser/kth/2D1432/2003/rltutorial.pdf.
- Könönen, V. (2004). *Hybrid Model for Multiagent Reinforcement Learning.* [documento en linea] Recuperado en 23/6/2006 de http://www.cis.hut.fi/kononen/publications/hybrid.pdf
- Paukku, T. (2005). *Using leJOS communication for games*. [documento en línea] Recuperado en 23/6/2006 de http://legolab.daimi.au.dk/Publications/pdf/lejos/UsingleJOSCommunicationForGames.pdf
- Sandholm, T. y Crites, R. (s.f.). Multiagent Reinforcement Learning in the Iterated Prisoner's Dilemma. [documento en línea] Recuperado de 23/6/2006 de http://www.cs.wustl.edu/~mas/Publications/sandholm-biosystems95.ps
- Shoham, Y., Powers, R. y Grenager, T. (2006). *If multi-agent learning is the answer, what is the question?* [documento en linea] Recuperado de 23/6/2006 de http://ai.stanford.edu/~shoham/www%20papers/LearningInMAS.pdf
- Stone, P. y Veloso, M. (2000). *Multiagent Systems: A Survey from a Machine Learning Perspective.* [documento en línea] Recuperado de 23/6/2006 de http://www.cs.cmu.edu/~mmv/papers/MASsurvey.pdf
- Tan, M. (s.f.). *Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents*. [documento en línea] Recuperado de 23/6/2006 De http://web.media.mit.edu/~cynthiab/Readings/tan-MAS-reinfLearn.pdf
- Vlassis, N. (2003). *A Concise Introduction to Multiagent Systems and Distributed AI*. [documento en línea] Recuperado de 23/6/2006 De http://www.science.uva.nl/~vlassis/cimasdai/cimasdai.pdf
- Walkman, L. y Kaymak, U. (2006). A Theoretical Analysis of Cooperative Behavior in Multi-Agent Q-learning. [documento en línea] Recuperado de 23/6/2006 De https://ep.eur.nl/bitstream/1765/7323/1/ERS+2006+006+LIS.pdf
- Zhu, S. (2003). *Learning to Cooperate*. [documento en linea] Recuperado de 23/6/2006 De http://www.cs.rochester.edu/~gildea/2005_Spring/**zhu**-dissertation.pdf

- Carrasquero, O. y Filaccione, E. (2002). *Diseño y construcción de un robot con el módulo RCX 1.0 para ambientes no predeterminados* (Tesis de Ingeniero en Informática, Universidad Católica Andrés Bello).
- Dos Santos, D. y Peñalver, R. (2003). *Sistema autónomo de navegación con reconocimiento de patrones geométricos regulares* (Tesis de Ingeniero en Informática, Universidad Católica Andrés Bello).
- Bagnall, Bryan (2001). *Lejos: Java for the RCX*. [página web en línea] Recuperado de 23/6/2006 De http://lejos.sourceforge.net/
- Eden, T., Knittel, A. y van Uffelen, R. (s.f). *Reinforcement Learning*. [página web en línea] Recuperado de 23/6/2006 De http://www.cse.unsw.edu.au/~cs9417ml/RL1/index.html
- Perez Uribe, A. (s.f.) *Introduction to Reinforcement learning*. [página web en línea] Recuperado de 23/6/2006 De http://lslwww.epfl.ch/~anperez/RL/RL.html

APENDICE A. DIAGRAMA DE CLASES. APLICACIÓN ROBOT



APENDICE B. DIAGRAMA DE CLASES. APLICACIÓN SERVIDOR



APENDICE C. PRUEBAS CON OBJETIVO CENTRAL. GRAFICAS OBTENIDAS

Sección 1: Pruebas con objetivo central, matriz inicial Q aleatoria.

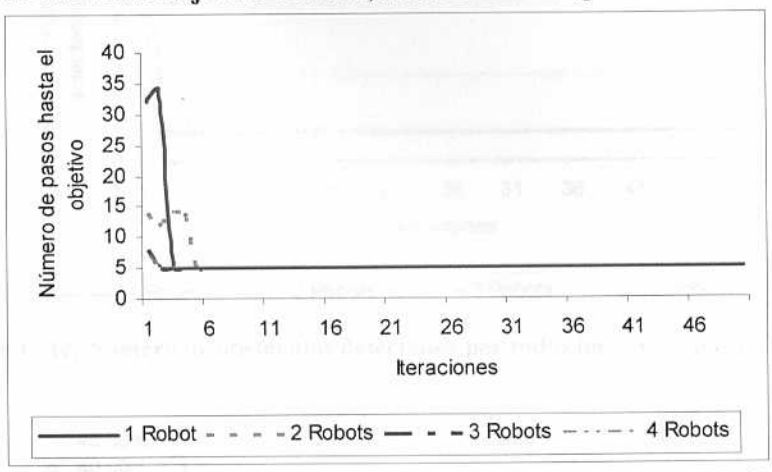


Figura C-1a. Numero de pasos hasta llegar al objetivo por iteración. Robot que llega al objetivo.

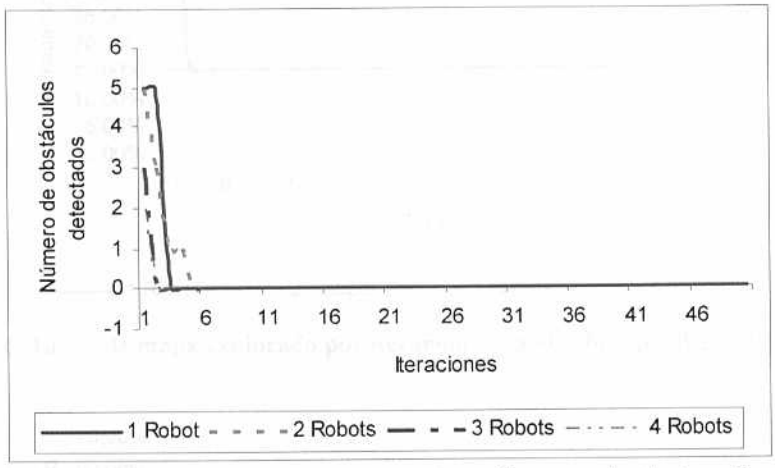


Figura C-1b. Numero de obstáculos detectados por iteración para el robot que llega al objetivo.

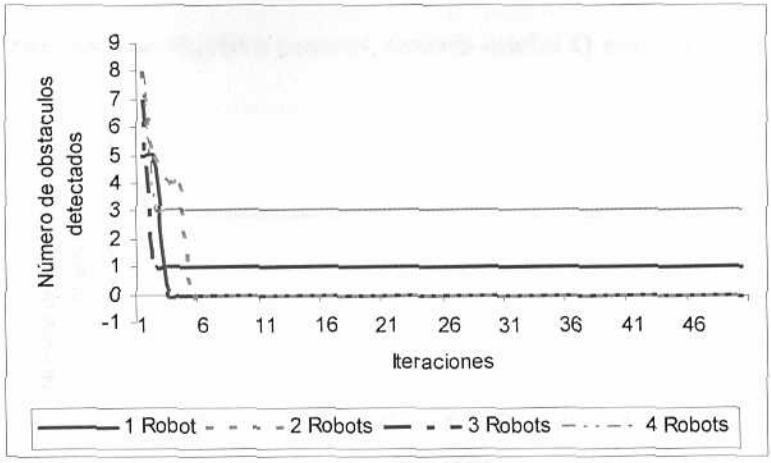


Figura C-1c. Numero de obstáculos detectados por todos los robots por iteración.

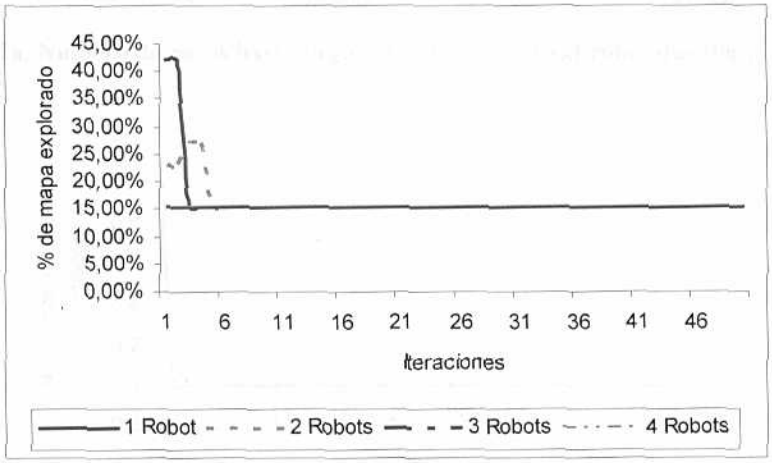


Figura C-1d. % de mapa explorado por iteración para el robot que llega al objetivo.

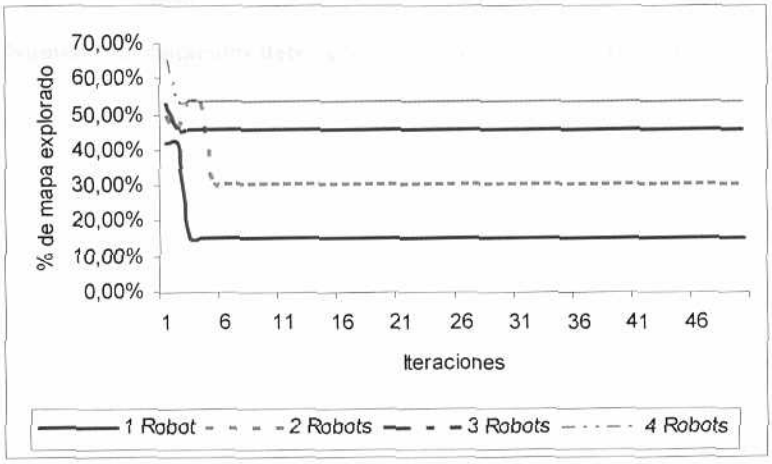


Figura C-1e. % de mapa explorado por todos los robots por iteración.

Sección 2: Pruebas con objetivo central, matriz inicial Q con ceros.

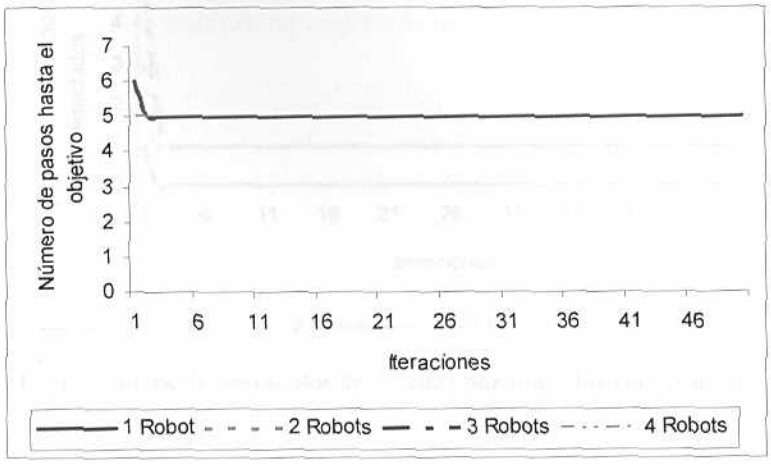


Figura C-2a. Numero de pasos hasta llegar al objetivo, para el robot que llega al objetivo.

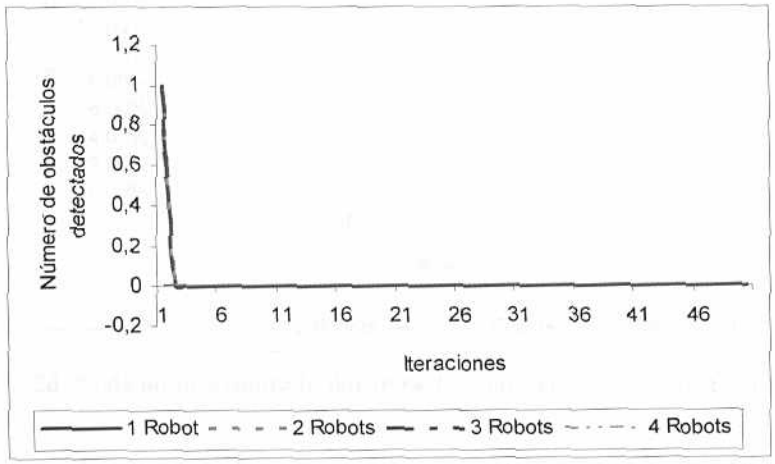


Figura C-2b. Numero de obstáculos detectados por iteración para el robot que llega al objetivo.

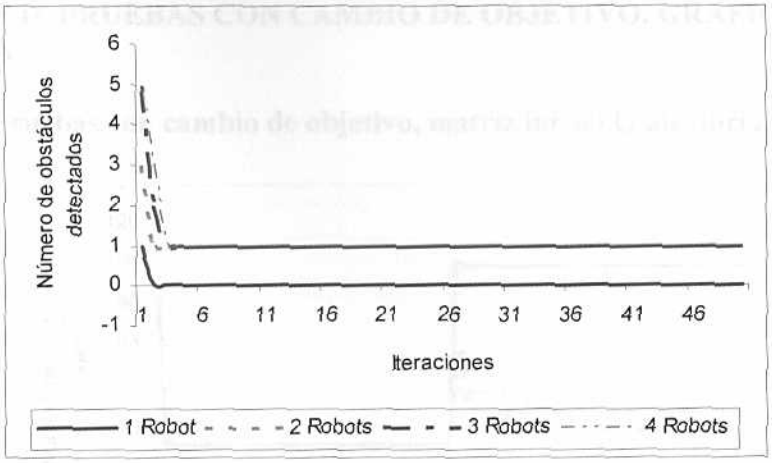


Figura C-2c. Numero de obstáculos detectados por todos los robots por iteración.

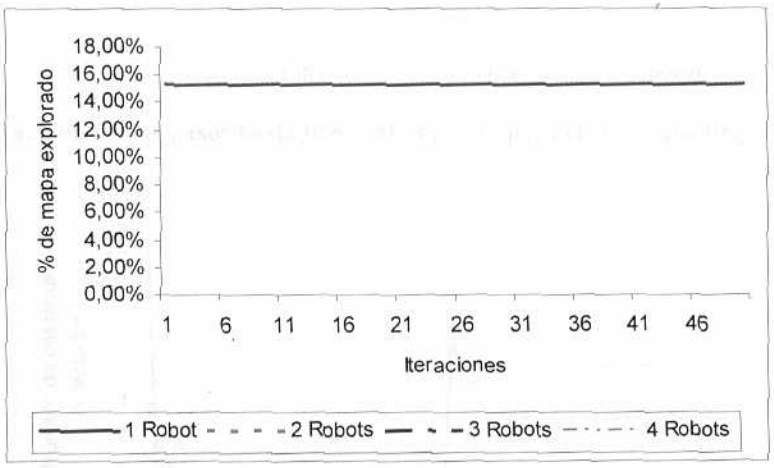


Figura C-2d. % de mapa explorado por iteración para el robot que llega al objetivo.

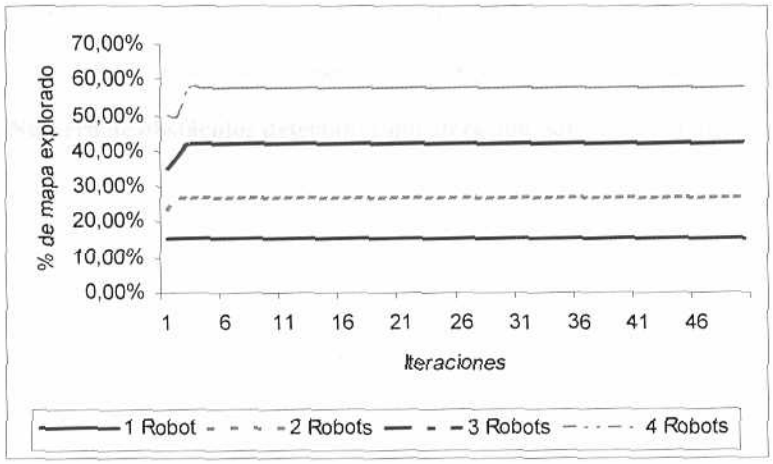


Figura C-2e. % de mapa explorado por todos los robots por iteración.

APENDICE D. PRUEBAS CON CAMBIO DE OBJETIVO. GRAFICAS OBTENIDAS

Sección 1: Pruebas con cambio de objetivo, matriz inicial Q aleatoria.

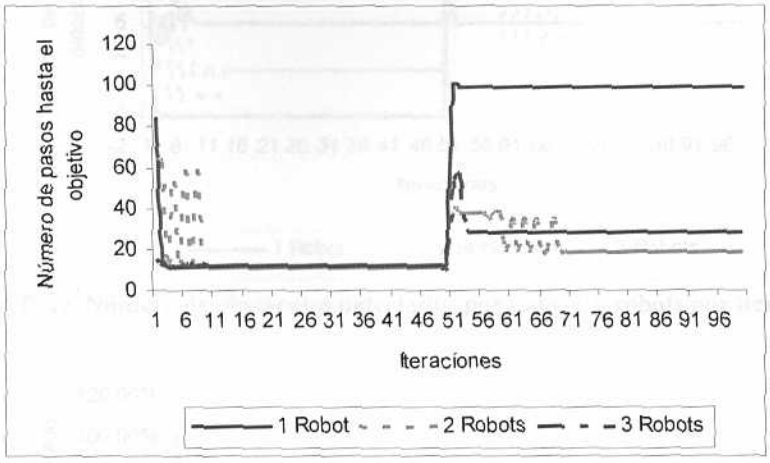


Figura D-1a. Numero de pasos hasta llegar al objetivo, para el robot que llega al objetivo.

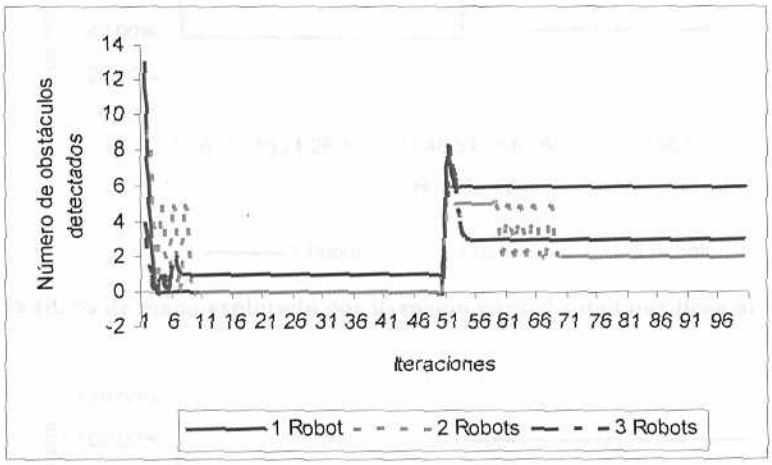


Figura D-1b. Numero de obstáculos detectados por iteración para el robot que llega al objetivo.

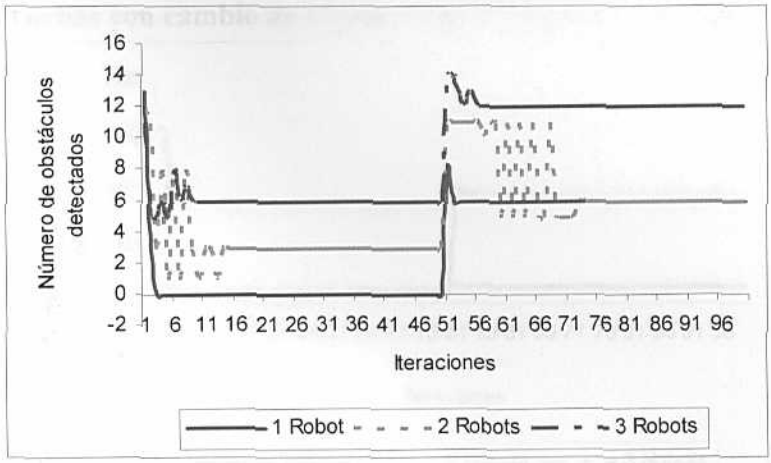


Figura D-1c. Numero de obstáculos detectados por todos los robots por iteración.

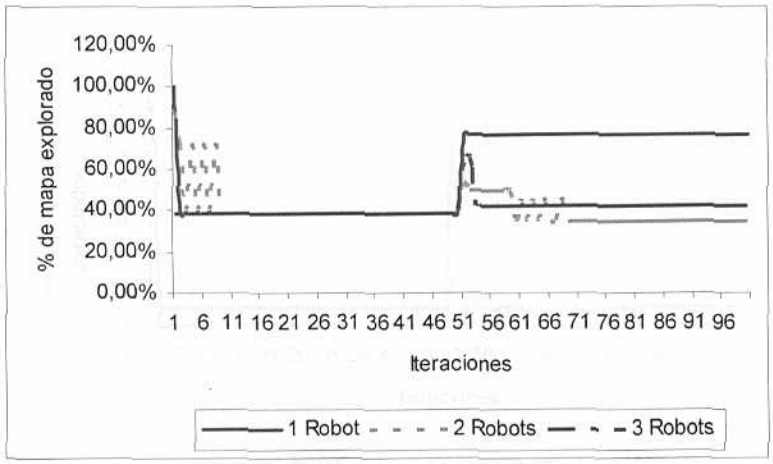


Figura D-1d. % de mapa explorado por iteración para el robot que llega al objetivo.

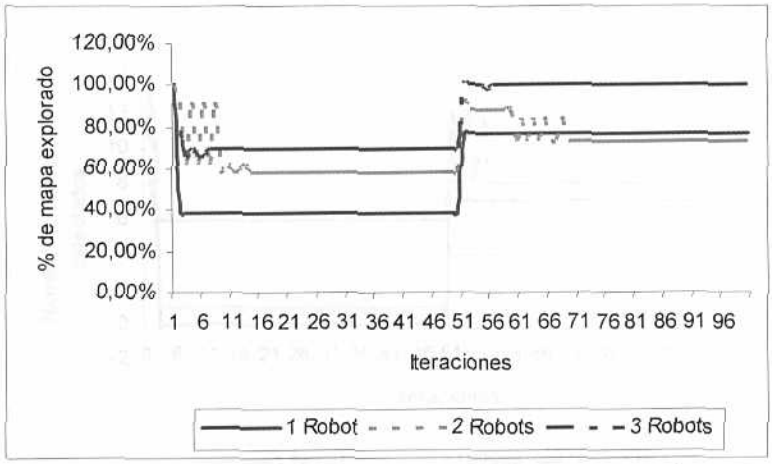


Figura D-1e. % de mapa explorado por todos los robots por iteración.

Sección 2: Pruebas con cambio de objetivo, matriz inicial Q con ceros.

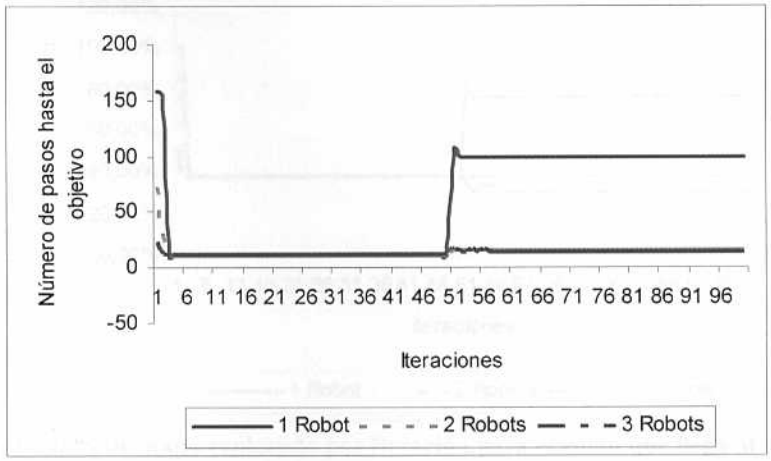


Figura D-2a. Numero de pasos hasta llegar al objetivo, para el robot que llega al objetivo.

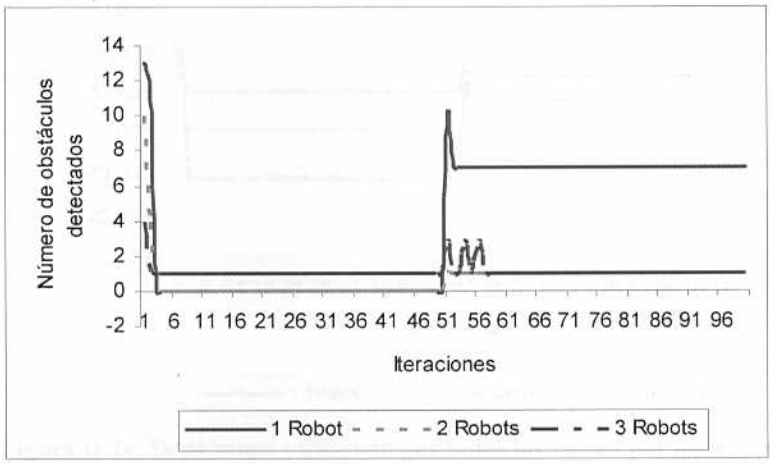


Figura D-2b. Numero de obstáculos detectados por iteración para el robot que llega al objetivo.

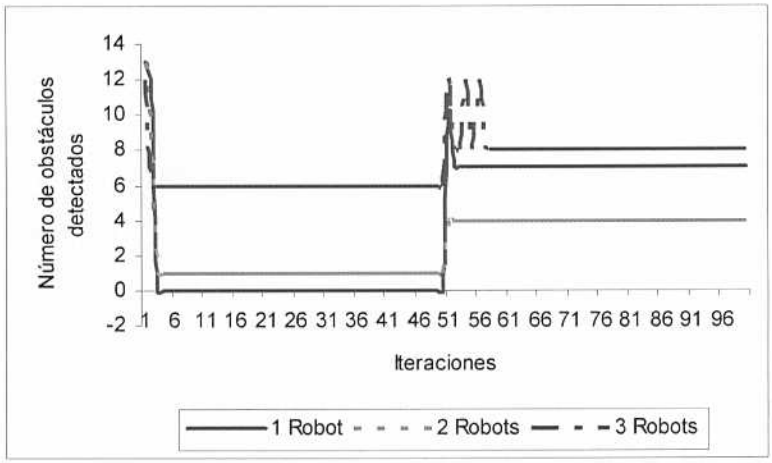


Figura D-2c. Numero de obstáculos detectados por todos los robots por iteración.

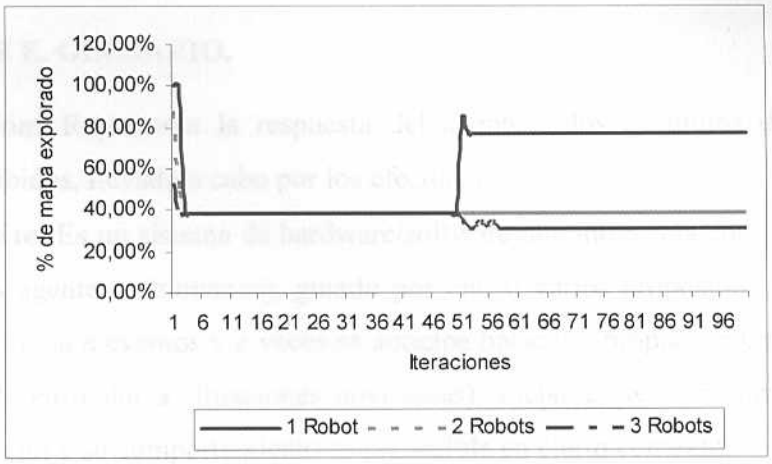


Figura D-2d. % de mapa explorado por iteración para el robot que llega al objetivo.

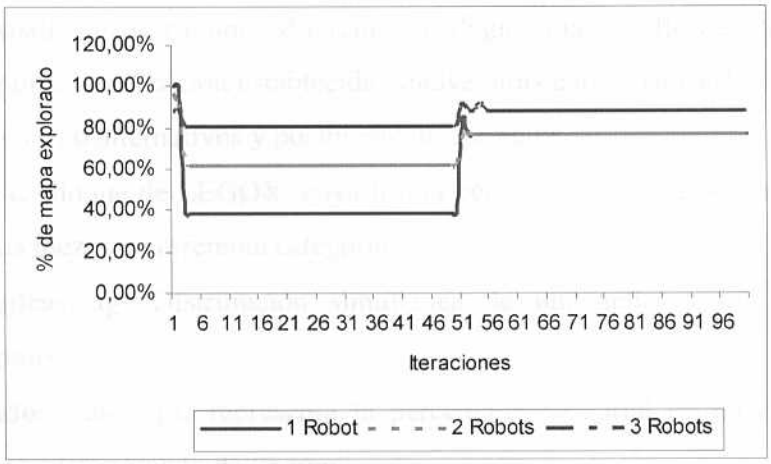


Figura D-2e. % de mapa explorado por todos los robots por iteración.

APÉNDICE E. GLOSARIO.

- **Acción:** Representa la respuesta del agente a los estímulos del ambiente percibidos, llevado a cabo por los efectores.
- **Agente:** *Es un sistema de hardware/software que interactúa con su entorno (u* otros agentes o humanos), guiado por uno o varios propósitos, es proactivo (reacciona a eventos y a veces se anticipa haciendo propuestas), adaptable (se puede enfrentar a situaciones novedosas), sociable (se comunica, coopera o negocia) y su comportamiento es predecible en cierto contexto.
- **Backtracking:** Técnica de programación basada en la recursividad en la que, al existir varios caminos diferentes a elegir, una vez llegado al final y no cumplirse la condición establecida, vuelve atrás para seguir buscando caminos diferentes o alternativos y posiblemente correctos.
- **Brick:** Bloque de LEGO®, cuya forma permite fácilmente adaptarse y unirse a otras piezas de diferente categoría.
- **Broadcasting:** Distribución simultanea de una señal a un conjunto de receptores.
- **Estado:** Valor que representa la percepción sensorial de un agente en un instante determinado de tiempo.
- **Factor de descuento:** Parámetro que reduce la influencia de los refuerzos recibidos en el tiempo.
- **Full Duplex:** Modo de comunicación de un dispositivo en el cual es capaz de enviar y recibir información al mismo tiempo.
- **Grid World:** Modelo de ambiente que se divide en una serie de secciones que contienen una ubicación unívoca.
- **Inteligencia Artificial:** Es una de las áreas de las ciencias computacionales encargadas de la creación de hardware y software que tenga comportamientos inteligentes

- **Log:** Archivo de texto que recoge información de la ejecución de una determinada aplicación.
- **Posicionamiento:** Procedimiento de ubicar a un agente de hardware/software en unas coordenadas especificas dentro de un ambiente.
- Convergencia: Es la propiedad de algunas sucesiones y series de ir progresivamente más cerca de un límite o valor tope. Una función matemática converge a un valor determinado después de un número elevado de iteraciones o repeticiones.
- **Protocolo:** Conjunto de reglas que controlan la secuencia de mensajes que ocurren durante una comunicación entre entidades que forman una red.
- **Refuerzo:** Es *un estímulo que modifica un reflejo condicionado o una* conducta aprendida aumentando su arraigo (refuerzo positivo o premio) o disminuyéndolo (refuerzo negativo o castigo).
- **Robótica:** *Es una ciencia o rama de la tecnología, que estudia el diseño y* construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano o que requieren del uso de inteligencia.
- **Sesión de aprendizaje:** Representa una iteración del experimento, mediante el cual el agente recorre una serie de estados hasta llegar al objetivo.
- **Tasa de Aprendizaje:** Coeficiente que maximiza el valor de las recompensas futuras acumuladas en el tiempo para un determinado agente.
- **Token:** Variable o testigo que circula a través de la red. Cuando una estación o nodo desea transmitir, debe esperar al paso del testigo en condiciones de transportar la información.
- **Valor óptimo:** En el caso de robótica, el valor óptimo es la medida que representa el menor esfuerzo en el robot para alcanzar su objetivo.