UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

EVOLUCIÓN DE LA ORGANIZACIÓN DE LOS COMPORTAMIENTOS DE UN ROBOT USANDO ALGORITMOS GENÉTICOS

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

VICENTE IVÁN SÁNCHEZ CARMONA

DIRECTOR DE TESIS:
DR. JESÚS SAVAGE CARMONA

CO-DIRECTOR DE TESIS:
DR. ÁNGEL FERNANDO KURI MORALES

México, D.F. **2010**

DEDICATORIA

A mi familia, por su ánimo y su apoyo

A mis maestros, por sus conocimientos y su paciencia

A CONACYT, por la beca recibida

ÍNDICE

Capítulo 1

INTRODUCCIÓN

Existen importantes beneficios en usar robots móviles autónomos para ejecutar diversas actividades que sean difíciles o riesgosas para los humanos, desde robots de servicio en casas y oficinas hasta robots de rescate. Imagine, por ejemplo, un robot que pueda investigar la vida marina a miles de metros de profundidad, o un robot que sea capaz de desactivar bombas, o tal vez mejor aún, un robot que pueda llevar a cabo la limpieza de toda la casa, todo esto sin la intervención humana.

Hoy en día se ha dado origen a una *revolución robótica* al empezar con robots que desempeñan tareas fáciles, como aspirar, servir de mascota y jugar con niños, pero si se quiere avanzar tecnológicamente se necesitan robots capaces de navegar en ambientes no estructurados, manipular objetos, capaces de entender, interpretar y representar el ambiente de una manera más *humana*, también tendrá que interactuar y comunicarse con los humanos. Cada una de estas capacidades es un problema muy difícil de implementar en un robot.

Todos estos robots denominados robots móviles autónomos tienen una característica en común, que deben trabajar en ambientes no estructurados y necesitan de algoritmos de navegación que les permitan navegar y evitar obstáculos en tiempo real en diversos ambientes.

Esta tesis tratará el tema de navegación mediante redes neuronales y algoritmos genéticos, desde la perspectiva de la robótica basada en comportamientos. A la robótica que emplea estos tres conceptos se le denomina Robótica Evolutiva, cada uno de estos temas será abordado.

En el contexto de la robótica dentro de la Inteligencia Artificial, ha habido dos escuelas de pensamiento. La primera escuela creía en la aproximación más convencional de Inteligencia Artificial de usar una representación como la base de todas las formas de inteligencia artificial. A esta rama se le conoce como Inteligencia Artificial simbólica, tradicional, clásica o deliberativa.

Bajo esta aproximación de inteligencia artificial, el ambiente debía tener una representación en estructuras de datos para el robot, esto es, se debía contar con una representación del mundo dentro del robot, para lo cual se necesita una gran cantidad de conocimiento moldeado de acuerdo a algún tipo de representación del conocimiento.

Los trabajos centrados bajo esta filosofía hacían funcionar al robot en la manera percepción-representación-acción mediante una interfaz entre estos tres módulos dentro del sistema, en otras palabras, se requiere que el robot mantenga estructuras de datos compuestas por símbolos que logren estructurar el mundo sobre el que opera para lograr percibir el ambiente, representar el conocimiento y actuar para conseguir el objetivo deseado.

La segunda escuela de pensamiento argumentaba contra el uso de una representación formal, ya que se creía que la formulación apropiada de una de éstas era casi un problema intratable. Por lo que se propone que el mecanismo de control tenga diversos módulos, uno para cada acción que el robot pueda ejecutar y que cada módulo esté en contacto directo con el ambiente, en lugar de interactuar con los otros módulos.

A esta rama se le conoce como Inteligencia Artificial sub-simbólica o reactiva, teniendo como principal precursor a Rodney Brooks. Décadas después del auge del enfoque clásico de la Inteligencia Artificial, Brooks propone un sistema que no necesite de un modelo interno del mundo, mejor que el mundo real sea el modelo mismo.

Las principales ideas de esta escuela se basan en que las representaciones simbólicas no son necesarias para la inteligencia, excepto en un sentido muy limitado, dado que el mundo es el mejor modelo de sí mismo, que debe existir una interacción entre el robot y el entorno, que el robot se debe considerar como un todo y se ha de construir con una aproximación ascendente; esto es, de lo más sencillo hacia lo más complejo, y no como en la Inteligencia Artificial tradicional donde se tiene una suma de módulos de conocimiento independientes.

En un robot construido bajo este modelo de inteligencia se tendrían módulos que estén en interacción con el ambiente, donde cada módulo es capaz de recibir los valores de los sensores y mandar señales eléctricas a los actuadores, sin necesidad de comunicar conocimiento a otro módulo para la ejecución de una acción.

A estos módulos se les conoce como comportamientos, ya que, encapsulan una serie de acciones que el robot puede ejecutar al haber percibido ciertos valores de sensado, tal y como sucede en los animales (de aquí que algunos autores lo intenten ligar con la etología). De acuerdo a los valores de sensado es el comportamiento que se activará y ejecutará el robot. De aquí que surge la Robótica basada en comportamientos.

Ahora surge una cuestión interesante, ¿qué sucede cuando los valores de sensado logran activar dos o más comportamientos al mismo tiempo?, ¿cuál debería ser el comportamiento a ejecutar en el tiempo t?

En ambos casos se necesitaría organizar los comportamientos para que se ejecutara primero el de mayor importancia de acuerdo a las circunstancias presentes en el ambiente, con el fin de que el robot siga en la persecución del objetivo propuesto.

A esta organización de los comportamientos la Robótica basada en comportamientos le tiene asignado un módulo especial, al cual se le conoce como *árbitro de comportamientos* y se encargará de organizar la ejecución de dichos comportamientos, con el fin de que la secuencia de activación de comportamientos sea la óptima para lograr que el robot ejecute el objetivo propuesto.

Este es el tema central de la tesis. El árbitro de comportamientos se implementará mediante una red neuronal cuyos pesos serán evolucionados mediante un algoritmo genético, o sea, la red neuronal tendrá al algoritmo genético como método de optimización o adaptación.

Esta red neuronal permitirá al robot navegar por un ambiente acotado bajo cierto tipo de estructura, ya que el lograr que el robot navegue bajo cualquier tipo de ambiente sin una representación interna de éste no ha sido logrado, es un problema abierto y sólo se han logrado casos muy específicos para un tipo de ambiente.

Una vez que hemos definido la filosofía de la Robótica basada en comportamientos, que es sobre este enfoque que se desarrollará el árbitro de comportamientos para navegación, se definirán los conceptos de red neuronal y algoritmo genético en el siguiente capítulo.

1.1 Definición del problema

Esta tesis tiene por objetivo principal desarrollar un controlador, con la ayuda de un algoritmo genético, que organice los comportamientos de un robot móvil autónomo. Este controlador será de tipo neuronal, o sea, será una red neuronal la que organice a dichos comportamientos, con el fin de que el robot pueda navegar por el ambiente donde es evolucionado el controlador.

Este trabajo está diseñado bajo la aproximación de la Robótica Evolutiva, que abarca la Robótica basada en comportamientos, las técnicas evolutivas tales como los Algoritmos Genéticos y algún tipo de controlador, que en este caso serán las Redes Neuronales.

Entonces, el trabajo se centra en evolucionar los pesos de una red neuronal que permita organizar los momentos de ejecución de comportamientos simples de navegación de un robot móvil autónomo (como redes neuronales, máquinas de estados, etc.).

Cabe mencionar que el término *navegación* no es el mismo para la robótica basada en la inteligencia artificial clásica que para la robótica basada en comportamientos. En la primera, el término de navegación se refiere a la planificación de una ruta a partir de un nodo origen del mapa del ambiente a un nodo destino de éste y la correcta ejecución de esta trayectoria.

En la robótica basada en comportamientos, la navegación se refiere a que el robot puede *vagabundear* por el ambiente sin tener necesariamente fijada una posición destino (y si la tuviera, el robot no tendría un mapa explícito del entorno), mientras es capaz de esquivar cualquier obstáculo que se le interponga en su camino, sea estático o dinámico éste.

En este trabajo, se hará uso también de la metodología de la Inteligencia Artificial clásica en la cual el robot posee conocimiento del entorno en forma de estructuras de datos compuestas por símbolos, al proporcionarle un mapa de activación de árbitros de comportamientos para que navegue por el ambiente donde fue evolucionado.

Este trabajo de tesis no pretende resolver el problema general de navegación de un robot móvil autónomo para cualquier ambiente no estructurado, lo que se pretende es tener varios comportamientos globales para un robot evolucionados bajo un cierto ambiente. Después, proporcionarle un mapa de activación de árbitros de comportamientos para que el robot pueda navegar de una región inicio a una región destino; este mapa le indicará el momento de ejecución de cada comportamiento global (cada uno de estos tiene un árbitro de comportamientos asociado para la organización de varios comportamientos simples).

El principal objetivo de esta tesis es mostrar que el árbitro de comportamientos propuesto funciona bien y tiene ciertas ventajas sobre otros árbitros de comportamientos. También pretende que la metodología aquí propuesta sea, en adelante, trabajada con más detalle para llegar a una solución más general del problema de navegación.

1.2 Motivación

La motivación de desarrollar un árbitro de comportamientos para la navegación de un robot móvil de forma autónoma es con el fin de que a un corto plazo los robots móviles puedan interactuar con el ambiente de manera más óptima al seleccionar la secuencia de comportamientos y/o acciones indicadas y resulten de programación más simple, ya que no habría necesidad de programar el mapa topológico del entorno, sino un mapa de activación de árbitros de comportamientos.

1.3 Organización de la tesis

Capítulo 1 Introducción: En este capítulo se aborda de manera general la temática de la tesis, la definición del problema y la motivación.

Capítulo 2 Antecedentes: En este capítulo se da una breve introducción a los conceptos básicos y necesarios para entender el desarrollo de la tesis.

Capítulo 3 Inteligencia Artificial Simbólica: En este capítulo se abordan las bases y filosofía de la Inteligencia Artificial clásica, así como su aplicación en la Robótica mediante un ejemplo.

Capítulo 4 Inteligencia Artificial Sub-simbólica: En este capítulo se abordan las bases y filosofía de la Inteligencia Artificial no deliberativa, la cual da pie a la Robótica basada en comportamientos; también, se muestra un esquema de un robot que trabaja bajo este esquema.

Capítulo 5 Robótica basada en comportamientos: En este capítulo se abordan las bases de esta rama de la Robótica, donde es necesario codificar comportamientos para dar inteligencia a un robot. Se abordan los conceptos

de comportamientos, control reactivo, codificación de comportamientos y la organización de los comportamientos en un robot móvil.

Capítulo 6 Robótica Evolutiva: En este capítulo se aborda la evolución de redes neuronales mediante algoritmos genéticos, como medio para obtener el comportamiento deseado de un robot. Se explica el algoritmo genético utilizado y cómo se evolucionaron redes neuronales para navegación y el árbitro de comportamientos.

Capítulo 7 Resultados: En este capítulo se presentan los resultados de la evolución del árbitro de comportamientos y una comparación con un árbitro de comportamientos entrenado bajo un algoritmo de aprendizaje supervisado clásico.

Capítulo 2

ANTECEDENTES

Antes de abordar la definición de una red neuronal, hay que distinguir entre las redes neuronales biológicas, que son las que existen en el cerebro humano, y las redes neuronales artificiales, que son las que se tratarán aquí y de ahora en adelante simplemente se nombrarán como redes neuronales.

Una red neuronal es un modelo de cómputo implementado en software o en dispositivos de hardware especializado que intenta capturar las características adaptativas y de comportamiento de las redes neuronales biológicas. Típicamente, una red neuronal está compuesta de varias unidades de procesamiento interconectadas llamadas *neuronas*.

En términos matemáticos, una red neuronal puede ser vista como un grafo dirigido, donde cada nodo implementa el modelo de una neurona. En el caso más simple, el modelo de una neurona es simplemente una suma ponderada de las señales entrantes por el peso (conexión entre neuronas) por donde cada señal pasa, transformada por una función de transferencia estática (típicamente no lineal) [7].

Las conexiones entre neuronas son denominadas *pesos* y tienen algún valor de tipo real. Las neuronas que reciben los valores de los sensores son denominadas *neuronas de entrada*, las neuronas que mandan la salida de todo el cómputo de la red neuronal hacia un actuador se denominan *neuronas de salida*. Cabe mencionar que existen otros tipos de neuronas (*neuronas ocultas*) donde el objetivo del cómputo que se realiza en ellas es distinto al mencionado anteriormente.

El número y tipo de neuronas y el conjunto de posibles interconexiones entre éstas define la *arquitectura* o *topología* de la red neuronal. Otro factor en las redes neuronales es el tipo y algoritmo de aprendizaje con el cual son entrenadas. Los tipos de aprendizaje son *supervisado* y *no supervisado* básicamente.

En el primer tipo de aprendizaje se le proporciona a la red neuronal una serie de datos de entrada con sus respectivas salidas (datos de entrenamiento), la red neuronal va ajustando sus pesos hasta que el cómputo de sus neuronas de salida empata con las salidas indicadas. En el aprendizaje no supervisado sólo se le proporcionan las entradas sin adjuntar las salidas o respuestas deseadas de parte de la red neuronal. Las red neuronal, por medio de la regla de aprendizaje estima una función de densidad de

probabilidad que describe la distribución de patrones pertenecientes al espacio de la entradas a partir de las muestras [14].

A partir de la función de densidad de probabilidad se pueden conocer regularidades en el conjunto de entradas, extraer rasgos, agrupar patrones según similitudes.

Por otro lado existen algoritmos de aprendizaje, cada uno ligado a uno de los dos tipos de aprendizaje antes mencionados. Un algoritmo de aprendizaje muy conocido de tipo supervisado es el método de *backpropagation* en el cual la red neuronal va midiendo el grado de error obtenido a la salida al comparar la salida actual con la salida deseada y va corrigiendo este error mediante la retropropagación del error y un algoritmo de descenso en gradiente.

El método que se empleará en esta tesis para el aprendizaje de las redes neuronales es un algoritmo genético, y dada la manera en que se trabajará con éste se puede incluir en el tipo de aprendizaje supervisado, ya que, aunque no se incluirán las salidas deseadas al conjunto de datos sensados por un sensor láser, se le va suministrando una señal de corrección a través de la *función de fitness*.

La manera en como se le dirá a la red neuronal que tan bien se está desempeñando es mediante la *función de fitness*, la cual es una característica propia de los Algoritmos Genéticos. A continuación se dará un esbozo de estos métodos evolutivos.

Los Algoritmos Genéticos son un tipo de algoritmo evolutivo, esto es, son un método de búsqueda en donde se pretende encontrar una solución óptima a un problema, y la búsqueda se hace mediante una analogía de la evolución biológica.

Dado un problema de búsqueda, con un espacio multi-dimensional de posibles soluciones, se escoge una *representación genética* tal que cada punto en el espacio de búsqueda es representado por una cadena de símbolos, el *genotipo* o *cromosoma*; esto es, cada posible solución al problema debe estar codificada a manera de ser entendible para el algoritmo genético.

Después, se genera una población inicial de cromosomas de manera aleatoria (el número de cromosomas generalmente se sigue de las facilidades de cómputo disponibles). A continuación, se evalúa cada posible solución mediante una función de evaluación, comúnmente llamada *función de fitness*, la cual da una calificación de que tan bueno es el cromosoma en la resolución del problema, mientras más alta sea la calificación, mayor probabilidad tiene el cromosoma de ser elegido para la siguiente fase.

Una vez evaluados los cromosomas se eligen aquellos que tienen las probabilidades más altas, a estos cromosomas se les aplicaran *operadores de evolución* para dar paso a la siguiente población; esto es, los cromosomas seleccionados serán recombinados usando operadores genéticos para crear nuevos cromosomas, y se espera que la población nueva sea más adecuada para obtener la solución al problema inicial.

Los operadores genéticos básicos son el *cruce* y la *mutación*. El operador de cruce típico funciona al tomar pares de cromosomas, seleccionar aleatoriamente un punto en la cadena que codifica a cada cromosoma (punto de cruce), y con referencia en este

punto tomar la parte izquierda del primer cromosoma, tomar la parte derecha del segundo cromosoma y unir ambas partes. Después, tomar la parte derecha del primer cromosoma y unir con la parte izquierda del segundo cromosoma. Con esto obtenemos dos cromosomas *hijos* a partir de los dos cromosomas seleccionados anteriormente.

El operador de mutación únicamente se aplica sobre un cromosoma al seleccionar aleatoriamente un símbolo de la cadena que codifica a dicho cromosoma y cambiar dicho símbolo por otro símbolo válido. Este operador se aplica una vez que se tiene la nueva población, con la idea de dotar de cierta novedad a ésta.

El conjunto de nuevos cromosomas reemplaza al anterior y el proceso se repite un cierto número de veces hasta que se alcanza un criterio de convergencia. El resultado de este proceso es que idealmente el algoritmo entregue un conjunto de valores que sean una solución óptima al problema inicial.

A continuación se muestra un esquema de trabajo de un algoritmo genético básico.

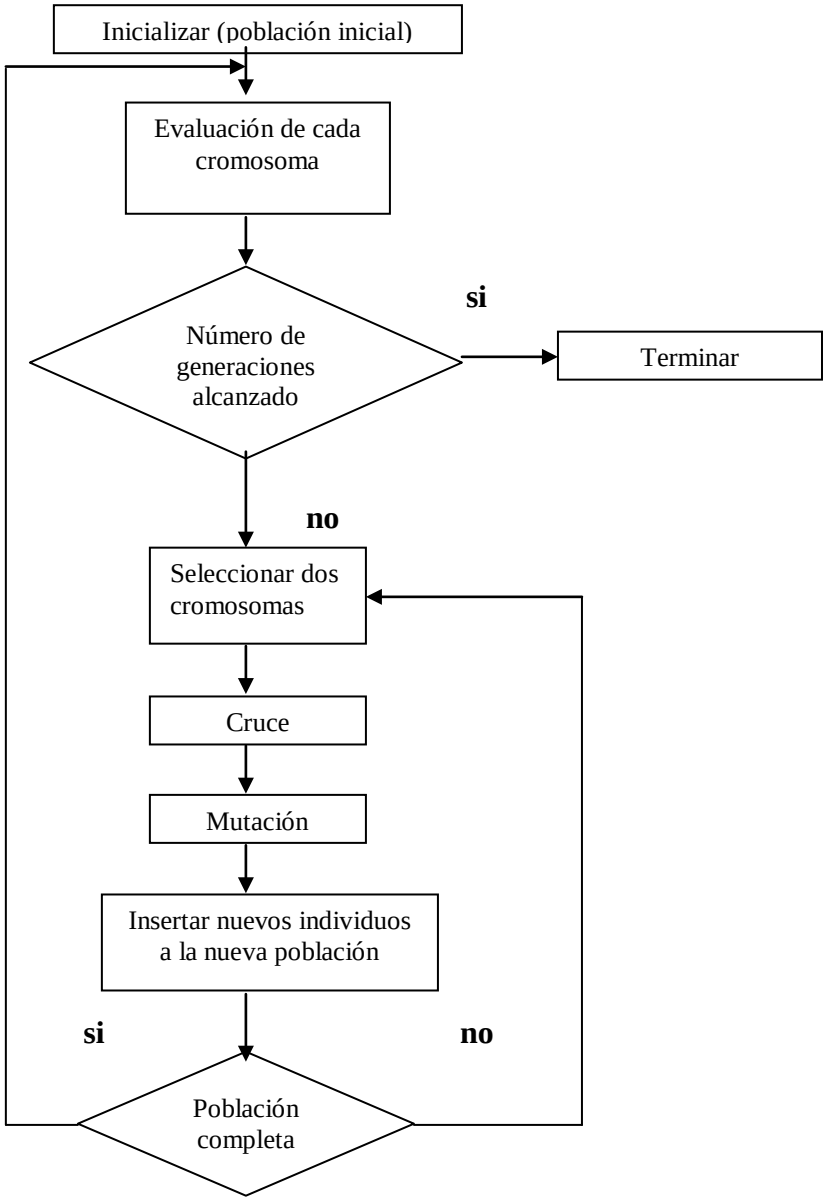


Figura 2.1. Esquema básico de un Algoritmo Genético.

Una vez explicada la forma básica de un algoritmo evolutivo surge la pregunta ¿Porqué debería ser usado un algoritmo evolutivo en la robótica? Primero que nada, un algoritmo genético permite una optimización tanto estructural como paramétrica del sistema bajo estudio, lo cual es importante en la robótica basada en comportamientos, ya que es raramente posible especificar por adelantado la estructura de, por ejemplo, el cerebro de un robot.

Segundo, un algoritmo genético tiene la habilidad de salir de mínimos locales de un espacio de búsqueda, por lo que, dado el tiempo suficiente, puede encontrar una solución cercana al óptimo global[1]. Finalmente, debido en parte a la naturaleza estocástica de un algoritmo evolutivo, se pueden encontrar diferentes soluciones viables al problema, punto importante en robótica.

Una vez abordados los temas de Robótica basada en comportamientos, Redes Neuronales y Algoritmos Genéticos, que son los principios básicos de la Robótica Evolutiva se dará una explicación más detallada de ésta.

Como se mencionó anteriormente, la Robótica Evolutiva se basa en la Robótica basada en comportamientos, en algoritmos evolutivos tales como los Algoritmos Genéticos, y en controladores que codifiquen comportamientos como las Redes Neuronales.

Diseñar un controlador para un robot inteligente y autónomo, que pueda planear su movimiento en tiempo real, en un ambiente desconocido y cambiante, es un gran reto para los investigadores que trabajan en este campo de la robótica.

Se han probado varios métodos por varios investigadores para resolver este problema. La Robótica basada en comportamientos es un sobresaliente resultado de tales esfuerzos. Aunque puede resolver los problemas de planeación de movimiento de un robot, tiene sus limitaciones, que podrían ser eliminadas usando el principio de la Robótica Evolutiva.

En la Robótica Evolutiva, el controlador de un robot se evoluciona dependiendo de las situaciones del ambiente que podría cambiar constantemente. En un ambiente cambiante e impredecible, el controlador de un robot no puede siempre ser diseñado por adelantado pero un controlador adecuado podría ser evolucionado, dependiendo de la situación del ambiente, usando un algoritmo genético.

Por lo que, en la Robótica Evolutiva, un controlador neuronal (red neuronal que sirve como controlador) adecuado es evolucionado mediante un algoritmo genético, y este controlador puede dar una solución aceptable en tiempo real a problemas del mundo real.

El inconveniente de las técnicas evolutivas es su lenta convergencia y la considerable cantidad de tiempo que tiene que ser invertido para llevar a cabo la evolución de un controlador en un robot real. Es por esto que la mayoría de las evoluciones de controladores se llevan a cabo en un simulador y después el controlador evolucionado es probado en el robot real. Para esta tesis este será el procedimiento.

[1] Un AG con elitismo converge al óptimo global [23].

Capítulo 3

INTELIGENCIA ARTIFICIAL SIMBÓLICA

La Inteligencia Artificial clásica, en su historia, se ha basado en otras disciplinas que han contribuido a su desarrollo y gestación. Una de las disciplinas que más han contribuido a la IA es la Filosofía, desde el año 428 a. C hasta nuestros días, con las teorías del conocimiento y la racionalidad. Aristóteles (384-322 a. C.) fue el primero en formular un conjunto preciso de leyes que gobernaban la parte racional de la inteligencia [24].

Algunos filósofos han manifestado la posibilidad de máquinas inteligentes como dispositivos para ayudarnos a definir lo que significa ser humano. René Descartes, por ejemplo, parece haber estado más interesado en un “hombre mecánico” como una metáfora más que como una posibilidad [5].

Preguntas tales como ¿cómo se genera la inteligencia mental a partir de un cerebro físico? Son cuestiones filosóficas que han sido estudiadas por investigadores de la Inteligencia Artificial con el fin de obtener programas que sean análogos a la mente humana.

Otra disciplina fundamental para la gestación de la Inteligencia Artificial son las Matemáticas, desde el año 800 aproximadamente hasta nuestros días. Los campos que más presencia han tenido en el desarrollo de teorías concernientes a la IA son la lógica, la probabilidad y la teoría de la computabilidad [24].

El concepto de lógica formal se remonta a los filósofos de la antigua Grecia, pero su desarrollo matemático comenzó realmente con el trabajo de George Boole, quien definió la lógica Booleana, misma que se utiliza en la electrónica digital, y creó la lógica de primer orden que se utiliza hoy como el sistema más básico de representación del conocimiento [24].

Dentro del área de la probabilidad está la regla de Bayes, la cual es una regla para la actualización de probabilidades subjetivas a la luz de nuevas evidencias. Esta regla ha derivado en una nueva área denominada análisis Bayesiano que conforman la base de las propuestas más modernas que abordan el razonamiento incierto en sistemas de IA.

En el área de la teoría de la computabilidad está presente la Máquina de Turing, la cual es capaz de calcular cualquier función computable, es en esta teoría donde se basa el funcionamiento de una computadora tal cual la conocemos hoy en día.

Otra disciplina que ha contribuido es la Economía, desde el año 1776 hasta el presente, con la teoría de la decisión, que combina la teoría de la probabilidad con la teoría de la utilidad y proporciona un marco completo y formal para la toma de decisiones [24]. Otro campo es la investigación de operaciones, la cual busca la optimización de funciones.

La Economía ilustra perfectamente cómo interactúan el ambiente externo e interno y, en particular, cómo el ajuste de un sistema inteligente a su ambiente externo se ve limitado por su capacidad para descubrir mediante el conocimiento y el cálculo de un comportamiento adaptativo apropiado [28].

La Neurociencia y la Psicología, han sido otros campos del conocimiento que han sido de gran interés por parte de investigadores de la IA, ya que intentan responder a preguntas como ¿cómo procesa información el cerebro?. El poder contestar esta pregunta daría la pauta para poder programar los procesos cognitivos presentes en la mente humana y así llegar a una verdadera inteligencia artificial.

La Ingeniería computacional es otra disciplina que está estrechamente ligada con la IA, el diseño de procesadores y computadores digitales que puedan soportar cómputos que requieren del máximo de recursos y de la mayor velocidad posible, como las búsquedas en un juego de ajedrez, son uno de los objetivos de la IA y de este campo.

Otro objetivo que persigue la IA es que un sistema pueda ser autónomo, operar bajo su propio control, este objetivo lo comparte con la Teoría de control y la Cibernética, ramas de la Ingeniería que buscan sistemas con retroalimentación y autorregulación, razón por la cual está estrechamente ligada la IA con la Teoría de control.

Después de haber citado las anteriores disciplinas, que comparten objetivos en común con la IA y que han ayudado a gestionarla como una ciencia en vez de una técnica, ya que sabemos que la IA persigue objetivos como conocer la mente humana, programar procesos cognitivos presentes en ésta, desarrollar teorías que sirvan para optimizar y tomar decisiones, encontrar soluciones algorítmicas a problemas complejos, encontrar la manera de automatizar todos estos procesos, desarrollar sistemas de cómputo que permitan realizar todos estos cómputos, ¿qué otro proceso le serviría a la IA?

La respuesta está en la última disciplina por mencionar (última por ahora, la IA sigue expandiendo su dominio a otras disciplinas), la Lingüística, que estudia la relación del lenguaje con el pensamiento. A la IA le interesa el procesamiento del lenguaje natural en un programa de cómputo, la representación del conocimiento para estructurarlo lo más cercano posible a la información almacenada en la mente humana.

Como se puede observar, la IA se nutre de varias disciplinas, intenta tomar teorías y piezas de conocimiento de cada una de ellas para poder resolver problemas complejos de los cuales no se cuenta con una solución algorítmica, o sea, no son computables a exactitud y necesitan de una inteligencia “extra-algorítmica” para poder ser resueltos, u optimizar algoritmos para resolver problemas computables.

Una vez visto todo el trasfondo con el que cuenta la IA, veremos el punto de vista clásico para manejar el conocimiento y realizar los procesos necesarios para la resolución de un problema, que es la IA simbólica, y nos enfocaremos principalmente a un sistema tradicional para dotar de inteligencia a un robot.

3.1 Inteligencia Artificial simbólica

La Inteligencia Artificial simbólica, es aquella que surge con el estudio de los sistemas de símbolos físicos basados en el conocimiento, propuesto por Newell y Simon. En estos sistemas se hace uso de operaciones lógicas aplicadas a bases de conocimiento.

La principal premisa de la Inteligencia Artificial tradicional o simbólica es que la inteligencia del robot es intrínsecamente un fenómeno computacional y, por tanto, puede ser estudiada y llevada a la práctica en distintos sistemas, aunque éstos estén fuera de su entorno de actuación.

A esta rama se le conoce también como aproximación basada en conocimiento, ya que se debe dotar del conocimiento del ambiente (mapa del ambiente) y de las acciones a ejecutar al robot por medio de estructuras de datos, esto es, todo lo que la inteligencia del robot manipulará para navegar por el ambiente, planear una acción y ejecutarla estará en términos de estructuras de datos.

Por lo cual, se puede decir que la Inteligencia Artificial clásica se funda en la hipótesis del sistema de símbolos, formulada en 1976 por Allen Newell y Herbert Simon, y afirma que "un sistema de símbolos físicos tiene los medios suficientes y necesarios para generar una acción inteligente" [24]. Esto significa que un robot o una persona tiene inteligencia si maneja al conocimiento en forma de estructuras de datos compuestas por símbolos.

Ahora veamos cómo es que un robot tiene inteligencia bajo esta aproximación simbólica. Los sistemas que siguen esta aproximación parten de una descomposición de los procesos, que el robot debe realizar, en tareas independientes que posteriormente se unen para lograr el objetivo planteado a dicho sistema.

La siguiente figura muestra un esquema de trabajo que un robot móvil ejecuta mediante este modelo tradicional.

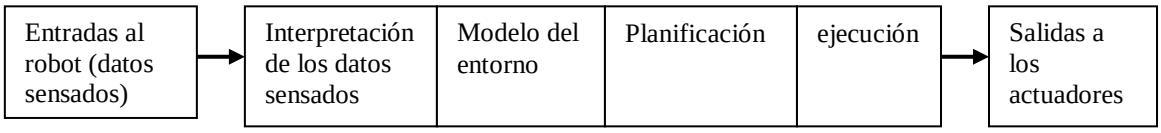


Figura 3.1. Arquitectura de un sistema basado en la aproximación tradicional de la Inteligencia Artificial.

Los módulos de la figura 3.1 están organizados de forma serial, por lo que la falla de alguno de éstos significa que el robot fracasaría en la tarea a implementar. A continuación se explicará brevemente cada módulo.

Interpretación de los datos sensados

En este módulo, el robot toma los valores obtenidos por medio de sus sensores y los compara en su base(s) de conocimiento y verifica qué tipo de información es la que debe aplicar al modelo de mundo interno. Los sensores más utilizados son el infrarrojo, el láser y los sonares.

Modelo del entorno

Una vez obtenida la información del entorno, y sabiendo cuáles son los movimientos a realizar, el robot debe computar un escenario (ruta) del entorno sobre el cual se guiará para realizar lo planificado; esto se logra mediante algoritmos de búsquedas. Aquí es donde se muestra claramente la necesidad de tener un modelo interno del entorno sobre el cual operará el robot.

Planificación

En este módulo, el robot debe hacer la planificación de los movimientos que realizará para lograr la tarea objetivo, en base al modelo interno del mundo que posee. Este módulo se logra bajo alguna máquina de inferencias.

Ejecución

Una vez que el robot se ha ubicado, ha planeado sus movimientos y ha trazado una ruta sobre la cual desplazarse, entonces manda señales eléctricas a sus actuadores (motores) para ejecutar la tarea objetivo.

Este enfoque parecía prometedor en los inicios de la Inteligencia Artificial, pero tenía tres debilidades:

1) Una baja velocidad de respuesta del robot, debido al uso de un mecanismo de control centralizado.
2) Un lento progreso en el estado-del-arte al estar dominado por resultados simbólicos.
3) Basta un pequeño cambio en el entorno para que el robot falle en su objetivo.

Estas desventajas dieron pie a la formación de un nuevo paradigma para la inteligencia del robot, uno que haría de lado el uso de la representación formal del conocimiento. Es así como surge la Inteligencia Artificial sub-simbólica que daría pie a la Robótica basada en comportamientos.

Capítulo 4

INTELIGENCIA ARTIFICIAL SUB-SIMBÓLICA

Esta aproximación surge como alternativa a la IA clásica (Inteligencia Artificial), su principal precursor, Rodney A. Brooks en [4] argumenta que la IA clásica ha enfatizado la manipulación abstracta de símbolos, cuya puesta a tierra en la realidad física rara vez se ha logrado. Esto es, la IA clásica, como se vio en el capítulo anterior, utiliza una representación del mundo y es con ese modelo que trabaja el robot, si algo en el mundo cambia, el modelo tendría que actualizarse, de lo contrario, con mucha dificultad se conseguirá el objetivo planteado.

Brooks en su artículo *Elephants Don´t Play Chess* (1990) da evidencia de las fallas de manejar un sistema de símbolos para dotar de inteligencia a un sistema y él propone trabajar con un modelo de sensado que excluya todo sistema de símbolos. De aquí el nombre de IA sub-simbólica.

4.1 Inteligencia Artificial sub-simbólica

La Inteligencia Artificial sub-simbólica es aquella que surge con la *arquitectura subsumida* para robots móviles propuesta por Brooks, y con el enfoque *animat* propuesto por Brooks y Wilson para agentes inteligentes en la década de los 1980. En esta arquitectura y en este enfoque, se pretende que el robot o el agente tengan un comportamiento de tipo etológico, o sea, que simulen el comportamiento de un animal.

Brooks en [4] formaliza esta alternativa, que el llama *"Nueva IA"*, que está basada en la *hipótesis física de puesta a tierra* y está en contraposición a la *hipótesis de sistema de símbolos* de [28].

Esta última hipótesis, como se vio anteriormente, afirma que la inteligencia opera sobre un sistema de símbolos, por lo que, el sistema de inteligencia central de un robot debe ser alimentado, por medio del sistema de percepción, de símbolos. Ahora, surge una interrogante, ¿cuál es la descripción correcta, en términos de símbolos, del mundo en torno al sistema de inteligencia?.

Brooks, en su hipótesis, afirma que es mejor que el robot perciba el mundo directamente a través de sensores, en lugar de ser alimentado por símbolos. En términos generales, afirma que para construir un sistema que sea inteligente es necesario tener sus representaciones puestas a tierra en el mundo físico. Entonces, la necesidad de trabajar con símbolos desaparece, ya que al estar sensando directamente del mundo, no hay necesidad de estar actualizando ningún modelo del mundo, el mundo es su mejor modelo.

Así, bajo este nuevo enfoque, el robot móvil no necesitaría tener un modelo del entorno sobre el cual operar. Su forma de ubicarse sería mediante la interacción que surja con su entorno, y la forma de interactuar con su entorno es mediante la información que provenga de sus sensores, que entrarán a un módulo que controlará sus actuadores como lo haría un animal controlando sus músculos y extremidades de acuerdo al estímulo sensado. Entonces, el robot puede tomar una acción dentro de un conjunto de acciones o *comportamientos*, dependiendo de lo que esté percibiendo.

El siguiente diagrama muestra el esquema de trabajo propuesto en esta tesis para un robot móvil, mediante el modelo basado en comportamientos.

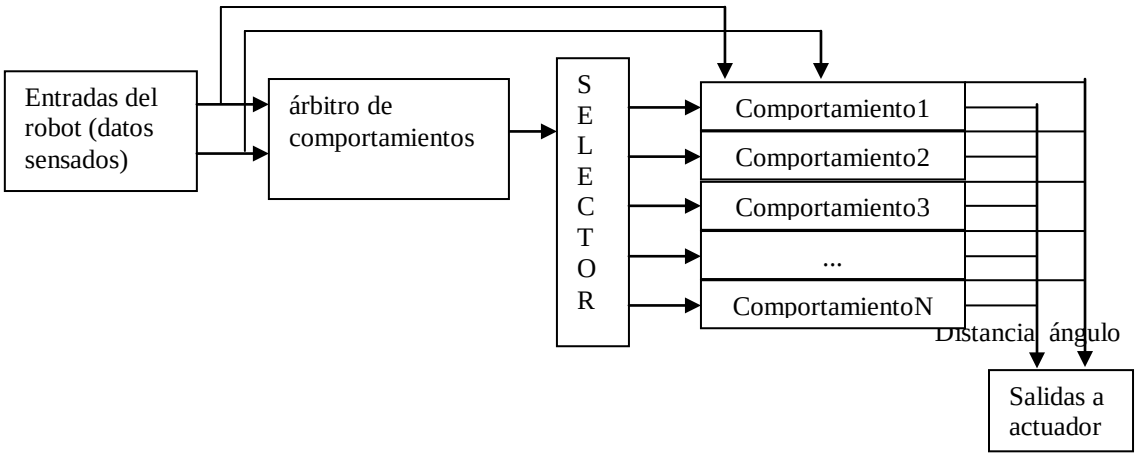


Figura 4.1. Arquitectura de un sistema basado en la aproximación de la robótica basada en comportamientos.

Cada módulo recibe el nombre de *comportamiento*, ya que es un módulo de control sobre los actuadores que tiene programada como salida una serie de estímulos eléctricos que harían que el robot tuviera el comportamiento descrito en la etiqueta de éste bajo los estímulos sensados.

Por ejemplo, el comportamiento de *explorar* haría que el robot, al sensar determinado(s) estímulo(s), avanzara hacia una determinada orientación, simulando lo que haría un animal al explorar un entorno.

En el caso del comportamiento *esquivar obstáculo* haría que el robot al sensar un obstáculo, ejecutara una secuencia de movimientos (mediante sus actuadores) para lograr evadir ese obstáculo que le impediría seguir ejecutando el comportamiento de explorar, por ejemplo.

El árbitro de comportamientos sólo puede seleccionar un comportamiento a la vez de los *N* existentes en el modelo, y cuando uno de estos comportamientos es seleccionado el árbitro de comportamientos le pasa las señales de entrada y este comportamiento tiene permitido pasar las señales de su cómputo resultante a los actuadores para controlar al robot.

Como se puede observar, el robot no tiene un modelo interno del entorno, por lo que no puede realizar planificaciones del modo que lo haría un robot con el modelo tradicional, aunque si puede realizar tareas específicas, ya que el poder responder a estímulos da muestra de que tiene una *inteligencia*, a este tipo de inteligencia se le llama *emergente*, ya que emerge de la interacción del robot con el entorno.

Nótese que bajo este modelo, el robot no necesita estar "atado" a un entorno, puede trabajar sobre cualquier entorno que tenga los estímulos que requiere sensar y que tenga el medio de desplazamiento adecuado para sus actuadores.

En el caso en que dos comportamientos sean activados por los datos sensados, se necesitaría de un árbitro que permitiera la ejecución del comportamiento prioritario. Este problema de la selección de comportamientos se ha implementado bajo diferentes arquitecturas. Por ejemplo, la arquitectura subsumida de Brooks [3] tiene su propia implementación del árbitro, existiendo más arquitecturas con su propio árbitro.

4.2 Aproximación simbólica vs. Aproximación sub-simbólica

Brooks en [4] hace un rápido análisis de ambas aproximaciones y una comparativa. La metodología de la IA clásica basa su descomposición de inteligencia en módulos de procesamiento de información funcional, cuyas combinaciones proveen el total del sistema de comportamiento.

La metodología propuesta por Brooks, basa su descomposición de inteligencia en módulos de generación de comportamientos individuales, cuya coexistencia y cooperación permiten emerger comportamientos más complejos.

De lo anterior podemos observar que en la IA clásica, ninguno de los módulos por sí solo genera el comportamiento del sistema total, es necesario combinar varios módulos o todos los módulos para obtener algún comportamiento del sistema. Entonces, para obtener una mejora en la capacidad del sistema habría que mejorar cada módulo.

En la IA sub-simbólica cada módulo, por sí mismo, genera un comportamiento, y para mejorar la capacidad del sistema simplemente habría que añadir nuevos módulos al sistema.

Una insuficiencia bien conocida de la IA clásica es el problema del marco. Los sistemas de símbolos en su forma pura asumen una verdad objetiva conocible, donde es imposible asumir cualquier cosa que no esté explícitamente establecida.

En un sistema basado en la hipótesis de puesta a tierra, éste eventualmente tiene que expresar todas sus metas y deseos como acciones físicas, y debe extraer todo su

conocimiento de los sensores, por lo que, el diseñador del sistema está forzado a hacer todo explícito.

Por otro lado, ambas aproximaciones aunque han demostrado cierta clase de éxito, sólo pueden recurrir a vagas esperanzas cuando se intenta generalizar el dominio de problemas a resolver.

La IA tradicional ha intentado demostrar razonamiento sofisticado en dominios pobres. La esperanza es que las ideas usadas generalizarán a un comportamiento robusto en dominios más complejos.

La IA sub-simbólica intenta demostrar tareas menos sofisticadas operando robustamente en dominios complejos con ruido. La esperanza es que las ideas usadas se generalizarán a tareas más sofisticadas.

Brooks visualiza las deficiencias de ambas aproximaciones y comenta que parecen ser complementarias y se pregunta si al combinar estas dos aproximaciones se podría obtener un sistema con más poder.

En esta tesis se hará una combinación de ambas aproximaciones. En el nivel bajo se encuentra un árbitro de comportamientos que sirve para navegar de una región a otra. En el nivel alto se encuentra un mapa de activación de árbitros de comportamientos que va indicando al robot qué árbitro activar para lograr trazar una navegación compuesta por varios árbitros.

Capítulo 5

ROBÓTICA BASADA EN COMPORTAMIENTOS

La IA sub-simbólica da pie al inicio de la Robótica basada en comportamientos. La arquitectura de subsumpción de Brooks muestra que es posible que un robot cuente con un sistema de control no jerárquico, descentralizado, que no maneje coordenadas ni modelos cinemáticos, y que sea basado en comportamientos, tal como su robot Genghis, mostrado en la figura 5.9.

Como método de codificación de comportamientos Brooks utiliza las máquinas de estados; en esta tesis se utilizan redes neuronales para el desarrollo del árbitro de comportamientos, para los comportamientos se utilizan redes neuronales. Otro método de codificación de comportamientos son los campos potenciales.

Adelante, se abordará el tema de árbitro de comportamientos que es el encargado de organizar a los comportamientos del robot (redes neuronales en este trabajo). De igual manera, se utilizará una red neuronal como árbitro en esta tesis.

5.1 Comportamientos

En la robótica basada en comportamientos, el cerebro de un robot se construye a partir de un repertorio de comportamientos. Cuando se define un comportamiento es importante distinguir entre el resultado deseado de un comportamiento y la manera en la cual se consigue el resultado.

Por ejemplo, en la robótica autónoma por lo general existe el comportamiento para esquivar obstáculos. Entonces, el resultado deseado de dicho comportamiento es esquivar obstáculos y un algoritmo como *girar 20 grados mientras se detecte obstáculo por los sensores, avanzar de lo contrario* es un ejemplo de una implementación detallada del comportamiento.

La palabra comportamiento deriva de los primeros experimentos que se realizaban con este tipo de robótica, donde los investigadores etiquetaban cada sistema de control por medio de la observación del resultado de la interacción del robot con el entorno cuando se activaba.

Así, si cuando un sistema de control se activaba en el robot, éste esquivaba obstáculos, al sistema de control se le llamaba comportamiento de esquivar obstáculos. Este comportamiento no es más que una estructura de control codificada sobre una base de procesamiento, como una red neuronal, y que en un entorno y robot determinado produce una actuación del robot etiquetada como tal comportamiento [26].

Ahora, hay que diferenciar entre la estructura de control sobre la cual es codificada el comportamiento y el comportamiento por si mismo. Esto es, una estructura de control puede contener una serie de instrucciones que al ser ejecutadas en un robot bajo cierto ambiente producen el comportamiento de, por ejemplo, esquivar obstáculos.

Entonces, la estructura de control contiene el algoritmo o serie de computaciones a ejecutar por un robot cuando ésta sea activada. Por ejemplo, una red neuronal que se activa cuando el robot detecta la cercanía de un objeto por medio de sus sensores.

En cambio, el comportamiento es el resultado de que la estructura de control haya sido activada y de los resultados obtenidos por el robot al ser controlado por ésta. Si se quiere ser preciso, un comportamiento es el resultado de la interacción entre el controlador implantado en un robot y el entorno.

5.1.1 Comportamientos inteligentes y robustos

Siguiendo la aproximación de la robótica basada en comportamientos, Brooks enumera los siguientes aspectos que se deben tomar en cuenta al momento de moldear un sistema de este tipo para producir comportamientos inteligentes y robustos [26] y [12].

- **Ubicación:** del inglés *situatedness*, indica que el robot al estar inmerso en el mundo real no requiere operar con representaciones abstractas de éste, sino con el mundo mismo. En los sistemas basados en la inteligencia artificial clásica, se cuenta con un modelo interno simbólico del ambiente, un robot bajo este enfoque en realidad no trabajaría con el mundo real, sino con un modelo de éste, ya que el dinamismo que existe en el mundo no es tomado en cuenta en dicho modelo. Entonces, se crea una confusión entre el conocimiento que posee el robot sobre el mundo y el mundo en el cual está operando debido a los cambios que se dan en éste. En cambio, los robots bajo la aproximación de los comportamientos utilizan el mundo real como su propio modelo, es decir, no existe un modelo interno del mundo, sino que establecen una relación directa entre los valores que van sensando con las acciones a ejecutar.

- **Incorporación:** del inglés *embodiment*, indica la participación activa del robot, en términos de sus características físicas con el mundo. Sin una incorporación real no existe significado para un robot; esto es, si el robot no tiene una participación netamente activa con el ambiente, no habrá una retroalimentación por parte de este último.

- **Emergencia:** del inglés *emergence*, indica que la inteligencia de un robot surge de la interacción con el entorno, no es una propiedad del robot o del entorno por separado. De aquí cabe decir que la inteligencia se encuentra en el ojo del observador, ya que la inteligencia es determinada dado el comportamiento global del robot y de la forma en que se da este comportamiento en relación al entorno.

- **Puesta a tierra:** del inglés *grounding*, indica que la información que el robot utiliza se recoge directamente del mundo y no a través de conocimiento en forma de estructuras de datos compuestas por símbolos como sucede en la Inteligencia Artificial clásica. En otras palabras, lo que se sugiere es que el robot tome las representaciones de conocimiento *puestas a tierra* en el mundo a través de sensores, o sea, que el conocimiento que posea no sea una representación simbólica sino datos del propio mundo.

5.2 Control reactivo vs. Control deliberativo

Por reactivo se entiende aquel comportamiento que únicamente utiliza la información actual de sus sensores para realizar una acción y no necesita de conocimiento en forma de símbolos sobre el mundo.

Así, un control reactivo, como una red neuronal, no necesita de alguna representación interna en forma de símbolos del mundo, ni de planificar las acciones a realizar; únicamente toma la información de sus sensores en el momento actual, hace los cómputos pertinentes y envía una señal eléctrica a los actuadores del robot.

En cambio, en un control deliberativo, el robot posee un modelo del mundo, planifica en base a este modelo los movimientos necesarios para lograr un objetivo y los ejecuta en el mundo real.

Las principales diferencias entre robots que poseen cada uno de los dos tipos de controles anteriores son la forma en que se manejan en el mundo, la manera en que manejan el conocimiento y la manera en que representan sus objetivos.

Un robot que posee un control deliberativo "*vive en un mundo idealizado"* donde el mundo no cambia y donde el conocimiento que posee sobre éste siempre es correcto, ya que no se *asoma* al mundo para trabajar en él, sino que trabaja con el modelo del mundo y con el conocimiento que el diseñador le proporcionó, y al hacer una planificación para lograr un objetivo explícito lo hace en base a dicho modelo, y si llega a cambiar el entorno entonces el robot muy probablemente falle en su cometido.

Un robot que posee un control reactivo *"vive en el mundo real"*, todas las acciones que realice serán en base a la información que obtenga del sensado del mundo en ese instante de tiempo, pero no se le puede dar un objetivo a planificar, ya que con un control reactivo no

es posible planificar. Un posible objetivo sería esquivar obstáculos, ya que es codificable como un comportamiento.

Nótese la diferencia entre ambos tipos de control en cuanto al objetivo. A un robot deliberativo es posible pedirle que vaya por un vaso de agua a la cocina, con la condición de que el entorno no cambie en lo absoluto. A un robot reactivo si se le pide ese vaso de agua no es seguro que lo traiga, ya que navegaría por el ambiente sin planificación alguna, y por tanto no es seguro que llegue hasta el lugar donde se encuentra el vaso.

Véase la figura 5.1, donde se muestran estas y otras diferencias entre estos tipos de control.

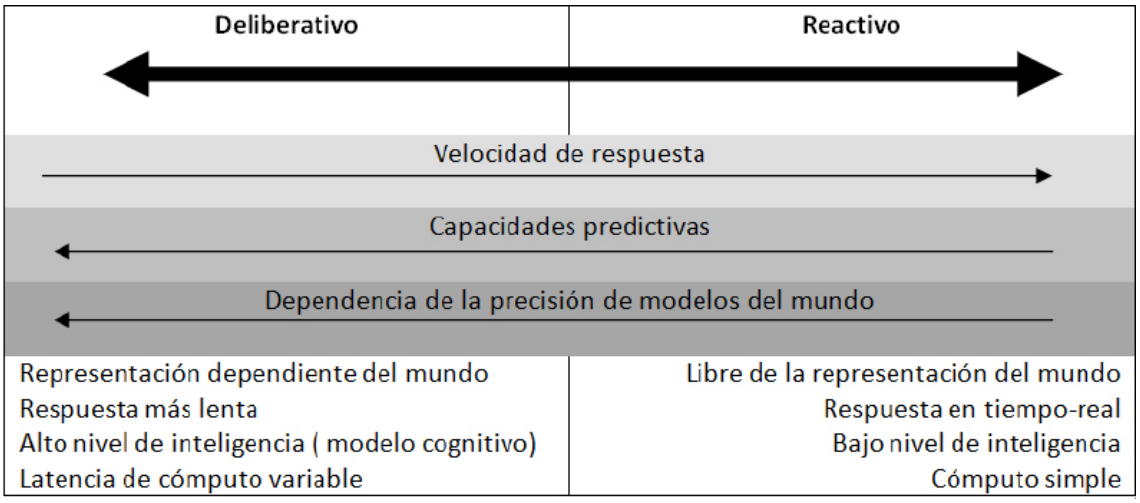


Figura 5.1. Propiedades de un control reactivo y un control deliberativo [19].

5.3 Codificación de comportamientos

Al titular *codificación de comportamientos* a este apartado, se hace referencia al uso de un controlador para programar los comportamientos de un robot. Se utiliza la palabra "codificar" para referirse al mapeo que existe de los valores de los sensores de un robot a un pulso eléctrico dirigido a los actuadores de éste, ya que el significado de dicha palabra, según la Real Academia de la Lengua Española, es "transformar mediante las reglas de un código la formulación de un mensaje".

En este caso, al tener un controlador se estarían transformando los valores sensados a movimiento de motores mediante las reglas de cómputo de dicho controlador. Los controladores que aquí se verán son de los más usados en la Robótica basada en comportamientos y en la Robótica Evolutiva, ya que, son propensos a ser evolucionados.

De acuerdo a Irman Harvey y sus colaboradores, existen tres posibles conjuntos generales de estructuras evolucionables que se pueden emplear para conseguir los controladores de comportamientos requeridos [26]. Estos son:

- Programas de control explícitos
- Expresiones matemáticas
- La definición de una estructura de procesamiento

Las dos primeras estructuras raramente se utilizan en la robótica basada en comportamientos. Por lo tanto, se concentrará la atención en la tercera opción. Dentro de ésta, las estructuras de procesamiento o cómputo más utilizadas son las redes neuronales, las máquinas de estados finitos y los campos potenciales, con un predominio de las redes neuronales debido a ventajas que a continuación se explicarán.

En los siguientes apartados se dará una explicación de estas tres estructuras de cómputo, haciendo mayor énfasis en las redes neuronales, por ser las utilizadas en el desarrollo del árbitro de comportamientos de esta tesis.

5.3.1 Redes Neuronales

En la introducción se había comentado que una red neuronal podía verse como un grafo dirigido, donde las unidades de procesamiento son las neuronas. Ahora explicaremos más a detalle todo esto.

Antes de comenzar, cabe mencionar que existen diferentes tipos de redes neuronales, aquí se explicará el modelo básico de una red neuronal y después se detallará el tipo de red neuronal utilizada en esta tesis, que es el *multiperceptrón*.

De manera básica, una red neuronal es una estructura de procesamiento distribuido en la que distinguimos entre nodos o neuronas, que realizan una función matemática, típicamente no lineal, que simulan grosso modo la frecuencia de la activación eléctrica en las neuronas biológicas, y conexiones o pesos entre las neuronas que simulan la eficiencia de la conexión sináptica entre las dendritas y axones de las redes neuronales biológicas.

5.3.1.1 Neurona

Como ya se mencionó, las redes neuronales están compuestas de nodos o neuronas, como la mostrada en la figura 5.2, que es una neurona básica. Estas neuronas están conectadas entre sí a través de *pesos* o conexiones dirigidas con un valor numérico.

La conexión de una neurona *i-ésima* a una neurona *j-ésima* sirve para propagar la activación de la neurona *i-ésima* a la *j-ésima*. Además cada conexión tiene asociado un valor numérico denominado *W*.

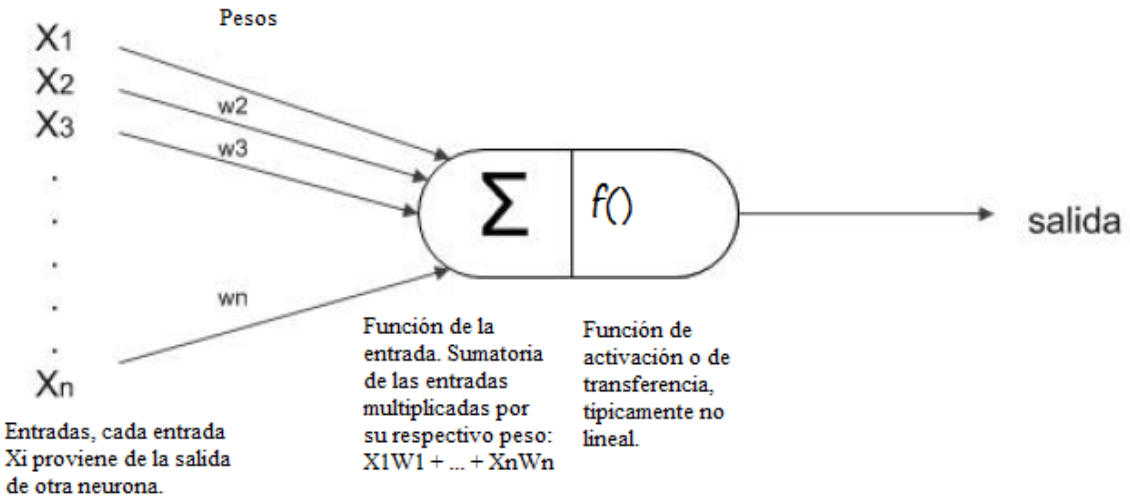


Figura 5.2. Modelo matemático básico de neurona de una red neuronal genérica. La activación de la salida de la neurona es $f(\sum Xi * Wi)$ donde *Xi* es la activación de la salida de la neurona *i* y *Wi* es la conexión o peso que va de la neurona *i* a la neurona actual. La entrada X_i de cada neurona proviene de la salida de otra neurona, excepto en las neuronas de la capa de entrada.

La operación de una neurona genérica [1] es un proceso de dos etapas. En la primera, cada entrada a la neurona es multiplicada por el peso por el cual pasa hacia la neurona actual. Después, se hace una sumatoria de estos productos. A esta primera etapa se le nombrará *S*.

Además de las entradas anteriormente mencionadas se define una entrada llamada *sesgo* o *umbral* (*b*), el cual mide la propensión de la neurona de activarse en la ausencia de las demás entradas. La primera etapa de cómputo de una neurona se define matemáticamente como:

$$S = (\sum_{i=1}^{n} Xi * Wi) + b \qquad (5.1)$$

Donde *n* es el número de neuronas que mandan su señal de salida hacia la neurona actual, *Wi* son los pesos que conectan a las neuronas que mandan su salida hacia la neurona actual con ésta, *Xi* son las señales de salida de las neuronas antes mencionadas y que son señales de entrada para la neurona actual.

Nótese que el sesgo representa el umbral de disparo de la neurona, es decir, el nivel mínimo que debe alcanzar el potencial de la sumatoria para que la neurona se active.

Para simplificar la notación, el sesgo por lo general se escribe como X_0W_0, donde $W_0 = b$ y $X_0 = 1$. Con esta notación, la primera etapa de cómputo de una neurona queda:

$$S = \sum_{i=0}^{n} Xi * Wi \qquad (5.2)$$

En la segunda etapa, el nivel de actividad obtenido de la primera etapa se utiliza para generar la actividad final de salida de la neurona al utilizar dicho nivel de actividad como entrada a una función típicamente no lineal, llamada función de activación o de transferencia. Así, se tendrá que la salida de la neurona, denominada *Y*, tenga un nivel de actividad continuamente graduado. Esto, matemáticamente queda:

$$Y = f(S)$$

Esto es:

$$Y = f(\sum_{i=0}^{n} Xi * Wi) \qquad (5.3)$$

Donde *f()* es la función de activación de la neurona.

Una razón de incorporar una función de activación es que la suma, *S*, es potencialmente ilimitada. Las neuronas reales están limitadas en el intervalo dinámico desde una tasa de disparo de salida igual a cero hasta un máximo de unos cuantos de cientos de potenciales de acción por segundo. Por tanto, la salida de la neurona debe ser restringida a un cierto intervalo [1].

Otra razón de tener una función de activación es para que la salida de la neurona quede a un valor restringido (cercano a +1) para indicar que está activa cuando se proporcionen las entradas correctas, e inactiva (cercano a 0 o -1) cuando se den entradas erróneas.

Una razón de que la función de activación sea no lineal es para no realizar transformaciones lineales consecutivas. Esto es, si no existiera la función de activación o ésta fuera lineal, la salida de una neurona sería la simple sumatoria, la cual al multiplicarse por un peso sería la entrada de otra neurona, y esta neurona haría otra sumatoria de sus entradas y haría un mapeo lineal de las entradas a su salida y así consecutivamente.

Cabe mencionar que las funciones de activación, aparte de ser no lineales, deben ser derivables, ya que el método de aprendizaje más común utiliza método de descenso de gradiente para que la red neuronal aprenda, y para que se pueda utilizar un método de descenso en gradiente la función de activación debe ser derivable.
Las funciones de activación más comunes son:

- Función sigmoide: $\frac{1}{1+e^{-X}}$
- Función tangente hiperbólica: tanh

5.3.1.2 Arquitectura básica de una red neuronal

Una vez definida la unidad de procesamiento central de una red neuronal, se expondrá la arquitectura básica de una red neuronal (véase fig. 5.3).

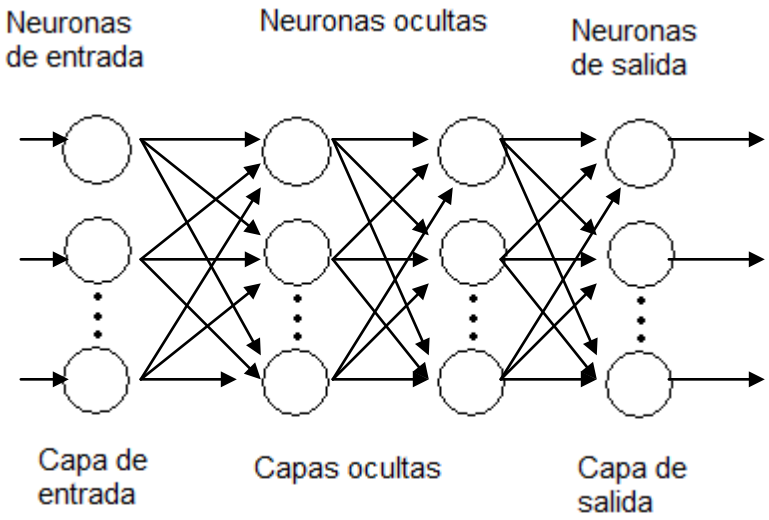


Figura 5.3. Una arquitectura de una red neuronal para control de un robot móvil.

La estructura de una red neuronal básica consta de los siguientes elementos:

- Una *capa de entrada*. Aquí las *neuronas de entrada* reciben los valores de los sensores o en forma general los datos de entrada. Estas neuronas no hacen ningún tipo de cómputo sobre los datos recibidos.

- Una *capa oculta*. En realidad puede haber más de una capa oculta o no haber capa oculta, pero lo más común es utilizar una. Aquí, las *neuronas ocultas* o *escondidas* están fuertemente conectadas con las neuronas de la capa de entrada, con las neuronas de la siguiente fase y con la neurona de umbral. Estas neuronas realizan el cómputo descrito en al apartado anterior. Gracias a esta capa se obtienen resultados con más grados de libertad, ya que la salida de estas neuronas mapean a un espacio no lineal.

- Una *capa de salida*. Las neuronas de esta capa reciben como entrada las salidas de las neuronas ocultas. Estas neuronas también realizan el procesamiento descrito con anterioridad. La salida de estas neuronas se conectan a los actuadores del robot, en forma general dan el cómputo final.

- *Pesos*. Son las conexiones que existen entre las neuronas y tienen un valor real. Cuando una red neuronal se dice que *aprende*, en realidad lo que sucede es que las conexiones van ajustando sus valores, de acuerdo a algún método de aprendizaje, tal que las entradas que recibe coincidan con las salidas deseadas.

Una definición interesante de red neuronal hace uso del concepto matemático de grafo, que describe la arquitectura del sistema. Una posible definición matemática de red neuronal utilizando el concepto de grafo toma la siguiente forma (ampliando lo explicado en la introducción) [14]:

Definición: una red neuronal es un grafo dirigido, con las siguientes propiedades:

1) A cada nodo *i* se asocia una variable de estado *Xi*

2) A cada conexión (*i*,*j*) de los nodos *i* y *j* se asocia un peso $Wij \in \Re$

3) A cada nodo *i* se asocia un umbral b_i

4) Para cada nodo *i* se define una función $f(Xj, Wij, b_i)$ que depende de los pesos de sus conexiones, del umbral y de los estados de los nodos *j* a él conectados. Esta función proporciona el nuevo estado del nodo.

En la terminología habitual de las redes neuronales, los nodos son las neuronas y las conexiones con valor son los pesos. Algunos de los conceptos, ya expuestos, quedarían redefinidos en este contexto de la siguiente manera:

- Se denomina *neurona de entrada* a las neuronas sin pesos entrantes.
- Se denomina *neurona de salida* a las neuronas sin pesos salientes.
- Las neuronas que no son de entrada ni de salida se denominan *neuronas ocultas*.
- Una red neuronal es *unidireccional* cuando no presenta ciclos cerrados de conexiones.
- Una red neuronal es *recurrente* cuando el flujo de información puede encontrar un ciclo de atrás hacia delante, es decir, una retroalimentación.

A continuación se describirá el *multiperceptrón*, que es la red neuronal más común, y es la que se usará en esta tesis.

5.3.1.3 Multiperceptrón

El multiperceptrón tiene como antecedente el perceptrón, propuesto por Rosenblatt en 1959, el cual es un simple modelo unidireccional, compuesto por dos capas de neuronas, una sensorial o de entrada, y otra de salida (véase la figura 5.3 y elimine las capas ocultas). El resultante es el modelo de una red neuronal perceptrón.

Las neuronas de entrada no realizan ningún tipo de procesamiento, las neuronas de salida realizan las dos fases de cómputo anteriormente señalado, aunque en la segunda etapa, la función de activación de una neurona de perceptrón es la función escalón.

Así, la salida del perceptrón puede tomar únicamente uno de dos valores, 0 o 1. La utilidad de que el perceptrón tome uno de estos dos valores es que puede utilizarse como clasificador, ya que, dado un vector de entrada, una neurona de salida responde con 0 si dicho vector no pertenece a cierta clase, y con 1 en caso de que si pertenezca.

Es fácil ver que una neurona tipo perceptrón solamente permite discriminar entre dos clases linealmente separables, es decir, cuyas regiones de decisión pueden ser separadas mediante una única condición lineal o hiperplano. De aquí la importancia de que una red neuronal posea neuronas ocultas, ya que éstas le dan un grado de no linealidad a la salida de la red neuronal.

El perceptrón fue, y es, poderoso y fácil de implementar. Desafortunadamente, todo lo que prometió en la década de los sesenta fue un gran potencial comercial que nunca se materializó: no habían muchos problemas prácticos que realmente requirieran de un perceptrón.

Para finales de la década, pocos análisis duros y unos cuantos resultados prácticos emergieron de los perceptrones, luego de años de trabajo y de grandes cantidades de dinero del gobierno y de la industria.

Marvin Minsky, científico estadounidense, considerado uno de los mayores exponentes de la Inteligencia Artificial, a la par de su trabajo de procesamiento de símbolos, escribió un libro junto a Seymour Papert, titulado *Perceptrons* (1969) en el cual evidenciaban las limitaciones del perceptrón.

El estoque en el libro de Minsky y Papert era que los perceptrones teóricamente tenían algunas limitaciones muy importantes y que no podían calcular ciertas cosas prácticas y de importancia.

A partir de la publicación de este libro, las redes neuronales decayeron fuertemente, hasta que se descubrió que añadirle una capa más producía salidas no lineales, con lo cual era posible hacer de las redes neuronales clasificadores no lineales. Este es el caso del multiperceptrón.

Si añadimos capas intermedias u ocultas a un perceptrón simple, obtendremos un perceptrón multicapa o multiperceptrón. La ventaja sobre el perceptrón es que al añadir una capa oculta es posible representar cualquier función continua de las entradas con una precisión arbitraria; con dos capas incluso se pueden representar funciones discontinuas [24], véase la figura 5.3 para ver un modelo de un multiperceptrón de dos capas escondidas.

Las principales características de los multiperceptrones son (notas de clase de "*Redes neuronales"* impartida por el Dr. Ángel Kuri):

1) Son aproximadores universales (con a lo más dos capas ocultas).

2) Aproximan a un clasificador Bayesiano.

3) Existen heurísticos para determinar la arquitectura y los parámetros libres.

4) Es progresiva.

5) Es fuertemente conexa.

6) La neurona de umbral alimenta cada capa.

7) Una capa escondida tiene como utilidad mapear a otra dimensión para resolver mejor el problema en cuestión.

8) Se tendrán tantas neuronas de entrada como variables independientes se tengan.

9) Se tendrán tantas neuronas de salida como grupos a clasificar se tengan.

10) El número de neuronas en la capa escondida (según un heurístico) es aproximadamente un tercio del tamaño de los datos de muestra que se tengan para entrenar a la red neuronal. Esto es:

$$(I+1)H+(H+1)O \approx \frac{S}{3}$$

Donde:

I: número de neuronas de la capa de entrada.
H: número de neuronas de la capa oculta.
O: número de neuronas de la capa de salida.
S: tamaño de la muestra.
La neurona añadida al número de neuronas de entrada y ocultas es la neurona de umbral.

Por lo que:

$$H \approx \frac{S-3O}{3(I+O+1)}$$

Veamos un poco más de cerca qué significa que un multiperceptrón es un aproximador universal. El desarrollo del multiperceptrón durante los últimos treinta años ha resultado curioso. Partiendo de un perceptrón y observando sus limitaciones computacionales, se llegó a la arquitectura perceptrón multicapa, y aplicándolo a numerosos problemas, se comprobó experimentalmente que éste era capaz de representar mapeos complejos y de abordar problemas de clasificación de gran envergadura, de una manera eficaz y

relativamente simple. Sin embargo, faltaba una demostración teórica que permitiese explicar sus aparentemente enormes capacidades computacionales.

Este proceso histórico comienza con McCulloch y Pitts (1943), quienes mostraron que una adecuada combinación de neuronas basadas en el modelo propuesto por ellos[1] podía representar cualquier función lógica; mucho más tarde, Denker y otros (1987) demostraron que toda función booleana podía ser representada por una red unidireccional de una sola capa oculta. Por las mismas fechas, Lippmann (1987) mostró que un perceptrón con dos capas ocultas bastaba para representar regiones de decisión arbitrariamente complejas. Por otra parte, Lapedes y Farber (1987) demostraron que un perceptrón de dos capas ocultas es suficiente para representar cualquier función arbitraria (no necesariamente booleana) [14].

Después, diversos grupos propusieron casi a la par teoremas muy similares que demostraban matemáticamente que un multiperceptrón convencional, de una única capa oculta, constituía en efecto, un aproximador universal de funciones (véase la figura 5.4).

Respecto al punto dos, un clasificador Bayesiano surge de la regla de Bayes y es un sistema probabilístico que asume que sabe todo sobre la distribución de las entradas posibles, esto es, qué tan probable es que un punto en particular pertenezca a cualquier clasificación dada. Un clasificador Bayesiano es el mejor clasificador posible.

Se había mencionado en la introducción que el aprendizaje de una red neuronal consiste en la modificación de sus pesos, tal que se conocen los datos de entrenamiento junto con las salidas deseadas. Entonces, la red neuronal al tener como entradas estos datos de entrenamiento, fuese capaz de dar las salidas deseadas. Abordemos un poco más detalladamente esta cuestión.

Una red neuronal trabaja en dos fases: en la primera se le presentan los datos de entrenamiento, que incluyen un conjunto de entradas con sus respectivas salidas deseadas, y con una configuración inicial aleatoria de los valores de los pesos de la red se hace el cómputo a partir de los datos de entrada y se verifica que las salidas concuerden con las salidas deseadas.

En la segunda fase, se aplica un algoritmo de aprendizaje para modificar los pesos de la red analizando los errores a la salida, a manera de obtener las salidas deseadas; esta es la base del aprendizaje supervisado. Tal como se había dicho en la introducción, existen dos tipos de aprendizaje, el *supervisado* y el *no supervisado*.

[1] La neurona de McCulloch-Pitts es una neurona de dos estados, que tiene un umbral fijo y recibe entradas de sinapsis excitadoras, las cuales tienen pesos idénticos. Las activaciones sinápticas se suman linealmente. La neurona también puede recibir entradas de sinapsis inhibidoras, cuya acción es absoluta; esto es, si la sinapsis inhibidora está activa, la neurona no se puede activar. Si la suma de las entradas excede el umbral, entonces la neurona se activa, de lo contrario permanece inactiva [1].

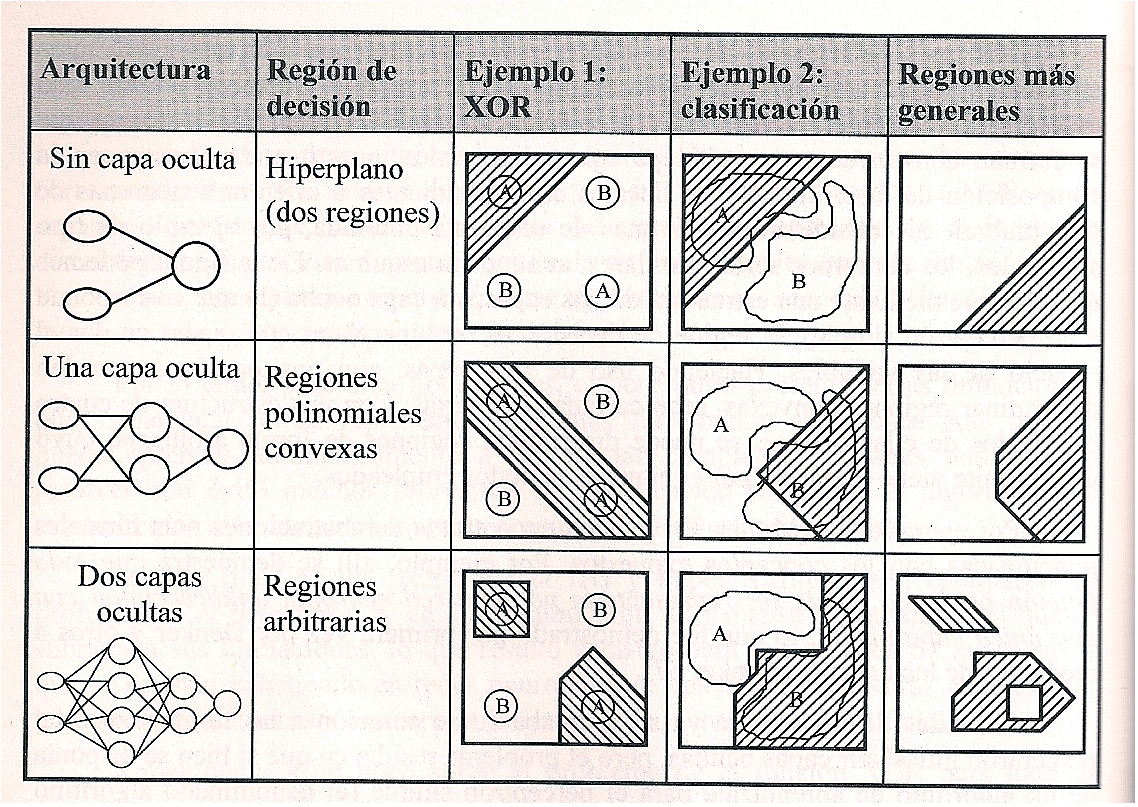


Figura 5.4. Tipos de regiones de decisión en el perceptrón [14].

5.3.1.4 Aprendizaje supervisado

En este tipo de aprendizaje se le proporciona a la red neuronal los datos de entrenamiento, que son datos de entrada, y las salidas que se desea tenga la red neuronal al computar dichas entradas. Entonces cada vez que a la red neuronal se le presente un vector de entrada, la RN computará la salida y se comparará con la salida deseada; a esta comparación se le denomina señal de error. Lo que se pretende es estimar una función multivariable desconocida $f: \Re^n -> \Re^m$ (la que representa la red neuronal) a partir de las muestras (x, y) $(x \in \Re^n, y \in \Re^m)$ tomadas aleatoriamente, por medio de la minimización iterativa de una función de error que surge de las señales de error [14].

Una variante de la aproximación clásica del aprendizaje supervisado es la metodología trabajada en esta tesis. Mediante la función de fitness se ajusta una señal de corrección que le indica a la red neuronal que controla en turno al robot, si va bien o va mal en su desempeño.

Los pesos serán ajustados, mediante un algoritmo de entrenamiento, a valores que hagan que la señal de error se reduzca hasta idealmente cero. El algoritmo de aprendizaje de tipo supervisado de los multiperceptrones es el *error backpropagation* o *retropropagación del error*; este algoritmo está basado en la corrección del error entre la salida deseada y la

salida obtenida por la red neuronal. De ahora en adelante será referenciado como *backpropagation*.

Básicamente, el aprendizaje de *backpropagation* consiste de dos fases: la fase *hacia delante* y la fase *hacia atrás*. En la primera fase, un patrón de actividad (vector de entrada) es aplicado a las neuronas de entrada de la red, propagándose los cómputos de las neuronas hacia las capas de adelante. Finalmente, se produce un conjunto de salidas, cada una correspondiendo con una neurona de salida, como la respuesta actual de la red. Durante esta primera fase, los pesos de la red neuronal están fijos. En la segunda fase, los pesos de la red son ajustados de acuerdo a una regla de corrección de la señal de error. Específicamente, la respuesta actual de la red es restada de la salida deseada para producir una señal de error. Esta señal de error se propaga hacia atrás a través de la red, en dirección opuesta de la dirección de las conexiones (pesos). De aquí el nombre de error *backpropagation* [9].

Para ver cómo funciona este método de aprendizaje, considere los pesos de una red neuronal, suponga que hay *n* pesos. Consideremos a este conjunto de pesos como un punto dentro de un espacio *n*-dimensional, que llamaremos *espacio de pesos*. Suponga que se tiene un conjunto de patrones que se requieren clasificar. Cada vez que un vector de entrada es dado a la red neuronal se obtiene una señal de error. Esto significa que cada conjunto de pesos se ha asociado con un valor de error escalar; si se cambian los pesos, se obtiene un nuevo valor de error. Los valores de error para cada conjunto de pesos definen una *superficie* en el espacio de pesos.

Entonces, una manera de ver el proceso de aprendizaje, consiste en requerir que se reduzca el error en el mayor grado posible, lo que se convierte ahora en un problema de minimización. Una manera obvia de encontrar un mínimo consistiría en examinar muchos puntos, esto es, el método de fuerza bruta. En general, esta aproximación no funcionará, ya que dependiendo de la complejidad del problema será el tiempo de búsqueda, y probablemente sobrepasaría la capacidad de cualquier computadora [1].

Este problema se puede ver como el del caminante en la niebla: no puede ver dónde se encuentran los valles, sin embargo, puede ver la topografía local alrededor de donde está parado. Entonces, una cosa que podría hacer el caminante es caminar siempre hacia abajo. Eventualmente llegará al punto en donde no pueda seguir bajando, porque todas las direcciones le llevarían hacia arriba, por lo que ese punto sería el más bajo del valle. A este punto se le conoce como *mínimo local*.

Pueden haber muchos mínimos locales, y algunos mínimos pueden estar más altos que otros. El mínimo más bajo se conoce como *mínimo global*. Desafortunadamente las superficies de error complejas tienen muchos mínimos locales, por lo que utilizar un algoritmo que siempre descienda no garantiza encontrar el mínimo global, excepto que sólo haya un mínimo y ese sea el global. Es por esto que el algoritmo de *backpropagation* utiliza un algoritmo de descenso de gradiente para intentar obtener el mínimo global.

Para este trabajo de tesis, el método a utilizar para entrenar al árbitro de comportamientos consta de un aprendizaje por refuerzo en conjunto con un algoritmo genético. Este es el funcionamiento: se tiene una población de cromosomas, donde cada cromosoma codifica

los pesos de una red neuronal, cada una de estas redes organizarán a comportamientos de navegación (encapsulados en redes neuronales o máquinas de estados); cabe mencionar que los pesos de las redes neuronales son, en un principio, valores reales aleatorios.

Cada red neuronal entrenada tendrá derecho a actuar en el ambiente bajo un cierto número de movimientos de parte del robot. Se evaluará la eficiencia de cada red neuronal, bajo un cierto criterio codificado en una función de fitness.

Después, se aplican los conceptos de cruza y mutación del algoritmo genético utilizado, se obtiene una nueva población de pesos de redes neuronales y se vuelve a evaluar su eficiencia. Esta eficiencia (*fitness*) es la señal de corrección que recibe cada red neuronal para decir si "va bien" o "va mal" en su proceso de controlar al robot. Este tipo de aprendizaje se llama por refuerzo debido a que no se le presenta el conjunto de entrenamiento a la red neuronal, en cambio se le da una señal de recompensa o refuerzo que pueden ser componentes o sugerencias de la utilidad actual a maximizar (por ejemplo, un buen movimiento).

En aprendizaje por refuerzo el objetivo es aprender cómo mapear situaciones a acciones para maximizar una cierta señal de recompensa, pero sin decirle al sistema cómo realizar la tarea. Este tiene que obtener experiencia útil acerca de los estados, acciones, transiciones y recompensas de manera activa para poder actuar de manera óptima, así el sistema trata de aprender un comportamiento mediante interacciones de prueba y error. En general, al sistema no se le dice qué acción debe tomar, sino que él debe de descubrir qué acciones dan el máximo beneficio. La evaluación del sistema ocurre en forma concurrente con el aprendizaje [6].

En este trabajo de tesis, el algoritmo genético es el encargado de dar la señal de corrección a la red neuronal que está siendo evaluada, por medio del *fitness*, entonces la red neuronal lo que hace es eventualmente llegar a mapear los valores del láser a señales de estímulo para el motor del robot, de tal forma de lograr el comportamiento requerido. De aquí se puede ver que las entradas a la red neuronal bien podrían ser otro estímulo relacionado con el ambiente y ésta de igual manera lograría mapear dicha entrada con la señal hacia el motor del robot para lograr el mismo comportamiento.

5.3.1.5 Aprendizaje no supervisado

A diferencia del aprendizaje supervisado, donde se proporcionaban a la red neuronal los datos de entrenamiento y las salidas deseadas, en este tipo de aprendizaje sólo se proporcionan los datos de entrenamiento, dejándole a la red neuronal la autoorganización de sus pesos para conseguir categorizar las entradas con características comunes.

El aprendizaje no supervisado se puede describir genéricamente como las estimación de la función de densidad de probabilidad $p(x)$ que describe la distribución de patrones x pertenecientes al espacio de entrada a partir de muestras. Esto es, a la red neuronal se le presentan multitud de patrones sin adjuntar la respuesta deseada. La red, por medio de la

regla de aprendizaje estima *p(x)*, a partir de lo cual pueden reconocerse regularidades en el conjunto de entradas, extraer rasgos, o agrupar patrones según su similitud [14].

5.3.2 Máquinas de Estados Finitos

Una máquina de estados finitos consiste de un número finito de estados y transiciones condicionales entre esos estados. A un momento de tiempo *t*, una máquina de estados lee un símbolo de entrada y produce un símbolo de salida mientras salta al siguiente estado objetivo [32].

Estas máquinas de estados fueron inventadas a mediados de la década de 1950 y fueron introducidas en el campo de la Inteligencia Artificial por Lawrence Fogel. La idea general era exponer a una máquina de aprendizaje una secuencia de símbolos, que son datos de entrada para ésta, y enseñarle a producir un símbolo de salida útil en respuesta a la entrada. En las máquinas de estados, los símbolos de entrada y salida son tomados de dos alfabetos discretos, el alfabeto de entrada y el alfabeto de salida, aunque comúnmente ambos alfabetos son iguales.

Las máquinas de estados son una alternativa a las redes neuronales, que aunque no son tan precisas como estas últimas, proveen de un marco de trabajo útil para la robótica basada en comportamientos. Sin embargo, la restricción de tener alfabetos finitos no es realista en la mayoría de las simulaciones robóticas, excepto probablemente aquellas donde se usen cuadrículas discretas. Para un ejemplo de máquinas de estados véase la figura 5.5, donde se muestra una máquina de estados que codifica el algoritmo de un robot móvil que evade obstáculos; para ver un ejemplo de la ejecución de esta máquina de estados en un robot móvil véase la figura 5.6.

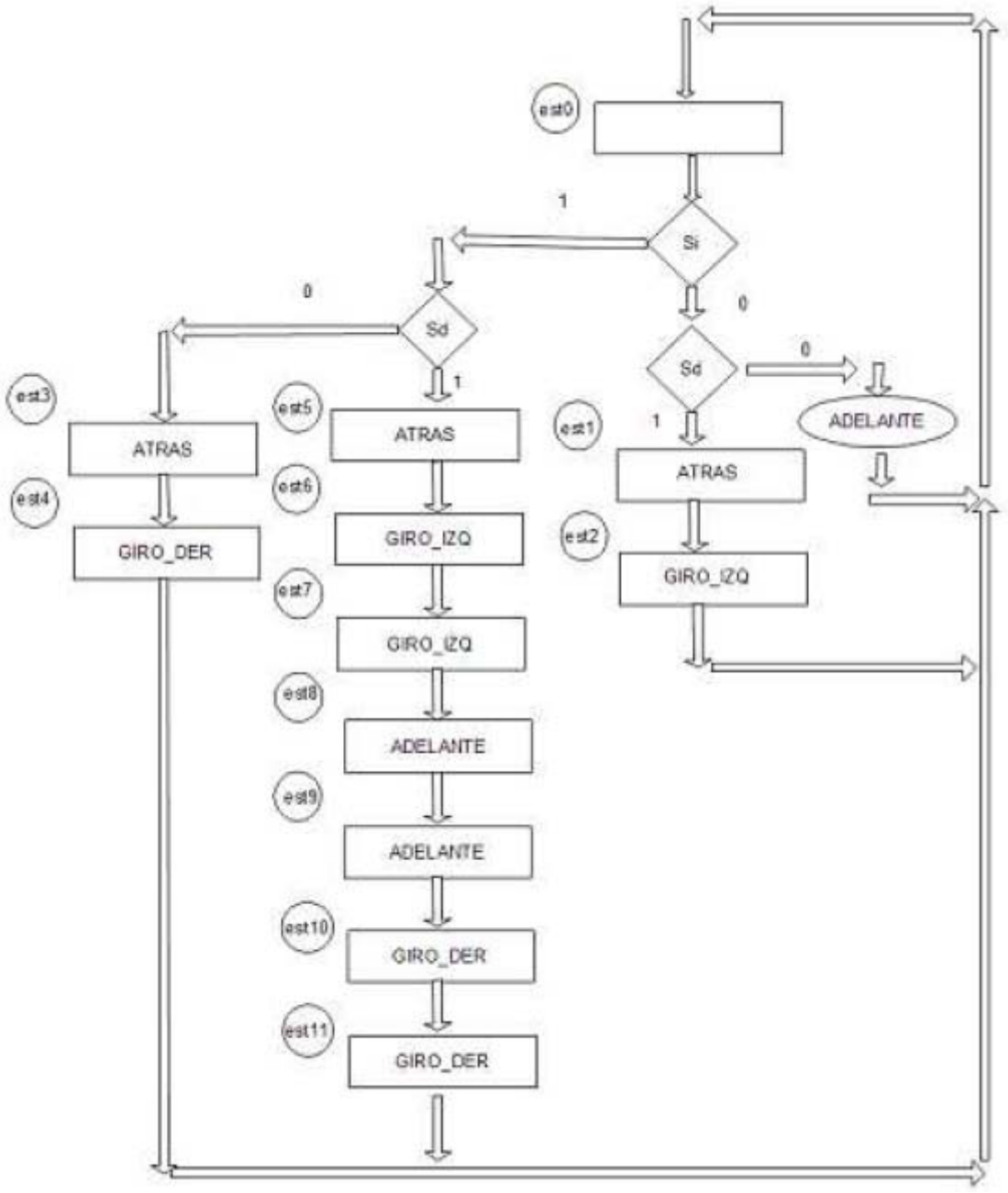


Figura 5.5. Algoritmo codificado en una máquina de estados de un robot móvil evasor de obstáculos [27].

Las máquinas de estados pueden ser extendidas para enfrentarse a los sistemas continuos. Una extensión es el uso de temporizadores internos, o sea relojes, que controlen el tiempo que se necesita para pasar de un estado a otro. Las máquinas de estados resultantes son conocidas como máquinas de estados finitos aumentadas.

Viendo a las máquinas de estados desde el punto de vista de la robótica evolutiva, es posible evolucionarlas, bajo la Programación Genética, en donde toda la máquina de estados es propensa a ser evolucionada, esto es, toda su arquitectura. Si utilizamos un

algoritmo genético, es posible evolucionar las constantes que existan en una máquina de estados.

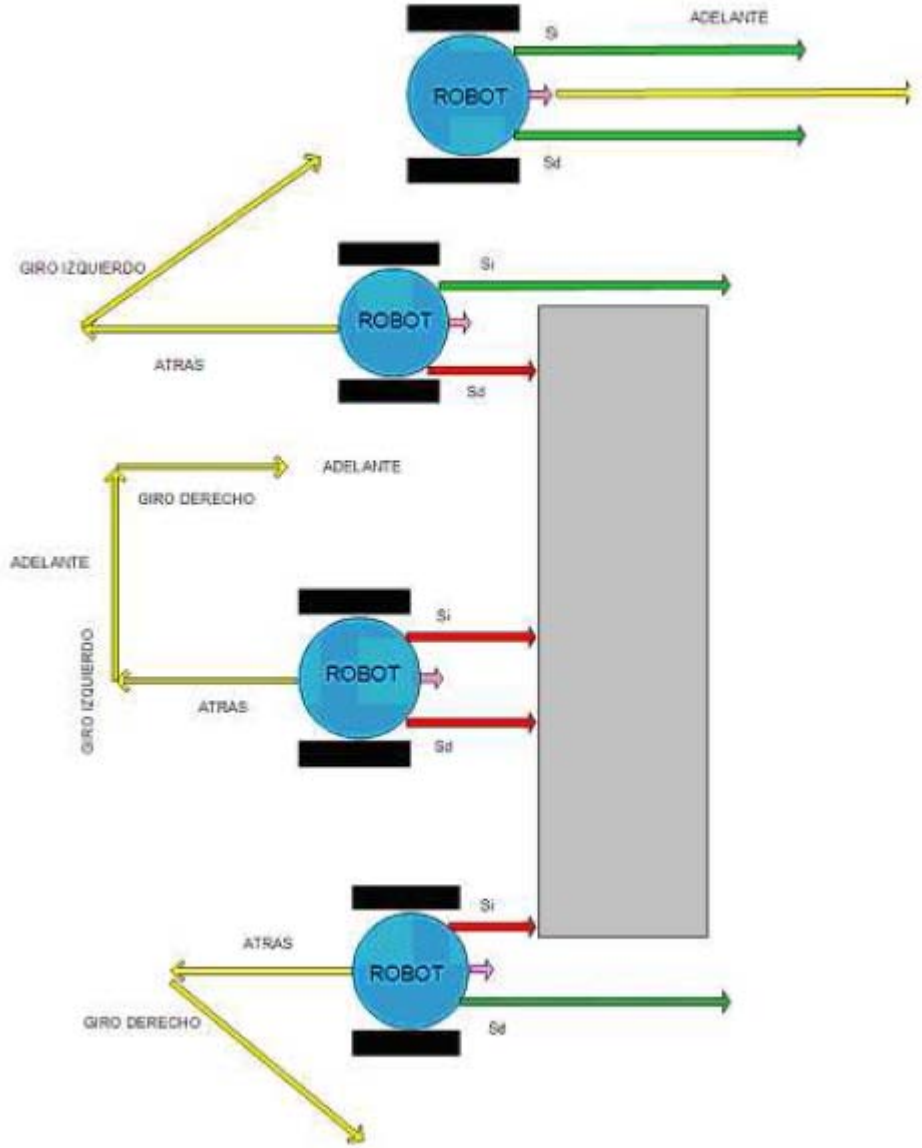


Figura 5.6. Robot móvil que evade obstáculos [27].

Por ejemplo, tómese la máquina de estados de la figura 5.5, en donde se tiene dos sensores, S_i y S_d, que al registrar un determinado valor *a* se envía una señal constante *b* a los motores del robot para echarse en reversa. Se pueden encontrar las constantes *a* y *b*, con un AG, tales que hagan que el robot evite chocar con un obstáculo de manera más eficiente.

5.3.3 Campos Potenciales

Los campos potenciales son usados como método para navegación, donde el robot es representado como una partícula bajo la influencia de un campo potencial escalar *U* [2]. El robot se mueve en la dirección sugerida por el gradiente negativo de un campo generado por los objetos en la vecindad del robot, esto para no chocar con ellos.

El punto destino es asignado a un campo potencial correspondiente a una pendiente colina abajo, mientras que los objetos son asignados a un campo potencial que genera una colina empinada.

Un campo potencial se define matemáticamente:

$U = Uatr + Urep$

Donde, *Uatr* y *Urep* son campos potenciales atrayente y repulsivo respectivamente.

El campo potencial atrayente tiene como objetivo jalar al robot, razón por la cual es ubicado en el punto destino, mientras que el campo repulsivo intenta empujar al robot lejos de los obstáculos.

El vector campo de fuerzas artificiales *F(q)* es dado por el gradiente de *U*:

$F(q) = -\nabla Uatr + \nabla Urep$

Donde ∇U es el vector gradiente de *U* en la posición *q(x,y)* del robot en un mapa bidimensional. De esta manera, *F* es definido como la suma de dos vectores, $Fatr(q) = -\nabla Uatr$ y $Frep(q) = \nabla Urep$, tal como se muestra en la siguiente ecuación.

$F(q) = Fatr(q) + Frep(q)$

La forma general de las funciones de campo potencial adecuadas fue propuesta por Kathib (1990), tal como sigue:

(a) Campo potencial de atracción $Uatr = \frac{1}{2}\xi d^2$

donde:

$d = |q - q_a|$

q es la posición actual del robot

q_a es la posición de un punto de atracción

ξ es una constante ajustable

Obtenemos el gradiente:

$$\nabla Uatr = D\frac{1}{2}\xi d^2$$

$$\nabla Uatr = \xi d$$

$$\nabla Uatr = \xi((x\text{-}x_a)^2 + (y\text{-}y_a)^2)^{1/2}$$

$$\frac{\partial Uatr}{\partial x} = \frac{\xi(x - x_a)}{((x - x_a)^2 + (y - y_a)^2)^{1/2}}$$

$$\frac{\partial Uatr}{\partial y} = \frac{\xi(y - y_a)}{((x - x_a)^2 + (y - y_a)^2)^{1/2}}$$

$$\nabla Uatr = \frac{\partial Uatr}{\partial x} i + \frac{\partial Uatr}{\partial y} j$$

$$\nabla Uatr = \frac{\xi((x - x_a)i - (y - y_a)j)}{((x - x_a)^2 + (y - y_a)^2)^{1/2}}$$

$$\nabla Uatr = Fatr = \frac{\xi(q - q_a)}{|q - q_a|} \qquad (5.4)$$

Para una compresión visual de la ecuación 5.4 y de los campos potenciales de atracción véase la figura 5.7.

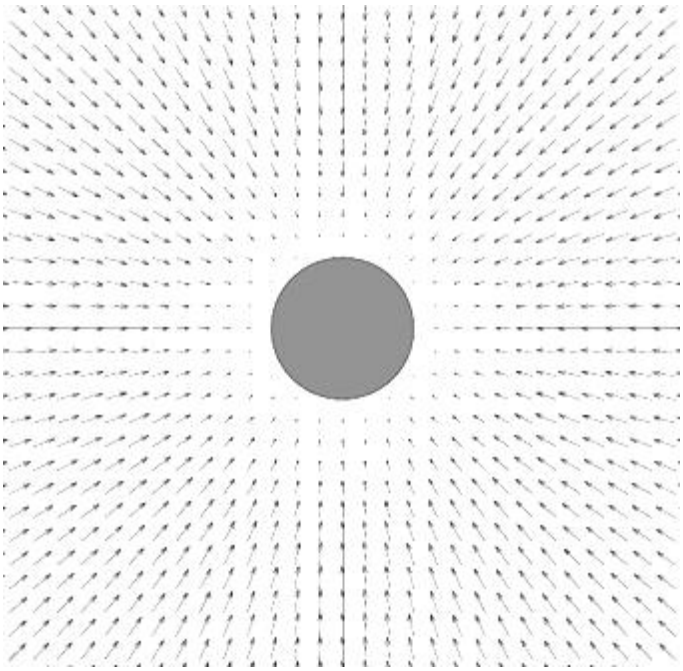


Figura 5.7. Representación de un campo potencial de atracción, donde el punto en el centro indica el punto destino del robot.

(b) Campo potencial de repulsión $$Urep = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{d}-\frac{1}{d_0}\right)^2 & si\ d \le d_0 \\ 0 & si\ d > d_0 \end{cases}$$

donde:

$d = |q - q_o|$

q es la posición actual del robot

q_o es la posición de un obstáculo

d_0 es la distancia de influencia

η es una constante ajustable

$$Urep = \frac{1}{2}\eta\left(\frac{1}{((x-x_o)^2+(y-y_o)^2)^{1/2}}-\frac{1}{d_0}\right)^2$$

$$\nabla Urep = -\eta\left(\frac{1}{|q-q_o|}-\frac{1}{d_0}\right)\frac{1}{|q-q_o|^2}\left(\frac{q-q_o}{|q-q_o|}\right)$$

$$\frac{\partial Urep}{\partial x} = \eta\left(\frac{1}{((x-x_o)^2+(y-y_o)^2)^{1/2}} - \frac{1}{d_0}\right)\left(\frac{-1}{2}\left(\frac{2(x-x_o)}{((x-x_o)^2+(y-y_o)^2)^{3/2}}\right)\right)i$$

$$\frac{\partial Urep}{\partial y} = \eta\left(\frac{1}{((x-x_o)^2+(y-y_o)^2)^{1/2}} - \frac{1}{d_0}\right)\left(\frac{-1}{2}\left(\frac{2(y-y_o)}{((x-x_o)^2+(y-y_o)^2)^{3/2}}\right)\right)j$$

$$\nabla Urep = \frac{\partial Urep}{\partial x}i + \frac{\partial Urep}{\partial y}j$$

$$\nabla Urep = -\eta\left(\frac{1}{((x-x_o)^2+(y-y_o)^2)^{1/2}} - \frac{1}{d_0}\right)\left(\frac{(x-x_o)i+(y-y_o)j}{((x-x_o)^2+(y-y_o)^2)^{3/2}}\right)$$

pero:

$$((x-x_o)^2+(y-y_o)^2)^{3/2} = (((x-x_o)^2+(y-y_o)^2)^{1/2})^3$$

$$((x-x_o)^2+(y-y_o)^2)^{3/2} = (((x-x_o)^2+(y-y_o)^2)^{1/2})^2((x-x_o)^2+(y-y_o)^2)^{1/2}$$

$$((x-x_o)^2+(y-y_o)^2)^{3/2} = |q-q_o|^2 + |q-q_o|$$

entonces:

$$\nabla Urep = Frep = -\eta\left(\frac{1}{|q-q_o|} - \frac{1}{d_0}\right)\frac{1}{|q-q_o|^2}\left(\frac{(q-q_o)}{|q-q_o|}\right) \qquad (5.5)$$

Un ejemplo visual de los campos potenciales de atracción y repulsión es la figura 5.8, donde se observa el campo potencial alrededor de un obstáculo, un campo potencial en el punto destino del robot y la trayectoria que realiza el robot para llegar a éste. En el caso de los campos potenciales en la Robótica Evolutiva, las constantes ajustables, tales como ξ y η son codificadas en un cromosoma para ser evolucionadas por un algoritmo genético, logrando que el robot tenga movimientos más suaves al pasar por un objeto y por un punto de atracción.

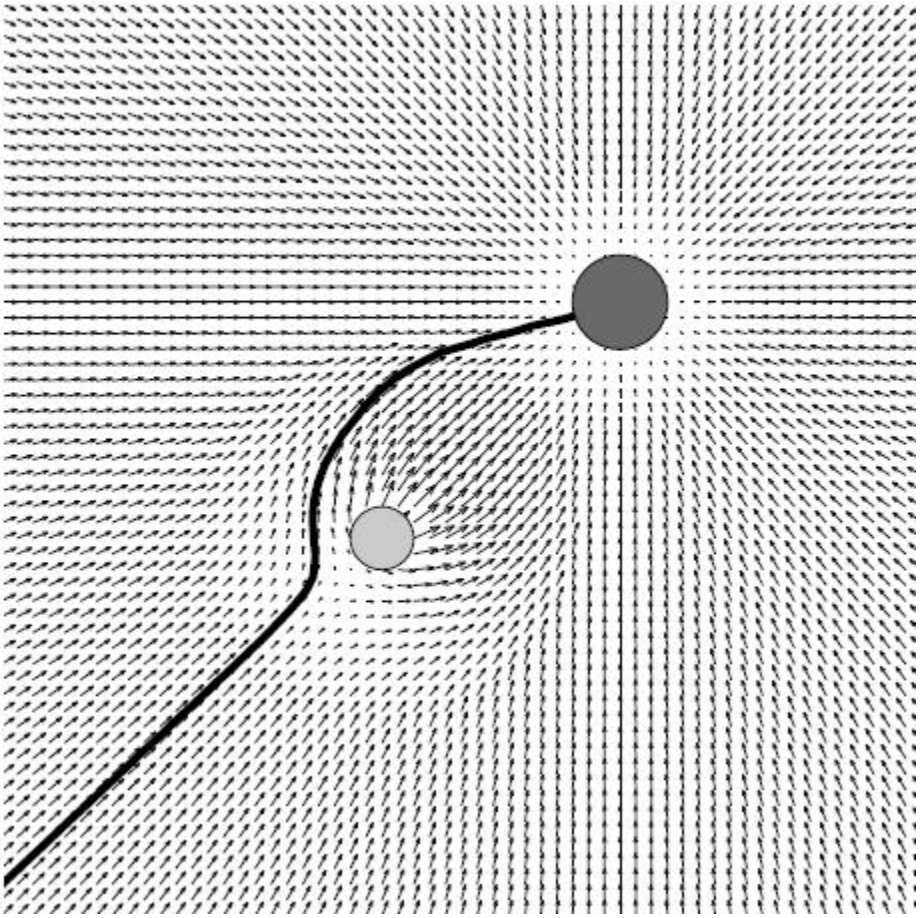


Figura 5.8. Se observa el campo potencial de repulsión alrededor del punto gris claro, que es un obstáculo, el campo potencial de atracción alrededor del punto gris oscuro, que es el punto destino del robot, y la trayectoria que recorre el robot para llegar a éste.

5.4 Métodos de Organización de Comportamientos

Como ya se había mencionado, en la robótica basada en comportamientos una tarea es dividida en un número de comportamientos básicos o simples por el diseñador y cada comportamiento básico es implementado en una capa por separado dentro del sistema de control del robot.

El sistema de control se construye incrementalmente capa por capa donde cada una de éstas es responsable de un comportamiento básico, como por ejemplo, el comportamiento de esquivar obstáculos.

Cabe mencionar que la relevancia de cada comportamiento variará en el tiempo. En situaciones normales el robot puede participar en cualquier tarea compleja que le sea designada; sin embargo, si ocurre una situación de riesgo para el robot, tal como un choque con un vehículo, entonces los comportamientos más básicos relacionados con mantener la

integridad del robot (tal como esquivar obstáculos o escapar) obtendrán de inmediato la más alta prioridad, por lo que se debe contar con un sistema para la organización de comportamientos que sea capaz de reasignar prioridades a los distintos comportamientos con que se cuenta de manera dinámica.

Otra razón de contar con un árbitro de comportamientos es que al tener la organización de varios comportamientos simples, se logra obtener un comportamiento global complejo, con metas a un plazo más largo que con los comportamientos simples por separado.

Existen dos arquitecturas de organización de comportamientos: la monolítica y la modular que se describen a continuación [26].

5.4.1 Arquitecturas monolíticas

Este tipo de arquitecturas considera realizar el comportamiento global en un único módulo que incluye todas las correspondencias entre los sensores y los actuadores; esto es, todos los comportamientos simples se fusionan en un único módulo de control, sea una red neuronal, una máquina de estados, u otro sistema de control.

La ventaja de considerar un único módulo es que no es necesario tener un conocimiento a priori sobre los posibles comportamientos simples y sobre la forma de organizarlos, sólo se tiene un módulo de control que intenta hacer todo lo que se requiera que haga el robot.

Claro que esto tiene grandes desventajas. Una de ellas es que no es posible reutilizar los comportamientos simples, (ya que no existen como tales); es decir, si se necesita un nuevo comportamiento porque se requiere una modificación en lo que podría ser un sub-comportamiento, sería necesario volver a diseñar el módulo de control.

Otra desventaja es que al no tener comportamientos simples, el sistema no puede crecer sin tener que diseñar, de nuevo, todo el módulo de control, o sea, no se le puede agregar un comportamiento simple individual más al módulo de control, ya que no está diseñado por capas de comportamientos simples individuales.

La desventaja más grande, a partir de las dos anteriores, es que al no tener comportamientos simples individuales, no se puede tener un árbitro de comportamientos, lo que trae dos puntos en disfavor de esta arquitectura: el diseño del módulo de control se hace mucho más complejo y el resultado de la interacción del robot en el ambiente no será tan eficiente como tener los comportamientos por separado y contar con un árbitro de comportamientos.

5.4.2 Arquitecturas modulares

En este tipo de arquitecturas se tienen comportamientos simples, cada uno codificado en un controlador, que al ser coordinados por un árbitro de comportamientos, generan un comportamiento más complejo.

Dentro de este tipo de arquitectura existen dos esquemas de organización de comportamientos: la jerárquica o centralizada, donde se cuenta con módulos de control de nivel más altos que los módulos de control de los comportamientos simples y deciden qué módulos de comportamientos se ejecutan en cada situación particular; y la distribuida, donde no existen jerarquías y todos los módulos de comportamientos compiten por el control de los actuadores del robot en cada instante.

5.4.2.1 Arquitecturas modulares jerárquicas

En este tipo de arquitecturas, el comportamiento global se descompone en comportamientos de menor nivel, que se codifican sobre controladores particulares. Los controladores de alto nivel pueden recibir información de los sensores o de los controladores de bajo nivel y decidir entre actuar directamente sobre los actuadores o seleccionar un comportamiento simple y activarlo.

La ventaja de este tipo de arquitecturas es que los comportamientos se pueden obtener individualmente, estableciéndose posteriormente la interconexión entre éstos. Además, es posible reutilizar los comportamientos obtenidos al realizar comportamientos de nivel superior.

El problema que se presenta es que la descomposición no es clara en todos los casos, dado que implica tener conocimiento de qué comportamientos simples pueden ser útiles y necesarios. Otro problema es que el diseñador debe especificar las interconexiones entre los comportamientos, lo que conlleva un potencial problema de *explosión de complejidad*.

5.4.2.2 Arquitecturas modulares distribuidas

En el caso de las arquitecturas modulares distribuidas no existe una jerarquía entre los comportamientos simples, y todos los comportamientos compiten por el uso de los actuadores por medio de algún arbitraje.

Su uso ha sido muy poco frecuente porque en la mayoría de los trabajos realizados no está claro cómo expandir un diseño existente para incorporar otro comportamiento simple, en términos del arbitraje de los comportamientos.

El árbitro de comportamientos es usualmente diseñado a través de un proceso de prueba y error. Es importante destacar que el número de capas incrementa con la complejidad del problema y para una tarea muy compleja, el diseño del árbitro de comportamientos podría superar la capacidad de cualquier diseñador.

El árbitro de comportamientos desarrollado en este trabajo de tesis se basa en una arquitectura jerárquica, ya que el árbitro de comportamientos está a un nivel arriba de los comportamientos simples. Una ventaja de este modelo es que dada la sencillez del arbitraje es fácil expandir el diseño existente para incorporar otro comportamiento simple. En el siguiente apartado se detallará sobre este árbitro.

5.4.3 Trabajos previos

En este apartado veremos algunos trabajos donde se ha implementado un árbitro de comportamientos, ya sea hecho a mano o evolucionado mediante un algoritmo evolutivo.

Empecemos por el árbitro de comportamientos más famoso y que fue el primero en su tipo, este árbitro le abrió el camino a la robótica basada en comportamientos, la **Arquitectura de Subsumpción**.

La arquitectura de subsumpción de Brooks es un ejemplo de las arquitecturas distribuidas [3]. Todos los comportamientos reciben información de los sensores y pueden mandar señales eléctricas directamente a los actuadores. Los comportamientos están organizados por capas, donde en la capa más baja se encuentra el comportamiento simple individual más básico, y en la capa más alta se encuentra el comportamiento simple individual más complejo.

Los comportamientos se interconectan mediante una red de inhibiciones, de forma que los comportamientos de un determinado nivel pueden inhibir los comportamientos de niveles inferiores.

Una característica importante de esta arquitectura es que es incremental en el sentido de que los comportamientos de mayor nivel se obtienen introduciendo nuevos módulos e interrelacionándolos con los existentes, de modo que los comportamientos de mayor nivel subsumen a los de menor nivel.

Figura 5.9. Robot Genghis, uno de los utilizados por Rodney Brooks en sus trabajos iniciales sobre arquitecturas subsumidas en el MIT. Actualmente está retirado y vive en el *Smithsonian Air and Space Museum*. http://groups.csail.mit.edu/lbr/genghis/.

De acuerdo a [12] Brooks creó esta arquitectura teniendo en cuenta las siguientes suposiciones.

- Un comportamiento complejo no necesariamente tiene que ser producto de un sistema de control complejo, más bien puede ser visto como el reflejo de entornos complejos.

- En este tipo de diseño la comunicación de las interfaces es importante. Al estar trabajando con un diseño incremental se sugiere que si se encuentran dificultades en algún momento, se regrese y se cambien los módulos, reduciéndolos o agrandándolos.

- Se desea construir robots baratos que puedan vagabundear en espacios humanos, pero sin la intervención de éstos.

- El mundo es tridimensional, no es sólo un mapa de dos dimensiones, por lo que el robot debe ser capaz de modelarlo en tres dimensiones.

- Si realmente deseamos que los robots nos ayuden en tareas humanas, no se deben construir mundos artificiales perfectos para los robots.

- Aunque los datos de los sensores de ultrasonido se obtienen fácilmente, no representan una fuente rica para describir el mundo, siendo los datos visuales mejores para este propósito. El sonar debe utilizarse en las interacciones de los niveles bajos, para evitar obstáculos en tiempo real.

- El robot debe ser capaz de darse cuenta cuando un sensor falla o comienza a dar lecturas erróneas y debe ser capaz de recuperarse rápidamente.

- Estamos interesados en fabricar robots que puedan *sobrevivir* por días, semanas y meses sin la asistencia humana, en entornos complejos y dinámicos, tales robots deben ser capaces de mantenerse a sí mismos.

Para ejemplo de esta arquitectura véase la figura 5.10, donde se muestran la interconexión de los comportamientos de un robot que puede *vagabundear* por el entorno, conectarse a la alimentación cuando se le esté acabando la batería, seguir luz y evitar obstáculos.

En esta arquitectura se cuenta con cinco comportamiento: “recargar”, “escapar”, “evitar obstáculos”, “seguir luz” y “avanzar”. Cada comportamiento puede activarse dependiendo de los valores de los sensores que le llegan. También, comportamientos como “recargar” pueden inhibir a otros comportamiento de menor nivel como “avanzar”, esto indicaría que el robot tiene poca batería y ha encontrado el sitio de recarga, indicado con una fuente de luz, entonces, el comportamiento de “recargar” inhibe al comportamiento de “avanzar” para que el robot se detenga y recargue su batería.

Una gran desventaja de esta arquitectura es el problema de la complejidad que se le presenta al diseñador al tener que definir la red de inhibiciones y activaciones, respetando

las ya realizadas si las hubiera, tarea nada sencilla a medida que el comportamiento global se hace más complejo.

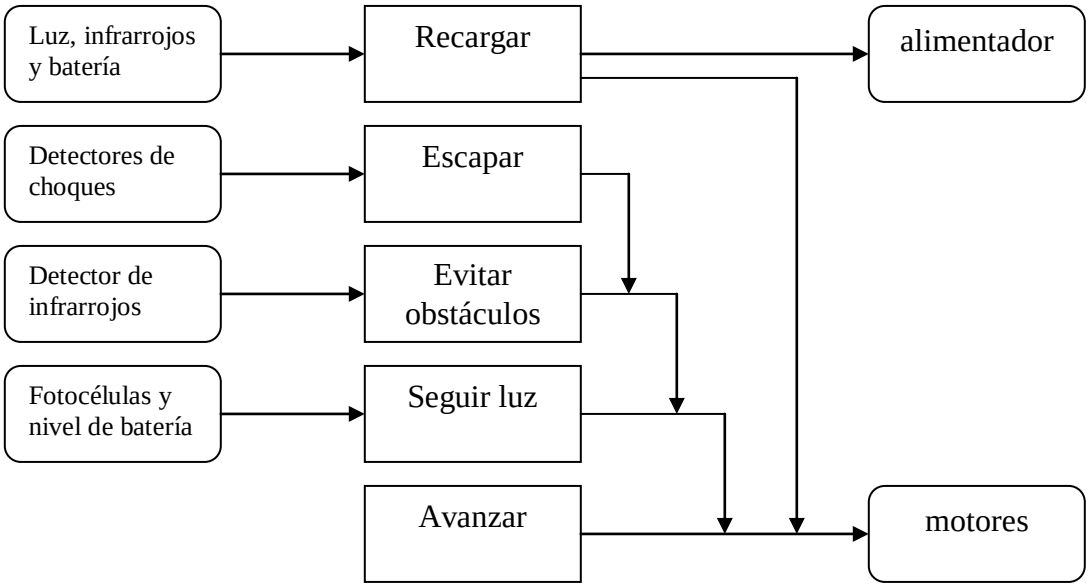


Figura 5.10. Ejemplo de una arquitectura subsumida. Se cuenta con cinco comportamientos, cada uno codificado en un controlador individual. Las flechas que salen de un determinado comportamiento *a* y llegan al comportamiento *b* indican que el comportamiento *a* subsume al comportamiento *b*.

Ahora veamos un método de arbitraje que cuenta con un modelo matemático para encontrar, dados los valores de sensores, cuál es el comportamiento óptimo a ejecutar en un instante de tiempo *t*. Este método se conoce como **Método de Función de Utilidad** [33], desarrollado en la Universidad de Tecnología de Chalmers en Suecia por Mattias Wahde.

En este método, la selección del comportamiento a ejecutar está basada en el valor de funciones de utilidad, que son evolucionadas mediante un algoritmo genético, en lugar de diseñarlas a mano.

Cada comportamiento B_j, $j = 1, ..., N$, donde N es el número de comportamientos, es asociado con una función de utilidad, cuyas variables son (un subconjunto de) las variables de estado del robot.

Las variables de estado del robot son de tres tipos: variables externas, denotadas por s_i, un ejemplo de estas variables son las lecturas de los sensores infrarrojos del robot; variables físicas internas, denotadas por p_i, un ejemplo de estas variables son las lecturas del sensor de batería; y las variables internas abstractas, denotadas por x_i, estas variables corresponden a lecturas de variables internas conocidas como hormonas bajo este método de organización de comportamientos.

En general, cada función de utilidad es dada por un polinomio estimado. Por ejemplo, el estimado para un polinomio de una función de utilidad de segundo grado de dos variables s_1 y x_1 es dado por:

$$U(s_1, x_1) = a_{00} + a_{10}s_1 + a_{01}x_1 + a_{20}s_1^{\,2} + a_{11}s_1x_1 + a_{02}x_1^{\,2}$$

Bajo este método, uno y sólo un comportamiento está activo a un momento dado. La selección de comportamientos en este método es sencilla: a todo momento, el comportamiento cuya función de utilidad tenga el valor más alto es el que se activa.

El problema aquí es definir las funciones de utilidad para cada comportamiento para generar una selección de comportamientos útil y confiable. Los investigadores que diseñaron este método utilizan algoritmos evolutivos para obtener las funciones de utilidad a partir de los estimados, tal como el estimado anterior.

El fitness es asociado con la ejecución de un comportamiento que desempeñe una tarea, mientras a los otros comportamientos se les toma como auxiliares, o sea, comportamientos que son necesitados, como el comportamiento de recarga de batería, pero que no incrementan el fitness del robot.

Una vez que se ha especificado la función de fitness, la tarea del algoritmo evolutivo es encontrar la combinación óptima de los coeficientes de los *N* polinomios de las funciones de utilidad, cada una de estas funciones correspondiente a un comportamiento.

El siguiente método de selección de comportamientos a tratar es uno desarrollado por investigadores de la Universidad de Osaka, llamado **selección de comportamientos acelerados por restricciones de activación / terminación** [30].

En este método se evolucionan parámetros de una función que indica la utilidad de cada comportamiento. Aparte, se aumenta cada comportamiento al añadirle restricciones de activación / terminación para acelerar el proceso evolutivo.

Las restricciones de activación sirven para reducir el número de situaciones en las cuales cada comportamiento es propenso a ser ejecutado. Las restricciones de terminación sirven para extraer secuencias de comportamientos útiles, donde cada una de éstas puede ser considerada como una sola acción, sin importar el número de comportamientos ejecutados que contenga. En otras palabras, la restricción de activación en un comportamiento indica si éste es ejecutable en el estado actual, mientras que la restricción de terminación indica la probabilidad de continuar ejecutando el comportamiento que ha sido seleccionado.

Estos investigadores aplican un algoritmo genético para obtener los parámetros óptimos de una función de conmutación que indique el mejor comportamiento a ejecutar de acuerdo al estado actual y para establecer la probabilidad de terminación del comportamiento en ejecución. Las restricciones de activación / terminación son codificadas en el cromosoma, pero no son tomadas en cuenta para la cruza ni la mutación.

De la figura 5.11, se observa un ejemplo de este método de selección donde la capa de arriba aprende la función de conmutación para seleccionar el comportamiento adecuado ya diseñado o aprendido. Bajo esta aproximación, este método no sólo sirve para coordinar y conmutar los comportamientos, sino también para seleccionar de manera adecuada los comportamientos y terminar el comportamiento en ejecución en el tiempo adecuado.

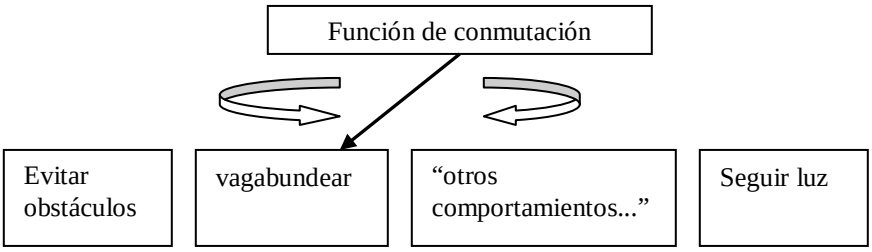


Figura 5.11. Arquitectura para selección de comportamientos mediante una función de conmutación.

El último método de organización de comportamientos a tratar es el **NEAT con módulos** [15]. Este método fue desarrollado por investigadores del Instituto de Investigación Informática de la Universidad de La Plata, Argentina, a partir del método NEAT, que es un método de evolución de redes neuronales.

El método de NEAT con módulos se basa en tener comportamientos simples codificados en redes neuronales cada uno, después se fusionan o mezclan estas redes neuronales para crear una única red neuronal que contenga las redes neuronales de los comportamientos simples y así obtener un único controlador capaz de decidir qué comportamiento ejecutar en qué momento.

La comunicación entre las redes neuronales fusionadas es establecida por evolución, lo que da como resultado una sola red neuronal que contiene el árbitro de comportamientos y a los comportamientos o módulos.

Dado que las tareas resueltas por cada comportamiento son parte de una tarea compleja a resolver por la red neuronal unificada, se espera que más de un módulo use las mismas entradas en la red neuronal unificada que cuando era un módulo individual, o que produzca la misma salida.

La red neuronal unificada tendrá como entrada la unión de las entradas de cada módulo. Los módulos se conectan a esas entradas sin ninguna modificación. Las salidas de la red neuronal unificada depende de la tarea a resolver y es por esto que la red tendrá tantas neuronas de salida como necesite el problema.

Más de un módulo puede generar la misma salida de la red, también es posible que diferentes módulos produzcan estímulos opuestos para entradas similares, ya que las tareas resultantes de estos comportamientos pueden ser contradictorias. Para permitir a la evolución ajustar la contribución de cada módulo a las salidas de la red neuronal unificada,

premiando las respuestas esperadas y haciendo compatibles estímulos opuestos, las neuronas de salida de cada módulo se convierten en neuronas escondidas.

Se añade una conexión a cada una de estas neuronas ahora escondidas que las conecte con la neurona de salida que produce la respuesta que fue originalmente producida por la antigua neurona de salida, ahora escondida. La conexión se establece con un peso de 1.0, así los estímulos originales alcanzan la neurona de salida sin ser afectados. Esta nueva conexión no es considerada como parte de ningún módulo, sino que pertenece a la red neuronal unificada. La figura 5.12 muestra un ejemplo donde se fusionan dos redes neuronales codificando cada una un comportamiento simple, el comportamiento de esquivar obstáculos y el comportamiento de encontrar luz.

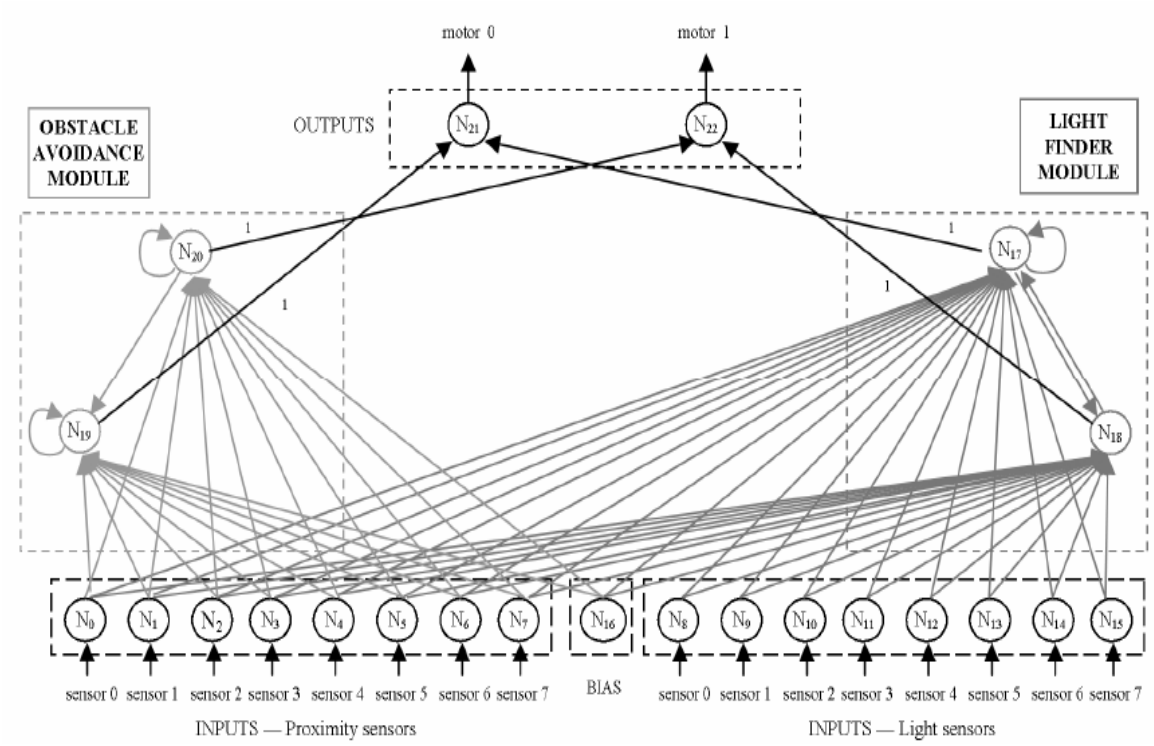


Figura 5.12. Red neuronal unificada obtenida después de combinar dos módulos con algunas neuronas de entrada en común [15].

5.5 Red Neuronal como Árbitro de Comportamientos

En el apartado anterior se mostraron cuatro árbitros de comportamientos, cada uno con la complejidad de programación que conllevan. En este apartado se mostrará el árbitro de comportamientos que se ha diseñado, con el objetivo de navegación, bajo aprendizaje supervisado, y contando con un mapa de activación de árbitros de comportamientos.

Actualmente no se ha logrado que un robot móvil logre navegar en cualquier ambiente, sin poseer un mapa de éste, sin chocar, sin quedarse atorado, y con movimientos suaves, es un problema abierto y que se ha acotado a ciertas circunstancias para tener éxito.

Por ejemplo, hay investigadores que evolucionan controladores para un ambiente en específico, o para dos ambientes, pero el grado de generalidad no es grande, siempre son casos acotados.

La metodología aquí propuesta para desarrollar el árbitro de comportamientos es la siguiente. Primero evolucionar redes neuronales multiperceptrón que codifiquen comportamientos simples de navegación para un ambiente, por ejemplo, evolucionar una red neuronal que codifique el comportamiento de esquivar obstáculos, evolucionar una red neuronal que codifique el comportamiento de atravesar un pasillo estrecho, etc. Se pueden tener de igual manera máquinas de estados que codifiquen comportamientos simples de navegación, no es forzoso contar siempre con redes neuronales para este propósito, de la misma manera, no es forzoso que las redes neuronales que codifican comportamientos simples sean evolucionadas.

Las redes neuronales tienen dos neuronas de entrada, cada una es un promedio de 341 sensores láser, una neurona de umbral, una capa escondida y dos neuronas de salida, donde una neurona le indica el avance, en decímetros, que debe tener el robot y la otra el ángulo de giro.

Una vez que se tienen comportamientos simples funcionando es momento de evolucionar el árbitro de comportamientos. El árbitro es una red neuronal que, al igual que las redes neuronales que codifican comportamientos simples, tiene dos neuronas de entrada, donde cada una recibe un promedio de 341 sensores láser, una neurona de umbral, una capa escondida y tiene una sola neurona de salida, la neurona de salida indica qué comportamiento simple es el que se ejecuta en ese instante.

Para una visualización gráfica de este árbitro véase la figura 5.13. Este árbitro es de tipo jerárquico, ya que el árbitro está un nivel arriba y decide qué comportamiento es el que se activa en cierto instante. A cada movimiento que el robot da, la red neuronal árbitro va seleccionando el comportamiento a ejecutarse, su decisión se basa en la señal de corrección que le proporciona la función de fitness.

De la figura 5.13 se puede apreciar que se podría tener *n* redes neuronales en conjunto con *n* máquinas de estados como comportamientos simples individuales de navegación, y podría haber *n* comportamientos más codificados bajo otra estructura de control. Si la red árbitro escogiese la red neuronal *RN1* entonces se le pasan los promedios de los sensores láser (mismas que el árbitro de comportamientos tuvo como entradas) a esta red neuronal para que tome el control de los motores del robot.

Uno podría preguntarse cuál es el criterio de la red neuronal para escoger a un comportamiento. La respuesta es que se obtiene una serie de señales de corrección que van indicando el desempeño de cada red neuronal árbitro, y es a través de la *evolución* que se

obtiene una red neuronal que se adapta al ambiente para lograr el objetivo propuesto. La metodología de entrenamiento de esta red, es la siguiente.

En un simulador se le indica al robot que vaya de un punto *a* del ambiente a un punto *b*. Se tiene inicialmente una población de cromosomas donde cada una codifica los pesos de una red neuronal. Se deja que cada red neuronal controle al robot y éste interactúe con el ambiente y después de cierto número de movimientos permitidos al robot, se mide la distancia que le faltó para llegar al punto *b*; éste es el *fitness*.

Después de que todas las redes neuronales controlaron al robot, se procede a hacer la cruza y mutación, con lo que se obtiene una nueva población de controladores y el proceso se repite. Nótese que es un aprendizaje supervisado, ya que aunque no se le proporciona a ningún controlador las salidas deseadas, el fitness actúa como supervisor del robot indicándole si *va bien o va mal*. Nótese también que es por prueba y error como la red neuronal árbitro aprende qué comportamiento activar en qué momento.

Una vez descrita la manera en que funciona el árbitro de comportamientos propuesto pasemos a ver las ventajas y desventajas de esta metodología de organización de comportamientos.

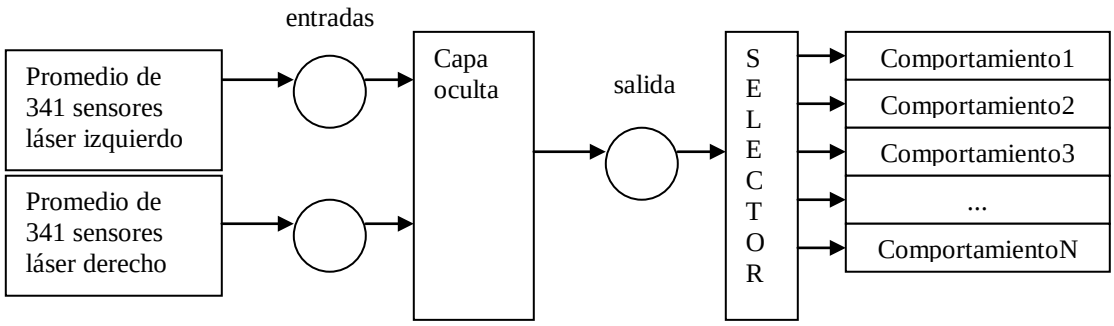


Figura 5.13. Red neuronal como árbitro de comportamientos. La salida de la red elige que comportamiento simple de navegación es activado.

Una clara ventaja es la sencillez del controlador, ya que es una red neuronal multiperceptrón que resulta sencilla de programar. Otra ventaja es que se pueden tener *n* comportamientos para ser organizados y se pueden agregar más, basta con a) escalar la salida del árbitro a un valor adaptado al número de controladores agregados más los *n* que ya se tenían, b) realizar otra evolución.

Una ventaja significativa es que cada comportamiento puede ser codificado en un sistema de control distinto, por ejemplo máquinas de estados, redes neuronales, y no hay problema de compatibilidad, incluso se pueden poner acciones simples para ser activadas, tales como avanzar *x* distancia, girar *y* grados.

La mayor de las ventajas, y que no se ve tan sencillo en algún árbitro de comportamientos de los descritos, es que el nivel de arbitraje puede crecer sin que haya mayores problemas de complejidad de diseño ni de programación; esto es, puede existir un controlador un nivel arriba de la red neuronal árbitro y entonces se tendría un árbitro de árbitros. Mientras crezca el número de niveles no crecerá exponencialmente el problema de complejidad, ya que siempre será una red neuronal controlando redes neuronales, el mismo diseño planteado.

Ahora, véase lo siguiente. Al tener un árbitro de comportamientos lo que se logra es organizar comportamientos para un ambiente. Supóngase que se hace esto para varios ambientes. El siguiente nivel sería la organización de árbitros de comportamientos, lo que se traduce en que si el robot es posicionado en un entorno *k* donde fue anteriormente evolucionado el controlador, entonces el árbitro de árbitros tiene la capacidad de seleccionar el árbitro de comportamientos de ese ambiente *k* de entre otros árbitros de otros ambientes.

Nótese cómo esta metodología permite una escalabilidad y flexibilidad en el diseño, en la programación y que es propensa a resolver el problema de navegación en ambientes semi-estructurados, ya que mientras el nivel de organización crezca, crecerá la flexibilidad de navegación en ambientes similares.
El resultado de esta metodología de trabajo podría derivar en crear bases de datos de comportamientos y de árbitros para distintos tipos de ambientes, y cuando se necesite evolucionar un árbitro a cualquier nivel de abstracción en un ambiente nuevo, bastará con incluir los árbitros y comportamientos simples previamente evolucionados.

Una alternativa más a la metodología de seleccionar comportamientos simples podría realizarse con un Sistema Experto[2], en donde se enmarca en un programa los conocimientos y experiencia del diseñador que podrían complementarse con algún mecanismo de razonamiento; claro que el nivel de éxito en la ejecución de esta tarea dependería de la pericia y experiencia del diseñador.

En los siguientes capítulos se abordará en detalle los temas correspondientes a algoritmos genéticos y la evolución de este árbitro de comportamientos.

[2] Un Sistema Experto (SE) es un sistema basado en computadora que integra bases de datos, memorias, mecanismos de razonamiento, agentes, algoritmos, heurísticas, para adquirir, representar, almacenar, generar y difundir conocimientos inicialmente adquiridos a través de varios expertos humanos dentro de un dominio especifico llamado “nube”. Con un Sistema Experto, se pueden dar recomendaciones y/o tomar acciones en las áreas de análisis, diseño, diagnóstico, planeación y control o dar solución a problemas o aplicar técnicas de enseñanza o en general recomendar, actuar y explicar las acciones que hay que tomar en actividades en las cuales normalmente, se requiere del conocimiento o saber de expertos humanos dentro de una nube especifica [13].

Capítulo 6

ROBÓTICA EVOLUTIVA

Como se vio en los capítulos anteriores, la robótica basada en comportamientos basa su aproximación en que el robot no debe tener un modelo del mundo, sino interactuar con éste para conocerlo. Esta interacción se da en términos de comportamientos.

Anteriormente se observó que existen diferentes formas de codificar los comportamientos, diferentes formas de organizarlos para hacer de un conjunto de comportamientos una forma más eficiente de lograr una tarea por parte del robot.

Sin embargo, esta aproximación cuenta con varios problemas de eficiencia, complejidad, efectividad. De acuerdo a [26] los siguientes son los más comunes e importantes:

- El diseño de mecanismos que relacionen los sensores y los actuadores y que lleven a un nivel de coordinación capaz de generar un comportamiento autónomo en el entorno real no es sencillo. De acuerdo a [26] Phil Husbands y otros investigadores de su grupo de trabajo de la Universidad de Sussex, indican que esto se debe al hecho de que los sistemas de control requeridos corresponden a sistemas dinámicos complejos difíciles de diseñar a mano.

- El proceso, bajo ciertas arquitecturas, de conectar los diferentes módulos de comportamientos se vuelve muy complejo por el hecho de que las interacciones necesarias son difíciles de seguir.

- De acuerdo a Inman Harvey, investigador de la Escuela de Ciencias Cognitivas y Computacionales de la Universidad de Sussex, no es claro en muchos casos, cómo debería descomponerse el sistema de control de un robot [8].

- Otros puntos que indica Harvey es que las interacciones entre módulos individuales no se limitan a meras conexiones entre ellos, sino que también incluyen interacciones mediadas vía el ambiente. Aparte, mientras crece la complejidad de un sistema, el número de interacciones potenciales entre módulos crece exponencialmente.

- De acuerdo a [26] Cliff menciona que la complejidad en un diseño aumenta más rápido que el número de partes o módulos que lo conforman, de hecho, aumenta en proporción al número de posibles interacciones entre ellos.

- Además, los comportamientos diseñados por un humano no son necesariamente los mejores para la tarea requerida, ya que el diseñador está interpretando lo que el robot percibe y cómo debería actuar, y probablemente no tomará en cuenta adecuadamente las características del robot, como su morfología, capacidades y recursos, que podrían llevar a la realización de la tarea de un modo más fácil y conseguir un comportamiento eficiente.

- De acuerdo a Nolfi, Floreano, Miglino y Mondada, investigadores del Instituto de Psicología en Italia, del Laboratorio de Tecnología Cognitiva en Italia, del Departamento de Psicología de la Universidad de Palermo en Italia y del Laboratorio de Microcomputación del Instituto Federal Suizo de Tecnología en Suiza, respectivamente, es extremadamente difícil coordinar las partes de un robot, tanto a nivel mecánico como de sistemas de control, así como también es difícil predecir la interacción resultante entre estos dos niveles [20].

- Otro punto que destacan los anteriores investigadores es que los robots autónomos interactúan con un ambiente externo, y por tanto, la manera en que éstos se comportan en el ambiente determina los estímulos que recibirán como entrada. Cada acción de un motor tiene dos efectos. Primero, determina qué tan bien se desempeña el sistema con respecto a la tarea asignada. Segundo, determina el siguiente estímulo de entrada que será captado por el sistema (este último punto afecta de manera significativa el éxito o fracaso de una secuencia de acciones). Determinar la acción correcta de un motor que el sistema debería desempeñar para obtener estímulos de entrada buenos, es extremadamente difícil, ya que cualquier acción de un motor puede tener consecuencias a largo plazo.

Considerando estos problemas descritos, parecería razonable utilizar un procedimiento automático que permita construir incrementalmente sistemas de control complejos para los robots, tal como un algoritmo genético, que construya gradualmente el sistema de control de un robot al explotar las variaciones en las interacciones entre el ambiente y el robot.

El algoritmo genético codificaría en un cromosoma al sistema de control y lo evolucionaría, ya sea la arquitectura entera del sistema de control o algunas características de éste que permitan mejorar la eficiencia del comportamiento en él codificado. A esta metodología se le conoce como Robótica Evolutiva.

La Robótica Evolutiva se basa fundamentalmente en algoritmos evolutivos, tales como los Algoritmos Genéticos, la Programación Genética, las Estrategias Evolutivas. El enfoque que se dará aquí se basa exclusivamente en Algoritmos Genéticos.

En la Robótica Evolutiva se pretende evolucionar un controlador adecuado para un robot, dependiendo de las situaciones del ambiente. En un ambiente dinámico e impredecible, el

controlador de un robot no siempre puede ser diseñado con anticipación, pero se podría obtener un controlador adecuado a un cierto ambiente usando algoritmos genéticos, o sea, evolucionándolo. El controlador obtenido puede ser una solución aceptable para los problemas del mundo en tiempo real.

En los siguientes apartados se detallará el algoritmo genético utilizado, también cómo se evolucionaron controladores individuales para comportamientos simples de navegación, y por supuesto, la evolución del árbitro de comportamientos.

6.1 Estado del arte

Actualmente no existe una metodología bien establecida para evolucionar sistemas de control para robots autónomos [20]. Cada grupo de investigación ha aportado su propia metodología como contribución a la Robótica Evolutiva, la mayoría de ellas muy parecidas, variando aspectos como el algoritmo genético utilizado, evolución en robot real o en simulador, el tipo de robot utilizado, entre otras. En este trabajo de tesis se adopta un metodología propia, aunque muy parecida a todas las demás metodologías.

Se ha intentado desarrollar controladores que permitan la navegación de un robot móvil en ambientes no estructurados y sin dotarle de un modelo del entorno, o sea, se ha intentado la navegación bajo la perspectiva de la robótica basada en comportamientos.

No se ha conseguido obtener un controlador o serie de controladores que permitan que un robot vagabundee por *n* entornos distintos sin chocar, sin quedarse atorado y con movimientos suaves, actualmente es un problema abierto. Se ha intentado atacar con todo tipo de controladores, desde máquinas de estados, lógica borrosa, campos potenciales, redes neuronales y hasta con algunas combinaciones de éstos, pero lo más que se ha logrado son casos específicos.

Como ya se mencionó en el apartado de definición del problema, este trabajo de tesis no pretende resolver el problema general de navegación de un robot móvil autónomo, sino acotarlo a un ambiente.

El principal objetivo de esta tesis es mostrar que el árbitro de comportamientos propuesto funciona bien y tiene las ventajas antes mencionadas sobre los demás árbitros ya detallados con anterioridad. También pretende que esta metodología sea, en adelante, trabajada con más detalle para llegar a una solución más general del problema de navegación.

6.2 Trabajos previos

En un resumen hecho por [26], los siguientes son algunos de los trabajos más representativos de la Robótica Evolutiva. En los últimos años de la década de los 1980 y primeros años de la década de los 1990, algunos investigadores como Inman Harvey, Phil Husbands y David Cliff de la Universidad de Sussex en Brighton, Inglaterra y Randall Beer junto con John Gallagher de la *Case Western Reserve University* en Ohio, Estados Unidos, propusieron la evolución artificial como un medio para automatizar el proceso de diseño de los sistemas de control de los robots móviles autónomos.

Otros exponentes de la Robótica Evolutiva son Karl Sims del *MIT Media Lab* y miembro de la empresa GenArts, quien usó la evolución artificial para crear "criaturas virtuales"; Wei-Po de la Universidad de Edimburgo, quien analiza la importancia de un acoplamiento cerebro-cuerpo en el diseño de un sistema robótico mediante algoritmos genéticos.

Craig Reynolds de *Sony Computer Entertainment*, quien evoluciona el número y posiciones de los sensores en un vehículo simulado en dos dimensiones; Angelo Cangelosi de la Universidad de Genova, Italia, quien ha estudiado la evolución de la comunicación entre agentes simulados.

Marco Dorigo y Marco Colombetti de la Universidad de Milán, Italia, quienes utilizan un sistema de clasificación en los controladores de los comportamientos de un robot y usan un algoritmo genético para descubrir nuevas reglas útiles en tiempo de vida del robot.

Dario Floreano del Laboratorio de Tecnología Cognitiva, Italia y Francesco Mondada del Laboratorio de Microcomputación del Instituto Federal de Tecnología de Lausanne, Suiza, quienes evolucionaron controladores para un robot Khepera y lograron reutilizarlos en la evolución de los comportamientos de un robot Koala.

Hitoshi Hemmi de los *ATR Laboratories de Kyoto*, Japón, quien emplea un lenguaje de descripción del hardware para evolucionar programas que producen las estructuras de hardware y los comportamientos requeridos de un robot.

Mattias Wahde de la *Chalmers University of Technology* en Gotemburgo, Suecia, quien estudia la robótica evolutiva y evolucionó el árbitro de comportamientos conocido como Método de Función de Utilidad [33]; Krister Wolff y Peter Nordin de la *Chalmers University of Technology*, quienes diseñaron un robot con alas capaz de aprender técnicas de vuelo mediante algoritmos genéticos.

John R. Koza (1992) de la Universidad de Stanford, quien usó técnicas de programación genética para evolucionar arquitecturas subsumidas. Nótese que esta área de la Robótica parece ser prometedora, no sólo por los resultados que hasta ahora se han obtenido, sino por los esfuerzos que se han sumado de los investigadores que confían en ella.

6.3 Algoritmo Genético utilizado

El AG (Algoritmo Genético) que se utilizó para evolucionar las redes neuronales que codifican comportamientos de navegación y la red neuronal que funge como árbitro, fue una modificación del Algoritmo Genético Ecléctico (EGA) [10].

Este algoritmo genético surge del intento de aproximarse a un algoritmo genético ideal, cuya definición de éste fue dada por Mitchell en [16]. Un AG se aproximará al AG ideal si:

a) Las muestras son independientes. La población debe ser lo suficientemente grande, el proceso de selección debe ser lo suficientemente lento y la tasa de mutación lo suficientemente alta, a fin de que ninguna locus (posición) se fije a un único valor en una gran mayoría de las cadenas (cromosomas) de la población.
b) Los Esquemas[1] deseados son secuestrados. La selección debe ser lo suficientemente fuerte para preservar los esquemas deseados que hayan sido descubiertos pero también lo suficientemente lento para prevenir *hitchhiking*[2] en algunos buenos esquemas.
c) La cruza es instantánea. La tasa de cruza tiene que ser tal que el tiempo para cruza que combina dos esquemas deseados sea pequeño con respecto al tiempo para descubrir dichos esquemas.
d) La cadena es grande. La cadena debe ser larga para que el factor de incremento de velocidad (del algoritmo genético ideal respecto al RMHC: *random mutation variation of a hill climber*, que en términos simples se denomina *escalador*[3]) para encontrar a todos los esquemas deseados sea significante.

Estos mecanismos no son mutuamente compatibles con el AG canónico, por lo que en [10] proponen un AG alternativo que cumpla con éstos, y lo llama **Algoritmo Genético Ecléctico** (EGA). El término "ecléctico" se refiere al intento explícito, de parte de los autores, de lograr un algoritmo que tome lo "mejor" de todas las estrategias, sin considerar la ortodoxia. En el EGA se incorpora lo siguiente:

1) Elitismo completo sobre un conjunto de tamaño *n* de la última población.
2) Un esquema de selección determinístico (opuesto al operador de selección proporcional tradicional).
3) Cruza anular.

---

[1] Un esquema es un conjunto de puntos del espacio de búsqueda que comparte algún patrón común [29], puede verse como una plantilla de similitud que representa a un subconjunto de arreglos con correspondencias en ciertas posiciones dado un alfabeto. Por ejemplo, para el alfabeto binario, se agrega el símbolo * que significa "no importa", podemos crear esquemas sobre el alfabeto {0,1,*}, por ejemplo, sea el esquema 0*1, este esquema contiene a las cadenas 001 y 011.

[2] Término utilizado para denotar uno de los problemas que tienen los AG, en términos formales se denomina correlación espuria, y se refiere al fenómeno que hace que ciertos segmentos de cadena del cromosoma, que por sí mismos no tienen mayor relevancia, sean propagados a través de generaciones debido a su similitud con los segmentos de cadena que tienen aptitudes elevadas, o sea, que proveen de un mejor *fitness* al cromosoma que lo contiene.

[3] Un escalador es un algoritmo que toma una cadena de manera aleatoria. Selecciona un *locus* aleatoriamente para ser mutado. Se evalúa la cadena. Si la mutación lleva a un igual o mejor *fitness*, entonces se guarda dicha cadena. Se repite el proceso de seleccionar un *locus* y mutar hasta que se encuentre una cadena óptima o hasta que se alcance un número de evaluaciones.

4) Invocación selectiva de un RMHC.
5) Auto-adaptación de la población de los siguientes parámetros: el número de hijos, probabilidad de cruza, probabilidad de mutación y probabilidad de invocación del RMHC.

Para este trabajo de tesis, no se consideró la auto-adaptación de la probabilidad de invocación del RMHC ni del número de hijos, y la auto-adaptación de la probabilidad de cruza y mutación tuvo un pequeño ajuste.

6.3.1 Elitismo completo

Por elitismo completo se entiende que se guarda una copia de los mejores *n* individuos hasta la generación *k*, donde una población consta de *n* individuos. En otras palabras, dado que se han evaluado *n*k* individuos (*n* individuos por cada generación) hasta la generación *k*-ésima, la población consistirá de los mejores *n* individuos hasta ese punto.

6.3.2 Selección determinística

En la selección determinística, no se cuenta con un proceso estocástico en base al *fitness* individual para determinar los cromosomas padre. Aquí, se ordena a la población actual de mejor a peor valor de *fitness* y se hace la cruza del individuo *i* con el individuo *n-i+1*. Este esquema de selección se llama *Modelo Vasconcelos*.

Esta estrategia parece destruir las buenas características del mejor individuo al cruzarlo deliberadamente con el peor individuo. Sin embargo, cuando se aplica en conjunción con el elitismo completo, esta estrategia conduce al análisis implícito de una variedad más amplia de esquemas (es decir, maximiza la exploración del paisaje de solución).

Anteriormente, se había dicho que en un AG ideal ningún locus debería fijarse a algún valor en una gran mayoría de cadenas de la población. Esto se logra claramente, ya que, determinísticamente se están rompiendo los locus problemáticos al hacer la cruza del mejor individuo con el peor individuo.

Otro punto a favor, es que al tener elitismo completo se preservan los esquemas deseados y, como antes, se evitan las correlaciones espurias al cruzar individuos disimilares.

Adicionalmente, la tasa de cruza debe ser tal que el tiempo para cruza que combine dos esquemas deseados sea pequeño en relación al tiempo de descubrimiento para dichos esquemas. El elitismo completo garantiza que, aún en la estrategia de Vasconcelos, el peor

individuo en la población *k* está entre los mejores del 1/*k* porciento. Por ejemplo, si *N* = 50 y buscamos en la población 20, los individuos en dicha población están entre los mejores 5% del total de los individuos analizados.

Un último requerimiento es que la cadena sea lo suficientemente larga para que el factor de velocidad asociado al tiempo para encontrar todos los esquemas deseados, sea significante. Podría parecer que para un número pequeño de esquemas un escalador podría funcionar mejor que un AG. Por lo que se incorpora un escalador al EGA.

6.3.3 Cruza anular

En la cruza anular, el genoma ya no es visto como una colección lineal de bits, sino como un anillo cuyo bit más a la izquierda es contiguo al bit más a la derecha. (véase la figura 6.1).

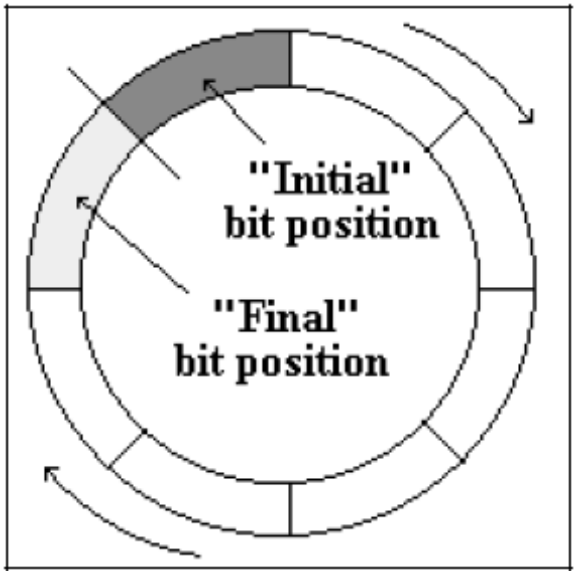


Figura 6.1. Genoma en forma de anillo [10].

Cuando se aplica la cruza anular, hay dos parámetros a considerar para cada intercambio:

a) El locus donde inicia la cruza. Esto es, la posición donde inicia el segmento a extraer del genoma de los padres.
b) La longitud del semi-anillo. Esto es, la cantidad de genes que serán extraídos. Para un genoma de longitud *l* existen *l* posibles locus y *l-1* posibles longitudes.

Véase la figura 6.2 donde se muestra un ejemplo de cruza de dos individuos y el individuo resultante.

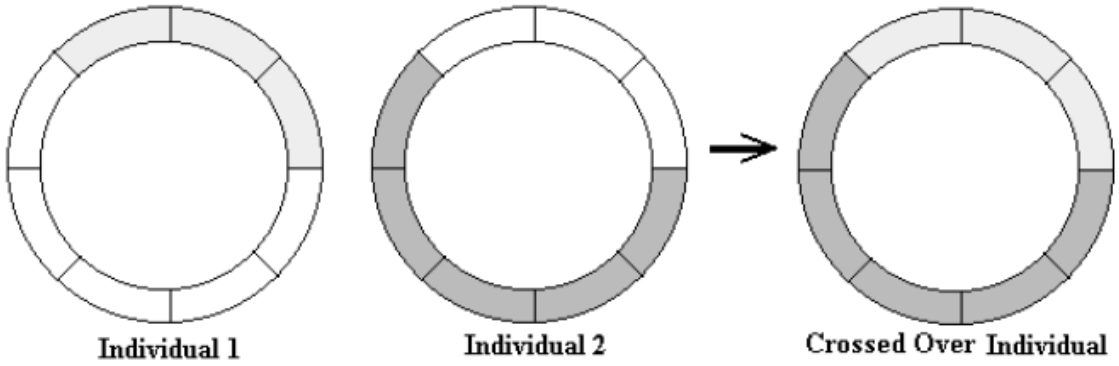


Figura 6.2. Cruza anular entre dos individuos padre y el individuo hijo resultante [11].

6.3.4 Auto-adaptación

Cuando se corre un AG hay varios parámetros que deben inicializarse a *priori*, como son la probabilidad de cruza (P_c), la probabilidad de mutación (P_m), el tamaño de la población (N). En muchos casos, el usuario intenta afinar estos parámetros al hacer un cierto número de corridas sobre diferentes casos "representativos". En el EGA, estos tres parámetros son auto-adaptables y van codificados en el genoma, de manera que los parámetros evolucionan junto con el individuo, aunque como se mencionó antes, una de las modificaciones hechas al EGA en este trabajo, es que el número de individuos no es evolucionado.

La idea detrás de la auto-adaptación es que el AG no solamente explore el paisaje o superficie de soluciones, sino también la superficie de parámetros. De esta manera, el genoma es dividido en dos sub-genomas: un *genoma de estrategia* y un *genoma de solución*. Ambos sub-genomas son sujetos a los operadores genéticos, véase la figura 6.3 donde se muestra el genoma utilizado en este trabajo de tesis.

Probabilidad de cruza	Probabilidad de mutación	Origen del individuo	Codificación del individuo
Pc	P_m	1 -> escalador, 0 -> cruza	

sub-genoma de estrategia	sub-genoma de solución

Figura 6.3. Genoma utilizado para evolucionar la solución y los parámetros del AG.

Cabe mencionar que en este AG se tiene como propósito el optimizar los valores medios de la población, en lugar de optimizar los valores de cada individuo. Esto se refleja en lo que sigue.

6.3.4.1 Probabilidad de cruza

La probabilidad de cruza P_c es codificada en el genoma de cada individuo. Para obtener esta probabilidad en la *k*-ésima generación se hace el siguiente cálculo:

$$(\mathrm{P_c})_k = \frac{1}{N}\sum_{i=1}^{N}(P_c)_i\left(\frac{f_i}{\overline{f}}\right) \qquad (6.1)$$

Lo que este cálculo indica es que hace el promedio ponderado de las probabilidades de cruza, por la proporción del *fitness* del individuo correspondiente respecto al *fitness* promedio de la población. La P_c resultante, será la utilizada en la *k*-ésima generación como la tasa de cruza.

6.3.4.2 Probabilidad de mutación

La probabilidad de mutación P_m es codificada en el genoma de cada individuo. Para obtener esta probabilidad en la *k*-ésima generación se hace el siguiente cálculo:

$$(\mathrm{P_m})_k = \frac{1}{N}\sum_{i=1}^{N}(P_m)_i\left(\frac{f_i}{\overline{f}}\right) \qquad (6.2)$$

Lo que este cálculo indica es que hace el promedio ponderado de las probabilidades de mutación, por la proporción del *fitness* del individuo correspondiente respecto al *fitness* promedio de la población. La P_m resultante, será la utilizada en la *k*-ésima generación como la tasa de mutación.

6.3.5 RMHC

Un RMHC (escalador) es capaz de superar el AG canónico en ciertas funciones. Para tomar ventaja del escalador en tales casos, se incluye un RMHC como parte del Algoritmo. Esto es, el EGA consiste de un AG auto-adaptativo más aparte un escalador.

¿Cómo determinamos cuando el escalador debería ser activado en lugar del AG? Determinamos esto con el siguiente mecanismo, que en el EGA original es auto-adaptativo pero aquí no lo es.

a) Primero, se definen dos límites:
    1) El porcentaje mínimo del escalador (η_λ)
    2) El porcentaje máximo del escalador (η_μ)

donde:

$$0 < \eta_\lambda < \eta_\mu \leq 1 \qquad (6.3)$$

El escalador será activado, por lo menos, η_λ del tiempo y, a lo más, η_μ del tiempo.

b) Segundo, en la generación *k*-ésima la eficiencia del escalador se evalúa de la siguiente manera:

$$\eta_\phi = \frac{1}{N}\sum_{i=1}^{N}(\iota_\eta)_i \qquad (6.4)$$

donde:

$$\iota_\eta = \begin{cases} 1 \; si \; el \; individuo \; \; fue \; encontrado \; \; por \; el \; escalador \\ 0 \; en \; otro \; caso \end{cases} \qquad (6.5)$$

Para determinar el valor de ι_η se incluye como elemento en el genoma (véase la figura 6.3). Este elemento es de tipo booleano. Tendrá el valor de 1 cuando el individuo haya sido encontrado por el escalador, y tendrá el valor de 0 en el caso contrario.

c) Tercero, denotar la probabilidad de invocar el escalador, P_H:

$$P_H = \begin{cases} \eta_\lambda \; si \; \eta_\phi < \eta_\lambda \\ \eta_\phi \; si \; \eta_\lambda \leq \eta_\phi \leq \eta_\mu \\ \eta_\mu \; si \; \eta_\phi > \eta_\mu \end{cases} \qquad (6.6)$$

d) Cuarto, generar un número aleatorio ρ_k, donde $0 < \rho_k \leq 1$. Preparar para invocar el escalador si $P_H \leq \rho_k$.

e) Quinto, una vez que el escalador está programado para empezar, la cadena sobre la cual operará es seleccionada aleatoriamente (de manera uniformemente distribuida) de entre los cinco primeros individuos de la población. Recuérdese que los individuos en la población están ordenados del mejor (individuo 1) al peor (individuo *N*).

El resultado de esta estrategia descrita es para garantizar que el escalador estará activo siempre que haya probado ser efectivo. La probabilidad de que el escalador anule al AG es, sin embargo, delimitada arriba por η_μ. Esto previene que el escalador tome el proceso por entero. Por otro lado, el escalador será invocado con $P_H \geq \eta_\lambda$ la cual, en turno, evita la

posibilidad de que debido a un desempeño pobre de parte del escalador en la *k*-ésima generación, el escalador sea excluido del resto del proceso.

Los escaladores están pensados para apuntar hacia puntos óptimos cuando el algoritmo ha alcanzado un espacio vecino en la superficie de soluciones. Sin embargo, aquí, el escalador sirve para dos propósitos: a) En efecto, apunta a puntos locales, cercanos al máximo, y b) Fuerza a que haya una variedad en la población al explorar nuevos esquemas que el AG de otra manera pasaría desapercibidos.

El hecho más notable acerca de esta mezcla, AG-escalador, es que, es de hecho el AG el que hace la búsqueda fina de óptimos locales, con el escalador sirviendo de agente de disparo, el cual localiza soluciones sub-óptimas muy eficientemente.

Ahora se puede ver mejor cómo es que el EGA se aproxima al AG ideal:

a) Por medio de la estrategia de Vasconcelos, ningún locus es fijado a un valor.
b) Por medio del elitismo completo, los esquemas deseados son secuestrados para preservar buenas características.
c) Por medio de la auto-adaptación, la cruza se escoge tal que dinámicamente se mantiene la mejor tasa de cruza.
d) El factor de aumento de velocidad se acerca al del RMHC, debido al hecho de que el mecanismo de un escalador está explícitamente incluido.

6.4 Evolución de los pesos de redes neuronales para navegación

Una manera de entrenar redes neuronales multiperceptrón es mediante el algoritmo *backpropagation*, como fue explicado anteriormente, pero una desventaja de utilizar este algoritmo del modelo conexionista es que hay que dar valores iniciales a ciertos parámetros, y dependiendo de éstos es el tiempo y grado de convergencia de la red neuronal.

Al utilizar el modelo evolutivo, o sea evolucionar o entrenar los pesos de una red neuronal mediante un algoritmo genético, el entrenamiento de la red neuronal no precisa de parámetros iniciales, salvo los indicados para el AG, y que en el algoritmo genético utilizado aquí son auto-adaptativos. Nótese la ventaja de entrenar una red neuronal bajo el paradigma evolutivo.

En este trabajo de tesis se evolucionaron los pesos de redes neuronales multiperceptrón con el fin de que aprendieran una ruta de navegación bajo ciertos ambientes. Después, estas redes neuronales fueron tomadas como comportamientos de navegación y fueron organizadas por un árbitro de comportamientos, el cual, como ya se había mencionado, es una red neuronal multiperceptrón y que fue entrenado siguiendo este modelo evolutivo.

La arquitectura de las redes neuronales que codifican los comportamientos simples posee dos neuronas de entrada, cada una de las cuales recibe el promedio de 341 sensores láser; se

tienen dos o tres neuronas en la capa oculta; dos neuronas de salida, una de éstas le indica al robot la distancia a desplazarse y la otra el ángulo que debe girar. Las neuronas de la capa oculta y las neuronas de salida tienen la función *tanh* como función de activación. Se cuenta con una neurona de sesgo, cuya entrada está fija al valor de 1, y que alimenta a las neuronas de la capa oculta y a las dos neuronas de la capa de salida. La siguiente figura ilustra esta arquitectura.

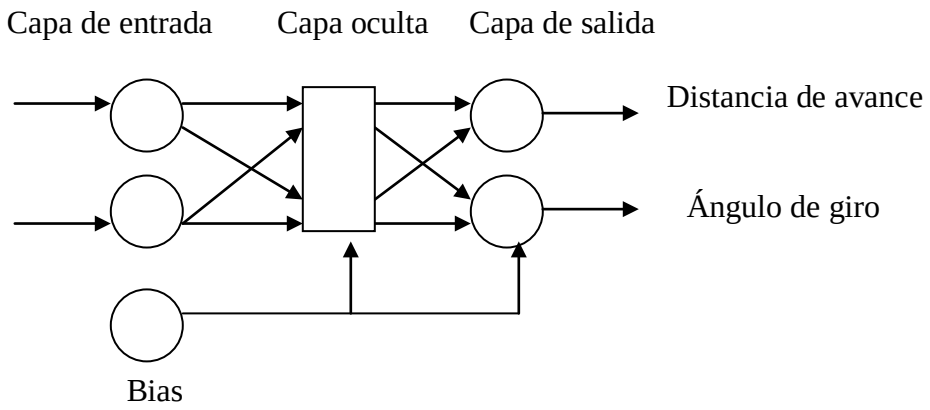


Figura 6.4. Arquitectura de las redes neuronales que codifican comportamientos simples de navegación.

La metodología para evolucionar una red neuronal es la siguiente. Se define un ambiente sobre el cual la red neuronal se entrenará. Se define un punto de inicio para el robot y un punto destino. Al ejecutar el algoritmo genético se le pasan estos parámetros, se crea una población inicial donde cada individuo (cromosoma) son los pesos de una red neuronal (el genoma consiste de 12 valores de punto flotante). Cada individuo se evalúa permitiendo que la red neuronal tome control del robot por un cierto número de acciones y midiendo la distancia que le faltó para llegar al punto destino.

Nótese que es aprendizaje supervisado, ya que la función de fitness es la distancia que le falta al individuo en turno para llegar al punto destino, por lo que puede verse como una señal de corrección que se le da a dicho individuo y le va indicando si va bien o va mal.

Nótese también la diferencia que existe entre entrenarlo con esta señal de corrección y darle los datos de entrenamiento (entradas y salidas deseadas). Al darle los datos de entrenamiento e intentar minimizar una señal de error lo que se logra es que la red neuronal se aprenda dicho conjunto de datos, en este caso que trace la ruta de navegación propuesta, y se espera que logre una buena extrapolación posteriormente. Mediante este entrenamiento se logra que la red neuronal descubra por si sola la ruta navegación entre dos puntos y logre así un comportamiento emergente.

6.5 Evolución de los pesos del árbitro de comportamientos

El árbitro de comportamientos es una red neuronal multiperceptrón, como fue descrita en el capítulo 5. La arquitectura consta de dos neuronas de entrada, cada una de las cuales recibe el promedio de 341 sensores láser; se tienen dos o tres neuronas en la capa oculta; se tiene una neurona en la capa de salida, la salida de esta neurona indica el comportamiento a activarse; se cuenta con una neurona de sesgo, la cual alimenta las neuronas de la capa oculta y de la capa de salida. La función de activación en las capas oculta y de salida es la función *tanh*.

La metodología para evolucionar un árbitro de comportamientos es similar a la descrita para las redes neuronales que codifican los comportamientos de navegación. Se define un ambiente sobre el cual la red neuronal se entrenará. Se define un punto de inicio para el robot y un punto destino. Al ejecutar el algoritmo genético se le pasan estos parámetros, se crea una población inicial donde cada individuo (cromosoma) son los pesos de una red neuronal (el genoma consiste de 9 valores de punto flotante). Cada individuo (árbitro) se evalúa permitiendo que la red neuronal conmute, a cada paso que efectúe el robot, entre los comportamientos antes obtenidos y observando la distancia que le falta para llegar al punto destino.

La diferencia es que la red neuronal que codifica al árbitro de comportamientos hace un mapeo de los sensores láser a la activación de un comportamiento. Una vez que el árbitro decide qué comportamiento ejecutar se le cede el control del robot a dicho comportamiento, y la red neuronal que codifica a dicho comportamiento será la encargada de hacer el mapeo del láser a las señales para el motor del robot.

En el siguiente capítulo se muestran los resultados de los experimentos realizados.

6.6 Ambientes de simulación

A continuación se muestran los ambientes utilizados para evolucionar los comportamientos de navegación y el árbitro de comportamientos.

Mundo1 **Mundo 2** **Mundo 3** **Mundo 4**

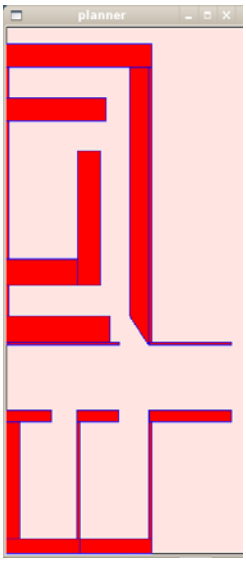


Figura 6.5. Mundo 1.

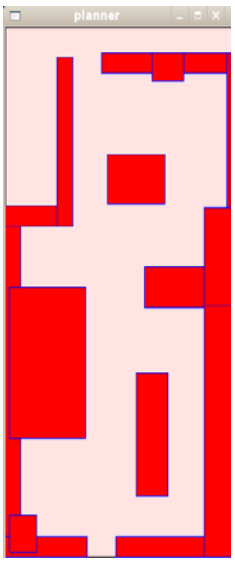


Figura 6.6. Mundo 2.

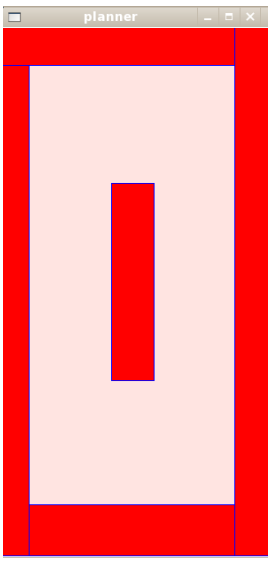


Figura 6.7. Mundo 3.

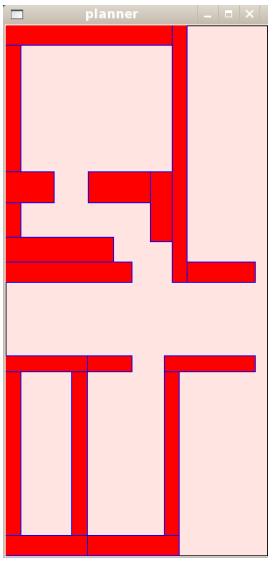


Figura 6.8. Mundo 4.

.

Mundo 5

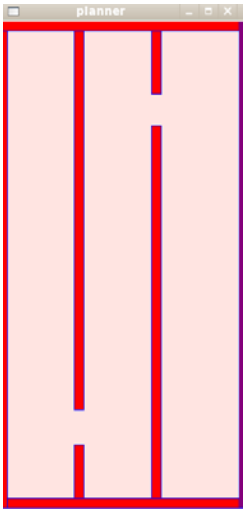


Figura 6.9. Mundo5.

Capítulo 7

RESULTADOS

7.1 Evolución de los pesos de redes neuronales que codifican comportamientos de navegación

A continuación se presentan los experimentos realizados y los resultados obtenidos de evolucionar los pesos de redes neuronales multiperceptrón para que el robot "aprendiera" a navegar de un punto a otro dentro de un ambiente, y así codificar un comportamiento. La arquitectura de las redes neuronales consta de 2 neuronas de entrada, una capa oculta con 2 neuronas cuya función de activación es la función *tanh*, y una capa de salida con 2 neuronas con la misma función de activación, donde una neurona de salida manda a un actuador la distancia que el robot debe trasladarse y la otra manda el ángulo que debe girar, véase la figura 7.1(c).

Las entradas a las redes neuronales son el promedio de 341 láser del lado izquierdo del robot, esto es $P_i = \frac{1}{341}\sum_{i=0}^{340} láser_i$, y 341 láser del lado derecho del robot, o sea $P_d = \frac{1}{341}\sum_{i=341}^{681} láser_i$, véase la figura 7.1. (b).

Para evolucionar cada red neuronal se lleva a cabo el siguiente procedimiento: primero se selecciona un ambiente, después se indica un punto de inicio (x_1,y_1) que es el punto de partida del robot, y un punto destino (x_2,y_2) que indica el punto al cual se desea que llegue éste. Cada cromosoma codifica los pesos de una red neuronal con la arquitectura mostrada en la figura 7.1(c). Se ejecuta el EGA y se permite que cada individuo controle al robot por un cierto número de acciones, donde cada acción comprende una distancia de avance y un ángulo de giro; se obtiene el *fitness* que es la distancia euclidiana del punto actual (x_a,y_a) al punto destino, esto es: $\sqrt{(x_2 - x_a)^2 + (y_2 - y_a)^2}$, o bien la distancia de Manhattan del punto

actual al punto destino, esto es: $|x_2 - x_a| + |y_2 - y_a|$; con esta función de evaluación se da una señal de corrección que indica "que tan bien" está actuando el individuo actual para lograr llegar al punto destino.

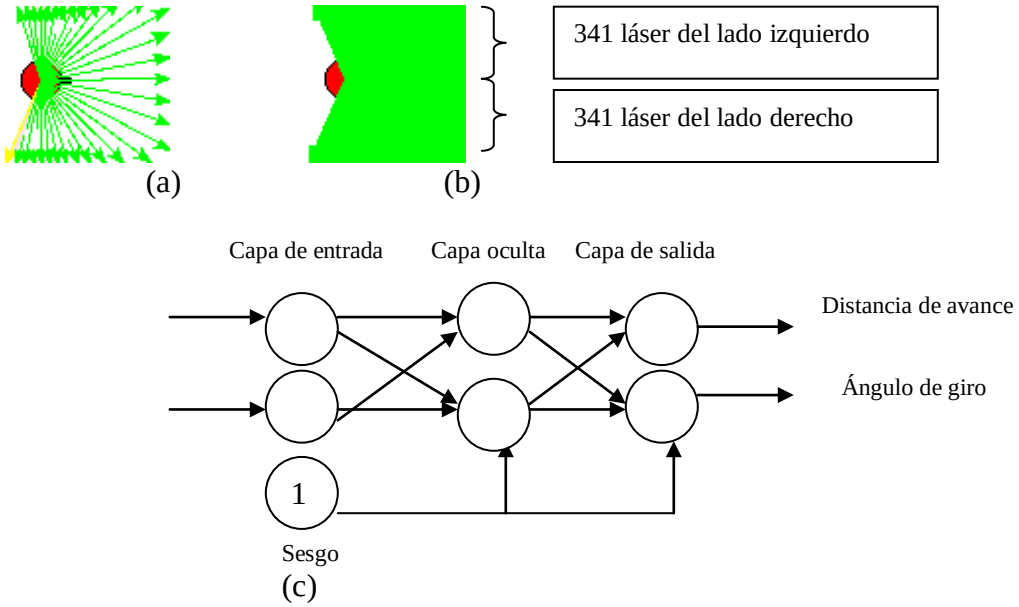


Figura 7.1. (a) Robot móvil con 32 rayos láser. (b) Robot móvil con 682 rayos láser. (c) Red neuronal utilizada para los experimentos de la sección 7.1.

Experimento 1

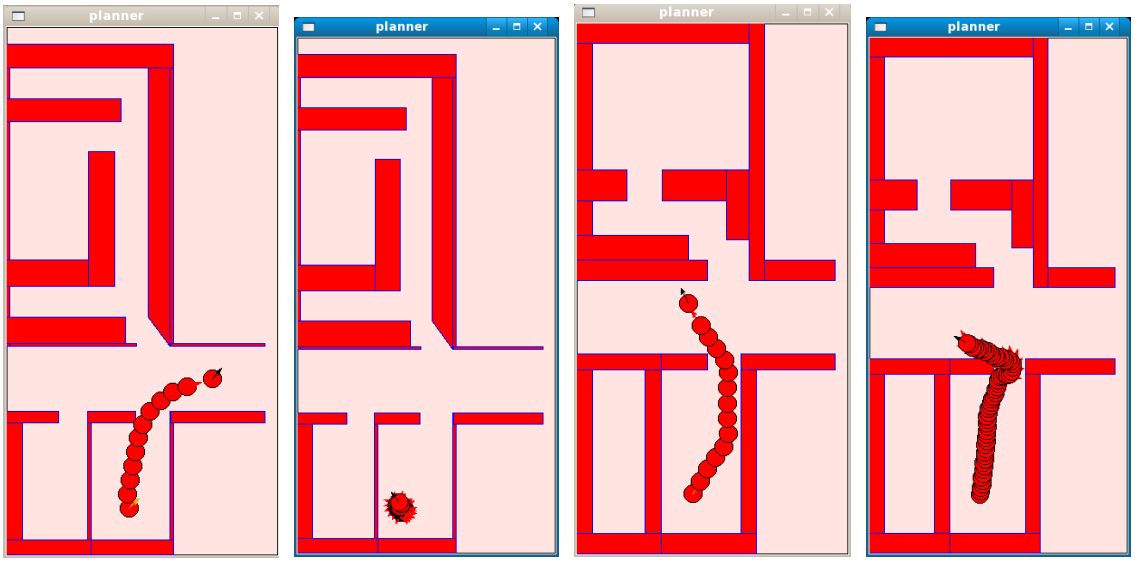
Se evolucionaron los pesos de una red neuronal con el objetivo de que aprendiera una ruta para salir de una habitación, hacia la derecha, en el ambiente mundo 1. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 60 acciones, cada acción consta de una distancia de avance y un ángulo de giro. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.2(a), 7.2(b) y 7.10 donde se muestra una imagen de un individuo exitoso, una imagen de un individuo no exitoso, o sea, que no realiza el comportamiento deseado, y una tabla con los *fitness* de los individuos mostrados en cada experimento, respectivamente.

Experimento 2

Objetivo: obtener una red neuronal que aprenda una ruta para salir de una habitación, hacia la izquierda, en el ambiente mundo 4. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 60 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véase la figura 7.2(c) y nótese que el individuo mostrado tiene el comportamiento emergente de evitar la pared, también véase la figura 7.2(d) donde se muestra un individuo no exitoso, que realiza giros innecesarios y se califica de oscilante.

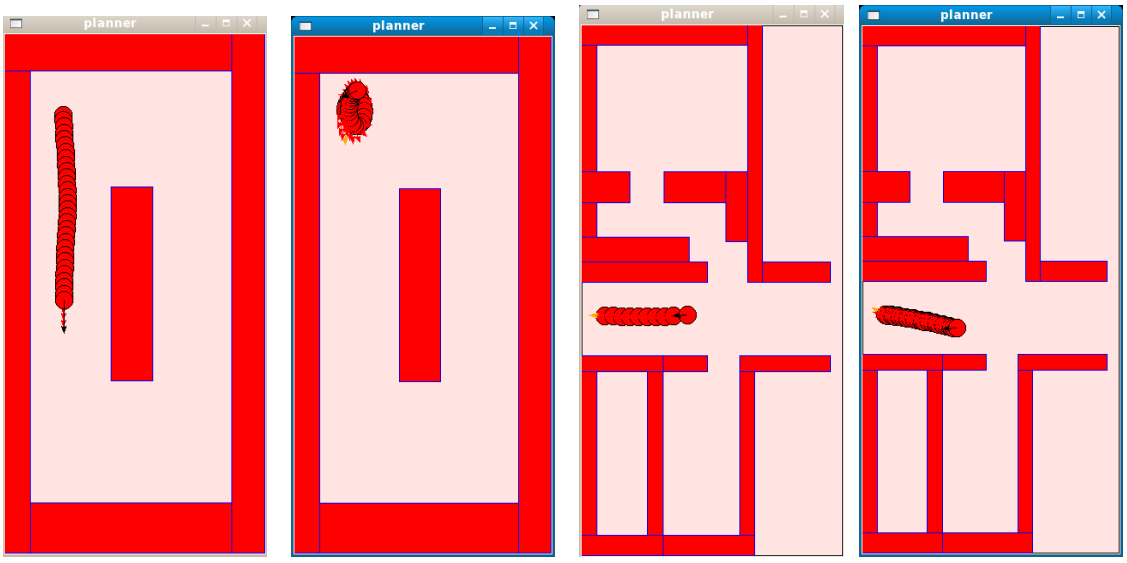
Experimento 3

Objetivo: una RN que aprenda una ruta para trazar una línea recta de arriba hacia abajo, en el ambiente mundo 3. Se utilizaron 50 individuos y 100 generaciones, y se permitió que cada individuo ejecutara 30 acciones. La métrica utilizada para evaluar a cada individuo fue la raíz de la distancia euclidiana. Véanse las figuras 7.3(a) y 7.3(b).



(a) (b) (c) (d)

Figura 7.2. (a) Experimento 1. (b) Experimento 1, individuo no exitoso, ya que no realiza el comportamiento requerido, ejecuta muchos giros sin avanzar significativamente de su posición inicial. (c) Experimento 2. (d) Experimento 2, individuo oscilante.



(a) (b) (c) (d)

Figura 7.3. (a) Experimento 3. (b) Experimento 3, individuo no exitoso. (c) Experimento 4. (d) Experimento 4, individuo no exitoso, se aleja del punto destino.

Experimento 4

Objetivo: trazar una línea recta de izquierda a derecha en reversa, en un pasillo del ambiente mundo 4. Se utilizaron 50 individuos y 100 generaciones, y se permitió que cada individuo ejecutara 30 acciones. La métrica utilizada para evaluar a cada individuo fue la raíz de la distancia euclidiana. Véanse las figuras 7.3(c) y 7.3(d).

Experimento 5

Objetivo: aprender una ruta para trazar una línea recta de izquierda a derecha, en el ambiente mundo 3. Se utilizaron 50 individuos y 100 generaciones, y se permitió que cada individuo ejecutara 60 acciones. La métrica utilizada para evaluar a cada individuo fue la raíz de la distancia euclidiana. Véanse las figuras 7.4(a) y 7.4(b). Ninguno de los individuos tuvo el comportamiento de evitar la pared y chocan con ella.

Experimento 6

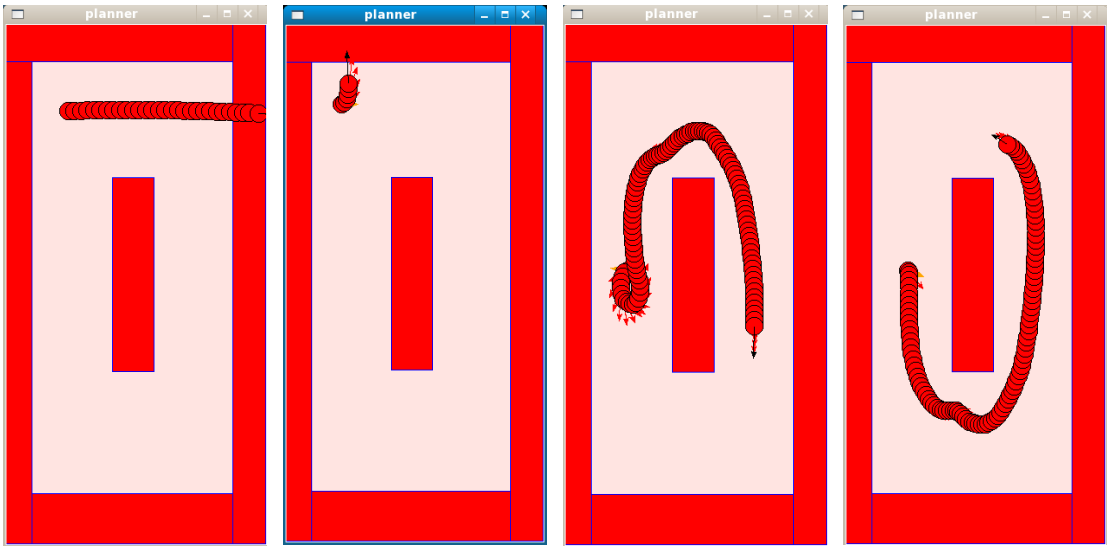
Objetivo: aprender una ruta para rodear un objeto, en el ambiente mundo 3. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 100 acciones. La métrica utilizada para evaluar a cada individuo fue la raíz de la distancia euclidiana. Véanse las figuras 7.4(c) y 7.4(d), nótese que en ambos casos el robot rodea el objeto, pero trazando rutas totalmente distintas; esta es una de las ventajas del paradigma evolutivo, que el robot puede descubrir varias rutas de navegación de manera emergente; también véase la figura 7.5(a) donde se muestra un individuo no exitoso.

Experimento 7

Objetivo: aprender una ruta para entrar en una habitación del ambiente mundo 4. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 40 acciones. La métrica utilizada para evaluar a cada individuo fue la raíz de la distancia euclidiana. Véanse las figuras 7.5(b) y 7.5(c).

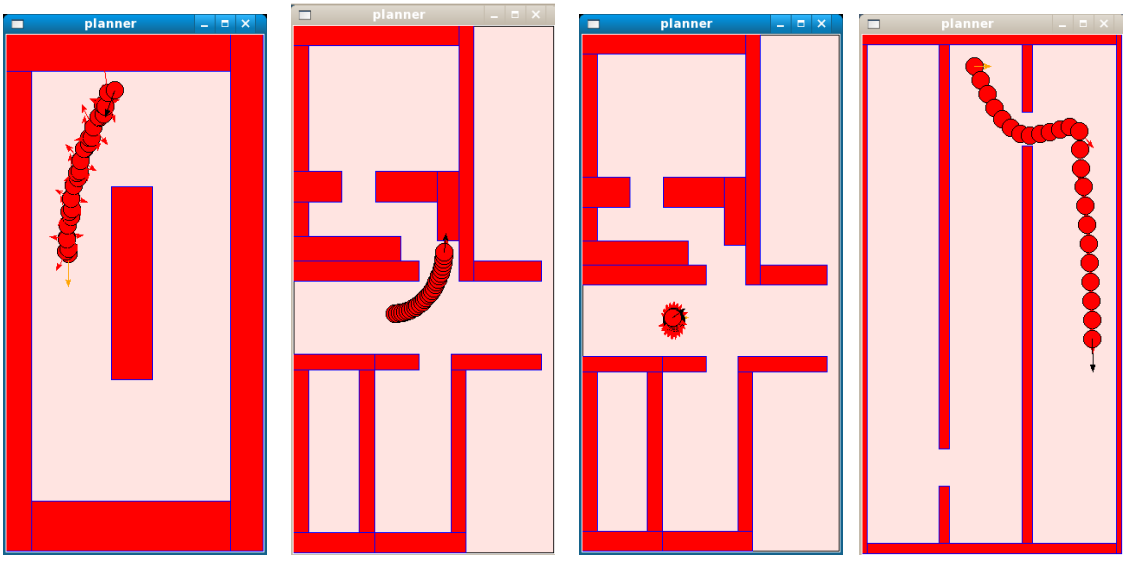
Experimento 8

Objetivo: navegar entre pasillos del ambiente mundo 5. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 70 acciones. La métrica utilizada para evaluar a cada individuo fue la raíz de la distancia euclidiana. Véanse las figuras 7.5(d) y 7.6(a).



(a) (b) (c) (d)

Figura 7.4. (a) Experimento 5. (b) Experimento 5, individuo no exitoso. (c) Experimento 6-a. (d) Experimento 6-b.



(a) (b) (c) (d)

Figura 7.5. (a) Experimento 6, individuo no exitoso. (b) Experimento 7. (c) Experimento 7, individuo no exitoso. (d) Experimento 8.

Experimento 9

Objetivo: aprender una ruta para salir de una habitación del ambiente mundo 1. Se utilizaron 50 individuos y 100 generaciones, y se permitió que cada individuo ejecutara 50 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.6(b) y 7.6(c).

Experimento 10

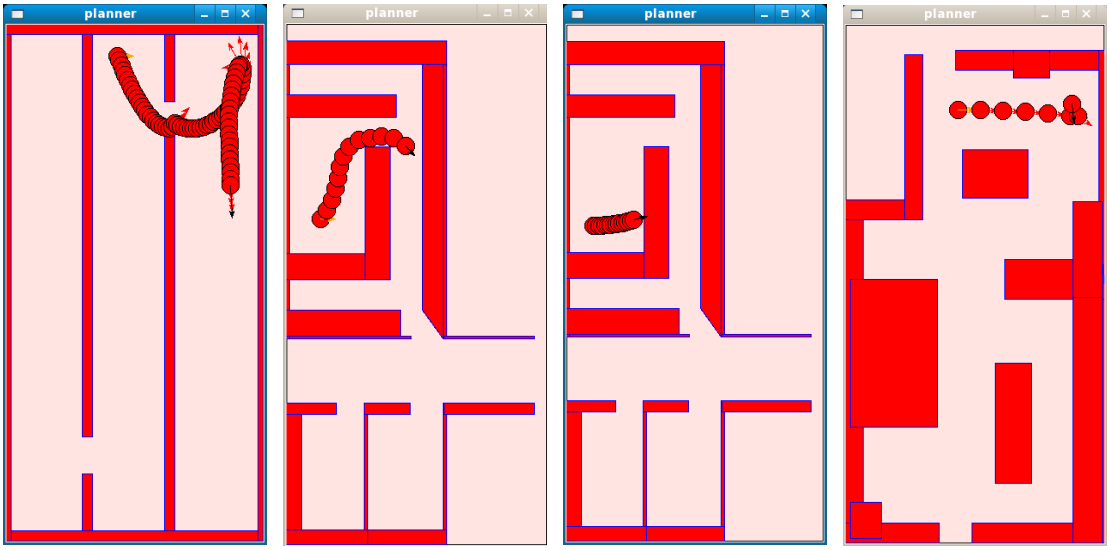
Objetivo: aprender una ruta para trazar una línea recta de izquierda a derecha en el ambiente mundo 2. Se utilizaron 50 individuos y 100 generaciones, y se permitió que cada individuo ejecutara 50 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véase la figura 7.6(d). Nótese que al llegar a la pared, el individuo en la imagen tiene el comportamiento emergente de evitar la pared y se echa para atrás. Véase la figura 7.7(a) donde se muestra un individuo no exitoso.

Experimento 11

Objetivo: aprender una ruta para rodear un obstáculo en el ambiente mundo 2. Se utilizaron 50 individuos y 150 generaciones, y se permitió que cada individuo ejecutara 50 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.7(b) y 7.7(c).

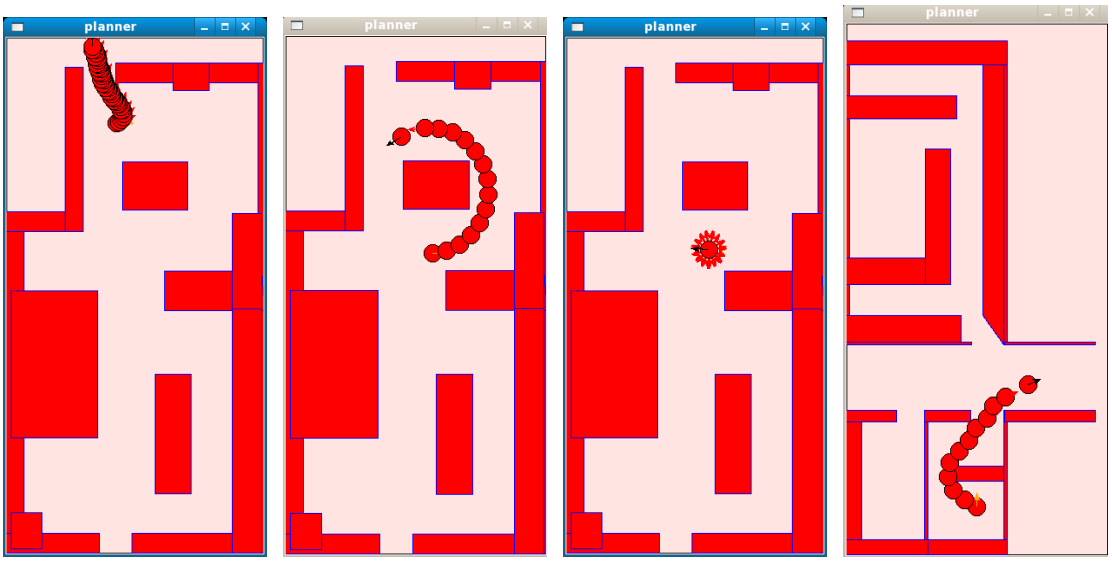
Experimento 12

Objetivo: aprender una ruta para rodear un obstáculo y salir de una habitación en el ambiente mundo 1. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 60 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.7(d) y 7.8(a). Nótese que el individuo mostrado logra el comportamiento emergente de esquivar el obstáculo.



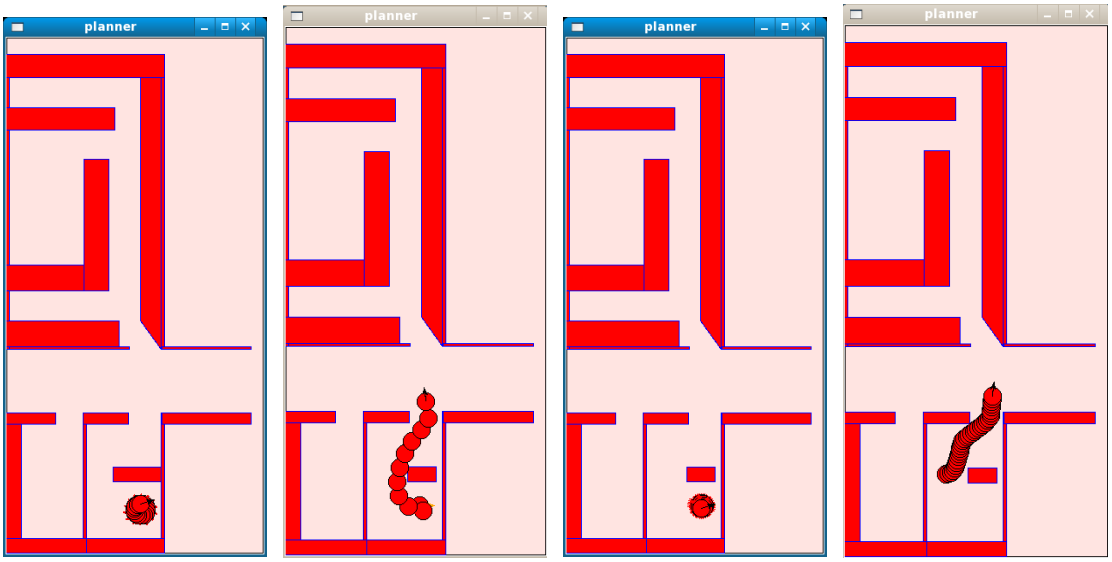
(a) (b) (c) (d)

Figura 7.6. (a) Experimento 8, individuo no exitoso. (b) Experimento 9. (c) Experimento 9, individuo no exitoso. (d) Experimento 10.



(a) (b) (c) (d)

Figura 7.7. (a) Experimento 10, individuo no exitoso. (b) Experimento 11. (c) Experimento 11, individuo no exitoso. (d) Experimento 12.



(a) (b) (c) (d)

Figura 7.8. (a) Experimento 12, individuo no exitoso. (b) Experimento 13. (c) Experimento 13, individuo no exitoso. (d) Experimento 14.

Experimento 13

Objetivo: aprender una ruta para rodear un obstáculo y salir de una habitación en el ambiente mundo1. Las diferencias de este experimento con el experimento 12 residen en que aquí el obstáculo es más pequeño y la métrica utilizada para evaluar a cada individuo

fue la distancia de Manhattan. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 52 acciones. Véanse las figuras 7.8(b) y 7.8(c).

Experimento 14

Objetivo: aprender una ruta para evitar un obstáculo y salir de una habitación en el ambiente mundo 1. El obstáculo es el mismo que en el experimento 13. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 35 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia de Manhattan. Véase las figuras 7.8(d) y 7.9.

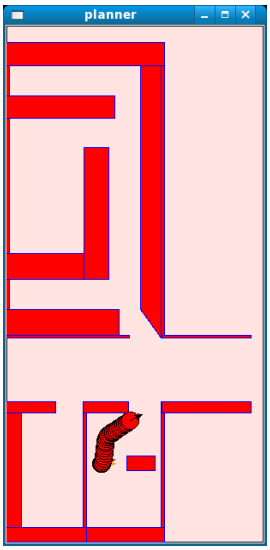


Figura 7.9. Experimento 14, individuo no exitoso.

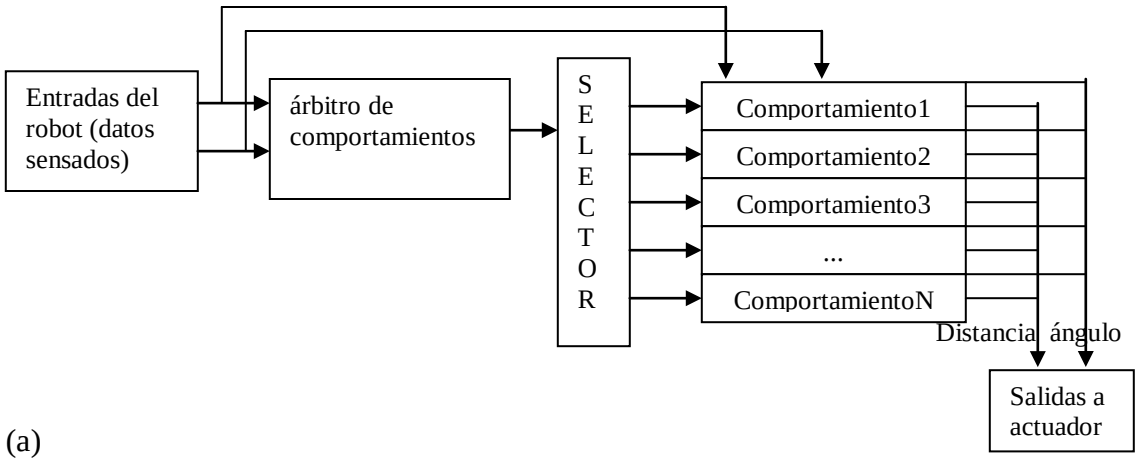
Experimento	Figura	Fitness (idealmente cero)
1	7.2(a)	0.187347
2	7.2(c)	5.811932
3	7.3(a)	0.031250
4	7.3(c)	0.892495
5	7.4(a)	0.562700
6	7.4(c)	1.447661
7	7.5(b)	0.106841
8	7.5(c)	2.074766
9	7.6(a)	1.030063
10	7.6(c)	0.332824
11	7.6(d)	0.016089
12	7.7(a)	1.298670
13	7.7(b)	0.013569
14	7.7(d)	2.946796

Figura 7.10. Tabla que muestra el *fitness* de los individuos indicados en la columna con el encabezado "Figura". El *fitness* indica la distancia en decímetros que le faltó al individuo para llegar al punto destino, por lo cual el fitness ideal es cero.

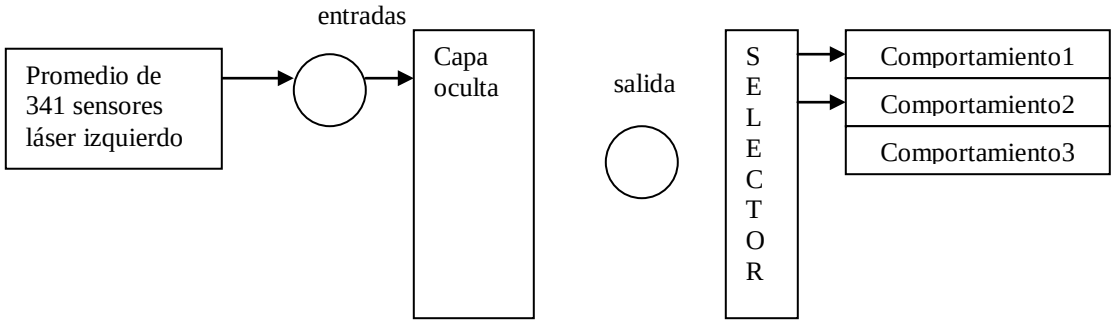
7.2 Evolución de los pesos de redes neuronales que codifican árbitros de comportamientos

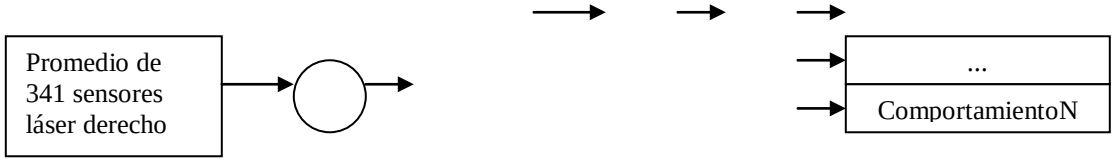
A continuación se presentan los experimentos realizados y los resultados obtenidos de evolucionar los pesos de redes neuronales multiperceptrón que codifican árbitros de comportamientos, con el fin de obtener un comportamiento más complejo de navegación a partir de comportamientos simples de navegación (los cuales fueron presentados en el apartado anterior) y algunos en conjunto con acciones simples.

El objetivo de un árbitro de comportamientos es la organización de la secuencia de ejecución de comportamientos simples, con el objetivo de mostrar un comportamiento global. Véanse las figuras 7.11(a) y 7.11(b) donde se muestran el esquema del árbitro de comportamientos propuesto en este trabajo de tesis y la arquitectura de una red neuronal codificando a un árbitro de comportamientos, anteriormente mostradas en el capítulo 4 y capítulo 5, respectivamente. En la figura 7.22 se muestra una tabla con los *fitness* de los individuos mostrados en los siguientes experimentos.



(a)





(b)

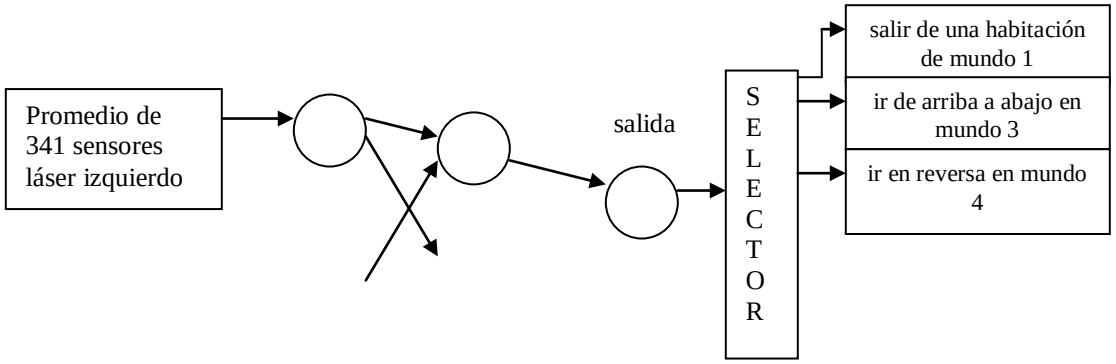
Figura 7.11. (a) Esquema del árbitro de comportamientos propuesto. (b) Red neuronal que codifica al árbitro de comportamientos propuesto.

Nótese que las entradas son las mismas que para las redes neuronales evolucionadas en el apartado anterior, el promedio de 341 láser del lado izquierdo del robot, esto es $P_i = \frac{1}{341}\sum_{i=0}^{340} láser_i$, y 341 láser del lado derecho del robot, o sea $P_d = \frac{1}{341}\sum_{i=341}^{681} láser_i$, véase la figura 7.1(b) para una mejor visualización de estas entradas.

Como se puede ver en la figura anterior, la arquitectura de las redes neuronales consta de 2 neuronas de entrada, una capa oculta con 2 o 3 neuronas cuya función de activación es la función *tanh*, y una capa de salida con sólo una neurona, con la misma función de activación. La neurona de salida actúa como un selector que elige un comportamiento a ejecutarse en un tiempo *t*, de entre *n* comportamientos disponibles. Cuando el árbitro de comportamientos elige un comportamiento se le pasa a éste los promedios de los láser y se le permite controlar al robot por el tiempo que dure una acción, o sea, por un movimiento de traslación y de rotación.

Experimento 15

Se evolucionaron los pesos de una red neuronal con el objetivo de que organizara de manera óptima los comportamientos de los experimentos: 1, 3, 4, 5, 6-a, 7, 8, para esquivar un obstáculo del ambiente mundo 2, véase la figura 7.12. Nótese que el ambiente en el cual se evoluciona a la red neuronal es distinto al de los comportamientos simples. Se utilizaron 50 individuos y 100 generaciones, y se permitió que cada individuo ejecutara 75 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.14(a) y 7.14(b) donde se muestra un individuo que logra el comportamiento global deseado, y un individuo no exitoso. También véase la figura 7.22 donde se muestra una tabla con el *fitness* del individuo mostrado.



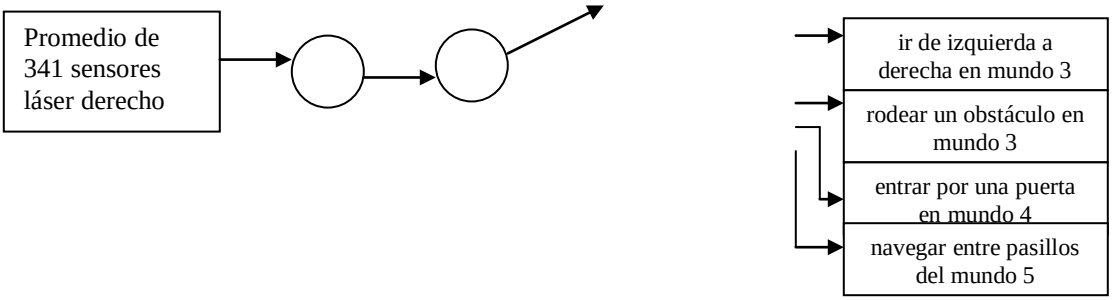


Figura 7.12. Esquema del árbitro de comportamientos del experimento 15.

Experimento 16

Objetivo: arbitrar de manera óptima los comportamientos de los experimentos: 3, 4, 5, 6-a, para esquivar un obstáculo y salir de una habitación del ambiente mundo 1. Véase la figura 7.13. Nótese que los comportamientos arbitrados son de otros ambientes y el árbitro logra ajustarlos al mundo 1. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 55 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.14(c) y 7.14(d). Este árbitro se utilizará posteriormente en el experimento 24.

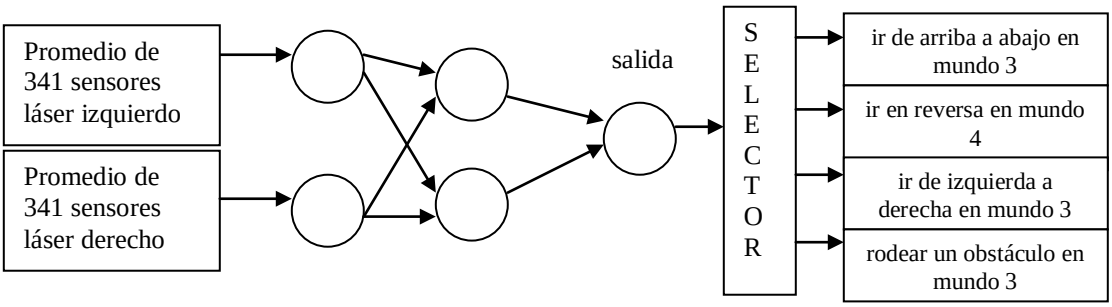
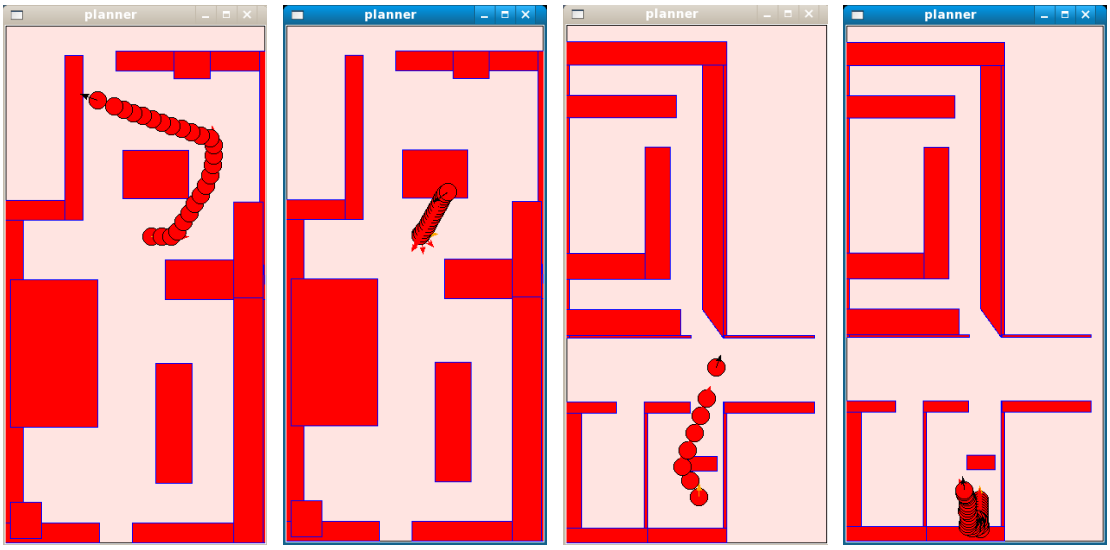


Figura 7.13. Esquema del árbitro de comportamientos de los experimentos 16 y 17.

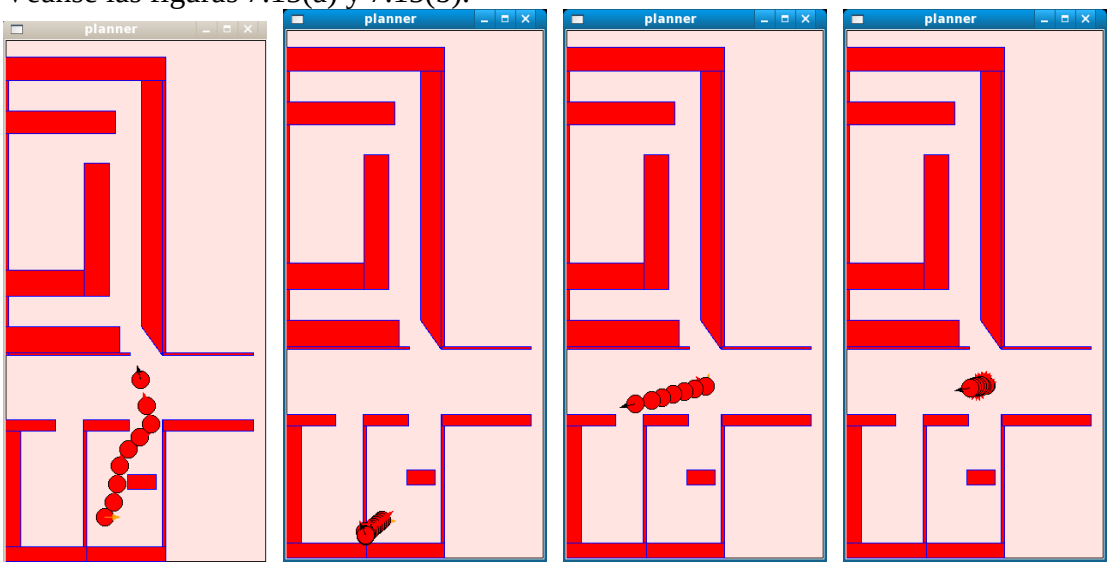


(a) (b) (c) (d)

Figura 7.14. (a) Experimento 15. (b) Experimento 15, individuo no exitoso. (c) Experimento 16. (d) Experimento 16, individuo no exitoso.

Experimento 17

Objetivo: arbitrar de manera óptima los comportamientos de los experimentos: 3, 4, 5, 6-a, para esquivar un obstáculo y salir de una habitación de mundo 1. Véase la figura 7.13. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 60 acciones. Este experimento se diferencia del experimento 16 en el punto de inicio de la evolución. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.15(a) y 7.15(b).



(a) (b) (c) (d)

Figura 7.15. (a) Experimento 17. (b) Experimento 17, individuo no exitoso. (c) Experimento 18. (d) Experimento 18, individuo no exitoso.

Experimento 18

Objetivo: arbitrar de manera óptima los comportamientos de los experimentos 3, 5, las acciones simples de girar 15° y de avanzar un decímetro como auxiliares, con el objetivo de cruzar un pasillo del ambiente mundo 1 y llegar al pie de la puerta de una habitación. Véase la figura 7.16. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 35 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.15(c) y 7.15(d). Este árbitro será utilizado en el experimento 24.

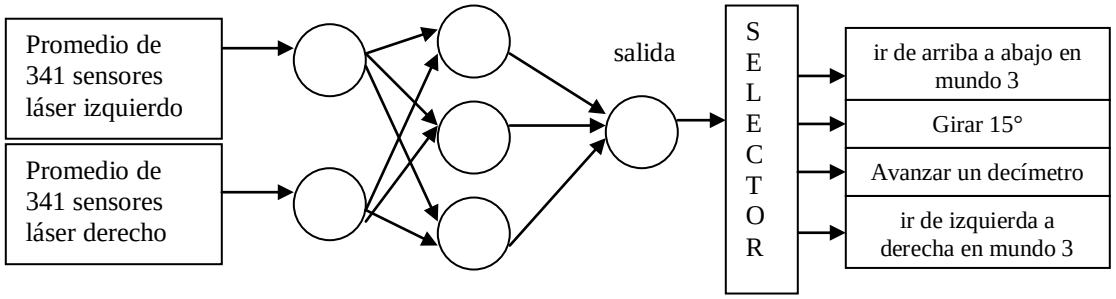


Figura 7.16. Esquema del árbitro de comportamientos de los experimentos 18, 21, 22, 23.

Experimento 19

Objetivo: arbitrar de manera óptima los comportamientos de los experimentos 3 y 5, y la acción simple de girar 15° como auxiliar, con el fin de entrar a la primera habitación, de izquierda a derecha, del ambiente mundo 1, una vez que el robot está parado frente a la puerta. Véase la figura 7.17. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 25 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.18(a) y 7.18(b). Este árbitro será utilizado posteriormente en el experimento 24.

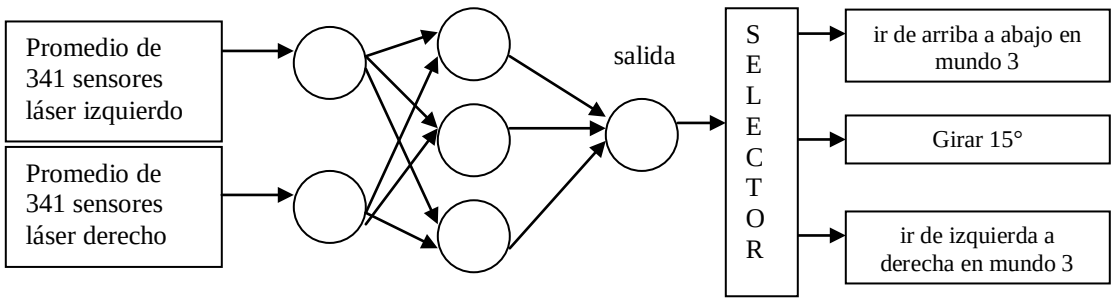
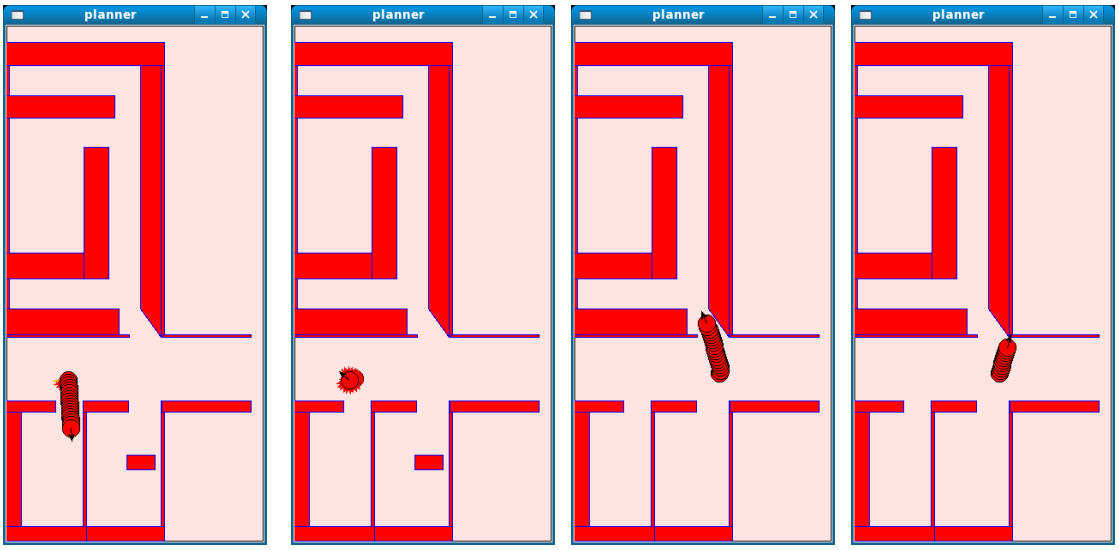


Figura 7.17. Esquema del árbitro de comportamientos del experimento 19.



(a) (b) (c) (d)

Figura 7.18. (a) Experimento 19. (b) Experimento 19, individuo no exitoso. (c) Experimento 20. (d) Experimento 20, individuo no exitoso.

Experimento 20

Objetivo: entrar a la habitación superior del ambiente mundo 1, esto al arbitrar los comportamientos de los experimentos 3, 4, 5, 6-a, y las acciones simples de girar 15° y avanzar un decímetro como auxiliares. Véase la figura 7.19. Se utilizaron 50 individuos y 100 generaciones, y se permitió que cada individuo ejecutara 25 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.18(c) y 7.18(d). Este árbitro será utilizado posteriormente en el experimento 24.

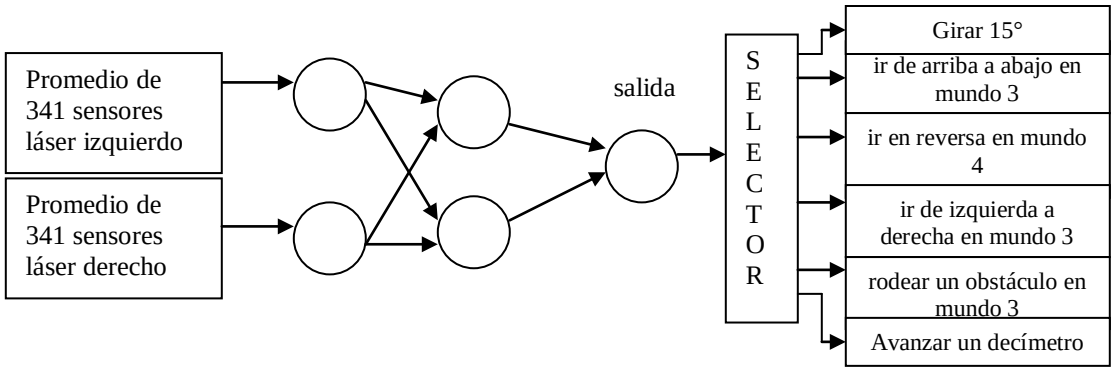
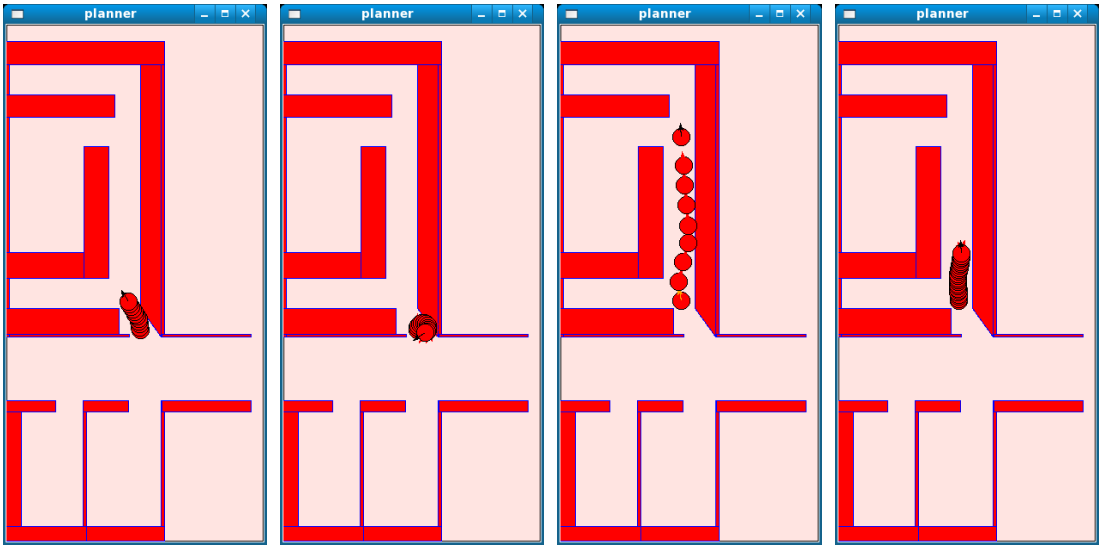


Figura 7.19. Esquema del árbitro de comportamientos del experimentos 20.

Experimento 21

Objetivo: llegar a la intersección de los dos pasillos de la habitación superior del ambiente mundo 1 partiendo de la entrada, al arbitrar los comportamientos de los experimentos 3, 5, y las acciones simples de girar 15° y de avanzar un decímetro como auxiliares. Véase la figura 7.16. Se utilizaron 50 individuos y 200 generaciones, y se permitió que cada individuo ejecutara 12 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.20(a) y 7.21(b). Los árbitros de los experimentos 20 y 21 bien se podrían haber hecho en un solo árbitro, la razón de haber hecho la división es que hubo problemas con la puerta, ya que en varias ocasiones la identificaba como un obstáculo, por lo cual se prefirió tener un árbitro especial para entrar y otro para llegar hasta la intersección de los pasillos. Este árbitro se utilizará posteriormente en el experimento 24.



(a) (b) (c) (d)

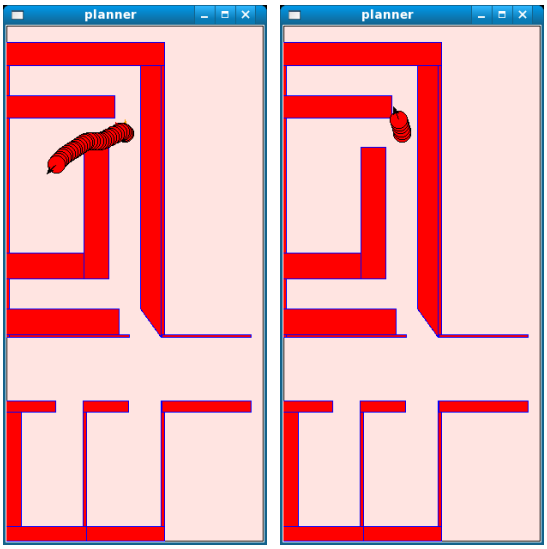
Figura 7.20. (a) Experimento 21. (b) Experimento 21, individuo no exitoso. (c) Experimento 22. (d) Experimento 22, individuo no exitoso.

Experimento 22

Objetivo: cruzar el pasillo vertical de la habitación superior del ambiente mundo 1 a partir de la intersección de los pasillos, al arbitrar los comportamientos de los experimentos 3, 5, y las acciones simples de girar 15° y de avanzar un decímetro como auxiliares. Véase la figura 7.16. Se utilizaron 50 individuos y 100 generaciones, y se permitió que cada individuo ejecutara 60 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.20(c) y 7.20(d).

Experimento 23

Objetivo: entrar al cuarto de la habitación superior del ambiente mundo 1 a partir del pasillo, al arbitrar los comportamientos de los experimentos 3, 5, y las acciones simples de girar 15° y de avanzar un decímetro como auxiliares. Véase la figura 7.16. Se utilizaron 50 individuos y 100 generaciones, y se permitió que cada individuo ejecutara 30 acciones. La métrica utilizada para evaluar a cada individuo fue la distancia euclidiana. Véanse las figuras 7.21(a) y 7.21(b).



(a) (b)

Figura 7.21. (a) Experimento 23. (b) Experimento 23, individuo no exitoso.

Experimento	Figura	Fitness
15	7.13(a)	0.168342
16	7.13(c)	0.288504
17	7.14(a)	0.230759
18	7.14(c)	1.508065
19	7.17(a)	4.823434
20	7.17(c)	0.560254
21	7.19(a)	3.054700
22	7.19(c)	0.366043
23	7.20(a)	3.650061

Figura 7.22. Tabla que muestra el fitness de los individuos indicados en la columna con el encabezado *Figura*. El *fitness* indica la distancia en decímetros que le faltó al individuo para llegar al punto destino, por lo cual el fitness ideal es cero.

7.2.1 Mapa de activación de árbitros de comportamientos

Un mapa de activación de árbitros de comportamientos es una representación de un ambiente en términos de regiones, en donde en cada región se ejecuta un árbitro deseado, a fin de que el robot vaya de una región del ambiente a otra. Cada comportamiento asociado

a una región es producto de un árbitro de comportamientos. Véase la figura 7.23 donde se muestra el mapa de activación de árbitros de comportamientos utilizado para el siguiente experimento.

Experimento 24

En este experimento se cuenta con el mapa de la figura 7.23, y se le pide al robot ir de la habitación derecha a la habitación izquierda del mundo 1. Para lograr esto, el robot activa el árbitro de comportamientos del experimento 16, que es para salir de la habitación representada por la región A, una vez que el robot sale de la habitación y llega a la región B se activa el árbitro para cruzar el pasillo (experimento 18), finalmente el robot llega a la región C donde se activa el árbitro para entrar a la otra habitación (experimento 19). Véanse las figuras 7.24, 7.25, 7.26, 7.27(a) y 7.27(b), donde se muestran ejecuciones de este experimento. Nótese que el robot puede partir de puntos y ángulos diversos de la región A, aún sin haber sido entrenada la red neuronal para dichos casos.

Otro caso del experimento es pedirle al robot que vaya de la habitación derecha inferior al cuarto de la habitación superior del ambiente mundo 1. Para conseguir esto se pone al robot en la región A, se activa el árbitro para salir de esta habitación, después el robot llega a la región B donde se activa el árbitro para entrar a la habitación superior (experimento 20), en seguida, en la región D se activa el árbitro para llegar a la intersección de los pasillos (experimento 21), en este punto, región E, se activa el árbitro para cruzar el pasillo vertical hasta llegar a la región F (experimento 22), finalmente se activa el árbitro para entrar al cuarto (experimento 23). Véanse las figuras 7.27(c), 7.27(d) y 7.28.

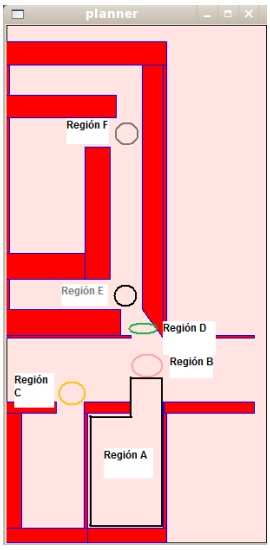
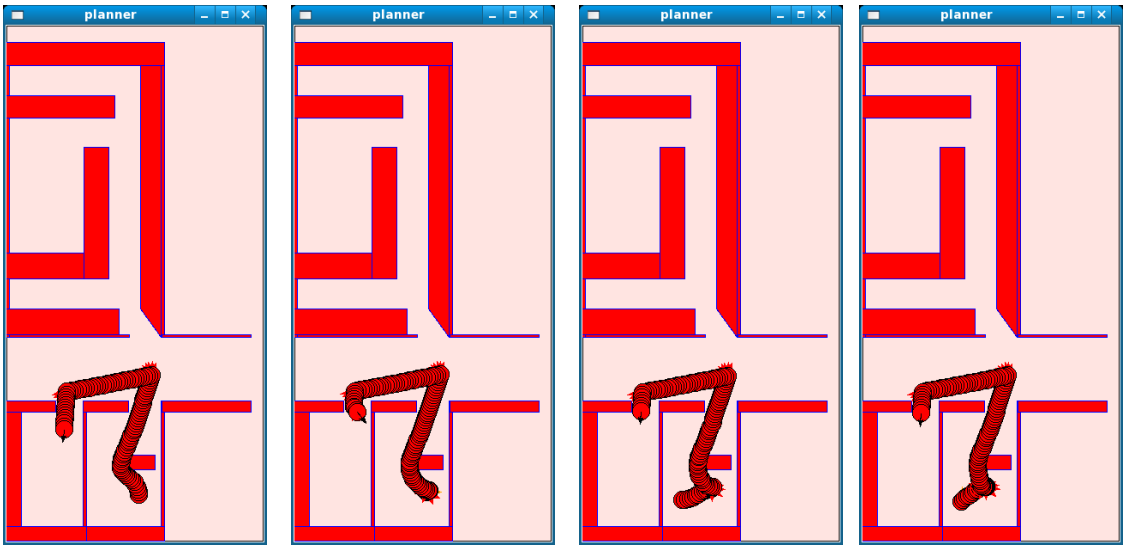
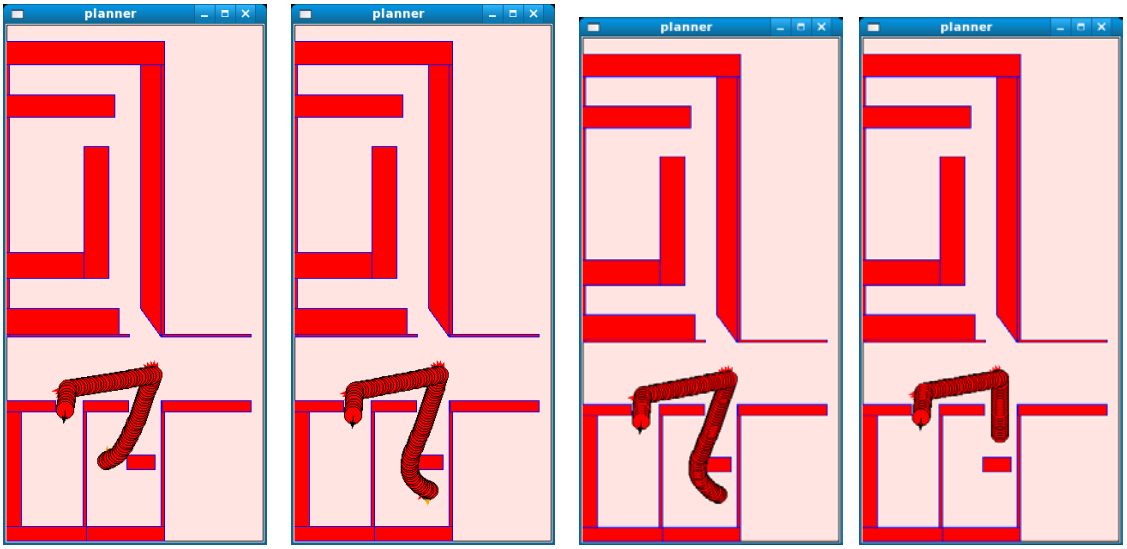


Figura 7.23. Mapa de activación de árbitros de comportamientos del experimento 24.



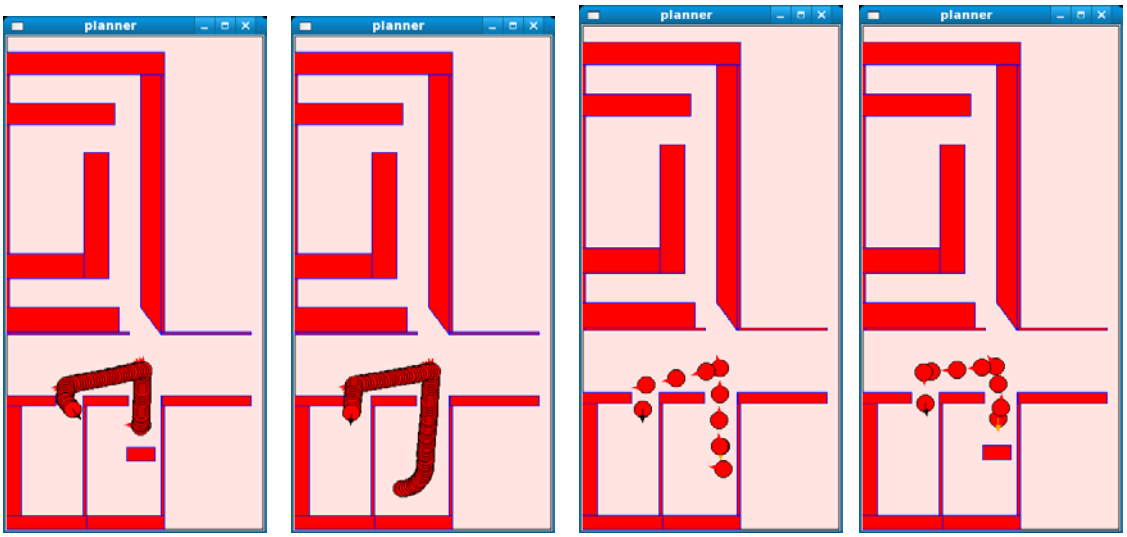
(a) (b) (c) (d)

Figura 7.24. Ejecuciones de los árbitros de comportamientos del experimento 24. El robot va de la habitación derecha a la habitación izquierda del ambiente mundo 1. (a) Posición inicial por debajo de un obstáculo, con ángulo inicial de 90°. (b) Posición inicial por debajo de un obstáculo, con ángulo inicial de 0°. (c) Posición inicial en la región izquierda inferior de la habitación, con ángulo inicial de 0°. (d) Posición inicial en la región izquierda inferior de la habitación, con ángulo inicial de 90°.



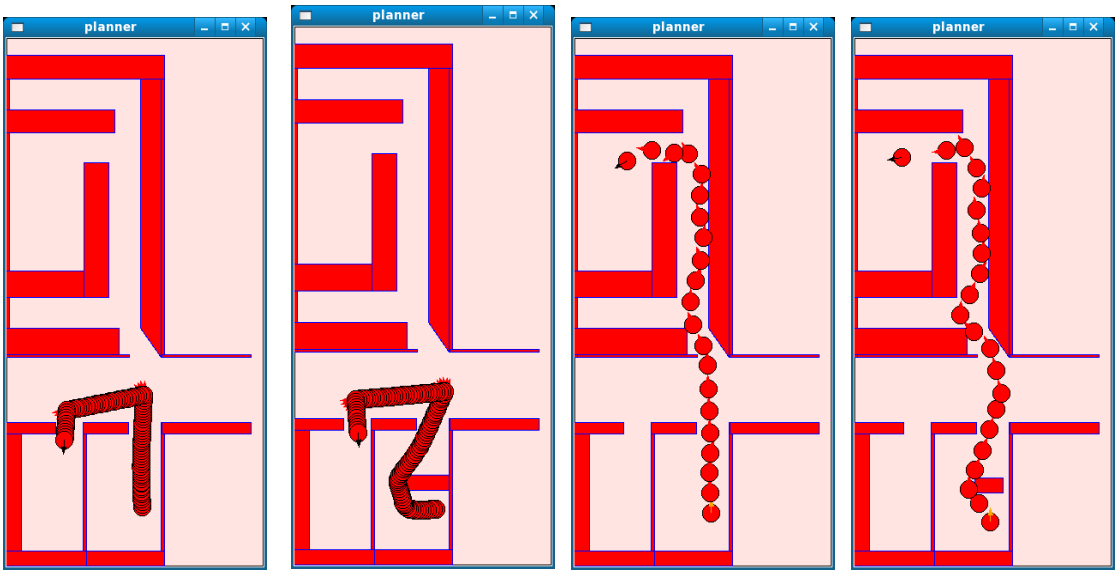
(a) (b) (c) (d)

Figura 7.25. Ejecuciones de los árbitros de comportamientos del experimento 24. El robot va de la habitación derecha a la habitación izquierda del ambiente mundo 1. (a) Posición inicial en la región izquierda media de la habitación, con ángulo inicial de 90°. (b) Posición inicial debajo del obstáculo, con ángulo inicial de 270°. (c) Posición inicial debajo del obstáculo, con ángulo inicial de 180°. (d) Posición inicial arriba del obstáculo, con ángulo inicial de 90°.



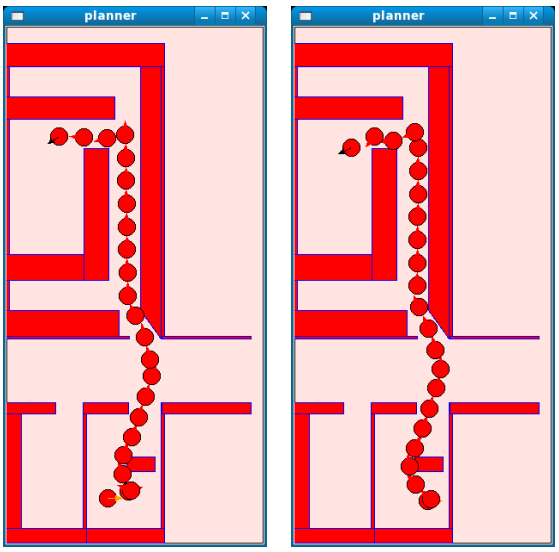
(a) (b) (c) (d)

Figura 7.26. Ejecuciones de los árbitros de comportamientos del experimento 24. El robot va de la habitación derecha a la habitación izquierda del ambiente mundo 1. (a) Posición inicial arriba del obstáculo, con ángulo inicial de 180°. (b) Posición inicial izquierda inferior de la habitación, con ángulo inicial de 0°, sin obstáculo. (c) Posición inicial derecha media de la habitación, con ángulo inicial de 270°, sin obstáculo. (d) Posición inicial arriba del obstáculo, con ángulo inicial de 270°.



(a) (b) (c) (d)

Figura 7.27. Ejecuciones de los árbitros de comportamientos del experimento 24. (a) El robot va de la habitación derecha a la habitación izquierda con una posición inicial derecha inferior y ángulo inicial de 90°, sin obstáculo. (b) El robot va de la habitación derecha a la habitación izquierda con una posición inicial debajo del obstáculo (aquí el obstáculo fue engrandecido casi al doble) y ángulo de 0°. (c) El robot va de la habitación derecha inferior al cuarto de la habitación superior, con una posición inicial en la esquina derecha inferior y ángulo de 90°, sin obstáculo. (d) El robot va de la habitación derecha inferior al cuarto de la habitación superior, con una posición inicial en la esquina derecha inferior (por debajo del obstáculo) y ángulo de 90°.



(a) (b)

Figura 7.28. Ejecuciones de los árbitros de comportamientos del experimento 24. El robot va de la habitación derecha inferior al cuarto de la habitación superior. (a) Posición inicial en la esquina izquierda inferior y ángulo de 0°. (b) Posición inicial en la esquina derecha inferior, por debajo del obstáculo y ángulo inicial de 0°.

Nótese que para poder ir de una región del ambiente a otra región es necesario, primero crear un mapa de activación de árbitros de comportamientos, y segundo, entrenar a un árbitro para que el robot se desplace de la región actual a la región deseada. Bien podría reutilizarse algún árbitro entrenado bajo un caso específico, de ser parecido geométricamente el espacio entre las nuevas regiones a desplazarse, de lo contrario sería difícil la reutilización de alguno de estos, a menos que el árbitro a reutilizarse tuviera un comportamiento emergente distinto al comportamiento para el que fue entrenado en primera instancia y resultara útil para el nuevo caso.

7.3 Comparación de resultados: EGA vs. Backpropagation

A continuación se presenta una RN que codifica a un árbitro de comportamientos, entrenada bajo el algoritmo de *backpropagation*, se muestran los resultados de navegación del robot al ser controlado por esta red neuronal, cuyo propósito es el mismo que la red neuronal del experimento 16, salir de la habitación derecha del ambiente mundo 1. Después, se muestra una comparativa entre ambos árbitros de comportamientos mediante dos tablas de resultados de ejecución de ambas redes neuronales en nueve casos distintos,

cada caso consta de veinte ejecuciones, también se presentan figuras alusivas a estos casos, presentando tanto casos exitosos como casos fallidos.

Experimento 25

Se entrenó una red neuronal mediante el algoritmo de *backpropagation* con momento, con el objetivo de que aprendiera a arbitrar de manera óptima los comportamientos de los experimentos: 3, 5, 6-a, para salir de la habitación derecha del ambiente mundo 1. La arquitectura de la red neuronal consta de 2 neuronas de entrada (véase la figura 7.1(b)), una capa oculta con 10 neuronas cuya función de activación es la función *tanh*, y una capa de salida con 3 neuronas cuya función de activación es la función sigmoide, donde cada neurona al ser activada elige a un determinado comportamiento (véase la figura 7.29). Se le presentaron 43 datos de entrenamiento. Al cabo de 200000 épocas se obtuvo un error promedio del 0.11%. Véanse las figuras 7.31, 7.32, 7.33 y 7.34, donde se muestran nueve ejecuciones de esta red neuronal, exitosas y fallidas. En la figura 7.30 se muestra una tabla reportando los nueve casos de ejecución, cada caso se ejecutó partiendo de veinte puntos de inicio vecinos a los puntos mostrados en la tabla.

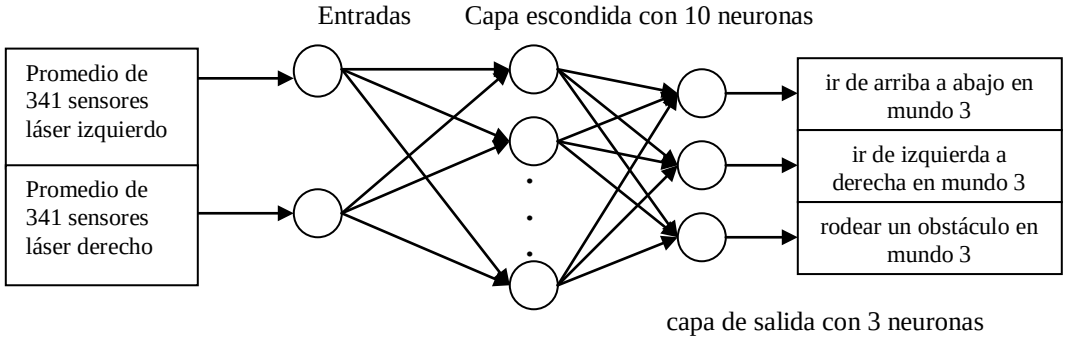
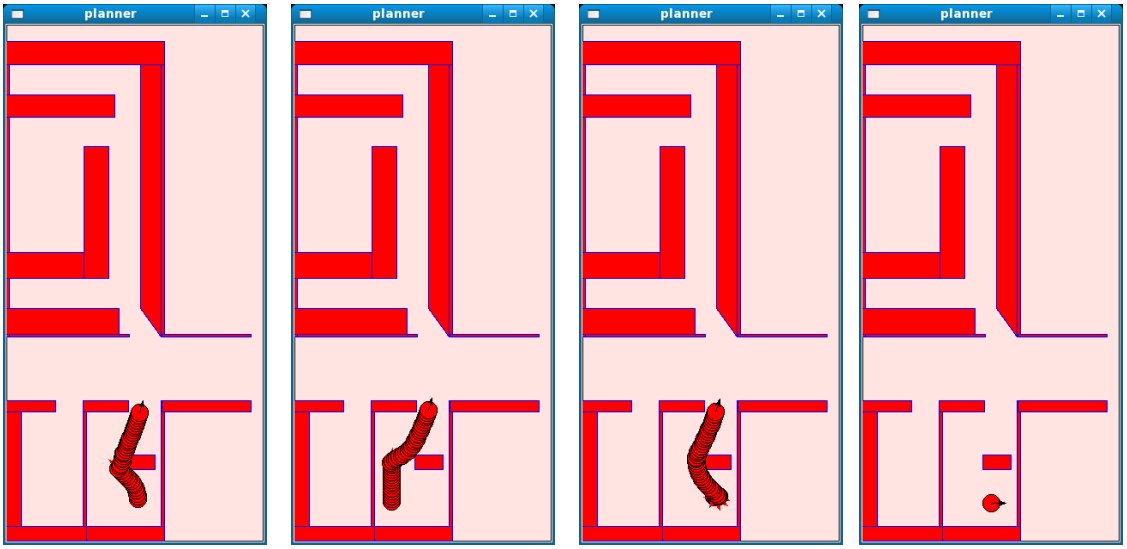


Figura 7.29. Arquitectura de la red neuronal que codifica un árbitro de comportamientos, entrenada bajo el algoritmo de backpropagation.

Figura	Posición inicial – coord. (x,y)	Ángulo inicial	Núm. de ejecuciones correctas	Núm. de ejecuciones fallidas
7.31(a)	Región derecha inferior, por debajo del obstáculo (43.91,16.2)	90°	20	0
7.31(b)	Región izquierda inferior (32.58,14.40)	90°	20	0
7.31(c), 7.31(d)	Región derecha inferior, por debajo del obstáculo (43.91,16.2)	0°	17	3
7.32(a), 7.32(b)	Región izquierda inferior (32.58,14.40)	270°	15	5
7.32(c)	Región izquierda inferior (32.58,14.40)	0°	20	0
7.32(d)	Región derecha inferior, sin obstáculo (43.91,16.2)	90°	0	20
7.33(a),	Región izquierda inferior, por debajo del	90°	4	16

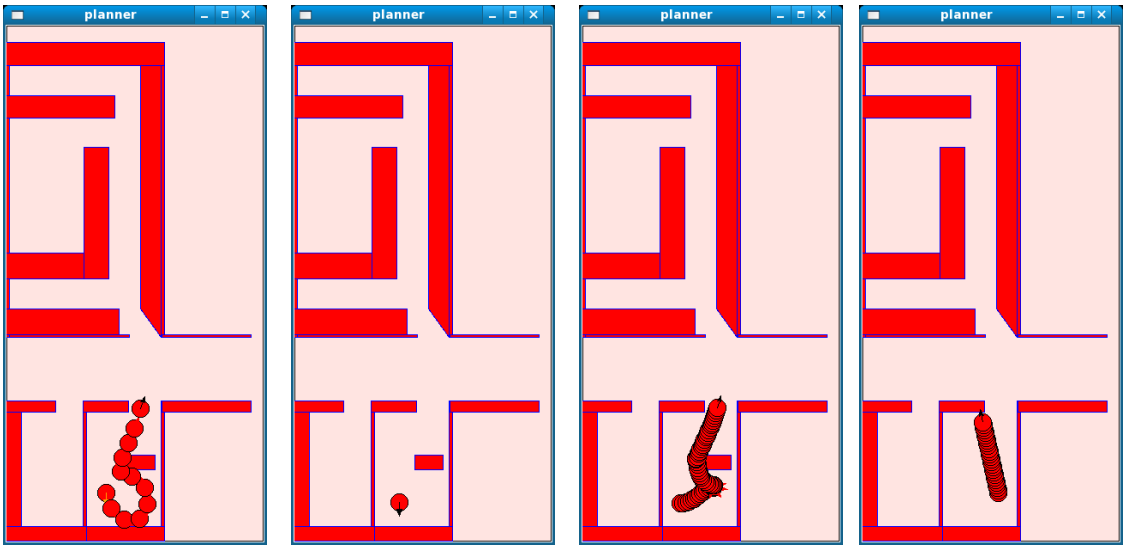
7.33(b)	obstáculo (desplazado a la izquierda) (43.91,16.2)			
7.33(c), 7.33(d)	Región derecha inferior, por debajo del obstáculo (aumentado a casi el doble de tamaño) (43.91,16.2)	0°	8	12
7.34(a), 7.34(b)	Región izquierda inferior, a un costado del obstáculo (aumentado a casi el doble de tamaño) (43.91,16.2)	90°	8	12

Figura 7.30. Tabla donde se reportan casos de ejecución del experimento 25.



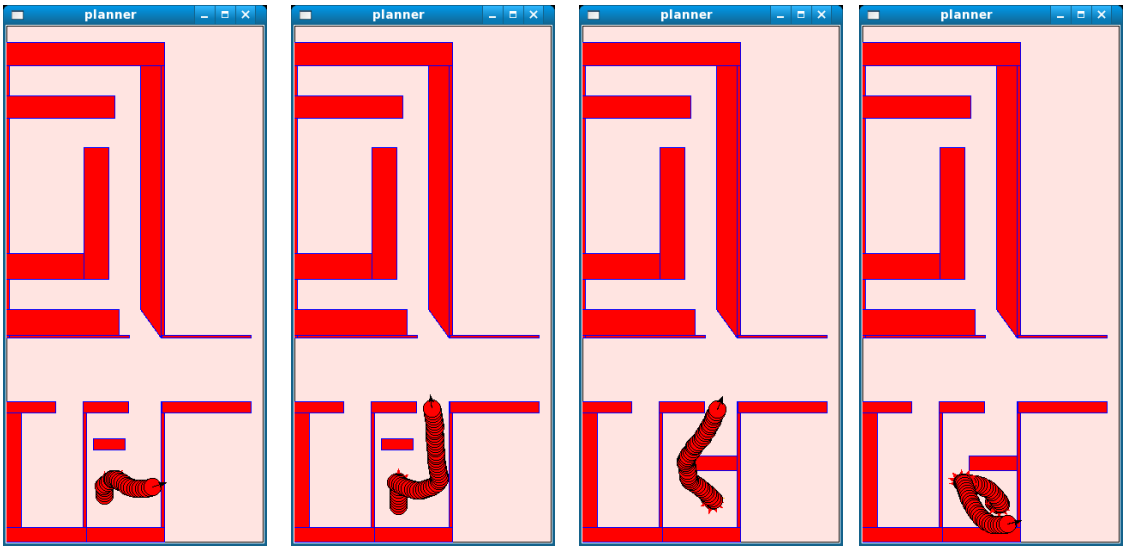
(a) (b) (c) (d)

Figura 7.31. Ejecuciones del árbitro de comportamientos del experimento 25. (a) Posición inicial derecha inferior por debajo del obstáculo con un ángulo de 90°. (b) Posición inicial izquierda inferior, con un ángulo inicial de 90°. (c) Posición inicial derecha inferior por debajo del obstáculo con ángulo de 0°. (d) Posición inicial derecha inferior por debajo del obstáculo, con ángulo de 0°, pero falla, el cómputo final es casi nulo y no se activa ninguna neurona de salida.



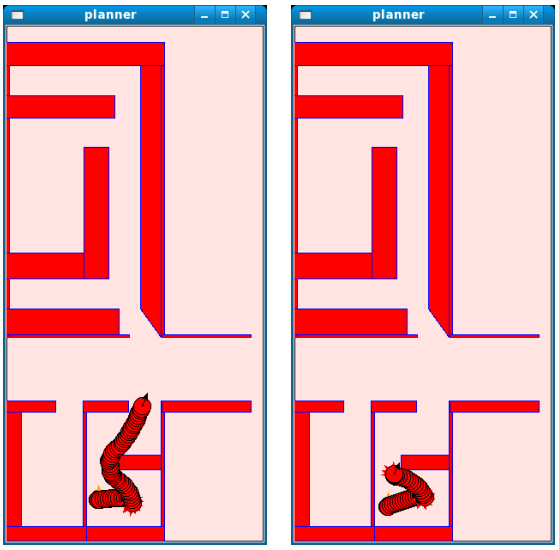
(a) (b) (c) (d)

Figura 7.32. Ejecuciones del árbitro de comportamientos del experimento 25. (a) Posición inicial izquierda inferior, con un ángulo inicial de 270°. (b) Posición izquierda inferior, con un ángulo inicial de 270° pero falla, el cómputo final es casi nulo y no se activa ninguna neurona de salida. (c) Posición izquierda inferior, con un ángulo inicial de 0°. (d) Posición derecha inferior sin obstáculo, con un ángulo inicial de 90°, pero falla en el comportamiento global.



(a) (b) (c) (d)

Figura 7.33. Ejecuciones del árbitro de comportamientos del experimento 25. (a) Posición inicial izquierda inferior por debajo del obstáculo desplazado a la izquierda, con ángulo de 90°, logra esquivar el obstáculo pero falla en el comportamiento global. (b) Posición inicial izquierda inferior por debajo del obstáculo desplazado a la izquierda, con ángulo de 90°. (c) Posición inicial derecha inferior por debajo del obstáculo agrandado a casi el doble de tamaño, con un ángulo inicial de 0°. (d) Posición inicial derecha inferior por debajo del obstáculo agrandado a casi el doble de tamaño, con un ángulo inicial de 0°, no logra pasar el obstáculo y por consiguiente no presenta el comportamiento global deseado.



(a) (b)

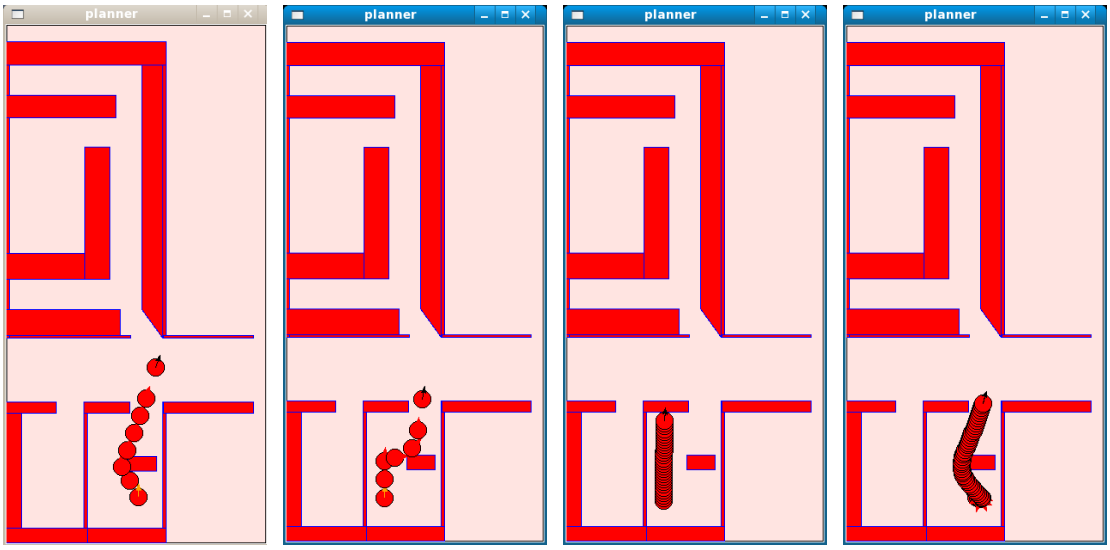
Figura 7.34. Ejecuciones del árbitro de comportamientos del experimento 25. (a) Posición inicial izquierda inferior, con un ángulo inicial de 90°, el obstáculo fue agrandado a casi el doble de tamaño. (b) Posición inicial izquierda inferior, con un ángulo inicial de 90°, el obstáculo fue agrandado a casi el doble de tamaño, no logra pasar el obstáculo y por consiguiente no logra el comportamiento deseado.

Ahora compárese las ejecuciones del árbitro de comportamientos del experimento 25 con las ejecuciones del árbitro de comportamientos del experimento 16. Véase la figura 7.35 donde se muestra una tabla con nueve casos de ejecución del árbitro de comportamientos del experimento 16, cada caso fue realizado veinte veces, con puntos de inicio vecinos a los puntos mostrados en la tabla. Véanse las figuras 7.31, 7.32, 7.33 y 7.34 donde se muestran ejemplos de estas ejecuciones, tanto exitosas como fallidas.

Figura	Posición inicial – coord. (x,y)	Ángulo inicial	Núm. de ejecuciones correctas	Núm. de ejecuciones fallidas
7.36(a)	Región derecha inferior, por debajo del obstáculo (43.91,16.2)	90°	20	0
7.36(b), 7.36(c)	Región izquierda inferior (32.58,14.40)	90°	12	8
7.36(d), 7.37(a)	Región derecha inferior, por debajo del obstáculo (43.91,16.2)	0°	14	6
7.37(b), 7.37(c)	Región izquierda inferior (32.58,14.40)	270°	9	11
7.37(d), 7.38(a)	Región izquierda inferior (32.58,14.40)	0°	17	3
7.38(b)	Región derecha inferior, sin obstáculo (43.91,16.2)	90°	20	0
7.38(c), 7.38(d)	Región izquierda inferior, por debajo del obstáculo (desplazado a la izquierda) (43.91,16.2)	90°	17	3
7.39(a), 7.39(b)	Región derecha inferior, por debajo del obstáculo (aumentado a casi el doble de tamaño) (43.91,16.2)	0°	16	4
7.39(c),	Región izquierda inferior, a un costado	90°	12	8

7.39(d)	del obstáculo (aumentado a casi el doble de tamaño) (43.91,16.2)			

Figura 7.35. Tabla donde se reportan casos de ejecución del experimento 16.

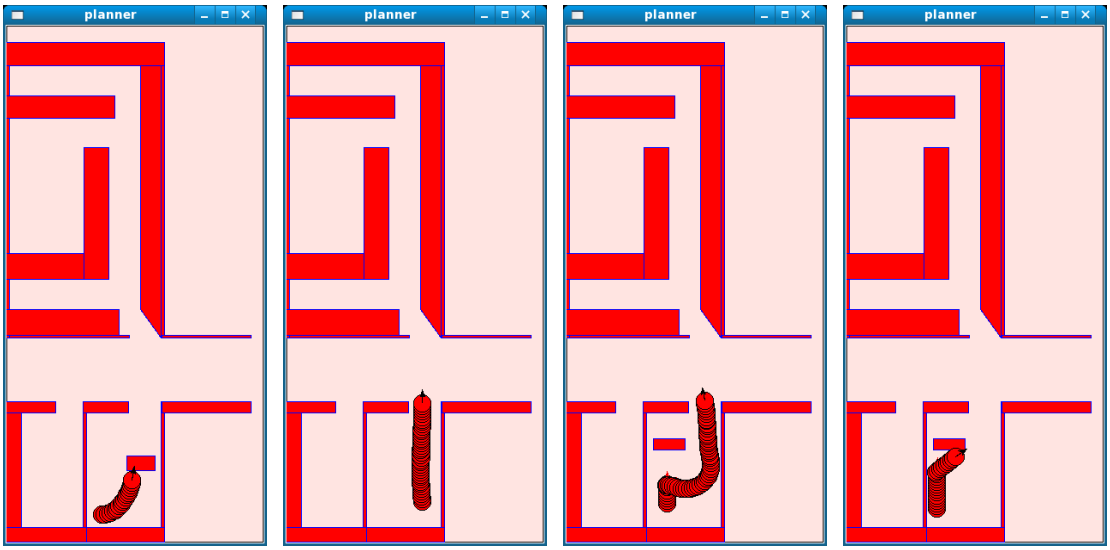


(a) (b) (c) (d)

Figura 7.36. Ejecuciones del árbitro de comportamientos del experimento 16. (a) Posición inicial derecha inferior por debajo del obstáculo, con ángulo de 90°. (b) Posición inicial izquierda inferior, con un ángulo inicial de 90°. (c) Posición inicial izquierda inferior, con un ángulo inicial de 90°, no logra salir de la habitación. (d) Posición inicial derecha inferior por debajo del obstáculo, con ángulo de 0°.

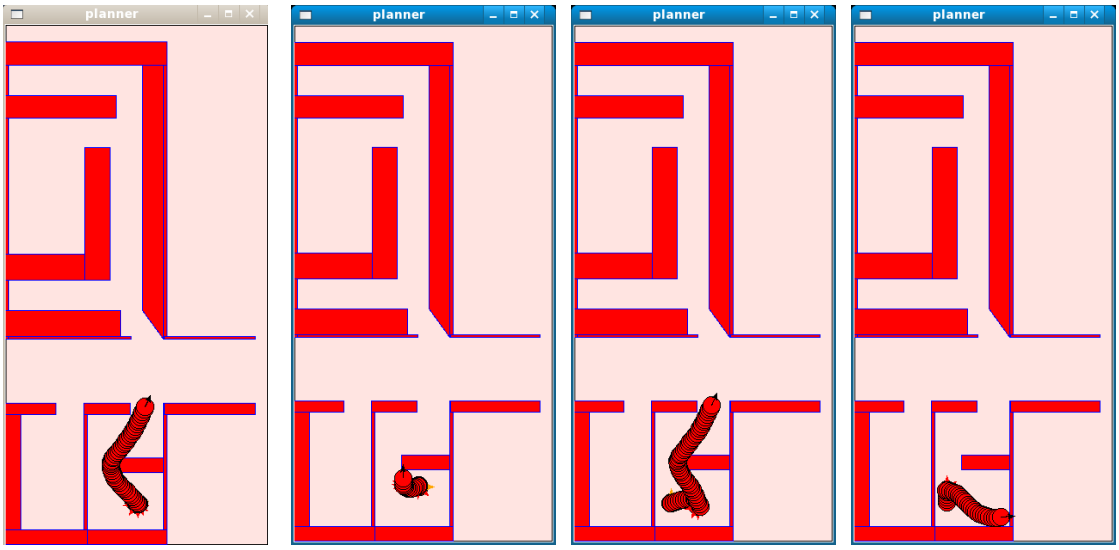
(a) (b) (c) (d)

Figura 7.37. Ejecuciones del árbitro de comportamientos del experimento 16. (a) Posición inicial derecha inferior por debajo del obstáculo, con ángulo de 0°, pero no logra salir de la habitación. (b) Posición inicial izquierda inferior, con un ángulo inicial de 270°. (c) Posición inicial izquierda inferior, con un ángulo de 270°, pero falla en la ejecución. (d) Posición inicial izquierda inferior, con un ángulo de 0°.



(a) (b) (c) (d)

Figura 7.38. Ejecuciones del árbitro de comportamientos del experimento 16. (a) Posición inicial izquierda inferior, con un ángulo de 0°, pero falla en la ejecución. (b) Posición inicial derecha inferior sin obstáculo, con un ángulo inicial de 90°. (c) Posición inicial izquierda inferior por debajo del obstáculo desplazado a la izquierda, con un ángulo inicial de 90°. (d) Posición inicial izquierda inferior por debajo del obstáculo desplazado a la izquierda, con un ángulo inicial de 90°, falla en la ejecución.



(a) (b) (c) (d)

Figura 7.39. Ejecuciones del árbitro de comportamientos del experimento 16. (a) Posición inicial derecha inferior, por debajo del obstáculo agrandado a casi el doble de tamaño, con ángulo de 0°. (b) Posición inicial derecha inferior, por debajo del obstáculo agrandado a casi el doble de tamaño, con ángulo de 0°, falla en la ejecución. (c) Posición inicial izquierda inferior al lado del obstáculo agrandado a casi el doble de tamaño, con un ángulo de 90°. (d) Posición inicial izquierda inferior al lado del obstáculo agrandado a casi el doble de tamaño, con un ángulo de 90°, falla en la ejecución.

Nótese que los árbitros de comportamientos de los experimentos 16 y 25 logran ejecutar correctamente la mayoría de los casos. Véase de la tabla 7.30 los cinco primeros casos de ejecución de la red neuronal entrenada bajo *backpropagation*, se tienen 8 ejecuciones fallidas por 100 ejecuciones que están dentro del dominio de entrenamiento. En cuanto a los casos fuera del dominio de entrenamiento, se tienen 4, correspondientes a los últimos cuatro renglones de la tabla 7.30, ahí se observan 60 ejecuciones fallidas por 80 ejecuciones.

En el caso de la red neuronal evolucionada bajo EGA se observan 0 ejecuciones fallidas por 20 ejecuciones que están en el dominio de entrenamiento, correspondiente al primer caso de la tabla 7.35, ya que sólo fue evolucionada para “aprender” una sola ruta de navegación, los demás casos se encuentran fuera de este dominio, y se cuentan 43 ejecuciones fallidas por 160 ejecuciones. Véase que esta red neuronal generalizó bien al seleccionar los comportamientos adecuados para que se cumpliera el comportamiento global.

Por otra parte, para explicar el número de ejecuciones fallidas en la red neuronal entrenada bajo *backpropagation*, bien se podría argumentar contra el diseñador al no facilitar el número suficiente de datos de entrenamiento, al no presentar el diseño óptimo de la red neuronal, o al no encontrar los valores óptimos de los parámetros de la red neuronal.

De estas comparativas podemos concluir que, evidentemente es importante el buen entrenamiento del árbitro de comportamientos, pero también es importante seleccionar los comportamientos adecuados. En ambos casos (experimento 16, experimento 25), los árbitros de comportamientos ejecutaron adecuadamente los casos expuestos, siendo 112 el

número de ejecuciones correctas para la red neuronal entrenada bajo *backpropagation*, de un total de 180 ejecuciones, y 137 ejecuciones correctas para la red neuronal evolucionada.

También véase que bajo el método propuesto en esta tesis, no fue necesaria una "intuición" de parte del diseñador para hacer el mapeo óptimo lecturas_láser -> activación_comportamientos, ya que el AG por medio de la señal de corrección se encargó de encontrar el modelo bajo el cual la red neuronal árbitro pudiera escoger el comportamiento adecuado en el momento adecuado, y no fue necesaria la intervención de un diseñador inexperto que le dijera al robot qué hacer en cada situación, el EGA encontró la secuencia cuasi óptima de activación de comportamientos.

Con respecto a ventajas y desventajas técnicas se mide el factor *costo de desarrollo* de ambas técnicas. Para la metodología presentada en este trabajo se tuvo que programar el algoritmo genético ecléctico, lo cual implica un tiempo de análisis del algoritmo genético, un tiempo de programación, además de que cada evolución se tardó entre 2 y 3 horas, con la posibilidad de tener que repetir la simulación por no encontrar, a criterio del diseñador, un individuo que satisfaciera el comportamiento global deseado.

Por otro lado, bajo la metodología clásica conexionista, no se programó el algoritmo de *backpropagation*, ya que se utilizó el software *DateEngine* para el entrenamiento de la red neuronal; sin embargo, si se requirió de un tiempo para el análisis de los comportamientos simples y del diseño de la red neuronal, sobretodo para hacer el conjunto de datos de entrenamiento, y por consiguiente, las pruebas necesarias de ensayo-error para afinar los parámetros inherentes a este modelo con el fin de reducir la tasa de error de aprendizaje.

CONCLUSIONES Y TRABAJO FUTURO

De los experimentos realizados, se puede observar que el robot aprende la ruta de navegación deseada pero sin haberle dicho a priori los movimientos deseados, el robot por sí solo explora varias rutas de navegación hasta encontrar una que satisfaga la función de *fitness* del algoritmo genético.

Cabe mencionar que al hacer el análisis de una evolución, no sólo se debe observar que se haya cumplido el criterio de convergencia, sino observar los comportamientos emergentes que se obtuvieron. En una sola evolución se pueden obtener comportamientos que cumplen con el objetivo deseado, pero que posiblemente sean muy diferentes, como el caso del experimento 6, donde el robot llega a su punto destino pero a través de rutas totalmente distintas.

Una ventaja de utilizar el paradigma evolutivo en conjunto con el modelo conexionista sobre el paradigma puramente conexionista, es que no hay que dar valores iniciales a ningún parámetro relacionado con el aprendizaje de la red neuronal, sólo algunos parámetros iniciales del algoritmo genético, y con la ventaja de que es auto-adaptativo eliminamos la incertidumbre de haber puesto valores erróneos.

La desventaja del modelo evolutivo es que es mucho más tardado hacer una evolución que aplicar el algoritmo de *backpropagation*. Otra clara desventaja es la complejidad de manejar un algoritmo genético para entrenar las redes neuronales, y asociado a estas dos desventajas está implícito el tiempo de cómputo.

Las ventajas del modelo del árbitro de comportamientos expuesto en esta tesis son su facilidad de incorporar comportamientos, de entrenamiento, de programación y sobretodo, la escalabilidad que posee, ya que después podemos tener un nivel superior al árbitro y que sería un árbitro de árbitros, y así sucesivamente (todo esto se había mencionado en el capítulo 5. La clara desventaja es saber qué comportamientos incorporar, y cuáles no incorporar al hacer una evolución, así como la programación del algoritmo genético y el tiempo que tarda una evolución.

De la comparación de los algoritmos de *backpropagation* vs. EGA, se puede ver que para entrenar adecuadamente la red neuronal bajo el algoritmo de *backpropagation* y con los comportamientos simples de navegación dados, es necesaria una “intuición” de parte del diseñador para hacer un buen mapeo de valores del sensor a selección de comportamientos, o en su defecto, hacer varias pruebas de ensayo-error para visualizar qué comportamientos deben activarse en el momento indicado.

Como trabajo a futuro, se propone crear una base de comportamientos, y posterior una base de árbitros, para que al hacer alguna evolución no se parta desde cero. Si se logra esto, se facilitaría mucho la navegación por entornos semi-estructurados.

La motivación de largo plazo es que una vez obtenidos los comportamientos de navegación para un entorno, el robot pueda hacer mapas conceptuales de éste, así la navegación del robot sería más parecida al procesamiento cognitivo humano del espacio. Después, lograr que los robots móviles puedan navegar por ambientes no estructurados, cambiantes con el tiempo y totalmente inciertos al aproximar la inteligencia de un robot a una forma de inteligencia cognitiva humana, esto es, programar aproximaciones de procesos cognitivos de los seres humanos en los robots y mediante un sistema de visión el robot podría conocer en qué tipo de ambiente se encuentra, navegar y operar en él sin necesidad de construirle un mapa.

Apéndice A

Evolución y entrenamiento de redes neuronales árbitro mediante EGA y Backpropagation

Evolución de los pesos de redes neuronales que codifican árbitros de comportamientos

El simulador utilizado para evolucionar los pesos de una red neuronal está programado bajo los lenguajes C y C++, y el ambiente gráfico bajo el lenguaje Tcl/Tk, por lo cual es necesario ejecutarlo bajo un ambiente linux. Las pruebas aquí mostradas fueron llevadas a cabo en una máquina virtual con Fedora 8. A continuación se presentan los pasos necesarios para llevar a cabo una evolución.

1) Si se requiere cambiar el número de acciones que tiene permitido ejecutar cada red neuronal al tomar el control del robot para ser evaluada, se debe ir a la ruta: /robots/robots/ViRbot y abrir el archivo GoToGenetics.cpp, buscar la variable 'stepCounter' ubicada dentro de un ciclo while(). Para cambiar el ángulo de inicio que tendrá el robot se debe buscar, en este mismo archivo, la variable 'theta' ubicada dentro de un ciclo for() que sirve para la evaluación de los individuos por generación.

2) Crear un archivo .h para guardar la red neuronal que codifica el árbitro de comportamientos. A continuación se presenta un ejemplo de este archivo:

```
/*****************************************************
 *                                                   *
 *    RED NEURONAL                                   *
 *    arbitroComportamientos.h                       *
 *                                                   *
```

```
 ******************************************************/

#include <math.h>

// A continuación se incluyen los archivos .h donde estan guardados //los
comportamientos simples de navegación, típicamente otras redes
//neuronales para este trabajo de tesis

#include "comportamiento1.h"
#include "comportamiento2.h"
//...
#include "comportamiento3.h"


// A continuación se declaran variables para la red neuronal


float pc, pm, wSIH1, wSIH2, wSDH1, wSDH2, wBiasH1, wBiasH2, wH1Out,
wH2Out, origen;


// La siguiente función realiza el promedio de los 341 láser del lado
//izquierdo y 341 láser del lado derecho del robot

void get_left_right_range_laser(int num_sensors, Raw observations, float
*left,float *right){
 int i,j;
 float sd=0,si=0;

for(i=0,j=1; i<341; i++,j++){

        sd=sd + observations.sensors[i];
 }


for(i=341,j=1; i<682; i++,j++){

        si=si + observations.sensors[i]/var;

 }

   *left=si/341;
   *right=sd/341;
}


// Las siguientes dos funciones realizan un promedio de sensores sonar
//e infrarrojos, respectivamente, funciones no utilizadas en este
//trabajo

void get_left_right_range_sonar(int num_sensors, Raw observations, float
*left,float *right){
 int i,j;
 float var;
 float sd=0,si=0;

 for(i=1;i<=num_sensors/2;i++){
```

```
      //var=2<<(i-1);
      var=i;
      sd=sd + observations.sensors[i]/var;
 }

 for(i=1,j=num_sensors;i<=num_sensors/2;i++,j--){
      //var=2<<(i-1);
      var=i;
      si=si + observations.sensors[j]/var;
 }


 *left = 2*si/num_sensors;
 *right= 2*sd/num_sensors;

}


void get_left_right_range_infrared(int num_sensors, Raw observations,
float *left,float *right){
 float SI,SD,SC;

 SD = observations.sensors[1];
 SC = observations.sensors[2];
 SI = observations.sensors[3];

 *left = (SI+SC)/2.0;
 *right= (SD+SC)/2.0;

}


//EN ESTA FUNCION SE COMPUTA LA RED NEURONAL
//el nombre de la siguiente función puede ser cambiado, al hacer esto
//se debe poner el nombre actual en el archivo ‘behaviors.h’ (ubicado
//en la misma ruta que este archivo) dentro de la función
//‘paralell_behaviors()’ en las líneas 243 y 246, mismas donde se
//invoca a la red neuronal presente. También, el nombre de este //archivo
‘arbitroComportamientos.h’ debe ser incluido en el archivo //antes
mencionado

Behavior  nombreFuncion(char *sensor, int num_sensors, int *num_obs, Raw
observations, int reset,CONSTPOTENCIALES *constante){

 Behavior gen_vector;
 static int num_steps=0;
 static float left,right;
 //DECLARACION DE LAS NEURONAS
 float SI, SD; //NEURONAS DE ENTRADA
 float H1, H2; //NEURONAS ESCONDIDAS
 float Out; //NEURONA DE SALIDA


 if(!strcmp(sensor, "ir"))
get_left_right_range_infrared(num_sensors,observations,&left,&right);
 else if(!strcmp(sensor, "sonar"))
get_left_right_range_sonar(num_sensors,observations,&left,&right);
```

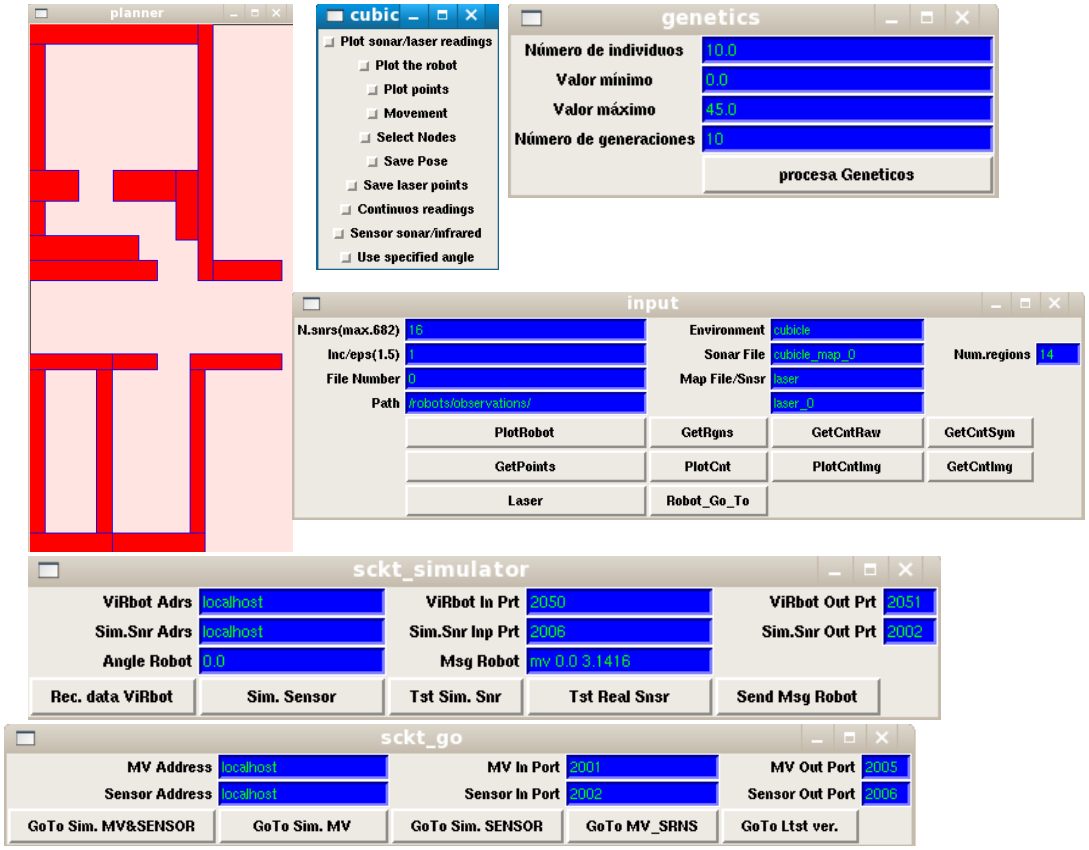
```
 else if(!strcmp(sensor, "laser"))
get_left_right_range_laser(num_sensors,observations,&left,&right);


 //Se obtienen los pesos evolucionados y se guardan en las variables
//declaradas al principio

 pc=constante->cromosoma[0].getFloatValue();
 pm=constante->cromosoma[1].getFloatValue();
 wSIH1=constante->cromosoma[2].getFloatValue();
 wSIH2=constante->cromosoma[3].getFloatValue();
 wSDH1=constante->cromosoma[4].getFloatValue();
 wSDH2=constante->cromosoma[5].getFloatValue();
 wBiasH1=constante->cromosoma[6].getFloatValue();
 wBiasH2=constante->cromosoma[7].getFloatValue();
 wH1Out=constante->cromosoma[8].getFloatValue();
 wH2Out=constante->cromosoma[9].getFloatValue();
 wBiasOut=constante->cromosoma[10].getFloatValue();
 origen=constante->cromosoma[11].getFloatValue();

 //LOS PROMEDIOS IZQUIERDO Y DERECHO DE LOS SENSORES SON PASADOS A LAS
//NEURONAS DE ENTRADA
 SIarbitro = left;
 SDarbitro = right;

 //SE COMPUTAN LA SUMATORIA DE LAS NEURONAS ESCONDIDAS
 H1 = SI*wSIH1 + SD*wSDH1 + 1*wBiasH1;
 H2 = SI*wSIH2 + SD*wSDH2 + 1*wBiasH2;

 //SE COMPUTA: TANH(SUMATORIA)
 H1 = tanh(H1);
 H2 = tanh(H2);

 //SE COMPUTA LA SUMATORIA DE LA NEURONA DE SALIDA
 Out = H1*wH1Out + H2*wH2Out + 1*wBiasOut;


 //SE COMPUTA: TANH(SUMATORIA)
 Out = tanh(Out);

 //Se escala la salida al número de comportamientos simples que se
//tengan, 3 por ejemplo
 Out = Out*3;

//Se divide el intervalo de salida entre en el número de
//comportamientos simples

if(Out >= -3  && Out < -1)
{
//se llama al comportamiento C1 y se le pasan los promedio de los //láser

gen_vector = C1(left, right);
}


else if(Out >= -1 && Out < 1)
{
```

```
//se llama al comportamiento C2 y se le pasan los promedio de los //láser

gen_vector = C2(left, right);
}


else if(Out >= 1 && Out <= 3)
{
//se llama al comportamiento C3 y se le pasan los promedio de los //láser

gen_vector = C3(left, right);
}


 return gen_vector;

} //Fin del archivo
```

3) Ir a la ruta: /robots/tcltk/tk8.4.5/unix y ejecutar el siguiente archivo:

$./wish cubicle.tcl

Aparecerán las siguientes ventanas:



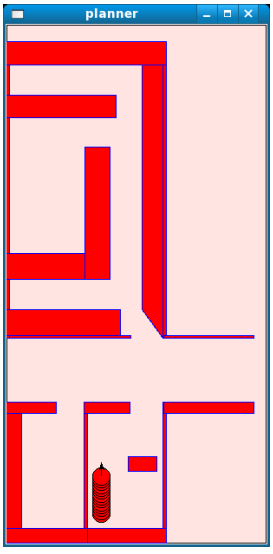
Los datos a ingresar son los siguientes:

- *Ambiente*. En la ventana ‘input’ en la casilla ‘Environment’ se especifica el ambiente sobre el cual se evolucionará el árbitro de comportamientos.
- *Punto de inicio y punto destino*. Una vez seleccionado el ambiente, se da un clic con el botón izquierdo del mouse, indicando el punto del cual partirá el robot cuando sea controlado por cada red neuronal, después se da un clic con el botón derecho del mouse indicando el punto al cual se quiere que arribe el robot.
- *Parámetros del AG*. El número de individuos, el número de generaciones y los valores mínimo y máximo que puede tomar cada gen de un cromosoma se indican en la ventana ‘genetics’.

Una vez que se dieron los datos anteriores se oprime el botón ‘procesa Geneticos’ ubicado en la ventana ‘genetics’. En seguida aparecerá un comando en consola, copiarlo y ejecutarlo desde la ruta: /robots/robots/ViRbot. La evolución ya habrá comenzado.

Entrenamiento de una red neuronal árbitro mediante backpropagation

1) El diseñador debe crear el conjunto de datos de entrenamiento, para lo cual se deja al robot andar por el ambiente deseado y que vaya tomando lecturas a cada tiempo *t*, después a estas lecturas se les adjuntarán los comportamientos a activar. Lo que se busca es el mapeo de lecturas láser a activación de comportamientos. Un ejemplo de esto es la siguiente figura:



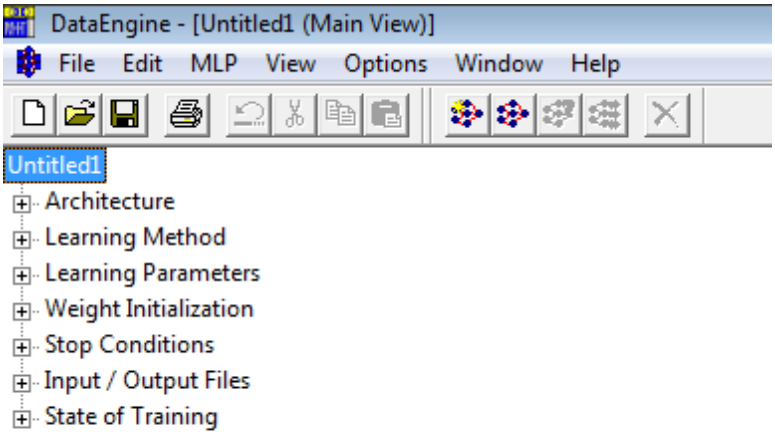
En esta figura se muestra una pequeña ruta cubierta por el robot, a su paso el robot guarda las lecturas del láser en un archivo, a estas lecturas se les adjuntarán posteriormente el o los comportamiento(s) que se requiere(n) activar bajo dichas lecturas. Esto se realiza a manera de cubrir la región del ambiente en donde se quiere entrenar a la red neuronal. Una vez que

se tienen los pares entrada-salida de la red neuronal se procede al entrenamiento bajo el algoritmo de backpropagation con el software *DataEngine*.

2) Exportar los pares entrada-salida obtenidos en el paso 1 a un archivo de Excel.

3) Abrir el software DataEngine y seleccionar, de la barra superior, la ruta: *File -> Import -> MS Excel...* con lo cual se importará el archivo obtenido en el paso 2. Guardar como archivo tipo *.dat*.

4) Seleccionar, de la barra superior, la ruta: *File -> New -> Multilayer Perceptron* con lo cual se creará un nuevo proyecto para entrenar una red neuronal. A continuación aparecerá la siguiente ventana.



5) En las pestañas que aparecen, se tienen opciones para determinar la arquitectura de la red neuronal, así como ajustar los parámetros necesarios.

6) Para empezar el entrenamiento, seleccionar, de la barra superior, la ruta *MLP -> Training* con lo cual se entrenará la red neuronal bajo el algoritmo de backpropagation.

Bibliografía

[1] Anderson James (2007). Redes Neurales. Primera edición. Alfaomega.

[2] Arámbula Cosío F. y Castañeda M. A. (2003). Autonomous Robot Navigation using Adaptive Potential Fields. *Mathematical and Computer Modelling*. Elsevier.

[3] Brooks, R. A. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation, Vol. 2, No. 1*, pp. 14–23.

[4] Brooks, R. A. (1990). Elephants Don't Play Chess. *Robotics and Autonomous Systems (6)*, pp. 3–15.

[5] Buchanan Bruce (2005). A (Very) Brief History of Artificial Intelligence. *American Association for Artificial Intelligence*.

[6] F. Morales Eduardo. Búsqueda, Optimización y Aprendizaje, en: *http://ccc.inaoep.mx/~emorales/Cursos/Busqueda04/principal.html*. Ciencias Computacionales, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Puebla, México.

[7] Floreano Dario, Dürr Peter y Mattiussi Claudio (2008). Neuroevolution: from architectures to learning. Springer-Verlag.

[8] Harvey Inman (1997). Artificial Evolution and Real Robots. School of Cognitive and Computing Sciences. University of Sussex.

[9] Haykin Simon (1999). Neural Networks A Comprehensive Foundation. Prentice-Hall International.

[10] Kuri M. Ángel, Villegas Q. C. (1998). A Universal Eclectic Genetic Algorithm. *Sixth Iberoamerican Conference on Artificial Intelligence*.

[11] Kuri M. Ángel (2000). A Comprehensive Approach to Genetic Algorithms in Optimization and Learning. Theory and Applications. Volume II: Applications. Centro de Investigación en Computación, Instituto Politécnico Nacional.

[12] Laureano C. Ana Lilia (2000). Interacción Dinámica en Sistemas de Enseñanza Inteligentes. Tesis Doctoral. UNAM.

[13] Marcellin Jacques Sergio (2010). Notas del Curso: "*Construcción de Sistemas Expertos*". Posgrado en Ciencia e Ingeniería de la Computación, UNAM.

[14] Martín del Brío Bonifacio, Sanz M. Alfredo (2007). Redes Neuronales y Sistemas Borrosos. Tercera edición. Alfaomega.

[15] Massa Germán, Vinuesa Hernán y Lanzarini Laura (2007). Modular Creation of Neural Networks for Autonomous Robot Control. *Inteligencia Artificial: Revista Iberoamericana de Inteligencia Artificial*, Vol. 11, no. 35, pp. 43-53.

[16] Mitchell M. (1996). An Introduction to Genetic Algorithms. Cambridge, MA. MIT Press.

[17] Mondada Francesco y Floreano Dario (1995). Evolution of neural control structures: some experiments on mobil robots. *Robotics and Autonomous Systems*. Laboratory of Microcomputing (LAMI). Swiss Federal Institute of Technology.

[18] Montes G. Fernando, Flandes E. Daniel y Pellegrin Z. Luis (2008). Action Selection and Obstacle Avoidance using Ultrasonic and Infrared Sensors. *In Frontiers in Evolutionary Robotics.* Edited by Hitoshi Iba, pp. 327-340.

[19] Morales A. Efrén (2009). Algoritmos Genéticos para la generación de comportamientos en robots móviles. Tesis de maestría. UNAM.

[20] Nolfi Stefano y Parisi Domenico (1994). Evolving non-Trivial Behaviors on Real Robots: an Autonomous Robot that Picks up Objects. *Proceedings of the 4th Congress of the Italian Association of Artificial Intelligence.* Berling: Springer-Verlag, pp. 243-254.

[21] Nolfi Stefano, Floreano Dario, Miglino Orazio y Mondada Francesco (1994). How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics. *Proceedings of the International Conference Artificial Life IV*. MIT Press, pp. 190-197.

[22] Pratihar Dilip (2003). Evolutionary Robotics – A review. *In: Sadhana*. Vol. 28, parte 6, pp 999-1009.

[23] Rudolph Günter (1997). Convergence Analysis of Canonical Genetic Algorithms. *IEEE Trans. Neural Networks.* Special issue on Evolutionary Computation.

[24] Russell Stuart y Norving Peter (2004). Inteligencia Artificial, un enfoque moderno. 2ª. Edición. Pearson Prentice Hall.

[25] Sandholt Hans (2002). A Study of the Generation and Organization of Behaviors for Autonomous Robots. Chalmers University of Technology. 2004.

[26] Santos José y Duro Richard (2005). Evolución Artificial y Robótica Autónoma. Alfaomega.

[27] Savage Jesús y Anaya Rubén. Prácticas de Diseño de Robots Móviles. Facultad de Ingeniería, UNAM.

[28] Simon Herbert A. (2006). Las ciencias de lo artificial. Colección *La razón durea*. Serie *Obras Clave* n. 1, Granada.

[29] Stephens Christopher y Poli Riccardo (2009). Taming the complexity of Evolutionary Dynamics. Springer.

[30] Uchibe Eiji, Yanase Masakazu y Asada Minoru (2002). Evolution for Behavior Selection Accelerated by Activation/Termination Constraints. *Lecture Notes in Computer Science*. Robocup 2001: Robot Soccer World Cup V. Springer Berlin / Heidelberg, pp. 51-71.

[31] Vasudevan Shrihari, Gächter Stefan, Harati Ahad y Siegwart Roland (2007). A Hierarchical Concept Oriented Representation for Spatial Cognition in Mobile Robots. *Lecture Notes in Computer Science*. Springer-Verlag, pp. 243-256.

[32] Wahde Mattias (2002). An Introduction to Adaptive Algorithms and Intelligent Machines. 2a. Edición. Chalmers University of Technology.

[33] Wahde Mattias, Pettersson Jimmy, Sandholt Hans y Wolf Krister (2005). Behavioral Selection Using the Utility Function Method: A Case Study Involving a Simple Guard Robot. *Proceedings of the 3rd International Symposium on Autonomous Minirobots for Research and Edutainment.* K. Murase et all editores, pp. 261-266.