Universidade da Coruña
FACULTADE DE INFORMÁTICA
Departamento de Computación

TESIS DOCTORAL

APROXIMACIÓN EVOLUTIVA PARA LA OBTENCIÓN INCREMENTAL DE ARQUITECTURAS MODULARES DE COMPORTAMIENTOS EN ROBOTS AUTÓNOMOS

José Antonio Becerra Permuy

UNIVERSIDADE DA CORUÑA

FACULTADE DE INFORMÁTICA

Departamento de Computación

TESIS DOCTORAL

APROXIMACIÓN EVOLUTIVA PARA LA OBTENCIÓN INCREMENTAL DE ARQUITECTURAS MODULARES DE COMPORTAMIENTOS EN ROBOTS AUTÓNOMOS

Autor: José Antonio Becerra Permuy

Directores: José Santos Reyes

Richard Duro Fernández

Fecha: Julio de 2003

UNIVERSIDADE DA CORUÑA

DEPARTAMENTO DE COMPUTACIÓN

D. José Santos Reyes y D. Richard J. Duro Fernández, Profesores Titulares de Universidad del Departamento de Computación de la Universidade da Coruña,

CERTIFICAN:

Que la memoria titulada "Aproximación evolutiva para la obtención incremental de arquitecturas modulares de comportamientos en robots autónomos" ha sido realizada por D. José Antonio Becerra Permuy bajo nuestra dirección en el Departamento de Computación de la Universidade da Coruña, y constituye la Tesis que presenta para optar al grado de Doctor.

Fdo. José Santos Reyes
Codirector de la Tesis Doctoral

Fdo. Richard J. Duro Fernández
Codirector de la Tesis Doctoral

Fdo. José Luis Freire Nistal
Director del Departamento de Computación

Agradecimientos

A José Santos Reyes y Richard Duro Fernández por su excelente trabajo de dirección, inagotable apoyo y todo el esfuerzo invertido en mi formación como investigador.

A mis padres, mi hermano y mi tía Marina Permuy por ayudarme en todo lo necesario a lo largo de estos años y por soportar mi mal humor en la etapa final de esta tesis.

A mis compañeros del Grupo de Sistemas Autónomos, del Laboratorio de Inteligencia Artificial del Departamento de Computación y de la empresa Universo Digital por el inmejorable ambiente de compañerismo del que he podido disfrutar en todo momento.

A José Baltasar García, Pilar Pérez, Marta Penas y María del Carmen Rodríguez por sus continuos ánimos.

Publicaciones

La investigación realizada en este trabajo ha dado lugar a las siguientes publicaciones científicas:

- J. A. Becerra, J. L. Crespo, J. Santos, R. J. Duro, "Incremental Design of Neural Controllers for an Infrasensorized Autonomous Robot", *Proceedings of Wiener's Cybernetics: 50 years of evolution*, pp. 163-166, 1999.
- J. L. Crespo, J. A. Becerra, R. J. Duro, J. Santos, "Visual Tracking in a Real Robot Through Higher Order Synapses", *Proceedings of Wiener's Cybernetics: 50 years of evolution*, pp. 167-169, 1999.
- J. A. Becerra, J. Santos, R. J. Duro, "Using Temporal Information in ANNs for the Implementation of Autonomous Robot Controllers", *Lecture Notes in Computer Science*, vol. 1607, pp. 540-547, Springer-Verlag, 1999.
- J. A. Becerra, J. Santos, R. J. Duro, "Progressive Construction of Compound Behavior Controllers for Autonomous Robots Using Temporal Information", *Lecture Notes in Artificial Intelligence*, vol. 1674, pp. 324-328, Springer-Verlag, 1999.
- J. Santos, R. J. Duro, J. A. Becerra, J. L. Crespo, F. Bellas, "Aspects of Evolution for Obtaining Real Robot Controllers", *Proceedings of Joint Conference on Information Sciences'2000*, pp. 1023-1026, 2000.
- R. J. Duro, J. Santos, J. A. Becerra, F. Bellas, J. L. Crespo, "Using Higher Order Synapses and Nodes to Improve the Sensing Capabilities of Mobile Robots", *Proceedings of European Symposium of Artificial Neural Networks'2000*, pp. 81-88, 2000.
- F. Bellas, J. A. Becerra, J. Santos, R. J. Duro, "Applying Synaptic Delays for Virtual Sensing and Actuation in Mobile Robots", *Proceedings of International Joint Conference on Neural Networks '2000*, 2000.
- R. J. Duro, J. Santos, J. A. Becerra, "Evolving ANN Controllers for Smart Mobile Robots", *Future Directions for Intelligent Systems and Information Sciences*, pp. 34-64, Springer-Verlag, 2000.
- J. A. Becerra, J. Santos, "Evolución Distribuida en la Obtención de Controladores para Robots Autónomos", *Actas del Simposio Español de Informática Distribuida'2000,* pp. 219-226, 2000.
- R. J. Duro, J. A. Becerra, J. Santos, "Improving Reusability of Behavior Based Robot Cognitive Architectures Obtained through Evolution", *Proceedings of Advanced Space Technologies for Robotics and Automation'2000*, 2000.
- J. Santos, R. J. Duro, J. A. Becerra, J. L. Crespo, F. Bellas, "Considerations in the Application of Evolution to the Generation of Robot Controllers", *Information Sciences*, vol. 133, pp. 127-148, Elsevier, 2001.
- F. Bellas, J. A. Becerra, R. J. Duro, "Using Evolution for Thinking and Deciding", *Advances in Fuzzy Systems and Evolutionary Computation*, pp. 242-247, World Scientific and Engineering Society Press, 2001.

- R. J. Duro, J. A. Becerra, J. Santos, "Behavior Reuse and Virtual Sensors in the Evolution of Complex Behavior Architectures", *Theory in Biosciences*, vol. 120, pp. 188-206, Urban & Fischer, 2001.
- J. R. Luaces, J. A. Becerra, R. J. Duro, P. González, I. López, "Managing Distributed Resources in the SVG Project", *Proceedings of Parallel Distributed and Network-based Processing'2002*, 2002.
- F. Bellas, R. J. Duro, J. A. Becerra, "Funciones de Calidad Muestreadas en Algoritmos Evolucionistas", *Actas del AEB'2002*, pp. 375-381, 2002.
- J. A. Becerra, J. Santos, R. J. Duro, "MA vs. GA in Low Population Evolutionary Processes with Mostly Flat Fitness Landscapes", *Proceedings of Joint Conference on Information Sciences'2002*, pp. 626-630, 2002.
- F. Bellas, J. A. Becerra, R. J. Duro, "Sampled Fitness Functions in Complex Problems", *Proceedings of Joint Conference on Information Sciences'2002*, pp. 631-638, 2002.
- J. A. Becerra, R. J. Duro, J. Santos, "Self Pruning Gaussian Synapse Networks for Behavior Based Robots", *Lecture Notes in Computer Science*, vol. 2415, pp. 837-843, Springer-Verlag, 2002.
- R. J. Duro, J. Santos, J. A. Becerra, "Some Approaches for Reusing Behaviour Based Robot Cognitive Architectures Obtained Through Evolution", *Biologically Inspired Robot Behavior Engineering*, pp. 239-260, Physica-Verlag, 2003.
- J. A. Becerra, J. Santos, R. J. Duro, "Multimodule Artificial Neural Network Architectures for Autonomous Robot Control Through Behavior Modulation", *Lecture Notes in Computer Science*, vol. 2687, pp. 169-176, Springer-Verlag, 2003.

Índice

1 Introducción

El objetivo fundamental del trabajo que se presenta en esta memoria es el establecimiento de una metodología y un conjunto asociado de herramientas y procedimientos que permitan obtener de forma estructurada, incremental y repetible sistemas de control para robots autónomos que han de realizar tareas en entornos dinámicos y no estructurados. Se pretende que dicha metodología y las estructuras que la soportan lleven a un procedimiento de obtención de los sistemas de control con la mínima intervención humana posible, y cuando ésta sea necesaria, que se limite a la especificación de los objetivos de una manera que resulte natural para el humano, sin tener que entrar en disquisiciones sobre cómo se ha de alcanzar el objetivo planteado en cada momento. En definitiva, se trata de permitir la obtención automática de los sistemas de control de robots y, por lo tanto, de todos los elementos que los componen, a partir de una indicación natural de los objetivos.

Por razones de eficiencia y de uso racional de recursos las estructuras a desarrollar para soportar la metodología habrán de contemplar una modularidad apropiada y una coherencia que lleve a la posibilidad de reutilización de elementos desarrollados para otras tareas del mismo robot o, incluso, de otros robots con morfologías distintas. En este sentido se pretende que todo lo que se vaya desarrollando pueda ser reutilizado permitiendo, de este modo, alcanzar sistemas de control complejos de forma incremental a partir de procesos individuales de obtención progresivamente más elaborados que hacen uso de estructuras diseñadas en las etapas anteriores y las complementan de una forma eficaz.

Un elemento a tener muy en cuenta en este trabajo es el hecho de que muchos objetivos de operación de un robot autónomo consisten en el desarrollo de comportamientos determinados durante largos períodos de tiempo y no en la consecución puntual de un determinado estado, esto es, se puede desear que un robot "explore" una zona o "siga una pared". Este tipo de objetivos presentan una alta degeneración en cuanto a su evaluación, es decir, existen múltiples conjuntos de acciones que llevan a resultados finales distintos pero sin una clara diferencia en cuanto a su adecuación al objetivo. Es, por lo tanto, de gran importancia desarrollar una estrategia de evaluación de los resultados obtenidos que permita alcanzar el objetivo propuesto.

Teniendo en cuenta los objetivos planteados, en el trabajo que se presenta se abordan básicamente cuatro problemas:

- La elección de una filosofía de base para la estructura de control de los robots que permita alcanzar los objetivos planteados.
- La definición de una algoritmia de automatización del proceso de diseño a partir de unas especificaciones de tarea a realizar.

- La definición de los paradigmas y elementos base que utilizaremos para la implementación de las arquitecturas de control.
- La especificación y formalización de la interacción entre elementos de la arquitectura y la definición de su proceso de construcción.

En cuanto a la elección de una filosofía de base para la estructura de control de los robots se tienen en cuenta dos aspectos que consideramos fundamentales: los robots para los que se desarrollan las estructuras de control son autónomos y van a operar en entornos dinámicos y no estructurados. Además, la estructura de control tiene que poder diseñarse de forma automática a partir de una especificación de tarea.

Estos aspectos conllevan una serie de requisitos en cuanto a la filosofía a utilizar. Del concepto de robot autónomo, que entendemos como aquel que debe operar correctamente y sin supervisión ante una amplia variedad de condiciones no consideradas en su programación previa, se puede deducir que la estructura de control ha de permitir la operación del robot y la consecución de las tareas asignadas al mismo sin supervisión constante por parte de humanos en entornos no totalmente conocidos, que cambian con el tiempo y con múltiples fuentes de ruido en cuanto a las percepciones del robot. Por otra parte, del hecho de que esta arquitectura se pueda diseñar de forma automática resulta la necesidad de disponer de información suficiente para poder evaluar automáticamente la calidad relativa de los diseños que se van obteniendo de modo que se puedan conseguir los mejores posibles. Es por lo tanto necesario contar con magnitudes medibles en este sentido.

Esto nos lleva a contemplar la utilización de filosofías de control basadas en la tarea, cuya consecución puede ser medible y que, en definitiva, es lo que cuenta para el robot, y no en el conocimiento que la soporta, difícilmente medible y cuya elicitación es innecesaria para la operación del sistema. Es por ello por lo que el trabajo se enmarca en el campo de la robótica basada en comportamientos en contraposición con filosofías más tradicionales basadas en conocimiento.

Sin embargo, aunque la metodología desarrolla controladores globales basados en comportamientos por las razones que se indican, proporciona una estructura formal que permite la utilización de módulos basados en cualquiera de dichas aproximaciones allí donde se requieran. Por lo tanto, más que contraponer estas filosofías, en este trabajo se busca una complementariedad al permitir la reutilización de componentes y desarrollos previos que han resultado exitosos y que, por medio de una simple encapsulación, pueden interaccionar con el resto de la arquitectura y participar en el proceso de diseño automático de una forma natural.

Respecto a la necesidad de la definición de una algoritmia de automatización del proceso de diseño a partir de unas especificaciones de tarea a realizar hay que tener en cuenta que si bien la robótica basada en comportamientos tiene como uno de sus principios la conexión directa entre sensores y actuadores, no parece razonable esperar que dicha conexión sea establecida por el humano, que se encontraría casi con los

mismos problemas que podría tener para plasmar eso mismo en forma de sistema basado en el conocimiento. Es más lógico el tratar de utilizar alguna herramienta que permita obtener automáticamente dicha conexión. De ahí que uno de los objetivos de este trabajo sea estudiar las herramientas adecuadas y desarrollar los mecanismos pertinentes para que dicha obtención automática sea factible a todos los niveles. Para ello, como herramienta de obtención automática de los controladores, se han seleccionado los algoritmos evolutivos. Éstos son ya ampliamente utilizados en la robótica basada en comportamientos, constituyendo la línea denominada robótica evolutiva, debido a que sus características los hacen adecuados para buscar soluciones a problemas que no pueden ser abordados analíticamente, como es éste el caso. Existen, sin embargo, muchos tipos de algoritmos evolutivos, cada uno con sus propias características, con lo que ha resultado necesario analizar la problemática de los distintos tipos de algoritmos evolutivos en este problema en concreto y ver cuál se adecua mejor. La problemática incluye no solamente los valores ideales para los parámetros que determinan el funcionamiento de cada algoritmo evolutivo, sino también otros factores importantes como dónde realizar la evaluación de los individuos (en el entorno real o en uno simulado), qué pasos se deben seguir para asegurar una transferencia correcta de comportamientos al robot real en caso de realizar la evaluación en un entorno simulado, tamaño de la población necesario, distribución inicial de ésta, efecto de dividir la población total en poblaciones más pequeñas que evolucionen parcialmente aisladas o qué aproximación seguir para calcular la calidad de los individuos de una forma eficiente y sin que lleve a *deceptividad* debido a la dinamicidad del entorno y, por lo tanto, de la función de calidad.

La definición de los paradigmas y elementos base que utilizaremos para la implementación de las arquitecturas de control es otro aspecto muy importante de este trabajo. Evidentemente, no sólo es necesario determinar cómo obtener los controladores, sino sobre qué base implementarlos. De los requisitos y objetivos formulados al inicio de esta introducción se deduce que lo que se requiere es un paradigma que permita una aproximación relativamente precisa a funciones extremadamente no lineales y que en muchos casos requieren tener en cuenta aspectos espaciales y temporales. En este trabajo se ha seleccionado como herramienta principal para dicha implementación las redes de neuronas artificiales (RNAs), aunque se ha tenido cuidado en el desarrollo de la metodología de no excluir la posible inclusión de módulos basados en otros paradigmas (programados, basados en reglas, fuzzy, etc.). Las RNAs, al igual que los algoritmos evolutivos, poseen una inspiración biológica y son fácilmente obtenibles de forma automática en combinación con éstos. Además, este tipo de estructuras, por su forma de operar altamente distribuida, poseen una cierta tolerancia al ruido, es decir, en este caso, a fallos en los sensores, que son habituales en los robots, y al ruido presente en los mismos. En este trabajo consideraremos distintas arquitecturas de RNAs, algunas de desarrollo propio, para permitir la realización de controladores que contemplen el procesado de información temporal, y mecanismos de selección de atención y de podado de manera que se disponga de los elementos básicos

para poder llevar a cabo las tareas habituales que se le pueden encomendar a un robot autónomo de manera eficiente.

La especificación y formalización de la interacción entre elementos de la arquitectura y la definición de su proceso de construcción es otro aspecto de gran importancia a considerar en este trabajo. El objetivo de permitir la reutilización de componentes y la construcción incremental de las estructuras de control nos llevará a la definición de una arquitectura modular. Este tipo de arquitectura contemplará comportamientos implementados mediante la conjunción de varios controladores (módulos) y su construcción podrá llevarse a cabo de forma incremental, con el fin de que sea fácil y rápido modificar o añadir comportamientos sobre los ya existentes. Obviamente, como en el proceso de definición de cualquier arquitectura multimódulo, será necesario estudiar y determinar los distintos tipos de relaciones que pueden darse entre los módulos. Por lo tanto, en este trabajo se estudian y formalizan dos tipos de relaciones, relaciones de activación-inhibición y relaciones de modulación, que, conjuntamente, proporcionan una base (en el sentido matemático) para el desarrollo de cualquier tipo de controlador multimódulo. Evidentemente, en el trabajo se trata también el procedimiento para obtener estas relaciones de forma automática llevando a una definición de cómo construir incrementalmente controladores complejos a partir de otros más simples ya existentes y de cómo transferir controladores entre distintos tipos de robots. Para la definición de este último procedimiento se introducen los conceptos de sensor y de actuador virtuales en relación a una arquitectura modular basada en comportamientos y se estudian las alternativas de introducción del interfaz correspondiente.

Así, esta memoria presenta una estructura de cuatro partes, donde el estado del arte correspondiente se presenta en cada una de ellas.

En una primera parte se introduce el marco general de este trabajo, detallando las propuestas de otros grupos para resolver los mismos problemas. En este apartado (capítulo 3) se describen las características de la robótica basada en comportamientos, en contraposición a la robótica basada en el conocimiento. Se introducen los problemas y ventajas de cada una y se justifica el porqué de la elección de la primera como marco en el cual desarrollar este trabajo, así como una primera descripción de cómo se pretenden resolver los problemas que presenta la robótica basada en comportamientos.

En una segunda parte (capítulo 4) se describe el enfoque empleado para abordar el problema de obtención automática de comportamientos para robots autónomos en términos de los elementos que la componen: cómo se implementarán los controladores y cómo se van a obtener. Previamente a la descripción de estos elementos metodológicos se verán las distintas alternativas que existen tanto para implementar los controladores como para obtenerlos automáticamente.

Como ya se ha mencionado, en este trabajo los controladores estarán formados por RNAs, que se obtendrán en un proceso evolutivo en el cual los individuos se

corresponderán con controladores candidatos para que el robot desempeñe su tarea, bien sean controladores de un solo módulo o multimódulo. El estudio de la aplicación de técnicas evolutivas a la construcción de sistemas de control para robots autónomos implica distintos aspectos tales como la asignación de calidad a los individuos, el tipo de operadores evolutivos que son adecuados para su aplicación a problemas con funciones de calidad muestreadas y variables en el tiempo, o el tipo y tamaño de poblaciones a utilizar.

Es evidente que la única posibilidad de evaluar un individuo consistirá en probar ese controlador en un entorno. El ideal desde un punto de vista de fiabilidad sería que esta evaluación se realizase en un entorno real con el controlador implementado en un robot real. En términos prácticos es mucho más eficiente y cómodo la utilización de entornos simulados. Surge entonces un problema en cuanto al cómo evaluar los individuos. En esta sección tratamos este problema y contemplamos en qué condiciones una simulación no exhaustiva permite obtener evaluaciones que lleven a controladores transportables al robot real en entornos reales. La simulación no puede ser muy precisa, o el coste computacional la hará prohibitiva. Por contra, si es muy sencilla el comportamiento puede ser inadecuado una vez trasladado a la realidad. Veremos, por ello, las condiciones necesarias para poder alcanzar un compromiso aceptable. Entre otras, deberán estar presentes diversos tipos de ruido (desviaciones sobre los valores teóricos) en todos los parámetros de la simulación que harán tolerante al controlador a las diferencias que hay entre el mundo real y el simulado. La utilización de ruido hará que la determinación de la calidad sea un proceso no totalmente determinista, es decir, la calidad de un individuo evaluado dos veces en un mismo entorno no tiene por qué ser la misma en ambas ocasiones. Esto trae consigo una serie de consecuencias que deberán tenerse muy en cuenta si no se quiere desvirtuar el proceso evolutivo como, por ejemplo, cuál es el número de evaluaciones por individuo necesario para que el cálculo de la calidad sea fiable.

Por otra parte, el propio proceso de cálculo de la calidad de un individuo, independientemente de lo comentado anteriormente, se puede concebir de diversas formas. Hay autores que planifican la asignación de calidad en términos de características deseadas en el comportamiento como, por ejemplo, maximización de la velocidad, minimización de distancia a un objeto, etc. Dado que esta aproximación implica por parte del diseñador un conocimiento muy preciso de las acciones que conforman una tarea y esto contradice el objetivo de que el planteamiento de la especificación de diseño resulte natural a un ser humano, en este trabajo se ha optado por un modelo energético, inspirado en las técnicas de Vida Artificial. Conceptualmente encaja perfectamente en la inspiración biológica que acompaña a las herramientas utilizadas y, lo que es realmente importante, ayuda al objetivo de tratar de minimizar el papel del humano en todo este proceso. Así, en cada comportamiento, se establecerán una serie de reglas de ganancia y/o pérdida de energía que indican al algoritmo

evolutivo de una forma cualitativa o semicuantitativa el objetivo a cumplir por el controlador, pero no cómo debe de hacerlo.

Todas estas decisiones y consideraciones referentes a cómo medir la calidad tendrán un reflejo en la forma del espacio de búsqueda de nuestros problemas. La selección adecuada de los algoritmos evolutivos a utilizar para obtener los controladores, en cuanto a cuáles son mejores, con qué parámetros y cuáles son sus limitaciones, forma parte fundamental de este trabajo. Es muy habitual encontrarse en los trabajos realizados dentro de este campo que los autores usen determinados algoritmos evolutivos de acuerdo al comportamiento de éstos en problemas tipo que, probablemente, en poco o nada se parezcan a los suyos o, simplemente, porque otros los han usado antes con éxito. Sin embargo, cada algoritmo evolutivo tiene sus puntos fuertes y sus puntos débiles. Dependiendo de cómo sea el espacio de búsqueda unos funcionarán mejor que otros. Por eso se ha realizado en este trabajo un análisis de las características de los espacios de búsqueda encontrados en el proceso de obtención de los controladores, determinando los aspectos relevantes de los mismos. Otros factores que se analizarán referentes a los algoritmos evolutivos son cómo se ven afectados éstos por situaciones de baja densidad poblacional (frecuentes en nuestros problemas, que son computacionalmente costosos), si realmente una distribución aleatoria inicial de los individuos es la mejor solución, cómo afecta el fenómeno de generación de razas a la evolución, cómo se ha de establecer el compromiso entre explotación y exploración, etc. Además, y dado ese coste computacional elevado, se paralelizará el software desarrollado para la implementación de todo este sistema, tanto para sistemas multiprocesador de memoria compartida como para sistemas de memoria distribuida, y se llevará a cabo un análisis de la eficiencia de dicha paralelización.

Para el estudio y comparación de distintas alternativas, así como para la implementación final de aquellas que se han considerado más adecuadas, en este trabajo se ha desarrollado un sistema completo de evolución y simulación de robots autónomos que hemos denominado SEVEN (*Simulation and EVolution ENvironment*) y al cual se hará referencia en todas las pruebas que se presentan, entorno en el que hemos simulado el comportamiento de los robots rodados usados en los diferentes experimentos, incluyendo sus sensores y actuadores. Por otra parte permite simular diversos tipos de entornos con múltiples elementos tales como luces, paredes, comida, veneno, etc., e implementa las diversas estrategias necesarias para que los controladores desarrollados puedan transportarse sin problemas a robots reales. SEVEN contempla todos los aspectos de paralelización y distribución multiplataforma que se requieren de un sistema de este tipo y que permite trabajar con poblaciones muy grandes y procesos de vida de los robots extensos para su evaluación.

Una tercera parte (capítulo 5) de este trabajo se dedica al estudio y definición de una serie de arquitecturas de RNAs que permiten aumentar las capacidades de procesamiento de los módulos, mediante la incorporación de retardos sinápticos a las RNAs que les proporcionarán la capacidad de procesar información temporal precisa, y

mediante el uso de sinapsis gausianas que funcionarán como mecanismos de atención y que añaden la capacidad de que el proceso evolutivo realice podas sobre la RNA para quedarse sólo con los elementos necesarios. El uso de información temporal, por su parte, no sólo es necesario para implementar determinados comportamientos, sino que permite superar situaciones de infrasensorización. En esta parte se presentarán diversos ejemplos y técnicas para mejorar los comportamientos que se pueden obtener mediante la utilización de los algoritmos planteados en secciones anteriores usando las capacidades proporcionadas por estas redes, especialmente en cuanto a su capacidad de seleccionar información relevante y subsanar problemas de fallos sensoriales mediante la consideración autónoma de información en el tiempo para la validación de percepciones.

Finalmente, en una cuarta parte de este trabajo (capítulo 6), se describe la arquitectura que soportará la construcción estructurada de sistemas complejos con el mínimo de intervención humana, maximizando además la reutilización de los controladores y permitiendo su obtención de forma incremental, utilizando los controladores obtenidos para comportamientos simples como base para dar lugar a comportamientos más complejos.

Como primer método para generar arquitecturas con múltiples controladores se presenta un mecanismo de selección. Para obtener un nuevo comportamiento a partir de módulos ya existentes en este caso sólo es necesario evolucionar un módulo selector cuya función consistirá en seleccionar qué módulo de una capa inferior se ha de ejecutar en cada momento. Se establece así una jerarquía entre los módulos, por cuanto la ejecución de los módulos ya existentes depende de la decisión del selector que se ha evolucionado. Obviamente, en la evolución de un módulo selector será necesario disponer de módulos de nivel inferior ya evolucionados y que en conjunto permitan la realización de la tarea requerida. Cuando éste no sea el caso, se plantea la posibilidad de coevolucionar con el módulo selector un módulo de bajo nivel adicional para suplir carencias en los módulos de bajo nivel disponibles. Este proceso se puede repetir tantas veces como sea necesario, dando lugar a controladores con una jerarquía en forma de árbol.

Se estudian distintos procedimientos para reutilizar en la arquitectura anterior módulos provenientes de robots físicamente diferentes mediante el uso de sensores y/o actuadores virtuales.

Dadas las limitaciones en cuanto a la generación de conjuntos de comportamientos con una variación continua entre ellos que presenta una arquitectura jerárquica basada en la selección, ampliamos los tipos de relación entre los módulos. Pasamos de una jerarquía en la que unos módulos activan/desactivan otros módulos, a otra en la que unos modulan las entradas o las salidas de otros, dando lugar a un abanico de posibilidades mucho mayor y proporcionando la capacidad a la arquitectura de obtener fácilmente comportamientos que son combinación de otros ya existentes. No

sólo eso, sino que mostramos cómo las activaciones/desactivaciones se pueden reformular como modulaciones llevando a que cualquier jerarquía siempre se pueda formular como una estructura con sólo dos niveles de módulos, acotando así el número de niveles y dando la posibilidad de que se ejecuten en paralelo todos los módulos, independientemente de las relaciones que existan entre ellos.

Con todo esto, llegamos a una metodología y una estructura que la soporta que permite obtener controladores para robots autónomos de forma automática e incremental, minimizando la intervención del diseñador que sólo necesita definir unas características básicas de los módulos a utilizar y del algoritmo evolutivo, así como establecer una serie de reglas de energía para el robot para cada comportamiento que desee implementar. La arquitectura resultante podrá obtenerse de forma incremental, reutilizando módulos previamente desarrollados para el mismo u otros robots, módulos que mediante procesos de modulación o selección podrán integrarse en arquitecturas cada vez más complejas. Con todo ello, estaremos en disposición de contemplar procesos de generación gradual e incluso paralela de arquitecturas de control de robots.

2 Objetivos

Como se ha comentado en la introducción, el objetivo de esta tesis es desarrollar una metodología y una serie de herramientas y métodos que permitan obtener automáticamente y con el mínimo posible de intervención por parte del diseñador controladores para robots autónomos que deben operar en entornos dinámicos y no estructurados. Esto implicará:

- Primeramente seleccionar la filosofía en la cual se va a enmarcar el trabajo. Existen, básicamente, dos aproximaciones a la robótica autónoma: la basada en el conocimiento y la basada en comportamientos. Deberemos ver cuál es más interesante para nuestro objetivo principal o si habrá que optar por una aproximación intermedia.
- Decidir qué estructuras se van a utilizar para implementar los controladores de los robots. Éstas deben permitir su obtención automática y deben tolerar situaciones con información incompleta o parcialmente errónea, por cuanto ambas situaciones son comunes en la percepción que el robot tiene de su entorno.
- Determinar un mecanismo que permita obtener los controladores para los robots autónomos. Este proceso deberá ser lo más automático posible y con el mínimo de intervención humana. Idealmente, ésta deberá limitarse a indicar, de alguna forma, cuál es la tarea que el robot debe realizar, pero en ningún caso cómo debe ser realizada.
- Como no existe ningún mecanismo perfecto que permita hallar la solución a cualquier problema bajo cualquier circunstancia, es de esperar que, sea cual sea el método seleccionado, habrá que estudiar qué circunstancias deben darse para que funcione correctamente o se maximicen sus probabilidades de éxito. Respecto a este mecanismo, habrá también que decidir si se ejecutará sobre robots reales en entornos reales o en un entorno simulado.
- Se tratará de que ambos, herramientas utilizadas para implementar controladores y mecanismos que logren su obtención, permitan dotar de ciertas características de alto nivel a los controladores, como pueden ser la capacidad de procesar información temporal para aquellos comportamientos que lo necesiten o de incorporar mecanismos de atención, de forma que el robot tenga la capacidad de seleccionar los datos relevantes de entre todos aquellos que percibe.
- El proceso de obtención de los controladores debe ser modular para permitir la reutilización de módulos y la construcción incremental de controladores complejos, ya que tener que obtener todos y cada uno de los controladores que implementan los comportamientos que debe realizar un robot partiendo de cero se convertiría en una tarea inabordable para comportamientos complejos.
- El punto anterior nos llevará a la necesidad de establecer mecanismos de conexión entre módulos, de forma que controladores ya existentes se puedan conectar con

otros nuevos para generar controladores más complejos. Esos mecanismos de conexión deberán poderse obtener también automáticamente y deberán ser flexibles y permitir la reutilización eficaz y eficiente de controladores ya existentes.

- Todo esto deberá ser implementado y su correcto funcionamiento validado en robots reales funcionando en entornos reales, para asegurarnos así de la validez de la metodología y herramientas desarrolladas.

3 Marco del trabajo

Tal y como se ha indicado, el objetivo en este trabajo es la realización tanto de una metodología como de una estructura que la soporte y que posibilite la obtención de comportamientos para robots autónomos de forma que éstos pueden llevar a cabo las tareas encomendadas en entornos dinámicos y no estructurados. Se pretende además que este proceso sea lo más automático posible, minimizando el papel humano en él, y que los controladores sean reutilizables y se puedan obtener incrementalmente.

A la hora de afrontar estos objetivos es necesario decidir el marco en el cual se va a encuadrar el trabajo, es decir, la filosofía con la que se pretende abordar el problema. Por ello, en este capítulo veremos las dos aproximaciones básicas que presenta la robótica autónoma (basada en el conocimiento y basada en comportamientos), sus características, ventajas e inconvenientes, para, finalmente, mostrar las razones que han llevado a seleccionar para este trabajo una aproximación basada en comportamientos, razones que se fundamentan principalmente en las diferencias de base de ambas filosofías y también en el objetivo de automatizar la obtención de los controladores.

Antes de nada, es necesario definir qué se entiende por robot autónomo y por comportamiento. La Asociación de la Industria Robótica Americana [52] define robot como "un manipulador reprogramable y multifuncional diseñado para mover material, partes, herramientas, o dispositivos especializados mediante movimientos variables programados para la realización de una variedad de tareas". No existe una definición estándar, en cambio, al respecto de qué se entiende por autonomía al referirnos a un robot, pero podemos establecer que para que un robot sea autónomo es necesario que sea capaz de reaccionar ante situaciones no consideradas en la programación de su control sin ninguna supervisión exterior. Aunque su control venga definido por un programa, el robot debe ser capaz de realizar en todo momento los movimientos necesarios para "sobrevivir" en su entorno y cumplir las tareas encomendadas, sin que su programa de control defina necesariamente, de modo explícito, todas las posibles acciones que debería realizar ante todas las posibles situaciones que se pueden presentar en su entorno. Es por esto que se dice que el robot autónomo debe ser no totalmente preprogramado.

Respecto a comportamiento, consideraremos que éste es el resultado de la interacción con el entorno de un controlador implantado en un robot determinado. El comportamiento es, pues, una etiqueta que asigna el observador. Alterar cualquiera de estos factores, el controlador, el robot o el entorno, puede dar lugar a que dicha interacción sea considerada como otro comportamiento distinto. Es, por tanto, muy importante diferenciar la estructura de control del robot del comportamiento que realiza en un entorno particular. Por tradición en el campo, y porque resulta mucho más fácil imaginarse lo que está haciendo el robot, en este trabajo se usarán en ocasiones de forma intercambiable comportamiento y controlador, refiriéndonos siempre al

controlador que mediante interacción con un entorno en particular produce ese modo de actuación y no al modo de actuación en sí.

Históricamente, la investigación en robótica comenzó su desarrollo considerando mayoritariamente entornos estructurados. Un entorno es estructurado si se pueden definir de modo inequívoco sus características, es decir, qué objetos existen, de qué tipo y en qué posición se encuentran, bien porque el entorno no cambia en el tiempo o porque los cambios posibles son predecibles y, por lo tanto, se pueden formalizar computacionalmente. La robótica tradicional implicaba ese perfecto conocimiento del entorno en el que el robot operaba y de las consecuencias de cada acción que debía llevar a cabo de forma que se pudiesen prever resultados no deseados. Para conseguir esto, los entornos en los que el robot operaba y la estructura física del mismo eran diseñados a medida y aislados de cualquier posible influencia no predicha por el diseñador. Esta aproximación al problema no es aplicable, por lo tanto, en el caso de robots que deban operar en entornos dinámicos, que cambian con el tiempo, y no totalmente conocidos en el sentido de que muchos cambios en el entorno no son predecibles, o bien, si lo fueran, el procesado de información requerido sería enorme.

A finales de los años 50 se empezaron a aplicar técnicas de inteligencia artificial a la robótica autónoma. En esta aproximación clásica a la robótica, el diseñador define, mediante entidades simbólicas, un modelo de mundo que permite realizar una planificación de los movimientos del robot para llegar a una determinada meta. Las arquitecturas que siguen esta aproximación basada en el conocimiento realizan una descomposición de los procesos que el robot debe realizar en tareas independientes, para posteriormente unirlas. La figura 1 muestra estas tareas: la interpretación de los datos provenientes de los sensores; el modelado del entorno en el que va a operar el robot, modelo sobre el que se planifica un conjunto de acciones que el robot realizará para la consecución de una determinada tarea, y estas acciones se llevan a cabo a través del módulo de ejecución que indica en los actuadores los valores adecuados a cada movimiento. El problema estriba en que el mundo es muy difícil de formalizar en cuanto aparecen cambios en el mismo a lo largo del tiempo. Además, tanto esta modelización del entorno, como la elección de los operadores que actúan sobre la capa de interpretación de los datos sensados ha sido realizada por el diseñador, lo cual no tiene por qué ser lo óptimo.

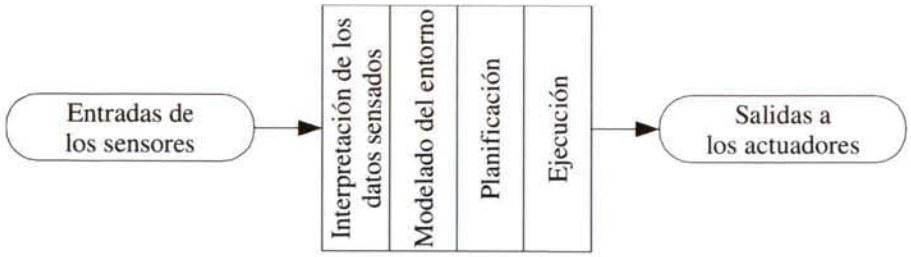


Figura 1: Esquema de la aproximación de la robótica basada en el conocimiento.

Como primer ejemplo relevante de esta aproximación tradicional debemos citar el robot *Shakey*, desarrollado en el Instituto de Investigación de Stanford en 1969. Este robot se movía en un entorno de oficina con objetos de formas y colores previamente especificados para su reconocimiento por el sistema visual. Un ordenador interno se encargaba de la ejecución de las secuencias de control sobre los motores, en tanto que un ordenador externo se encargaba del resto de etapas del modelo, el procesado de las imágenes de la cámara y la planificación de las acciones. Esta planificación se basaba en un sistema de resolución de problemas denominado *STRIPS (STanford Research Institute Problem Solver)* sobre un modelo de entorno simbólico realizado mediante lógica de predicados de primer orden. Las tareas que realizaba eran tales como mover un objeto de una posición a otra, pero el coste computacional del sistema era tal que el robot alcanzaba únicamente una velocidad aproximada de 2 metros por hora [32].

Otro ejemplo relevante en esta perspectiva de robótica clásica es el robot *Cart*, desarrollado también en Stanford en 1977, en este caso por el investigador Hans Moravec. El robot utilizaba el sistema visual para realizar una tarea de navegación, añadiendo obstáculos a su modelo de entorno conforme los encontraba. El *Cart* utilizaba en este caso un algoritmo de búsqueda en grafo para determinar el camino más corto, minimizando al mismo tiempo ángulos necesarios de giro y los requisitos energéticos de los diferentes movimientos. Al igual que *Shakey*, el coste computacional, determinado principalmente por su fase de planificación, era muy elevado y el robot apenas se desplazaba un metro cada diez o quince minutos. Un recorrido completo de su entorno requería alrededor de cinco horas. A principios de los 80, Moravec abandona Stanford y continúa su trabajo en la Universidad de Carnegie-Mellon en la que desarrolla un nuevo robot al que denominó *CMU Rover*. Éste presentaba una forma cilíndrica, con sensores de infrarrojos y ultrasonidos y una cámara en su parte posterior, configuración que poseen muchos de los robots utilizados actualmente en investigación. La arquitectura cognitiva seguía siendo la clásica, aunque ahora las distintas partes del sistema mantenían una comunicación mediante mensajes a través de una estructura de pizarra.

El inconveniente de base presente en esta aproximación es que nuestro conocimiento no tiene por qué ser el que necesite el robot y lo que nosotros creemos que tiene que hacer el robot para solventar el problema no tiene por qué ser la solución más adecuada. Un ser humano no es un robot, existen diferencias perceptuales, actuadoras y cognitivas que deben ser tenidas en cuenta por los ingenieros a la hora de implementar un sistema basado en el conocimiento en un robot. Cuánto más complejo es el comportamiento a resolver o la plataforma que ha de resolverlo más patente se hace este problema.

Se puede considerar a William Grey Walter como un precursor en las ideas que luego desembocarían en la robótica basada en comportamientos. Utilizando relés y tubos de vacío como elementos de conmutación, engranajes de contadores de gas y motores eléctricos como impulsores, empezó a construir criaturas artificiales durante la

década de los 40, tomando como premisas que el robot debería interaccionar directamente con el entorno y su comportamiento debería surgir de dicha interacción. Su trabajo dio lugar en 1948 a lo que denominó *Machina Speculatrix* [105][106], dos robots del tamaño de una caja de zapatos con forma de tortuga, cada uno con dos motores eléctricos (impulsión y conducción), tres ruedas (la delantera tenía la tracción y dirigía el robot), un sensor de contacto (dispuesto en forma de faldón alrededor del robot), y un sensor de luz orientado en la dirección de avance del robot. Simplemente con esto, los robots realizaban comportamientos como, por ejemplo, dirigirse hacia una fuente de luz, salir de colisiones (mediante movimientos aleatorios hasta que el robot no percibía colisión) y exploración (también aleatoria). Además, dichos comportamientos se veían afectados por el nivel de carga de su batería. Esta filosofía se diferenciaba profundamente tanto de sus contemporáneos como de la mayoría de los investigadores que le sucedieron en las tres décadas siguientes, que trataban de construir modelos de mundo y, sobre estos, modelos de razonamiento para los robots.

Tres décadas más tarde, en 1984, el psicólogo alemán Valentino Braitenberg [11] diseñó una serie de vehículos que no llegó a implementar en plataformas reales, en los que mediante conexiones excitadoras e inhibidoras y permitiendo enlaces directos entre los sensores y los motores, al igual que las Tortugas de Walter, se demostraba la capacidad de obtener una serie de comportamientos complejos. Así, por ejemplo, uno de ellos presentaba un único sensor y controlaba la velocidad de un único motor. Si el sensor era de temperatura, se movía más rápido en regiones calientes que en regiones frías. Otro de los robots planteados presentaba dos sensores de luz y conexiones excitadoras a dos motores siguiendo dos esquemas diferentes, que producían un movimiento de separación y acercamiento a la luz respectivamente. En otro tercer ejemplo, similar al anterior, las conexiones inhibían la velocidad de los motores cuanto mayor era el nivel de luz sensado. Se adivinaban los comportamientos de movimiento más rápido en presencia de luz débil y deceleración en presencia de luz intensa, con las mismas características de acercamiento y fobia a la luz anteriores. Si además la respuesta de los motores no era proporcional al valor de los sensores se consiguían patrones más complejos, con movimientos oscilatorios entre dos fuentes luminosas y diferentes movimientos circulares en torno a una luz. Estos ejemplos mostraban que, para vehículos simples, es fácil adivinar lo que van a hacer ante un esquema de conexión dado. A medida que se introducen respuestas no lineales y esquemas de conexionado complejos, predecir los comportamientos resultantes se vuelve una tarea muy difícil, incluso aunque sean comportamientos puramente reactivos, como en estos casos.

La robótica basada en comportamientos se puede considerar que nace a partir de la segunda parte de la década de los 80 como una serie diversa de métodos (sistemas reactivos [62][98], autómatas situados [95], agentes situados [3], arquitecturas subsumidas [13][12][15], redes de acción [83], etc.) que tratan de solventar los problemas ya comentados de la robótica tradicional, teniendo en común la idea

fundamental de que la funcionalidad del robot debe ser una propiedad que emerja de su interacción con el entorno en el que se desenvuelve, que será dinámico. Posteriormente se ha ido precisando la definición de robótica basada en comportamientos [7][15][70][73] y varios autores le han atribuido algunas características inspirándose en organismos presentes en la naturaleza (especialmente insectos), haciendo énfasis en los propios comportamientos y en la rapidez de reacción, y no en el conocimiento que subyace en esos comportamientos ni en una hipotética planificación necesaria para llevarlos a cabo: las representaciones explícitas no son siempre necesarias para la inteligencia y debe haber una interacción directa entre el robot y su entorno, construyéndose el modelo cognitivo del robot a partir de la interacción entre ambos. El sistema completo forma parte o está situado en el entorno, sin necesidad de modelado, ya que "el mundo es el mejor modelo de sí mismo" [14]. Así, de acuerdo a la definición de Arkin, los comportamientos pueden verse como "parejas estímulo-respuesta, moduladas por la atención y determinadas por la intención" [7], donde la atención prioriza las tareas y proporciona una organización en el uso de los recursos sensoriales, mientras que la intención determina los comportamientos a activar dependiendo de los objetivos o tareas que el robot deba alcanzar o realizar. Son, en esta aproximación, esos comportamientos los bloques constructivos del sistema cognitivo, claramente modular, y en ellos se evitarán representaciones explícitas del conocimiento y se tomará el comportamiento animal como modelo a seguir. En la figura 2 se puede ver el esquema de esta aproximación a la robótica, donde todos los módulos, independientemente de su complejidad, se conectan directamente con los sensores y con los actuadores.

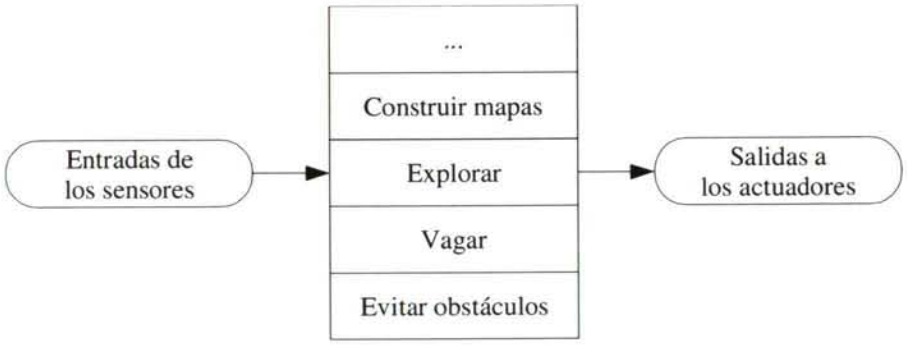


Figura 2: Esquema de la aproximación a la robótica basada en comportamientos.

Describiremos a continuación, de una forma más detallada, las diferencias entre los dos tipos de aproximaciones, en su estado más puro, tal y como apunta Maes en [71], teniendo en cuenta que, en la práctica, son habituales los sistemas más o menos híbridos:

- Modelado del sistema.

La robótica basada en el conocimiento modela un sistema utilizando el conocimiento que se tiene del dominio del problema, esto es, modela el propio conocimiento. El sistema mantiene una representación interna simbólica de las acciones, las metas y los eventos, y a partir de esta representación realiza un proceso de razonamiento para determinar qué acciones se deben realizar para alcanzar las metas teniendo en cuenta los eventos que se van produciendo (planificación). Esto tiene un doble inconveniente. Por una parte, frecuentemente, el conocimiento que tenemos del dominio es incompleto, impreciso o, simplemente, el dominio cambia con el tiempo. Debido a esto, es una tarea ardua construir un sistema robusto. Pero hay otro problema más profundo, ya que el robot posee unas cualidades distintas a las del ser humano, con lo que el conocimiento representado puede no ser el más adecuado para realizar su tarea, o la representación empleada no ser la más idónea. Por el contrario, la ventaja de la robótica basada en el conocimiento es que permite responder a preguntas acerca del dominio y determinar en cada momento por qué el sistema se comporta como se comporta.

La robótica basada en comportamientos no modela el conocimiento sino los comportamientos. La ventaja es que reproducir un comportamiento suele ser mucho más fácil que reproducir el proceso lógico que da lugar a ese comportamiento, lo que simplifica la construcción de sistemas robustos. Además, si se usan mecanismos como el aprendizaje o la evolución, el diseñador sólo determina, de alguna manera, cuán bueno es el controlador que se tiene en cada momento y es el propio proceso de aprendizaje o evolución el que va ajustando este controlador hasta obtener el comportamiento deseado. El diseñador no necesita describir su funcionamiento. Como inconveniente, la robótica basada en comportamientos presenta el hecho de que no modelar el conocimiento hace que resulte mucho más difícil averiguar por qué un sistema se comporta de la forma en la que lo hace.

- Tipo de control.

En la robótica basada en el conocimiento se sigue una descomposición en módulos funcionales, como denota la figura 1, tales como la percepción, ejecución y planificación. Esa descomposición es completamente secuencial.

En la robótica basada en comportamientos, aunque se puede dar el control centralizado [20], existe la posibilidad de establecer una estrategia de control distribuido, donde varios comportamientos compiten entre ellos para decidir cuál controla los actuadores en cada momento [85][84]. Incluso pueden ejecutarse varias acciones simultáneamente, bien sobre actuadores distintos o bien sobre los mismos actuadores.

- Relación con el entorno.

En la robótica basada en el conocimiento, el sistema no ve el entorno tal cual es, sino que lo ve filtrado a través de la perspectiva del diseñador, que define una taxonomía sobre los datos de los sensores y determina qué datos son importantes y cuáles no.

Esto no sólo es difícil en entornos dinámicos, sino que debido a la diferencia entre los aparatos sensoriales de robot y ser humano, la taxonomía puede no ser la más adecuada para el robot. Una decisión equivocada por parte del diseñador en la valoración de la importancia de los datos provocará un funcionamiento inadecuado por parte del robot. Este problema se ha denominado "symbol grounding problem" [17][40].

En la robótica basada en comportamientos, el sistema está directamente conectado a sensores y actuadores, con lo que no existe el riesgo de una mala apreciación de la realidad del dominio del problema por parte del diseñador, en cuanto a datos de sensores y acciones a llevar a cabo se refiere, al no estar presente en esta fase del diseño. Además, se elimina una capa de procesamiento, con lo que se agiliza el cálculo de la siguiente acción a realizar.

- Tiempo de reacción.

Como hemos visto anteriormente, en la robótica basada en el conocimiento existe una etapa en la que se decide qué acción llevar a cabo una vez se ha construido el modelo del entorno. Este proceso no es interrumpible: mientras el sistema analiza los datos y toma una decisión sobre lo que debe hacer, cualquier cambio en el entorno es ignorado o bien su procesado es pospuesto.

En la robótica basada en comportamientos, al poder ejecutarse todos los comportamientos simultáneamente en competencia mutua, la aparición de un nuevo dato importante puede activar instantáneamente un comportamiento que interrumpa a cualquier otro que se estuviese ejecutando.

Incluso entre los sistemas basados en el conocimiento, este problema es ampliamente reconocido y es muy habitual la utilización de técnicas basadas en comportamientos para aquellas tareas de más bajo nivel, que precisan de un tiempo de reacción reducido y que pueden entrar en acción asíncronamente, al margen de los módulos deliberativos basados en el conocimiento. Así, se aprovechan las facilidades para la planificación que da la robótica basada en el conocimiento y la rapidez de actuación de la robótica basada en comportamientos. Un ejemplo de esta técnica híbrida se encuentra en el trabajo de Alwan y col. [5].

- Recursos necesarios.

Un sistema basado en el conocimiento suele necesitar de unos recursos de memoria y capacidad de proceso para obtener respuestas en el tiempo necesario muy superiores a los de un sistema basado en comportamientos. Esto, en cuanto se quieren implementar varios controladores, puede incapacitar al sistema basado en el conocimiento para muchos robots de pequeño tamaño, con una CPU simple y poca memoria. Es, también, especialmente importante, en el caso de utilizar un sistema basado en el conocimiento para implementar los controladores, que la CPU sea potente para intentar minimizar el problema comentado en el punto anterior.

Los sistemas basados en comportamientos no sólo suelen consumir menos recursos, sino que son muchas veces susceptibles de implementaciones eficientes por hardware.

- Adaptación al entorno.

Muchos de los sistemas basados en el conocimiento, una vez diseñados, son de difícil modificación en tiempo de ejecución. En cambio, en los sistemas basados en comportamientos y, en palabras de la propia Maes [71], existe un fuerte énfasis en mecanismos de adaptación, lo cual significa que el sistema mejora sus estructuras (y, por ende, su comportamiento) a lo largo del tiempo mediante su propia experiencia en el entorno. Otro aspecto en el que también se hace énfasis en la robótica basada en comportamientos es en la aproximación incremental en la que el diseñador evoluciona gradualmente un sistema más sofisticado añadiendo nuevas estructuras a un sistema ya en funcionamiento. La estrategia es así "de abajo hacia arriba" y se habla de comportamiento emergente como el resultado de la interacción de los diversos componentes con el entorno.

Como se puede ver, cada enfoque tiene sus ventajas y sus inconvenientes. Parece un hecho aceptado que para los comportamientos de más bajo nivel, aquellos considerados como meramente reactivos y donde la rapidez de acción es crítica, la robótica basada en comportamientos es mucho más adecuada. Sin embargo, la complejidad de los comportamientos obtenidos en la robótica basada en comportamientos está por detrás de la obtenida en la robótica basada en el conocimiento por las dificultades que presenta la primera para desarrollar, a partir de meras conexiones entre sensores y actuadores, comportamientos deliberativos. Para superar este problema de escalabilidad en la robótica basada en comportamientos es necesario determinar cómo dividir comportamientos complejos en módulos cuya obtención individual sea factible, mecanismos concretos de combinación para estos módulos, métodos para la obtención automática tanto de los módulos como de las conexiones entre ellos y una metodología que permita que este proceso sea realizado de forma incremental.

En este trabajo se ha considerado que los problemas de la robótica basada en el conocimiento forman parte de su naturaleza y serán difícilmente superables. Por contra, la idea detrás de la robótica basada en comportamientos resulta muy atractiva, ya que permite un camino para trabajar de forma mucho más descentralizada y eficiente sin que por ello presente limitaciones intrínsecas. Por estas razones, en este trabajo se ha optado por una aproximación basada en comportamientos, profundizando en ella y tratando de sentar las bases para que la obtención de comportamientos cada vez más complejos sea factible y fácilmente escalable. Para alcanzar este objetivo se ha considerado fundamental que los comportamientos surjan automáticamente con la menor intervención posible por parte del diseñador y que se puedan obtener incrementalmente, utilizando comportamientos básicos para generar otros más complejos. La robótica basada en comportamientos, si bien no está directamente ligada a esta obtención

automática minimizando el papel del diseñador, presenta facilidades para su realización precisamente debido a sus características de conexión directa entre sensores y actuadores sin necesidad de fases de interpretación, modelado, planificación y ejecución secuencial.

En los siguientes capítulos se abordarán los problemas de cómo automatizar el diseño de los controladores, cómo implementar éstos y cómo organizarlos para dar lugar a controladores que implementen comportamientos complejos. En cada uno de ellos se comentará el estado del arte y las aportaciones de otros investigadores en el campo como paso previo a la explicación de cómo se ha afrontado en este trabajo.

4 Automatización del diseño

Una vez seleccionada la robótica basada en comportamientos como la aproximación fundamental a seguir para la consecución del objetivo deseado de obtener automática e incrementalmente comportamientos para robots autónomos que han de operar en entornos dinámicos, es necesario establecer una metodología que permita alcanzar dicho objetivo, así como una arquitectura que dé soporte a dicha metodología y sobre la que implementar los comportamientos.

En este capítulo veremos los elementos que conforman la metodología creada: qué herramientas se han seleccionado, cómo se deben utilizar, qué aportaciones se han realizado así como un estudio de los problemas y consideraciones que se deben de tener en cuenta en todo el proceso para obtener unos resultados satisfactorios. Queda para el apartado 6.2 la descripción de la arquitectura desarrollada para dar soporte a la metodología.

Las primeras decisiones que deben tomarse son, obviamente, cómo y con qué se van a implementar los comportamientos. En una primera parte de este capítulo veremos las distintas alternativas que existen, tanto para implementar los comportamientos como para obtenerlos automáticamente para, posteriormente, en la segunda parte, estudiar en profundidad la solución adoptada.

Las posibilidades para implementar los controladores son varias: autómatas de estados finitos, conjuntos de reglas, programas en algún tipo de lenguaje, redes de neuronas artificiales (RNAs), etc. Pero, como veremos, éstas últimas presentan una serie de características que las hacen idóneas para los requisitos de este trabajo: su inspiración biológica, su capacidad de procesamiento distribuido que las hace tolerantes al ruido y la facilidad para dotarlas de capacidad de procesamiento de información temporal y de mecanismos de atención. Aunque tanto el resto de la metodología como la arquitectura es aplicable a otras estructuras de control, las RNAs son el mecanismo preferido por estas razones, tal y como veremos.

Como mecanismos de obtención automática de las estructuras de control consideraremos el aprendizaje y los algoritmos evolutivos. Dadas las dificultades en ocasiones presentes para utilizar métodos de aprendizaje, y las ventajas de los algoritmos evolutivos en cuanto a realizar una búsqueda partiendo desde múltiples puntos en el espacio de soluciones en lugar de uno sólo y de permitir una total libertad en cuanto a las estructuras de control utilizadas, se optará por utilizar éste último. Algoritmos evolutivos, sin embargo, existen muchos. Veremos las características de aquellos más comunes y que han sido usados previamente por otros investigadores en problemas de robótica autónoma, como son los algoritmos genéticos, las estrategias evolutivas y la programación evolutiva. Introduciremos también otro tipo de algoritmo evolutivo más reciente, los algoritmos macroevolutivos, que no habían sido hasta la fecha utilizados en robótica y que poseen unas características que los hacen muy adecuados para nuestros problemas, como podremos comprobar en este mismo capítulo.

Además de seleccionar un método adecuado para obtener los controladores es necesario determinar dónde se van a obtener. Pudiera parecer obvio que el proceso de obtención, ya sea mediante aprendizaje o evolución, se debe llevar a cabo en el robot real, pero hay varias razones para que esto no se pueda realizar siempre. El coste en tiempo necesario puede ser muy elevado si el comportamiento es difícil de obtener. Además, el robot puede ser dañado al principio del proceso, cuando su comportamiento es errático. Esto nos llevará a considerar la simulación como una herramienta más idónea para llevar a cabo el proceso de obtención de los controladores ya que puede ser realizada siempre, y permite además acelerar la evolución al poder transcurrir en ella los eventos a una velocidad superior a la que se produce en el entorno real. Sin embargo, habrá que estudiar bajo qué condiciones la simulación permite obtener controladores que funcionan correctamente en el entorno real.

La elección de las herramientas a utilizar sólo es el primer paso para establecer la metodología buscada. Es necesario determinar cómo se deben utilizar para que la obtención de los resultados deseados sea posible. Como veremos, el problema aquí planteado tiene sus propias características que lo separan de otros problemas típicos a los que se aplican estas mismas herramientas y, por tanto, todas y cada una de las elecciones que determinen el funcionamiento de éstas debe ser analizada con cuidado. Esto será lo que se tratará en la segunda parte de este capítulo.

Al finalizar el capítulo tendremos estudiados y analizados todos los elementos necesarios que definen la metodología (con qué obtener los controladores, cómo hacerlo, dónde) para saber cómo aplicarlos al objetivo de obtener controladores para robots autónomos de forma automática minimizando el papel del diseñador humano, y estaremos listos para estudiar la arquitectura que soportará esta metodología y que permitirá que el proceso pueda ser incremental y los controladores puedan ser reutilizados (capítulo 6). Previamente a eso, veremos cómo aumentar las capacidades de las RNAs que componen los controladores, proporcionándoles características que les permitirán procesar información temporal, seleccionar los datos relevantes de entre todos los que entran en la RNA y de podar por sí mismos neuronas y conexiones sinápticas (capítulo 5).

4.1 Estructuras de control

En la robótica basada en comportamientos se han utilizado diferentes alternativas a la hora de plasmar los controladores, como programación directa en algún tipo de lenguaje, automátas de estados finitos, conjuntos de reglas, redes de neuronas artificiales, etc. En este trabajo se han utilizado mayoritariamente redes de neuronas artificiales como estructuras de control, al igual que la mayoría de autores, por ser estructuras con propiedades adecuadas a la problemática de la robótica. Haremos en este apartado unos comentarios sobre las diferentes alternativas así como una justificación de nuestra elección.

Las "arquitecturas subsumidas" de Brooks [13], quizás el primer ejemplo de paradigma para la construcción de controladores dentro de la nueva perspectiva basada en comportamientos, son redes de autómatas de estados finitos extendidos (AFSM) con un funcionamiento asíncrono, es decir, se ejecuta cada uno independientemente de los otros, donde cada AFSM representa un comportamiento. Un AFSM no es más que un autómata de estados finitos convencional al que se le ha añadido un conjunto de registros (donde pueden dejar los datos de los sensores, por ejemplo) y de temporizadores, descrito en un lenguaje de alto nivel. El envío de un mensaje de un AFSM a otro o la finalización de un temporizador son dos formas adicionales de activar un cambio de estado en un AFSM. Los AFSMs pueden estar directamente conectados a sensores o actuadores y se organizan en capas de tal forma que cada capa contiene los comportamientos de una determinada prioridad. Existe un tipo especial de conexión, inhibidora, que se utiliza para que un comportamiento bloquee la salida de otro comportamiento con menor prioridad.

Otra de las primeras contribuciones a la robótica basada en comportamientos fueron los "autómatas situados" [95], en los cuales el diseñador especifica los planes a seguir por el robot mediante un lenguaje de alto nivel diseñado para el propósito. Una vez elaborado el plan, un compilador genera primero un programa formado por pares condición-acción y luego, a partir de dichos pares, un autómata que aplica el plan diseñado y en el que los sensores son las entradas y los actuadores las salidas. Aunque este método rompe con uno de los principios de la robótica basada en comportamientos al especificar el plan a seguir por parte del diseñador, fue uno de los primeros intentos por superar los problemas de la robótica tradicional. Por contra, sigue los postulados de dicha aproximación al no existir interpretación de los datos de los sensores, sino que dichos datos son utilizados directamente por el autómata generado. Además, no existe un procesado simbólico en tiempo real, sino que el plan se convierte en tiempo de compilación en un autómata que une directamente los sensores con los actuadores.

Se ha hecho uso también de sistemas de clasificación. Por ejemplo, Doringo y Colombetti [26], los utilizan en diferentes comportamientos sencillos en combinación con aprendizaje por refuerzo -a través del "algoritmo de reparto de crédito" en IA (Holland [48])- y con una búsqueda evolutiva que obtiene automáticamente las

categorizaciones simbólicas adecuadas a considerar en las reglas de producción, salvando así, en buena medida, el *symbol grouding problem* [17][40], de definición simbólica por el diseñador de la realidad en la que está inmerso el robot.

Diversos autores han utilizado también lógica borrosa en sus controladores. Por ejemplo, Cooper [21] y Hoffmann y Pfister [46] han utilizado la evolución simulada para obtener controladores borrosos, al igual que Matellán y col. [74] para comportamientos sencillos con el robot Khepera, obteniendo las reglas borrosas adecuadas con un algoritmo genético, aunque partiendo de una definición previa de los conjuntos borrosos del rango de los sensores de infrarrojos y de los conjuntos borrosos de las velocidades en ambas ruedas del Khepera. Saffioti [96] argumenta que la propiedad clave por la cual los sistemas basados en lógica borrosa pueden conseguir controladores robustos es el mecanismo de interpolación inherente en estos sistemas que engloba la idea de que entradas similares producen acciones similares. En el lado contrario de la balanza el autor resalta que los sistemas basados en lógica borrosa se adaptan bien a tareas en las cuales los operadores humanos pueden especificar el proceso e introducir conocimiento heurístico, pero son bastante menos útiles cuando no se utiliza preprocesamiento, como es el caso general de la aproximación de "abajo-arriba" propia de la robótica basada en comportamientos.

En los dos primeros ejemplos anteriores, precursores de la robótica basada en comportamientos, aunque se eliminan algunos problemas de la robótica basada en el conocimiento, el diseñador sigue teniendo que, o bien trazar un plan de acción, la forma en la que el robot se comportará para abordar un determinado objetivo, o bien, en las arquitecturas subsumidas, decidir qué comportamientos tienen que tomar el control en cada situación posible, con la definición de las inhibiciones, supresiones o temporizadores en cada nivel de comportamiento, cuya complejidad crece a medida que se incrementan tales niveles. A principios de los 90, diferentes grupos tornaron pronto sus miradas hacia métodos de diseño automático y, en concreto, al uso de la evolución artificial, comenzando a consolidar la disciplina de robótica evolutiva. Es además necesario que las estructuras de control permitan también un nivel de adaptación en tiempo de actuación del robot, o en "tiempo de vida" tomando la terminología al uso en el campo de Vida Artificial, que posibilite la adaptación a los cambios rápidos en el entorno, además de la robustez que pueda conseguir la evolución artificial dependiendo de la variedad de situaciones con que se haya encontrado un individuo-controlador en el tiempo de evolución.

Como hemos dicho, en el campo destaca el uso de las redes de neuronas artificiales como estructuras de control de los comportamientos en robots autónomos [43][18][87][34][10][67][99][63], entre otros motivos porque mediante entrenamiento o evolución se pueden obtener automáticamente RNAs que se traduzcan en el comportamiento deseado.

Entre las características de las RNAs como controladores de comportamientos podemos citar las siguientes:

- Funcionan como aproximadores no lineales universales. Utilizando el suficiente número de elementos computacionales permiten aproximar cualquier función y, por tanto, implementar cualquier controlador.
- Inmunidad al ruido. Las RNAs, gracias a ser estructuras altamente distribuidas, toleran bien el ruido, presente de forma notable tanto en los sensores como en los actuadores de los robots. Esta es una gran ventaja en comparación con otras estructuras de procesado simbólico como sistemas de clasificación.
- Existen métodos de aprendizaje establecidos en topologías concretas de RNAs, que pueden permitir la mejora del comportamiento durante la vida del robot. Por ejemplo, la combinación de los métodos más adecuados en robótica, basados en la idea de refuerzo, con el algoritmo ampliamente utilizado de retropropación del error [1][75]. La combinación de evolución y aprendizaje puede considerar a este último como un refinamiento o una búsqueda local en la vecindad de cada solución considerada en la evolución, y que permite la posibilidad de considerar "estrategias de Lamarck" (en las que las soluciones refinadas en tiempo de vida revierten en el genotipo), o lo contrario, en las que la búsqueda local permite un suavizado del espacio de búsqueda. Esta última combinación se ha utilizado en el campo de Vida Artificial para demostrar el conocido como *Efecto Baldwin* [45].
- Igualmente, en procesos evolutivos y si los cromosomas representan directamente los parámetros de las RNAs, los genes son elementos mucho más sencillos de manipular que si usamos autómatas o algún tipo de lenguaje. Esto facilita la elección de las operaciones a realizar sobre los cromosomas, disminuyendo así el riesgo de decisiones de diseño poco afortunadas: "las primitivas consideradas por el proceso de evolución deben ser del más bajo nivel posible de forma que se eviten las selecciones no adecuadas por parte del diseñador humano" (Nolfi y col. [88]).

Resumiendo, las RNAs se comportan como aproximadores no lineales universales fácilmente manipulables que posibilitan la obtención automática de controladores y que, además, presentan otras características interesantes, como la inmunidad al ruido. Todas estas razones han motivado la elección de las RNAs como la herramienta a utilizar para implementar los comportamientos en forma de controladores. Como veremos en el capítulo 5, la incorporación de características adicionales a las RNAs como puedan ser los retardos sinápticos o las sinapsis gausianas, aportarán todavía más ventajas, como la posibilidad de considerar instantes temporales concretos y de seleccionar automáticamente los datos de entrada relevantes.

4.2 Obtención automática de los controladores

Una vez determinado cómo plasmar los comportamientos en forma de controladores es necesario decidir cómo obtener estos. En este apartado veremos cómo realizarlo de forma automática, por cuanto éste es uno de los requisitos que nos hemos planteado al principio del trabajo. Nos encontraremos básicamente con dos métodos: el aprendizaje y los algoritmos evolutivos, y explicaremos la razón de seleccionar éstos últimos como el método a utilizar.

Si los controladores son realizados por el diseñador surgen dos problemas. El primero es la dificultad cuando el comportamiento a implementar es complejo y el segundo es el hecho de que los controladores elaborados a mano no tienen por qué ser los mejores, debido a que el diseñador no percibe el entorno como el robot, ni posee su mismas capacidades a la hora de interaccionar con dicho entorno, ni tampoco posee necesariamente toda la información necesaria para decidir cuál es la mejor implementación para el controlador.

Por tanto, la utilización de un método que proporcione automáticamente el controlador es altamente deseable. En el caso de haber seleccionado como herramienta para implementar los controladores las redes de neuronas artificiales, como así ha sido en este trabajo, es obligatorio e inherente a la herramienta, puesto que las RNAs fueron concebidas para ser obtenidas mediante entrenamiento, y no para establecer los pesos manualmente. Existe una herramienta alternativa al entrenamiento como método para obtener automáticamente las RNAs: los algoritmos evolutivos. Estos presentan una serie de ventajas respecto al aprendizaje: se pueden aplicar en cualquier caso, independientemente de la topología y características de la RNA, realizan la búsqueda de la solución desde múltiples puntos simultáneamente y son muy fácilmente paralelizables.

En cualquier caso, obtener el controlador automáticamente no quiere decir que el diseñador esté totalmente libre de tomar decisiones de diseño. Tanto los algoritmos de entrenamiento como los de evolución están controlados por una serie de parámetros que determinarán su funcionamiento y esos parámetros, en principio, tendrán que ser ajustados por el diseñador.

Aprendizaje

Para obtener los pesos u otros parámetros de una red de neuronas artificiales se han definido en la bibliografía una serie de procesos de aprendizaje consistentes fundamentalmente en la adaptación de los valores de dichos pesos por medio de algún algoritmo de búsqueda de extremos sobre el error respecto a las salidas deseadas. Los más establecidos conllevan procesos de descenso de gradiente, tal como el tradicional de retropropagación del error (*backpropagation*). Los problemas de este tipo de algoritmos son, fundamentalmente:

- Son específicos para una arquitectura o topología de red.
- Por su naturaleza de optimización por movimiento de un solo punto en el espacio de búsqueda tienden a quedarse en máximos locales cuando el espacio es complejo.
- Para entrenar, normalmente, es necesario conocer conjuntos de entrada-salida.

Evidentemente, dado que el objetivo de nuestro trabajo es obtener arquitecturas complejas de comportamientos, pretendemos no ceñirnos necesariamente a una topología determinada. De hecho, en el capítulo 5 se verán algunas topologías y estructuras de red que proporcionan a las RNAs capacidades tales como manejar información temporal o realizar podado automático de la RNA, que no admiten, en principio, un mismo algoritmo de entrenamiento. Por lo tanto, habremos de buscar otra solución que permita considerar la adaptación de los parámetros de cualquier tipo de topología y que, además, permita hacerlo a partir de evaluaciones indirectas, cualitativas, sobre comportamientos de larga duración y no utilizando pares entrada-salida para cada paso. A continuación veremos cómo los algoritmos evolutivos cumplen con dichos requisitos.

Algoritmos evolutivos

Norbert Wiener en 1948 en su libro "*Cybernetics, or control and communication in the animal and the machine*" [108], ya planteaba de forma teórica la posibilidad de utilizar evolución para obtener los sistemas de control para robots. Posteriormente, en los últimos años de la década de los 80 y primeros años de la década de los 90, algunos investigadores, como Irman Harvey [43], Phil Husbands [49] y Dave Cliff [19] en la Universidad de Sussex en Brighton, y Randall Beer y John Gallagher [10] de la *Case Western Reserve University* en Cleveland, propusieron la evolución artificial como un medio para automatizar el proceso de diseño de este tipo de sistemas, dando lugar a lo que se empezó a conocer como "robótica evolutiva". A partir de ese momento, muchos autores contemplaron esta posibilidad y desarrollaron diferentes mecanismos de evolución y estrategias para obtener controladores de robots con el objetivo de su operación autónoma en entornos estructurados y no estructurados. En palabras de Dave Cliff y col. [18], utilizando esta nueva aproximación se cambia el punto de vista y, en vez de decidir cómo van a ser generados los comportamientos adaptativos, se pasa a decidir ahora qué comportamientos van a ser generados.

Estos algoritmos hacen uso de una analogía entre el proceso evolutivo en la naturaleza y la búsqueda de una solución en un espacio determinado. Su funcionamiento general se puede apreciar en la figura 3.

Básicamente, tenemos una población inicial de individuos. Cada individuo es una posible solución al problema (un punto en el espacio de soluciones) y viene caracterizado por uno o varios cromosomas. Esta población evoluciona con el tiempo: los individuos se combinan y sufren mutaciones en los genes de sus cromosomas, de tal

forma que las posibilidades que tienen de pasar, ellos o sus descendientes, a la siguiente generación son proporcionales a lo cerca que estén de la solución del problema. La forma en la que los individuos se combinan, cómo tienen lugar las mutaciones, la frecuencia de ambas formas de reproducción y sobre qué individuos se producen dan lugar a los diversos tipos de algoritmos evolutivos.

Sean:
- I *espacio de búsqueda*
- $F: I \rightarrow \mathbb{R}$ *la función de calidad*
- μ *el tamaño de la población de los padres*
- λ *el tamaño de la población de los descendientes*
- $P(t) = (a_1(t), ..., a_\mu(t)) \in I^\mu$ *la población en la generación* t
- $r: I^\mu \rightarrow I^\kappa$ *operador de recombinación*
- Θ_r *parámetros que determinan el funcionamiento de* r
- $m: I^\kappa \rightarrow I^\lambda$ *operador de mutación*
- Θ_m *parámetros que determinan el funcionamiento de* m
- $s: I^\lambda \rightarrow I^\mu$ *operador de selección*
- Θ_s *parámetros que determinan el funcionamiento de* s
- ι *criterio de parada*
- Θ_ι *parámetros del criterio de parada*

Entonces:
- $t \leftarrow 0$
- $P(t) \leftarrow inicializar(\mu)$
- $F(t) \leftarrow evaluar(P(t), \mu)$
- *mientras* $(\iota(P(t), \Theta_\iota) \neq cierto)$ *hacer*
  - $P'(t) \leftarrow recombinar(P(t), \Theta_r)$
  - $P''(t) \leftarrow mutar(P'(t), \Theta_m)$
  - $F(t) \leftarrow evaluar(P''(t), \lambda)$
  - $P(t+1) \leftarrow seleccionar(P''(t), F(t), \mu, \Theta_s)$
  - $t \leftarrow t+1$

Figura 3: Funcionamiento general de un algoritmo evolutivo.

En el caso que nos ocupa, los individuos son los controladores candidatos para un determinado comportamiento del robot. Los genes del cromosoma pueden ser los parámetros que describen una RNA, sentencias en un determinado lenguaje, etc., según el método que sigamos en la descripción de un controlador. Para determinar lo cerca que está un individuo de la solución óptima se observa el comportamiento del robot cuyo controlador es el que se corresponde con los genes de dicho individuo, y se le asigna una calidad en función de lo bien que ha realizado ese robot la tarea que el diseñador pretende que complete. Esta calidad determina las posibilidades de

reproducirse del individuo, y así se van obteniendo nuevas poblaciones de individuos cuyo comportamiento se parece cada vez más al comportamiento deseado. Si usamos RNAs para codificar los controladores, el uso de algoritmos evolutivos se ha extendido ampliamente [80][67][79][66][88][97][75][94] porque permite obtenerlas automáticamente con una mayor libertad y un menor riesgo de caer en máximos locales que mediante entrenamiento.

Los algoritmos evolutivos realizan una búsqueda partiendo de muchos puntos dispersos por el espacio de soluciones y la calidad de un individuo sólo se establece al final de su vida, cuando es fácil determinar si se comporta correctamente o no. Estos algoritmos carecen, por tanto, de los problemas del aprendizaje tradicional. Sin embargo, el aprendizaje sigue siendo útil, puesto que la búsqueda en el espacio se realiza mediante pasos más pequeños. Por ello, cuando puede ser utilizado, el aprendizaje se usa para mejorar una solución inicial obtenida mediante un proceso evolutivo. El aprendizaje también se puede usar durante el proceso evolutivo modificando su dinámica mediante el efecto Baldwin [8], de forma que los individuos con más probabilidades de reproducirse no sólo son aquellos mejor dotados genéticamente para realizar ese comportamiento, sino también aquellos mejor dotados genéticamente para aprender a realizar ese comportamiento u otro relacionado [87].

Bajo la denominación de algoritmos evolutivos, o computación evolutiva, se engloban todas las técnicas que tratan de simular el proceso de la evolución natural [23]. Los tipos principales son los algoritmos genéticos, las estrategias de evolución y la programación evolutiva, que describiremos a continuación, y que se diferencian fundamentalmente en los mecanismos de reproducción utilizados. También describiremos un nueva aproximación, basada en un ecosistema con especies, conocida como algoritmos macroevolutivos, y que ofrece unos resultados prometedores.

Algoritmos genéticos.

Las bases de los algoritmos genéticos fueron propuestas por John Holland a principios de la década de 1960 y la idea de los algoritmos genéticos como tales fue propuesta por él mismo en 1975 [47]. El funcionamiento de este tipo de algoritmos se puede resumir en lo siguiente.

Los valores de los genes de los individuos de la población inicial se inicializan aleatoriamente. Se calcula la calidad de los individuos de la población inicial. A partir de ese momento, el proceso evolutivo es un bucle en el cual se seleccionan los individuos a reproducirse siguiendo algún criterio relacionado con la calidad, se obtiene una población de hijos mediante el cruce y mutación de individuos, se evalúan los individuos de la nueva población, se seleccionan los individuos de la población de los padres a ser reemplazados y se funden ambas poblaciones.

Originalmente, los cromosomas estaban codificados en binario, es decir, cada gen era un 0 o un 1. Posteriormente se han utilizado otras codificaciones como, por ejemplo, codificar cada gen como un número real.

El método original de selección de los individuos a reproducirse es la selección proporcional, también conocido como selección por "ruleta". Cada individuo tiene una probabilidad de ser seleccionado para reproducirse igual a su calidad dividida por la suma de las calidades de todos los individuos. Este método, en algunas ocasiones, tiene el problema de aplicar muy poca presión selectiva. Por ejemplo, si se desea hallar el valor de x que maximiza $y=ax^2+b$ y b es muy grande, todos los individuos tendrán una calidad parecida y, por tanto, una probabilidad de reproducirse muy similar. Una forma de arreglar este problema es escalando las calidades. Sin embargo, puede existir el problema contrario. Si durante el proceso evolutivo aparece un individuo con mucha mayor calidad que el resto de los individuos, la población rápidamente convergerá hacia ese individuo, ya que su probabilidad de reproducirse es mucho mayor que la de los demás. Si ese individuo se correspondía con un máximo local, la evolución probablemente nunca llegue a la solución real.

Existen otros dos tipos de selección que pretenden arreglar este problema. Una es la selección por posición [107] en la que los individuos son ordenados por su calidad y su probabilidad de selección es una función de su posición en esa lista ordenada. Alterando dicha función se altera la presión selectiva. El otro tipo de selección es la selección por torneo [38], donde se escoge aleatoriamente un grupo de individuos y dentro de este grupo se selecciona para reproducirse aquel que tiene mayor calidad. Este proceso se repite hasta que se tienen todos los individuos que se necesitan para reproducirse. Cambiando el tamaño de ese grupo se altera la presión de selección: a mayor tamaño, mayor presión.

El cruce es el operador principal en los algoritmos genéticos. Consiste en escoger dos individuos, un punto de cruce, y crear dos individuos nuevos de tal forma que cada uno tenga los primeros n genes de un padre y los siguientes m-n genes del otro (siendo m la longitud total del cromosoma en genes). Existen otras variantes de cruce, usando, por ejemplo, dos o tres puntos de cruce en lugar de uno. Otra variante más consiste en que los descendientes se crean alternando los genes de los padres (cruce uniforme). La idea del operador cruce se basa en que, dados dos individuos con buena calidad pero por razones distintas (la buena calidad de los individuos viene dada por el valor de dos grupos de genes distintos en cada individuo), al cruzarse, el individuo resultante se quede con los genes que provocaban una buena calidad en ambos padres, dando lugar a un individuo todavía mejor. Esto evidentemente no se produce siempre, sino con una cierta probabilidad, pero cuanta mayor es la calidad de un individuo más veces se reproduce y, por tanto, mayor es la probabilidad de que la parte de su genoma causante de su buena calidad se mezcle con la parte correspondiente de otro individuo, dando lugar a individuos cada vez mejores, que tendrán una mayor probabilidad de reproducirse en posteriores generaciones, guiando así el proceso evolutivo.

Si la codificación es binaria, es decir, cada gen es un 0 o un 1, la mutación consiste en cambiar el gen con una determinada probabilidad. Si la codificación es distinta se amplia la definición de forma que cada gen se cambia, también con una determinada probabilidad, por otro gen válido siguiendo una distribución uniforme. En este caso existen, sin embargo, otras posibilidades. Si los genes son números reales, es frecuente que la mutación consista en sumar o restar al valor del gen una cantidad aleatoria con una distribución distinta a la uniforme (una normal, por ejemplo). La mutación en los algoritmos genéticos es un medio de introducir variedad en la población y, por eso, la probabilidad de mutación suele ser baja. Habitualmente se escoge de tal manera que de media se produzca una mutación en cada cromosoma.

Para seleccionar los individuos a ser reemplazados también se tienen varias alternativas. Podemos reemplazar los peores individuos o reemplazar aleatoriamente. En el primer caso aceleramos el proceso evolutivo y perdemos variedad. En el segundo podemos perder, quizás para siempre, buenos individuos y frenar la convergencia o incluso impedirla.

Holland dotó a su algoritmo inicial de una base teórica mediante la cual se garantizaba la convergencia, aunque no necesariamente a la mejor solución. Para ello define el concepto de "esquema" como una partición del espacio de búsqueda. Dado que los genes en su planteamiento eran bits, los individuos y, por tanto, los puntos en el espacio de búsqueda, se representan como cadenas de bits de longitud L. Un hiperplano en este espacio puede representarse como una cadena de longitud L formada por elementos del conjunto {0, 1, #}, donde # actúa como comodín y sustituye al 0 y al 1. Partiendo de lo que denominó "building block hypothesis" (hipótesis de los bloques constructivos), en la que afirmaba que mediante la recombinación de dos individuos buenos se obtiene otro individuo todavía mejor si los operadores genéticos conservan los bloques de elementos definidos (0s y 1s) de ambos individuos que contribuyen a su buena calidad, demostraba la convergencia apoyándose en el hecho de que si dos individuos son instancias de un mismo esquema, sus descendientes también lo serán [47].

Como se ha visto, existen muchas variantes de algoritmos genéticos, y la elección de una u otra dependerá totalmente del dominio del problema, ya que no hay ninguna que sea mejor que todas las demás en todos los casos. Existe además un inconveniente adicional. Hay una serie de problemas, denominados deceptivos [36], en los que el proceso de evolución no llega, o es muy difícil que llegue, a la solución óptima, y se queda en algún máximo local del espacio de soluciones. A veces, incluso el cruce de dos individuos con elevada calidad, aun conservando los bloques de genes que determinan esa buena calidad tanto por parte de un antecesor como por parte del otro, da lugar a un individuo de peor calidad que sus antecesores. Esto suele darse cuando la bondad de un gen dentro del cromosoma depende mucho de otros genes (epístasis). Es decir, un individuo con un determinado valor para un gen puede tener un valor alto o bajo de calidad dependiendo del valor de otro gen. Esta correlación entre los genes

puede provocar que al cruzar dos individuos con un valor alto de calidad obtengamos individuos con un valor bajo de calidad si los padres representan soluciones buenas pero distintas. Desgraciadamente, cuando los cromosomas representan pesos de RNAs suele existir una gran correlación entre los genes. Debido a esto, es frecuente no utilizar el operador cruce y en su lugar simplemente copiar el individuo varias veces en la nueva población después de las mutaciones pertinentes, o bien usar estrategias evolutivas en lugar de algoritmos genéticos. La primera solución es aplicada, por ejemplo, en [80][67][79][66][88], donde no se realizan cruces y se fija una probabilidad de mutación un poco más elevada respecto a si hubiera cruce (entre el 10 y el 30%) pero sin llegar a la mutación de todos los genes como en las estrategias evolutivas. Las mutaciones consisten en sumar o restar pequeñas cantidades a los pesos de la RNA (genes).

Estrategias evolutivas.

Las estrategias evolutivas, que fueron ideadas por Rechenberg [92] y Schwefel [100], se diferencian principalmente de los algoritmos genéticos en que el operador que guía el proceso evolutivo es la mutación y no el cruce. De hecho, en la propuesta inicial de estrategias evolutivas no existía el operador cruce, y la generación de individuos nuevos consistía en sumar a cada individuo un vector aleatorio, cuyos valores, entre 0 y 1, seguían una distribución normal, multiplicado por un escalar mayor que 0 y comprobando si la calidad del nuevo individuo era mejor que la del individuo original. En caso afirmativo lo reemplazaba en la población y si no el individuo original pasaba a la siguiente generación.

Actualmente, se puede considerar que existen, básicamente, dos tipos de estrategias evolutivas. Las estrategias evolutivas ($\mu+\lambda$) donde μ individuos generan λ hijos y se seleccionan para la siguiente generación los μ mejores individuos del total $\mu+\lambda$, y las estrategias evolutivas (μ,λ) donde μ individuos generan λ hijos y se seleccionan para la siguiente generación los μ mejores individuos de los λ hijos. En ambos casos, los λ hijos se generan mutando todos los genes de los cromosomas. Esa mutación consiste en sumar un valor a cada gen. El total de los valores sumados a los genes suele seguir una distribución de media cero. El valor máximo a sumar a los genes suele venir fijado por una regla o puede incluirse dentro del cromosoma como un gen más. Igualmente, dicho valor puede ser el mismo para todos los genes del cromosoma o puede haber varios valores para distintos grupos de genes. Inicialmente [92], los genes estaban codificados como números reales, pero nada impide usar otra codificación, adaptando el operador mutación según sea necesario.

El uso de estrategias de evolución no está, hasta el momento, muy extendido a la hora de generar controladores de robots, aunque Salomon [97], Meeden [75][1] y Harvey [41] las han usado y demostrado, al menos en los casos concretos en los que las han

1 La autora denomina en este artículo algoritmo genético al algoritmo evolutivo empleado, pero es, de hecho, una estrategia evolutiva, ya que no realiza cruces y muta todos los genes de cada individuo, sumando/restando cantidades en un rango en torno al valor original.

aplicado, una mayor velocidad en la obtención de una solución aceptable que los algoritmos genéticos.

Igualmente, han surgido variantes de las estrategias de evolución que aplican el operador de cruce entre dos individuos antes de aplicar el operador mutación, con lo que la diferencia entre los algoritmos genéticos y las estrategias de evolución se hace a veces confusa. Y es que no hay consenso en cuanto a la eliminación del operador de cruce, ya que, aun reconociendo que cuando hay epístasis puede ser perjudicial, es un operador que permite atravesar rápida y eficazmente "valles" en el espacio de búsqueda y salir de máximos locales[2]. Por ello, en la práctica, casi todo el mundo utiliza "su versión" de algoritmo evolutivo en la que intervienen, en mayor o menor medida, ambos operadores. Parece razonable que cuanto mayor es el grado de epístasis de un problema, mayor debe ser la importancia de la mutación y menor la del cruce, aunque sin llegar éste a desaparecer completamente. La dificultad estriba en que, en general, no existe todavía un método útil para determinar el grado de epístasis de un problema.

Programación evolutiva.

La programación evolutiva, originalmente definida por Fogel [35], fue planteada desde un principio, a diferencia de las otras técnicas evolutivas, como una alternativa a la inteligencia artificial clásica, una forma de obtener comportamientos inteligentes sin basarse en heurísticas.

Es similar a las estrategias evolutivas, pero hace hincapié en que lo realmente importante es el fenotipo (el controlador final y, por ende, el comportamiento) y no el genotipo (la codificación del individuo en el cromosoma). Así, los cromosomas son programas en un lenguaje de alto nivel, que describen un autómata de estados finitos (como era originalmente) o una RNA. La mutación es más compleja que en los casos anteriores, y consiste en cambiar una sentencia o grupo de sentencias de ese lenguaje de alto nivel por otra sentencia o grupo de sentencias (que no tiene por qué ser de la misma longitud), comprobando que no se generan individuos imposibles, incorrectos sintáctica o semánticamente. Por tanto, estamos hablando aquí de que, inevitablemente, los cromosomas van a ser de longitud variable.

Existe una variante, a caballo entre la programación evolutiva y los algoritmos genéticos, denominada programación genética [64], en la que los cromosomas son programas en un lenguaje de más bajo nivel y donde también existe el operador cruce, en el que dos programas se cruzan por un punto determinado, teniendo en cuenta que el intercambio de dos trozos de código de dos cromosomas debe resultar sintáctica y semánticamente correcto. Dain [22] utiliza este método. Para ello define un lenguaje con las operaciones básicas que puede realizar el robot, obteniendo con facilidad un

2 En todo este trabajo se habla de máximos locales, ya que los problemas en robótica evolutiva suelen consistir en maximizar el valor de una función de calidad. Nada impide, sin embargo, que un problema consista en minimizar el valor de una función con lo que, en ese caso, se debe hablar de mínimos locales.

comportamiento para seguir paredes. Otro ejemplo de programación genética en robótica es el trabajo de Reynolds [94], que utilizando como lenguaje el LISP obtiene un controlador con una función similar a la anterior: seguir pasillos evitando colisiones. Sin embargo, ninguno comprueba los resultados en robots reales con lo que no se puede observar el comportamiento en presencia de ruido y en entornos reales no cuadriculados.

El mayor inconveniente de la programación evolutiva radica en que en cada problema se deben fijar las operaciones permitidas sobre los cromosomas y establecer una codificación fenotipo-genotipo que sea flexible y al mismo tiempo permita obtener resultados en un tiempo razonable. Como resultado de esto el lenguaje utilizado suele ser demasiado específico y no permite su extrapolación a otros problemas diferentes. Pese a ello, la programación evolutiva es un campo muy prometedor, sobre todo aplicado a RNAs, ya que permite, mediante una correcta elección del lenguaje de alto nivel empleado, evolucionar la topología completa de una red, y no sólo los parámetros de una topología predefinida como es habitual en el entrenamiento y el resto de técnicas evolutivas. Este proceso, conocido como desarrollo, es utilizado, entre otros, por Kaelbling [62] y Meyer [63][78].

Algoritmos macroevolutivos.

Un algoritmo macroevolutivo es un tipo muy reciente de algoritmo evolutivo que fue presentado por Marín y Solé en 1999 en [72]. La diferencia fundamental de esta aproximación estriba en que mientras en los otros algoritmos evolutivos una población de individuos evoluciona mediante el principio de que cuanto mejor es un individuo más probabilidades tiene de reproducirse, en los algoritmos macroevolutivos se considera una nueva escala evolutiva, la de especies, y así un ecosistema formado por un conjunto de especies evoluciona mediante la extinción de las peores y la apropiación de su nicho ecológico por parte de especies nuevas. Aunque tanto en las implementaciones originales de los autores como en la realizada en este trabajo (que veremos en el apartado 4.4.3) una especie está formada por un único individuo, es importante tener esta diferencia de concepto en mente porque es la base de los algoritmos macroevolutivos y representa una puerta abierta a que cada especie esté formada por varios individuos y evolucione independientemente.

El funcionamiento de los algoritmos macroevolutivos se presenta en la figura 4. Básicamente, se calcula una matriz de conexión, en la que cada elemento mide la diferencia de calidades entres dos especies ponderándola por la distancia en el espacio de soluciones que existe entre ellas, de forma que dicha diferencia de calidad es más importante cuanto más cerca están las especies en dicho espacio. Si la suma de todos los elementos de la matriz de conexión en los que interviene una especie es positiva (coeficiente de supervivencia positivo, según la definición de los autores), dicha especie sobrevive. En caso contrario, se extingue. Si sobrevive se mantiene tal cual en la siguiente generación. Si se extingue es sustituida por una nueva especie colonizadora

que puede ser generada aleatoriamente o puede derivarse de la especie extinguida y de otra cualquiera que sobrevive.

Sean:

d *la dimensionalidad del espacio de búsqueda*

$f:\mathbb{R}^d \rightarrow \mathbb{R}$ *la función de calidad*

$p_i = (p_i^1, \ldots, p_i^d)$ *la especie* i

$W_{i,j}$ *la conexión entre la especie* i *y la especie* j

h_i *el coeficiente de supervivencia para la especie* i

p_b *una especie con coeficiente de supervivencia* ≥ 0

p_n *una especie generada aleatoriamente*

$\xi \in |0,1|$ *número aleatorio con distribución uniforme*

$\lambda \in |-1:1|$ *número aleatorio con distribución uniforme*

ρ *y* τ *parámetros del algoritmo*

Entonces:

$t \leftarrow 0$

Inicializar cada p_i

Evaluar cada p_i

mientras no se cumpla el criterio de parada hacer

Calcular la matriz de conexiones W *donde* $W_{i,j} = \left\{ \begin{array}{l} \dfrac{f(p_i) - f(p_j)}{|p_i - p_j|}, si\, |p_i - p_j| \neq 0 \\ 0, en\ caso\ contrario \end{array} \right\}$

Calcular el coeficiente de supervivencia $h_i(t) = \sum_{j=1}^{p} W_{i,j}(t)$

Calcular el estado $S_i(t+1) = \left\{ \begin{array}{l} 1, si\, h_i(t) \geq 0 \\ 0, en\ caso\ contrario \end{array} \right\}$

Colonización: $p_i(t+1) = \left\{ \begin{array}{l} p_i(t), si\, S_i(t+1) = 1 \\ p_b(t) + \rho\, \lambda\, (p_b(t) - p_i(t)), si\, S_i(t+1) = 0\ y\ \xi > \tau \\ p_n, si\, S_i(t+1) = 0\ y\ \xi \leq \tau \end{array} \right\}$

$t \leftarrow t+1$

Figura 4: Funcionamiento de un algoritmo macroevolutivo.

Hay dos parámetros que determinan el funcionamiento del algoritmo macroevolutivo: ρ, que marca la distancia máxima que una nueva especie generada a partir de una extinta y una superviviente puede tener respecto a ésta última, y τ, que controla el porcentaje de especies aleatorias que se generan en cada generación.

En definitiva, ρ determina cómo se lleva a cabo la explotación, es decir, cómo se realiza la búsqueda en las proximidades de individuos ya existentes. Cuánto más pequeño sea más cerca estarán las nuevas especies de las supervivientes y, por tanto, mayor será la presión evolutiva, puesto que más rápido se abandonarán las zonas del espacio de búsqueda con aparentemente peores especies.

Por su parte, τ controla el balance entre explotación y exploración (búsqueda de nuevo material genético, nuevos individuos no necesariamente cerca, o incluso

preferentemente alejados, de los ya existentes). Éste balance, como veremos en el apartado 4.4.3, es fundamental. Si τ es muy grande se generarán muchas especies aleatoriamente y estaremos ante una evolución guiada por la exploración, lo cual es útil para buscar nuevo material genético. Si es muy pequeño se generarán pocas especies aleatoriamente y estaremos ante una evolución guiada por la explotación, lo que quiere decir que se realizará una búsqueda exhaustiva en las proximidades de los individuos ya existentes, a costa de no buscar nuevo material genético en zonas no exploradas. Lo habitual es que este parámetro actúe como una "temperatura" y decrezca con el tiempo, de forma que se empiecen generando muchos individuos aleatoriamente y, a medida que avanza el proceso evolutivo, cada vez se generen menos individuos aleatorios.

Una ventaja de los algoritmos macroevolutivos respecto a otros es que solamente hay dos parámetros que determinan su funcionamiento, con lo que ajustarlo a un problema determinado es mucho más fácil. Pensemos, por ejemplo, en los algoritmos genéticos, donde es necesario seleccionar el tipo de selección, el tipo de cruce, el tipo de mutación, la probabilidad con la que se realizan ambos operadores genéticos, el tipo de reemplazo, etc., y donde además muchos de estos factores tienen a su vez otros parámetros que ajustar.

La otra ventaja de los algoritmos macroevolutivos es su particular forma de realizar la búsqueda por el espacio de soluciones, dando la oportunidad a que se reproduzcan todas las especies antes de ser extinguidas, lo cual se mostrará muy útil, como veremos, en problemas propensos a caer en máximos locales. Esto se traduce en que el algoritmo, en líneas generales, tiene una menor presión evolutiva que los algoritmos genéticos o las estrategias evolutivas, ya que toda especie, antes de desaparecer, puede dejar descendencia en la población con mucha mayor probabilidad que en los otros algoritmos evolutivos. Además, al tener todas las especies supervivientes la misma probabilidad de ocupar el nicho dejado por las que se extinguen, no existe el problema del superindividuo. No sólo la presión es menor, sino que la dinámica de la evolución tiende a generar agrupamientos alrededor de las mejores especies, con lo que la explotación es también muy eficiente.

4.3 Alternativas en la realización del proceso de evaluación de los controladores

Un elemento clave a tener en cuenta en el desarrollo de controladores mediante evolución es dónde realizar los procesos de evaluación que ésta implica. Podría parecer que lo ideal sería realizarlos sobre el robot real y en varios entornos representativos del conjunto de entornos en los que se va a desenvolver. Pero la lentitud del proceso, así como la posibilidad de dañar físicamente el robot mientras éste no ha aprendido/evolucionado para llevar a cabo su tarea correctamente, hace necesario el planteamiento de alternativas basadas en mayor o menor medida en la simulación. Pero la simulación también tiene sus propios problemas, ya que, al no ser ésta perfecta, la transferencia exitosa de comportamientos al robot real desde la simulación no es trivial. En este apartado trataremos estas distintas posibilidades, viendo cuáles son las ventajas y los inconvenientes de cada una.

Evaluación en entornos reales

Como se ha dicho, idealmente, la evaluación de los controladores debería realizarse en el robot o robots reales. Sin embargo, esto tiene una serie de problemas. El primero es que el tiempo necesario para realizar la evolución se dispara en cuanto el controlador deja de ser trivial. El segundo es que hay que disponer de mecanismos que permitan la reubicación de los robots en el entorno en caso de ser necesario. Por ejemplo, para que el robot inicie cada evaluación en una posición aleatoria en el entorno o en caso de quedarse atascado, para que pueda continuar moviéndose. Y el tercero es que la propia integridad física de los robots puede verse amenazada en las primeras generaciones. Estos inconvenientes restringen la evaluación en robots reales a comportamientos sencillos en entornos controlados. Floreano y Mondada [33][34][82] han empleado este método pero en comportamientos muy sencillos (evitar obstáculos y navegación) y donde se pone de manifiesto la gran cantidad de tiempo necesario para la evolución.

Evaluación en entornos simulados

La alternativa a la evaluación en un robot físico es la evaluación en un entorno simulado por ordenador. Este método no sufre de los inconvenientes del método anterior. La simulación no tiene por qué producirse en tiempo real con lo que transcurre mucho más rápido. Mover los robots en el entorno no es problema y, obviamente, no van a sufrir daños materiales. Además, el entorno puede cambiar a lo largo de la evolución de la forma que se desee, facilitando la obtención de controladores más robustos, y es posible incluir en el cálculo de la calidad de los individuos factores que no estarían disponibles si la evaluación fuese en el robot real [79].

Sin embargo, la evaluación en un entorno simulado no carece de problemas. Estos derivan del hecho de que la simulación que hagamos nunca va a ser un modelo perfecto de la realidad, por lo que habrá que desarrollar mecanismos que permitan que los

controladores que se comportan correctamente en un entorno simulado lo hagan también en el mundo real. Para ello es necesario tener en cuenta una serie de consideraciones que pasamos a concretar a continuación.

Para empezar, la respuesta de los sensores y de los actuadores en el simulador se debe obtener a partir del comportamiento real de dichos sensores y actuadores en el robot real, no a partir de su comportamiento teórico (especificaciones). En este camino, muchos autores prefieren capturar la respuesta real de los sensores y de los actuadores ante las superficies y los objetos que se va a encontrar el robot en la realidad, antes que modelar matemáticamente dichos sensores y actuadores, ya que dos sensores o actuadores, aun siendo de idéntico modelo, pueden tener respuestas significativamente distintas [68][67][57][66]. Esto funciona muy bien cuando la variedad de objetos y superficies no es muy grande, como es habitual en los entornos construidos a medida, pero puede resultar dificultoso en entornos más libres.

Por otra parte, tampoco es conveniente que la simulación sea excesivamente compleja y, por tanto, lenta, puesto que entonces estaríamos perdiendo parte de las ventajas de una evaluación en un entorno simulado. Jakobi [54][53][55] muestra que, al menos en los ejemplos que él plantea, es posible obtener comportamientos en entornos simulados que se comporten correctamente en el entorno real utilizando una simulación sencilla. Para ello define la siguiente metodología:

- Definir con precisión el comportamiento, qué es lo que queremos que realice el robot.
- Identificar el "conjunto base de características" del entorno real, es decir, aquellas características que son relevantes para el comportamiento que queremos obtener.
- Definir un modelo de las interacciones de los miembros del conjunto base con el robot, tanto con sus sensores como con sus actuadores. Este modelo deberá ser implementado en la simulación.
- Definir un test de calidad adecuado, de tal forma que podamos estar seguros de que un individuo con una elevada calidad se comporta realmente como queremos.
- Todos los aspectos de la implementación que no formen parte del conjunto base deben ser variados aleatoriamente entre evaluación y evaluación de cada individuo, y de una forma lo suficientemente intensa como para que el comportamiento del robot no sea fiable a no ser que ignore dichos aspectos de implementación. Se dirá entonces que el controlador es "exclusivo respecto al conjunto base".
- Todos los aspectos del conjunto base deben ser variados aleatoriamente dentro del rango para el cual queremos que el comportamiento funcione correctamente, de tal forma que el controlador no sea fiable si el robot no se comporta correctamente en todo ese rango de valores. Se dirá entonces que el controlador es "robusto respecto al conjunto base". Estas variaciones en los elementos del conjunto base tienen que

ser lo suficientemente grandes como para que el robot se comporte adecuadamente en el entorno real, pero no excesivamente grandes, como puedan serlo las aplicadas a los aspectos que no forman parte del conjunto base, sino será imposible obtener el controlador.

En virtud de la aplicación de la metodología definida por Jakobi y, aunque no nos preocupase que la simulación fuese muy compleja, ésta no va a ser nunca perfecta y, como consecuencia, deberemos introducir ruido en diversos aspectos de la simulación, para que los controladores toleren las diferencias que sin duda habrá entre la simulación y el entorno real y sean lo más robustos posible. Existen varias formas de ruido. El ruido tradicionalmente añadido a sensores y actuadores consiste en pequeñas variaciones aleatorias en los valores sensados/aplicados. Otra forma de ruido que parece producir buenos resultados consiste en simular pequeñas alteraciones en la posición de los objetos ante los sensores [79], las cuales se pueden traducir en grandes variaciones en los valores sensados dependiendo del modelo del sensor. Este tipo de ruido simula el distinto comportamiento que pueden tener los sensores dependiendo del tipo de superficie de los objetos, de la luz presente en el entorno, etc., y es especialmente útil con algunos tipos de sensores como los de infrarrojos.

Uno de los autores que llevan a cabo la evolución en un entorno simulado es Dain [22], que la realiza sin añadir ningún tipo de ruido, pero el entorno es muy simple y no comprueba los resultados en robots reales. Reynolds [94] añade ruido a la simulación y realiza varias ejecuciones con un mismo controlador, quedándose siempre con la de menor calidad en busca de comportamientos robustos, pero tampoco comprueba los resultados en robots reales. Miglino y col. [80] sí comprueban el comportamiento en un robot real, aunque con un entorno hecho a medida, añadiendo ruido en la simulación y constatando un mejor rendimiento del robot con él, hasta el punto de considerar su uso en la evaluación imprescindible.

Pese a todas las consideraciones que deben ser tenidas en cuenta para que los controladores obtenidos mediante simulación funcionen correctamente en el robot real, la evaluación en entornos simulados es el caso más habitual, dada la imposibilidad de obtener controladores complejos en un tiempo razonable realizando la evaluación sobre robots reales, y es el que ha sido utilizado en este trabajo.

Evaluación mixta

Una variante seguida por algunos autores consiste en realizar la mayor parte de la evaluación de los individuos en un entorno simulado dejando que las últimas generaciones se realicen sobre robots reales, hasta que se obtiene una calidad similar a la que teníamos al final de la evolución en el entorno simulado. Cuando esto es posible, no tenemos que ser tan cuidadosos con los mecanismos de transferencia a la realidad, ya que ésta la realiza el propio algoritmo evolutivo. Esta es la solución empleada por Lund y Miglino [68] y Miglino y col. [79] que observan cómo, efectivamente, cuando el

comportamiento en el entorno real no es tan bueno como en el simulado, prolongar la evolución en el entorno real durante unas generaciones devuelve el comportamiento a un nivel de calidad similar al previamente obtenido en el entorno simulado. Sin embargo, la evaluación en el entorno real en esas últimas generaciones a veces ni siquiera es posible, ya sea porque es muy costoso, porque no se disponen de los medios necesarios o porque es excesivamente complejo.

4.4 Estudio de la solución adoptada: evolución de RNAs en entorno simulado

Una vez determinadas las herramientas que se van a utilizar tanto para implementar los controladores como para obtenerlos, es necesario analizar cómo se deben utilizar éstas para la consecución de nuestros objetivos, lo cual veremos en este apartado.

Como hemos visto en los apartados anteriores de este capítulo, se ha optado en este trabajo por implementar los controladores utilizando RNAs y obtenerlos automáticamente en entornos simulados utilizando algoritmos evolutivos, validando finalmente los controladores obtenidos ejecutándolos en robots reales. Las RNAs se codificarán en los cromosomas de forma directa, es decir, cada parámetro a evolucionar (peso, bias, pendientes de las sigmoides, retardos sinápticos, etc.) será un gen. Los valores de los genes estarán en el intervalo [-1:1] y deberán ser desnormalizados a la hora de construir la RNA asociada a cada cromosoma. Por ejemplo, si el valor de un peso de una sinapsis estuviese en el rango [-6:6], los genes asociados a pesos sinápticos tendrían que ser multiplicados por 6 para generar la RNA.

Para comprobar la validez de la metodología planteada se ha desarrollado un entorno de evolución y un simulador integrados (descrito en detalle en los apéndices), al que se ha denominado SEVEN (de *Simulation and EVolution ENvironment*), que implementará todas las técnicas aquí descritas y que será utilizado para obtener los controladores dentro de esta metodología. SEVEN permite diseñar entornos simulados sobre los que realizar las evoluciones y probar los resultados antes de comprobarlos en los robots reales. La misma aplicación realiza las evoluciones y, como veremos, es capaz de aprovechar eficientemente los recursos computacionales disponibles. Como robots reales en los cuales probar los controladores se han utilizado un Rug Warrior y un Pioneer 2-DX (ambos descritos también en los apéndices).

El primer problema que debe ser abordado viene determinado por la elección de la simulación como método por el cual llevar a cabo la obtención de los controladores. Como se comentó en el apartado 4.3, la simulación de un entorno no puede ser perfecta y, aunque pudiese serlo, tampoco interesa que lo sea por el elevado coste computacional que tendría y que nos haría perder una de las ventajas de la simulación, que es su mayor rapidez frente a la utilización del entorno real en la obtención de los controladores. Veremos, por tanto, qué pasos se deben seguir y qué técnicas se deben utilizar para que los controladores obtenidos mediante evoluciones en entornos simulados funcionen correctamente en los robots reales.

Una vez solucionado ese problema, entraremos ya en consideraciones relacionadas con la utilización de algoritmos evolutivos para la obtención automática de los controladores en la metodología desarrollada. La primera consideración será la filosofía a seguir a la hora de asignar calidades a los individuos. Veremos que existen diversas alternativas y se explicará cuál se ha seleccionado y la forma en la que se ha desarrollado. Posteriormente, veremos el comportamiento de varios tipos de algoritmos

evolutivos en la obtención de controladores, para determinar cuáles se comportan mejor y en qué circunstancias. Aquí, se verá también cómo afecta el tamaño de la población y su distribución inicial al funcionamiento de los algoritmos evolutivos, y se presentarán y analizarán los efectos en la evolución del uso del concepto de raza que, básicamente, consiste en dividir la población total en poblaciones más pequeñas que evolucionan por separado con intercambios de individuos (migraciones) periódicos. Finalmente, se elaborará un algoritmo que permitirá la distribución y paralelización de SEVEN, favorecidas precisamente por el efecto de división en la evolución producida por las razas.

Así, al finalizar este apartado, habremos analizado todas las consideraciones relevantes respecto a la obtención automática de controladores en la metodología aquí presentada. En el capítulo siguiente veremos cómo la utilización de RNAs con conexiones sinápticas de mayor complejidad que las empleadas tradicionalmente en el campo de la robótica nos permitirá añadir características que aumenten la capacidad de los controladores. Así, el uso de retardos sinápticos permitirá a los controladores procesar patrones temporales determinados, mientras que la utilización de sinapsis gausianas proporcionará la capacidad de evolucionar controladores que utilizan sólo aquellos datos sensoriales necesarios, y al propio algoritmo evolutivo la capacidad de reducir el tamaño de la RNA durante la evolución, si ésta está inicialmente sobredimensionada.

Finalmente, en el capítulo 6 se desarrollará una arquitectura multimódulo sobre la que utilizar esta metodología y que nos permitirá cumplir con el objetivo de obtener controladores para robots autónomos de forma automática e incremental.

4.4.1 Transferencia de comportamientos al entorno real

Como ya se ha comentado en el apartado 4.3, para obtener comportamientos robustos para robots reales, realizando la evaluación en entornos simulados, que son una simplificación del entorno real, es necesario tener en cuenta una serie de consideraciones, como pueden ser las características topológicas del entorno o entornos simulados a utilizar o qué tipos de ruido se deben emplear para compensar la falta de precisión en el modelado del entorno en la simulación. En este apartado hablaremos sobre estas consideraciones y presentaremos algún ejemplo en el que se podrán apreciar las implicaciones y los efectos de las técnicas empleadas para lograr que los controladores obtenidos en mundos simulados funcionen en entornos reales.

Una primera consideración a tener en cuenta es que el mundo simulado utilizado en la evolución debe ser suficientemente representativo del mundo real al cual se va a tener que enfrentar el robot. Las características topológicas del entorno simulado deben ser similares a las presentes en la realidad o el controlador será inadecuado. Puede ser incluso necesario tener que utilizar más de un entorno simulado si no podemos representar en un único entorno todas las posibilidades que el robot puede encontrarse en la realización de la tarea, en cuyo caso será imprescindible evaluar cada individuo en varias ocasiones, al menos una vez en cada uno de ellos, siendo su calidad un promedio de las calidades obtenidas en cada uno de los entornos. Si el porcentaje de aparición de esos entornos en la realidad no es igual, es decir, unos son más frecuentes que otros, entonces esta diferencia puede reflejarse en el cálculo de la calidad del individuo. Por ejemplo, si un entorno se suele presentar el doble de veces que otro, parece lógico que la calidad obtenida por un individuo en ese entorno tenga un peso doble a la calidad obtenida en el otro. La utilización de más de un entorno durante la evolución posee, además, otras ventajas:

- Evitar que el controlador memorice el entorno utilizado y sea incapaz de generalizar su comportamiento a otros entornos.
- Facilitar la evolución en el caso de que el comportamiento sea lo suficientemente difícil como para no existir un gradiante en la calidad de los individuos iniciales, que poseen controladores aleatorios. En este caso, se puede empezar la evolución con un entorno sencillo e irlo complicando a medida que transcurren las generaciones.

Otra consideración muy importante para que el comportamiento obtenido en simulación funcione, y de manera robusta, en el robot real, es la utilización necesaria de ruido en la simulación. Como se comentó en el apartado 4.3, y de acuerdo a la metodología desarrollada por Jakobi [54][53][55], todos los aspectos del conjunto base deben ser variados aleatoriamente dentro del rango para el cual queremos que el comportamiento funcione correctamente, de tal forma que el controlador no sea fiable si el robot no se comporta correctamente en todo ese rango de valores. Esas variaciones aleatorias son las que se suelen denominar ruido en la simulación. En el mundo real,

todo, desde los sensores y actuadores del robot, hasta el entorno que lo rodea, tiene un comportamiento extremadamente difícil de modelar con detalle. Por ejemplo, para sensores tales como infrarrojos o sónares, es necesario tener en cuenta cosas tales como las propiedades reflectantes de la superficie de los objetos: el comportamiento de los sensores de infrarrojos se ve afectado por el color de dicha superficie, mientras que los sónares lo son por su forma y textura. Los actuadores tienen otras dificultades. Por ejemplo, para el caso de las ruedas, la distinta superficie sobre la que éstas giran puede causar deslizamientos, patinazos, etc., que hacen que ante idénticos valores proporcionados a los motores se obtengan distintas respuestas por parte del robot. El entorno no sólo puede afectar al robot por las características de los objetos que lo componen, sino que además puede cambiar dinámicamente (objetos móviles, "accidentes", etc.). Además, los problemas en sensores y actuadores no siempre vienen motivados por el entorno, porque los propios componentes mecánicos del robot no tienen un comportamiento constante a lo largo del tiempo, ya que sufren desgastes.

Debido a esto, la simulación nunca será perfecta. Pero tampoco interesa que lo sea. Cuánto más realista sea la simulación más tiempo de cálculo consumirá y, por tanto, más tiempo será necesario para obtener un resultado. Se debe, por tanto, buscar un equilibrio entre una simulación simple (rápida pero más propensa a producir individuos finales que no se comporten adecuadamente en el entorno real) y una excesivamente compleja (que no permita obtener resultados en un tiempo razonable).

Cuando no se puede modelar algo con perfección se recurre a modelos estadísticos, y ésta es también aquí la solución habitual. Una medida común consiste en introducir ruido aditivo de media cero en los sensores y actuadores, es decir, en cada movimiento del robot se suman pequeñas cantidades tanto a los valores sensados como a los introducidos en los actuadores, siguiendo esos valores alguna distribución de media cero. De esta forma, ambos son tolerantes a pequeñas desviaciones respecto a su comportamiento esperado. Pero esto en ocasiones no es suficiente, ya que hay otros factores que no pueden ser modelados en la simulación de forma completa y/o realista y que no tienen ese comportamiento de ruido blanco, sino que se traducen en perturbaciones sistemáticas en determinados parámetros del entorno. En este trabajo se han planteado técnicas de introducción de ruido alternativas con el fin de hacer tolerantes los comportamientos a dichas perturbaciones. Debido a la naturaleza de estas perturbaciones, los ruidos generados con estas técnicas se han denominado sistemáticos. Dependiendo de la causa de origen, tenemos tres tipos de ruido sistemático que describiremos a continuación: ruido de generalización, ruido sistémico y ruido temporal. Todos ellos se tratarán de la misma forma: fijando un conjunto de valores determinado para cada factor que se desee modelar con ruido sistemático, y seleccionando un elemento de ese conjunto al azar en cada evaluación de un individuo. Se podría definir un intervalo y seleccionar aleatoriamente cualquier valor de ese intervalo manteniéndolo constante durante todo el periodo de evaluación, pero de esta última forma los valores de los extremos del intervalo son seleccionados en raras

ocasiones y, además, dadas las capacidades de generalización de las RNAs, no lo hemos encontrado necesario.

- Ruido de generalización. El objetivo de este tipo de ruido es hacer los comportamientos más robustos ante cambios en la naturaleza del entorno o en la configuración del robot. Cuatro situaciones distintas han dado lugar al uso de ruido de generalización para el primero de los robots considerados, un Rug Warrior. La primera es que un pequeño golpe en un sensor de luz o en un emisor de infrarrojos puede hacer que éste apunte en otra dirección, alterando así el comportamiento del robot. Por ello es conveniente que la orientación de dichos sensores pueda variar en cada ejecución. Para permitir que los comportamientos que usan los sensores de luz sean tolerantes a diversas intensidades de luz ambiente, y que los sensores de infrarrojos sean tolerantes a diversos colores en la superficie de los objetos, también se ha empleado ruido de generalización en la intensidad de luz ambiente y en el rango de alcance de los sensores de infrarrojos. Finalmente, debido a que distintos niveles de carga en las baterías provocan que el robot se desplace más o menos en cada movimiento para una misma orden, se ha empleado ruido de generalización en la relación orden/distancia recorrida.
- Ruido sistémico. Como en todo aparato que no sea exclusivamente digital, no hay dos robots iguales. Para que el controlador funcione adecuadamente en distintas unidades del mismo robot, a veces será necesario introducir también alteraciones en los parámetros que determinan el modelo del robot, de forma similar al caso anterior. Por ejemplo, el Rug Warrior utilizado posee un defecto en el funcionamiento del conjunto motor-rueda izquierdo, que provoca que ante órdenes iguales para ir a una determinada velocidad, la rueda izquierda gire considerablemente menos que la derecha, variando además esa diferencia aleatoriamente dentro de un margen amplio.
- Ruido temporal. Cuando el controlador tiene que lidiar con información temporal, será necesario hacer que éste tolere cambios en el tiempo transcurrido entre eventos temporales. Dependiendo de la naturaleza de esos cambios, hará falta ruido blanco o ruido sistemático.

Cualquier tipo de ruido dificulta y ralentiza el proceso de obtención del comportamiento, y en especial el ruido sistemático. Éste último puede provocar que el entorno sea percibido de formas muy distintas en cada evaluación del robot y que la estrategia óptima sea también distinta, forzando al robot a tomar soluciones de compromiso. Dado que el ruido es aleatorio, las condiciones de evaluación para dos individuos determinados pueden no ser totalmente justas, en el sentido de que uno de ellos puede encontrarse con entornos más difíciles que el otro. Además, el mejor individuo de una generación puede obtener una calidad mucho peor en la siguiente generación si los entornos, debido al ruido, son percibidos de forma totalmente distinta. Este problema se producirá especialmente en las primeras generaciones, y para evitar que el algoritmo no converja a una solución debido a él, se debe aumentar el número de

evaluaciones de cada individuo, buscando un compromiso para evitar una ralentización excesiva del proceso evolutivo.

Todo este ruido puede parecer un inconveniente en la obtención de los controladores mediante simulación y favorecer la tesis de una evolución sobre un entorno real. Pero en realidad, aunque la incorporación eficiente de diversos tipos de ruido en la evolución puede llegar a ser compleja, el ruido, especialmente el sistemático, proporciona una ventaja enorme a la evolución sobre un entorno simulado. Añadiendo numerosos tipos de ruido variados y en cantidades considerables se obtienen controladores muy robustos que se enfrentan en su evolución a entornos y situaciones muy dispares, algo que sería muy complejo de proporcionar en una evolución sobre un entorno real, ya que habría que cambiar constantemente las características de éste para obtener un comportamiento que pueda afrontar con éxito una amplia variedad de entornos perceptualmente diferentes.

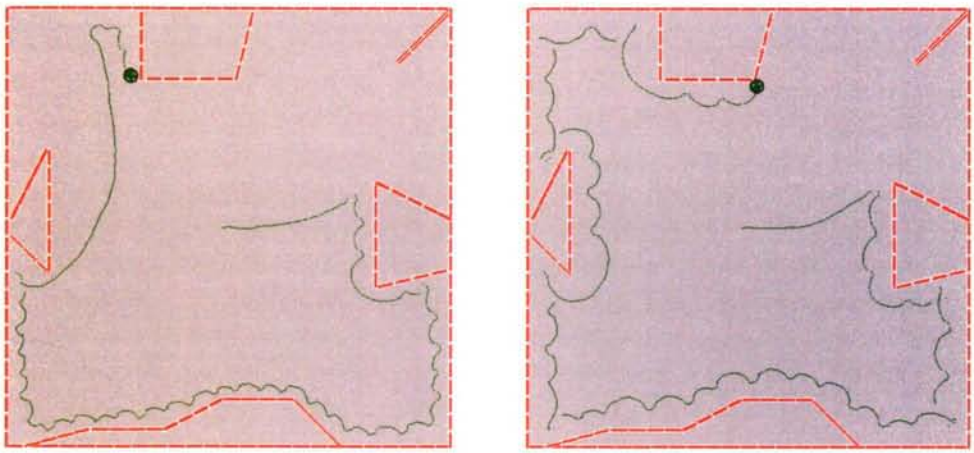


Figura 5: Comportamiento de búsqueda y seguimiento de paredes para el Rug Warrior usando neuronas de habituación. En la imagen de la izquierda se ha cambiado ligeramente la orientación del sensor de infrarrojos izquierdo. En la imagen de la derecha se ha cambiado el voltaje proporcionado por las baterías a 6.2V.

A continuación veremos un ejemplo de la importancia del correcto tratamiento del ruido en la evolución. El comportamiento que nos permitirá analizar esto es el de seguir paredes con búsqueda para el Rug Warrior utilizando neuronas de habituación (este tipo de neuronas y este ejemplo se comentan en detalle en el apartado 5.1). Básicamente, el Rug Warrior tiene que, partiendo de una posición alejada de las paredes, localizar una y comenzar a recorrer el entorno manteniéndose cerca de las paredes y sin chocar. Como primera explicación diremos que, al utilizar los sensores binarios de infrarrojos del Rug Warrior, es necesario el uso de información temporal para que el robot pueda discernir las situaciones necesarias para seguimiento y búsqueda de una pared. En dicho apartado 5.1 se ve cómo, utilizando ese tipo de neuronas de entrada que manejan implícitamente el tiempo, el robot es capaz de realizar la misión encomendada. La evolución allí

presentada tiene el habitual ruido de media cero en los valores de los sensores y actuadores. Como se comenta en el apartado mencionado, el zigzag de la trayectoria se explica por la naturaleza binaria de los sensores que obliga al robot a realizar esos movimientos para saber si se aleja o se acerca de una pared.

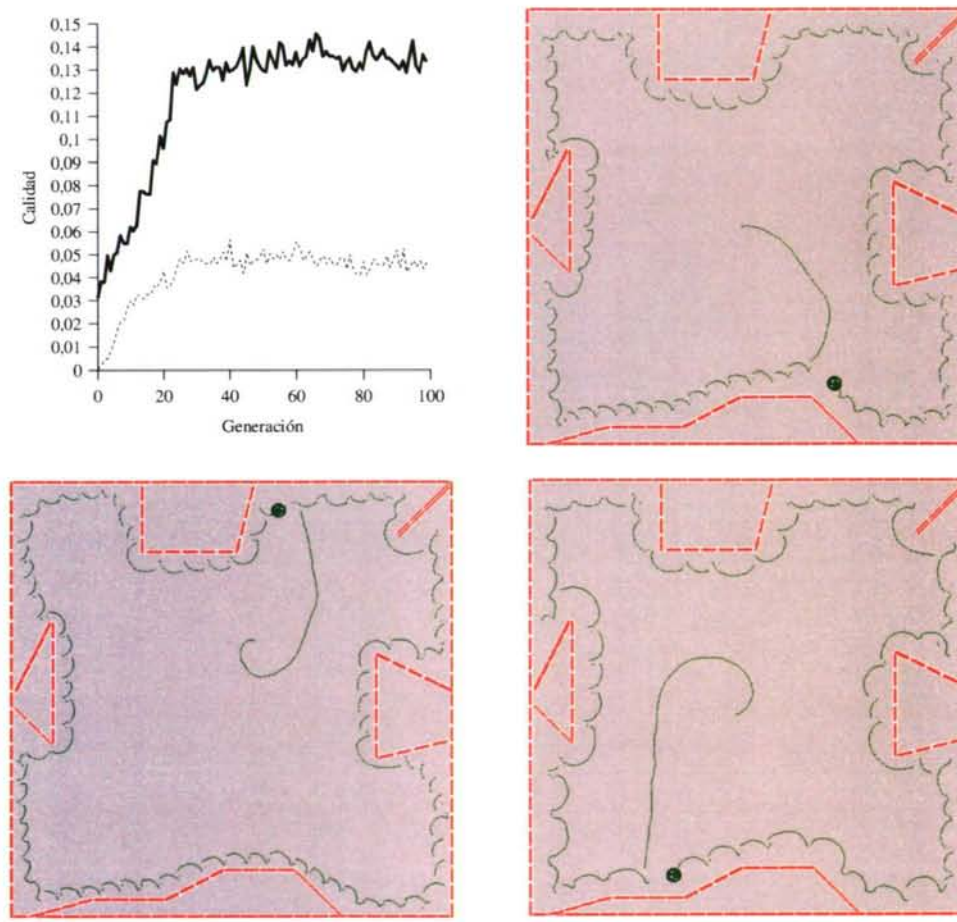


Figura 6: Comportamiento de búsqueda y seguimiento de paredes para el Rug Warrior usando neuronas de habituación y ruido sistemático. La gráfica en la parte superior izquierda representa la calidad del mejor individuo y la calidad media de la población en una evolución. La imagen superior derecha se corresponde a la trayectoria con los valores habituales en los parámetros a los cuales se aplica el ruido sistemático. En la parte inferior, en la imagen de la izquierda se ha cambiado la orientación del sensor de infrarrojos izquierdo y en la imagen de la derecha se ha cambiado el voltaje proporcionado por las baterías a 6.2V.

En la figura 5 observamos cómo el comportamiento obtenido anteriormente falla ante un ligero cambio en el ángulo de uno de los sensores (imagen izquierda) y ante una

batería con más carga de la normal (imagen derecha). En el primer caso, al variar el ángulo del sensor, el robot pierde en ocasiones la trayectoria y, en otras, incluso llega a colisionar al no darle tiempo a reaccionar. En el segundo caso, la carga excesiva de la batería supone que el robot se desplaza más en cada movimiento, lo que a veces le lleva también a colisionar con la pared. Introduciendo en la evolución ruido sistemático en la posición de los sensores, su alcance máximo, en la carga de las baterías y en la distancia recorrida por la rueda izquierda, obtenemos el comportamiento que se puede observar en la figura 6, en la cual apreciamos cómo el comportamiento ha ganado en robustez.

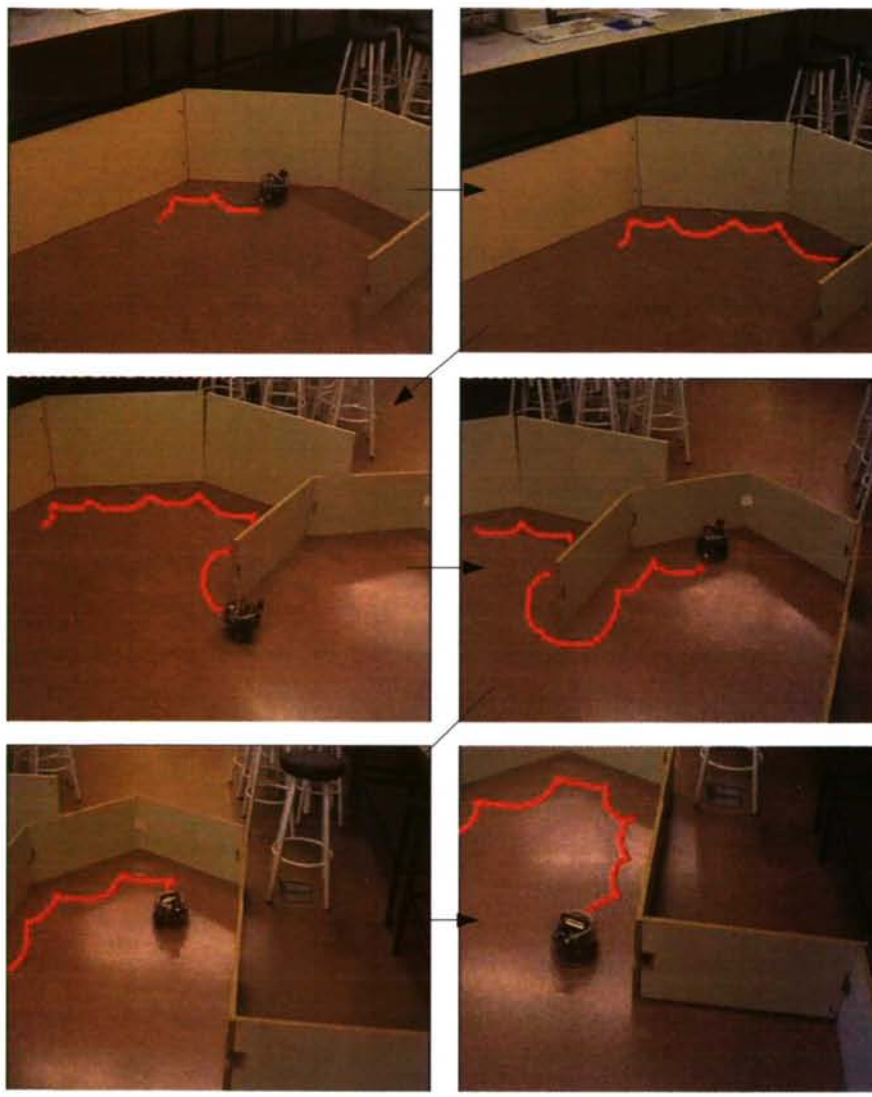


Figura 7: Comportamiento de búsqueda y seguimiento de paredes sobre el robot real (Rug Warrior) utilizando todos los tipos de ruidos vistos.

Para evitar las colisiones y generalizar su comportamiento a situaciones ambientales diversas, el robot se ve obligado a desempeñar estrategias conservadoras como, por ejemplo, a realizar un movimiento más oscilante. Esto se traduce en una menor calidad del mejor individuo respecto a la que se obtenía antes sin ruido sistemático, pero el comportamiento funciona ahora en un mayor rango de configuraciones del entorno. En la figura 7 podemos ver este comportamiento, obtenido utilizando todos estos tipos de ruido, sobre el robot real.

El ruido sistemático es, pues, imprescindible para hacer comportamientos realmente robustos, especialmente en robots de baja calidad como el Rug Warrior. En los apéndices están descritos todos los tipos de ruido sistemático utilizados en concreto para cada sensor/actuador.

4.4.2 Cálculo de la calidad

Independientemente del tipo de algoritmo evolutivo empleado y de dónde se realice la evaluación de los individuos, los procesos de selección para la reproducción y de individuos que pasan a la siguiente generación necesitan que a cada individuo se le asigne una calidad, es decir, una cifra indicativa de lo bien o mal que realizaría el robot real la tarea encomendada utilizando como controlador el indicado por el cromosoma correspondiente al individuo.

Existen varias alternativas posibles en el cálculo de la calidad, desde un punto de vista del momento en el cual se asigna ésta. Se puede utilizar una perspectiva local, asignando en cada movimiento una puntuación al robot en función de lo bien que ha realizado ese movimiento y tomando al final como su calidad la suma de las puntuaciones obtenidas en cada paso. El problema estriba en determinar la bondad de cada movimiento individualmente, ya que muchas veces dicha bondad depende de lo que ha realizado en el pasado o de lo que realice en el futuro. Pero hay otro problema más profundo: el asignar una calidad a cada movimiento individualmente presupone que el diseñador conoce la estrategia a seguir en cada instante por el robot para realizar la tarea encomendada. Dado que la percepción del entorno y, en general, la forma de interaccionar con él por parte del diseñador y del robot son distintas, el diseñador puede creer que una determinada estrategia es la más correcta y, en realidad, dicha estrategia puede ser inabordable para el robot por sus propias limitaciones o bien no ser la óptima. Este método es el tradicionalmente empleado por la mayoría de los investigadores que utilizan algoritmos evolutivos para obtener controladores para robots autónomos, y suele implicar sucesivos cambios en la función de calidad hasta que se obtiene el comportamiento deseado.

La otra posibilidad es utilizar una perspectiva global: al finalizar el periodo de evaluación se le asigna una calidad en función de su comportamiento global en el conjunto del periodo de evaluación. El problema aquí es, por una parte, cómo determinar dicha calidad global y, por otra, como no hay una valoración de los movimientos individuales, en numerosas ocasiones el proceso evolutivo explota "debilidades" en la definición de la función de calidad y produce como resultado controladores que optimizan la función de calidad pero no se comportan como realmente queremos. La calidad aquí puede ser asignada de forma externa o interna. En el primero de los casos, es un agente externo al robot el que fija el valor de la calidad, lo que permite hacer uso de variables que el robot desconoce. Esto puede ser útil en entornos estructurados, pero su utilidad es más dudosa en ambientes muy ruidosos o que varían con frecuencia. Un ejemplo extremo de asignación de la calidad de forma global y externa se da en el trabajo de Lund [69], donde la calidad de los controladores, ante la dificultad de encontrar una fórmula que la calculase, la determinan visualmente un grupo de niños para comportamientos sencillos con robots Lego. El problema presente en ese trabajo era, obviamente, el tiempo de evolución necesario, debido a la naturaleza de la evaluación. Cuando la calidad se asigna de forma interna, el robot es capaz de

juzgar él mismo su calidad mediante pistas que el diseñador pone en el entorno. Utilizando este método se puede establecer una analogía con el concepto de energía interna del individuo. El robot posee un determinado nivel de energía al inicio de su periodo de evaluación, y al relacionarse con su entorno dicho nivel de energía aumenta o disminuye. Así, sólo es necesario diseñar un conjunto de reglas que rijan dicha relación; por ejemplo, si el robot detecta un objeto con un determinado sensor incrementa su energía, en caso contrario la disminuye, etc.

Este último método, la aproximación energética, por ser el que menos intervención humana requiere, ha sido el empleado en este trabajo. Por tanto, a la hora de calcular la calidad en cada evaluación se utiliza un modelo energético: el individuo, a medida que se desenvuelve por el entorno, pierde o gana energía según sus acciones, y su calidad viene determinada únicamente por el nivel de energía al finalizar su periodo de evaluación. La descripción de cada una de estas acciones es muy básica: por ejemplo, sensar un objeto con un determinado sensor, colisionar con un objeto, etc. Esta descripción nunca contiene indicaciones de cómo realizar la tarea, es decir, no se utiliza la velocidad de las ruedas, ni la orientación del robot, ni la distancia a un objeto, ni nada que pueda forzar la evolución hacia la obtención de un comportamiento que cumpla unas determinadas características que el diseñador puede pensar necesarias o correctas, pero que a lo mejor no lo son. Hay que tener en cuenta que la percepción del entorno por el robot es muy diferente a la que tiene un ser humano, y una estrategia aparentemente correcta para el diseñador puede ser subóptima o irrealizable para el robot. Por ello, el diseñador debe dar el menor número de indicaciones posibles. Este modelo energético, de clara inspiración en los trabajos de Vida Artificial, facilita además la comprensión del problema al diseñador, ya que se establece una analogía entre el proceso evolutivo simulado y el ciclo de vida/evolución de una especie en la propia naturaleza.

Pongamos, por ejemplo, el comportamiento de seguir paredes para el Rug Warrior comentado en detalle en el apartado 5.1 y analizado en el apartado anterior bajo la perspectiva de su correcta transferencia al robot real. Si definimos la función de calidad de forma que ésta sea función del número de veces que el robot detecta la pared, éste evoluciona hasta dar lugar a un controlador que dirige al robot hacia una pared y luego lo detiene, haciendo que sense continuamente la misma parte de la pared.

Podemos, para corregir ese problema, introducir en la función de calidad las velocidades de desplazamiento de sus dos ruedas. Si la calidad se incrementa sólo cuando detecta paredes y de forma directamente proporcional a la mayor de las velocidades de sus ruedas, obtenemos un comportamiento como el que se puede observar en la parte izquierda de la figura 8: el robot se dirige a una pared y avanza lentamente girando sobre sí mismo a gran velocidad. Con este movimiento el robot evita las colisiones y está sensando en muchas ocasiones una pared (debido a la peculiar disposición de sus dos sensores de infrarrojos), maximizando la calidad porque lo hace con una de las dos ruedas desplazándose a gran velocidad.

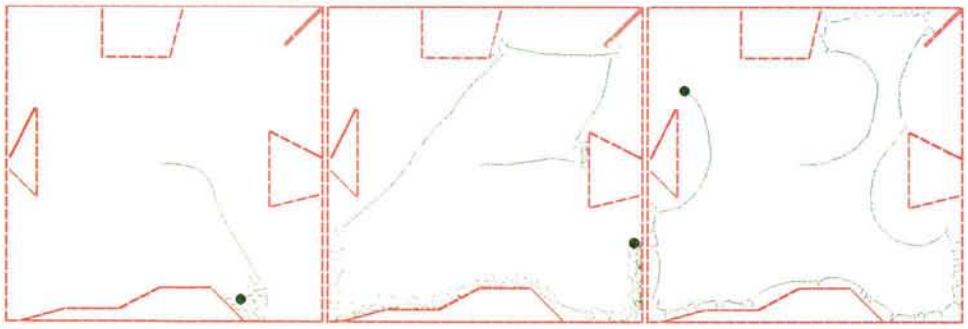


Figura 8: Comportamiento de seguir paredes para el robot Rug Warrior utilizando distintas funciones de calidad.

Si en vez de utilizar la mayor de las velocidades utilizamos la velocidad media en la función de calidad, para intentar así que el robot siga las paredes de la forma más recta posible, el comportamiento obtenido es el de la parte central de la figura 8: tras dirigirse a una pared el robot empieza a recorrerla, pero haciendo giros poco bruscos para mantener una velocidad media elevada. En la función de calidad le estamos premiando que sense las paredes el mayor número de veces, pero que cuando lo haga sea con una velocidad media alta en la suposición de que ambas cosas son compatibles. Pero no lo son. De hecho, la solución óptima a esa función de calidad implica que el robot no de prácticamente ningún giro cerrado pues, en caso contrario, aunque sensaría muchas veces la pared, lo haría con una velocidad media reducida. Le es más ventajoso dar giros más abiertos aunque eso implique aumentar el riesgo de abandonar la proximidad de la pared. Esto es así porque el Rug Warrior no conoce la distancia a la pared y no puede colisionar con ella, con lo que se ve obligado a realizar movimientos continuos de izquierda a derecha y viceversa y ese movimiento oscilante debe de ser muy pronunciado para poder abordar todas las curvas. Una forma de arreglar esto podría ser penalizar el tiempo que el robot está sin sensar una pared, pero esto puede dificultar enormemente la evolución. Si la penalización es muy elevada puede que no se llegue a obtener el comportamiento deseado, ya que los mejores individuos de las primeras generaciones lo único que hacen es evitar las colisiones y con esa función de calidad serían fuertemente penalizados, impidiendo así que éstos evolucionasen paulatinamente a seguir las paredes. Si la penalización es muy pequeña, perdemos el efecto que se buscaba con ella. Por ello, habría que ajustar muy cuidadosamente la penalización para obtener el comportamiento deseado, lo cual es complejo.

Finalmente, si en vez de la velocidad mayor o la media intentamos obtener el comportamiento utilizando la velocidad menor de ambas ruedas (parte derecha de la figura 8) nos encontramos con un problema muy similar al de la velocidad media: el robot no realiza curvas pronunciadas porque sensar paredes de esa forma apenas le proporciona un incremento de calidad debido a que una de las dos ruedas tiene una velocidad muy reducida.

Como acabamos de comprobar, es realmente difícil que el diseñador pueda imaginar cómo percibe el robot el entorno, por lo que cuantas más indicaciones sobre cómo realizar su labor se le den al robot más posibilidades tenemos de obtener comportamientos subóptimos desde el punto de vista de lo que queremos que el robot realice.

El modelo energético empleado en ese ejemplo (descrito en detalle en el apartado 5.1 y en el que, básicamente, el robot incrementa su energía al sensar "comida" dispuesta en las paredes que desaparece al ser percibida, obligando así al robot a moverse) carece de todos estos problemas y permite obtener la solución fácilmente. Es preciso hacer notar que, en ocasiones, el modelo energético aplicado no es directamente trasladable a una hipotética evolución en un entorno real con un robot real. Por ejemplo, en este comportamiento el Rug Warrior no puede discernir la "comida" con sus sensores de infrarrojos. Esto es un punto adicional a favor de la evolución en entorno simulado, que permite la utilización con facilidad de modelos energéticos, más o menos complejos, de difícil implementación en un entorno real.

Por último, es necesario hacer notar también que, si bien en muchas aplicaciones el cálculo de la calidad es totalmente determinista, es decir, a un individuo determinado le corresponde siempre la misma calidad, este no es el caso de la robótica evolutiva. La presencia de los elementos de carácter estocástico introducidos en la simulación o, si el proceso evolutivo tiene lugar en el robot real, a la imposibilidad de tener un control absoluto sobre todos los parámetros del entorno, provoca que el proceso de cálculo de la calidad para un individuo dado no sea totalmente determinista. Si no se tiene en cuenta esto, pueden darse oscilaciones en los valores de las calidades durante el proceso evolutivo que impidan la convergencia hacia una solución adecuada, ya que el mejor individuo en una generación determinada puede ser mediocre en la siguiente. Para tratar de solucionar esto cada individuo puede ser evaluado varias veces y su calidad final promediada. A mayor varianza entre las calidades en cada evaluación, será necesario un mayor número de evaluaciones para obtener un valor fiable. En el apartado 6.2.3 se analizan con detalle las implicaciones de una elección equivocada del número de evaluaciones.

4.4.3 Análisis de distintos tipos de algoritmos evolutivos en la obtención de controladores

Como se ha comentado en el apartado 4.2, existen numerosos tipos de algoritmos evolutivos distintos. En este apartado veremos un pequeño análisis de algunos de ellos aplicados a la obtención de controladores para robots autónomos. Primeramente compararemos los más habituales, algoritmos genéticos y estrategias evolutivas y, posteriormente, veremos con un ejemplo concreto la dificultad presente en este tipo de problemas analizando su espacio de búsqueda, así como el comportamiento de los algoritmos macroevolutivos, hasta ahora no utilizados en robótica evolutiva.

Algoritmos genéticos, estrategias evolutivas y algoritmos híbridos

En la figura 9 podemos ver gráficas resultado de promediar 10 evoluciones del comportamiento de alcanzar luz con el Rug Warrior (descrito en 5.1) cambiando dos parámetros: el tipo del algoritmo evolutivo empleado y la distribución inicial de la población en el espacio de búsqueda.

Los algoritmos evolutivos empleados son:

- Un algoritmo genético donde los genes son números en punto flotante, con corte en un sólo punto y mutación consistente en cambiar un gen por un valor aleatorio en el rango [-1:1].
- Una estrategia evolutiva en la que la mutación consiste en sumar un valor aleatorio obtenido calculando la función x^3 al resultado de escoger un número aleatorio en el rango [-1:1]. Con esta forma de mutación la mayor parte de los descendientes se encontrarán en las proximidades de los padres, mientras que algunos de ellos se situarán más alejados en el espacio de búsqueda.
- Un algoritmo mixto, en el cual la mitad de la veces, de promedio, se aplica la reproducción propia de un algoritmo genético (cruce + mutación) y la otra mitad de las veces se realiza una modificación de todos los genes del cromosoma con la fórmula aplicada en la estrategia evolutiva. Esto se hace con el objetivo de aumentar la fase de explotación durante toda la vida de la evolución, es decir, que no siempre se realicen cruces de individuos, sino que en ocasiones se realicen búsquedas en sus vecindades.

En todos ellos, el tamaño de la población es de 400 individuos y la selección es por torneo con una ventana de tamaño 20 (el mismo tipo de selección que será aplicado en todas las evoluciones de este trabajo que utilicen alguno de estos tres tipos de algoritmos evolutivos). La población se reemplaza completamente por sus descendientes al acabar cada generación, excepto el 3% de los mejores individuos que permanecen. Las distribuciones iniciales de la población que se han probado son:

- Distribución aleatoria. La habitual en la bibliografía: los genes de los individuos se generan aleatoriamente y se espera que los individuos queden repartidos de forma

más o menos equidistante en el espacio de búsqueda. El problema que puede surgir es que si el espacio de búsqueda es muy grande respecto al tamaño de la población, dicha equidistancia no se producirá y la densidad de individuos en zonas igual de grandes del espacio de búsqueda puede ser muy diferente.

- Distribución en *grid*. Es una de las dos distribuciones ideadas en este trabajo para probar alternativas a la distribución aleatoria. El espacio de búsqueda se representa como un hipercubo y los individuos se sitúan en las aristas de dicho hipercubo. Esta distribución funciona bien si al menos hay individuo por cada vértice.
- Distribución en hiper-diagonal. La otra distribución nueva, implementada con el algoritmo genético utilizado en este trabajo (genes en punto flotante y cruce por un punto) en mente. Generaliza el concepto de una diagonal, que une dos vértices opuestos de un cuadrado, a n dimensiones y tiene la característica de que genera mucho material genético diverso, lo cual es conveniente dada la codificación y el tipo de cruce de la implementación de algoritmo genético realizada, por cuanto el cruce no altera los valores de los genes y la mutación es un operador, en principio, con una frecuencia de actuación pequeña y bastante disruptivo con el fin de generar variedad.

De los efectos de variar la distribución inicial hablaremos más adelante, en el apartado 4.4.4.

En lo que al tipo de algoritmo evolutivo empleado se refiere, y para este comportamiento, podemos ver en las gráficas de calidad de la figura 9 que el algoritmo genético y el híbrido descrito anteriormente son los que mejor se comportan, mientras que la estrategia evolutiva (las gráficas a la izquierda) es la peor parada. Esto nos indica que en la superficie generada por el espacio de búsqueda los individuos con buena calidad están muy separados y el operador escogido para la estrategia evolutiva tiene muy difícil el moverse entre uno y otro. El algoritmo genético y el algoritmo híbrido alcanzan la misma solución, pero la evolución es un poco más rápida en el caso del algoritmo híbrido.

En la figura 10, similar a la anterior pero para el comportamiento de seguir paredes del Rug Warrior, vemos cómo es el algoritmo híbrido el que alcanza la mejor solución. En este caso, la estrategia evolutiva llega a una solución cercana a la del algoritmo híbrido si se deja un mayor número de generaciones. Este hecho, indica que la superficie generada por la función de calidad utilizada en este problema dispone de un gradiente que el operador de mutación utilizado en la estrategia evolutiva sí puede utilizar en este caso para ir buscando una mejor solución progresivamente, aunque en ocasiones sigue necesitando dar "saltos" hacia otras zonas del espacio de búsqueda alejadas, como demuestra el hecho de que en ocasiones la evolución se quede atascada temporalmente. El algoritmo genético utilizado avanza más rápidamente, pero no logra afinar tanto en la solución final por carecer de un operador que le permita realizar tal acción. El algoritmo híbrido, por contra, utiliza el cruce para moverse rápidamente entre

zonas alejadas del espacio de búsqueda y la mutación de la estrategia evolutiva para afinar la solución, llegando, en resumen, a una mejor solución más rápido que los otros algoritmos.

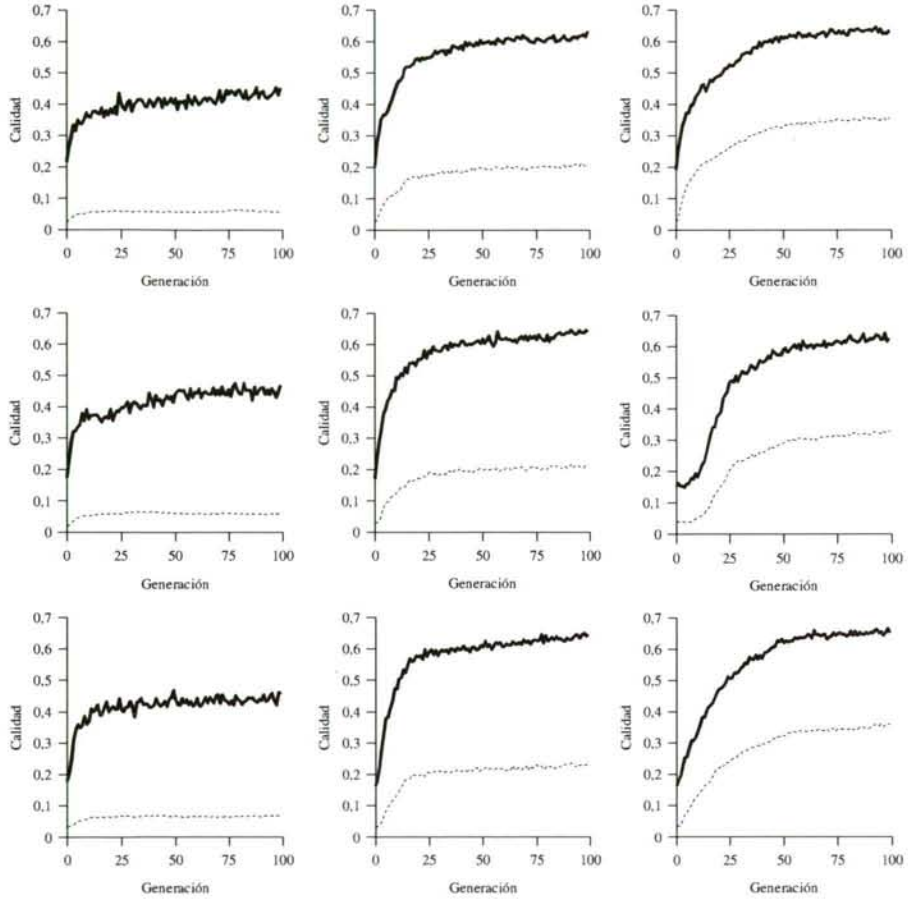


Figura 9: Media de las calidades del mejor individuo y calidad media de la población para 10 evoluciones del comportamiento de alcanzar luz para el Rug Warrior. De izquierda a derecha: estrategia evolutiva, algoritmo mixto y algoritmo genético. De arriba a abajo: distribución de la población inicial aleatoria, en grid y en hiper-diagonal.

La idea tras este sencillo algoritmo híbrido es, por tanto, aprovechar las ventajas de un algoritmo genético y de una estrategia evolutiva, evitando incluir en el cromosoma los parámetros que determinan la mutación, como se realiza en algunas implementaciones de estrategias evolutivas, lo cual aumentaría la longitud del cromosoma, hecho éste que no es deseable en nuestros problemas donde el ratio entre el tamaño de la población y la dimensionalidad del espacio de búsqueda ya es de por sí muy desfavorable. Así, el operador cruce del algoritmo genético se muestra como un

operador muy útil para la exploración, mientras que la forma de mutación ideada permite realizar una búsqueda eficaz en las inmediaciones de los individuos ya existentes (explotación).

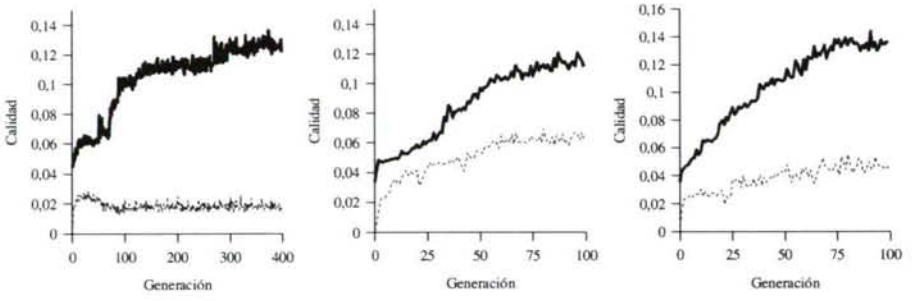


Figura 10: Comportamiento de seguir paredes para el Rug Warrior. De izquierda a derecha: estrategia evolutiva con distribución en grid, algoritmo genético con distribución en hiper-diagonal y algoritmo mixto también con distribución en hiper-diagonal.

Algoritmos macroevolutivos

En la evolución de los comportamientos obtenidos para el Rug Warrior en este trabajo se han utilizado algoritmos genéticos e híbridos entre algoritmos genéticos y estrategias evolutivas. Se ha visto que estos problemas de obtención de controladores son mucho más difíciles que los utilizados habitualmente en la bibliografía para comparar distintos tipos de algoritmos evolutivos, debido a una serie de características como el reducido ratio número de individuos/dimensión del espacio de búsqueda motivado por el elevado coste computacional del cálculo de calidad de los individuos, evaluación no determinista de dicha calidad, alta epístasis, etc. La dificultad es, en realidad, mucho mayor de lo que se podría pensar. A continuación se ilustrará esta problemática con más detalle usando un ejemplo de comportamiento para el Pioneer 2-DX. Este robot es de mucha mayor calidad que el Rug Warrior, tanto en sus sensores, como en sus actuadores y en su capacidad de procesamiento. Por ejemplo, en vez de 2 sensores de infrarrojos binarios, posee un anillo de 16 sónares que, por lo general, son bastante precisos. Pero este mayor número de sensores trae consigo un mucho mayor tamaño en la red neuronal necesaria para implementar un controlador. El mayor rango de valores posibles para los sensores hace también que la RNA tenga que considerar mayor precisión a la hora de tomar decisiones. El Pioneer 2-DX es también un robot más grande que el Rug Warrior, con una serie de inercias que no pueden ser despreciadas, lo que se traduce en que utilizar en dos ocasiones un mismo valor para un actuador puede traer consigo velocidades distintas en el instante siguiente dependiendo de la velocidad en el instante actual. Todo esto se traduce en RNAs más grandes, con más neuronas y más conexiones, y, por tanto, espacios de búsqueda que crecen exponencialmente, lo que dispara el número de individuos necesario para obtener una probabilidad razonable de alcanzar una solución si usamos los algoritmos evolutivos habituales.

En general, a mayor presión en un algoritmo evolutivo, más rápida es la convergencia a una solución, pero mayores son también las probabilidades de que dicha solución sea subóptima. Cuán grandes son esas probabilidades dependerá de muchos factores, como el número de individuos frente a la dimensionalidad del espacio de búsqueda, la forma de la función para la cual se está tratando de encontrar el óptimo, etc. Por eso, en cada campo de aplicación de los algoritmos evolutivos podemos encontrarnos con una combinación de parámetros ideal distinta, que nos lleven a diferentes formas de equilibrio entre la exploración (buscar zonas no exploradas en el espacio de búsqueda) y la explotación (buscar mejores individuos en las cercanías de los individuos buenos ya existentes). Cuanto más costosa en tiempo es la evaluación de un individuo, menos población podremos usar y, por tanto, más crítico se vuelve el encontrar ese punto de equilibrio.

Una posibilidad para tratar de solucionar este problema es concentrar la búsqueda alrededor de varios individuos buenos, y no de uno sólo. Una forma de conseguir esto consiste en la utilización de razas que veremos en el apartado 4.4.4. Otra forma es utilizar operadores genéticos que produzcan un efecto de agrupamiento alrededor de varias soluciones candidatas. En biología este fenómeno es conocido como "nitching". Goldberg [37] habla de nicho para referirse al rol desempeñado por un organismo en un entorno y de especie para referirse a aquellos organismos con características similares. Aplicado a la evolución artificial, este fenómeno significa el agrupamiento de los individuos alrededor de zonas determinadas del espacio de búsqueda, que serán aquellas que albergan posibles soluciones al problema. Una forma de lograr esto es utilizando el operador de "crowding" [24], mediante el cual cuando un individuo nuevo es generado reemplaza a aquel más similar de los ya existentes en la población. Menczer y col. [76] usan un método de selección local para mantener la diversidad en problemas multiobjetivo donde evolucionan RNAs. Por cada individuo se genera uno similar, éste es evaluado en un entorno que posee un nivel de recursos determinado que son consumidos por los individuos al evaluarse, y si el nivel de energía final del individuo es superior a un determinado umbral permanece en la población con la mitad de la energía, en caso contrario él y el padre son eliminados.

Como hemos visto cuando se describió el funcionamiento de los algoritmos macroevolutivos, éstos también producen un efecto similar, ya que las especies que se van generando a partir de las que se extinguen se van agrupando alrededor de las que sobreviven por la propia naturaleza del operador de colonización. Durante la realización de esta tesis se hicieron numerosas pruebas, al igual que con los otros algoritmos evolutivos, con los algoritmos macroevolutivos. De entre ellas, se ha seleccionado el siguiente ejemplo, el comportamiento de seguir paredes con el Pioneer 2-DX, que permitirá apreciar perfectamente el comportamiento de los algoritmos macroevolutivos en este tipo de problemas comparado con uno de los algoritmos evolutivos tradicionales: un algoritmo genético.

La implementación que se ha realizado en este trabajo de los algoritmos macroevolutivos incorpora una modificación motivada por la naturaleza no determinista del cálculo de calidad de los individuos. El funcionamiento de los macroevolutivos provoca que parte de la población, aquellas especies que sobreviven, pase de una generación a otra sin modificación alguna. En la implementación original, estos individuos, lógicamente, no se vuelven a reevaluar ya que su calidad es resultado de aplicar una función y no va a cambiar. Pero en estos problemas, como ya hemos visto, esa calidad sí puede cambiar de una evaluación a otra. Computacionalmente sería muy costoso reevaluar cada especie superviviente en cada generación, pero sí se puede, con un incremento muy pequeño en el coste computacional, mejorar la fiabilidad en el cálculo de la calidad de los individuos. Para ello, cada especie superviviente proveniente de la generación anterior se evalúa una única vez más y se actualiza su calidad media de forma ponderada. Se compara la nueva calidad para la especie con la que traía de la generación anterior y si ésta ha disminuido en más de un determinado valor (normalmente el 5%, aunque este valor es configurable) se incrementa el número de evaluaciones para las especies nuevas que se generen en la evolución a partir de ese momento. Ese incremento es en una cantidad igual al número de evaluaciones especificado para cada individuo al principio de la evolución. Por ejemplo, en una evolución que establezca el número de evaluaciones en 16, los individuos pueden ver incrementado ese número a 32, 48, 64... hasta que el cálculo de la calidad es fiable según el criterio establecido. Mediante el uso de este mecanismo de incremento automático del número de evaluaciones se puede detectar cuando dicho número establecido para la evolución es insuficiente para medir con fiabilidad la calidad de un individuo. En el apartado 6.2.3 veremos un ejemplo del funcionamiento de este mecanismo y de su importancia.

Antes de pasar a esa comparación entre algoritmos genéticos y macroevolutivos veremos un ejemplo de cuán difíciles pueden llegar a ser los problemas que abordamos. Para ello, hemos seleccionado el mejor individuo de todas las evoluciones y hemos visto cómo cambiaba su calidad alterando únicamente 4 de los 76 genes que conforman su cromosoma. Los 4 genes que van a ser recorridos para todos sus valores se corresponden con dos pesos de conexiones entre dos entradas y una neurona de la capa oculta y con otros dos pesos de conexiones que van desde esa misma neurona de la capa oculta a las dos neuronas de salida. En la figura 11 representamos 4 gráficas que se corresponden con 4 parejas distintas de valores para los genes modificados de las conexiones que van entre la neurona de la capa oculta y las 2 neuronas de salida. Los valores de los otros dos genes modificados son mostrados en todas las gráficas en los ejes x e y. Este análisis es muy similar al realizado por Janson y Frenzel [56], que mostraron porciones de la superficie de error para la evolución de una RNA variando únicamente un gen para demostrar la dificultad en la obtención de la solución óptima. En estas gráficas podemos apreciar varios hechos que demuestran la dificultad presente en este problema:

- Existen grandes áreas del espacio de soluciones en las cuales todos sus puntos tienen la misma calidad, con lo que el algoritmo evolutivo no tiene información útil en esas zonas que guíe la evolución.
- Existen puntos con elevada y similar calidad que están separados por zonas con mucha menor calidad, lo que es una doble dificultad para el algoritmo evolutivo: debe superar esas zonas con menor calidad para llegar a otras con alta calidad pero muy similar a aquella en la que ya se está, lo cual puede confundir al algoritmo evolutivo respecto a por dónde seguir.
- Si comparamos las dos gráficas superiores, vemos que variando únicamente dos genes podemos pasar de un máximo a una solución pésima, peor que las que la rodean que se corresponden con puntos que bajo ninguna circunstancia son máximos ni tan siquiera locales.

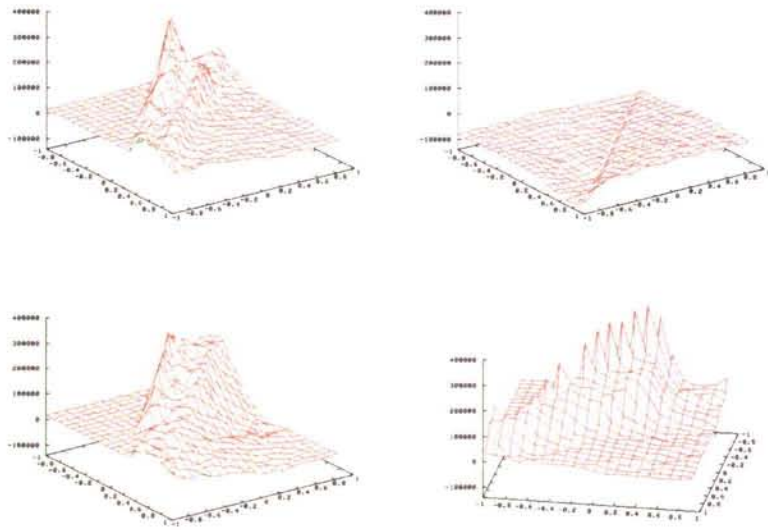


Figura 11: Muestra de un fragmento del espacio de búsqueda para el comportamiento de seguir paredes para el Pioneer 2-DX. Dicho espacio tiene 76 dimensiones. 72 están fijadas a los valores obtenidos para el mejor individuo. Dos dimensiones son recorridas para todo el rango de posibles valores en los ejes x e y de las gráficas y las otras dos dimensiones son recorridas en el tiempo, utilizando en ambos casos muestras cada 0.1. Las cuatro gráficas se corresponden a 4 parejas distintas de valores para esas dos dimensiones que son recorridas en el tiempo.

Ante esto, se puede pensar sobre la idoneidad de las funciones de calidad utilizadas. Por lo general, se debe tratar de que, a la hora de crear el modelo energético

en el entorno haya la mayor información posible para el robot acerca de la idoneidad de su comportamiento y que esa información exista en el mayor número posible de puntos. Pero será inevitable que en ocasiones el espacio de búsqueda sea complicado y, lo que es peor, por lo general no podremos preverlo. Pero eso puede suceder independientemente del mecanismo empleado para generar la función de calidad con lo que este problema, si bien se debe tener en cuenta, no debe hacernos considerar la posibilidad de rechazar la idoneidad del modelo energético cuyas ventajas en términos de una mayor libertad para el robot para desempeñar la tarea y de un menor análisis de la situación para el diseñador, son considerables.

Estamos, por tanto, ante un problema complejo donde la epístasis es enorme: un valor para un gen puede ser óptimo o pésimo dependiendo de los valores de otros genes. Nos interesa, por tanto, usar un algoritmo en el que haya una fase de exploración importante, al menos durante una parte de la evolución, para escapar de las zonas con calidad constante, pero, al mismo tiempo, con una fase de explotación que permita buscar meticulosamente las zonas alrededor de las mejores soluciones.

En la figura 12 podemos ver las características de la RNA evolucionada y de los algoritmos empleados en las distintas comparaciones. Para el caso del algoritmo macroevolutivo, el parámetro ρ valía 0.5 y τ variaba linealmente de 1 a 0 entre la generación 0 y la 1200, siendo permanentemente 0 desde esa generación. La RNA que forma el controlador tiene como entradas los 8 sónares frontales del Pioneer 2-DX.

Paramétros de la RNA	
Nº neuron. capa entrada (1)	8
Nº neuron. capa oculta (2)	6
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	-
Valor máximo de un peso	6

Parámetros del AE	
Tipo	GA/MA
Distribución inicial	Diagonal/Aleatoria
Prob. Cruce	0.8/-
Prob. Mutación	0.1/-
Razas	8
Poblaciones por raza	1
Generaciones	2000
Migración local	40
Migración global	80
Población	480/800/1600
Evaluac. por individuo	8
Pasos de vida	1000

Figura 12: Parámetros de la evolución del comportamiento de seguir paredes para el Pionner 2-DX.

En la gráfica izquierda de la figura 13 vemos las calidades del mejor individuo para 3 evoluciones (con 480, 800 y 1600 individuos) utilizando el algoritmo genético y otras 3 (también con 480, 800 y 1600 individuos) usando el macroevolutivo. Lo primero que observamos en esta gráfica es que, a igualdad de población el macroevolutivo se comporta mejor que el genético, excepto en el caso de 1600 individuos, población suficiente para que el genético también alcance la mejor solución. O dicho de otra manera, el macroevolutivo con la mitad de población iguala los resultados del genético.

El distinto comportamiento de ambos algoritmos también se aprecia claramente en la gráfica. El algoritmo genético evoluciona más rápidamente al principio debido a que los mejores individuos se reproducen mucho más que los restantes. Sin embargo, pronto se queda atascado al perder variedad genética y sólo poder adquirir ésta mediante las mutaciones. Si la evolución nos ha llevado a la solución correcta, perfecto, pero el riesgo de caer en máximos locales es elevado y mayor cuanto menor es la población. El macroevolutivo, por contra, realiza una evolución mucho más suave, dominada en un principio por una fase de exploración que poco a poco, a medida que τ va decreciendo, deja paso a la explotación. Por supuesto, sigue existiendo la posibilidad de caer en un máximo local, pero ésta es ahora menor por moverse el algoritmo evolutivo desde las zonas de menor calidad a las de mayor, no casi exclusivamente por las zonas de mayor calidad.

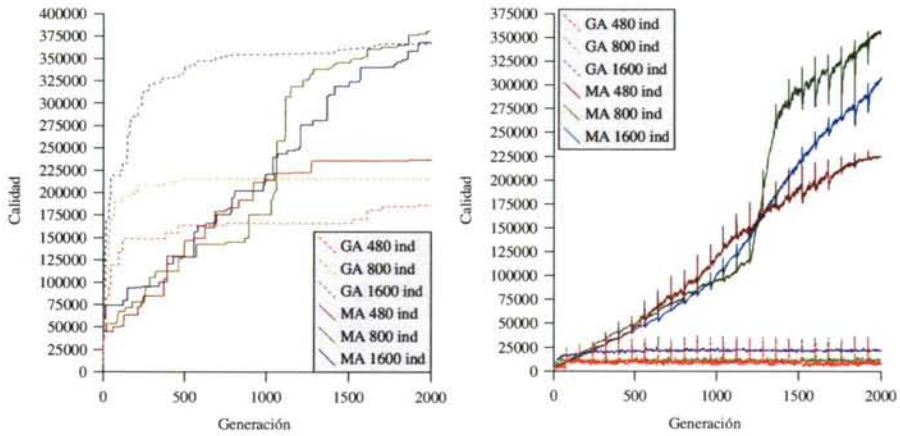


Figura 13: Calidad del mejor individuo (a la izquierda) y calidad media (a la derecha) para 6 evoluciones del comportamiento de seguir paredes para el Pioneer 2-DX.

En la gráfica derecha de la figura 13 también podemos ver la calidad media de la población para las 6 evoluciones anteriores. Se aprecia claramente la dificultad, que ya conocíamos, del problema, ya que los operadores de cruce y de mutación en el algoritmo genético generan en la mayor parte de las ocasiones individuos con baja calidad, llevando a calidades medias muy bajas. Se observa también cómo eso mismo no sucede en el caso del macroevolutivo, lo que indica, dado que la función que estamos optimizando es la misma, que los individuos se agrupan a medida que pasa el tiempo alrededor de los mejores individuos, o de lo contrario la calidad media no podría alcanzar esos valores.

Por tanto, el funcionamiento de los algoritmos macroevolutivos en este tipo de problemas con amplias superficies con valores de calidad similares, con picos esporádicos con mayor calidad y una elevada epístasis, se ha mostrado muy adecuado.

La evolución es más lenta en un principio, pero es más constante, menos propensa a caer en máximos locales y con un buen equilibrio entre exploración y explotación.

Ahora, usando este mismo ejemplo, veremos cómo afectan los parámetros ρ y τ al funcionamiento del macroevolutivo. Respecto a ρ, que define el entorno de vecindad alrededor de las especies que sobreviven cuando se generan especies nuevas a partir de las que se extinguen, las pruebas llevadas a cabo para diferentes valores de este parámetro no mostraron una clara tendencia para valores razonables entre 0.3 y 0.8. Por eso, al igual que Marín y Solé en [72], utilizamos 0.5 en las restantes pruebas. Los resultados de las pruebas con τ requieren un análisis detenido. En la figura 14 vemos la calidad del mejor individuo para 8 evoluciones con distintos valores de este parámetro.

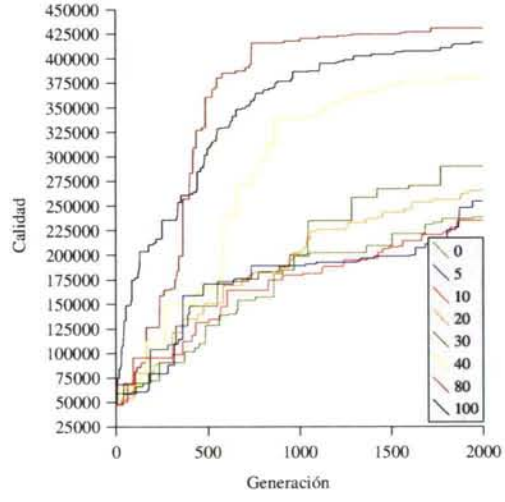


Figura 14: Calidad del mejor individuo para 8 evoluciones del comportamiento de seguir paredes para el Pioneer 2-DX cambiando el parámetro τ a través de NR (0, 5, 10, 20, 30, 40, 80 y 100).

Existen numerosas formas de variar el parámetro τ. En este trabajo se ha escogido hacerlo de forma lineal de 1 a 0 hasta una determinada generación, a partir de la cual valdrá siempre 0 (fase de explotación pura). A ese número de generaciones con τ=0 le hemos denominado NR (*non randomness*) y es el indicado en la leyenda de la figura 14. En dicha gráfica podemos observar cómo cuanto mayor es la fase de pura explotación, más rápida es la evolución y antes se llega a la solución. Sin embargo, es preciso hacer notar que también aumentan las posibilidades de caer en máximos locales pues estaremos haciendo menos exploración y podemos dejar fuera de la búsqueda zonas del espacio de soluciones no cubiertas por la población inicial. Por tanto, es útil dejar reservado un número de generaciones para explotación pura en problemas como éste donde los máximos están en zonas muy delimitadas y que deben ser exploradas con minuciosidad, pero no hacer ese valor exagerado pues aumentaremos el riesgo de caer

en máximos locales si la población es pequeña. De ahí, que se haya escogido un valor de NR = 40 para el resto de pruebas.

En este apartado hemos visto cómo el equilibrio entre explotación y exploración en el algoritmo evolutivo es fundamental en este tipo de problemas, con superficies en las funciones de calidad en ocasiones muy difíciles y con un ratio población / dimensionalidad del espacio de búsqueda muy desfavorable debido al elevado tiempo de cálculo necesario para evaluar la calidad de los individuos. Así, una estrategia mixta entre algoritmo genético y estrategia evolutiva parece aprovechar las ventajas de ambos. Y cuando la situación es todavía más complicada, como se ha mostrado en el caso de comportamientos para el Pioneer 2-DX, donde el mayor número de sensores impone también un mayor número de dimensiones en el espacio de búsqueda, los algoritmos macroevolutivos, hasta ahora no probados en problemas de este tipo, se muestran muy eficaces.

4.4.4 Tamaño y distribución de la población

En este apartado veremos que tanto el tamaño como la distribución de la población en el espacio de soluciones son dos factores a tener en cuenta para la obtención de resultados satisfactorios en las evoluciones, debido a la elevada dimensionalidad del espacio de búsqueda frente a tamaños de población que son relativamente reducidos por el elevado tiempo de procesado necesario para evaluar cada individuo.

Tamaño de la población

El tamaño de la población es un factor muy importante en un algoritmo evolutivo, y esa importancia se acentúa cuanto más complicado es el espacio de búsqueda en términos de gradiente disponible y de máximos locales. Desafortunadamente, en términos generales, no se puede garantizar, ni tampoco saber, cuando un algoritmo evolutivo va a converger a la mejor solución sin conocer el espacio de búsqueda, lo que, obviamente, no sucede en nuestro caso y, por tanto, tampoco podemos saber a priori cuando el tamaño de la población que está siendo empleada es suficiente. Pero de la ejecución de numerosas pruebas en este trabajo se extraen una serie de indicaciones que podemos seguir para percatarnos de cuándo el tamaño de la población es reducido:

- Si ejecutamos varias evoluciones de un mismo problema y la calidad de la mejor solución es muy dispar de una evolución a otra, es indicativo de que el tamaño de la población es escaso, y que sólo en ocasiones podemos estar encontrando el camino correcto en la evolución.
- Si la calidad del mejor individuo a lo largo de una evolución avanza en escalones esporádicos, es señal de que la superficie generada por la función de calidad en el espacio de búsqueda contiene áreas muy grandes donde la calidad es constante y que el problema es, por tanto, difícil. Aunque esto no es indicativo de si el tamaño de la población es suficiente o no, la dificultad del problema debe hacernos desconfiar y, probablemente, no estará de más hacer alguna evolución con un tamaño de población mayor para comprobar si la solución encontrada mejora o no.
- Si la población converge muy rápidamente, estaremos ante la presencia de uno o varios superindividuos. Éstos tanto pueden coincidir con la solución buscada como no hacerlo, lo cual significaría que el método de selección impone una presión evolutiva demasiado alta. Por eso también es conveniente repetir la evolución para comprobarlo. Si efectivamente esa es la solución y la convergencia se ha producido muy rápido, la población estaba claramente sobredimensionada o bien el problema es muy fácil.

Distribución inicial de la población

Respecto a la distribución de la población en el espacio de soluciones tenemos, a su vez, que tener en cuenta dos consideraciones independientes: la distribución de la población en el momento inicial, al empezar la evolución, y el hecho de si la creación de

subpoblaciones parcialmente aisladas unas de otras puede ayudar o no a explorar mejor el espacio de búsqueda y reducir el riesgo de que la evolución se vea dominada por un superindividuo.

Lo habitual en los algoritmos evolutivos es que los individuos de la población inicial sean aleatorios, es decir, que sus genes sean obtenidos aleatoriamente. Con esto, implícitamente, se está buscando que los individuos estén repartidos de forma homogénea en el espacio de búsqueda. Pero cuando el tamaño de la población es muy reducido comparado con la dimensionalidad de dicho espacio, pueden surgir problemas y no producirse el efecto deseado. En esos casos, puede ser beneficioso distribuir explícitamente los individuos en el espacio tratando de conseguir ese efecto de equidistancia entre los individuos. Otro motivo por el cual puede ser interesante distribuir la población siguiendo un algoritmo determinado es que así se pueden favorecer las propiedades de explotación concretas de cada algoritmo evolutivo. Por ejemplo, en el caso de un algoritmo genético donde los genes sean números reales y con cruce por un único punto, una distribución en hiperdiagonal (una diagonal generalizada a n dimensiones), combinada con una presión evolutiva baja, puede ser beneficiosa por cuanto disponemos en la población inicial de una mayor variedad de material genético.

En la figura 9 podemos ver cómo afectan una distribución en *grid*, que trata de distribuir los puntos equitativamente en el espacio de búsqueda, y una hiperdiagonal a los tres tipos de algoritmos evolutivos comentados en el apartado 4.4.3. La diferencia no es muy grande, pero hay una ligera ventaja a favor de la distribución en *grid* para la estrategia evolutiva (0.47 en la calidad máxima frente a 0.45 en los otros dos casos) y a favor de la hiperdiagonal para el algoritmo genético (0.65 frente 0.63 y 0.62). Es preciso hacer notar que estamos hablando de la media de 10 evoluciones y que el comportamiento es sencillo de obtener. Para problemas más complejos o con ratios todavía más desfavorables entre las dimensiones del espacio de búsqueda y el número de individuos, las diferencias podrán ser mayores.

Razas

Como último punto en este apartado, comprobaremos si la división de la población en subpoblaciones, limitando la interacción entre ellas con el fin de reducir los efectos perjudiciales de la aparición de superindividuos, es realmente beneficiosa o no.

Aunque la búsqueda de una solución por parte de un algoritmo evolutivo se realiza explorando múltiples puntos del espacio de soluciones simultáneamente, dado que los individuos con mayor calidad tienen más probabilidades de reproducirse, la población tenderá, con el paso de las generaciones, a converger alrededor de una serie de los mejores individuos y, probablemente, con el paso del tiempo, alrededor de un único individuo. Esta convergencia, obviamente deseable en cualquier algoritmo de búsqueda, es, sin embargo, un problema cuando se produce alrededor de uno o varios máximos locales, en vez de alrededor del máximo global. Una de las formas de tratar de

minimizar la probabilidad de que esto suceda es dividir el global de la población en subpoblaciones y limitar el intercambio de material genético entre ellas.

Los individuos de una raza sólo pueden reproducirse con individuos de su misma raza. El intercambio de material genético entre las razas se produce mediante la migración, proceso en el cual el mejor o mejores individuos de cada raza son copiados a otras razas. Existen dos tipos de migraciones: locales y globales. En las locales los mejores individuos sólo son copiados a las razas vecinas. Para ello, obviamente, hay que establecer una relación de vecindad entre las razas. En nuestro caso, se hace disponiéndolas sobre una circunferencia, de forma que cada raza tiene dos vecinas. En las migraciones globales los mejores individuos son copiados a todas las razas.

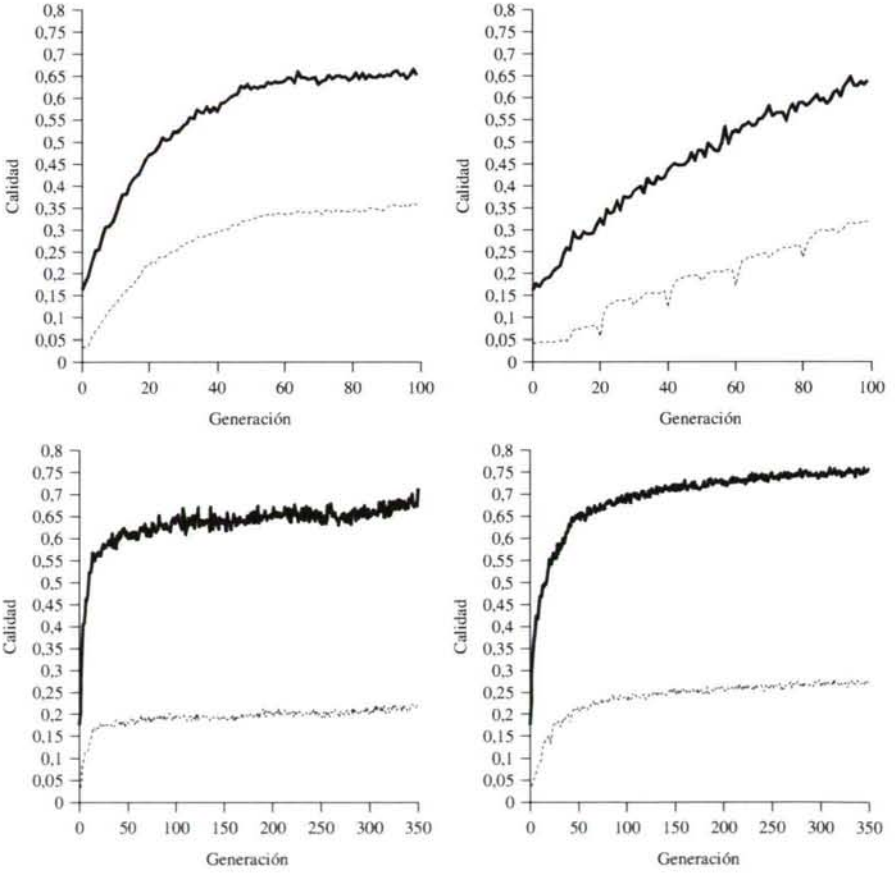


Figura 15: Media de las calidades del mejor individuo (línea gruesa) y calidad media (línea punteada) para 10 evoluciones del comportamiento de alcanzar luz para el Rug Warrior. Gráfica superior izquierda: 1 raza y 400 individuos. Gráfica superior derecha: 8 razas y 400 individuos. Gráfica inferior izquierda: 1 raza y 800 individuos. Gráfica inferior derecha: 8 razas y 800 individuos.

En la figura 15 se puede apreciar el efecto de las razas. La evolución va, en un principio, un poco más lenta, pero acaba superando el caso en el que no hay razas. Esto se debe a que los mejores individuos que pueden aparecer en una raza tardan en propagarse a otras razas, dándoles a éstas la oportunidad para que exploren zonas probablemente distintas del espacio de búsqueda. Si en un momento determinado aparece un buen individuo en una raza, sus probabilidades de reproducirse aumentarán en detrimento de las de los otros individuos. Dividiendo la población en razas, acotamos ese efecto a los individuos de la raza en la que apareció el individuo mientras no se produzcan migraciones. Las migraciones son necesarias, ya que si no tenemos poblaciones independientes que explorarán el mismo espacio de búsqueda y no es eso lo que queremos. Simplemente se busca bajar la presión evolutiva, acotando temporalmente el área de influencia de los mejores individuos. Otra cosa que se aprecia en la figura 15 es que cuanta menor es la población más lenta va la evolución al tener razas de muy pocos individuos que exploran poco eficientemente el espacio.

Ahora veremos cómo afectan las razas a los algoritmos macroevolutivos, para ver si se mantienen estas ventajas. Para ello usaremos el ejemplo visto en el apartado 4.4.3 de seguimiento de paredes para el Pioneer 2-DX utilizando poblaciones de 1600 individuos.

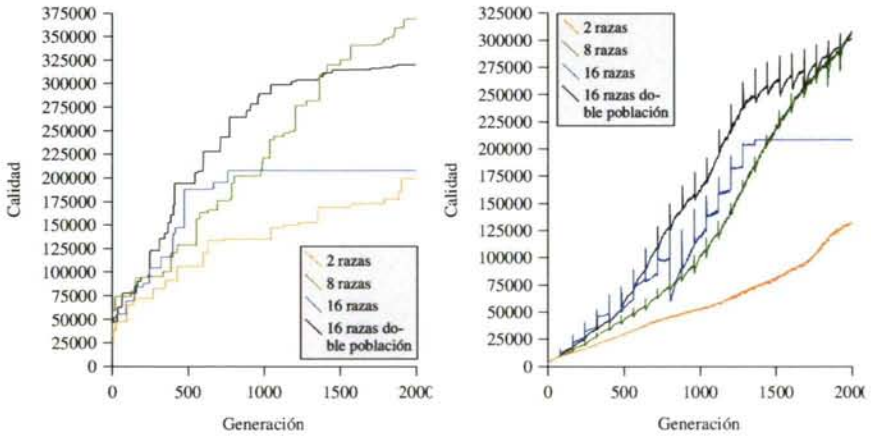


Figura 16: Calidades máxima y media del comportamiento de seguir paredes para el Pioneer 2-DX cambiando el número de razas.

En la figura 16 se aprecia claramente que la evolución mejora al aumentar el número de razas de forma muy notable. Esto es cierto hasta llegado un punto en el cual las razas tienen tan pocos individuos que no pueden realizar una búsqueda eficiente, como se puede observar en el caso de 16 razas con 1600 individuos, donde cada raza tiene únicamente 100 individuos. En la gráfica de las calidades medias observamos un efecto curioso: en ocasiones la calidad media sube para bajar nuevamente a

continuación. Estos incrementos temporales en la calidad media se corresponden con migraciones cuando la población se encuentra fuertemente agrupada en torno a buenos individuos distintos en cada raza. De esta forma, cuando un buen individuo llega proveniente de otra raza sustituyendo al peor de la raza de destino, la calidad media aumenta, pero dada la forma de la función de calidad donde, recordemos, tenemos grandes zonas con baja calidad y esporádicos picos con buena calidad, los descendientes de la especies que abandonan la población tienen en la mayor parte de las ocasiones mala calidad, y por eso la calidad media baja tan abruptamente.

Como vemos, los algoritmos macroevolutivos se benefician de la misma manera, si no más, que los algoritmos genéticos con la utilización de razas, lo que será un factor más a tener en cuenta a la hora de seleccionar el algoritmo evolutivo ideal para obtener un comportamiento.

4.4.5 Distribución y paralelización del algoritmo

Como se ha visto, el tamaño de la población es un factor crítico a la hora de obtener soluciones satisfactorias en un algoritmo evolutivo. En el caso que nos ocupa, esta necesidad se acentúa debido a dos razones:

- La alta dimensionalidad del espacio de búsqueda.
- La forma de dicho espacio, con múltiples máximos y grandes áreas de calidad constante que no aportan información alguna al algoritmo evolutivo.

Obviamente, a mayor tamaño de población, mayor el tiempo necesario para obtener una solución. Esta aseveración es todavía más preocupante en la robótica evolutiva, donde el tiempo de evaluación de cada individuo es elevado comparado con el resto de etapas del algoritmo. Hay que tener en cuenta que cada individuo de la población es evaluado tras haberse desenvuelto en un entorno durante un determinado intervalo de tiempo. Aun en el caso de que el entorno sea simulado y, por tanto, el robot se mueva a mayor velocidad que en la realidad, cada paso del robot en ese entorno implica numerosos cálculos para reflejar la dinámica de dicho entorno: nuevos valores de los sensores y de los actuadores para cada robot, movimiento de los robots, tratamiento de las colisiones, etc. Además, dado que la simulación del entorno no es perfecta, cada individuo debe ser evaluado varias veces y su calidad promediada.

Para aumentar la velocidad en la obtención de la solución se puede recurrir a la paralelización del algoritmo evolutivo. El elevado tiempo de evaluación de cada individuo favorecerá además que ésta sea eficiente.

Los algoritmos evolutivos paralelos pueden clasificarse de acuerdo al grano de cómputo utilizado [4][16][101]. En una primera posibilidad, denominada "algoritmos genéticos de micrograno (*mgGA*)" por Stracuzzi [101] y "paralelización global" por Cantú-Paz [16], tenemos un proceso maestro y varios procesos esclavos. El proceso maestro se encarga de enviar los individuos a cada uno de los procesos esclavos, los cuales evalúan los individuos, recibir los resultados de dicha evaluación y realizar todos los demás pasos del algoritmo evolutivo. Los *mgGA* tienen el problema de obligar a todos los procesos a sincronizarse frecuentemente, con lo que no se adaptan adecuadamente a entornos distribuidos donde no todos los procesadores involucrados tienen la misma potencia computacional. Además, en muchos casos, no todas las evaluaciones de cada individuo tienen la misma duración. Este problema es mayor cuanto mayor es el tiempo de evaluación. Un último problema es el elevado tráfico que se produce desde y hacia el proceso maestro, que puede convertirse en un cuello de botella. Este otro problema, al contrario que el anterior, es mayor cuanto menor es el tiempo de evaluación.

El segundo tipo de algoritmos evolutivos paralelos se conoce como "algoritmos genéticos de grano fino (*fgGA*)" [9]. En éstos, la población total se divide en varias subpoblaciones. Para ello, los individuos se disponen en una red n-dimensional, donde

habitualmente n=2, y las fronteras entre las poblaciones se diseñan de forma que se solapen algunos individuos, que pertenecerán así a más de una población. La reproducción sólo puede producirse entre individuos de una misma población, pero, al haber individuos que forman parte de más de una, se produce una migración implícita de individuos entre diferentes subpoblaciones en cada generación. Nuevamente, las comunicaciones pueden resultar muy elevadas dependiendo del tamaño de las poblaciones y el grado de solapamiento entre ellas, ya que todas las razas transmiten/reciben en cada generación los individuos correspondientes a las zonas de solape con sus razas colindantes.

Finalmente, un tercer tipo de algoritmos evolutivos paralelos son los "algoritmos genéticos de grano grueso (*cgGA*)". Aquí también se divide la población total en varias poblaciones pero, al contrario que en la solución anterior, ahora las fronteras están bien delimitadas. El intercambio de material genético entre poblaciones se produce mediante migraciones explícitas, que no se suelen producir en todas las generaciones, sino más esporádicamente. Por tanto, el tráfico de los *cgGA* es potencialmente mucho menor que en los casos anteriores. Dentro de los *cgGA* podemos diferenciar tres tipos de modelos:

- Islas aisladas. No se produce migración en absoluto.
- Islas síncronas [16]. La migración se produce siempre en las mismas generaciones en todas las poblaciones, es decir, éstas se sincronizan en el momento de realizar la migración del mejor o mejores individuos, ya sea a todas las demás islas o únicamente entre islas vecinas.
- Islas asíncronas [39]. No se necesita esa sincronización, ya que en el proceso de migración no se requiere que las islas implicadas estén en el mismo número de generación, con lo que es más adecuado para entornos distribuidos con procesadores heterogéneos.

Por otra parte, e independientemente de la clasificación anterior, se conocen como algoritmos genéticos multinivel (*nGA*) a aquellos que evolucionan no sólo las poblaciones que codifican posibles soluciones a un problema, sino también el propio *GA*, es decir, los parámetros que determinan la evolución. Este esquema de dos niveles se puede extender tanto como se quiera, siendo su mayor inconveniente el elevado coste computacional.

La paralelización en SEVEN[3] se ha realizado siguiendo un esquema propio de dos niveles y que puede considerarse un híbrido entre los modelos de *mgGA* y *cgGA* con islas asíncronas.

En un primer nivel, la población total se divide en razas o islas, siguiendo el modelo explicado previamente. Cada raza evoluciona independientemente de las otras y no existe, por tanto, ningún tipo de sincronización entre ellas, lo que favorece el

3 Existen dos versiones, una utilizando hilos *posix* para máquinas con arquitectura de memoria compartida y otra utilizando MPI (*Message Passing Interface*) [77] para máquinas con arquitectura de memoria distribuida y para *clusters*.

rendimiento de la paralelización, aun en el caso de que los procesadores que llevan a cabo la evolución tengan distinta capacidad computacional. El único intercambio de información entre las razas se produce a la hora de realizar las migraciones. Para evitar que una raza que lleva más generaciones que otra dirija la evolución de esa otra raza hacia la zona del espacio de búsqueda donde se encuentran sus mejores individuos, se impide que una raza introduzca en su población individuos de otra raza de una generación mayor. Cada raza, al acabar una generación, envía su mejor o mejores individuos a las demás razas, las cuales van almacenando esa información en una matriz destinada al respecto. Cuando es la hora de realizar la migración, cada raza selecciona dentro de esa matriz los mejores individuos de las otras razas que verifican que el número de generación en la cual fueron obtenidos es menor o igual al número de generación actual de la raza que va a tomar los individuos. Es decir, en el proceso de migración, una raza no va a utilizar individuos provenientes de generaciones posteriores de otras razas.

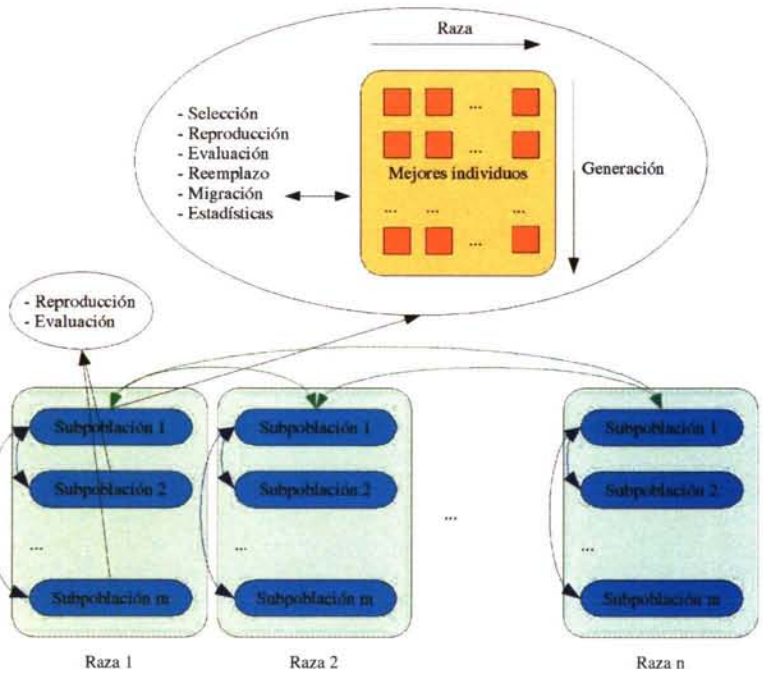


Figura 17: Modelo seguido en la paralelización.

El otro nivel de paralelización, de menor granularidad, se ha implementado dividiendo a su vez las razas en subpoblaciones, de forma que cada subpoblación puede evolucionar en un procesador diferente. Las subpoblaciones deben sincronizarse al finalizar la evaluación de sus individuos ya que, una vez finalizada dicha evaluación, una de las subpoblaciones (se ha seleccionado la primera para ello) debe realizar una serie de tareas que no pueden ser paralelizadas, como es el caso de la selección, la

migración, etc. Para estas tareas, dicha subpoblación necesita conocer todos los individuos de la raza, por lo que todas las subpoblaciones de su misma raza le enviarán los individuos en el momento en el que acaben de evaluarlos. Ella, a su vez, les enviará los nuevos individuos una vez se ha producido la selección. Estas labores adicionales de la primera subpoblación respecto a las otras son despreciables dada la diferencia en tiempo de computación de la fase de evaluación respecto a las demás. Sin embargo, la necesidad de sincronizarse en cada generación, hace que este nivel de paralelización sea, obviamente, mucho menos eficaz que el anterior y se debe tratar de que todos los procesadores en los que se alojan las subpoblaciones para una raza dada tengan la misma capacidad computacional. Pese a ello, este nivel de paralelización es necesario, ya que, en ocasiones, tendremos más procesadores disponibles que razas deseadas.

Los parámetros de número de razas, poblaciones en cada raza, frecuencia de la migración local y frecuencia de la migración global son configurables en el entorno. Si establecemos el número de razas en 1, este modelo es equivalente a un *mgGA*. Si el número de poblaciones por raza es 1 estamos ante un *cgGA* de islas asíncronas. Si anulamos la migración global y establecemos la frecuencia de migración local a 1, el esquema es similar a un *fgGA*.

Para comprobar la eficacia de estas paralelizaciones se han llevado a cabo una serie de pruebas usando el Rug Warrior y el comportamiento de seguir paredes. Los siguientes parámetros de la evolución han sido idénticos en todas las pruebas realizadas: híbrido de algoritmo genético y estrategia evolutiva, selección por torneo, probabilidad de cruce $p_c = 0.5$ (en caso de no producirse el cruce todos los genes del cromosoma mutan sumando a cada uno x^3, donde x es un número aleatorio en el intervalo [-1:1]), reemplazo de todos los padres por los hijos excepto el 3% de los mejores padres que se conservan, población de 1000 individuos, 1000 pasos de vida en el entorno simulado por cada evaluación de un individuo, 16 evaluaciones por individuo y 100 generaciones de duración de la evolución. Las evoluciones que implicaban 10 o menos procesadores fueron llevadas a cabo en un *cluster* de 10 procesadores AMD Athlon a 500 Mhz. Aquellas que implicaban más procesadores se llevaron a cabo usando dos *clusters* en localizaciones físicas distintas. El primero es el ya comentado y el segundo uno formado por 8 biprocesadores Intel Pentium III a 550 Mhz.

Razas	Comunic.	Evaluación	Intercambio	Migración	Reproducción	Selección	Total	% comunic.
1	0,01639	113398,50370	0,19710	0,00005	1,47790	7,33344	113407,52858	0,00145%
5	0,05257	20633,80706	0,03210	0,00009	0,29224	0,32492	20634,50897	0,02548%
10	0,06776	9521,34201	0,01577	0,00017	0,14280	0,09119	9521,65970	0,07117%

Tabla 1: Tiempos (en segundos) de las distintas fases en la evolución. El número de procesadores es igual al número de razas.

En la tabla 1 se pueden ver los tiempos de las distintas fases de la evolución con diferente número de razas. En todos los casos hay un procesador por cada raza, con lo que sólo se está teniendo en cuenta el primer nivel de distribución, de grano grueso. A

medida que aumentamos el número de razas aumenta ligeramente el tiempo de comunicación, debido a que la migración global implica más razas (cada raza recibe individuos de más razas y envía sus mejores individuos a más razas). Este aumento es, sin embargo, insignificante, como se puede apreciar, y el rendimiento se incrementa prácticamente de modo lineal.

En la figura 18 se analiza únicamente el segundo nivel de distribución. En ellas se muestra el *speedup* para el caso de 1 y 5 razas a medida que se aumenta el número de procesadores manteniendo fijo el número de razas, es decir, a medida que se aumenta el número de subpoblaciones por raza. La necesidad, en este caso, de sincronización entre las subpoblaciones de una misma raza rebaja el *speedup* respecto al óptimo a medida que se incrementa el número de éstas. En la figura 19 podemos ver también esta pérdida de eficiencia del algoritmo en dicha situación. En esa gráfica, vemos cómo, para un número fijo de 10 procesadores, al aumentar el número de razas y, por tanto, disminuir el número de subpoblaciones por raza, aumenta la eficiencia del algoritmo, que se ha definido como el tiempo necesario para obtener una solución con 10 razas dividido entre el tiempo necesario para obtener la solución con x razas, donde x va desde 1 hasta 10.

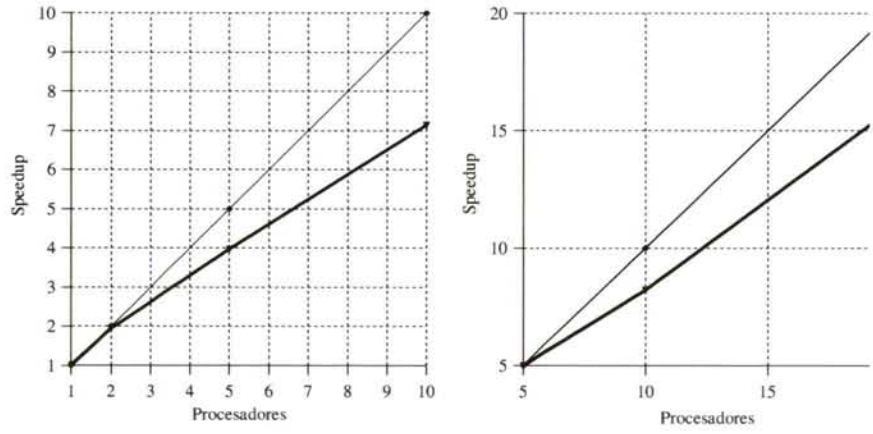


Figura 18: Speedup para 1 raza (izquierda) y 5 razas (derecha). La línea delgada representa el ideal y la gruesa representa el real.

A la vista de estos resultados uno podría pensar que lo ideal es usar siempre tantas razas como procesadores hay disponibles. Y desde un punto de vista de eficiencia computacional eso es así. Sin embargo, hay que recordar que el algoritmo evolutivo tiene sus propias restricciones en cuanto al tamaño de la población de cada raza. Así, como se ha visto, no es conveniente, por lo general y para este tipo de problemas, tener razas con menos de 200 individuos. De ahí que la división de razas en subpoblaciones sea interesante cuando disponemos de más procesadores que razas queremos emplear.

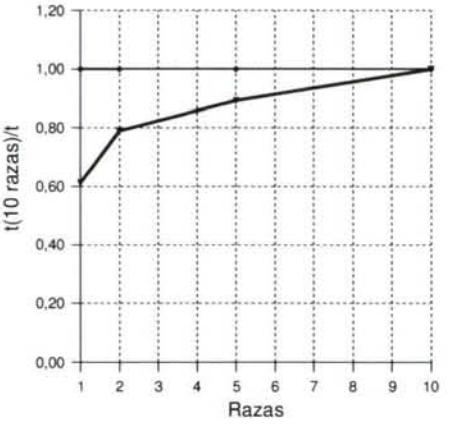


Figura 19: Degradación del rendimiento en función del número de subpoblaciones (la línea delgada representa el caso ideal y la gruesa el real).

Otra consideración necesaria es que este esquema de parelelización se ha realizado teniendo en cuenta los tiempos de evaluación de cada individuo (el ejemplo utilizado en las pruebas es uno de los que tienen un mayor tiempo de evaluación) y el hecho de que los *clusters* de que se disponían estaban en lugares geográficos dispersos. Si el tiempo de evaluación de los individuos fuese todavía mayor y siempre el mismo y el *cluster* fuese homogéneo, o bien los procesadores estuviesen todos conectados por una red de muy alta velocidad y baja latencia, la paralelización más adecuada probablemente fuese un *mgGA*.

4.4.6 Conclusiones sobre el proceso de obtención automática de controladores

En este apartado hemos analizado las implicaciones y problemas que surgen de las decisiones tomadas respecto a los elementos que componen la metodología desarrollada para obtener controladores, la cual está basada fundamentalmente en la evolución en entornos simulados de RNAs. También hemos visto cómo abordar y solucionar dichos problemas.

Realizar las evoluciones para obtener los controladores en entornos reales tiene una penalización tal en términos de tiempo necesario para comportamientos no triviales que se hace casi obligada la utilización de entornos simulados. Esta utilización de entornos simulados obliga a tomar una serie de medidas para asegurar que los controladores obtenidos en ellos funcionan correctamente en entornos reales. Esas medidas se resumen, fundamentalmente, en utilizar los tipos de ruido adecuados. Entre éstos está el habitual ruido consistente en pequeñas alteraciones en la simulación de los valores sensados y en las actuaciones del robot. Sin embargo, ese ruido hemos visto que no es suficiente para hacer los controladores tolerantes a cambios esporádicos pero permanentes en el entorno, tales como anomalías en el funcionamiento del robot motivadas por accidentes, cambios en la carga de las baterías, etc. Para ello es necesario utilizar un nuevo tipo de ruido, que se ha denominado sistemático, que permite obtener controladores tolerantes a esos cambios y, por tanto, más robustos.

Existen numerosas formas de plantear el cálculo de la calidad en los procesos evolutivos para asignársela a los individuos, pero hemos visto que un planteamiento energético del problema, haciendo que las interacciones del robot con el entorno se traduzcan en pérdidas o ganancias en un nivel de energía interno, minimiza el papel del diseñador en el proceso así como los efectos perjudiciales que su interpretación subjetiva del problema pueden tener sobre el proceso evolutivo debido a una función de calidad que refleje de forma indirecta su percepción de lo que debe realizar el robot para solucionar el problema.

Se ha realizado una comparación de diversos algoritmos evolutivos a la hora de obtener controladores con esta metodología y se ha visto en dicha comparación la importancia de un adecuado equilibrio entre exploración y explotación. Dada la dificultad del problema, motivada por una gran dimensionalidad frente a un reducido tamaño de población (por el alto coste en términos computacionales de la evaluación de los individuos), debe haber una fase de exploración inicial importante y la presión evolutiva no puede ser muy fuerte. Debido a la superficie en el espacio de búsqueda que presentan algunas de las funciones de calidad usadas, con grandes áreas de idéntica calidad y esporádicos picos donde ésta es mejor, la fase de explotación, al menos al final de la evolución, tiene también una gran importancia y debe de tratar de buscar en el espacio de soluciones de forma exhaustiva alrededor de los mejores individuos. Todas estas características traen como consecuencia que los algoritmos

macroevolutivos, hasta ahora no empleados para la obtención de controladores para robots autónomos, sean candidatos idóneos para su utilización en evoluciones de controladores en esta metodología y, en general, en problemas con características similares.

También hemos visto cómo, por las razones ya comentadas de alta dimensionalidad frente a escasa población, la distribución de la población en el espacio de búsqueda es un factor importante, tanto en lo que se refiere a la población inicial, como a la división de la población global en subpoblaciones que evolucionan casi independientemente (razas). Algunos algoritmos evolutivos han mostrado un mejor comportamiento con algunas distribuciones en particular, caso de los algoritmos genéticos aquí planteados con la distribución en hiperdiagonal, y el uso de razas se ha mostrado muy beneficioso siempre y cuando las razas tengan un tamaño mínimo de población suficiente.

Finalmente, la utilización de razas y la necesidad de acelerar el proceso evolutivo ha desembocado en una implementación muy eficaz del entorno de evolución implementado para la realización de los experimentos en este trabajo (SEVEN), consiguiendo *speedups* casi lineales bajo determinadas condiciones.

5 Implementación de los bloques constructivos

Como se ha comentado en el capítulo anterior, las redes de neuronas artificiales son el método fundamental que se ha utilizado en este trabajo para implementar los módulos que forman los controladores. En este capítulo veremos mecanismos adicionales no utilizados habitualmente en robótica y que nos ayudarán a aumentar las posibilidades de los controladores.

Las RNAs más habituales en robótica son aquellas formadas por varias capas donde los elementos de procesamiento (neuronas) de cada una se conectan con todos los de la siguiente. Esas conexiones contienen un valor, peso, que multiplica la salida de la neurona de origen y es el resultado de esa multiplicación el valor que recibe la neurona de destino. Cada neurona tiene una función de activación que suele trabajar sobre la suma de todos los valores que le llegan. Esa función de activación suele ser una sigmoide, una tangente hiperbólica o una simple recta. En este trabajo se han explorado algunas modificaciones en este modelo, con el fin de:

- Reducir el tiempo de procesamiento necesario para el controlador. El tiempo de cómputo de un módulo, desde que recibe los datos hasta que proporciona su salida, es fundamental para que el robot pueda reaccionar a tiempo y evitar situaciones comprometidas para su integridad física o la de los elementos que le rodean. Si además el robot tiene capacidades de procesamiento reducidas, cualquier factor que ayude a reducir el tamaño de los módulos será bienvenido.
- Incorporar la capacidad de procesar información temporal a la red. Como veremos, hay ciertos comportamientos que para poder ser realizados necesitan inevitablemente de la capacidad de procesar información temporal por parte de los controladores que los soportan. Además, también veremos que añadir esta capacidad a los módulos redundará en un comportamiento más eficaz en algunos casos en los que la información temporal no es imprescindible y cómo puede ayudar a suplir situaciones de infrasensorización, independientemente de que dicha infrasensorización se deba a la propia naturaleza del robot o venga dada por una pérdida accidental de sus capacidades sensoriales.
- Introducir mecanismos de atención y podado automático en la red. En este trabajo se ha seguido una representación directa del fenotipo en el genotipo y no se ha proporcionado al algoritmo evolutivo la capacidad de desarrollo, es decir, de usar cromosomas de longitud variable que codifiquen ese proceso de desarrollo que permite obtener redes del tamaño mínimo necesario [78]. Sin embargo, veremos que es posible obtener ventajas similares de forma indirecta, introduciendo en las RNAs estructuras que proporcionen una ventaja evolutiva cuando el tamaño de la red se reduce.

Así, en este capítulo veremos cómo la introducción de neuronas de habituación y de retardos en las conexiones sinápticas permitirá dotar a las RNAs de la capacidad de procesar información temporal e, incluso, de reducir su tamaño necesario en algunos

casos. El de uso sinapsis gausianas dotará a las RNAs de un mecanismo de atención y a los algoritmos evolutivos de un método indirecto para realizar podado automático de RNAs.

5.1 Procesado de información temporal: retardos y neuronas de habituación

La inclusión en una red de neuronas artificiales de la capacidad de procesar información temporal puede ayudar a mejorar el comportamiento del robot en términos de acercarlo a lo que esperamos de él. Por ejemplo, si los datos provenientes de los sensores poseen una gran cantidad de ruido, considerar información de instantes anteriores puede ayudar a reducir los efectos de éste. También puede permitir al robot realizar predicciones sobre lo que sensará a continuación, permitiéndole tomar mejores decisiones. Santos y Duro presentan un ejemplo [99] donde un robot predice la trayectoria de un objetivo móvil y lo intercepta en el mínimo tiempo posible. En ocasiones, procesar información temporal es incluso la única forma de implementar el comportamiento. Esto puede deberse a las propias exigencias del comportamiento, o bien a una limitación en las capacidades sensoriales del robot, de forma que los sensores produzcan salidas iguales ante situaciones distintas que además requieren acciones distintas por parte del robot. La única forma de evitar esta ambigüedad es incrementar la dimensionalidad del espacio sensorial. En analogía al "Embedding Theorem" [102] la dimensionalidad puede incrementarse considerando los datos sensados en instantes previos.

Se han propuesto diferentes métodos para proporcionar la capacidad de procesar información temporal a una RNA:

- Recurrencia. Se dice que hay recurrencia en una RNA cuando la información que sale de una neurona puede volver a ella, tal cual o procesada, por algún camino en la red, bien a través de conexiones de una neurona a sí misma o bien a través de otras neuronas. Dentro de este último caso tenemos tres formas habituales:
  - Todas las neuronas de una misma capa pueden estar conectadas entre sí, de tal forma que una neurona de dicha capa tiene acceso a los estados de activación previos que han sido procesados por otras neuronas de su misma capa. Beer y Gallagher [10] usan este método para el control de las patas en un robot hexápodo.
  - Una segunda forma de recursión consiste en convertir las salidas de las neuronas de una capa en entradas de las neuronas de la capa anterior [60]. En este caso, los estados de activación previos de una neurona son procesados en una primera instancia por neuronas de otra capa, ya que no hay conexiones directas de una neurona a sí misma, pero posteriormente pueden ser también procesados por las neuronas de la misma capa, en cuanto las salidas de la capa anterior se convierten en entradas para ella.
  - Finalmente, otra forma típica de abordar la recursión es mediante unidades contextuales o de memoria [31]. Éstas son neuronas que reciben como entradas las salidas de las neuronas de la capa oculta, y cuyas salidas son exclusivamente entradas para dichas neuronas de la capa oculta. De esta forma las neuronas de la capa oculta reciben una información más compleja que la que produce una

conexión de una neurona a sí misma (porque involucra información pasada de otras neuronas además de ella misma) y la red tiene un comportamiento más estable que el de una red con interconexión total. Este método es el aplicado por Miglino y col. [80] para obtener un comportamiento de "wandering"[4] en un robot que sufre de infrasensorización. La tarea requerida y el lastre de la infrasensorización le obligan a tener en cuenta información temporal para poder llevar a cabo el comportamiento deseado.

Las conexiones sinápticas recurrentes permiten obtener comportamientos que necesitan mantener un estado que, de alguna forma, resume la historia de la RNA en los instantes anteriores. Sin embargo, no es fácil con recurrencias usar información de instantes anteriores concretos. Para ello es necesario recurrir a otros mecanismos como los que describiremos a continuación.

- Transformar el dominio del tiempo en el del espacio mediante el uso de ventanas temporales. Cuando interesan los datos de un sensor, no sólo para el instante actual, sino también para instantes anteriores, se presentan esos datos de instantes anteriores como si fueran sensores adicionales. Es decir, en vez de tener una neurona para un sensor, tenemos n neuronas, una con el valor en el instante t, otra en el instante $t-1$ y así hasta $t-n+1$. Un ejemplo del uso de ventanas temporales lo tenemos en el trabajo de Janusz y Riedmiller [57], quienes usan esta técnica para evitar obstáculos con un robot Khepera del que sólo usan algunos sensores y, debido a esta limitación, necesitan tener en cuenta las entradas a los sensores en el instante actual y en el instante anterior. Las ventanas temporales permiten usar fácilmente dicha información, pero la red debe procesar íntegramente todos los datos dentro de la ventana temporal que se desea considerar, aunque no sean necesarios todos ellos, lo cual aumenta el tamaño de la red y, por tanto, el tiempo necesario para ejecutar el controlador en el robot. Otra dificultad añadida es la estimación del tamaño de la ventana temporal, es decir, cuántos instantes temporales es necesario tener en cuenta.
- Conexiones sinápticas con retardos. Los retardos son elementos que se sitúan en las conexiones sinápticas entre las neuronas y que retardan la propagación de los datos en el tiempo que especifican. El valor del retardo se puede establecer manualmente, u obtener automáticamente mediante entrenamiento [27] o evolución. Este método ha sido usado por Santos y Duro [99] para obtener varios comportamientos que implican la predicción de la ruta de un robot por otro robot. El uso de retardos modificables en las conexiones sinápticas permite, mediante entrenamiento o evolución, reducir el número de conexiones necesarias respecto al uso de una ventana temporal, al tiempo que mantiene la característica de permitir utilizar con facilidad información de instantes anteriores determinados.

4 "Wandering" es un comportamiento habitual en robótica autónoma consistente en recorrer la mayor superficie posible, no visitada previamente, en un tiempo determinado.

Neuronas de habituación

En este trabajo se ha optado fundamentalmente por este último método de utilizar retardos en las conexiones sinápticas, ya que permite tener en cuenta patrones temporales determinados sin implicar un aumento excesivo en la complejidad de la red y, además, su entrenamiento es mucho más fácil que en el caso de redes recurrentes. Y si bien en este trabajo no se han usado procesos de aprendizaje, sí está previsto en trabajos futuros poder incorporar aprendizaje en tiempo de vida del robot, con lo que también se tuvo en cuenta esta ventaja.

Adicionalmente, se ha desarrollado un nuevo tipo de neurona que hemos denominado neurona de habituación (HAN) y que permite incorporar muy eficazmente un determinado procesamiento de información temporal que será muy útil en algunos casos. Su modelo puede verse en la figura 20 y un ejemplo de respuesta ante un patrón determinado en la figura 21. Básicamente, la HAN disminuye su valor de activación con el tiempo si la entrada es constante, y se reinicia a su valor máximo cuando la entrada cambia. Su valor máximo depende también de la entrada: 1 cuando ésta es 1 y -1 cuando ésta es 0. Esto se ha definido así para adecuarla a las RNAs utilizadas en este trabajo, donde sus salidas pertenecen siempre al intervalo [-1:1], y a los sensores de infrarrojos del Rug Warrior, que sólo proporciona 0 o 1, aunque el modelo es fácilmente adaptable a otras circunstancias.

$$
\begin{aligned}
&x \in \{0,1\}\ \textit{entrada a la HAN}\\
&S \in [-1:1]\ \textit{salida de la HAN}\\
&\beta \in [0:1]\ \textit{coeficiente de habituación}\\
&x_0 = 0 \rightarrow S_0(x_0) = -1\\
&x_0 = 1 \rightarrow S_0(x_0) = 1\\
&t \geqslant 0, x_{t+1} = x_t \rightarrow S_{t+1}(x_{t+1}) = S_t(x_t) * \beta\\
&t \geqslant 0, x_{t+1} \neq x_t, x_{t+1} = 0 \rightarrow S_{t+1}(x_{t+1}) = -1\\
&t \geqslant 0, x_{t+1} \neq x_t, x_{t+1} = 1 \rightarrow S_{t+1}(x_{t+1}) = 1
\end{aligned}
$$

Figura 20: Modelo de la neurona de habituación.

El primer ejemplo que veremos mostrará la utilidad de este último tipo de neurona. Consistirá en obtener un controlador para un comportamiento de seguir paredes para el Rug Warrior utilizando sus dos sensores de infrarrojos. El seguimiento de paredes [86][61][22], junto con sus variantes de recorrer pasillos [25] y circuitos cerrados [34][97], es un comportamiento frecuentemente empleado como ejemplo en robótica autónoma. En este comportamiento, el robot debe recorrer un entorno a la mayor velocidad posible manteniéndose lo más cerca posible de las paredes en él presentes. En ocasiones, se exige que el robot sea además capaz de localizar una pared en caso de no sensarla inicialmente.

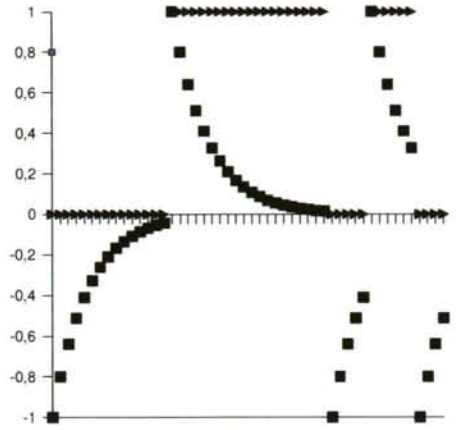


Figura 21: Respuesta de una neurona de habituación con β = 0.8 (los triángulos representan las entradas en el tiempo y los cuadrados las salidas).

Con el Rug Warrior, este comportamiento es más complejo de lo que pueda parecer inicialmente, debido al hecho de que los sensores de infrarrojos son binarios. Dichos sensores de infrarrojos no devuelven un valor distinto en función de la distancia a la que se encuentre el objeto detectado, sino que únicamente devuelven 1 o 0 dependiendo de si lo detectan o no. Esto, unido a que el ruido en el funcionamiento del sensor produce en ocasiones que lo que debiera ser un 1 sea un 0 y viceversa, provoca que el robot no sepa nunca ni su distancia real a una pared ni el ángulo con el que la está abordando. Obtendremos un controlador para el Rug Warrior que trate de implementar este comportamiento utilizando los sensores de infrarrojos. La evolución se llevará a cabo en un entorno con diversas paredes, empezando siempre cerca de una de ellas y con el siguiente modelo energético:

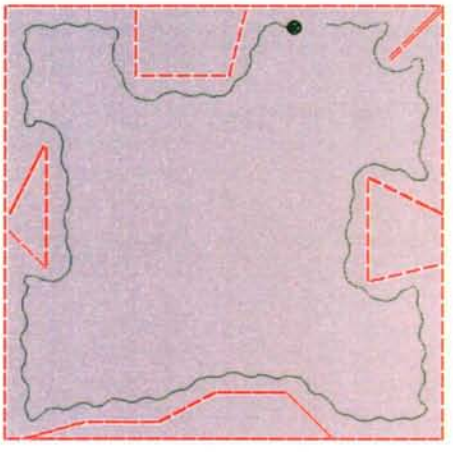
- Las paredes están formadas por ladrillos, todos del mismo tamaño.
- Cada ladrillo tiene comida adherida en su superficie.
- Esta comida es el alimento del robot, de tal forma que cuando el robot sensa un ladrillo come la comida presente en él y su nivel de energía aumenta de forma directamente proporcional a la cantidad de comida que haya.
- La comida en un ladrillo aumenta desde el momento en el que el robot deja de sensar el ladrillo hasta alcanzar su cantidad máxima o volver a ser comida por el Rug Warrior.
- La calidad de cada individuo en el proceso evolutivo es su nivel de energía al final de su vida.

El robot debe evolucionar, pues, hasta un comportamiento que le permita maximizar su nivel de energía, lo que conseguirá recorriendo el mayor número posible de ladrillos distintos en el menor tiempo posible, es decir, siguiendo las paredes.

En la figura 22 tenemos los datos de la RNA y del algoritmo evolutivo empleados para la obtención de este comportamiento de seguir paredes. Las entradas en la RNA se corresponden con los sensores de infrarrojos y sus funciones de activación son la identidad. En esa misma figura vemos la calidad para una evolución y el comportamiento resultante. La evolución es sencilla y el resultado se obtiene rápidamente.

Paramétros de la RNA	
Nº neuron. capa entrada (1)	2
Nº neuron. capa oculta (2)	6
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	-
Valor máximo de un peso	6

Parámetros del AE	
Tipo	GA
Distribución inicial	Diagonal
Prob. Cruce	1
Prob. Mutación	0,03
Razas	4
Poblaciones por raza	1
Generaciones	100
Migración local	10
Migración global	20
Población	256
Evaluac. por individuo	8
Pasos de vida	1000



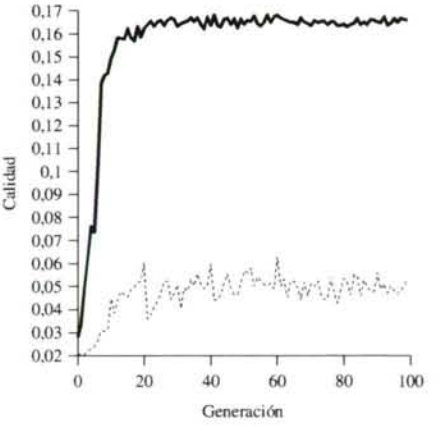


Figura 22: Comportamiento de seguir paredes para el Rug Warrior.

En la figura 23 vemos la calidad y el comportamiento resultante para una evolución en la que el robot, en vez de empezar pegado a la pared, empieza en el centro del entorno (con una orientación distinta en cada evaluación) y donde, además, si el robot choca y se queda atascado, se vuelve a colocar en dicho centro del entorno. Vemos cómo el robot no es capaz de trazar correctamente algunas curvas. Esto se debe a la falta de tratamiento de información temporal, la cual es necesaria para deshacer

situaciones ambiguas. Cuando el robot no percibe ningún objeto, no sabe si lo ha hecho previamente o no, con lo que no puede decidir cuál es la mejor estrategia, si girar bruscamente porque se está alejando de una pared, o si andar de forma más rectilínea porque todavía no ha localizado ninguna pared. Necesita, por tanto, establecer soluciones de compromiso, que le permitan, cuando no sensa nada, acercarse a una pared estando alejado de ella y volver a una pared lo más pronto posible cuando ya está recorriendo una. El robot, en cuanto sensa una pared, tiene que empezar a girar en sentido contrario porque no sabe cuan cerca está de ella y debe evitar los choques. Una vez deja de sensarla, si gira muy pronto se acercará antes a ella, pero entonces, si el robot no tiene cerca ninguna pared, bien porque ha perdido la pista de la que tenía, bien porque aparece al principio de su periodo de evolución alejado de cualquiera, quedará dando círculos sobre sí mismo. Obviamente, si el robot gira muy levemente cuando deja de sensar una pared, su calidad también será pobre, ya que tardará mucho en volver a ella y, por tanto, en incrementar su nivel de energía. De ahí, la necesidad de ese balance.

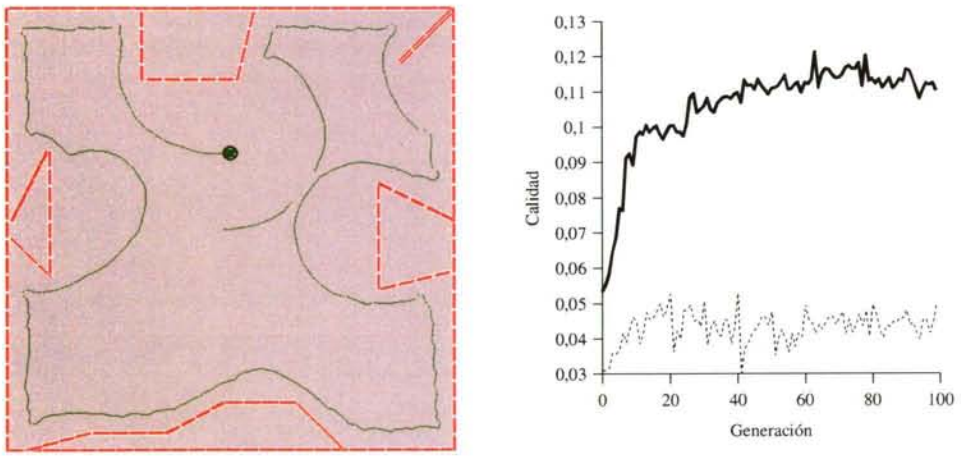


Figura 23: Comportamiento de búsqueda y seguimiento de paredes para el Rug Warrior.

A continuación realizamos una evolución con los mismos parámetros pero sustituyendo las neuronas de la capa de entrada por neuronas de habituación (HAN) e incorporando el parámetro β de éstas en el cromosoma. En la figura 24 vemos la calidad y el comportamiento resultante. Ahora, usando las HAN, el controlador es perfectamente capaz de llevar a cabo la tarea. El controlador podría ser incluso evolucionado con un menor número de neuronas en la capa oculta (se han obtenido resultados igualmente buenos con sólo 2 neuronas en la capa oculta), ya que gran parte del procesamiento necesario para seguir las paredes se consigue ajustando la β de las HAN.

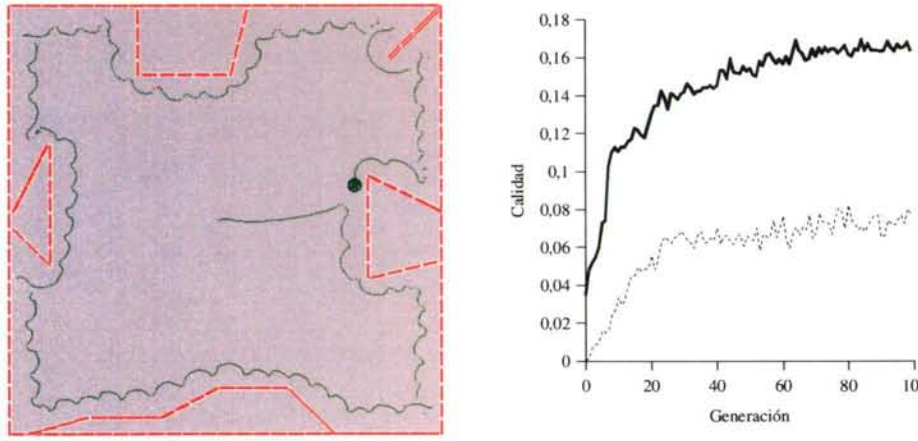


Figura 24: Comportamiento de búsqueda y seguimiento de paredes para el Rug Warrior empleando neuronas de habituación.

Retardos sinápticos

Como se comentó previamente, el uso de retardos sinápticos es, en general, la opción escogida en este trabajo para añadir a las RNAs la capacidad de procesar explícitamente patrones temporales. En la figura 25 podemos ver el esquema de una red con retardos en las conexiones sinápticas. Estas conexiones, además de un peso, tienen un retardo asociado (cuyo valor se obtendrá en la evolución al igual que el peso) que indica el tiempo que tarda la salida de una neurona de la capa de entrada en llegar a la correspondiente neurona de la capa oculta. En el esquema, las conexiones sinápticas con retardos temporales sólo están presentes entre las neuronas de la capa de entrada y la capa oculta, pero en realidad no tiene por qué ser así, y es, por supuesto, generalizable a tantas capas como se quiera. Sin embargo, la longitud del cromosoma no debe alargarse innecesariamente, y, en muchos casos, es suficiente con tener los retardos entre esas dos capas.

Es preciso notar que, para que se siga verificando el criterio establecido en [54] sobre la transferencia de comportamientos obtenidos en entornos simulados a un robot real, es necesario añadir también ruido a esta nueva dimensión. El ruido, en términos temporales, se traduce como variaciones en los instantes en los que se producen los eventos en un determinado patrón. Estas variaciones se pueden deber a distintas causas:

- La propia naturaleza del patrón de eventos que recibe el robot. Puede que ésta no sea totalmente determinista y que causas más allá de nuestro control, debidas a algún factor del entorno, conocido o no, alteren las propiedades del patrón a lo largo del tiempo.
- Limitaciones en la capacidad de cálculo del robot. Un cambio en el tiempo transcurrido entre obtenciones consecutivas de datos de los sensores, si esta operación es síncrona, o en el tiempo transcurrido entre el procesado de los datos

de los sensores, si es asíncrona, puede traer como consecuencia que el controlador vea un mismo patrón repetido en el tiempo como patrones distintos si no es capaz de percibir la misma secuencia de eventos.

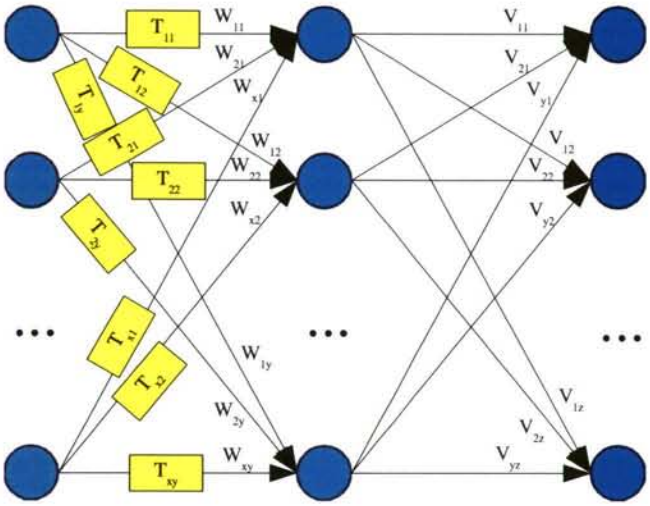


Figura 25: RNA con retardos sinápticos.

Por tanto, la RNA debe ser capaz de tolerar variaciones razonables en la duración y separación de los eventos de un determinado patrón.

El tipo de ruido más adecuado a introducir en la RNA dependerá de la causa de las variaciones. Por ejemplo, si las variaciones son pequeñas oscilaciones continuas en la distancia temporal entre eventos, deberá emplearse un ruido de media cero. Por contra, si la variación es más casual y, una vez producida, se prolonga en el tiempo, será necesario algún tipo de ruido sistemático. En los ejemplos que veremos a continuación, el entorno exterior al robot se comporta de forma estable, y es el propio robot el causante de la aparente divergencia entre eventos temporales en el entorno. Así, la causa principal es la diferencia en tiempo de cómputo que puede ser necesaria para ejecutar los distintos controladores, motivada por la reducida potencia computacional del Rug Warrior. Un simple cambio en el número de neuronas de una RNA puede producir diferencias en la percepción de un patrón temporal. Otras fuentes de problemas son la carga de la batería y los distintos coeficientes de rozamiento en la superficie por la que se desplaza el robot, ya que si el robot se mueve más despacio, percibirá los cambios en el entorno de forma distinta. Dadas estas posibles fuentes de problemas, el tipo de ruido más adecuado en este trabajo es el ruido sistemático, ya que el tiempo de cálculo de un controlador o el nivel de carga de la batería varía con poca frecuencia en el tiempo. Tras analizar las diferencias en la percepción de los eventos temporales debidas a estas dos causas, se optó por introducir el ruido sistemático de forma que, en todos los comportamientos que implican a controladores con retardos sinápticos, el tiempo que

transcurre entre dos adquisiciones de datos consecutivas por parte del Rug Warrior se escoge aleatoriamente al principio de cada evaluación entre los elementos del conjunto {0.2s, 0.3s}.

El comportamiento de seguir paredes comentado anteriormente también puede obtenerse empleando retardos en vez de, o además de, las neuronas de habituación, aunque no hay una ventaja apreciable sobre el uso en solitario de las neuronas de habituación.

El primer ejemplo que veremos de utilidad de los retardos sinápticos consiste en implementar un controlador para el Rug Warrior que le permita cazar un objeto luminoso. Primero lo obtendremos sin utilizar retardos sinápticos y posteriormente los añadiremos para discutir sus ventajas.

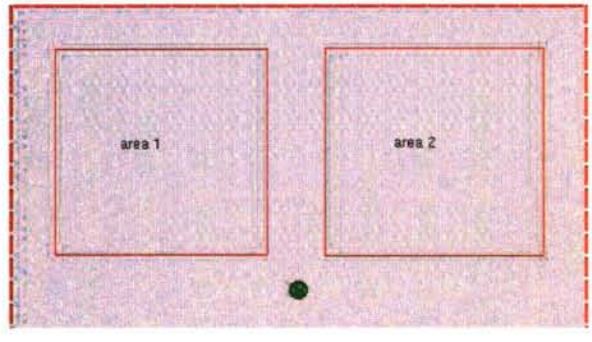


Figura 26: Entorno usado en la evolución del comportamiento de alcanzar luz y homing para el Rug Warrior.

El Rug Warrior deberá, por tanto, utilizar sus sensores de luz para alcanzar un objeto luminoso. Dicho objeto estará, en un principio, inmóvil y fácilmente localizable (como mucho el robot tendrá que girar ligeramente para localizarlo). Así, el entorno en el que se llevará a cabo la evolución se puede apreciar en la figura 26. El robot empieza cada ciclo de evaluación en el centro del entorno y con una orientación de 90º. El objeto luminoso se sitúa aleatoriamente en alguna posición dentro de alguna de las dos zonas cuadradas marcadas en la figura 26. El motivo de no colocar tanto el objeto como el robot aleatoriamente en cualquier punto del entorno es para facilitar la convergencia del algoritmo. De colocarse aleatoriamente ambos en el entorno, la calidad del mejor individuo oscilaría enormemente, ya que las situaciones pueden ser tan dispares que se puede dar el caso de que un individuo obtenga una calidad media elevada simplemente porque en una o dos ejecuciones el objeto apareció muy cerca de su posición inicial. Esto puede ser un gran problema, sobre todo en las primeras generaciones y dificultar innecesariamente la evolución. Es preferible obtener ahora un comportamiento que produzca que el robot alcance en el menor tiempo posible el objeto luminoso cuando lo tiene a la vista, y posteriormente, utilizando éste y otros comportamientos adicionales,

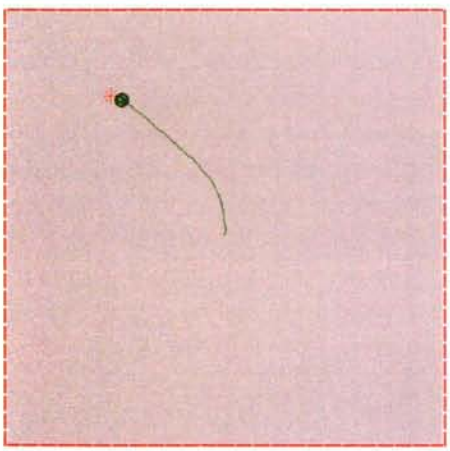
obtener un comportamiento global que además busque el objeto luminoso, lo cual se verá en el apartado 6.2.1.

El modelo energético empleado en la evolución es el siguiente:

- El robot comienza su evaluación con un nivel de energía de 1.
- Su energía decrece en cada instante de tiempo 0.005 hasta que contacta con el objeto luminoso. Esta cantidad de pérdida está seleccionada de tal manera que si alcanza el objeto en el último paso, tendrá una calidad de 0.5.
- Si no alcanza el objeto en su tiempo de ejecución, se le asigna calidad 0. La razón de esto es evitar que un individuo que alcanza el objeto luminoso hacia el final de su vida tenga una calidad muy similar a la de otro individuo que no lo alcanza.
- La evaluación se detiene en el momento en el que el robot alcanza el objeto luminoso o expira su tiempo de ejecución.

Paramétros de la RNA	
Nº neuron. capa entrada (1)	2
Nº neuron. capa oculta (2)	6
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	-
Valor máximo de un peso	6

Parámetros del AE	
Tipo	GA
Distribución inicial	Diagonal
Prob. Cruce	1
Prob. Mutación	0,02
Razas	4
Poblaciones por raza	1
Generaciones	1000
Migración local	10
Migración global	20
Población	256
Evaluac. por individuo	32
Pasos de vida	100



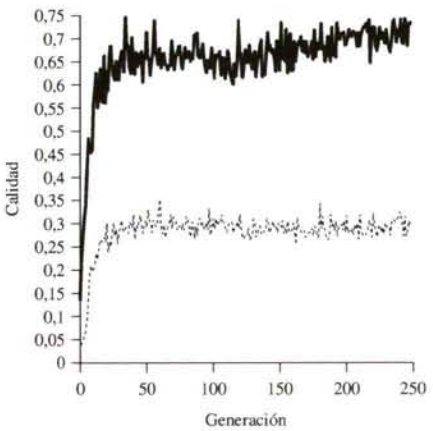
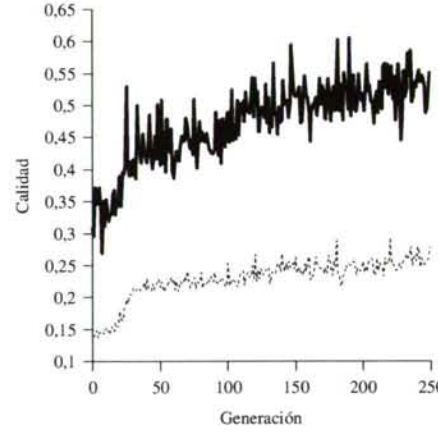


Figura 27: Comportamiento de alcanzar luz para el Rug Warrior sin tratamiento de información temporal.

En la figura 27 podemos ver los datos de la RNA utilizada, los parámetros del algoritmo evolutivo, la gráfica de calidad y el comportamiento de una evolución. El comportamiento es sencillo de obtener y no precisa de mayores comentarios.

Veamos ahora qué pasa si el objeto luminoso es móvil en vez de estático. En la figura 28, en la parte inferior izquierda, tenemos el controlador previamente obtenido tratando ahora de alcanzar un objeto luminoso móvil. Vemos cómo no es capaz de hacerlo al ir ambos a igual velocidad y no ser capaz de predecir los movimientos. En ocasiones podrá alcanzarlo, pero, en general, habrá muchas veces en las que no pueda hacerlo.

Paramétros de la RNA	
Nº neuron. capa entrada (1)	2
Nº neuron. capa oculta (2)	6
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Retardos.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	16
Valor máximo de un peso	6



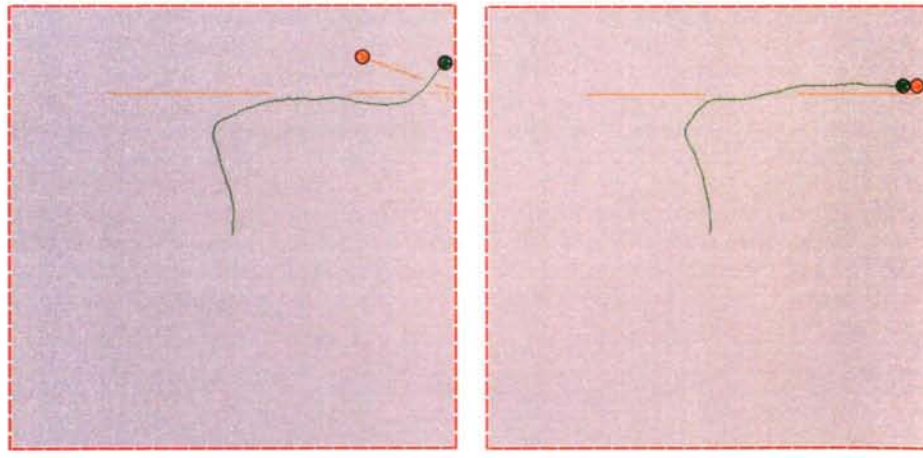


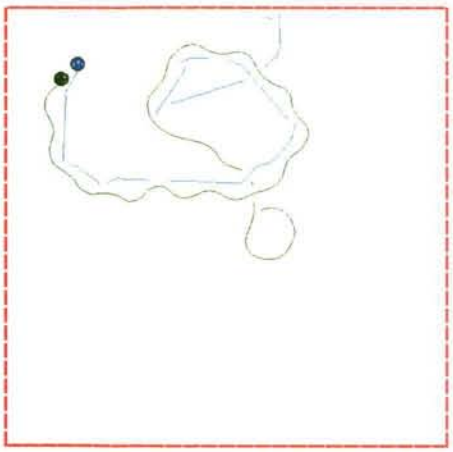
Figura 28: Comportamiento de alcanzar luz móvil para el Rug Warrior sin retardos sinápticos en la RNA (inferior izquierda) y con retardos (inferior derecha). En la parte superior se muestran los parámetros de la RNA con retardos y la gráfica de calidad máxima y media de una evolución.

Para obtener un comportamiento que sea más eficaz con objetos móviles obtendremos un nuevo controlador, esta vez utilizando retardos sinápticos entre las neuronas de la capa de entrada y las de la capa oculta. El entorno será el mismo que en el caso anterior y el resto de parámetros permanecen también igual. Las zonas de posible ubicación del objeto luminoso también permanecen iguales. La diferencia estriba en el objeto luminoso que ahora será otro robot, portando una luz, cuyo comportamiento será simplemente el de andar en línea recta a su velocidad máxima, girando cuando detecte un obstáculo. Para evitar que el segundo robot se dirija hacia el primero por casualidad, alterando artificialmente la calidad de éste y dificultando así el proceso evolutivo, el ángulo de partida en el entorno del robot con la luz será de 0° o 270° si se ubica en el cuadrado de la izquierda, o de 180° o 270° si se ubica en el de la derecha.

Como se aprecia en la figura 28, la calidad del mejor individuo oscila considerablemente. Esto se debe a que, a pesar de las restricciones impuestas en la evolución, las situaciones a las que se pueden enfrentar dos individuos dados pueden llegar a ser considerablemente distintas. Pese a ello, el comportamiento obtenido es satisfactorio.

Paramétros de la RNA	
Nº neuron. capa entrada (1)	1
Nº neuron. capa oculta (2)	6
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Retardos
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	32
Valor máximo de un peso	6

Parámetros del AE	
Tipo	GA
Distribución inicial	Diagonal
Prob. Cruce	1
Prob. Mutación	0,02
Razas	4
Poblaciones por raza	1
Generaciones	1000
Migración local	-
Migración global	100
Población	1000
Evaluac. por individuo	32
Pasos de vida	100



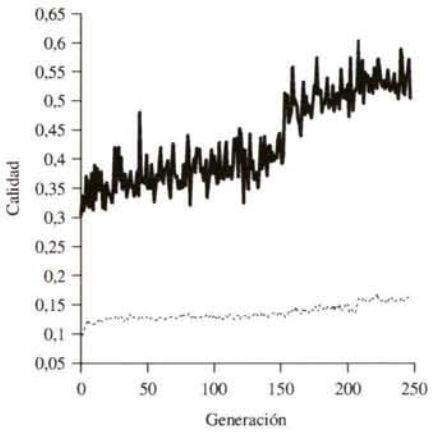


Figura 29: Comportamiento de alcanzar luz móvil para el Rug Warrior usando un único sensor de luz.

En la parte inferior derecha de la figura 28 se observa cómo los retardos han dado su fruto y el comportamiento del robot es ahora mucho más eficaz, prediciendo la ruta que sigue el robot con la luz. En las imágenes no se puede apreciar en qué momento pasó cada robot por cada punto, pero se ve cómo el robot alcanza su objetivo cuando éste contacta con la pared, mientras que antes no sólo no fue capaz de alcanzarlo en ese momento, sino que después perdió de vista el robot con la luz.

En el ejemplo anterior se aprecia, por tanto, cómo la utilización de información temporal mejora el rendimiento de un comportamiento. Veremos ahora que puede también suplir carencias en la sensorización debidas, por ejemplo, a un mal funcionamiento de algún componente del robot en un momento dado. Para ello, volveremos a obtener el mismo comportamiento que anteriormente pero ahora usando únicamente uno de los sensores de luz. En la figura 29 vemos los datos de la evolución y el resultado de una de ellas. Como se puede observar, el robot es capaz, utilizando un único sensor, de perseguir al objeto luminoso móvil utilizando sensorización activa. Es decir, suple su infrasensorización mediante movimientos adicionales. En este caso, dado que sólo dispone de un sensor de luz y no es capaz de discernir si el objeto se encuentra a izquierda o derecha, sólo a qué distancia se encuentra, realiza movimientos constantes de izquierda a derecha y viceversa para triangular la posición del objetivo. Y eso puede hacerlo gracias a la capacidad de almacenar información sobre instantes anteriores. En ocasiones, el robot pierde su presa, pero en esos casos gira sobre sí mismo hasta localizarla de nuevo.

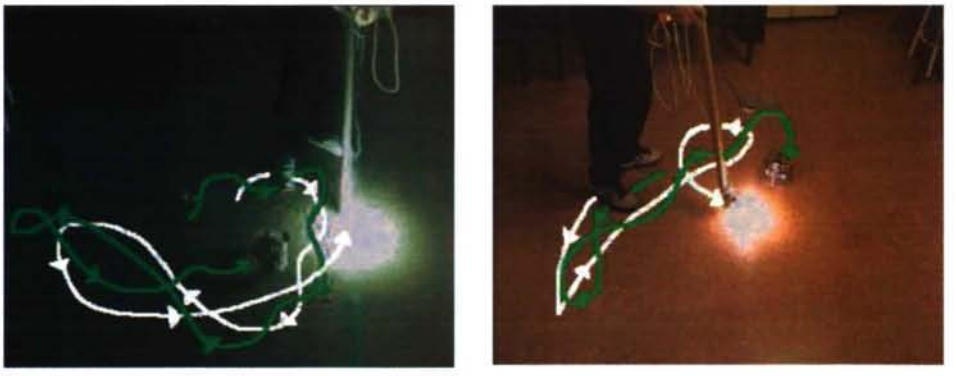


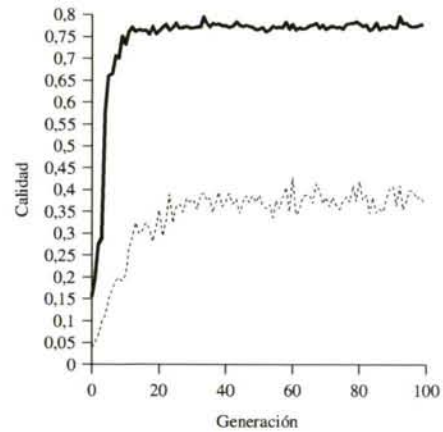
Figura 30: Comportamiento sobre el robot real de alcanzar luz móvil para el Rug Warrior usando un único sensor de luz. La línea blanca representa la trayectoria de la luz y la línea verde la trayectoria del Rug Warrior.

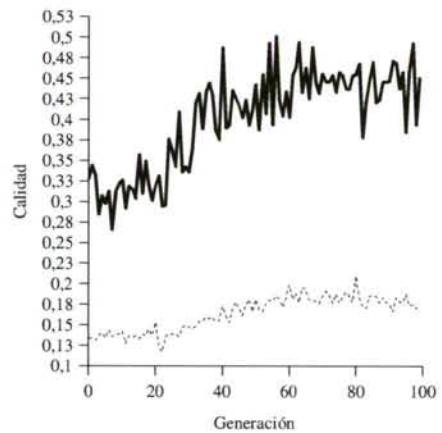
En la figura 30 podemos ver este comportamiento sobre el robot real bajo dos condiciones de luz opuestas, oscuridad total (a la izquierda) y luz ambiente (a la derecha). Como podemos observar, las trayectorias coinciden en apariencia con la observada en el simulador en la figura 29. La estrategia es la ya comentada de avanzar en zigzag para, mediante triangulación, calcular la posición de la luz. En ambas fotos

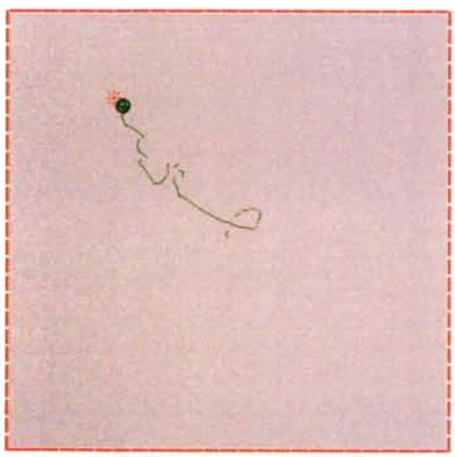
(en la parte izquierda de la foto "nocturna" y en la parte superior de la foto con luz ambiente) puede verse también un ejemplo de situación en la que el robot pierde la luz y gira sobre sí mismo para volver a encontrarla.

Paramétros de la RNA	
Nº neuron. capa entrada (1)	1
Nº neuron. capa oculta (2)	6
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Retardos
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	16
Valor máximo de un peso	6

Parámetros del AE	
Tipo	GA
Distribución inicial	Diagonal
Prob. Cruce	1
Prob. Mutación	0,02
Razas	4
Poblaciones por raza	1
Generaciones	100
Migración local	10
Migración global	20
Población	256
Evaluac. por individuo	32
Pasos de vida	100







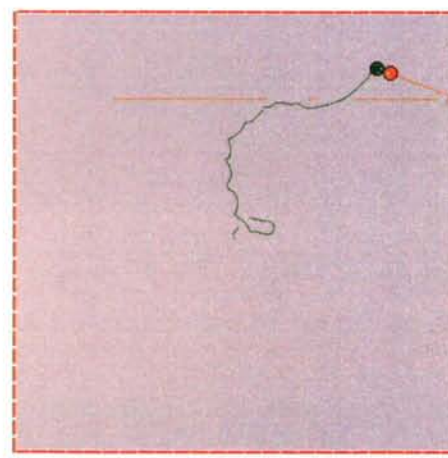


Figura 31: Comportamiento de alcanzar fuente emisora de calor (estática a la izquierda y móvil a la derecha) para el Rug Warrior utilizando el pirosensor.

En la figura 31 presentamos los datos correspondientes a este mismo comportamiento pero utilizando el pirosensor en vez de un sensor de luz. El comportamiento es muy similar aunque más dificultoso en su ejecución y menos efectivo por el peor funcionamiento del pirosensor comparado con los sensores de luz. Ahora el robot avanza en línea recta hasta que detecta que empieza a alejarse de su objetivo, momento en el que comienza a girar hasta localizar su posición de nuevo y volver a dirigirse hacia ella con la ruta corregida.

A continuación veremos otro ejemplo más de comportamiento utilizando retardos sinápticos. En este caso será un comportamiento que no podrá llevarse a cabo de ninguna manera sin la utilización de información temporal. El comportamiento en cuestión es *homing*, en el cual, el robot, que está perdiendo energía en cada instante de tiempo, debe dirigirse a su "casa" evitando caer en trampas. Su hogar viene representado por una luz intermitente, con un periodo de 0.6 segundos, estando la mitad del tiempo encendida y la otra mitad apagada. La trampa es una luz fija. Ambos objetos estarán estáticos y a simple vista del robot, es decir, no deberá buscarlos. Un comportamiento más complejo en el que deba buscar el hogar se presenta en el apartado 6.2.1.

El entorno es el mismo que en los problemas anteriores, con el añadido de que, además de ubicar el objeto con la luz fija aleatoriamente en uno de los dos cuadrados, se colocará el objeto con la luz intermitente aleatoriamente en el otro cuadrado. La razón para colocar los objetos de esta forma es la misma que en el apartado anterior: facilitar la convergencia a una solución. Si ponemos los objetos aleatoriamente en el entorno, puede darse el caso de que estén muy próximos o incluso de que el objeto con la luz fija esté entre el robot y el objeto con la luz intermitente, haciendo muy difícil en ambos casos que el robot pueda alcanzar el objeto con la luz intermitente y disminuyendo enormemente la calidad media del individuo que se encuentre con esta situación en alguna de sus evaluaciones, aunque en realidad sea un buen individuo.

El modelo energético empleado es el siguiente:

- El robot empieza con un nivel de energía de 1.
- En cada paso pierde 0.005 unidades de energía. De esta forma, un robot que alcanzase el hogar en el último instante de tiempo tendría un valor energético de 0.5.
- Si alcanza la casa o cae en la trampa su periodo de evaluación finaliza.
- Si cae en la trampa o agota su periodo de evaluación sin alcanzar la casa pierde toda la energía restante.

Como en todos los otros comportamientos donde se utilizan RNAs con retardos temporales, es necesaria la utilización de ruido temporal. Aquí, este ruido tiene un efecto devastador en el tiempo de obtención del controlador, ya que se altera el patrón que percibe el robot, y éste no debe de aprender un patrón fijo, algo fácil, sino aprender

a reconocer un objeto que a veces está encendido y a veces apagado, y a distinguirlo del hecho de perder de vista el objeto con la luz fija al girar hacia otro lado, algo perceptualmente muy similar al objeto con la luz intermitente y que obligará al robot a seguir estrategias de movimiento complejas.

Paramétros de la RNA	
Nº neuron. capa entrada (1)	2
Nº neuron. capa oculta (2)	6
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Retardos
Tipo conex. entre (2) y (3)	Pesos
Valor máximo de un retardo	32
Valor máximo de un peso	6

Parámetros del AE	
Tipo	GA/EE
Distribución inicial	Diagonal
Prob. Cruce	0,5
Prob. Mutación	-
Razas	1
Poblaciones por raza	1
Generaciones	1000
Migración local	10
Migración global	20
Población	400
Evaluac. por individuo	32
Pasos de vida	100

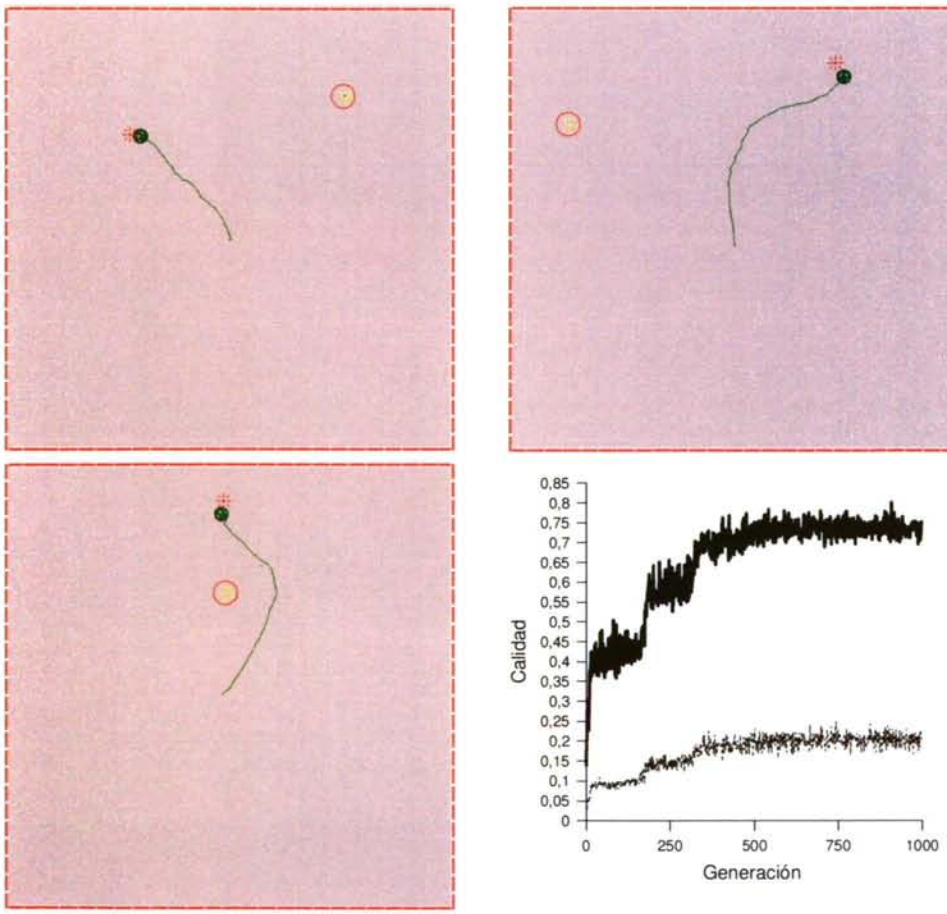


Figura 32: Comportamiento de homing para el Rug Warrior.

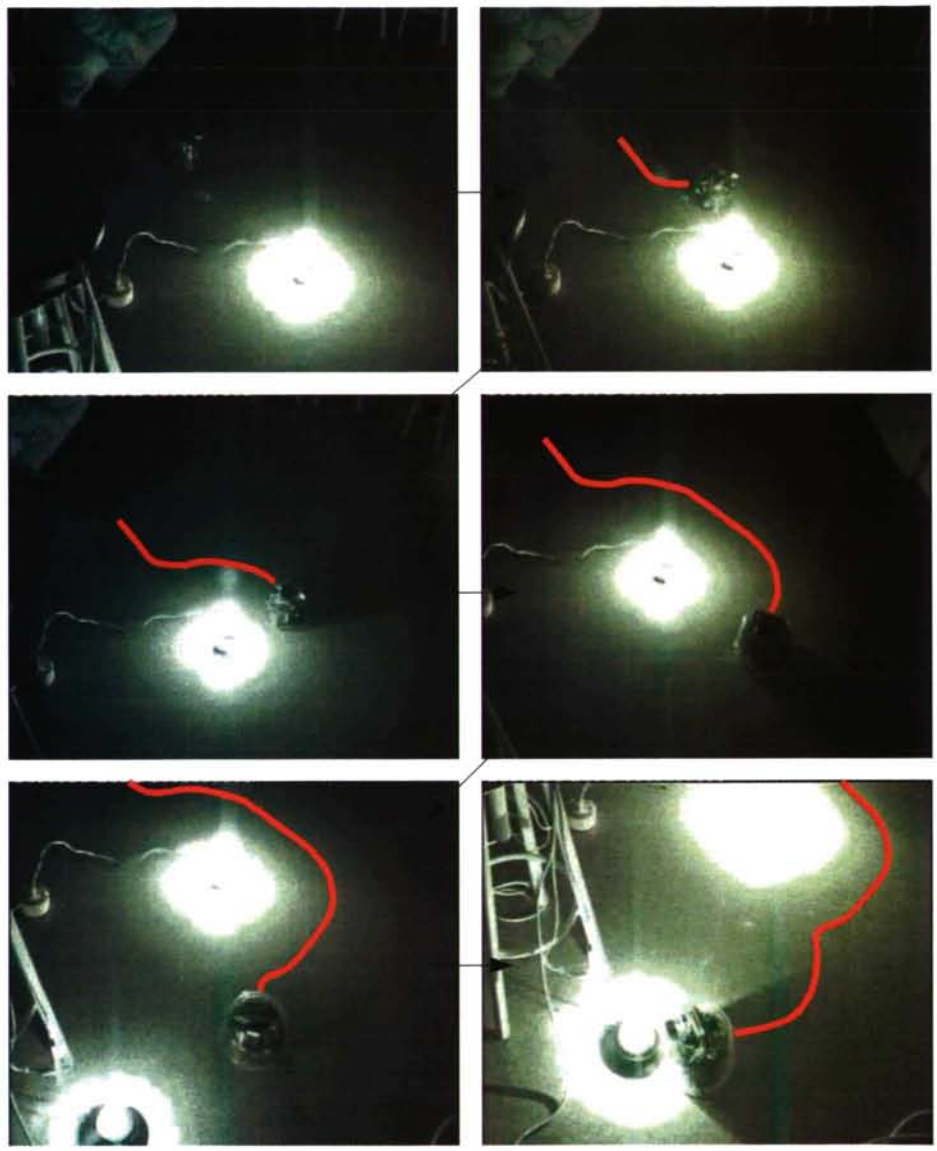


Figura 33: Comportamiento de homing para el Rug Warrior sobre el robot real.

En la figura 32 se indican los parámetros empleados para la RNA y el algoritmo evolutivo. En la misma figura se puede observar también la calidad de una evolución y tres ejecuciones del controlador obtenido en las que se aprecia que el robot es capaz de distinguir las dos luces y cumplir su misión. También se observa nuevamente, en el número de generaciones necesario para obtener el comportamiento y en la oscilación presente en la calidad del mejor individuo, el efecto del ruido temporal dificultando la

evolución. Asimismo, vemos también en la figura inferior izquierda como el robot es capaz de generalizar y afrontar con éxito situaciones muy diferentes a las encontradas durante su evolución, demostrando que la simplificación en las condiciones de la evolución no perjudicó su capacidad de generalización, ya que es capaz de esquivar la luz fija y dirigirse al hogar estando la primera interpuesta en su camino, no habiéndose dado nunca esa situación durante la evolución. Ese mismo caso, esta vez sobre el robot real, podemos contemplarlo en la figura 33.

En este apartado hemos visto, por tanto, cómo el añadir la capacidad de procesamiento de información temporal a las RNAs es beneficioso para mejorar comportamientos ya existentes, superar condiciones de infrasensorización y realizar comportamientos que de otra forma serían imposibles de realizar. Esto lo hemos hecho con un nuevo tipo de neurona (HAN) que maneja información temporal implícitamente en su función de activación, y que es muy útil en determinados casos, y mediante la utilización de retardos sinápticos en las conexiones, que se ha mostrado como un método muy potente de introducir el tratamiento de información temporal explícito cuando es necesario trabajar con patrones temporales de sensorización.

5.2 Sinapsis gausianas: mecanismo de atención y podado automático de RNAs

En este apartado vamos a describir cómo la utilización de sinapsis gausianas en las RNAs que empleamos como controladores ayudará a la metodología descrita en este trabajo a ligar, a través de las sensorizaciones, los comportamientos a las características relevantes de los entornos. Más en concreto, las sinapsis gausianas permitirán que cada módulo tenga por sí mismo la capacidad de actuar sólo ante patrones sensoriales determinados, ignorando el resto y evitando comportamientos imprevisibles en entornos totalmente distintos a los usados en su obtención. Es decir, funcionarán como auténticos mecanismos de atención. Y esto se conseguirá de forma automática en la propia evolución, sin intervención alguna del diseñador. Como veremos, esto se traducirá en un comportamiento más robusto y permitirá también reducir el tamaño efectivo de las RNAs que implementan los controladores.

Cuando no se usa un mecanismo evolutivo que contemple cromosomas de longitud variable y que fomente la obtención de soluciones con cromosomas lo más pequeños posible (lo cual, no está exento de problemas), se suele utilizar un cromosoma sobredimensionado, que en nuestro caso se corresponde con una RNA sobredimensionada, y, posteriormente, jugando con los parámetros, se trata de obtener una solución con un cromosoma más pequeño si es necesario. Una RNA sobredimensionada presenta una serie de problemas. El primero es que es más proclive a memorizar los casos encontrados durante la evolución y perder capacidad de generalización (sobreentrenamiento). El segundo es que no debemos olvidar que el robot, en una situación real, estará ejecutando continuamente varios controladores, correspondientes a los comportamientos que deba realizar, y que cuanto mayor sea el tamaño de las RNAs, mayor será el tiempo transcurrido entre sensorización y acción, con lo que nos conviene que las RNAs sean lo más pequeñas posibles.

Existen numerosos métodos para reducir el tamaño de RNAs eliminando pesos y neuronas (podado) [93]. Uno comúnmente utilizado consiste en introducir un decaimiento en los pesos [44][91], de forma que se substrae una pequeña cantidad del valor de cada peso durante el entrenamiento. Este decaimiento es proporcional a la magnitud de la conexión. De esta forma, las conexiones desaparecen a menos que sean reforzadas durante el entrenamiento. Estos mecanismos, sin embargo, si bien pueden usar o ignorar una conexión, no permiten usar un intervalo de las señales que circulan por ella como único a tener en cuenta. De esta forma, la RNA tiene que compensar mediante el uso de otras conexiones y nodos los valores fuera de ese intervalo de interés cuando éstos tienen lugar.

Como se ha visto hasta ahora, la mayor parte de las RNAs utilizadas en la robótica basada en comportamientos son redes multicapa *feedforward* o recurrentes [10][34][43][54], donde las neuronas de cada capa se conectan con todas las neuronas de la capa siguiente y los pesos sinápticos determinan la importancia de cada conexión.

Aunque estas redes han mostrado su elevada plasticidad y capacidad de adaptación a diferentes clases de problemas, también se ha comprobado que, debido a la limitada capacidad de procesado de sus nodos y conexiones, poseen serios problemas de escalabilidad que obligan a usar redes de tamaño considerable para problemas complejos, lo que se traduce en mayores tiempos para obtener la solución y mayor riesgo de caer en máximos locales.

Para tratar de evitar estos problemas, una aproximación obvia es tratar de aumentar la capacidad de procesado de los nodos o las conexiones sinápticas. En este trabajo se ha optado por lo segundo mediante la utilización de sinapsis gausianas. Las sinapsis gausianas son sinapsis en las que se sustituyen los pesos por funciones gausianas [28] (figura 34).

Sean:
x *valor a la entrada de la sinapsis*
$f(x)$ *valor a la salida de la sinapsis*
w *peso*
A *amplitud*
B *varianza*
C *centro*
Sinapsis tradicional:
$f(x) = w * x$
Sinapsis gausiana:
$f(x) = A * e^{-B*(x-C)^2}$

Figura 34: Sinapsis gausiana.

La elección de las sinapsis gausianas viene dada porque, a través de la modificación de sus tres parámetros (centro, amplitud y varianza), la función gausiana permite a la sinapsis establecer qué rango de valores de los que le entran son relevantes y en qué grado. Además, las redes con este tipo de conexiones pueden ser entrenadas mediante un algoritmo de gradiente descendiente [28], lo cual es una ventaja si se quiere dotar al robot de capacidad de aprendizaje en vida.

Para mostrar las ventajas de este tipo de conexiones frente a las conexiones tradicionales, utilizaremos para la comparación el ya conocido ejemplo de seguimiento de paredes pero con el robot Pioneer 2-DX. En la figura 35 podemos ver los parámetros utilizados en las evoluciones, así como las gráficas de calidades, tanto máximas como medias. Todos los parámetros que determinan las gausianas en las conexiones de la RNA evolucionada estaban en el cromosoma y fueron obtenidos, por tanto, mediante evolución. En las evoluciones se probó con distintos tamaños de redes, cambiando el número de neuronas en la capa oculta (4, 8 y 12) para comprobar cómo se comportaba el algoritmo evolutivo a medida que el tamaño de la red iba creciendo, manteniendo el tamaño de la población constante.

Parámetros de la RNA	
Nº neuron. capa entrada (1)	8
Nº neuron. capa oculta (2)	4/8/12
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Gausianas
Tipo conex. entre (2) y (3)	Gausianas
Valor máximo de un retardo	-
Valor máximo de un peso	6

Parámetros del AE	
Tipo	MA
Distribución inicial	Aleatoria
Prob. Cruce	-
Prob. Mutación	-
Razas	8
Poblaciones por raza	2
Generaciones	2000
Migración local	40
Migración global	80
Población	1600
Evaluac. por individuo	8
Pasos de vida	1000

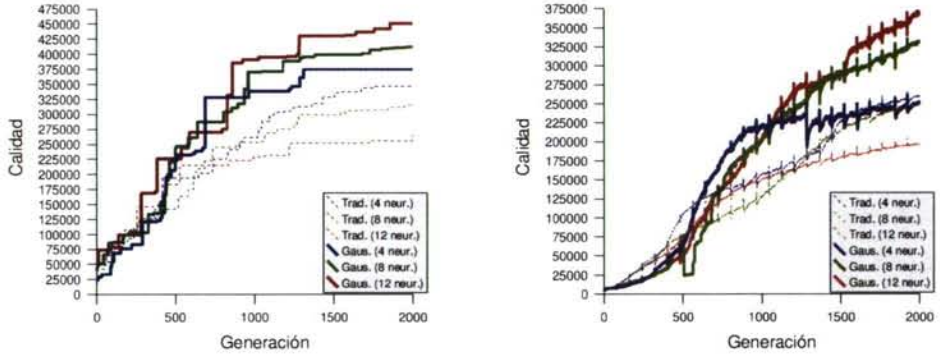


Figura 35: Parámetros de evolución, calidad máxima (izquierda) y calidad media (derecha) para 6 evoluciones del comportamiento de seguir paredes para el robot Pioneer 2-DX comparando la utilización de sinapsis gausianas con sinapsis tradicionales. Además del tipo de sinapsis, la diferencia entre las distintas evoluciones es el número de neuronas en la capa oculta de la RNA que implementa el controlador.

Se comprobaron varias cosas. La primera es que la calidad cuando se usaba una RNA con sinapsis gausianas era siempre mayor que cuando se utilizaba una RNA con sinapsis tradicionales. También se puede apreciar la dificultad de encontrar una solución en el espacio de búsqueda en la calidad media de las evoluciones: las poblaciones enseguida se agrupan en torno a los máximos locales y los descendientes de individuos presentes en máximos locales diferentes suelen tener una mala calidad debido a que las zonas del espacio de búsqueda entre dichos máximos locales suelen ser áreas de baja calidad, de ahí que la calidad media suba en las migraciones para bajar rápidamente a continuación. Lo curioso, por ser menos obvio en un principio, fue que al aumentar el tamaño de la red, la calidad con una RNA con sinapsis tradicionales disminuía. Esto es así porque la longitud del cromosoma aumenta enormemente (60, 108 y 156 genes para 4, 8 y 12 neuronas en la capa oculta). Cabe recordar que un aumento en el tamaño del cromosoma significa un aumento en el número de dimensiones del espacio de búsqueda, que implica un aumento exponencial en la extensión de la superficie de dicho espacio.

Así, al aumentar el espacio de búsqueda sin hacerlo la población, el algoritmo evolutivo tiene mayores dificultades para encontrar la solución. Sin embargo, eso no sucedió cuando la RNA utilizaba sinapsis gausianas.

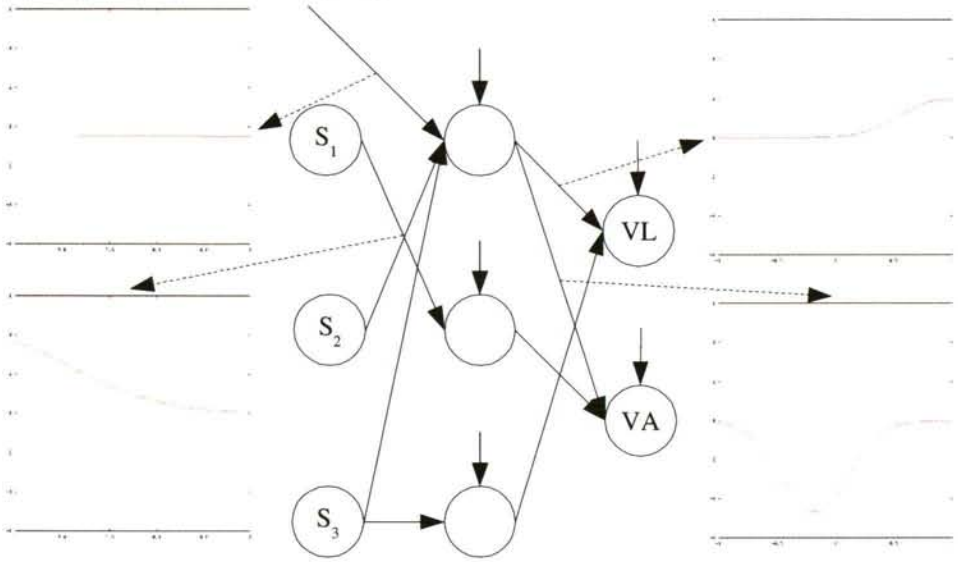


Figura 36: Ejemplo de RNA obtenida para el comportamiento de seguir paredes para el robot Pioneer 2-DX utilizando sinapsis gausianas y partiendo de una RNA con 8 neuronas en la capa de entrada y 12 en la capa oculta.

En la figura 36 se puede ver la razón para ello. En dicha figura, se muestra la red del mejor individuo en la evolución con una RNA con 12 neuronas en la capa oculta. Se han eliminado las conexiones cuya gausiana presenta un valor de 0 para todo el rango de posibles valores de entrada. También se han eliminado las neuronas cuyas conexiones de salida tienen siempre un valor de 0 para todos los posibles valores de salida. Como se puede ver, la RNA final es de un tamaño mucho más reducido que el esperado. No sólo eso, sino que en todas las evoluciones, independientemente del número de neuronas de partida en la capa oculta, se llegó a redes de un tamaño similar (2 o 3 neuronas de entrada y 2 o 3 neuronas en la capa oculta). El cómo llegó el algoritmo evolutivo a esas soluciones es gracias a que dicho algoritmo tiene ahora muchas mayores facilidades para anular una conexión (y por extensión una neurona), con lo que eliminar elementos redundantes es mucho más fácil. Para eliminar una conexión con gausiana, el algoritmo puede poner a 0 la amplitud, pero también puede poner una varianza muy grande o, mucho más fácil, desplazar el centro a un valor fuera del rango efectivo de valores que pueden circular por dicha conexión. Por contra, con una conexión tradicional sólo puede poner a 0 el peso para lograr eliminar la conexión. Eliminar aquellas conexiones innecesarias facilita en gran medida la labor del algoritmo evolutivo porque le permite reducir la dimensionalidad del espacio de búsqueda. De lo

contrario, tiene que ajustar cuidadosamente todas las sinapsis para que los cómputos realizados con ellas no distorsionen el resultado que debe proporcionar la red a la salida (recordemos que estos problemas poseen un alto grado de epístasis, es decir, el valor óptimo de un gen depende del valor que tomen otros genes). Si una red con tres neuronas en la capa oculta es suficiente para realizar la labor, uno podría preguntarse por qué la calidad final fue mejor en el caso con 12 neuronas en la capa oculta que en el caso con 4. La respuesta es que, sobre todo durante las primeras generaciones, las conexiones pueden ser eliminadas en el proceso de reproducción de los individuos, pudiéndose dar el caso de que dichas conexiones sean necesarias. Posteriormente, el propio proceso evolutivo puede volver a activarlas, pero eso supone un esfuerzo adicional puesto que implica establecer los valores adecuados en los tres parámetros de las gausianas. Por eso, proporcionar una red inicial sobredimensionada, no es perjudicial cuando se utilizan sinapsis gausianas e incluso puede resultar beneficioso, siempre y cuando no sean excesivamente grandes.

En la figura 36 se aprecia cómo son las gausianas para algunas conexiones. Se puede ver cómo existen conexiones que proporcionan un valor constante independientemente de la entrada y cómo hay conexiones que filtran el rango de valores de entrada para sólo considerar aquellos que son necesarios. Por ejemplo, la conexión que sale de S_1 posee una gausiana que provoca que se ignoren valores sensoriales del sónar correspondiente cuando estos están más allá de un determinado valor.

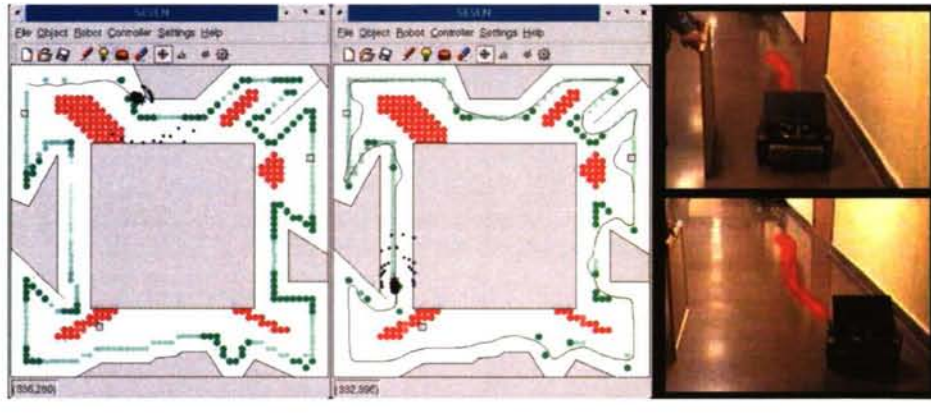


Figura 37: Comparación del comportamiento de seguir paredes para el robot Pioneer 2-DX frente a un mundo no visto en la evolución. A la izquierda un controlador con sinapsis tradicionales. En el centro un controlador con sinapsis gausianas. A la derecha, el controlador con sinapsis gausianas en el robot real.

En la figura 37 podemos ver el comportamiento del robot ante un entorno distinto del empleado durante la evolución. Las figuras izquierda y central muestran el robot en el simulador y las de la derecha en un entorno real. En las imágenes del simulador se aprecian círculos de color verde y de color rojo. Los primeros representan comida y los

segundos veneno, de forma que el robot incrementa su energía, y por tanto su calidad, al pasar por las zonas de comida y la decrementa al pasar por las zonas de veneno. El mundo utilizado en la evolución es muy similar al que se aprecia en las figuras del simulador excepto que en él no existían las paredes del centro de la habitación. Al cambiar el mundo, el controlador obtenido con sinapsis tradicionales (figura de la izquierda) no es capaz de realizar su tarea y choca, porque los valores que le entran por los sensores de la derecha (el robot sigue las paredes dejándolas a su izquierda) no estuvieron presentes nunca en la evolución. Sin embargo, eso no sucede en el controlador obtenido con sinapsis gausianas (figuras central y derecha). Al haber eliminado éste las conexiones innecesarias, las correspondientes a los sensores que están a su derecha entre ellas, el hecho de que haya o no objetos más o menos cerca a su derecha no le afecta, por cuanto sólo usa sensores a su izquierda/frente para seguir las paredes por su izquierda. Evidentemente, también podríamos haber obtenido un comportamiento de seguir paredes satisfactorio con sinapsis tradicionales aumentando el número de evaluaciones por individuo e incluyendo casos de evaluación dónde el robot se encontrase con entornos con paredes en el centro de la habitación pero, como hemos comprobado, esto no ha sido necesario utilizando sinapsis gausianas.

Las sinapsis gausianas, por tanto, no sólo permiten obtener redes más pequeñas, reduciendo el riesgo de caer en un máximo local con cromosomas muy grandes en el proceso, sino que permiten obtener controladores más robustos que sólo utilizan la información que precisan para su tarea, haciendo que sea mucho más difícil que conjuntos de datos sensoriales no presentes durante la evolución puedan trastornar el comportamiento del robot.

5.3 Conclusiones sobre la implementación de los bloques constructivos

En este capítulo hemos visto tres mecanismos que nos permiten aumentar las posibilidades de los controladores. Por una parte, las neuronas de habituación y los retardos sinápticos proporcionan a las RNAs la capacidad de procesar información temporal. Las neuronas de habituación incluyen esa capacidad en su modelo de funcionamiento, dependiendo su nivel de activación del tiempo que su entrada se mantiene constante. Los retardos sinápticos simulan sinapsis de distinta longitud y permiten a las RNAs tratar con patrones temporales determinados. La capacidad de procesar información temporal se ha mostrado útil para compensar situaciones de infrasensorización en los robots, para llevar a cabo tareas que por propia definición necesitan del tratamiento de información temporal y para mejorar comportamientos que son posibles pero poco eficientes sin dicho tratamiento.

Las sinapsis gausianas, por su parte, permiten que las RNAs, en el proceso evolutivo, aprendan para tener en cuenta únicamente aquellas entradas y aquellos intervalos de valores que son realmente relevantes para las tareas que tiene que llevar a cabo el robot, funcionando así como un mecanismo de atención y permitiendo la poda automática de las RNAs durante la evolución.

Llegado este punto tenemos definida una metodología que nos permite obtener, de forma automática y reduciendo el papel del diseñador al mínimo, controladores individuales, implementados utilizando RNAs, mediante algoritmos evolutivos. También hemos estudiado los problemas que pueden surgir en la aplicación de esta metodología y cómo solucionarlos, así como una serie de mecanismos adicionales que aumentan la capacidad de los bloques constructivos. En el siguiente capítulo pasaremos a estudiar cómo se pueden interconectar estos controladores individuales para generar estructuras multimódulo que permitan obtener incrementalmente controladores que dan lugar a comportamientos complejos y reutilizar módulos, tanto para un mismo robot como para robots distintos.

6 Arquitectura de comportamientos

En el capítulo 4 hemos visto la metodología que nos permite obtener los controladores que implementan los comportamientos deseados de forma modular y reduciendo al mínimo el papel del diseñador. A continuación veremos la arquitectura desarrollada en este trabajo para dar soporte a esa metodología, y que determinará las posibilidades de organización de los módulos para que éstos generen comportamientos complejos a partir de otros más simples.

Existen, en líneas generales, dos tipos posibles de arquitecturas para organizar comportamientos implementados modularmente: las arquitecturas jerárquicas y las arquitecturas distribuidas. En las arquitecturas jerárquicas, los módulos se organizan en niveles de forma que los módulos de niveles superiores pueden inhibir o activar a los de niveles inferiores, pero no a la inversa. En las arquitecturas distribuidas tal distinción no existe, cualquier módulo puede interactuar con cualquier otro, y se ha de desarrollar algún mecanismo que permita, mediante estas interacciones, determinar qué módulo o qué módulos deben tomar el control del robot en cada momento. En una primera parte de este capítulo veremos con más detalle estas alternativas, sus ventajas e inconvenientes y ejemplos de implementaciones.

En la segunda parte del capítulo se describirá la solución desarrollada. Ésta parte de un planteamiento inicial jerárquico para dar lugar a una arquitectura final con características de ambas aproximaciones y que trata de aprovechar sus respectivas ventajas y corregir sus inconvenientes. Así, veremos que la arquitectura tiene dos posibles representaciones. Una es jerárquica y permite fácil e intuitivamente construir de forma incremental controladores complejos reutilizando otros ya existentes. Cualquier controlador representado de esta forma puede convertirse directamente a otra representación con únicamente dos niveles de módulos: un primer nivel con módulos que pueden tomar el control de los actuadores, y un segundo con módulos que se relacionan entre sí para decidir cómo deben comportarse los del primer nivel.

La arquitectura posibilitará diversos tipos de conexión entre los módulos: la selección, como forma de implementar el típico mecanismo de inhibición/activación, y la modulación, como una forma de combinar varios módulos que podrán tomar el control de los actuadores simultáneamente pero con distinta determinación. Experimentaremos con dos tipos posibles de modulación, una que actúa modificando los datos percibidos por los módulos y otra que actúa modificando los valores de actuación. Veremos también como las selecciones, tal cual se planteará la arquitectura, se podrán contemplar como casos particulares de modulaciones de actuadores, lo cual nos permitirá realizar la transformación entre las dos posibles representaciones comentadas anteriormente.

Al final, tendremos una arquitectura que soporta la metodología presentada anteriormente y que permite la consecución de todos los objetivos planteados de

obtención incremental y reusabilidad mediante un amplio rango de interacciones posibles entre los módulos.

6.1 Alternativas

Como se describió en el capítulo 3, el mayor problema de la robótica basada en comportamientos es la obtención de comportamientos complejos. En este capítulo analizaremos las alternativas empleadas para tratar de resolver este problema.

Teóricamente, podría obtenerse cualquier comportamiento en un único módulo. Las primeras aproximaciones dentro de la robótica evolutiva presentaban este tipo de arquitecturas monolíticas, esto es, el controlador, independientemente de la complejidad del comportamiento, se obtenía como una única entidad en un único proceso. Este es el mejor método cuando el comportamiento es sencillo, tal y como se ha visto en los primeros capítulos de esta memoria, ya que no es necesario realizar ninguna división en módulos ni implementar ningún mecanismo que obtenga esa división y, por tanto, la participación del diseñador es más fácilmente minimizable.

Para comportamientos complejos, sin embargo, esta aproximación no es viable. Hay un primer problema de escalabilidad en la imposibilidad de reutilizar controladores previos para obtener otros más complejos, pero el mayor problema tiene que ver con la dificultad de plantear la obtención de un controlador complejo partiendo de cero sin dividir el proceso en subtareas. Por ejemplo, si quisiésemos hacerlo usando la metodología presentada en este trabajo, tendríamos que definir una función de calidad lo suficientemente gradual para que las poblaciones iniciales de individuos (muy alejados del comportamiento correcto) proporcionen información al algoritmo evolutivo de forma que éste sepa qué individuos son mejores. Tendríamos también que determinar la configuración neuronal necesaria para implementar el controlador. Por otra parte, un único módulo excesivamente complejo va en contra de uno de los principios de la robótica basada en comportamientos ya que puede implicar que el tiempo de reacción del robot sea demasiado elevado.

Debido a todo esto, parece claro que una solución obvia es particionar el problema grande en problemas más pequeños. O, en este caso, dividir ese comportamiento de muy alto nivel en comportamientos más simples que se puedan obtener más fácilmente para luego, de alguna forma, unirlos. En este caso, los retos estriban en cómo identificar esos comportamientos de menor nivel y determinar dónde acaba uno y empieza otro. Esta no es una tarea fácil, sobre todo teniendo en cuenta que el comportamiento global puede deberse a la interacción simultánea de varios comportamientos de bajo nivel, por lo cual lo que se visualiza no tiene por qué ser unívocamente identificable con un único comportamiento de bajo nivel.

Otro factor a tener en cuenta es la propia definición de comportamiento en esta filosofía, donde se establece una relación entre el robot y entorno, de tal forma que el comportamiento está ligado a ambos y si el entorno cambia puede que el comportamiento también cambie y ya no pueda ser etiquetado de la misma forma que en el entorno original. Basta imaginar, por ejemplo, un comportamiento de seguimiento de paredes en un entorno sin paredes.

Además, es necesario tener en cuenta que, salvo cuando solamente son necesarios unos pocos módulos para implementar el controlador, la complejidad en el diseño crece con el número de posibles interacciones entre los módulos (Cliff [18]). Por eso, la importancia ya comentada de minimizar el papel del diseñador se vuelve todavía más fundamental y se extiende, no sólo a los módulos, sino a las relaciones entre ellos. Harvey [42], de forma más detallada, enumera tres dificultades a la hora de obtener un controlador concebido de forma modular:

- La descomposición de un sistema de control de un robot en componentes no es siempre evidente.
- Las interacciones entre los módulos son más complejas que simples conexiones y algunas de ellas vienen determinadas por el entorno.
- A medida que la complejidad del sistema crece, la interacción entre los módulos crece exponencialmente.

Respecto a cómo pueden interaccionar los módulos para dar lugar a controladores más complejos, Dorigo y Colombetti enumeran las siguientes posibilidades [26]:

- Suma independiente.

Dos o más comportamientos accionan actuadores distintos simultáneamente.

- Combinación.

Dos o más comportamientos accionan el mismo conjunto de actuadores simultáneamente para dar lugar a un comportamiento nuevo. Por ejemplo, pueden estar activos en un robot simultáneamente los comportamientos de esquivar obstáculos y seguir un objetivo móvil, dando lugar a un comportamiento en el que el robot esquiva obstáculos pero los movimientos para ello lo mantienen en la persecución del objetivo.

- Inhibición.

Un comportamiento inhibe a otro comportamiento. Esto implica una jerarquía, establecida o aprendida, entre los comportamientos. Teniendo varios comportamientos simples y una red de inhibiciones entre ellos también se pueden obtener comportamientos complejos.

- Secuencia.

Varios comportamientos se ejecutan unos tras otros en una secuencia que no tiene por qué ser fija, sino que puede variar en función de los datos de los sensores o estados internos, para dar lugar a un comportamiento distinto y más complejo.

Estas interacciones entre los comportamientos se pueden tratar de conseguir mediante diversas arquitecturas de controladores. Pfeifer y Scheier [90] realizan una clasificación basándose en la forma de tomar el control sobre los actuadores por parte de los módulos, distinguiendo entre arquitecturas competitivas y arquitecturas

cooperativas. En las arquitecturas competitivas un único módulo pone valores en los actuadores en cada instante de tiempo y los otros módulos son inhibidos o ignorados (ejemplos son las arquitecturas subsumidas [12] y la red de activaciones presentada por Urzelai y col. en [104]). En las arquitecturas cooperativas, las salidas de dos o más módulos son combinadas, normalmente mediante una suma ponderada, antes de enviarlas a los actuadores (un ejemplo es el uso de campos de potencial que se corresponden a comportamientos [6][7]).

Atendiendo al número de módulos, las arquitecturas se pueden clasificar, como ya hemos visto, en monolíticas, donde un único controlador implementa todos los comportamientos, o modulares, donde existen varios controladores y cada uno de ellos implementa un comportamiento de mayor o menor nivel (existen diversas granularidades). Si usamos arquitecturas modulares tenemos a su vez dos clases según el tipo de control aplicado a los controladores: arquitecturas centralizadas o jerárquicas y arquitecturas distribuidas.

En las arquitecturas jerárquicas los módulos se organizan en una jerarquía, de forma que se establecen niveles en su estructura. Los módulos del más bajo nivel reciben los datos directamente de los sensores y sólo pueden actuar sobre los actuadores, mientras que los módulos de niveles superiores pueden recibir información tanto de los sensores como de los controladores de niveles inferiores y, dependiendo de la arquitectura utilizada, pueden actuar sobre los actuadores o seleccionar de alguna forma qué módulos de niveles inferiores pueden actuar o no y cómo deben de hacerlo. En algunos modelos, los controladores de un nivel determinado también se comunican con los controladores de su mismo nivel. Aquí es fácil construir nuevos controladores a partir de otros ya existentes, ya que sólo basta añadir nuevos módulos y niveles a jerarquías ya existentes. El inconveniente es que la participación del diseñador puede ser elevada ya que, en principio, es él quien tiene que determinar cuántos módulos tiene que haber, cómo se relacionan y qué tiene que hacer cada uno.

Las arquitecturas subsumidas propuestas por Brooks [12] son un ejemplo de arquitectura jerárquica. El controlador global se divide en módulos, los cuales se agrupan en niveles, cada uno de ellos encargado de realizar una tarea básica, y uniéndolos entre sí mediante conexiones activadoras o inhibidoras, de forma que cada módulo, que sólo puede ser activado o inhibido por un módulo de un nivel superior, se activa sólo cuando es necesario. Pero este modelo presenta el problema habitual ya comentado de las arquitecturas jerárquicas: es el diseñador quien establece qué módulo o módulos se corresponden con un comportamiento y cuándo y de qué forma tienen que activarse. Aunque hay una conexión directa entre el entorno y las reacciones del robot, el diseñador tiene un papel decisivo, que lleva a muchos de los problemas presentes en la robótica basada en el conocimiento, además de la dificultad en el escalado de esta solución, cuando el número de módulos y conexiones entre ellos aumenta.

Otro ejemplo de arquitectura jerárquica es el presentado por Colombetti, Dorigo y Borghi [20], donde cada módulo es un sistema clasificador. La aplican a dos ejemplos. En uno, un robot *AutonoMouse* (diseñado por los autores) tiene que buscar y dirigirse a una luz. El controlador está formado por esos dos módulos de bajo nivel y uno de alto nivel que conmuta entre ambos. En otro ejemplo, en el que la arquitectura se vuelve en realidad un híbrido entre jerárquica y distribuida, un robot *Hamster* (un robot de tipo Robuter) tiene que llevar comida al "nido". El controlador tiene un módulo de alto nivel que selecciona en cada momento uno de entre tres módulos de bajo nivel (dejar nido, coger comida, dirigirse a nido) cuya salida es sumada, por lo general, a la salida de un cuarto módulo de bajo nivel (evitar obstáculos), aunque a veces inhibe a éste último (al recoger la comida).

En las arquitecturas distribuidas no existen prioridades, ni jerarquías ni clases entre los módulos, sino que todos ellos compiten de alguna forma por el control de los actuadores en cada instante de tiempo. La participación del diseñador es, en principio, menor que en las arquitecturas jerárquicas, y también es posible reutilizar módulos, pero la dificultad a la hora de obtener comportamientos que impliquen un alto número de módulos puede ser elevada debido a la potencial complejidad de las relaciones entre ellos. Un concepto en el que se basan numerosos autores que defienden el uso de este tipo de arquitecturas frente a las jerárquicas es el de considerar la estructura modular como un fenómeno emergente [90], como una consecuencia propia del proceso de creación y no como una imposición del diseñador. En esta dirección se han visto trabajos (Urzelai y col. [104], Nolfi [84] y Passeman [89]) en los que, dentro de estructuras cognitivas de base neuronal evolucionadas para ser controladores de robots autónomos, se puede determinar a posteriori la existencia de módulos que pueden ser identificados con comportamientos de menor nivel dentro del comportamiento global y que, efectivamente, surgieron de la interacción del robot con el entorno. Aunque estos trabajos son realmente interesantes desde un punto de visto teórico, presentan problemas prácticos relacionados con la escalabilidad y dificultades para establecer una metodología que permita la reutilización de esos módulos identificados para la obtención de posteriores y más complejos controladores.

Un ejemplo de arquitecturas distribuidas es la teoría de esquemas de motores de Arbib [6] y Arkin [7], donde explican el comportamiento en términos de control concurrente de muchas actividades, sin ningún tipo de arbitraje y donde cada comportamiento contribuye en diversos grados a la respuesta global del robot. En los ejemplos de Arkin, cada comportamiento tiene asociado un vector que se corresponde con un campo de potencial y la coordinación entre comportamientos se obtiene sumando los vectores.

Otro ejemplo es el presentado por Urzelai y col. [104]. Los módulos han sido previamente obtenidos o poseen capacidad de aprendizaje (por refuerzo). Poseen una salida adicional, a mayores de los valores para los actuadores, que representa cuánto "desea" el módulo participar en cada momento en la acción a realizar por el robot.

Además reciben como entrada también esos niveles de activación correspondientes al resto de los módulos. Esta red de activación es obtenida mediante evolución y, en cada momento, el módulo con mayor nivel de activación toma el control de los actuadores. La arquitectura es incremental ya que si se añade un nuevo módulo, sólo se evoluciona la parte nueva de la red de activación, aunque esto implica una restricción en cuanto al conjunto de posibles soluciones. Como ejemplo, primero obtienen una arquitectura distribuida para un robot Khepera que debe explorar un entorno buscando zonas de recarga para su batería, con módulos para movimiento en línea recta, evitar colisiones, seguimiento de luces y recarga. Posteriormente, y partiendo de la arquitectura inicial, la expanden con módulos para coger y soltar objetos con la pinza del Khepera, obteniendo un comportamiento global de limpieza del entorno.

Por supuesto, existen arquitecturas híbridas que no pueden considerarse ni puramente jerárquicas ni puramente distribuidas. Tal es el caso de la presentada por Nolfi en [85] y [84], donde se usa una única RNA para implementar un comportamiento en el cual el robot debe mantener un entorno limpio, recorriéndolo y arrojando fuera de él los objetos que encuentre. La modularidad se obtiene porque la RNA tiene los nodos de salida replicados y un mecanismo de selección que decide qué nodos actúan sobre los actuadores. Ese mecanismo de selección está compuesto por neuronas que reciben información de los sensores y determinan las salidas a seleccionar. En esta arquitectura no es necesario determinar qué tiene que hacer cada comportamiento ni las conexiones entre ellos, aunque el número máximo de posibles subcomportamientos está preestablecido por las posibles combinaciones de los nodos de salida que determina el mecanismo de selección. Por contra, los comportamientos no pueden ser reutilizados. En estos trabajos, Nolfi establece una diferencia entre una descripción "distal" y otra "proximal" de los comportamientos. La primera hace referencia a la descripción que hace de cada comportamiento un observador externo. La segunda se describe desde el punto de vista del sistema sensorial-motriz del robot, cómo reacciona éste ante distintas percepciones. Nolfi observa cómo comportamientos que, de acuerdo a una descripción "distal", pueden ser considerados básicos, se corresponden en realidad, desde un punto de vista "proximal", con comportamientos resultado de la contribución de varios módulos distintos. Por esto, concluye que no hay una correspondencia entre ambos tipos de descripciones, es decir, entre comportamiento y módulo, y que, por tanto, debe evitarse una división manual de un comportamiento en módulos ya que esto puede limitar la solución final a una que cumpla las restricciones impuestas por el diseñador, sin que por ello sea la mejor solución.

6.2 Solución desarrollada

En el apartado anterior hemos visto las distintas alternativas en cuanto a arquitectura a la hora de obtener comportamientos complejos. Las ventajas e inconvenientes de estas alternativas se pueden resumir gráficamente en la figura 38. Las arquitecturas monolíticas son las susceptibles de necesitar menos intervención por parte del diseñador al no tener que realizar éste ninguna subdivisión en módulos, pero también son aquellas que presentan una mayor dificultad a la hora de obtener un comportamiento complejo por la necesidad de hacerlo en único proceso, y, por supuesto, su reusabilidad es prácticamente inexistente. Las arquitecturas jerárquicas y distribuidas poseen mayores capacidades de reutilización al estar los controladores divididos en módulos. Las jerárquicas necesitan, en principio, un mayor grado de intervención por parte del diseñador respecto a las distribuidas, al tener que establecer éste las relaciones entre los módulos; pero obtener comportamientos complejos es, también a priori, más sencillo por haber un menor número de relaciones entre módulos e imponer la jerarquía un control sobre cuál decide qué en cada momento, mientras que en las distribuidas todos los módulos pueden estar conectados entre sí y se necesita algún proceso mediante el cual se pongan de acuerdo para decidir qué módulo o módulos toman el control del robot en un instante determinado.

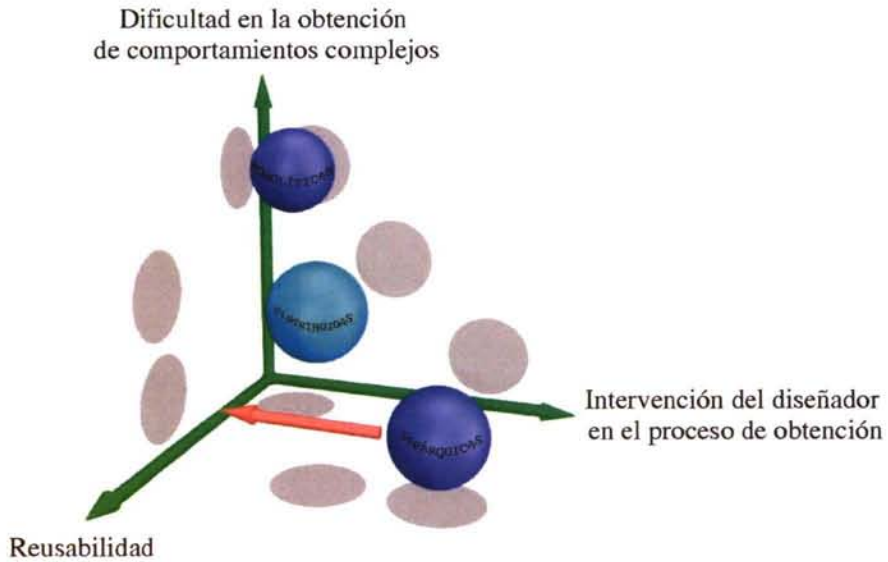


Figura 38: Tipos básicos de arquitecturas.

Dado que no es posible obtener en un tiempo razonable controladores que implementen comportamientos complejos de forma monolítica, es necesario utilizar algún tipo de arquitectura modular. Ya que se han considerado como los dos puntos más

importantes la minimización del papel del diseñador y la facilidad en la reutilización de los módulos obtenidos, en este trabajo se ha partido de una arquitectura jerárquica por ser la que ofrece a priori una menor dificultad en la obtención de comportamientos complejos y se ha desarrollado hasta dar lugar a una arquitectura que posee características de ambas aproximaciones modulares en un intento de corregir sus respectivos puntos débiles.

Primero veremos el proceso de construcción automático de la arquitectura utilizando el mecanismo de selección, el cual implementa las activaciones / inhibiciones habituales en las arquitecturas jerárquicas, con objeto de demostrar sus posibilidades de reutilización. Posteriormente, veremos cómo se puede expandir la reutilización de módulos a robots de diferente morfología. Finalmente, se describirá el más flexible y capaz mecanismo de modulación, en dos de las tres variantes posibles. Veremos, además, que las selecciones pueden ser consideradas como un caso particular de modulación. Gracias a esta propiedad, se comprobará que cualquier controlador puede representarse de forma alternativa en una estructura con sólo dos niveles de módulos, donde los módulos del nivel inferior toman el control de los actuadores y los del nivel superior deciden de forma conjunta cuáles de los del nivel inferior, y con qué grado de participación, realizan ese control de actuadores.

6.2.1 Selección

Como se ha comentado en el apartado anterior, la arquitectura aquí desarrollada parte en principio de una estructura jerárquica donde los módulos se organizan en niveles. Esta estructura la representaremos en forma de árbol. Así, los módulos del nivel inferior, nodos hoja en la terminología de árboles, son los que son susceptibles de tomar el control de los actuadores, mientras que los restantes módulos determinan cuál de sus hijos debe ser activado en cada momento (cada nodo sólo activa a uno de sus hijos). En cada paso del robot el control fluye desde el nodo raíz y llega hasta un nodo hoja que toma el control de los actuadores.

La idea principal en este esquema es que, para obtener un controlador que implemente un determinado comportamiento, el diseñador proporciona al sistema aquellos módulos que él considera que pueden ser necesarios y éste evoluciona un módulo que se situará en un nivel superior a los ya existentes y que seleccionará en cada momento un módulo de los ya disponibles para tomar el control de los actuadores. En este proceso se le pueden proporcionar al módulo que está siendo evolucionado más módulos de un nivel inferior de los realmente necesarios, con el único posible handicap de que el tiempo necesario para obtener la solución sea mayor. Si, además, se necesita algún módulo de bajo nivel adicional para realizar una actuación que ninguno de los módulos disponibles es capaz (aunque el diseñador desconoce qué actuación) se puede obtener simultáneamente con el módulo de alto nivel en la misma evolución.

Este proceso se puede repetir sucesivamente, de forma que módulos que en su momento fueron obtenidos como módulos de alto nivel, estén ahora disponibles para obtener un módulo de un nivel todavía mayor, dando lugar así a la estructura de módulos en forma de árbol ya mencionada. En la figura 39 podemos ver un esquema del funcionamiento del mecanismo. Las conexiones que se aprecian de distintos nodos a un único nodo de nivel inferior son en realidad una forma de representar la reutilización de los módulos pero, para ser estrictos, habría que duplicar ese nodo tantas veces como fuese necesario de forma que sólo tuviese un padre.

La arquitectura sería fácilmente extensible para evolucionar más de un módulo de bajo nivel simultáneamente, lo que, en la práctica, la asimilaría a la arquitectura de Nolfi [85][84] comentada en el apartado anterior. Pero el tiempo de evolución necesario sería, sin duda, mayor, ya que el espacio de búsqueda se incrementaría notablemente. Además, aunque es cierto que permitiendo la posibilidad de evolucionar más de un módulo de bajo nivel simultáneamente se reduce el problema de una división manual de los comportamientos, no se elimina totalmente. Sigue siendo necesario especificar cuántos módulos queremos. Y si tenemos una idea de *cuántos* son necesarios es porque tenemos una idea de *cuáles* son necesarios. Por tanto, se ha considerado un camino más adecuado para evitar una división manual de comportamientos en módulos el evolucionar muchos módulos que implementen comportamientos básicos y conocidos y

dejar que el sistema determine cuáles necesita y cómo debe de usarlos antes que priorizar la evolución simultánea de un número aleatorio de módulos.

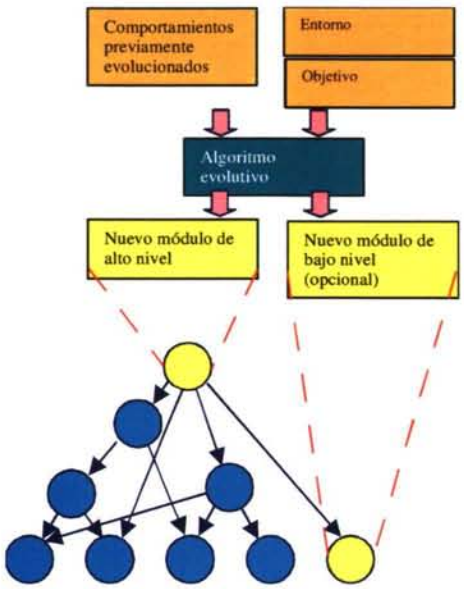


Figura 39: Esquema del funcionamiento de la evolución incremental.

Es importante recalcar que sólo los módulos de más bajo nivel pueden tomar el control de los actuadores; los restantes se limitan a determinar cómo dichos módulos toman ese control. Sin embargo, también es necesario tener en cuenta que, en la estructura de árbol que toman las arquitecturas cognitivas en este modelo, no todas las ramas tienen por qué tener la misma altura. Es decir, cada módulo de bajo nivel puede tener por encima un número distinto de módulos de mayor nivel. Las decisiones de un módulo de alto nivel pueden ser modificadas por otro módulo de alto nivel, pues no hay límite en la profundidad del árbol.

Habitualmente, cada módulo, independientemente de si es de bajo nivel o no, será una RNA. Esto no es una limitación de la arquitectura, ya que, de hecho, se han realizado pruebas usando conjuntos de reglas como módulos (ver los ejemplos de modulación de salidas en el apartado 6.2.3), sino una elección en la metodología por las consideraciones mencionadas en el apartado 4.1: la tolerancia a información imprecisa ("ruido"), el aprendizaje en vida, etc. Los módulos reciben como entradas los valores de los sensores. Estos sensores, como veremos, pueden ser sensores de los que dispone el robot, sensores ficticios que se corresponden con estados internos, energéticos o emocionales del robot, o sensores virtuales, esto es, sensores que el robot no posee y cuyo funcionamiento es simulado mediante otros que sí posee, con funciones más o menos complejas, y que le permitirán usar módulos provenientes de otros robots. Las

salidas de los módulos serán valores a introducir en los actuadores o un valor que indica qué módulo del nivel inferior debe ser seleccionado.

A continuación presentaremos varios ejemplos de comportamientos implementados mediante controladores obtenidos de forma modular en los que podremos observar el funcionamiento del mecanismo de selección.

El primer ejemplo es el ya conocido de seguir paredes con el Rug Warrior, pero en este caso lo obtendremos de forma modular en vez de hacerlo de forma monolítica para conseguir un comportamiento más robusto. Así, el controlador tendrá tres módulos: dos de bajo nivel (uno de seguir paredes y un segundo que permite escapar de colisiones cuando éstas se producen) y un módulo selector en el nivel superior. El módulo de seguir paredes ya ha sido obtenido previamente en el apartado 5.1. Es necesario obtener ahora el módulo de escapar de colisiones. Este módulo, por sí sólo, es suficiente para apreciar una primera ventaja de la metodología presentada en este trabajo. Una colisión es una situación que puede darse, potencialmente, independientemente del controlador que se esté ejecutando. Por muy bueno que sea el controlador que se tenga, siempre se puede dar un hecho inesperado que traiga como consecuencia una colisión. Por tanto, todo controlador debería ser capaz de afrontar dichas situaciones saliendo de ellas. Pero no parece razonable que, cada vez que se obtiene un controlador, se incluyan en éste los sensores de colisiones para obtener dicha capacidad. Será mucho más eficiente obtener un módulo que implemente este comportamiento una vez y dejar que, a partir de ese momento, todos los controladores nuevos puedan usarlo incorporándoles este módulo en su estructura. Así, reducimos el tiempo de obtención total del controlador global y la complejidad de cada módulo gracias a la reutilización y, además, podemos concentrarnos en obtener un controlador de salir de colisiones realmente robusto, enfrentándolo en la evolución a una gran variedad de situaciones de colisión, mucho mayor de la que podría encontrarse seguramente en la evolución de un controlador para cualquier otra tarea.

Por tanto, vamos a evolucionar el controlador de salir de colisiones. La evolución se lleva a cabo en un entorno en el que el robot aparece siempre pegado a una pared. La orientación del robot en la pared variará de tal forma que en cada evaluación dicha orientación sea distinta y en el conjunto de evaluaciones se exploren todas las posibles combinaciones de activaciones de los tres sensores de contacto del Rug Warrior. El modelo energético empleado en la evolución es el siguiente:

- El robot empieza con un nivel de energía de 1.
- En cada paso el robot pierde 0.05 puntos de energía.
- Si el robot deja de sensar colisión acaba su vida con el nivel de energía que tiene en ese momento.
- Si el robot finaliza su periodo de evaluación (10 pasos) sin dejar de sensar colisión, su nivel de energía final se rebaja a 0.

Como se puede apreciar, la calidad de un individuo que logre separarse de la pared oscilará entre 1 y 0.5, mientras que si no lo consigue, su calidad será 0. Esto se hace así, de nuevo, para proporcionar una diferencia notable en calidad a aquel individuo que logra cumplir su misión de separarse de la pared, aunque sea en el último instante, frente a aquel que es incapaz de hacerlo. En la figura 40 se pueden ver los parámetros de la red y del algoritmo evolutivo empleados, así como la gráfica de calidad (en la parte inferior izquierda).

Parámetros de la RNA	
Nº neuron. capa entrada (1)	3
Nº neuron. capa oculta (2)	6
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	-
Valor máximo de un peso	6

Parámetros del AE	
Tipo	GA/EE
Distribución inicial	Aleatoria
Prob. Cruce	0,5
Prob. Mutación	0,03
Razas	4
Poblaciones por raza	1
Generaciones	100
Migración local	10
Migración global	20
Población	64
Evaluac. por individuo	64
Pasos de vida	10

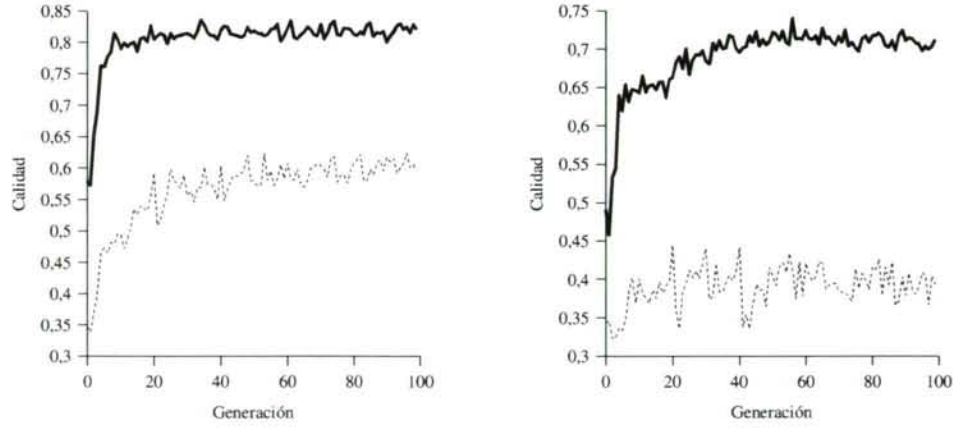


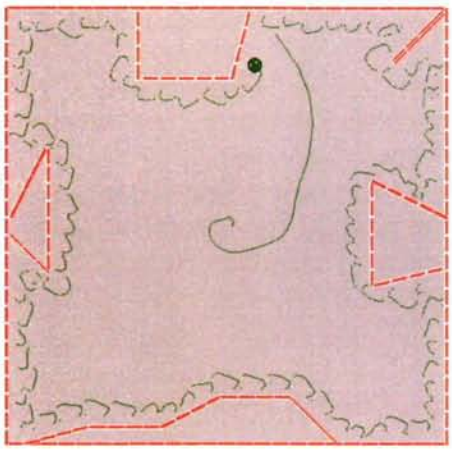
Figura 40: Comportamiento de salir de colisiones para el Rug Warrior permitiendo moverse hacia atrás (gráfica de calidad de la izquierda) y no permitiéndolo (gráfica de calidad de la derecha).

El controlador obtenido hace que, cuando la colisión es frontal, el robot se mueva hacia atrás. Esto, totalmente lógico, puede no sernos interesante, por cuanto el robot no dispone de otros sensores en la parte trasera más que el de colisiones y, por tanto, para cualquier otro comportamiento no es interesante que el robot se mueva hacia atrás. Para evitar esto, se hizo una pequeña modificación en la función de calidad, de forma que si el robot se mueve hacia atrás, pierde toda la energía de golpe y finaliza su periodo de evaluación. Con esta penalización, el robot evoluciona a un comportamiento que, en el caso de chocar con la parte delantera, rota sobre sí mismo hasta detectar la colisión en la

parte trasera (es necesario recordar que el Rug Warrior es de base circular) y entonces se aleja de la pared en línea recta. En la parte inferior derecha de la figura 40 se puede ver la calidad de una evolución con esta modificación.

Parámetros de la RNA	
Nº neuron. capa entrada (1)	5
Nº neuron. capa oculta (2)	2
Nº neuron. capa salida (3)	1
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	1
Valor máximo de un peso	6

Parámetros del AE	
Tipo	GA
Distribución inicial	Aleatoria
Prob. Cruce	1
Prob. Mutación	0,06
Razas	4
Poblaciones por raza	1
Generaciones	1000
Migración local	10
Migración global	20
Población	64
Evaluac. por individuo	16
Pasos de vida	1000



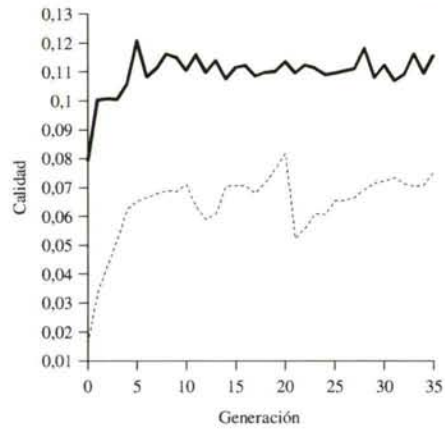


Figura 41: Comportamiento de seguir paredes con el Rug Warrior y 3 módulos.

Una vez se han obtenido los módulos de bajo nivel sólo queda obtener el módulo selector. Éste tiene 5 entradas, para los sensores de infrarrojos y de colisiones, y una única salida cuyo valor es binarizado de forma que o activa el módulo de seguir paredes o el de salir de colisiones. Las características de la RNA que implementa el módulo selector y del algoritmo evolutivo empleado se presentan en la figura 41, y el esquema del controlador global en la figura 42. El entorno es el mismo que el empleado en la obtención del comportamiento de seguir paredes monolítico. La única diferencia es que ahora, si el robot colisiona y se queda atascado, en vez de ser situado automáticamente en el centro del entorno, su periodo de evaluación finaliza. Esto se hace así para imponer una mayor penalización en el caso de colisión, de forma que el algoritmo evolutivo se dirija rápidamente hacia una solución que activa el módulo de salir de colisiones cuando es necesario por la ventaja evolutiva que esto supone.

En la figura 41 se muestra la gráfica de calidad y las características de una evolución de este comportamiento. Como se puede observar, la evolución es muy fácil, por cuanto lo único que tiene que aprender el selector es a activar el módulo de salir colisiones cuando es pertinente.

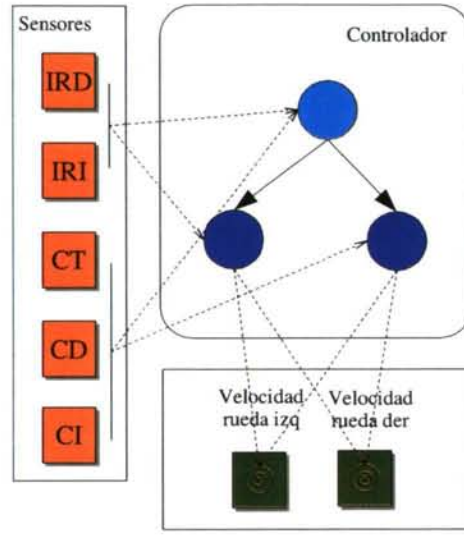


Figura 42: Esquema del controlador para los comportamientos de seguir paredes con el Rug Warrior y 3 módulos.

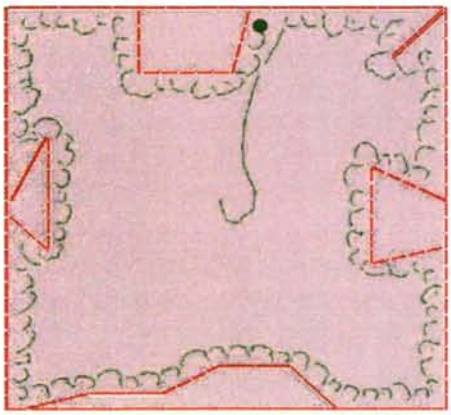
Ahora, con este mismo comportamiento, veremos cómo la arquitectura permite evolucionar simultáneamente un módulo de bajo nivel (el de seguimiento de paredes) y otro de nivel superior (un selector) para obtener el comportamiento global. Como ya se ha comentado, esto es útil cuando se quiere obtener en un mismo proceso un módulo de bajo nivel que no ha sido obtenido previamente bien porque el diseñador no tiene claro que sea necesario un módulo de bajo nivel adicional o bien, en caso de que detecte su necesidad, no sabe qué función es realmente la que éste tiene que desempeñar. Obviamente, aquí no se da ésta ultima situación, pero, como veremos, este experimento nos será muy útil para analizar las implicaciones de la coevolución de módulos, ya que sus ventajas son evidentes pero no así sus inconvenientes.

El entorno utilizado es el mismo que en el ejemplo anterior, así como la estructura del controlador (ver figura 42). Las características de la RNA del módulo selector y del algoritmo evolutivo se pueden ver en la figura 43. La única diferencia en el proceso de obtención de calidad es que el robot, mientras come, sólo aumentará su nivel de energía cuando el selector selecciona el módulo de bajo nivel que está siendo coevolucionado. Esto es necesario para acelerar el proceso evolutivo y facilitar la tarea al selector: queremos que el módulo de bajo nivel que está siendo evolucionado siga paredes, por

tanto premiemos que el selector use ese módulo para seguir paredes. En caso contrario, el proceso evolutivo puede optar por un camino en la evolución que le lleve a un selector que trate de usar el módulo de escapar de colisiones para ello y sólo implemente en el módulo de bajo nivel que está siendo evolucionado un subcomportamiento de seguir paredes que sólo se comporta correctamente cuando los movimientos del módulo de escapar de colisiones no son utilizados por el selector, dividiendo así la lógica de "seguir paredes" entre el selector y el nuevo módulo de bajo nivel.

Parámetros de la RNA	
Nº neuron. capa entrada (1)	5
Nº neuron. capa oculta (2)	2
Nº neuron. capa salida (3)	1
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	1
Valor máximo de un peso	6

Parámetros del AE	
Tipo	GA/EE
Distribución inicial	Aleatoria
Prob. Cruce	0,1
Prob. Mutación	0,06
Razas	4
Poblaciones por raza	1
Generaciones	1000
Migración local	10
Migración global	20
Población	64
Evaluac. por individuo	16
Pasos de vida	1000



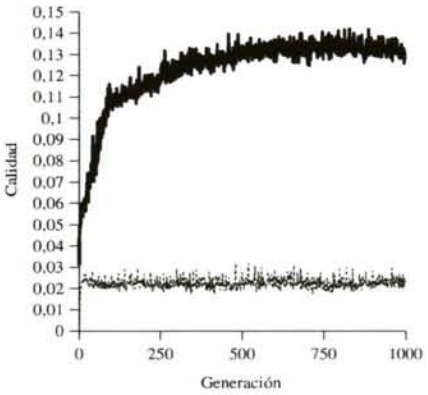


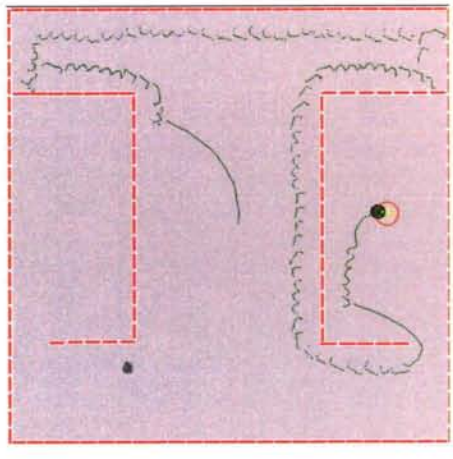
Figura 43: Comportamiento de seguir paredes para el Rug Warrior con 3 módulos y coevolución.

En la figura 43 podemos ver también la calidad y la trayectoria del robot para una evolución de este comportamiento. Como se puede observar, el comportamiento final es prácticamente idéntico, pero la evolución es mucho más dificultosa (el número de generaciones necesario es mucho mayor a la suma de las necesarias para obtener por separado los dos módulos que están siendo coevolucionados), lo cual es lógico dado que ahora el proceso evolutivo tiene que generar dos módulos en vez de uno, siendo uno de ellos bastante complejo. La conclusión es, por tanto, bastante clara: la coevolución sólo es ventajosa cuando nos falta algún módulo de bajo nivel correspondiente a algún comportamiento que no hemos sabido identificar. En caso contrario, es preferible

evolucionar los módulos incrementalmente, de uno en uno, ya que el proceso es mucho más rápido.

Parámetros de la RNA	
Nº neuron. capa entrada (1)	7
Nº neuron. capa oculta (2)	3
Nº neuron. capa salida (3)	1
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	1
Valor máximo de un peso	6

Parámetros del AE	
Tipo	GA
Distribución inicial	Aleatoria
Prob. Cruce	1
Prob. Mutación	0,04
Razas	4
Poblaciones por raza	1
Generaciones	1000
Migración local	10
Migración global	20
Población	64
Evaluac. por individuo	32
Pasos de vida	1000



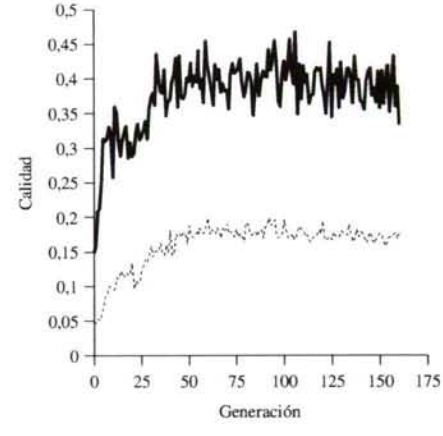


Figura 44: Comportamiento de buscar y alcanzar luz para el Rug Warrior con 4 módulos.

El siguiente ejemplo es un nuevo comportamiento de buscar y alcanzar luz que será obtenido haciendo uso de módulos de bajo nivel para dirigirse hacia la luz (ver 5.1), seguir paredes y escapar de colisiones. El esquema de este controlador puede verse en la figura 46. El entorno empleado, así como los parámetros de la RNA del selector y del algoritmo evolutivo pueden verse en la figura 44. La RNA evolucionada tiene como entradas los sensores de infrarrojos, de luz y de colisiones, y una salida que selecciona el módulo de bajo nivel a activar. El robot empieza cada evaluación desde el centro del entorno, y la luz se encontrará ubicada en el centro de una de las dos habitaciones seleccionada aleatoriamente. El robot, por tanto, deberá buscar primero la luz para luego dirigirse a ella. La función de calidad es la misma que la empleada para el comportamiento de alcanzar luz. Si el entorno fuese mucho más complejo, es probable que hubiese que modificar la función de calidad de alguna forma para el caso de que ningún individuo de la población inicial fuese capaz de alcanzar la luz y, por lo tanto, no hubiese gradiente alguno en las calidades para guiar el proceso evolutivo. En este comportamiento, la posibilidad de evolucionar simultáneamente el módulo de dirigirse a

la luz al mismo tiempo que el selector no es una buena opción, por cuanto dicho módulo sólo se tiene que activar al final de la ejecución, cuando el robot ya es capaz de localizarla, y la luz se aborda siempre prácticamente desde el mismo punto, con lo que el módulo de alcanzar luz sería menos robusto que el obtenido individualmente con un número de situaciones posibles mucho mayor. Sí se podría coevolucionar, sin embargo, un módulo adicional de forma que el algoritmo evolutivo pudiese dar lugar a comportamientos de búsqueda más eficientes que la selección entre los ya existentes.

Si observamos la gráfica de calidad de la figura 44 vemos que la evolución es sencilla y en unas 60 generaciones tenemos ya el controlador final, cuyo comportamiento básico es el esperado: usar los módulos de seguir paredes y escapar de colisiones para buscar la luz y el de alcanzar luz para llegar a ella una vez está dentro de la habitación. En la figura 45 podemos ver el comportamiento sobre el robot real.

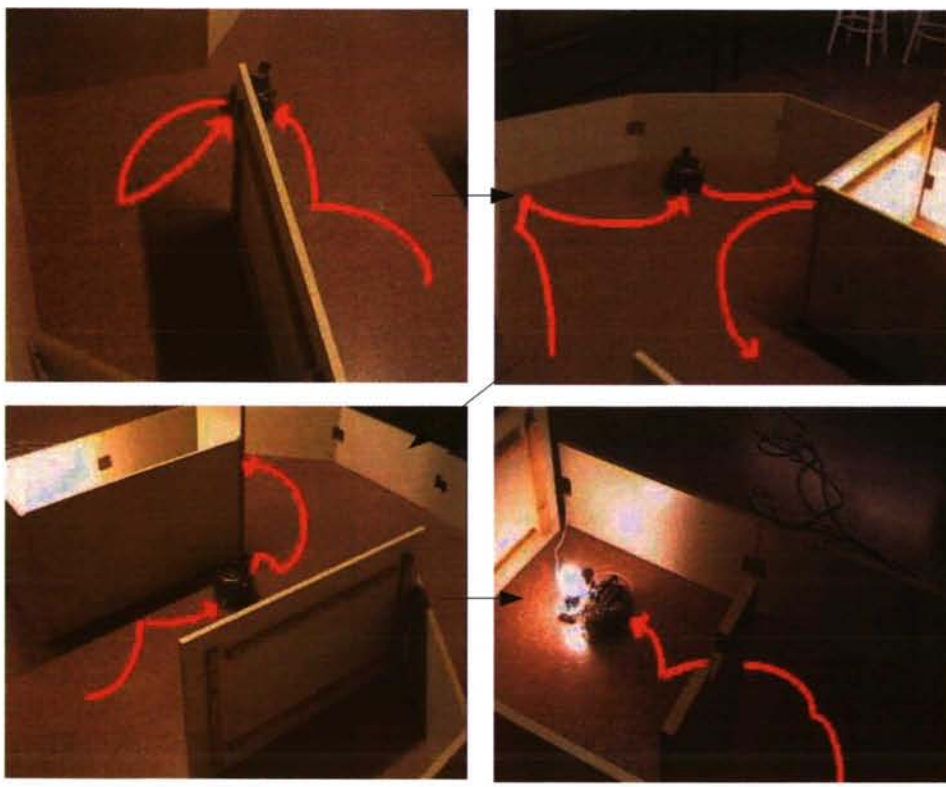


Figura 45: Comportamiento de buscar y alcanzar luz sobre el robot real para el Rug Warrior con 4 módulos.

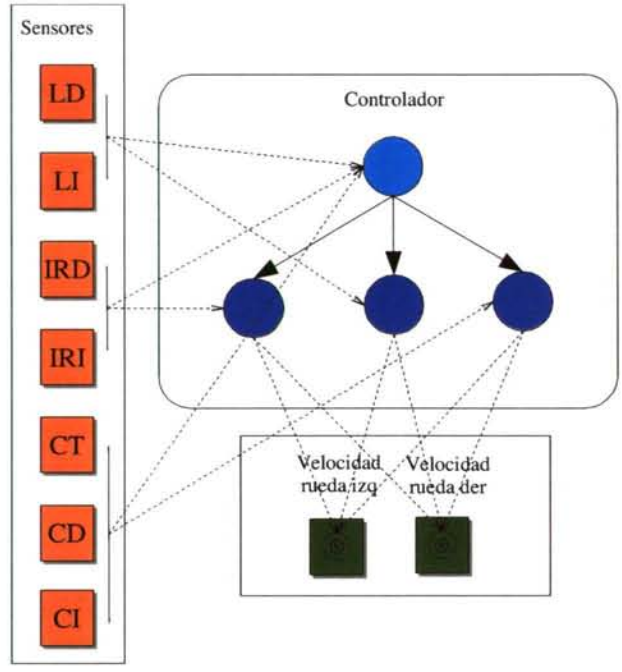


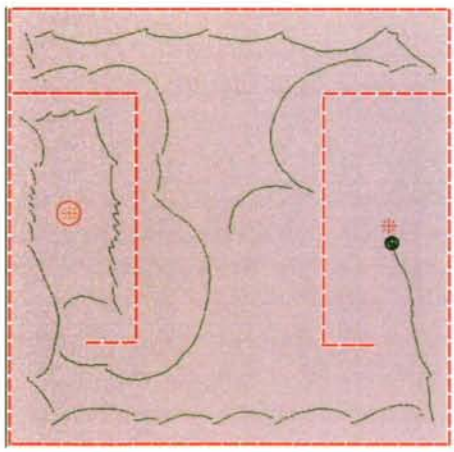
Figura 46: Esquema del controlador del comportamiento de buscar y alcanzar luz, y también del comportamiento de homing, con 4 módulos para el Rug Warrior.

El último ejemplo nos permitirá comprobar cómo, a veces, el diseñador se puede equivocar en planear la estrategia a seguir por el robot, con lo que se aprecia la ventaja de la determinación automática de las conexiones entre los módulos. Este comportamiento que veremos a continuación es el mismo comportamiento de *homing* visto previamente en el apartado 5.1, pero ahora tanto la trampa como la comida están escondidas en habitaciones de un entorno como el empleado en el ejemplo anterior, aunque con las puertas de las habitaciones más grandes. La función de calidad empleada es la misma que para el comportamiento de *homing* simple. El controlador tendrá 3 módulos de bajo nivel (escapar de colisiones, seguir paredes y *homing* simple) y un selector que evolucionaremos ahora (ver esquema en figura 46), que tendrá como entradas los sensores de infrarrojos, luz y colisiones y una salida que activará el módulo de bajo nivel correspondiente. Los parámetros de la RNA evolucionada y del algoritmo evolutivo están en la figura 47. En esa misma figura tenemos los datos de la evolución de la calidad del mejor individuo y un ejemplo del comportamiento del robot con el controlador obtenido en una evolución. Respeto a la calidad poco hay que decir, la evolución es sencilla. Lo interesante está en el comportamiento final obtenido. El selector no usa los módulos de seguir paredes y salir de situaciones de colisión para localizar su hogar, sino que usa los de *homing* simple y salir de situaciones de colisión.

Mientras no detecta paredes usa el módulo de *homing* que, en ausencia de luz, describe un ligero arco hacia la derecha. En cuanto detecta una pared, usa el módulo de salir de colisiones hasta que deja de detectarla, aprovechando la característica de este módulo de que, en ausencia de colisión, gira bruscamente a la izquierda. Cuando el robot entra en la habitación donde se encuentra la trampa, el módulo de *homing* lo mantiene a una distancia prudencial de ella. Finalmente, cuando entra en la habitación donde se encuentra su lugar de reposo, nuevamente el módulo de *homing* toma el control y lo guía en este caso hacia su hogar.

Parámetros de la RNA	
Nº neuron. capa entrada (1)	7
Nº neuron. capa oculta (2)	3
Nº neuron. capa salida (3)	1
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	1
Valor máximo de un peso	6

Parámetros del AE	
Tipo	GA
Distribución inicial	Aleatoria
Prob. Cruce	1
Prob. Mutación	0,04
Razas	4
Poblaciones por raza	1
Generaciones	1000
Migración local	10
Migración global	20
Población	64
Evaluac. por individuo	16
Pasos de vida	1000



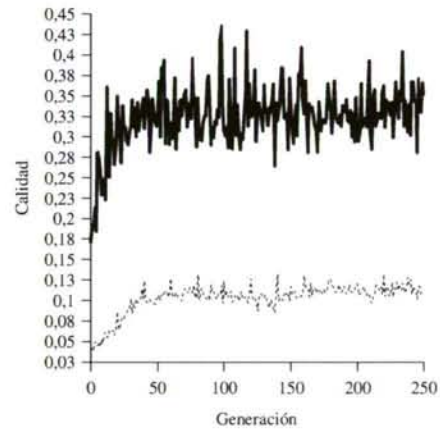


Figura 47: Comportamiento de homing para Rug Warrior con 4 módulos.

Usando el entorno del comportamiento anterior, con las puertas más estrechas, este comportamiento de *homing* usaba, como el otro, el módulo de seguir paredes para localizar el hogar. Esto nos muestra cómo el algoritmo evolutivo, haciendo uso de características de los módulos que pueden pasar inadvertidas para el diseñador, puede encontrar mejores soluciones a un problema determinado que las que el diseñador pudiera indicar manualmente, lo que es un punto a favor, a sumar al de la mayor automatización del proceso, de la obtención automática de conexiones entre módulos.

Tanto este comportamiento como el anterior, pero especialmente éste, serían muy complicados de obtener monolíticamente, ya que primero tendrían que aprender a recorrer el entorno buscando las luces sin que eso supusiese ninguna ganancia en términos de calidad y posteriormente aprender a dirigirse a la luz correcta. Además, afrontan muchas menos situaciones frente a las luces de las que han afrontado los módulos de bajo nivel existentes, con lo que serán menos robustos.

Si sustituimos el módulo de seguir paredes monolítico por el de dos niveles comentado en este mismo apartado, tendremos un controlador de *homing* de 3 niveles como el presentado en la figura 48, lo cual es interesante si la evolución es en el entorno con las puertas estrechas, ya que el seguimiento de paredes es ahora más fiable.

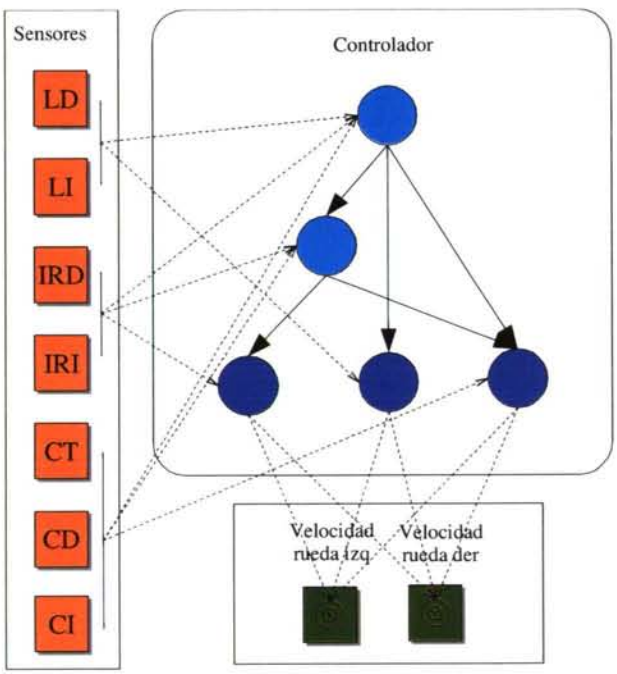


Figura 48: Esquema del comportamiento de homing con 5 módulos para el Rug Warrior.

En este apartado hemos visto el funcionamiento del mecanismo de selección, que permite obtener controladores más eficientes (caso del de seguir paredes), controladores que serían casi imposibles de obtener monolíticamente (como el de *homing* compuesto) y, en general, cómo, reutilizando módulos, obtener controladores para comportamientos nuevos puede ser un proceso muy rápido. Hemos visto también las ventajas de que la obtención de las conexiones entre módulos sea automática (más cómodo para el diseñador al automatizar el proceso y evitar errores de apreciación por su parte que pueden dar lugar a comportamientos subóptimos) y cómo es posible substituir módulos

en controladores ya existentes siempre y cuando los comportamientos que implementan sean similares.

6.2.2 Transferencia de comportamientos entre robots de distinta configuración física

Como ya se ha comentado previamente, la posibilidad de reutilización de módulos o grupos de módulos que implementan comportamientos es muy importante a la hora de poder obtener controladores complejos en robots autónomos. Sin esa posibilidad, la obtención de cada controlador se convierte en una tarea muy costosa cuando no imposible. Pero no sólo cabe la posibilidad de reutilizar módulos para implementar distintos comportamientos en un mismo robot, ya que también sería deseable la reutilización entre robots con distintas configuraciones físicas.

La mayor parte de las veces los controladores se obtienen para un determinado tipo de robot en un determinado entorno. Un intento de aumentar la reusabilidad de los controladores lo supone el trabajo de Floreano y Mondada en [34], donde los autores evolucionan controladores para un robot Khepera que posteriormente son usados como "semillas" en la evolución de controladores para un robot Koala. En pocas generaciones obtienen controladores válidos para el segundo robot. Sin embargo, ambos robots son similares en términos de capacidades sensoriales y actuadoras y poseen el mismo software, simplemente uno es de tamaño mayor al otro.

En este trabajo se ha abordado la transferencia de controladores entre robots de distinto tipo usando los conceptos de sensor y actuador virtuales. Definimos un sensor virtual como un módulo que transforma datos provenientes de uno o más sensores existentes en el robot, de forma que esos datos transformados simulan el comportamiento de otro sensor que no está presente en dicho robot. Y definimos un actuador virtual como un módulo que transforma comandos destinados a ser introducidos en actuadores que no existen en el robot en comandos que serán introducidos en otros actuadores que sí existen en ese robot (es posible que un comando puede traducirse en una secuencia de comandos y viceversa). De esta forma, se puede llevar a cabo la tarea asociada al controlador en un robot distinto al utilizado inicialmente para obtenerlo.

Por consistencia con la metodología, tanto sensores como actuadores virtuales pueden ser implementados con RNAs y obtenidos de forma automática, como cualquier otro módulo. En este caso, en lugar de evolución, puede ser interesante usar aprendizaje, pues la tarea a realizar por el módulo es una simple transformación entre dos espacios conocidos. Cuando los sensores o actuadores virtuales son muy sencillos también es posible implementarlos manualmente, utilizando conjuntos de reglas o cualquier otra herramienta que se considere adecuada. Aun en el caso de utilizar RNAs y obtener los sensores/actuadores virtuales automáticamente, es obvio que el proceso de virtualización introduce un factor humano: la elección de cuáles serán las salidas de los sensores virtuales o las entradas de los actuadores virtuales, es decir, cuáles son las sensorizaciones estándar a considerar por el resto de la arquitectura de control y qué salidas estándar proporciona esa arquitectura de control. Esta elección, sin embargo,

suele ser bastante obvia y la única restricción que impone el diseñador al sistema es la propia de la reducción en el espacio sensorial que este proceso acarrea, ya que lo habitual es simular un tipo de sensor con otro mejor (como veremos en nuestros ejemplos), ya que lo contrario no suele ser posible. Si la información sigue siendo suficiente no hay problema, y si no lo es probablemente la virtualización no sea posible para ese controlador.

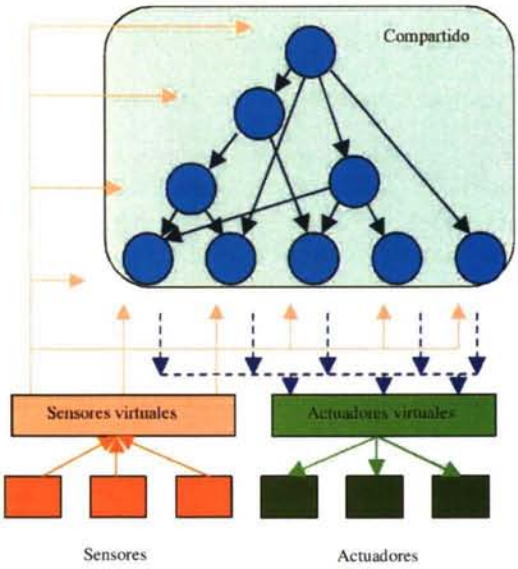


Figura 49: Arquitectura para trasladar un controlador compuesto por varios módulos entre distintos tipos de robots.

Hemos definido dos formas de introducir sensores/actuadores virtuales en la arquitectura. Por una parte se pueden acoplar de forma que funcionen como interfaz entre los sensores/actuadores reales presentes en el robot y las entradas/salidas de los módulos de un controlador obtenido para otro robot morfológicamente distinto y que están diseñados para interaccionar con el entorno utilizando un conjunto de sensores/actuadores diferente. Así, podemos migrar un controlador íntegramente de un robot a otro simplemente introduciendo los sensores/actuadores virtuales que sea necesario. En la figura 49 podemos ver un esquema que representa esta posibilidad.

La otra alternativa es introducir sensores virtuales sólo en los módulos de alto nivel, aquellos que no toman el control de los actuadores sino que toman decisiones, y obtener módulos específicos de bajo nivel para cada robot. En la figura 50 tenemos el esquema para este otro mecanismo de transferencia de controladores.

La primera alternativa permite una migración más rápida de los controladores de un robot a otro, mientras que la segunda permite obtener en ocasiones unos

comportamientos más precisos y que hacen un mejor uso de todas las capacidades sensoriales y actuadoras de cada robot. A continuación introduciremos ejemplos de ambas alternativas de virtualización que mostrarán las posibilidades de cada una, así como esta última afirmación al respecto del mejor aprovechamiento de las capacidades sensoriales y actuadoras de cada robot en el segundo caso.

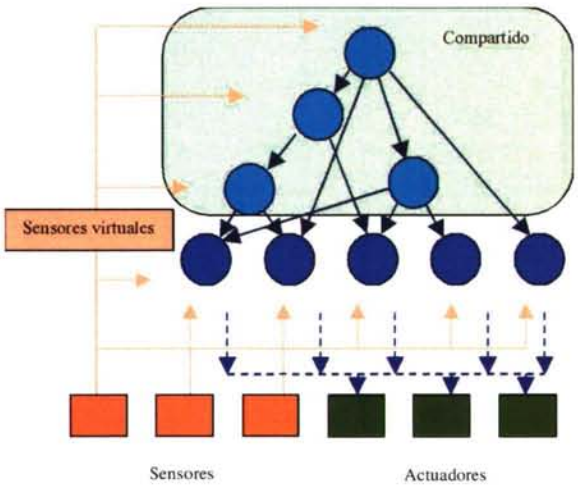


Figura 50: Arquitectura para trasladar un comportamiento entre distintos tipos de robot cuando los módulos de bajo nivel son particulares a cada uno.

El primer ejemplo se trata de un comportamiento de seguimiento de luces obtenido en primera instancia para el Rug Warrior y que habíamos presentado en el apartado 5.1 al explicar la mejora que introducía en algunos comportamientos el tratamiento de información temporal. El comportamiento está formado por un único módulo y, una vez obtenido el controlador, éste se convertirá en un controlador que implementará un comportamiento de seguimiento del objeto más cercano para el Pioneer 2-DX mediante el uso de sensores y actuadores virtuales. Se crean dos sensores virtuales para el Pioneer 2-DX que relacionan las entradas de los sónares frontales con las salidas que proporcionarían los sensores de luz del Rug Warrior si éste viera un objeto luminoso a la misma distancia que el Pioneer 2-DX ve el objeto más cercano. Se usan los cuatro sónares delanteros izquierdos para simular el sensor de luz izquierdo y los cuatro sónares delanteros derechos para hacer lo mismo con el sensor de luz derecho, de forma que aquel sónar que detecta un objeto más cercano es el que se usa para transformar su valor en el que proporcionaría el sensor de luz del Rug Warrior. Los actuadores virtuales simplemente convierten los comandos que se le transmiten al Rug Warrior (velocidad para la rueda izquierda y para la rueda derecha) en los comandos que recibe el Pioneer 2-DX (velocidad lineal y velocidad angular). Tanto los sensores

como los actuadores virtuales son muy sencillos, simples ecuaciones, con lo que se han implementado manualmente.

Es necesario tener en cuenta que tanto los sensores como los actuadores del Pioneer 2-DX son mucho más precisos y menos ruidosos que los del Rug Warrior, con lo que no es necesario tener en cuenta el ruido en la elaboración de los sensores/actuadores virtuales, ya que el controlador proveniente del Rug Warrior ya ha sido obtenido para tratar con mucho más ruido del que se encontrará el Pioneer 2-DX. Otro caso muy distinto sería si el proceso de virtualización fuese en sentido inverso, es decir, adaptar un controlador de un robot con una sensorización/actuación determinada a otro robot con sensorización /actuación más ruidosa. En ese caso, el proceso de obtención de los sensores/actuadores virtuales sería mucho más complejo y puede que incluso imposible, ya que los sensores/actuadores virtuales deberían tratar de atenuar un ruido que no estuvo presente en la obtención del controlador ya existente y, por tanto, para el cual no está preparado.

En la figura 51 podemos ver el comportamiento del Pioneer 2-DX utilizando el controlador de alcanzar luz del Rug Warrior y los sensores y actuadores virtuales aquí descritos. Vemos cómo el Pioneer 2-DX sigue a una persona, el "objeto" más cercano en el rango de alcance de sus sensores de sónar frontales, de la misma forma que el Rug Warrior trataría de alcanzar a un objeto luminoso.

Figura 51: Comportamiento sobre el robot real de alcanzar luz, obtenido para el Rug Warrior y trasladado al Pioneer 2-DX en forma de comportamiento de seguir al objeto más cercano mediante la utilización de sensores de luz virtuales.

El segundo ejemplo de virtualización que consideraremos es el seguimiento de paredes compuesto para el Rug Warrior que estudiamos en el apartado 6.2.1 y que será

adaptado para su uso en el Pioneer 2-DX siguiendo las dos alternativas comentadas previamente, lo cual nos permitirá apreciar las ventajas e inconvenientes de cada una de ellas.

En el primer caso se migra el controlador completamente, obteniendo para ello los correspondientes sensores y actuadores virtuales. Los actuadores virtuales son los mismos mencionados en el ejemplo anterior. Los sensores virtuales son cinco: dos de infrarrojos y tres de contacto. Todos ellos se simulan con los sónares del Pioneer 2-DX. Para los sensores de infrarrojos izquierdo y derecho se han tomado los sónares 1 y 2 y los 5 y 6 respectivamente (ver figura 88), ya que con ellos se cubre una abertura similar en grados a la del Rug Warrior con sus sensores de infrarrojos. Dado que los sensores de infrarrojos del Rug Warrior son binarios, sólo dicen si hay objeto o no, pero no dicen la distancia, los sensores virtuales también proporcionan una salida binaria. En este caso, el umbral se ha establecido en 75 centímetros, el equivalente al alcance máximo del Rug Warrior teniendo en cuenta el mayor tamaño del Pioneer 2-DX. Algo similar sucede con los sensores de contacto. El Rug Warrior tiene tres y en función de cuáles se activan el robot sabe por dónde ha colisionado. Por tanto, no es suficiente que el Pioneer 2-DX use sus *encoders* para simular los sensores de contacto del Rug Warrior, ya que esa información no es suficiente para detectar el origen de la colisión. Así, para simular los sensores de contacto se ha empleado un mecanismo similar al caso de los sensores de infrarrojos, usando los sónares 1 a 3 para el sensor de contacto izquierdo, los 4 a 6 para el derecho y los 9 a 14 para el trasero, siendo en este caso el umbral de distancia 20 centímetros.

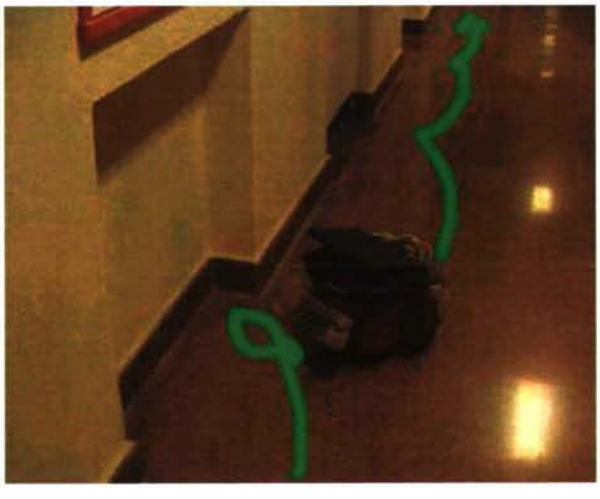


Figura 52: Comportamiento de seguir paredes sobre el robot real, obtenido para el Rug Warrior y trasladado al Pioneer 2-DX mediante sensores virtuales (infrarrojos y de contacto) y actuadores virtuales.

Como se puede observar en la figura 52, el Pioneer 2-DX realiza un movimiento idéntico al realizado por el Rug Warrior. Sin embargo, el Rug Warrior evolucionó para salir de las situaciones de colisión girando en redondo sobre sí mismo, lo cual es una buena opción teniendo en cuenta que el robot es redondo. Pero el Pioneer 2-DX es rectangular, con lo que en ocasiones no es capaz de salir de las situaciones de colisión con este controlador. La otra peculiaridad es que el Pioneer 2-DX se ve obligado a realizar la misma estrategia de seguimiento de paredes que el Rug Warrior: un movimiento zigzagueante debido a esa naturaleza binaria de los sensores de infrarrojos, cuando en realidad recibiendo la información directamente de los sónares podría hacerlo mejor.

Paramétros de la RNA	
Nº neuron. capa entrada (1)	16
Nº neuron. capa oculta (2)	6
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	-
Valor máximo de un peso	6

Parámetros del AE	
Tipo	MA
Distribución inicial	Aleatoria
Prob. Cruce	-
Prob. Mutación	-
Razas	2
Poblaciones por raza	1
Generaciones	1000
Migración local	40
Migración global	80
Población	100
Evaluac. por individuo	8
Pasos de vida	100

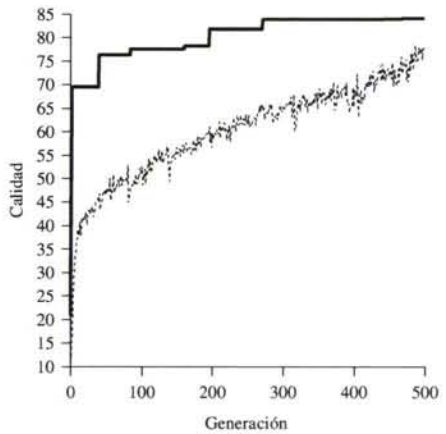


Figura 53: Parámetros y calidad de la evolución del comportamiento de salir de colisiones para el Pioneer 2-DX.

En la segunda parte de este experimento, los módulos de bajo nivel hacen uso de las características sensoriales propias del Pioneer 2-DX, mientras que el de alto nivel es reutilizado y usa los sensores virtuales ya comentados. Así, el módulo de seguir paredes es el visto en el apartado 4.4.3 mientras que el módulo de salir de colisiones fue obtenido para este ejemplo. En la figura 53 se pueden ver los datos de este módulo. Este controlador es fácil de obtener y la solución fue alcanzada mucho antes de agotar las 1000 generaciones que se establecieron como parámetro de duración de la evolución. La función de calidad empleada para el módulo de salir de colisiones fue la siguiente:

- El robot comienza su evaluación con un nivel de energía de 100 en estado de colisión con una pared (escogida al azar entre 8 posiciones distintas).
- En el entorno hay veneno cerca de la pared, de forma que mientras que el robot no se separe de la pared estará perdiendo energía.
- La evaluación se detiene en el momento en el que el robot sale de la zona con veneno o expira su tiempo de ejecución.

Como se aprecia en la figura 54, el resultado es ahora un comportamiento óptimo, al devolver los sónares la distancia a la que se encuentran los objetos y ser capaz así de salir de todas las situaciones de colisión, ya que la estrategia es ahora más adecuada para la fisiología del robot: primero anda hacia atrás girando en sentido contrario al de la pared y luego continúa hacia adelante.

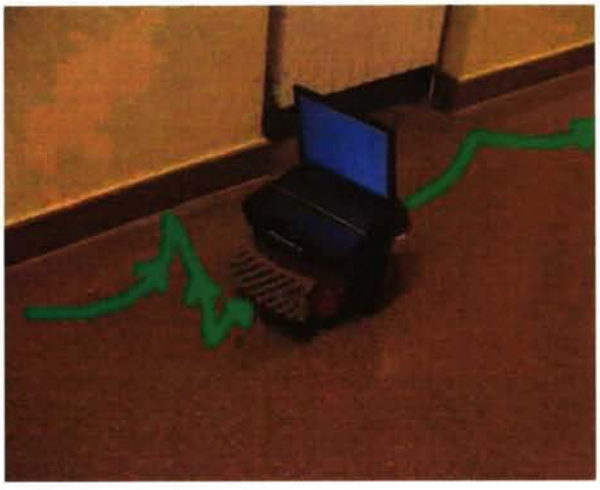


Figura 54: Comportamiento sobre el robot real de seguir paredes para el Pioneer 2-DX, utilizando sensores virtuales (infrarrojos y de contacto) para el módulo de alto nivel de la arquitectura obtenida para el Rug Warrior, y comportamientos de bajo nivel con la sensorización del propio Pioneer 2-DX.

La elección, por tanto, acerca de en qué nivel introducir la sensorización virtual depende de qué es lo que se pretende conseguir. Si se pretende migrar rápidamente un controlador de un robot a otro y el comportamiento se considera adecuado lo más rápido es introducir la sensorización virtual a todos los niveles. Si, por el contrario, se quiere mejorar el comportamiento y el robot destinatario del controlador posee una mejor capacidad sensorial que el robot para el cual originalmente se obtuvo el controlador, es más adecuado evolucionar de nuevo los módulos de bajo nivel y usar los sensores virtuales en los de alto nivel.

6.2.3 Modulación

Como se ha indicado en el capítulo 3, una de las definiciones básicas de la aproximación a la robótica que estamos considerando es que los comportamientos pueden verse como "parejas estímulo-respuesta, moduladas por la atención y determinadas por la intención" (Arkin [7]). Por tanto, ya Arkin apuntaba a la modulación como mecanismo para la adaptación de los comportamientos a circunstancias perceptuales cambiantes. En la propia naturaleza hay multitud de ejemplos en los que la modulación tiene lugar y es útil. Como veremos a continuación, tal modulación puede llevarse a cabo también en nuestra arquitectura de control y, nuevamente, con claras ventajas.

La primera cuestión a resolver es qué se entiende por modulación. En este trabajo entenderemos modulación como una "modificación leve" de la funcionalidad de un módulo de un controlador, esto es, la correspondencia que establece entre estímulos y respuestas. La noción de "modificación leve" es necesaria, ya que no queremos que la modulación sea cualquier tipo de modificación, lo que nos llevaría que a partir de cualquier función podríamos obtener otra función totalmente distinta (lo que no nos interesa, ya que para obtener una función totalmente distinta sería mejor obtenerla partiendo de cero), sino que sea una alteración de la funcionalidad de un módulo para adaptar éste a cambios en el entorno, entendiendo éste en toda su globalidad, sin que dicha alteración produzca un cambio en la etiqueta que el diseñador le haya atribuido visualmente al comportamiento implementado por dicho módulo.

Entre los autores que han realizado algún trabajo en modulación están Husbands y col. [50], que utilizan "redes de gases" en las cuales los nodos en una red distribuida espacialmente pueden emitir "gases". Este procedimiento imita la difusión de óxido nítrico, que actúa como neurotransmisor, participando en el mundo biológico en diferentes procesos de modulación. En la red artificial, los "gases" se difunden a través de la red y modulan la función de transferencia de los nodos (cambiando la pendiente de la función sigmoide de los mismos) dependiendo de la concentración del gas en el punto espacial en el que se sitúa el nodo. Los autores aplican este controlador neuronal a una tarea de discriminación de dos objetos diferentes, necesitando muchas menos generaciones en la evolución, típicamente un orden de magnitud menos, para obtener unos resultados similares a los obtenidos con redes neuronales con nodos binarios.

Ishiguro y col. [51] presentan un trabajo similar en el cual los nodos de la red emiten neuromoduladores, pero utilizando receptores específicos para localizar su efecto en una neurona o en una sinapsis. Tal efecto está codificado en el genotipo que define de qué forma modificar las propiedades de las sinapsis, como por ejemplo su peso de conexión, incluyendo la posibilidad de eliminarla, activarla o cambiar su naturaleza de sinapsis excitadora o inhibidora, mientras que en las celdas se determina su umbral de activación. Los autores utilizan la estructura de modulación para el control

de movimiento de un organismo simulado en dos dimensiones que se mueve utilizando una única pata con dos grados de libertad.

Los ejemplos anteriores realizan una modulación en un único módulo o RNA. Otro ejemplo, ya con diferentes estructuras de redes, lo constituye la propuesta de redes moduladoras y moduladas (Duro y Santos [29]), en la cual una red o redes modulan el comportamiento de otras RNAs influenciando los pesos de sus conexiones. Los autores han aplicado la estructura al reconocimiento de patrones en 2D bajo diferentes rotaciones.

En esta misma línea, Meyer y col. [78] presentan un ejemplo de modulación en el control de un robot Khepera. Primeramente evolucionan un módulo neuronal para desplazarse evitando obstáculos, con luz ambiente de intensidad media, para posteriormente evolucionar un segundo módulo que, en función del nivel de luz ambiente, modula al primero realizando conexiones a los nodos neuronales de éste, para que se siga realizando el comportamiento. Los autores presentan un ejemplo similar para el movimiento y evitación de obstáculos para un robot hexápodo, en el que un primer módulo genera un comportamiento de caminar en línea recta, mientras que un segundo módulo obtenido a posteriori modula al primero cuando se detecta la presencia de un objeto.

Dentro de la arquitectura establecida en este trabajo distinguiremos tres tipos de modulación:

- Modulación mediante la alteración de las entradas al módulo (datos recibidos de los sensores). La posibilidad de realizar este tipo de modulación se introduce en la arquitectura mediante un módulo específico, que denominaremos modulador de sensores. Este tipo de módulo podrá tener como entradas, al igual que cualquier otro módulo, cualquier tipo de sensor, incluyendo sensores virtuales y variables de estado interno del robot. Sus salidas modularán los sensores de sus nodos descendientes (utilizando la representación en árbol del controlador vista en el apartado 6.2.1), mediante su multiplicación por los valores generados por dichos sensores. Un modulador de sensores puede tener uno o más hijos. Su número de salidas será el número de sensores que se desea modular multiplicado por el número de hijos, ya que se pretende que los valores de modulación puedan ser distintos para cada uno de sus hijos. El modulador no sólo puede modular los sensores de sus nodos hijos, sino también de todos los descendientes de estos hijos, ya que sus modulaciones se propagan por todos ellos. Es decir, todos aquellos nodos descendientes del modulador, sea cual sea el grado en la descendencia, si tienen un sensor del tipo que es modulado por el modulador, verán su valor afectado por la modulación. A diferencia de los módulos de selección, que escogían un único hijo para ser ejecutado, todos los hijos de un modulador se ejecutarán. Dadas las características en la definición de controlador en este trabajo, cada hijo de un modulador podrá ejecutarse, si así se desea, en paralelo, ya que la

ejecución de un nodo y sus descendientes no tiene efectos colaterales en la ejecución de sus hermanos y los descendientes de éstos. Un controlador puede tener tantos módulos de este tipo como sean necesarios y en cualquier punto del árbol que representa su estructura, con la única restricción de no poder ser un nodo hoja que, recordemos, son aquellos nodos que toman el control de los actuadores. Si en un camino del controlador hay más de un modulador que module los mismos sensores, dichos valores de modulación se multiplican. Por ejemplo, supongamos que un modulador modula un sensor "x" con un valor "y" y su hijo modula ese mismo sensor con un valor "z". Si el nieto usa ese sensor, su valor vendrá multiplicado por "y*z". Más formalmente, la modulación de entradas puede expresarse de la siguiente forma:

Sean:

$f_u(s_1,s_2,...,s_n)$ *el valor para el actuador* **u** *calculado por el módulo* **f** *ante las entradas* $(s_1,s_2,...,s_n)$

$m_i(s_1',s_2',...,s_m')$ *el valor de modulación para el sensor* **i** *calculado por el módulo* **m** *ante las entradas* $(s_1',s_2',...,s_m')$

$f_u^m(s_1,s_2,...,s_n)$ *el valor para el actuador* **u** *tras la modulación de* **f** *por* **m** *ante las entradas* $(s_1,s_2,...,s_n)$

entonces:

$$f_u^m(s_1,s_2,...,s_n)=f_u(m_1(s_1',s_2',...,s_m')*s_1,m_2(s_1',s_2',...,s_m')*s_2,...,m_n(s_1',s_2',...,s_m')*s_n)$$

- Modulación mediante la alteración del procesamiento interno del módulo. Consiste en alterar el funcionamiento de un módulo modificando su procesado interno. Este tipo de modulación no ha sido contemplada en este trabajo, aunque es posible el análisis de sus posibilidades en un trabajo futuro.

  Sean:

  $f_u(s_1,s_2,...,s_n)$ *el valor para el actuador* **u** *calculado por el módulo* **f** *ante las entradas* $(s_1,s_2,...,s_n)$

  $m(s_1',s_2',...,s_m')$ *el valor de modulación para el módulo* **f** *calculado por el módulo* **m** *ante las entradas* $(s_1',s_2',...,s_m')$

  $f_u^m(s_1,s_2,...,s_n)$ *el valor para el actuador* **u** *tras la modulación de* **f** *por* **m** *ante las entradas* $(s_1,s_2,...,s_n)$

  entonces:

  $$f_u^m(s_1,s_2,...,s_n)=(f_u \circ m(s_1',s_2',...,s_m'))(s_1,s_2,...,s_n)$$

- Modulación mediante la alteración de las salidas del módulo (valores a introducir en los actuadores). Este tipo de modulación es muy similar a la modulación sobre sensores excepto que trabaja sobre los actuadores. Existe, al igual que antes, un tipo de módulo destinado a esta tarea (modulador de actuadores) que puede tener

también cualquier tipo de sensores como entrada y que puede estar ubicado en cualquier punto del árbol que representa la estructura del controlador, excepto en una hoja, y puede hacerlo tantas veces como haga falta. Sus salidas son en este caso valores que modularán los actuadores de todos sus descendientes que sean nodos hoja mediante una operación de multiplicación. Su número de salidas será el número de hijos por el número de actuadores que se desea modular, porque también aquí se quiere poder propagar modulaciones distintas a través de los diferentes hijos. Los hijos se ejecutarán todos y los resultados de la modulación se propagarán por sus descendientes de la misma forma que los moduladores de sensores, multiplicándose si existe en un camino más de un modulador que module los mismos actuadores.

Sean:

$f_u(s_1,s_2,...,s_n)$ *el valor para el actuador* **u** *calculado por el módulo* **f** *ante las entradas* $(s_1,s_2,...,s_n)$

$m_u(s_1',s_2',...,s_m')$ *el valor de modulación para el actuador* **u** *calculado por el módulo* **m** *ante las entradas* $(s_1',s_2',...,s_m')$

$f_u^m(s_1,s_2,...,s_n)$ *el valor para el actuador* **u** *tras la modulación de* **f** *por* **m** *ante las entradas* $(s_1,s_2,...,s_n)$

entonces:

$$f_u^m(s_1,s_2,...,s_n)=m_u(s_1',s_2',...,s_m')*f_u(s_1,s_2,...,s_n)$$

En la figura 55 se puede ver un ejemplo de cómo sería un controlador ya con todos los elementos que se han definido en este trabajo. Hay que resaltar varias propiedades interesantes. Los nodos selectores son equivalentes a moduladores de actuadores que propagan un valor de activación de 1 para todos los actuadores en el hijo seleccionado por el nodo selector y de 0 para todos los actuadores en los restantes hijos para cada posible conjunto de valores para las entradas del nodo selector. Para que esto sea así se requiere que un nodo no pueda tener dos padres. Así, es preciso recordar que cuando en la representación de un controlador aparece tal circunstancia (como en la figura 55 para el nodo modulador de sensores), se trata de una simplificación representacional indicando que los nodos son idénticos, pero a efectos de la lógica del controlador es como si fueran distintos y dicha lógica puede pasar por el nodo en más de una ocasión (hasta un máximo de tantas veces como caminos haya desde la raíz) con lo que modulaciones de antecesores que se encuentren en distintos caminos no interferirán unas con otras. Esto, además, posibilita la propiedad comentada con anterioridad de que cada hijo y sus descendientes puede ejecutarse en paralelo respecto a sus hermanos y los descendientes de éstos. Otra característica que es necesario comentar es que, desde el momento en el que hay al menos un modulador, ya sea de sensores o actuadores, con más de un hijo, más de un módulo de actuación será ejecutado, con lo que es preciso definir qué sucede cuando los que se ejecutan introducen valores en los mismos actuadores. Hemos elegido que los valores para los

actuadores que proporcionen los módulos de actuación se vayan sumando en acumuladores asociados a cada actuador. Sólo cuando todos los módulos de actuación hayan sido evaluados, los valores finales serán introducidos en los actuadores.

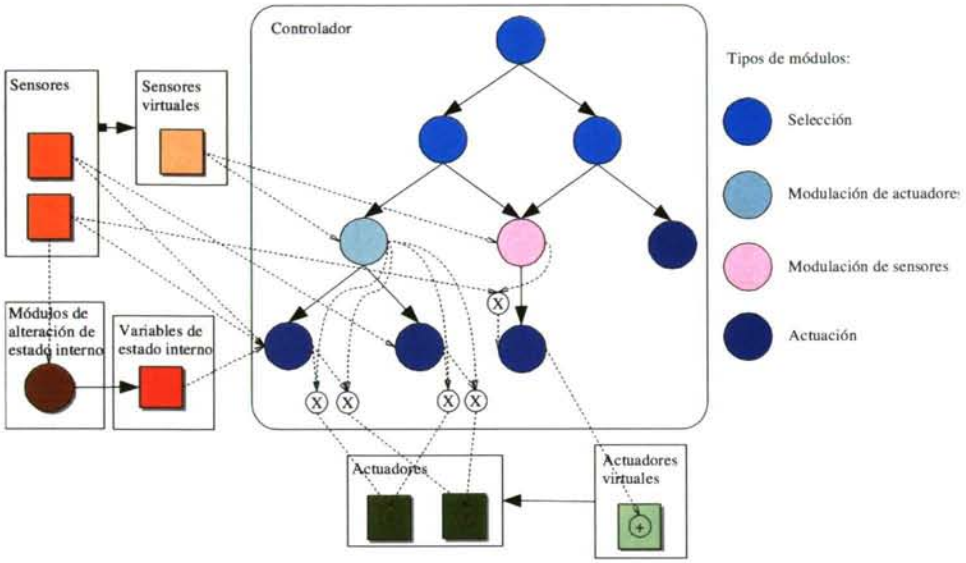


Figura 55: Esquema final de la arquitectura.

Teniendo en cuenta la equivalencia que puede ser establecida entre módulos selectores y moduladores de actuadores, las operaciones que se realizan entre datos de sensores y sus modulaciones y entre valores para actuadores y sus modulaciones (multiplicaciones y sumas), así como el hecho de que dichas operaciones poseen las propiedades conmutativa y asociativa, todo controlador, independientemente de cuántos niveles tenga y de qué tipo sean sus nodos, se puede representar alternativamente siguiendo un esquema como el de la figura 56 con sólo dos niveles. En el nivel inferior los módulos actuadores y en el superior los módulos moduladores, tanto los de sensores como los de actuadores. Habrá tres matrices de conexión en esta representación. Una entre los sensores, en cualquiera de sus formas, los moduladores de sensores y los módulos de actuación, de forma que habrá una conexión multiplicativa entre cada sensor que utilice un módulo de actuación y cada salida de un modulador de sensores que module dicho sensor y que se encuentre en algún camino desde la raíz del controlador hasta dicho módulo de actuación. Otra matriz de conexión será entre los módulos de actuación, los moduladores de actuadores y los actuadores, de forma que habrá una conexión multiplicativa entre cada actuador de un módulo de actuación y cada salida de un modulador de actuadores que module dicho actuador y que se encuentre en algún camino desde la raíz del controlador hasta dicho módulo de actuación. La última matriz será entre los resultados de los productos de la anterior

matriz y los actuadores, de forma que todos los productos que involucren a un mismo actuador se sumarán antes de aplicarlos en él.

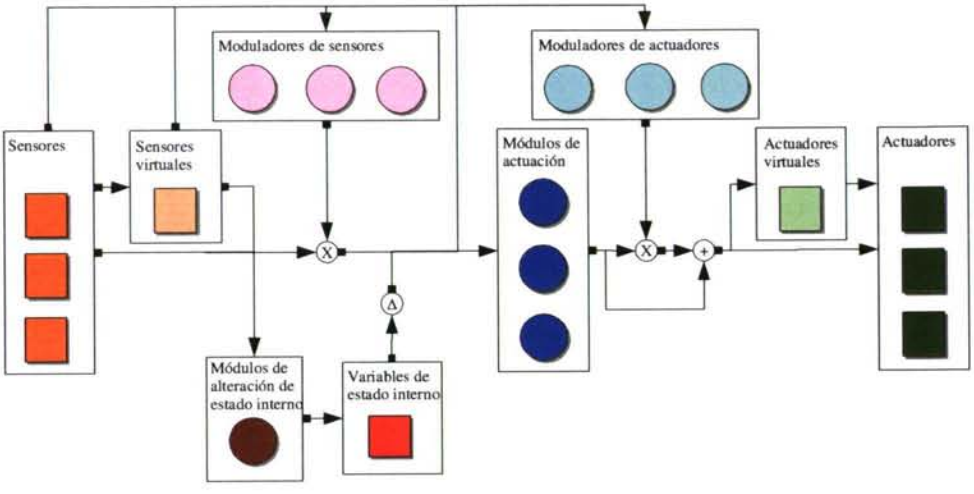


Figura 56: Esquema final de la arquitectura (normalizado a dos niveles).

La primera representación esquematiza la jerarquía de módulos y, por tanto, cómo fluye el control. En ella se aprecia de qué módulos depende la ejecución de cada módulo, cuáles modula un modulador (sus descendientes) y es, además, la representación más adecuada para diseñar los controladores añadiendo componentes, reutilizando los ya existentes, etc. La segunda representación tiene la ventaja ser independiente del número de niveles y módulos y podría ser útil como paso intermedio a una posible implementación por hardware del controlador en el robot. Es preciso notar que, por claridad, cada línea en la figura 56 representa en realidad múltiples líneas y oculta las matrices de conexión que se han mencionado. Asimismo, en todos los cruces de líneas de esa figura, los datos son susceptibles de dirigirse a todos los destinos apuntados por las flechas a las cuales se puede llegar desde dicho cruce. Por ejemplo, puede haber conexiones desde los sensores hasta los módulos de alteración de estado interno y, quizás multiplicados por la salida de algún modulador de sensores, hasta módulos de actuación, de modulación de sensores y de modulación de actuadores.

. A continuación veremos ejemplos tanto de modulaciones de sensores como de modulaciones en los actuadores, en los que se apreciará cómo funciona y las ventajas que aporta la modulación.

Modulación de entradas

El primer ejemplo que veremos será de modulación en los sensores, y está basado en el comportamiento de seguir paredes con el Pioneer 2-DX. Lo modificaremos para que realice ese seguimiento de paredes a distancias distintas dependiendo del valor de un

estímulo externo, que será la luz ambiente. El controlador de seguir paredes está ahora modulado por un módulo que alterará sus entradas, es decir, los valores que proporcionan los sónares de la parte delantera, en función del valor de luz ambiente sensado. El Pioneer 2-DX no dispone de sensores de luz, por lo que se han obtenido de otro robot para poder utilizarlos. El entorno empleado y la función de calidad son los mismos que para el comportamiento de seguir paredes que obtuvimos para el Pioneer 2-DX, con la siguiente diferencia: con un valor de luz ambiente equivalente a una bombilla de 60W situada a 10 cm de los sensores, la comida está a la distancia habitual de la pared, mientras que con el valor de luz ambiente utilizado para modelar los sensores de luz la comida está más separada. Así, el modulador tendrá dos entradas (los sensores de luz) y ocho salidas (los valores de modulación para los sensores del módulo inferior). Las características de la red y del algoritmo evolutivo se pueden ver en la figura 57, así como el esquema del controlador, la gráfica de calidad y el comportamiento obtenido para una evolución. El problema en sí parece muy sencillo, por cuanto la red sólo tiene que aprender unos valores fijos que sólo cambiarán cuando cambie la luz sensada. Efectivamente, el problema es sencillo, pero no tanto como pueda parecer en una primera impresión, por dos razones:

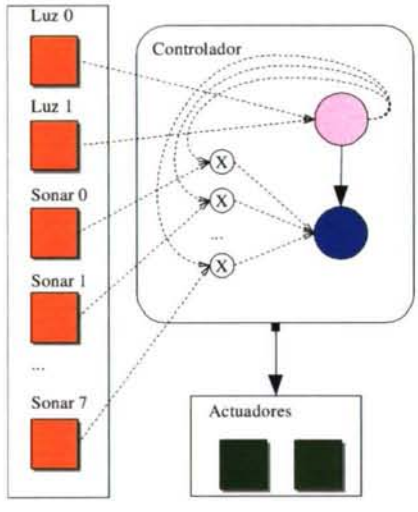
- La primera es que la comida se ha puesto manualmente utilizando el interfaz gráfico de SEVEN, con lo que en realidad no siempre está exactamente a la misma distancia de la pared.
- La solución no es tan sencilla como escalar todos los valores sensados por un mismo factor, ya que un mismo escalado en una entrada no tiene por qué implicar un mismo cambio en la activación en las neuronas a la que está conectada.

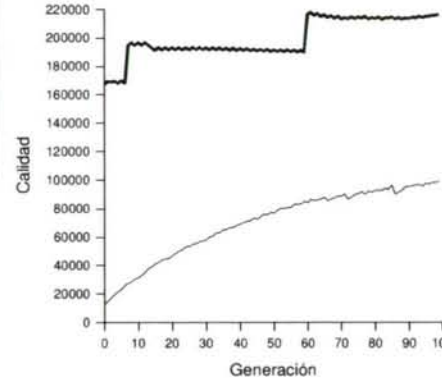
La evolución lidia con estas dificultades de una forma eficaz e inesperada en un principio, ya que no intenta que la RNA evolucionada module todos los sensores, lo cual puede ser muy complicado e incluso imposible. En su lugar, desactiva los innecesarios en ambos casos y juega con los otros para lograr el efecto deseado, mejorando incluso el comportamiento original que está siendo modulado. En la figura 58 vemos los niveles de activación (que pueden oscilar entre 0 y 2) que la red emplea en cada uno de los dos casos vistos en la evolución. Las columnas a la izquierda se corresponden con la luz de la bombilla situada a 10 cm y los de la derecha con luz ambiente habitual. Los sensores 3 y 7 son desactivados[5] en ambos casos y la red juega principalmente con el sensor 1 (el sensor 6 no es tan importante puesto que está a la derecha del robot y éste sigue las paredes dejándolas a su izquierda), de forma que cuando la intensidad de luz es menor el sensor ve incrementado su valor y el robot cree ver los objetos más cerca de lo que en realidad están y, por tanto, se separa más de la pared. La modulación de entrada no sólo permite, pues, adaptar comportamientos a nuevas situaciones, sino también hacer podado de una red ya existente.

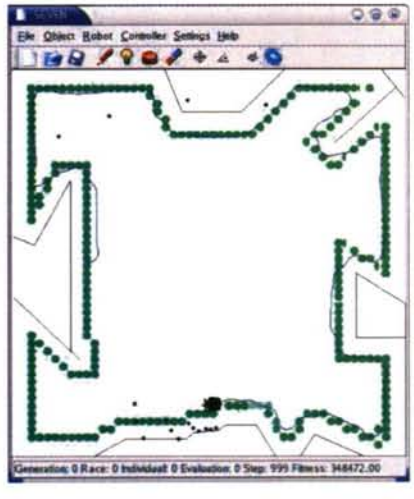
5 En los apéndices se puede consultar la ubicación de los sensores de sónar.

Paramétros de la RNA	
Nº neuron. capa entrada (1)	2
Nº neuron. capa oculta (2)	11
Nº neuron. capa salida (3)	8
Tipo conex. entre (1) y (2)	Gausianas
Tipo conex. entre (2) y (3)	Gausianas
Valor máximo de un retardo	-
Valor máximo de un peso	6

Parámetros del AE	
Tipo	MA
Distribución inicial	Aleatoria
Prob. Cruce	-
Prob. Mutación	-
Razas	1
Poblaciones por raza	1
Generaciones	100
Migración local	-
Migración global	-
Población	800
Evaluac. por individuo	100
Pasos de vida	1000







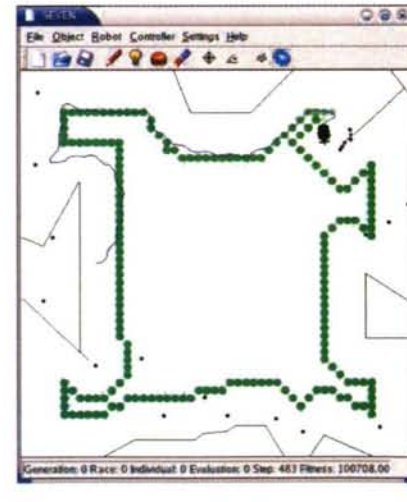


Figura 57: Comportamiento de seguir paredes para el Pioneer 2-DX modulando sus entradas para que siga las paredes a distancias distintas en función de la luz ambiente.

En la figura 59 podemos ver este comportamiento sobre el robot real. En el pasillo en el cual se desplazaba el Pioneer 2-DX algunas persianas estaban bajadas y otras subidas, generando zonas de mayor y menor intensidad luminosa. El alcance de esas

zonas se aprecia claramente en dicha figura, ya que cuánta menos luz percibía el robot más se alejaba de la pared.

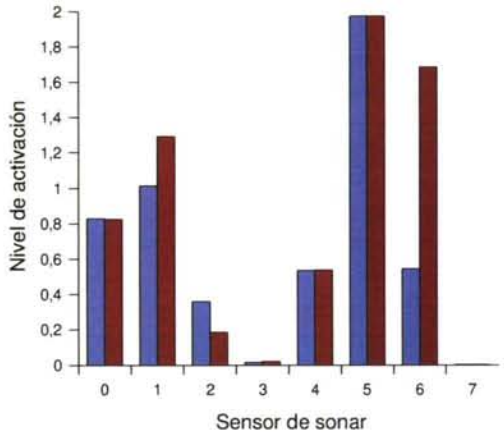


Figura 58: Valores de modulación de las entradas para los ejemplos del comportamiento de seguir paredes modulado por la luz ambiente para el Pioneer 2-DX.

Figura 59: Comportamiento de seguir paredes modulado por la luz ambiente para el Pioneer 2-DX. La línea roja muestra la trayectoria seguida por el robot.

Modulación de salidas

El siguiente ejemplo nos permitirá apreciar la modulación sobre actuadores. El comportamiento consiste en, utilizando dos módulos de bajo nivel que implementan dos comportamientos de "ir hacia presa" y "escapar de depredador", realizar una misión que podríamos calificar de "sobrevivir", en la cual el robot debe realizar ambas cosas, es decir, cazar una presa escapando de su propio depredador, pero dicha función será realizada dependiendo de un valor energético interno que denominaremos "consumo".

Estos ejemplos difieren de todos los anteriores en que han sido planteados obviando, en un principio, las limitaciones sensoriales del robot en el cual se iban a ejecutar los controladores finales obtenidos (el Pioneer 2-DX en este caso). Esto se ha hecho así porque este planteamiento de distinción entre presa y depredador no era fácilmente realizable con el Pioneer 2-DX y en este ejemplo se pretendía, ante todo, mostrar muy claramente el funcionamiento de la modulación de salidas y las ventajas presentes en su utilización. Así, el modulador recibe datos de sensores virtuales "perfectos", en el sentido de que la información que proporcionan carece de error, y los módulos de bajo nivel son módulos basados en reglas que reciben también información 100% precisa de la localización del depredador y de la presa. De esta forma, aislando el problema de las limitaciones sensoriales del robot utilizado, podremos analizar sin interferencias el funcionamiento del mecanismo de modulación. Al final del apartado de modulación conjunta de entradas y salidas se describe cómo se trasladaron estos controladores al robot real, lo cual se realizó sustituyendo, básicamente, tanto los sensores virtuales como los módulos de bajo nivel por otros de peor funcionamiento (provenientes en algún caso de otro robot) pero que permitieron ejecutarlos satisfactoriamente sobre el robot real.

En los ejemplos vistos hasta ahora toda la información sensorial que recibían los módulos provenía del exterior. Esto no tiene por qué ser así. El robot puede tener estados internos asociados a su funcionamiento, y esa información puede ser relevante para su comportamiento. El ejemplo habitual es la carga de la batería. En las figuras 55 y 56 se puede apreciar la ubicación en la arquitectura de este tipo de sensorización. Existen una serie de variables de estado interno que pueden ser accedidas por cualquier módulo que las necesite mediante una función de transformación que, en la implementación actual, es el cambio respecto al valor en el instante anterior, aunque, obviamente, no tiene por qué ser siempre así. Los valores de esas variables de estado interno son alterados por unos módulos específicos que, a su vez, pueden recabar información de sensores para realizar dicha alteración.

Por tanto, el modelo energético empleado para obtener este comportamiento es el siguiente:

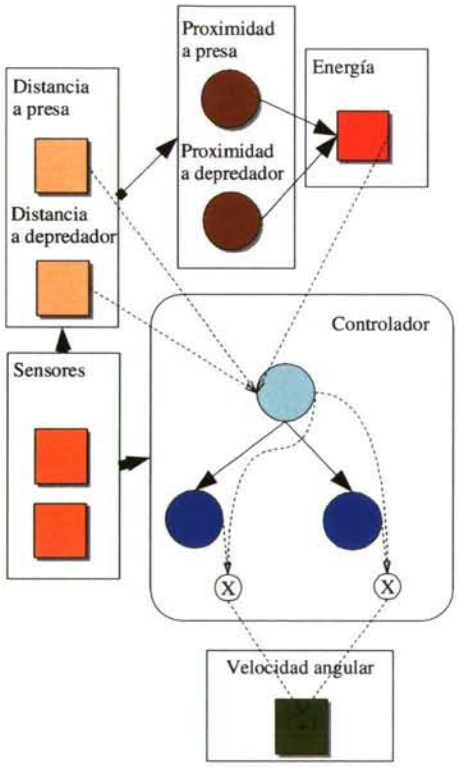
- El robot gana energía por estar cerca de la presa, más energía cuanto más cerca, de forma lineal desde 5 metros (ganancia 0) a 0 metros (ganancia de 20 unidades de energía) de distancia hasta la presa.
- El robot pierde energía por estar cerca del depredador, más energía cuanto más cerca, de forma lineal desde 2 metros (pérdida de 0) a 0 metros (pérdida de 100 unidades de energía).
- El robot tiene un consumo asociado, que puede ser de 20 unidades de energía por unidad de tiempo o de 400.

Estos valores han sido escogidos de forma que, cuando el consumo es pequeño, al robot le compensa no caer en el radio de acción del depredador y dar un rodeo amplio

para acercarse a la presa porque la pérdida por acercarse al depredador es mayor que la pérdida por consumo. Sin embargo, cuando el consumo es elevado la situación es la inversa, y el robot desprecia la proximidad al depredador y se dirige lo más recto posible a la presa (esquivándolo, obviamente, si no nunca alcanzaría la presa y sería "devorado") ya que el consumo le hace perder mucha energía cada instante que pasa.

Paramétros de la RNA	
Nº neuron. capa entrada (1)	3
Nº neuron. capa oculta (2)	6x2
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	-
Valor máximo de un peso	6

Parámetros del AE	
Tipo	MA
Distribución inicial	Aleatoria
Prob. Cruce	-
Prob. Mutación	-
Razas	1
Poblaciones por raza	1
Generaciones	20/60/100
Migración local	-
Migración global	-
Población	100
Evaluac. por individuo	110/250/250
Pasos de vida	300



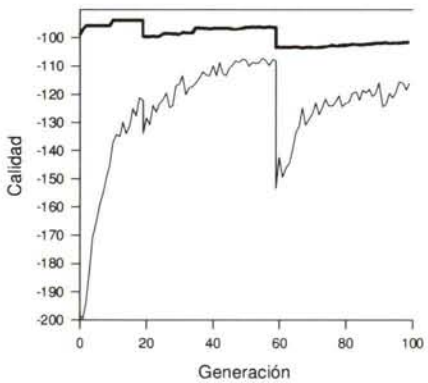


Figura 60: Características y datos de obtención en una evolución del comportamiento de supervivencia para el Pioneer 2-DX.

La evolución se llevó a cabo en un mundo vacío, en el que el robot, la presa y el depredador estaban siempre en la misma posición, y sólo la orientación del robot variaba. Las características del módulo de alto nivel, el algoritmo evolutivo empleado y las gráficas de calidad se pueden ver en la figura 60. Respecto al módulo de alto nivel

que estamos evolucionando, decir que sólo evoluciona para modular la velocidad angular (lo cual es suficiente, como veremos a continuación). Además, vemos un primer ejemplo de utilización de módulos de alteración del estado interno del robot, los etiquetados como "proximidad a presa" y "proximidad depredador", que, en función de los datos recibidos por los sensores virtuales "distancia a presa" y "distancia a depredador", modifican la energía del robot (que también se ve alterada por el consumo), cuyo valor es utilizado por el modulador para llevar a cabo su tarea.

La evolución es fácil, pero usaremos este ejemplo también para ver cómo se puede realizar evolución incremental, complicando el entorno paulatinamente. Esto es útil cuando se considera que obtener el comportamiento usando toda la complejidad del entorno que se puede encontrar el robot desde el principio es muy difícil, cuando los requisitos para el comportamiento han cambiado, o simplemente porque las posibilidades que tuvimos en cuenta en un principio no cubren suficientemente bien el conjunto de situaciones que puede tener que afrontar el robot.

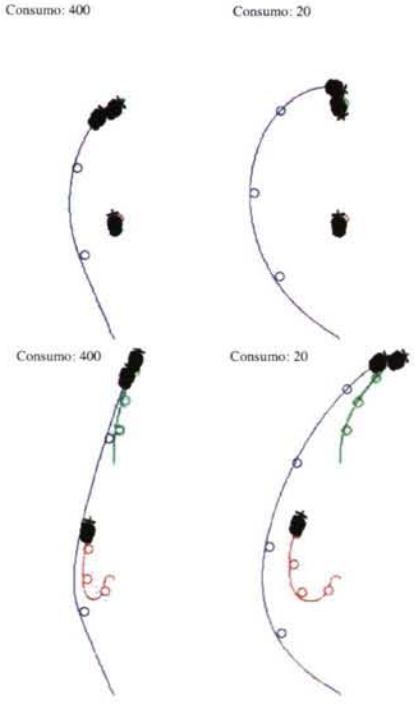


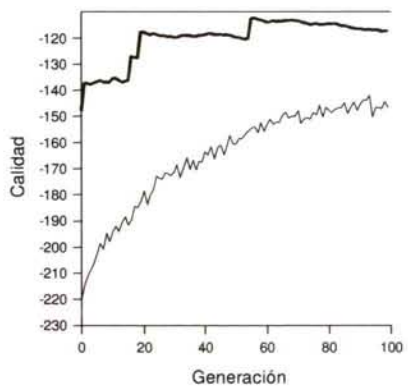
Figura 61: Comportamiento de supervivencia para el Pioneer 2-DX. Las imágenes superiores se corresponden con una evolución con depredador y presa estáticos y las inferiores con una evolución con depredador y presa móviles. Cada circunferencia en la trayectoria representa que el robot ha realizado 50 pasos desde la circunferencia anterior.

El problema se complicó en una segunda fase haciendo que la coordenada "y" de la posición inicial del robot variase aleatoriamente, lo que, teniendo en cuenta que la orientación ya variaba antes, es equivalente a variar la posición del robot en todo el entorno pero manteniendo siempre el depredador en la recta que une las posiciones iniciales del robot y la presa. En las imágenes superiores de la figura 61 se muestran los resultados.

Finalmente, el problema se volvió a complicar haciendo que tanto depredador como presa fuesen a su vez robots móviles. La presa tratará de escapar de nuestro robot mientras que el depredador tratará de cazarlo (imágenes inferiores de la figura 61). Los comportamientos de presa y depredador han sido definidos mediante un sencillo control algorítmico y sus velocidades máximas se han establecido en valores inferiores a los del robot que está siendo evolucionado para asegurar que el robot puede cumplir la tarea. Es importante hacer notar que cuando se realiza una continuación de la evolución, modificando el entorno como se ha hecho aquí, es necesario mantener casos de evaluación con el entorno viejo, sino corremos el riesgo de que el controlador evolucione a otro que se comporte bien en los entornos nuevos, pero "olvide" paulatinamente cómo hacerlo en los utilizados con anterioridad.

Paramétros de la RNA	
Nº neuron. capa entrada (1)	3
Nº neuron. capa oculta (2)	6x2
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	-
Valor máximo de un peso	6

Parámetros del AE	
Tipo	MA
Distribución inicial	Aleatoria
Prob. Cruce	-
Prob. Mutación	-
Razas	1
Poblaciones por raza	1
Generaciones	100
Migración local	-
Migración global	-
Población	100
Evaluac. por individuo	120
Pasos de vida	300



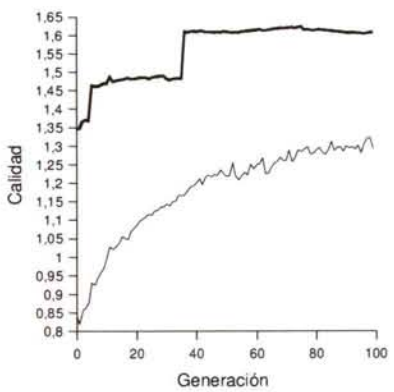


Figura 62: Parámetros y gráfica de calidad para la evolución del comportamiento consistente en dirigirse hacia una presa evitando un depredador que a su vez trata de cazarlo (gráfica de la izquierda) o de interponerse entre ambos (gráfica de la derecha).

A continuación, probaremos a obtener este último comportamiento, con presa y depredador móviles, desde el principio, a modo de comparación. Se probó, además, una variante, en la cual el depredador es reemplazado por un defensor de la presa que trata en todo momento de interponerse en el camino de nuestro robot hacia dicha presa. Nuevamente, como en el caso anterior, para asegurarnos que el robot podrá llevar a cabo la misión, la presa y el depredador tendrán su velocidad lineal y angular limitada respecto al robot que está siendo evolucionado. Así, en el primero de los casos, el depredador tendrá una velocidad lineal máxima de 130 mm/seg y la presa de 70 mm/seg. En el segundo caso, la velocidad lineal máxima de ambos estará limitada a 70 mm/seg y la angular a 2°/seg. El robot a evolucionar tendrá siempre unas velocidades máximas de 200 mm/seg (lineal) y 40°/seg (angular).

En la figura 62 podemos ver los parámetros de la red moduladora y del algoritmo evolutivo empleado, así como las gráficas de calidad para una evolución en la que el depredador trata de cazar a nuestro robot (izquierda) y otra para una evolución en la que el depredador trata de interponerse entre nuestro robot y la presa (derecha). En la figura 63 podemos ver el comportamiento final obtenido. Las imágenes de la parte superior se corresponden con la evolución donde el depredador intenta cazar a nuestro robot y las de la parte inferior con la evolución donde el depredador se interpone. En esa misma figura, vemos, debajo de la imagen con el comportamiento, los valores de la modulación para cada ejecución. Es interesante destacar dos puntos. Primero, que el controlador resultante es capaz de interpolar su comportamiento para valores intermedios de consumo pese a no haberse encontrado con ellos durante la evolución. Esto es importante porque vemos como utilizando moduladores de salidas y comportamientos básicos opuestos podemos obtener un continuo de comportamientos intermedios fácilmente (si la interpolación no fuese de nuestro agrado, bastaría con añadir algún caso intermedio a la evolución para ajustar la modulación a lo que nosotros deseamos). El otro punto a destacar es cómo la evolución ha llevado a una solución en la que sólo se modula la velocidad angular del módulo de escapar de depredador, mientras que se mantiene constante el valor de modulación para el módulo de dirigirse a presa.

La evolución no es ahora trivial, pero sigue siendo fácil y los comportamientos son satisfactorios. El único cambio que se ha realizado en los parámetros de evolución ha sido el número de evaluaciones por individuo. Ya que ahora tenemos un número mucho mayor de interacciones con el entorno, al tener tres robots móviles que además adaptan su estrategia continuamente dependiendo de los movimientos de los otros robots, es necesario que el número de evaluaciones sea mucho mayor para que la medición de la calidad sea fiable. Como, en cualquier caso, el número de individuos necesario es pequeño, podemos elevar el número de evaluaciones sin graves problemas. Es importante recalcar que, por supuesto, dentro de unos márgenes, es más importante que el número de evaluaciones sea lo suficientemente alto que usar un número mayor de individuos. Si el cálculo de la calidad no es fiable, la evolución oscilará continuamente

entre puntos en el espacio de búsqueda independientemente del número de individuos presentes en la evolución.

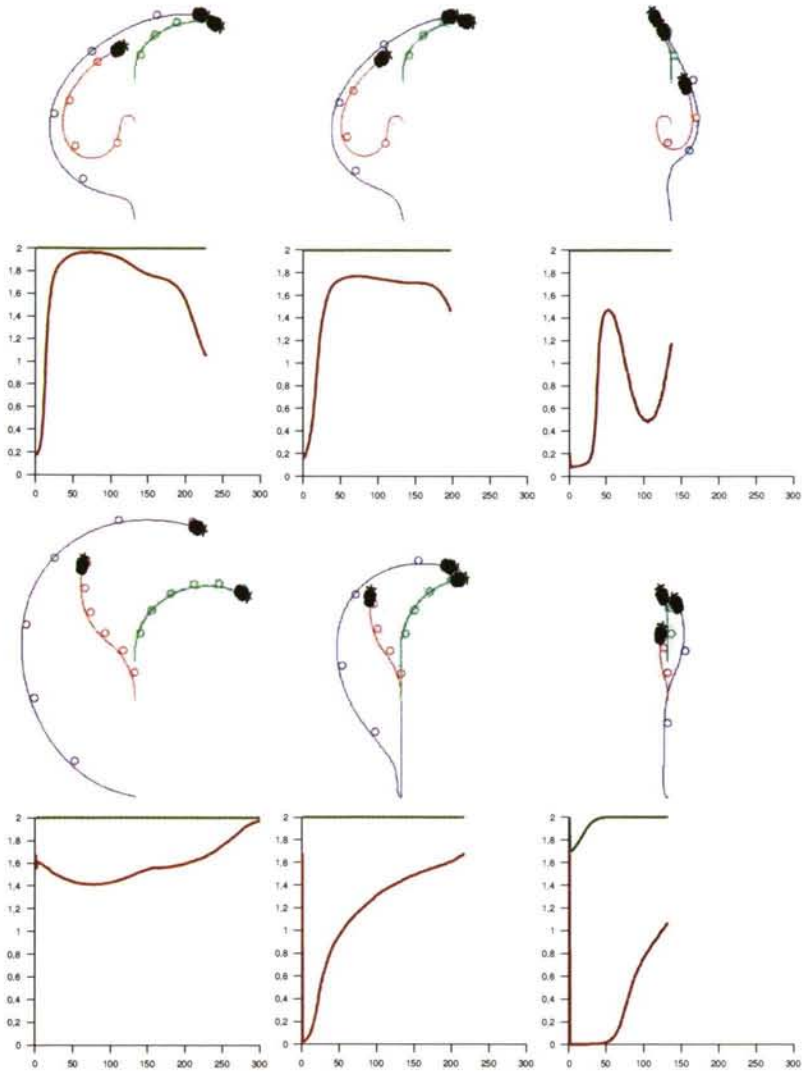
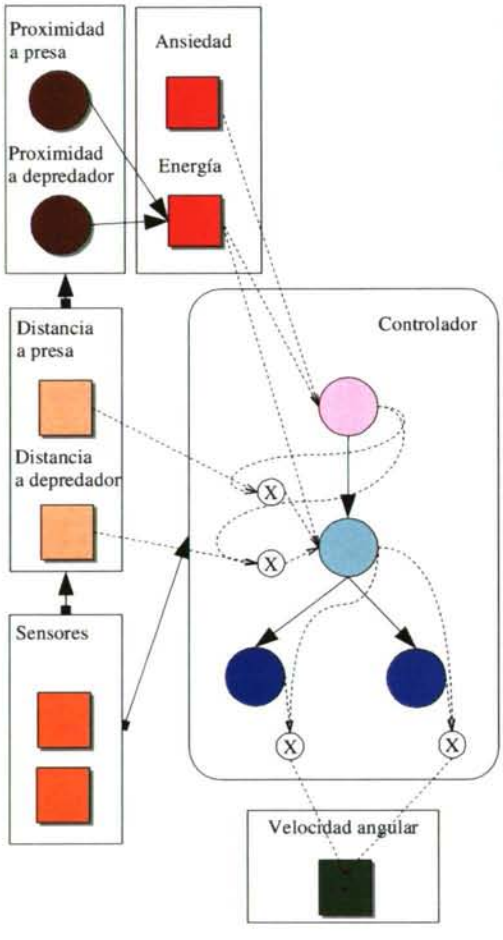


Figura 63: Ejemplos del comportamiento consistente en dirigirse hacia una presa evitando un depredador que se dirige al robot que está siendo evolucionado (gráficas superiores) o que se interpone entre ambos (gráficas inferiores) para tres diferentes consumos (20, 200 y 400). Se muestran en cada caso la modulación sobre la velocidad angular de los módulos de bajo nivel (rojo escapar de depredador y verde ir hacia presa). Los ejes de abscisas representan el tiempo y los ejes de ordenadas representan el valor para la modulación en cada instante de tiempo.

Modulación conjunta de entradas y salidas



Paramétros de la RNA	
Nº neuron. capa entrada (1)	2
Nº neuron. capa oculta (2)	6x2
Nº neuron. capa salida (3)	2
Tipo conex. entre (1) y (2)	Tradic.
Tipo conex. entre (2) y (3)	Tradic.
Valor máximo de un retardo	-
Valor máximo de un peso	6

Parámetros del AE	
Tipo	MA
Distribución inicial	Aleatoria
Prob. Cruce	-
Prob. Mutación	-
Razas	1
Poblaciones por raza	1
Generaciones	100
Migración local	-
Migración global	-
Población	100
Evaluac. por individuo	200
Pasos de vida	300

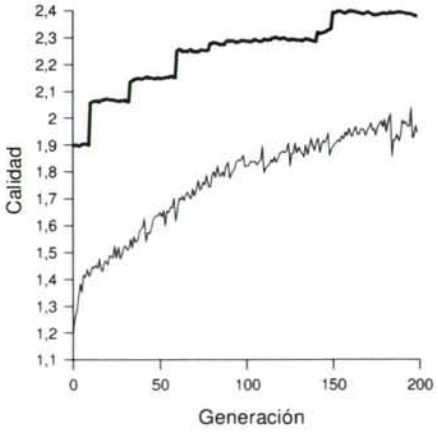


Figura 64: Parámetros y gráfica de calidad para la evolución del comportamiento consistente en dirigirse hacia una presa evitando un depredador que se interpone entre ambos y, al mismo tiempo, minimizando el nivel de ansiedad.

Por supuesto, no hay nada que impida la utilización en un mismo controlador de distintos tipos de modulaciones. A continuación expandiremos el ejemplo anterior para usar conjuntamente modulación sobre actuadores y sobre sensores. Supondremos que el robot, además de tener que maximizar su nivel energético interno, tiene también que minimizar su nivel de "ansiedad". La ansiedad dependerá de un factor externo (ruido ambiental) y el controlador global tendrá un módulo situado en su parte superior que, en

función del nivel de ansiedad, modulará los sensores de su módulo descendiente. Ambos factores, energía y ansiedad, entrarán en el cálculo de la calidad como un simple sumatorio (ambos se consideran igual de importantes). En la figura 64 se puede ver el esquema final del controlador.

En la figura 64 tenemos también los datos del algoritmo evolutivo empleado y de la RNA que forma el módulo evolucionado así como la gráfica de calidad. Aunque la evolución duraba en principio 100 generaciones, se prolongó durante otras 100 al comprobar que la calidad media estaba muy alejada de la máxima y que seguía en trayectoria ascendente. Esto era indicativo de dos cosas: primero, que la evolución todavía no había convergido y que era probable que se pudiese encontrar todavía una mejor solución; segundo, esa propia dificultad en la convergencia, unido al hecho de que la calidad media estuviese tan lejos de la máxima, indicaba que la solución o soluciones óptimas estaban ubicadas en picos hiperdimensionales estrechos en su parte más alta que requerían, por tanto, un mayor periodo de explotación. Por esto, la evolución se continuó otras 100 generaciones y, efectivamente, la solución, ya de por sí buena, se mejoró. Al final de la evolución la calidad media parece estabilizarse, pero aún permanece lejos de la máxima, con lo no cabe descartar que, con más generaciones, se pudiese mejorar todavía más la solución.

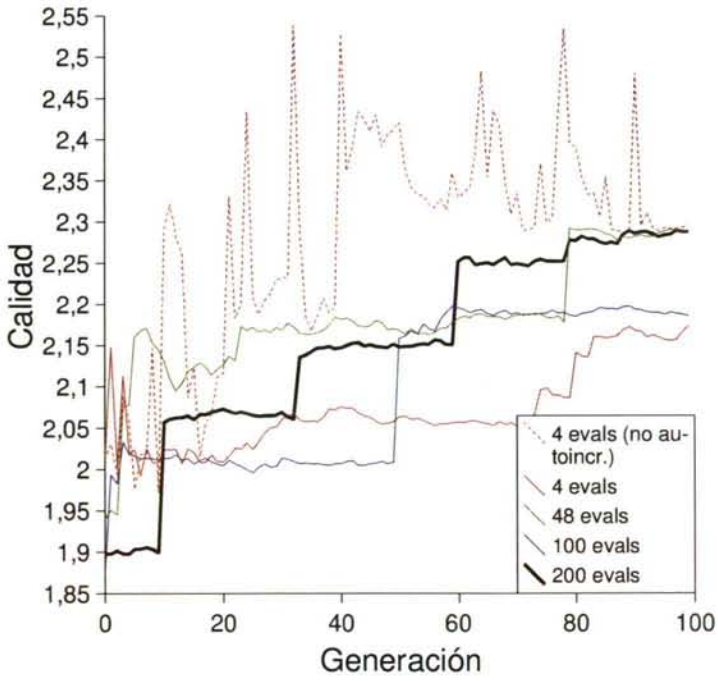


Figura 65: Efecto de un número inadecuado de evaluaciones por individuo.

En la gráfica de calidad de la figura 65 tenemos la calidad máxima de distintas evaluaciones de este mismo comportamiento usando un número distinto de evaluaciones por individuo. Esto nos sirve para ilustrar la importancia de un número adecuado de evaluaciones por individuo. En las evoluciones con 4, 48 y 100 evaluaciones iniciales por individuo (que acabaron con 164, 180 y 196 evaluaciones respectivamente) se puede observar cómo la calidad máxima oscila en la primeras generaciones y cómo esa oscilación es mayor cuanto menor es el número de evaluaciones. Se podría pensar que con el mecanismo de autoincremento del número de evaluaciones, explicado en el apartado 4.4.3, ya tenemos una solución adecuada al problema, independientemente del número de evaluaciones inicial. Pero eso no es así. El mecanismo puede corregir el problema si no hay una discrepancia muy alta, pero si ésta existe no es suficiente, ya que el número de evaluaciones se va incrementando poco a poco (de 4 en 4 en este caso), pero mientras no se llega a un número suficiente, la evolución está viciada: la medición de la calidad no es fiable y los individuos que desaparecen no tienen por qué ser necesariamente los peores. Esto hace que, en el mejor de los casos, la evolución sea más lenta y se tarde más en llegar a la solución y, en el peor, que nos quedemos en un máximo local por haber descartado erróneamente zonas del espacio de búsqueda. Por eso, cuando se observa una discrepancia muy grande entre el número inicial de evaluaciones y el final al que llega el mecanismo de incremento automático, es mejor volver a iniciar la evolución con un número mayor (alrededor del indicado por el mecanismo) de evaluaciones por individuo. Las gráficas de calidad mencionadas son un claro ejemplo de lo comentado.

Cabe preguntarse también por qué las evoluciones con un número insuficiente de evaluaciones iniciales no llegaron al mismo número final de evaluaciones. Existen dos posibles razones para que esto suceda. La primera es que es posible que muchos individuos dentro del espacio de búsqueda presenten una misma calidad para situaciones distintas, mientras que puede haber un pequeño número de ellos que, por tener una discriminación más fina del espacio sensorial, presenten ante esas mismas situaciones calidades distintas. La segunda es que hay que tener en cuenta que, cuando ya han transcurrido muchas generaciones, los individuos que permanecen en la población desde hace tiempo ya han superado el número mínimo de evaluaciones por individuo, el cual sólo se aplica a los individuos generados aleatoriamente, que si son peores que los ya existentes se irán enseguida de la población sin posibilidad de que el mecanismo de autoincremento compruebe si el número mínimo de evaluaciones por individuo es suficiente. Además, puede ser que el algoritmo evolutivo empleado en este caso, un macroevolutivo, entre en zona de no generación de más individuos aleatorios antes de que el mecanismo de autoincremento establezca el número idóneo de evaluaciones por individuo.

En la mencionada gráfica se puede ver también la calidad máxima de una evolución con sólo 4 evaluaciones por individuo y con el mecanismo de autoincremento desactivado. Se puede observar cómo existen oscilaciones muy grandes en el valor de la

calidad para el mejor individuo y la evolución apenas avanza. No es de fiar la calidad máxima obtenida ya que seguramente se tratará de un individuo que ha sido evaluado en casos favorables. Efectivamente, la figura 67 muestra la ejecución del mejor individuo de esta última evolución con un incremento de energía de -20 y un valor de ansiedad de 400. Si lo comparamos con el correspondiente mejor individuo de la evolución con 200 evaluaciones para dichos valores de energía y ansiedad en la figura 66, vemos que éste último lo hace mucho mejor (es un caso en el que la elevada ansiedad convierte en mejor comportamiento aquel que hace al robot ir directo a su presa).

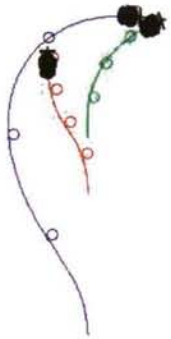


Figura 67: Ejemplo de calidad engañosa cuando el número de evaluaciones por individuo no es suficiente.

En la figura 66 se presentan varios ejemplos del comportamiento final para distintos valores de incremento de energía (función del consumo) y de ansiedad, y los valores de las modulaciones (rojo para el sensor de distancia al depredador y el actuador de velocidad angular del módulo que escapa de él, verde para el sensor de distancia a la presa y el actuador de velocidad angular del módulo que se dirige a ella), tanto de actuadores (líneas punteadas) como de sensores (líneas continuas), para dichos ejemplos. Las modulaciones de los actuadores son en realidad las ya vistas en el ejemplo anterior, donde cabe recordar que el controlador prácticamente sólo jugaba con el módulo de escapar del depredador para conseguir los resultados deseados. Aquí sucede algo similar: dado que el módulo cuyos sensores están siendo modulados ignora en la práctica al sensor que mide la distancia a la presa, dicho sensor carece de importancia, y así recibe un valor de 0 (lo que permitiría "podar" la RNA) y el módulo varía, nuevamente, un único factor: la distancia al depredador. Además, sólo juega con él cuando el valor de ansiedad es mayor que la pérdida de energía en cada paso, pues de lo contrario el módulo inferior ya toma la decisión correcta.

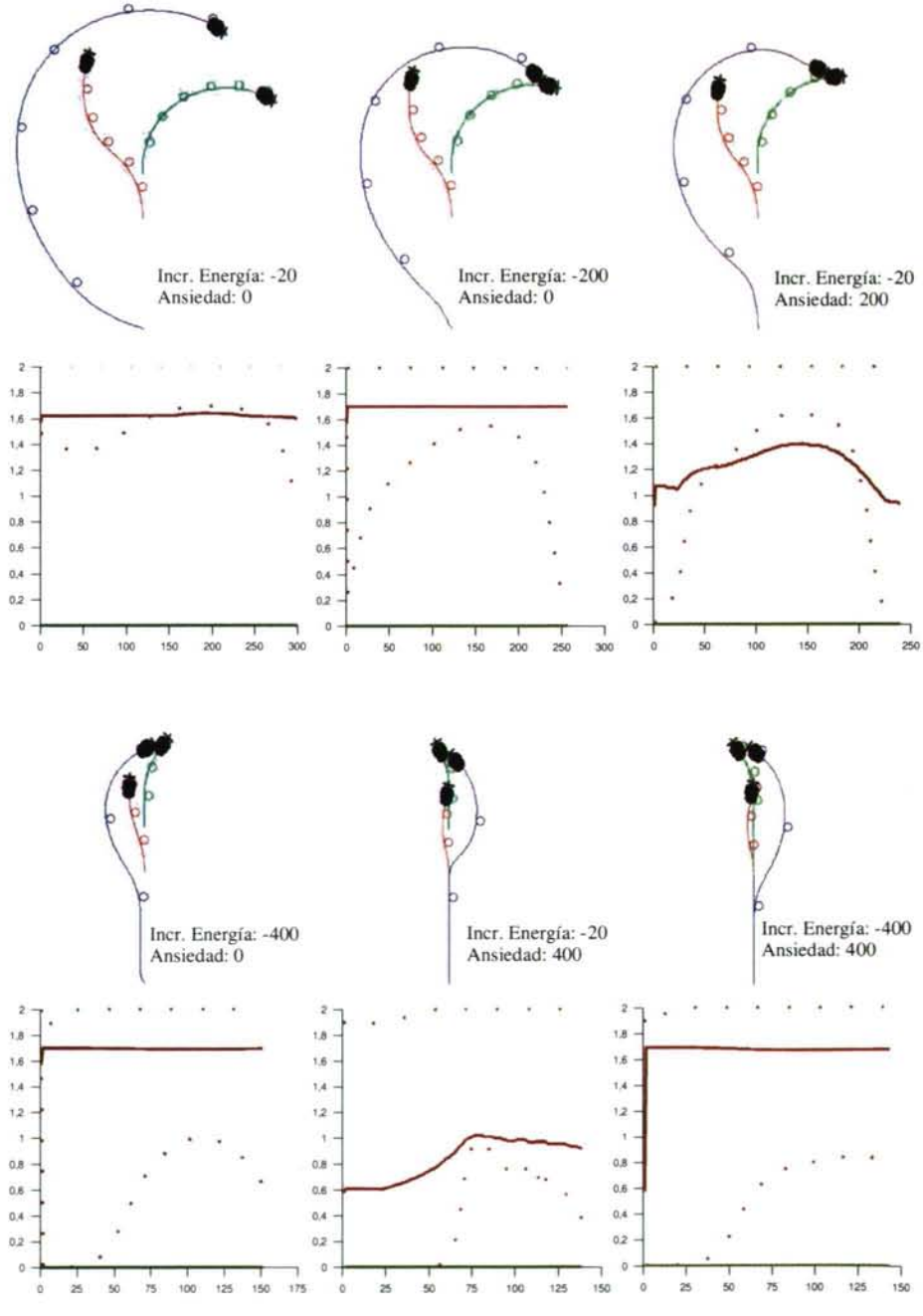


Figura 66: Ejemplos del comportamiento consistente en dirigirse hacia una presa, evitando un depredador que se interpone entre ambos y, al mismo tiempo, minimizando el nivel de ansiedad. Los ejes x representan el tiempo y los ejes y representan el valor para las modulaciones en cada instante de tiempo.

Como se comentó al principio del apartado de modulación de salidas, los ejemplos descritos que utilizan ésta se obtuvieron utilizando sensores virtuales que proporcionan información precisa, sin ningún tipo de ruido, a los distintos módulos que componen el controlador global. A la hora de trasladar los controladores que implementan estos comportamientos de modulación al robot real se contemplaron dos alternativas. Una consistió en utilizar únicamente los sensores de sónar y, mediante el uso de información temporal, obviar los sensores virtuales de distancia a presa y distancia a depredador y tratar de identificar a los robots por sus velocidades relativas (los módulos de bajo nivel eran simplemente escapar de objeto móvil y acercarse a objeto móvil) El controlador obtenido funciona correctamente aunque en ocasiones no puede realizar su tarea, ya que, dependiendo de las trayectorias de los tres robots involucrados, se pueden producir situaciones ambiguas en las percepciones de las velocidades.

La otra posibilidad consistió en utilizar nuevamente los sensores de luz obtenidos de otro robot para identificar a la presa, reservando el uso de los sonares para identificar al depredador. Así, los sensores virtuales de distancia a presa y distancia a depredador se implementaron mediante el uso de los sensores de luz y de sonar respectivamente, y los módulos de bajo nivel de alcanzar presa y huir de depredador se sustituyeron por módulos de alcanzar luz y escapar de objeto móvil. El controlador resultante (sin necesidad de evolución, sólo se intercambian unos módulos por otros) funciona también correctamente, sin la existencia de ambigüedades, aunque con un comportamiento más oscilante por cuanto la distancia a la presa es poco fiable al depender de la cantidad de luz sensada y ésta de la orientación del robot respecto a la presa. Otra limitación estriba en que la presa no puede ser detectada por los sónares o la confundirá con el depredador. Para ello, el papel de presa lo desempeñó un humano apoyado contra una pared, y por tanto indistinguible de ésta, sosteniendo una luz al alcance de los sensores de luz pero lo suficientemente alta para no ser detectada por los sónares. El depredador era también otra persona que se movía a una velocidad más o menos constante y similar a la del depredador en el simulador. En la figura 68 se puede ver el comportamiento del robot con este último controlador correspondiente a las imágenes del simulador que se ven en la parte superior de la figura 63, y cómo realiza la modulación en función del consumo (indicado internamente en algunas pruebas y simulado con ruido percibido por el micrófono en otras, alejándose del depredador en situaciones de silencio / bajo consumo o asumiendo más riesgos con ruido elevado / consumo elevado).

Resumiendo, en este capítulo hemos visto la modulación como un mecanismo más de interacción entre módulos. Un mecanismo mucho más poderoso que la selección, de forma que, de hecho, ésta puede verse como un caso específico de modulación. Esto nos permite mostrar una representación alternativa para la arquitectura de controladores en la cual sólo hay dos niveles de módulos, uno formado por aquellos que toman el control de los actuadores y otro formado por los que, interactuando entre sí, deciden cuáles y de qué forman lo hacen los del primer nivel.

Hemos visto también como la modulación permite obtener un continuo de comportamientos entre aquellos proporcionados por los módulos inferiores, interpolando incluso comportamientos para casos no contemplados en la evolución. Además, la modulación de entrada mejora incluso los comportamientos de bajo nivel realizando, en la práctica, un podado de las redes que componen éstos.

Figura 68: Modulación de actuadores sobre el robot real (Pioneer 2-DX).

7 Conclusiones y futuras vías de investigación

En este trabajo se ha presentado una aproximación evolutiva para la obtención incremental de arquitecturas modulares de comportamientos en robots autónomos. Dicha aproximación se basa en los principios de la robótica basada en comportamientos, es decir, en la relación directa entre el robot y el entorno en el que se mueve y en que los comportamientos surgen como resultado de dicha relación. Se han sentado unas bases y una metodología que permiten obtener controladores que dan lugar a comportamientos complejos dentro de esta aproximación a la robótica autónoma, tratando de minimizar en todo momento la participación del diseñador en el proceso.

Se han seleccionado las redes de neuronas artificiales como herramienta principal a utilizar para implementar los controladores de comportamientos, fundamentalmente por su tolerancia al ruido y su característica de aproximadores universales, y se ha recurrido a los algoritmos evolutivos para obtener éstos de forma automática, aunque la metodología permite la utilización de otros paradigmas. La evolución se lleva a cabo en un entorno simulado y la función de calidad sigue siempre un modelo energético. Ese modelo energético permite aislar lo máximo posible al diseñador humano del proceso evolutivo, al limitarse el primero a establecer reglas de ganancia y pérdida de energía en el entorno que llevarán a la obtención del controlador que implemente el comportamiento deseado. Toda esta serie de decisiones, orientadas a cumplir las premisas de las que parte el trabajo, tienen sus implicaciones en cuanto a dificultad del problema. Así, hemos visto que es necesaria la introducción de ruido en la simulación para que los controladores obtenidos sean aplicables al robot real y que hacen falta tipos de ruido específicos, normalmente no considerados, como los ruidos sistemáticos que hemos definido, para lograr que los controladores toleren anomalías en el funcionamiento del robot o determinados cambios en el entorno. La introducción de ruido provoca también que la evaluación de la calidad sea un proceso no determinista, con lo que el número de evaluaciones por individuo se convierte en un factor clave. Cuando éste es insuficiente la evolución puede oscilar entre soluciones parciales que se comportan correctamente sólo en un subconjunto de todas las posibles configuraciones que puede adoptar el entorno, dando como consecuencia un controlador final que no poseerá la robustez requerida. La introducción de un mecanismo de autoincremento en el número de evaluaciones nos ha permitido paliar este problema.

La obtención mediante evolución de los controladores, combinada con el uso de RNAs y funciones de calidad que siguen un modelo energético, se ha mostrado como una herramienta adecuada para obtener controladores de forma automática y minimizando el papel del diseñador. Pero también se ha comprobado que este tipo de problemas son difíciles, debido principalmente a dos circunstancias. La primera es que el tiempo de evaluación de los individuos es considerable pese a realizarse la evolución en un entorno simulado, ya que algunos comportamientos implican periodos de evaluación prolongados en cuanto a movimientos que deben ser simulados, y esto lleva a utilizar tamaños de población pequeños. La otra circunstancia que dificulta la

evolución es que, en ocasiones, la función de calidad lleva a espacios de búsqueda muy complicados, con superficies que presentan zonas amplias con calidad muy similar en las que solamente hay puntos aislados y dispersos con mayor calidad. Es decir, hay poca información de gradiente en la superficie generada por las funciones de calidad en los espacios de búsqueda, de forma que el algoritmo evolutivo puede caer muy fácilmente en máximos locales. Esta circunstancia, además, es impredecible en cuanto a cuándo va a suceder, ya que, obviamente, la forma de los espacios de búsqueda no es conocida a priori. Para solucionar estos problemas son necesarias una serie de técnicas que en otras aplicaciones de algoritmos evolutivos no son necesarias. Se han analizado diversos tipos de algoritmos evolutivos y se ha visto cómo las peculiaridades de los espacios de búsqueda en estos problemas traen como consecuencia que soluciones híbridas entre algoritmos genéticos y estrategias evolutivas y, especialmente, los algoritmos macroevolutivos, con un adecuado balance entre exploración y explotación, tengan un mejor comportamiento que otros tipos de algoritmos evolutivos.

Se ha comprobado también que, debido al tamaño reducido de la población comparado con la dimensionalidad del espacio de soluciones, la distribución de la población inicial es relevante y, sobre todo, que la división de la población en razas disminuye la probabilidad de caer en máximos locales al limitar el efecto del superindividuo y permitir que subespacios del espacio de búsqueda sean explorados de forma aislada durante un cierto número de generaciones. Además, la división en razas de la población ha favorecido la paralelización y distribución de los algoritmos evolutivos, que se ha implementado siguiendo un método propio que se ha mostrado muy eficaz y que posibilita obtener controladores para comportamientos complejos en tiempos reducidos si se dispone de los recursos computacionales necesarios.

Hemos propuesto formas de aumentar la capacidad de los controladores introduciendo en las redes de neuronas artificiales elementos de mayor complejidad que los utilizados habitualmente en robótica, como los retardos y las neuronas de habituación, que proporcionan la capacidad de procesar información temporal, y las sinapsis gausianas. Las neuronas de habituación se han mostrado como un mecanismo muy eficaz de incorporar en determinados casos capacidad de procesado de información temporal con un incremento mínimo en la longitud del cromosoma, mientras que los retardos sinápticos son un método más general que posibilitan a una RNA tratar con patrones temporales específicos y que, como hemos visto, permiten que el robot supere situaciones de infrasensorización o sensorización defectuosa y mejore además su comportamiento en casos en los que dicha capacidad no es imprescindible pero sí conveniente. La sinapsis gausianas, por su parte, se han mostrado capaces de funcionar como un mecanismo de atención, haciendo que una RNA sólo considere aquellos datos realmente relevantes para la tarea para la cual va a funcionar como controlador e ignore los restantes, lo cual permite además que el algoritmo evolutivo realice un podado automático sobre las RNAs cuando hay sensores o neuronas que son totalmente innecesarios.

Dado que, por elevados que sean los recursos computacionales a nuestro alcance, no es factible ni razonable tener que obtener siempre un nuevo comportamiento partiendo de cero, se han propuesto mecanismos que permiten construir comportamientos complejos de forma gradual, obteniendo primero comportamientos sencillos que luego pueden ser combinados de distintas maneras para dar lugar a esos otros comportamientos más complejos. Así, tal y como sucede en los organismos vivos, el robot aprende paulatinamente a realizar tareas y va usando su experiencia pasada para aprender cosas cada vez más complejas. Las distintas formas de combinarse de los módulos aquí planteadas son también obtenidas automáticamente siguiendo los mismos métodos que para obtener los comportamientos individuales, sin que el diseñador tenga que hacer otra cosa que establecer la relación, en términos energéticos, entre el robot y su entorno.

Se han planteado diversas formas de relaciones entre módulos: selección, modulación de entradas y modulación de salidas. Cada una de estas formas de relación añade posibilidades a la arquitectura global, tanto en términos de aumentar las capacidades como de aumentar la reutilización. Así, la selección permite obtener comportamientos de alto nivel cuyo funcionamiento consiste en la activación en cada momento del módulo adecuado de bajo nivel entre aquellos de los que se dispone. La modulación de salidas permite además combinar los módulos de bajo nivel disponibles para dar lugar a comportamientos nuevos, combinación lineal de los ya existentes. Los moduladores de salidas poseen además la capacidad de interpolar comportamientos para situaciones sensoriales, ya sean internas o externas, no contempladas en la evolución, con lo que el rango de comportamientos útiles que se pueden obtener a partir de otros ya existentes se amplía enormemente. Por su parte, la modulación de entradas permite alterar comportamientos ya existentes de forma que realicen su tarea original pero valorando de forma distinta la información sensorial a su disposición. Adicionalmente, se ha visto cómo una arquitectura cualquiera con estos módulos y tipos de relación puede normalizarse a una forma en la que sólo existan dos niveles de módulos, con lo que se puede aprovechar fácilmente el paralelismo hardware en el robot, de existir éste, para acelerar la ejecución del controlador global.

Complementando la posibilidad de reutilizar módulos para un robot dado (o distintos robots del mismo tipo), se ha propuesto un mecanismo que, mediante los conceptos de actuadores y sensores virtuales, permite reutilizar dichos módulos en robots de distinto tipo. Estos elementos de actuación y sensorización virtual pueden acoplarse a controladores ya existentes en dos niveles distintos. Pueden interponerse entre todos los módulos de un controlador y el conjunto de sensores y actuadores reales del robot, permitiendo así trasladar íntegramente un controlador de un robot a otro, o pueden utilizarse los sensores virtuales para todos los módulos del controlador excepto los de más bajo nivel, aquellos que toman el control de los actuadores, que son obtenidos de nuevo haciendo uso de las características sensoriales y actuadoras de cada robot en concreto.

Todos los algoritmos y mecanismos utilizados y desarrollados en este trabajo han sido implementados en un software de desarrollo propio al que se ha denominado SEVEN (*Simulation and EVolution ENvironment*) y que permite tanto llevar a cabo los procesos evolutivos para obtener los controladores como comprobar visualmente el funcionamiento de éstos en un entorno simulado, que también será construido desde el propio software, antes de trasladarlos al robot real. Asimismo, los controladores obtenidos en los ejemplos han sido probados en robots reales (un Rug Warrior y un Pioneer 2-DX) para verificar que la metodología desarrollada era eficaz en su objetivo de lograr controladores que llevasen a cabo comportamientos robustos sobre robots reales.

En definitiva, se han cumplido los objetivos planteados inicialmente en cuanto a establecer una metodología que permita la obtención automática e incremental de controladores para robots autónomos que posibilite la obtención de comportamientos complejos. Para ello se han seleccionado y desarrollado una serie de herramientas (cuyo adecuación ha sido estudiada) que permiten la aplicación de esta metodología, así como creado una arquitectura que la soporta mediante la capacidad de combinar comportamientos, de diversas formas perfectamente definidas, para dar lugar a otros más complejos. La arquitectura permite también reutilizar controladores de un robot en otro distinto y, adicionalmente, hemos visto formas de aumentar las capacidades de los bloques constructivos de la arquitectura respecto a las soluciones habituales empleadas en robótica autónoma.

En el trabajo futuro, un primer paso consistirá en integrar aparatos sensoriales más potentes, como puedan ser cámaras de vídeo, de forma que se puedan aplicar estas técnicas para obtener comportamientos más complejos, lo cual no es alcanzable con los sensores que poseen los robots que estuvieron disponibles para la realización de este trabajo. Sin duda, aquí tendrá una importancia decisiva el concepto de sensor virtual, puesto que, por ejemplo, no cabe esperar que todos los módulos que componen un controlador tengan que trabajar directamente con los datos recibidos por una cámara.

Se estudiarán también mecanismos de desarrollo de genotipo a fenotipo frente a las codificaciones directas empleadas en este trabajo, de forma que el algoritmo evolutivo pueda obtener estructuras de control más complejas con una codificación genotípica compacta.

También se espera integrar técnicas de aprendizaje *online*, en tiempo de vida del robot, como puede ser aprendizaje por refuerzo en alguno de los módulos o en las estructuras de modulación. Otro mecanismo de adaptación muy interesante y que se espera poder integrar con este trabajo es el presentado por Francisco Bellas en su tesis de doctorado "MDB: Mecanismo cognitivo darwinista para agentes autónomos", en la cual se obtiene en tiempo de vida y de forma automática un modelo de entorno sobre el que se evalúan estrategias de actuación. Así, se espera poder desarrollar una arquitectura cognitiva para robots en la cual se utilice en la parte de más alto nivel el

MDB para, utilizando a su vez las arquitecturas multimódulo aquí desarrolladas, construir modelos del entorno, generar estrategias en distintas escalas de tiempo y utilizarlas cuando sea necesario, funcionando asíncronamente respecto a los módulos de bajo nivel para no afectar a la capacidad de reacción del sistema.

Apéndices

En estos apéndices se presenta una descripción de los robots empleados para probar la metodología, el Rug Warrrior y el Pioneer 2-DX, y las herramientas desarrolladas integradas en el entorno SEVEN (*Simulation and EVolution ENvironment*), el cual permite obtener los controladores siguiendo la metodología expuesta en la tesis, pudiéndolos probar en simulación antes de trasladarlos al robot real.

El Rug Warrior es un robot de bajo coste, con un aparato sensorial y actuador de baja calidad y de comportamiento muy ruidoso. Su utilización, sin embargo, ha permitido comprobar el correcto funcionamiento de las técnicas desarrolladas para trasladar con éxito los controladores obtenidos en simulación a robots reales, para contrarrestar el efecto de sensores y actuadores de baja calidad y para compensar situaciones de infrasensorización. En el apartado de los apéndices dedicado a él veremos la descripción del robot y del funcionamiento de sus componentes, así como el modelado que de éstos se ha hecho en SEVEN y de los tipos de ruido que ha sido necesario aplicar en la simulación.

El Pioneer 2-DX es un robot más moderno y mucho más capaz. De mayor tamaño, sus sensores y actuadores son mucho más precisos, hasta el punto de hacer innecesarios los tipos de ruido utilizados para obtener comportamientos en el caso del Rug Warrior. Su utilización, como hemos visto, ha planteado, sin embargo, nuevos problemas debido a su mayor número de sensores que se traduce en espacios de búsqueda de mayor dimensionalidad. Aquí veremos sus principales características y cómo se han modelado en SEVEN.

Si se quería comprobar el correcto funcionamiento de la metodología desarrollada así como de las herramientas empleadas para llevarla a cabo era necesario disponer de un software que, por una parte, permitiese evolucionar los controladores utilizando los algoritmos deseados y soportando la arquitectura de comportamientos desarrollada y, por otra parte, actuase como simulador para comprobar el funcionamiento de los controladores antes de trasladarlos al robot real y permitiese crear mundos simulados en los cuales llevar a cabo las evoluciones. Dado que gran parte del trabajo aquí presentado implica herramientas y métodos propios, como la arquitectura de los comportamientos, parte de las técnicas para la transferencia de controladores al robot real, etc., y no existía un software que aunase ambas características de evolución más simulación interactiva, se desarrolló un software propio que se denominó SEVEN con todas las características deseadas. En la última parte de estos apéndices veremos todas las características de este software.

Robot Rug Warrior

Este robot es un robot circular, pequeño, sencillo y barato construido en torno a un microcontrolador MC68HC11A1 [58][59]. Posee como actuadores dos motores de corriente continua (DC) acoplados a sendas ruedas y un altavoz. Como sensores tiene un receptor de infrarrojos (con dos emisores asociados), dos sensores de luz, uno de transiciones de calor, tres de contacto, dos de velocidad (uno por rueda) y un micrófono.

El tamaño del robot es lo suficientemente grande como para poder desenvolverse con libertad por un entorno humano cerrado, como puede ser una oficina o laboratorio. Los sensores son, por el contrario, de baja calidad, muy ruidosos e imprecisos. Así, por ejemplo, el receptor de infrarrojos es binario, es decir, devuelve un uno si detecta un objeto y un cero si no lo detecta, pero no devuelve ningún tipo de información referente a la distancia a la que se encuentra dicho objeto, y el sensor de transiciones de calor no proporciona información direccional, sólo detectaremos cambios en la temperatura ambiente sin saber de dónde proceden. La escasa calidad y número de los sensores determinará fuertemente cómo han de ser los comportamientos para él obtenidos.

Comparado con el Khepera, uno de los robots más empleados en robótica basada en comportamientos, el Rug Warrior, debido a su mayor tamaño, puede ser utilizado en entornos reales, y no construidos a medida, donde el mayor rango de objetos y materiales hace que el ruido presente en sensores y actuadores sea considerablemente mayor. Debido también a su mayor tamaño, las inercias son mucho mayores. El Khepera posee, en su configuración básica, una menor variedad de sensores, pero un mayor número total de ellos y son además de mayor calidad, lo que por una parte reduce el número de posibles comportamientos y por otra redunda en una mejor y más completa información del entorno por parte del robot en aquellos comportamientos posibles. La obtención de comportamientos con el Rug Warrior es, pues, considerablemente más problemática que con el Khepera, pero permite una mayor variedad de comportamientos y probar éstos en un mundo no prefabricado. Además, es una buena herramienta para comprobar la robustez de las técnicas empleadas para obtener los comportamientos.

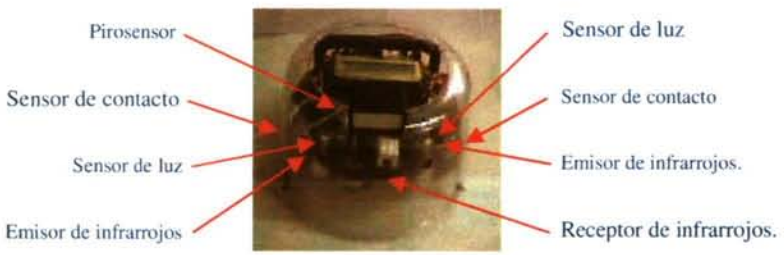


Figura 69: Sensores del Rug Warrior.

Características

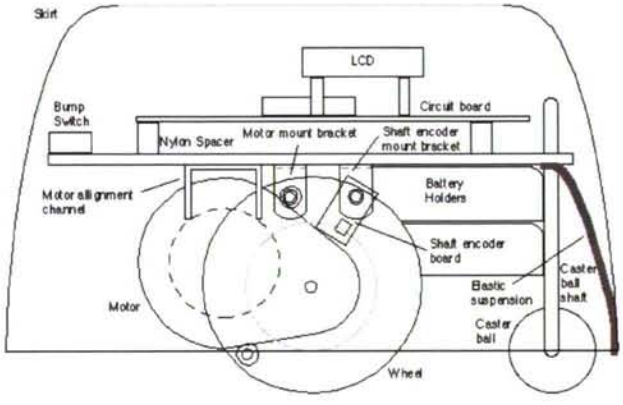


Figura 70: Esquema del Rug Warrior.

Como se puede ver en las figuras 69 y 70, el Rug Warrior es un robot circular de pequeñas dimensiones (18.54 cm de diámetro y 12.06 cm de alto), aunque suficientes para poder desenvolverse por un entorno de oficina. La carcasa de plástico transparente sirve para que, con los tres sensores de contacto de que dispone, se pueda detectar una colisión en cualquier lugar del contorno del robot. A cambio, dicha carcasa produce en ocasiones reflejos que degradan el funcionamiento de los sensores de luz. La pequeña esfera localizada en la parte trasera del robot sirve para que éste mantenga el equilibrio. El robot tiene una pequeña pantalla LCD, que indica su estado y que además sirve de interfaz de salida para los programas que se ejecutan en él.

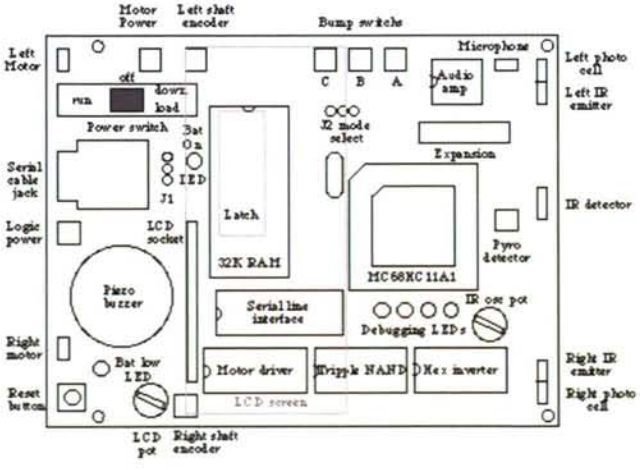


Figura 71: Esquema de la circuitería del Rug Warrior.

Como ya se ha mencionado previamente, el Rug Warrior está diseñado en torno al microcontrolador Motorola MC68HC11A1. Un microcontrolador es un circuito integrado que combina en su interior un pequeño microprocesador y circuitería adicional de procesado de señales, como por ejemplo convertidores analógico/digitales (ADC), contadores de pulsos o interfaces serie. En concreto, el MC68HC11A1 incluye un procesador que puede funcionar a velocidades entre los 921.6 KHz. y los 3 MHz. (en el caso del Rug Warrior funciona a 2 MHz.), con un bus de direcciones de 16 bits y un bus de datos de 8 bits multiplexado con el de direcciones. El número de ciclos de reloj necesarios para ejecutar una instrucción oscila entre 2 y 41, por tanto el tiempo oscila entre 1 y 20.5 microsegundos. Aunque el microprocesador puede direccionar 64 Kb. de RAM, el Rug Warrior sólo tiene 32 Kb., a los que hay que añadirles 256 bytes de RAM y 512 bytes de EEPROM en el propio microcontrolador (el Rug Warrior original llevaba el MC68HC11A0 que se diferencia del A1 en la ausencia de la EEPROM). El MC68HC11A1 incluye 8 ADC de 8 bits cada uno, un contador de pulsos, una interfaz serie y varios puertos digitales multifunción. En la figura 71 se puede ver la arquitectura de la placa madre del robot, la posición de la CPU, la memoria y los sensores. En la figura 72 se muestra un esquema del microcontrolador.

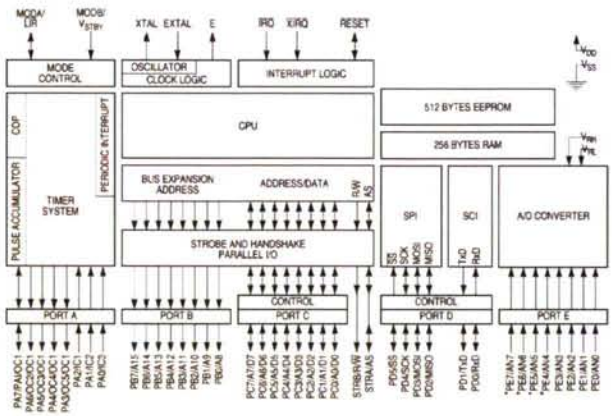


Figura 72: Esquema del microcontrolador del Rug Warrior.

El robot tiene como interfaz con el programador un lenguaje, el *Interactive C* (IC), el cual es un subconjunto del lenguaje C. Sólo existen los tipos de datos char (1 byte), int (2 bytes), long (4 bytes) y float (4 bytes). No hay gestión dinámica de memoria, ni estructuras de datos exceptuando el array[6]. La única entrada de datos posible son los sensores, y la salida de datos es siempre por la pantalla LCD. Dicho lenguaje es además interpretado, no compilado[7]. Una vez encendido el robot, se carga en él el IC y una serie

6 Existe una versión comercial del IC que permite el uso de arrays bidimensionales, estructuras y macros.

7 Existe también una versión del IC que permite compilar directamente a código máquina, pero, desgraciadamente, está en una fase muy primitiva (no permite que una función llame a otra, por

de rutinas que se encargan de la gestión de procesos y de la interacción con el usuario (mediante línea de comandos). A partir de ese momento se pueden cargar los programas de usuario. Cada programa, en el proceso de carga, es compilado a un lenguaje de pila, el cual es interpretado y convertido a código máquina únicamente en tiempo de ejecución. Esto tiene las siguientes ventajas:

- Detección de errores en tiempo de ejecución.
- Facilidad de implementación del compilador.
- Facilidad en la portabilidad del IC a otros procesadores.
- Código objeto pequeño.
- Facilidad en la implementación del multiproceso. Un proceso viene definido únicamente por su pila y su contador de programa, por lo que cambiar de proceso sólo implica cambiar el puntero de pila y el contador de programa.

  El inconveniente es una mayor lentitud en la ejecución de los programas.

poner un ejemplo) y el autor ha abandonado su desarrollo.

Sensores

Los sensores del Rug Warrior son variados aunque de escasa calidad. Dispone de dos sensores de luz, un receptor de infrarrojos (con dos emisores asociados), un sensor de transiciones de calor, tres sensores de contacto, dos sensores de velocidad (uno por rueda) y un micrófono. Veremos a continuación una descripción de los sensores utilizados y los problemas que generan las particularidades de su funcionamiento.

Contacto

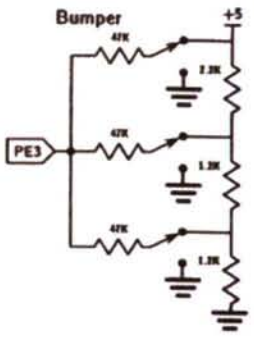


Figura 73: Esquema de los sensores de contacto del Rug Warrior.

El Rug Warrior tiene tres sensores de contacto, cuyo esquema de funcionamiento se puede observar en la figura 73. Las resistencias están calculadas de tal manera que el ADC asociado al puerto PE3 devuelve un byte, donde cada uno de los tres bits menos significativos se corresponde con uno de los sensores, estando a uno si hay contacto y a cero en caso contrario.

Para que los sensores de contacto sean útiles, éstos deben detectar una colisión en cualquier punto del contorno exterior del robot. Para ello, los diseñadores optaron por situar estratégicamente tres sensores, separados entre sí aproximadamente 120º, y colocar encima del robot una carcasa de plástico rígido transparente que posee una cierta holgura en su movimiento, de tal manera que, en caso de colisión, la carcasa se mueve y activa uno o más de los sensores de contacto. Esta carcasa, sin embargo, empeora el funcionamiento de los sensores de luz, ya que en ocasiones produce reflejos.

Estos sensores de contacto, al ser puramente mecánicos, no presentan apenas ruido. Pueden aparecer problemas, sin embargo, si el objeto que colisiona con el robot lo hace por la parte superior o por el borde inferior de la carcasa de plástico, en cuyo caso, el movimiento de ésta no será suficiente para activar los sensores.

Dado que el robot sólo posee dos sensores de infrarrojos, que aun en el caso de que un objeto se encuentre en el rango de uno de ellos éstos no indican la distancia a la que se encuentran los objetos, y que además cuando dichos objetos están muy cerca del robot los sensores de infrarrojos no los detectan aunque estén enfrente (debido a que sólo hay un receptor centrado entre los dos emisores), los sensores de contacto son imprescindibles para saber, en caso de colisión, donde está el objeto y, por tanto, hacia donde moverse.

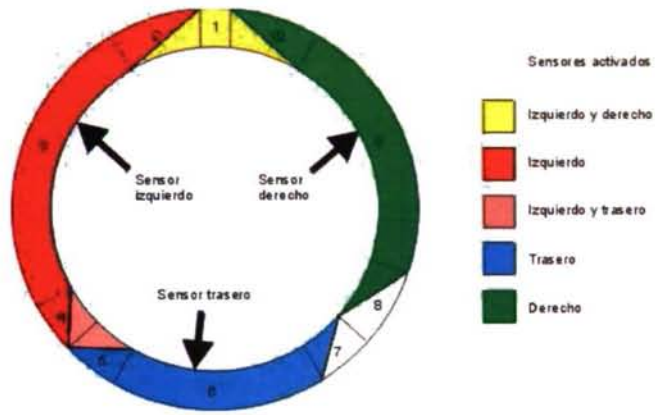


Figura 74: Modelo de los sensores de contacto del Rug Warrior.

En la figura 74 se representa el modelo que se ha realizado en SEVEN de estos sensores. La circunferencia exterior simboliza el contorno del robot con carcasa, mientras que la interior el contorno del robot sin ella. Las flechas negras indican la posición de los sensores de contacto. Los colores codifican las zonas de activación de los sensores. Se puede observar cómo existen cuatro tipos de zonas en el contorno del robot según el punto en el que se produzca la colisión con un objeto. En la zona 1 siempre se activan los sensores izquierdo y derecho. En las zonas 3, 6 y 9 sólo un sensor es activado. En las zonas 2, 4, 5 y 10 se activa siempre al menos un sensor, activándose otro sensor tan sólo cuando la colisión es lo suficientemente fuerte. Finalmente, en las zonas 7 y 8 como mucho se activará un sensor si la colisión es intensa.

Para no complicar en exceso el modelo en el simulador, y como en cualquier caso las zonas indicadas son aproximadas y también hay ruido en los valores sensados (dependiendo, por ejemplo, de la altura a la que se produce la colisión), sólo se tiene en cuenta la circunferencia exterior y cuando se produce una colisión se calcula un número aleatorio entre 0 y 1, y si ese número es menor que una determinada probabilidad, se activa el sensor correspondiente. Esta probabilidad depende de la posición de la colisión y de cada sensor y se asigna utilizando como referencia la figura 74 (la circunferencia exterior tiene asociada probabilidad 0 y la interior 1, y según el valor del número aleatorio calculado, el punto de colisión y el color correspondiente, se activarán 0, 1 o 2 sensores).

Infrarrojos

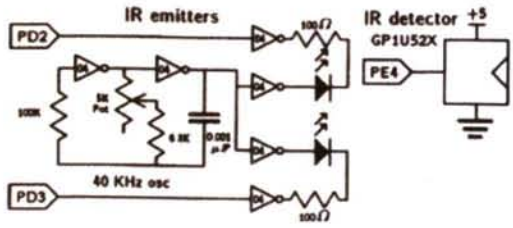


Figura 75: Esquema de los sensores de infrarrojos del Rug Warrior.

Los sensores de infrarrojos del Rug Warrior están formados por dos emisores, siendo cada emisor un LED que emite con una longitud de onda de 880 nanometros, y un único receptor. Este receptor responde a una portadora de 40 KHz., por lo que los emisores deben emitir en dicha frecuencia (en teoría, porque en la práctica el receptor, por imprecisión en el proceso de fabricación, puede comportarse mejor con otras frecuencias, como así sucedía en el Rug Warrior empleado). Dado que sólo hay un receptor, los emisores deben emitir alternándose en el tiempo: primero emite uno, se comprueba si el receptor detecta algo, luego se hace lo mismo con el otro emisor y así repetidamente. Notar que, aunque el detector está conectado a un puerto que tiene asociado un ADC, esta característica no es usada, ya que la salida del receptor es digital: 0 voltios (detecta) o 5 voltios (no detecta).

Como ya hemos mencionado, los sensores de infrarrojos del Rug Warrior son de muy baja calidad. Son binarios, sólo indican la presencia o no de un objeto, y sólo son dos emisores con un único receptor común situado entre los emisores, por lo que el rango de detección es muy estrecho. Debido a estas características de los sensores de infrarrojos, los comportamientos que los usan requieren en ocasiones manejar información temporal, ya que, por ejemplo, el robot no sabe si se está acercando o alejando de un objeto y para ello necesita saber si el objeto fue detectado en instantes anteriores. Pero ni siquiera el uso de información temporal permite manejar la información proporcionada por estos sensores con seguridad, ya que su funcionamiento es extremadamente ruidoso. Además, mientras el ruido en sensores de infrarrojos que devuelvan un valor proporcional a la distancia del objeto detectado simplemente causa que la distancia sea errónea pero se sigue indicando la presencia del objeto (salvo que éste esté en el límite de detección), en los sensores utilizados el ruido puede causar que no se detecte un objeto que sí se detectó en el instante anterior y que ahora está aún más cerca, causando la falsa impresión de que nos alejamos de él. Finalmente, el anclaje de los emisores en la placa es muy débil, y la simple manipulación del robot puede moverlos, cambiando su rango de detección y alterando el comportamiento del robot. Debido a todo esto, y como se pudo observar en el apartado 4.4.1, los comportamientos

que los utilizan, cuando se desea evitar las colisiones, realizarán tácticas muy conservadoras en sus movimientos.

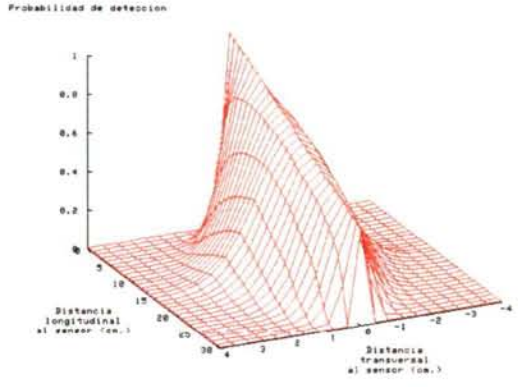


Figura 76: Modelo de los sensores de infrarrojos del Rug Warrior.

Al igual que sucedía con los sensores de contacto, el modelado es probabilístico y la probabilidad de que un objeto sea detectado dependerá de la posición relativa al sensor en la que se encuentre. En la realidad existe una asimetría en las zonas de detección, debida a la existencia de un único receptor para dos emisores, y a su ubicación entre ambos. En el simulador se ha obviado dicha asimetría para simplificar la simulación.

Se han incluido los siguientes tipos de ruido en la simulación:

- La posición de los sensores varía al principio de cada evaluación del robot mediante la suma a su orientación original de una cantidad perteneciente al conjunto $\{-\pi/8, 0, \pi/8\}$. Esto, como se comenta en el apartado 4.4.1, se hace para simular la fragilidad de las conexiones de los sensores, que puede dar lugar a que la simple manipulación del robot por una persona altere ligeramente su posición.
- Otro ruido sistemático utilizado es el consistente en modificar el alcance máximo de los sensores haciendo que en ocasiones éste se reduzca al 70% del indicado en la figura 76. La utilidad de este ruido es hacer los comportamientos tolerantes a distintos tipos de objetos, ya que la rugosidad y el color de cada superficie dan lugar a rangos de detección diferentes. De todas formas, pese a la utilización de este ruido, es preciso notar que el funcionamiento de los comportamientos que usan los sensores de infrarrojos no es perfecto, debido a la propia naturaleza de éstos. Así, por ejemplo, difícilmente detectarán un objeto completamente negro.

Luz

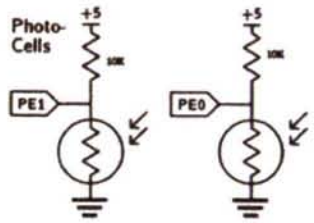


Figura 77: Esquema de los sensores de luz del Rug Warrior.

Los sensores de luz están formados por una fotorresistencia, o fotocélula, la cual se comporta como un potenciómetro, con la excepción de que el cambio en la resistencia es producido por un cambio en la intensidad de la luz que incide en ella, y devuelven un valor en el intervalo [0,255] (a menor número mayor intensidad).

Estos sensores son bastante ruidosos, debido sobre todo a la carcasa de plástico que lleva el robot para el correcto funcionamiento de los sensores de contacto. Su conexión con la placa del robot es igual de endeble que en el caso de los emisores de infrarrojos, con lo que poseen el mismo problema asociado a esa debilidad en la conexión: un golpe en la manipulación del robot puede hacer que éstos "miren hacia otro lado" y modificar así el comportamiento del robot que fue desarrollado con los sensores en una determinada orientación.

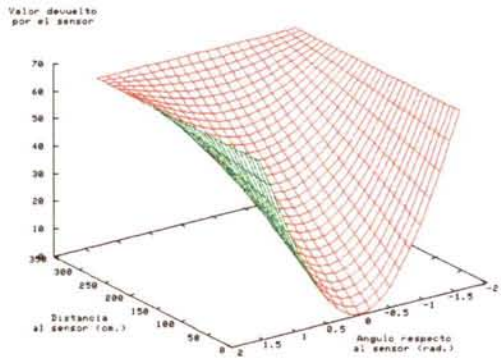


Figura 78: Modelo de los sensores de luz del Rug Warrior.

Los sensores de luz, a diferencia de los infrarrojos, no son binarios y sí devuelven un valor dependiente, en este caso, de la intensidad luminosa percibida. Para modelarlos

se han realizado mediciones con tres niveles de luz ambiente distintos y utilizando como foco luminoso una bombilla de 100 W. La respuesta de estos sensores se ha aproximado con la función de la figura 78. Esta función varia en función del sensor en cuestión y de la luz ambiente sensada[8].

Se han incluido los siguientes tipos de ruido en la simulación:

- Al valor que le correspondería según el modelo de la figura 78 se le suma/resta una cantidad aleatoria como mucho del 5 %, ya que éste es el tanto por ciento de variación en las mediciones que se ha observado en el robot real.
- Ruido sistemático en su orientación, idéntico en motivación y forma de aplicación al empleado con los sensores de infrarrojos.
- Ruido sistemático también similar a uno de los tipos de ruido visto en los sensores de infrarrojos, el que variaba el alcance máximo, y que consiste en alterar al principio de cada evaluación la intensidad de la luz ambiente, seleccionando un valor entre los elementos de un conjunto. Esto es para permitir que los comportamientos que usan estos sensores sean tolerantes a cambios en la iluminación del entorno y no se limiten a establecer relaciones entre valores concretos de intensidad y acciones, sino que las relaciones sean entre cambios de intensidad y acciones. Los conjuntos de valores utilizados en este ruido son {85, 125, 255} para el sensor izquierdo y {65, 95, 255} para el sensor derecho. El hecho de que los conjuntos sean distintos se debe a la distinta respuesta de los sensores de la unidad Rug Warrior empleada (el sensor derecho es más sensible que el izquierdo).

Calor

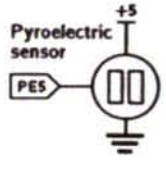


Figura 79: Esquema del sensor térmico del Rug Warrior.

El sensor de transiciones de calor, o pirosensor, está formado por dos cristales cuyas propiedades eléctricas se alteran con el calor. Con una temperatura normal devuelve, cuando no hay movimiento delante de él, un valor constante. Cuando un objeto con una mayor temperatura pasa por delante de él, el sensor devuelve un valor mayor, mientras que si el objeto es más frío, devuelve un valor menor. Sin embargo, una vez que la temperatura deja de variar, el sensor tarda un tiempo en volver a mantener un valor

8 La gráfica se corresponde con el sensor derecho cuando éste sensa el valor 65 como luz ambiente.

constante, y está durante unos instantes oscilando sobre dicho valor constante. Esto lo hace muy difícil de modelar; sin embargo, dado que en la práctica el robot está continuamente moviéndose, este problema aparece relativamente pocas veces. Obtener comportamientos útiles con este sensor es complejo, ya que sólo hay uno y no es posible saber en qué dirección se acerca o se aleja un objeto.

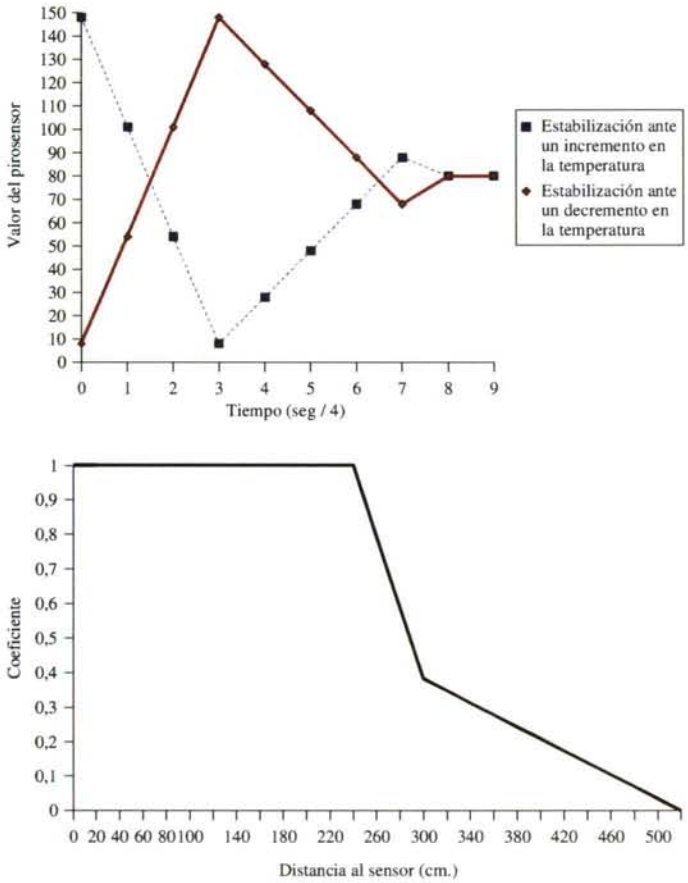


Figura 80: Modelo del pirosensor del Rug Warrior.

En la figura 80 está representado el modelado del sensor de transiciones de calor, o pirosensor, ante un aumento o decremento en la temperatura percibida. Lo primero que se debe hacer notar es que la información obtenida con el pirosensor que incorpora el robot utilizado es distinta a la del Rug Warrior descrito en [59], ya que no distingue el acercamiento o alejamiento de un objeto, sino únicamente transiciones de calor. Además, tampoco distingue cuando estas transiciones son de un objeto que aparece por la derecha o por la izquierda. El segundo factor importante en el comportamiento del pirosensor se observa en la primera de las gráficas. El sensor, una vez que se ha

producido el cambio en la temperatura ambiente, oscila durante unos pocos segundos hasta que vuelve a estabilizarse. Esto es importante, ya que el alejamiento o acercamiento del mismo objeto u otro distinto antes de la finalización del tiempo de estabilización puede ser, dependiendo del momento concreto en el que se produzca, difícil de distinguir por el robot. Finalmente, la reacción del pirosensor varía, como es lógico, con la distancia a la que se encuentra el objeto con una temperatura diferente. Para tener en cuenta esto, el valor que corresponda en la primera gráfica es multiplicado por un coeficiente, cuyo valor se obtiene con otra función que puede verse en la segunda gráfica. Ambas son una aproximación de los datos medidos en el robot real.

El pirosensor se ha mostrado como el sensor más ruidoso de todos, si exceptuamos el de velocidad. Esto se debe a su excesiva sensibilidad, lo que provoca que, en la práctica, el valor devuelto por el sensor esté oscilando continuamente, en mayor o menor medida, en un entorno habitual de oficina o laboratorio, con objetos que desprenden calor como ordenadores, radiadores de calefacción, etc.

Así, se han utilizado los siguientes ruidos en el simulador:

- Se ha multiplicado el ruido clásico consistente en sumar o restar pequeñas cantidades, siendo la cantidad sumada o restada de hasta un 15 %, eso sí, obviamente sin superar los valores máximo y mínimo proporcionados por el sensor.
- Se anula periódicamente (un 15% de las ocasiones) de forma aleatoria el efecto del cambio en la distancia entre el robot y el objeto. Es decir, aunque el robot se esté moviendo continuamente con respecto al objeto, en ocasiones se simula que el acercamiento/alejamiento no es suficiente para ser notado y se continua con el ciclo correspondiente (ver figura 80). Esto sucede en la realidad y es debido a que el calor desprendido por objetos o personas no posee un gradiente homogéneo ni en tiempo ni en distancia, con lo que un acercamiento o alejamiento de x cm puede ser apreciable en un momento dado y despreciable en el siguiente.
- Cuando no se detecta ningún objeto con una temperatura distinta a la ambiental, en vez de devolver 80, su valor una vez que está estabilizado, se devuelve aleatoriamente cualquier valor entre 80 y 140, para simular la presencia en el entorno de objetos cálidos pero que no son los que queremos perseguir. Estos dos últimos tipos de ruido dificultarán mucho la obtención de comportamientos por su carácter imprevisible y engañoso.

Velocidad

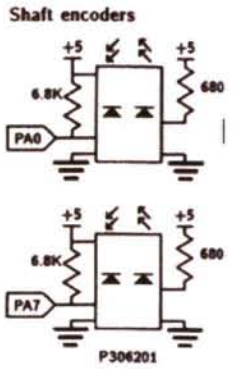


Figura 81: Esquema de los encoders del Rug Warrior.

Los sensores de velocidad consisten en una superficie con rayas alternas de color blanco y negro (foto-reflectores), que se encuentran unidas a las ruedas. Enfrente a cada una de ellas, pero fijados al chasis, se encuentran unos LED y unos fototransistores que perciben la transición entre los dos tipos de franjas a medida que las ruedas giran, generando una señal de pulsos que pueden ser contados haciendo uso del contador de pulsos integrado en el MC68HC11A1. La resolución de los foto-reflectores del Rug Warrior es de 16 pulsos por vuelta completa de una rueda.

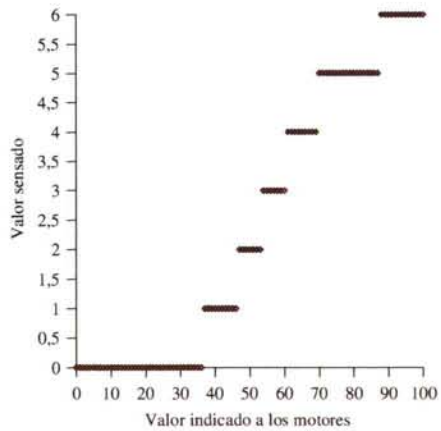


Figura 82: Modelo de los sensores de velocidad del Rug Warrior.

El funcionamiento de los sensores de velocidad es sencillo y podría parecer que no son susceptibles de tener mucho ruido; sin embargo, hay dos factores que limitan su utilidad. Uno de ellos es el hecho de que, salvo que se tomen las medidas cada mucho

tiempo, devuelven un valor idéntico para velocidades significativamente diferentes. El otro es que, si la superficie es resbaladiza, se producen pequeños patinazos, especialmente en arrancadas y paradas bruscas, que hacen que el valor sensado no sea indicativo de la distancia recorrida. Nuevamente este problema se reduce si las medidas se toman en intervalos de tiempo lo suficientemente grandes, pero esto disminuye también el rango de posibles usos para los sensores.

En la figura 82 se aprecia el valor medio devuelto por los sensores de velocidad en función de la distancia recorrida. Sin embargo, este valor puede oscilar enormemente, por lo que el ruido incorporado en el simulador es muy fuerte, de hasta un 50 % (ruido tradicional, no sistemático). Esto limita ciertamente la utilidad de estos sensores.

Actuadores

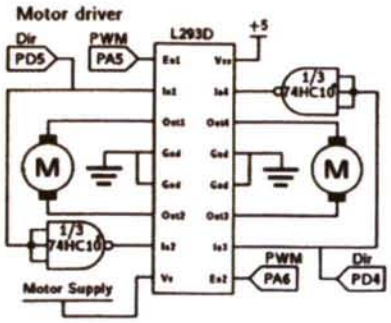


Figura 83: Esquema de los motores del Rug Warrior.

El Rug Warrior posee dos motores de corriente continua de 6V, conectado cada uno de ellos a una de las ruedas (de 6.35 cm de diámetro). Los motores reciben la alimentación de forma separada respecto a la circuitería, para no introducir ruido adicional en ésta. Además, la circuitería posee un regulador de voltaje de tal manera que el voltaje proporcionado a la circuitería no exceda nunca los 5 voltios.

Existe una función para indicarle a un motor la velocidad a la que tiene que hacer girar las ruedas, siendo ésta un número en el intervalo [-100,100]. Destacar que dos unidades del mismo modelo no tienen por qué comportarse igual. De hecho, los dos motores del Rug Warrior utilizado aplican velocidades ligeramente distintas a las ruedas ante llamadas con idéntico valor a la función que controla la velocidad.

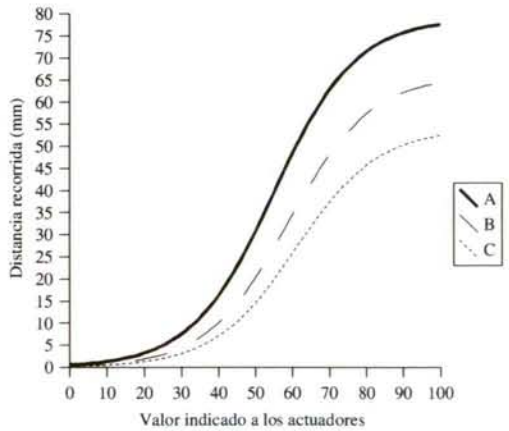


Figura 84: Modelado de los actuadores del Rug Warrior.

El conjunto de los actuadores del Rug Warrior es también muy ruidoso. Los motores no son de precisión, como pueda ser el caso, por ejemplo, del Khepera, y su peso es el suficiente como para que la inercia y los patinazos sean un problema apreciable. Además, la distancia recorrida por el robot varía de una forma no líneal a medida que las baterías se descargan y el voltaje que proporcionan varía. Debido a esto, se han utilizado los siguientes tipos de ruido en la simulación:

- Ruido aditivo de media cero de hasta un 15%.
- Ruido sistemático consistente en seleccionar aleatoriamente, al principio de cada evaluación del robot en el proceso evolutivo, una de las curvas de respuesta que se ven en la figura 84. De esta forma se obtienen controladores tolerantes a cambios, dentro de una margen razonable, en la carga de las baterías.
- Adicionalmente, un defecto en el Rug Warrior empleado ha obligado a introducir un ruido sistemático adicional. La rueda izquierda recorre habitualmente, a igualdad de potencia indicada a los motores, menos distancia que la derecha. Esta diferencia varia además aleatoriamente dentro de un rango relativamente amplio. Para simular este comportamiento la diferencia de la rueda izquierda con la derecha es seleccionada aleatoriamente al principio de cada evaluación, pudiendo ser del 10% o ninguno. Se podría pensar por qué no incluir este ruido también en la rueda derecha y así hacer los comportamientos todavía más robustos, pero esto traería consigo un problema de difícil solución. La diferencia entre la distancia recorrida por las ruedas es lo suficientemente grande como para que, de no aplicar una corrección, el robot se quede dando círculos si está en un espacio alejado unos dos metros de cualquier pared. Pero esta diferencia no es lo suficientemente grande como para que los ruidosos sensores de velocidad proporcionen una referencia válida, salvo que se tomen las medidas en intervalos de tiempo muy grandes. Pero tomar estas medidas en intervalos de tiempo grandes implicaría que durante todo ese intervalo de tiempo el robot estaría realizando movimientos incorrectos. Cuando quisiese rectificar podría ser demasiado tarde. Así, si la rueda izquierda siempre va más lenta, el robot evoluciona para aplicar en todo momento una estrategia correctora, pero si en cambio ambas ruedas pudiesen tener ese problema, el robot no sabría a que atenerse. Por esto, y teniendo en cuenta que al fin y al cabo la rueda derecha nunca recorre menos distancia que la izquierda no se ha incluido este ruido en dicha rueda.

Robot Pioneer 2-DX

El Pioneer 2, fabricado por *ActivMedia Robotics* [2], es el otro robot empleado en esta tesis para ejemplificar los comportamientos y demostrar la viabilidad de la metodología desarrollada. Existen varias configuraciones del Pioneer 2, como el Pioneer 2-DX con 2 ruedas motrices y una tercera de apoyo, y el Pioneer 2-AT con 4 ruedas en lugar de 2. Además, el equipamiento puede variar. Así, el robot puede llevar un equipo de comunicación por radio, ordenador integrado, uno o dos arrays de sónares, pinza, sensores de contacto y cámara de vídeo. En la figura 85 se puede ver una foto con el Pioneer 2-DX y el Pioneer 2-AT. El Pioneer 2 utilizado en este trabajo es el 2-DX con un anillo de sónares (formado por dos arrays, uno frontal y otro trasero, de 8 elementos cada uno) y se ha utilizado un ordenador portátil, conectado por puerto serie al microcontrolador del Pioneer 2-DX, como soporte hardware donde ejecutar los controladores obtenidos.

Como el Rug Warrior, posee como actuadores dos motores conectados a sendas ruedas y una tercera rueda de apoyo, más pequeña, situada en la parte trasera. Pero ahí se acaban las similitudes. Es un robot considerablemente mayor y con unas capacidades también mucho mayores. Aunque con un aparato sensorial menos diverso, éste es mucho más preciso y el robot, en líneas generales, mucho menos sensible a los problemas derivados del ruido. Además, su capacidad computacional no sólo es mucho mayor, sino que es escalable, ya que el S.O. del robot sólo se encarga de aplicar los comandos y de proporcionar información a otro ordenador que es el que realiza toda la lógica de los controladores. Ese ordenador puede ir integrado en el robot, consistir en un portátil acoplado encima de éste, como ha sido el caso de este trabajo, o ser cualquier ordenador o red de ordenadores que se comunique con el robot vía radio.

Figura 85: Foto del Pioneer2-DX (a la izquierda) y del Pioneer 2-AT (a la derecha).

Características

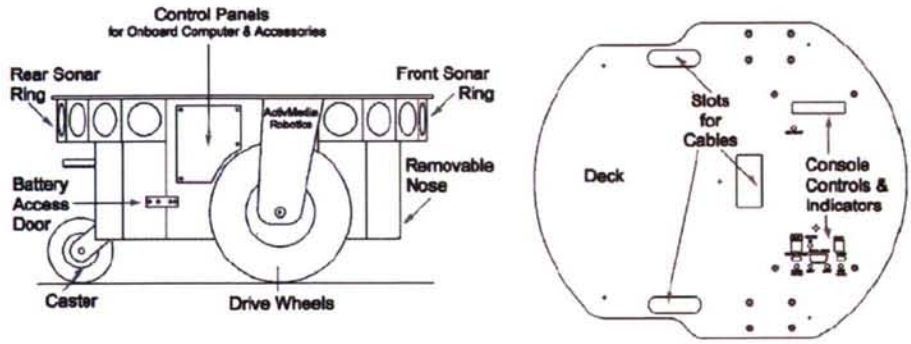


Figura 86: Esquema del Pioneer 2-DX.

El Pioneer 2-DX es un robot con base octogonal (su longitud y anchura máximas son 500 mm y 380 mm respectivamente) que se desplaza gracias a dos actuadores conectados a sendas ruedas. El robot usa una tercera rueda de apoyo para mantener la estabilidad. En la figura 86 se puede ver un esquema del Pioneer 2-DX y sus principales características.

El sistema operativo se ejecuta en el controlador del robot, pero éste se ocupa sólo de las tareas más básicas, como recibir los datos de los sensores y proporcionarlos a las aplicaciones que los necesiten, y recibir de éstas los comandos de movimiento y aplicarlos a los actuadores. Ambas tareas sufren un procesamiento por parte del sistema operativo. Los datos de los sónares no se proporcionan directamente a la aplicación, sino que se procesan para averiguar la distancia a la que se encuentra el objeto sensado, devolviendo ésta a la aplicación. Igualmente, el robot espera recibir como comandos para los actuadores las velocidades deseadas y el sistema operativo se encarga de traducir esas velocidades en los comandos pertinentes para los actuadores, haciendo las correcciones necesarias en el tiempo para que las órdenes dadas se cumplan. Por ejemplo, si se indica una velocidad de 50 mm/s, el sistema operativo tratará de mantener dicha velocidad mientras no se le comunique otra, enviando los comandos necesarios a los actuadores, que pueden cambiar en el tiempo si los valores de los *encoders* le indican que la velocidad real no es la que debiera, por ejemplo, porque el robot está patinando. Los controladores, por tanto, no se ejecutan en el microcontrolador, como en el caso del Rug Warrior, sino en un ordenador u ordenadores adicionales. Aunque el Pioneer 2-DX puede llevar un ordenador embebido en el propio chasis, en este trabajo hemos optado por usar un ordenador portátil acoplado sobre el robot por ser una solución más flexible. En la figura 87 se puede ver un diagrama de la placa del microcontrolador.

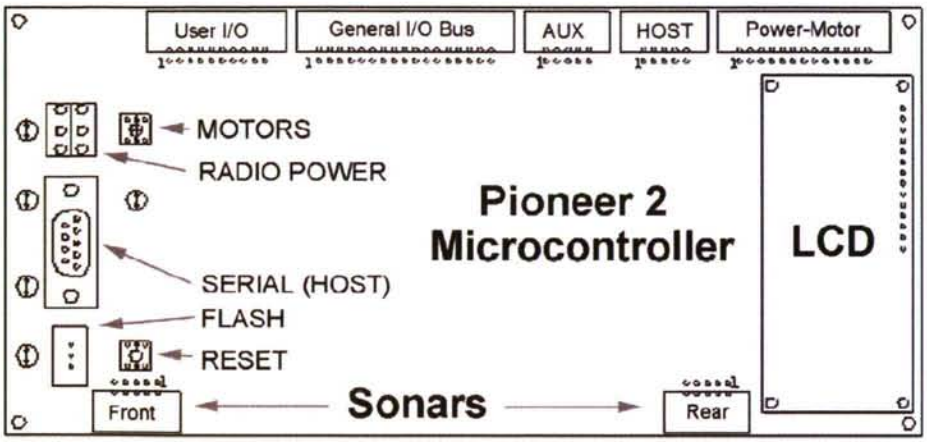


Figura 87: Esquema de la circuitería del Pioneer 2-DX.

A la hora de programar, *ActivMedia Robotics* proporciona varias librerías. Dos son las principales: *ARIA* y *Saphira*. La segunda proporciona un entorno de alto nivel en una arquitectura cliente-servidor, en la que la parte servidor actúa como simulador del Pioneer 2-DX y la parte cliente define el control en un lenguaje propio desarrollado a tal efecto. Dado que el trabajo realizado en esta tesis implicaba una arquitectura propia, se ha utilizado la otra librería, *ARIA*, la cual proporciona un acceso a más bajo nivel al sistema operativo del Pioneer 2-DX y además proporciona el código fuente, lo que es útil en determinadas ocasiones para averiguar el funcionamiento del sistema operativo del robot y cómo se relaciona con las aplicaciones.

Las ventajas del Pioneer 2-DX respecto al Rug Warrior son varias:

- El robot tiene unas mayores dimensiones, con lo que es más fácil que se desenvuelva por entornos no controlados sin que su integridad física corra peligro. Por contra, las inercias son mayores, con lo que las aceleraciones y desaceleraciones son más complicadas porque llevan más tiempo. El propio sistema operativo del robot impone unos límites en estos factores para no dañar la mecánica del robot y para que su sistema interno odométrico funcione correctamente. Esto hecho ha de tenerse en cuenta en la simulación, porque limita la rapidez con la que el robot puede cambiar de acción.
- Los sensores y actuadores son mucho más precisos. Si bien sólo dispone de sónares y *encoders* como sensores, estos son mucho más precisos que los sensores del Rug Warrior. Si a esto añadimos el hecho de que el sistema operativo del robot se encarga de hacer automáticamente las correcciones necesarias para mantener los valores de velocidades indicados por los comandos de actuación, se ha hecho innecesario utilizar ruidos sistemáticos como los empleados en el caso del Rug Warrior.

- El hecho de poder utilizar cualquier ordenador o conjunto de ordenadores para ejecutar los controladores, le dota de mucha mayor potencia y flexibilidad.

Sensores

Como ya se ha comentado, la unidad de Pioneer 2-DX utilizada en este trabajo, sólo dispone de sónares y *encoders* aunque, eso sí, su funcionamiento es mucho mejor que el de los sensores del Rug Warrior.

Sónares

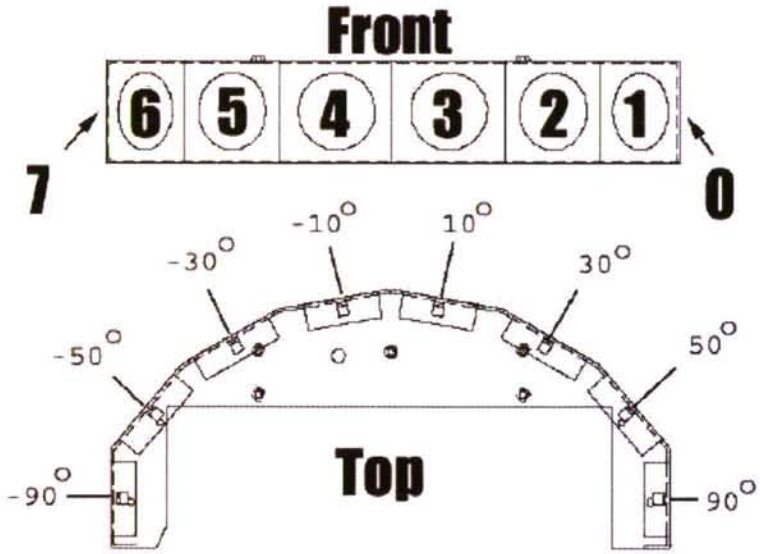


Figura 88: Esquema del array frontal de sónares del Pioneer 2-DX.

El Pioneer 2-DX dispone de dos arrays de sónares de 8 elementos cada uno. Uno de ellos situado en la parte frontal y otro en la parte trasera. Los sónares son de bastante calidad y, además, el sistema operativo del Pioneer 2-DX realiza un postprocesado de sus valores, de forma que los controladores reciben valores muy fiables para objetos situados a distancias inferiores a los 3 metros. Para distancias mayores el sistema operativo devuelve valores aleatorios entre 7 y 8 metros. Es preciso notar que los sónares, por la naturaleza de su propio funcionamiento, no son fiables cuando el ángulo de incidencia con la superficie del objeto es pequeño, ya que le puede llegar al receptor una cantidad de energía inferior a la que le correspondería por la distancia, de forma que el valor que indica el sensor será erróneo. Esto no es habitual, pero sucede en ocasiones con superficies muy lisas y cuando dicho ángulo de incidencia es pequeño. La otra inconveniencia es que existen dos ángulos muertos en los laterales del robot debido a la separación entre el array delantero y el trasero.

Velocidad

Los *encoders* que posee el Pioneer 2-DX son muy fiables excepto a velocidades elevadas. Debido a que a esas velocidades el funcionamiento del robot no es adecuado, porque el sistema operativo usa la información de los *encoders* para tratar de cumplir los comandos recibidos, se ha limitado la velocidad lineal máxima a 20 cm/s y la angular a 40°/s en todos los controladores obtenidos para este robot, velocidades a las que la información de los *encoders* es fiable casi al 100% en superficies no deslizantes.

Actuadores

Los actuadores del Pioneer 2-DX son dos motores acoplados a sendas ruedas como en el caso del Rug Warrior, pero aquí mucho más precisos. Además, como ya se ha comentado, el propio sistema operativo del robot usa la información de los *encoders* para garantizar que los comandos indicados a los actuadores se cumplen. El único problema asociado a los actuadores son los tiempos de aceleración y desacelaración, motivados por el peso del robot, que impiden que éste alcance las velocidades indicadas casi instantáneamente, a diferencia del Rug Warrior, lo cual es una dificultad añadida a la hora de obtener los controladores.

Entorno de simulación: SEVEN

Para comprobar la viabilidad de la metodología descrita en este trabajo se desarrolló SEVEN (*Simulation and EVolution ENvironment*). Inicialmente, SEVEN nació como una adaptación de GENIAL [30] y el *Khepera Simulator* [81], cogiendo del primero la implementación de algoritmo genético y del segundo el simulador del Khepera, adaptado para simular el Rug Warrior, para así tener una aplicación base sobre la que probar la metodología que se estaba desarrollando. Posteriormente, SEVEN fue reescrito desde cero en C++ para facilitar el mantenimiento y la modularidad de un programa que estaba en constante cambio.

SEVEN dispone, a grandes rasgos, de dos módulos: un evolucionador de comportamientos y un simulador. El evolucionador de comportamientos implementa todos los algoritmos que se han comentado en este trabajo y su objetivo es, dado un fichero de configuración en el que se le especifican todos sus parámetros de funcionamiento, obtener el controlador o controladores que implementan un determinado comportamiento para un robot dado. El evolucionador, que puede ser compilado para utilizar hilos *posix* o MPI si así se desea, invoca, a la hora de evaluar los individuos durante la evolución, al otro módulo, el simulador, para comprobar cómo se desenvuelve dicho individuo en el entorno. El simulador, que dispone de un interfaz gráfico que puede ser activado o desactivado, puede ser también invocado directamente por el usuario para ver el comportamiento de un determinado controlador sobre un entorno sin tener que probarlo en el robot real o para diseñar mundos que posteriormente serán usados en la evolución o en la validación.

Configuración

El comportamiento de ambos módulos de SEVEN viene determinado por una serie de parámetros almacenados en un fichero de configuración. Al igual que todos los otros ficheros empleados por SEVEN, éste se almacena en modo texto, para poder ser editado / visualizado manualmente sin necesidad de usar SEVEN. Dado que todos los ficheros se leen / escriben muy esporádicamente, la pérdida de rendimiento respecto a utilizar otra representación es despreciable. El fichero de configuración posee las siguientes opciones:

- Número de robots. La evolución o la simple simulación se lleva a cabo en un entorno (o varios) que puede contener varios robots. En el caso de que haya más de un robot, el primero de ellos es el principal y es para el que, si está en proceso de evolución, se obtendrá el controlador.
- Fichero de mundo. Nombre del fichero que contiene la descripción del mundo en el cual se llevará a cabo la evolución o la simulación. Actualmente, la evolución puede utilizar más de un entorno, pero en ese caso los nombres de los ficheros están especificados en el código fuente.

- Algoritmo evolutivo a utilizar. Puede ser un algoritmo genético, una estrategia evolutiva, el algoritmo mixto entre ambos descrito en este trabajo, o un macroevolutivo.
- Tipo de distribución inicial de la población. Puede ser aleatoria, en cuadrícula o en diagonal generalizada a n dimensiones.
- Probabilidad de cruce. No es relevante en el caso del macroevolutivo.
- Probabilidad de mutación. No es relevante en los casos de estrategia evolutiva ni macroevolutivo.
- Número de razas.
- Poblaciones en cada raza. Este parámetro indica el número de subpoblaciones en las que se divide cada raza y determina, junto con ésta, el número de hilos / procesos que utilizará SEVEN (= número de razas * poblaciones en cada raza).
- Número de generaciones de la evolución.
- Número de generaciones entre dos migraciones locales.
- Número de generaciones entre dos migraciones globales.
- Número de individuos en la población.
- Número de evaluaciones por individuo para promediar su calidad.
- Número de pasos del individuo en cada evaluación.

Los siguientes parámetros estarán tantas veces como robots se hayan especificado en el primer parámetro:

- Tipo de robot. Rug Warrior o Pioneer 2-DX.
- Velocidad máxima en mm/s.
- Velocidad rotacional máxima en grados/s. Sólo es relevante para el Pioneer 2-DX.
- Nombre del fichero que contiene la descripción del controlador utilizado por este robot. En el caso del primer robot, el nombre del fichero determina también el comportamiento para el cual se quiere obtener el controlador.

Entornos

Uno de los parámetros que aparecen en el fichero de configuración de SEVEN es el nombre del fichero que guarda la descripción del entorno simulado en el cual se llevará a cabo la evolución. Los entornos se diseñan desde el interfaz gráfico del simulador, aunque al estar los ficheros en formato texto es posible modificarlos directamente con un editor de texto. Están construidos por paredes y luces, y el suelo está dividido por una cuadrícula imaginaria de forma que cada posición de esa cuadrícula puede tener asociada una cantidad de comida o veneno. El formato del fichero de descripción del entorno es el siguiente:

- Número de paredes.
- Número de luces.
- Dimensiones (mm x mm).
- Lista de las paredes. Cada pared es un segmento de recta del que se guardan las coordenadas de sus extremos y varios parámetros previamente calculados de la ecuación de la recta en varias formas, para acelerar las operaciones aritméticas durante la simulación.
- Lista de las luces. Cada luz es un círculo del que se guardan las coordenadas de su centro y un parámetro que se usa en algunos comportamientos para indicar si la luz es constante o intermitente y en otros comportamientos para indicar si emite radiación positiva o negativa para el robot.
- Mapa de comida. El mundo tiene asociado un mapa que indica para cada punto si hay comida o no en el suelo en ese punto. Asimismo, hay un segundo mapa que indica cuál es la cantidad máxima de comida que puede albergar ese punto del entorno.

Controladores

Otro de los parámetros del fichero de configuración es el nombre del fichero que describe el controlador a evolucionar o utilizar en el proceso de simulación. Este fichero guarda la siguiente información:

- Versión de fichero. Dado que las modificaciones en SEVEN en ocasiones implicaban cambios en el formato de los ficheros de controladores, este campo se añadió para comprobar, a la hora de cargar el controlador, que su formato es el esperado por la aplicación.
- Número total de RNAs que incluye el controlador.
- Número de sensores virtuales.
- Número de módulos de modificación de estado interno.
- Número de RNAs que serán evolucionadas simultáneamente.
- Número total de parámetros a evolucionar (= longitud del cromosoma).

A continuación, en el fichero está guardada información de los módulos que componen el controlador. Primero aparecen, en recorrido in-orden, los módulos (de selección, modulación y actuación) que componen la jerarquía del controlador cuando éste se expresa en forma de árbol, y posteriormente los módulos que se corresponden con sensores virtuales y con módulos de modificación de estado interno. Por cada módulo del controlador habrá la siguiente información:

- Nombre. Si es una RNA este nombre es también el nombre del fichero que contiene la información sobre esa RNA.

- Tipo de módulo: selección, modulador de actuadores, modulador de sensores, actuación, sensor virtual, modificación de estado interno.
- Operación. Indica si el módulo va a ser evolucionado o no.
- Tipo de las entradas. Indica qué sensor está asociado a cada entrada del módulo.
- Tipo de las salidas. Indica qué actuador (o sensor si es un modulador de sensores) está asociado a cada salida del módulo en el caso de los módulos de actuación y modulación.
- Número de nodos descendientes.

Módulos

Por cada módulo del controlador que sea un RNA existirá un fichero con la descripción de dicha RNA. Como ya se ha comentado, la arquitectura es independiente de cómo estén implementados los módulos, pero por decisión en la metodología éstos son RNAs. Sólo en algunos ejemplos, cuando el diseño del módulo era trivial (por ejemplo, los sensores y actuadores virtuales que se han empleado), se utilizaron otro tipo de módulos, pero en ese caso, por su carácter de excepcionalidad, se embebieron directamente en el código fuente de la aplicación y no se desarrolló un tipo determinado de fichero para guardar información sobre su funcionamiento. El fichero correspondiente a un módulo implementado mediante una RNA guarda la siguiente información:

- Versión de fichero.
- Uso de energía por uso de sensor. Indica si el uso de los sensores tiene un coste energético asociado.
- Número de capas. Por cada capa se almacena el tipo y el número de neuronas en esa capa. Se asume interconexión total entre las neuronas de una capa y la siguiente.
- Número de parámetros en la RNA.
- Retardo máximo de una conexión sináptica con retardo variable.
- Valor máximo de un peso.
- Valor de todos los parámetros de la RNA. Estos son: pendiente de las HAN, pesos, retardos, parámetros de las gausianas, bias y pendientes de las sigmoides.

Módulo de simulación

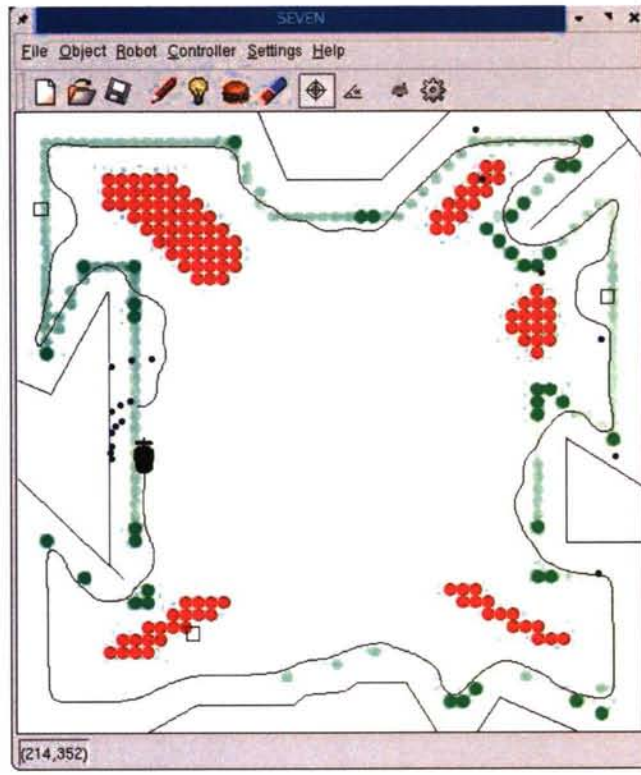


Figura 89: Aspecto visual del módulo simulador de SEVEN.

Como se ha comentado, el simulador fue adaptado inicialmente del *Khepera Simulator*, y se reescribió desde cero cuando el programa fue convertido a C++. El simulador puede ejecutarse con o sin entorno gráfico, estando éste realizado usando las librerías de QT [103] y KDE [65]. Lo primero es útil cuando se quiere diseñar un entorno para ser usado en evolución o simulación, comprobar el funcionamiento de un controlador en el simulador u observar en tiempo real el comportamiento en la evolución de los distintos individuos a medida que son evaluados. Lo segundo es lo habitual a la hora de evolucionar controladores, ya que la visualización del comportamiento de los robots durante la evaluación ralentiza, lógicamente, la evolución. El simulador consta de una clase para el entorno, otra para el robot y numerosas clases para el entorno gráfico. La clase que representa el entorno, o mundo, es la encargada de leer y escribir los ficheros en los que se guarda la información sobre los entornos y contiene información sobre las misiones que pueden llevar a cabo los robots en los entornos, así como objetos que representan a los robots que están presentes en un momento dado en el entorno. La clase que representa el robot contiene información y métodos comunes a todos los tipos de robots que existen en SEVEN y es especializada en clases particulares para cada tipo de

robot con la información y los métodos específicos de ese tipo. Actualmente, existen clases para el Rug Warrior y el Pioneer 2-DX.

En la figura 89 se puede ver una captura de pantalla del simulador. La parte central y mayor de la ventana es el mundo en el cual se desarrolla la simulación. Ahí se pueden ver los elementos constructivos que componen los entornos: las paredes (líneas negras), el robot (en otros comportamientos puede haber más de uno), la comida (las bolas verdes), el veneno (las bolas rojas) y las distancias a las cuales los sónares perciben los objetos (los puntos azules). Los puntos de comida más claros son aquellos por los que el robot ha pasado.

La aplicación permite crear entornos nuevos construidos a partir de paredes, puntos de comida, puntos de veneno y luces. Se pueden cargar controladores y entornos en cualquier momento y el robot se puede mover paso a paso o sin interrupción.

Módulo de evolución

El módulo evolucionador implementa todos los algoritmos comentados en este trabajo. Como se ha dicho, la evolución puede llevarse a cabo visualizando o no las evaluaciones de los individuos, aunque lo habitual es que no se visualicen. Al finalizar cada generación, se escriben en ficheros, uno para cada raza, la población total, la calidad de todos los individuos y la calidad del mejor individuo. También se escribe en otro fichero la calidad del mejor individuo para toda la población y la calidad media de todos los individuos, así como los ficheros correspondientes a las RNAs codificadas en el cromosoma de ese mejor invidivuo. Estos ficheros se escriben tanto de cara a poder extraer estadísticas e información de cómo ha transcurrido el proceso evolutivo al acabar éste, como para poder continuar dicho proceso evolutivo en caso de que éste aborte prematuramente (bien intencionadamente enviando la señal SIGTERM al proceso o bien debido a causas ajenas a la voluntad) o porque se desea continuar la evolución más allá del número de generaciones previsto inicialmente. Si se envía la señal SIGTERM al proceso, éste acaba la generación en curso y luego finaliza su ejecución tras grabar los ficheros correspondientes.

El código fuente se puede compilar con soporte para varios modos de ejecución alternativos:

- Modo monoproceso con soporte gráfico. Este modo es necesario si se quiere utilizar el entorno gráfico, implementado en QT/KDE. Durante la evolución, el entorno gráfico puede ser habilitado/deshabilitado en tiempo de ejecución enviando la señal SIGUSR1 al proceso.
- Con soporte para hilos *posix*. Este modo es el más adecuado para realizar evoluciones en máquinas multiprocesador con memoria compartida. Con la implementación actual, un único proceso/hilo no puede llevar a cabo la evolución de más de una raza, con lo que este modo también debe usarse cuando se quiere emplear más de una raza en una máquina con un único procesador.
- Con soporte para MPI. Este modo es el más adecuado para realizar evoluciones en máquinas multiprocesador con memoria distribuida o *clusters*. Su funcionamiento está comprobado con MPICH y LAM/MPI, las dos implementaciones gratuitas de MPI más difundidas, aunque el primero ha sido probado más exhaustivamente por ser el que estaba instalado en todas las máquinas/*clusters* a los que se tenía acceso. La explicación de ambas paralelizaciones se describió en el apartado 4.4.5.

Bibliografía

[1] Ackley, D. H. and Littman, M. L., "Generalization and Scaling in Reinforcement Learning", *Advances in Neural Information Processing Systems* 2, pp. 550-557, 1990.

[2] ActivMedia Robotics, "Complete Robotics Solutions", *http://www.activmedia.com*.

[3] Agree, P. and Chapman, D., "Pengi: An Implementation of a Theory of Activity", *Proceedings of the 6th Nacional Conference on Artificial Intelligence (AAAI-87)*, pp. 268-272, 1987.

[4] Alba, E. and Troya, J. M., "A Survey of Parallel Distributed Genetic Algorithms", *Complexity* 4, 4, pp. 31-52, 1999.

[5] Alwan, M., Cheung, P. Y. K., Saleh, A. and Obeid, N. E. C., "Combining Goal-Directed, Reactive and Reflexive Navigation in Autonomous Mobile Robots", *Proceedings of the 1996 Australia and New Zealand Conference on Intelligent Information Systems*, pp. 346-349, 1996.

[6] Arbib, M. A., "Schema Theory", *The Encyclopedia of Artificial Intelligence*, pp. 1427-1443, Wiley-Interscience, 1992.

[7] Arkin, R. C., *Behaviour Based Robotics*, MIT Press, 1998.

[8] Baldwin, J. M., "A New Factor in Evolution", *American Naturalist* 30, pp. 441-451, 1896.

[9] Baluja, S., "A Massively Distributed Parallel Genetic Algorithm (mdpGA)", *Technical Report, CMU-CS-92-196R*, 1992.

[10] Beer, R. D. and Gallagher, J. C., "Evolving Dynamical Neural Networks for Adaptive Behaviour", *Adaptive Behaviour* 1, 1, pp. 91-122, 1992.

[11] Braitenberg, V., *Vehicles: Experiments in Synthetic Psychology*, MIT Press, 1984.

[12] Brooks, R., "Achieving Artificial Intelligence through Building Robots", *MIT AI Lab Memo, 898*, 1986.

[13] Brooks, R., "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation* 2, 1, pp. 14-23, 1986.

[14] Brooks, R., "Intelligence without Reason", *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, pp. 569-595, 1991.

[15] Brooks, R., "Intelligence without Representation", *Artificial Intelligence* 47, pp. 139-159, 1991.

[16] Cantú-Paz, E., "A Summary of Research on Parallel Genetic Algorithms", *IlliGAL Report, 95007*, 1995.

[17] Chalmers, D. J., "Subsymbolic Computation and the Chinese Room", *Symbolic*

and Connectionist Paradigms: Closing the Gap, pp. 25-48, Lawrence Erlbaum Associates, 1992.

[18] Cliff, D., Harvey, I. and Husbands, P., "Incremental Evolution of Neural Network Architectures for Adaptive Behaviour", *Technical Report, CSRP256*, 1992.

[19] Cliff, D., Harvey, I. and Husbands, P., "Explorations in Evolutionary Robotics", *Adaptive Behavior* 2, pp. 73-110, 1993.

[20] Colombetti, M., Dorigo, M. and Borghi, G., "Behaviour Analysis and Training - A Methodology for Behaviour Engineering", *IEEE Transactions on Systems, Man and Cybernetics* 3, 26, pp. 365-380, 1996.

[21] Cooper, M. G., "Evolving a Rule-Based Fuzzy Controller", *Simulation* 65, 1, pp. 67-72, 1995.

[22] Dain, R. A., "Developing Mobile Robot Wall-Following Algorithms Using Genetic Programming", *Applied Intelligence*, 8, pp. 33-41, 1998.

[23] Darwin, C. R., *On The Origin of Species by Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life*, Murray, 1859.

[24] De Jong, K. A., *An Analysis of the Behavior of a Class of a Genetic Adaptive Systems*, University of Michigan, 1975.

[25] Dekhil M., Sobh, T. M. and Efros, A. A., "Commanding Sensors and Controlling Indoor Autonomous Mobile Robots", *Conference on Control Applications*, 1996.

[26] Dorigo, M. and Colombetti, M., *Robot Shaping: An Experiment in Behavior Engineering*, MIT Press, 1998.

[27] Duro, R. J. and Santos, J., "Discrete Time Backpropagation for Training Synaptic Delay Based Artificial Neural Networks", *IEEE Transactions on Artificial Neural Networks* 10, 4, pp. 779-789, 1999.

[28] Duro, R. J., Crespo, J. L. and Santos, J., "Training Higher Order Gaussian Synapses", *Lecture Notes in Computer Science* 1606, pp. 537-545, 1999.

[29] Duro, R. J., Santos, J. and Gómez, A., "Synaptic Modulation Based Artificial Neural Networks", *Lecture Notes in Computer Science* 930, pp. 31-36, 1995.

[30] Duro, R. J., Santos, J. and Sarmiento, A., "GENIAL: An Evolutionary Recurrent Neural Network Designer and Trainer", *Lecture Notes in Computer Science* 1105, pp. 295-301, 1996.

[31] Elman, J. L., "Finding Structures in Time", *University of California CRL Technical Report, 8801*, 1988.

[32] Fikes, R. E. and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence* 2, 5, pp. 189-208, 1971.

[33] Floreano, D. and Mondada, F., "Evolution of Homing Navigation in a Real Mobile Robot", *IEEE Transactions on Systems, Man and Cybernetics, Part-B* 26, pp. 396-407, 1996.

[34] Floreano, D. and Mondada, F., "Evolutionary Neurocontrollers for Autonomous Mobile Robots", *Neural Networks*, 11, pp. 1461-1478, 1998.

[35] Fogel, L. J., *On the Organization of Intellect*, University of California, 1964.

[36] Goldberg, D. E., "Simple Genetic Algorithms and the Minimal, Deceptive Problem", *Genetic Algorithms and Simulated Annealing*, pp. 74-88, 1987.

[37] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[38] Goldberg, D. E. and Deb, K., "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms", *Foundations of Genetic Algorithms*, pp. 69-93, 1991.

[39] Goodman, E. D., Punch, W. F. and Lin, S., "Coarse-Grain Parallel Genetic Algorithms: Categorization and New Approach", *Sixth IEEE Symposium on Parallel and Distributed Processing*, pp. 28-37, 1994.

[40] Harnad, S., "The Symbol Grounding Problem", *Physica D*, 42, pp. 335-346, 1990.

[41] Harvey, "Evolutionary Robotics and SAGA: The Case for Hill Crawling and Tournament Selection", *Artificial Life 3: Proceedings of the Santa Fe Conference*, pp. 299-326, 1993.

[42] Harvey, I., "Artificial Evolution and Real Robots", *Proceedings of International Symposium on Artificial Life and Robotics (AROB)*, pp. 138-141, 1996.

[43] Harvey, I., Husbands, P. and Cliff, D., "Issues in Evolutionary Robotics", *Proceedings of the Second International Conference on Simulation of Adaptive Behaviour (SAB92): From Animals to Animats*, pp. 364-373, 1993.

[44] Hinton, G. E., "Learning Distributed Representations of Concepts", *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, pp. 1-12, 1986.

[45] Hinton, G. E. and Nowlan, S. J., "How Learning can Guide Evolution", *Complex Systems* 1, pp. 495-502, 1987.

[46] Hoffmann, F. and Pfister, G., "Automatic Design of Hierarchical Fuzzy Controllers Using Genetic Algorithms", *Proceedings of the Second European Congress on Intelligent Techniques and Soft Computing (EUFIT 94)*, pp. 1516-1522, 1994.

[47] Holland, J., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[48] Holland, J., "Adaptive Algorithms for Discovering and Using General Patterns in Growing Knowledge Bases", *International Journal of Policy Analysis and Information*

Systems 4, 2, pp. 217-240, 1980.

[49] Husbands, P., Harvey, I., Cliff, D. and Miller, G, "The Use of Genetic Algorithms for the Development of Sensorimotor Control Systems", *From Perception to Action*, pp. 110-121, IEEE Computer Society Press, 1994.

[50] Husbands, P., Smith, T., O'Shea, M., Jakobi, N., Anderson, J. and Philippides, A., "Brains, Gases and Robots", *Perspectives Neural Comp.* 2, pp. 51-63, 1998.

[51] Ishiguro, A., Otsu, K., Fujii, A. and Uchikawa, Y., "Evolving an Adaptive Controller for a Legged-Robot with Dynamically-Rearranging Neural Networks", *Proc. Supp. 6th Int. Conf. on Simulation of Adaptive Behavior*, pp. 235-244, 2000.

[52] Jablonski, J. and Posey, J., "Robotics Terminology", *Handbook of Industrial Robotics*, pp. 1271-1303, 1985.

[53] Jakobi, N., "Half-Baked, Ad-Hoc and Noisy Minimal Simulations for Evolutionary Robotics", *Proceedings of the 4th European Conference on Artificial Life (ECAL 97)*, pp. 348-357, 1997.

[54] Jakobi, N., "Evolutionary Robotics and the Radical Envelope of Noise Hypothesis", *Adaptive Behavior* 6, 2, pp. 325-368, 1997.

[55] Jakobi, N., Husbands, P. and Harvey, I., "Noise and the Reality Gap: the Use of Simulation in Evolutionary Robotics", *Lecture Notes in Artificial Intelligence*, pp. 704-720, 1995.

[56] Janson, D. J. and Frenzel, J. F., "Training Product Unit Neural Networks with Genetic Algorithms", *IEEE Expert* 8, 5, pp. 26-32, 1993.

[57] Janusz, B. and Riedmiller, M., "Self-learning Neural Control of a Mobile Robot", *Proceedings of IEEE International Conference on Neural Networks*, pp. 2358-2363, 1995.

[58] Jones, J. L., *The Mobile Robot Assembly Guide (with Interactive C Manual)*, A. K. Peters Ltd., 1995.

[59] Jones, J. L. and Flynn, A. M., *Mobile Robots: Inspiration to Implementation*, A. K. Peters Ltd., 1993.

[60] Jordan, M. I., "Attractor Dynamics and Parallelism in a Connectionist Sequential Machine", *Proceedings of the 1986 Cognitive Science Conference*, pp. 531-546, 1986.

[61] Jung, D. and Zelinsky, A., "Whisker Based Mobile Robot Navigation", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 96*, pp. 497, 1996.

[62] Kaelbling L., "An Architecture for Intelligent Reactive Systems", *Proceedings of the 1986 Workshop Reasoning about Actions and Plans*, pp. 395-410, 1987.

[63] Kodjabachian, J. and Meyer, J. A., "Evolution and Development of Control Architectures in Animats", *Robotics and Autonomous Systems* 16, pp. 161-182, 1995.

[64] Koza, J. R., "Hierarchical Genetic Algorithms Operating on Populations of Computer Programs", *Proc. 11th Int Joint Conf on Artificial Intelligence*, pp. 768-774, 1989.

[65] KDE Project, "KDE: A Powerful Open Source Graphical Desktop Environment for Unix Workstations", *http://www.kde.org*.

[66] Lund, H. H., "Evolving Robot Control Systems", *Proceedings of the First Nordic Workshop on Genetic Algorithms and their Applications*, pp. 85-95, 1995.

[67] Lund, H. H. and Hallam, J., "Sufficient Neurocontrollers can be Surprisingly Simple", *Department of Artificial Intelligence (University of Edinburg) Research Paper, 824*, 1996.

[68] Lund, H. H. and Miglino, O., "From Simulated to Real Robots", *Proceedings of IEEE 3rd International Conference on Evolutionary Computation*, pp. 362-365, 1996.

[69] Lund, H. H., Miglino, O., Pagliarini, L., Billard, A. and Ijspeert, A., "Evolutionary Robotics - A Children's Game", *Proceedings of IEEE 5th International Conference on Evolutionary Computation*, 1998.

[70] Maes, P., "Bottom-up Mechanism for Behaviour Selection in an Artificial Creature", *Proceedings of the 1st International Conference on Simulation of Adaptive Behaviour (SAB90)*, pp. 238-246, 1990.

[71] Maes, P., "Behavior-Based Artificial Intelligence", *From Animals to Animats 2: Proceedings of the 2nd International Conference on the Simulation of Adaptive Behavior*, pp. 2-10, 1993.

[72] Marín, J. and Solé, R. V., "Macroevolutionary Algorithms: A New Optimization Method on Fitness Landscapes", *IEEE Transactions on Evolutionary Computation* 3, 4, pp. 272-286, 1999.

[73] Mataric, M. J., "Integration of Representation into Goal Driven Behaviour Based Robotics", *IEEE Transactions on Robotics and Automation* 8, 3, pp. 304-312, 1992.

[74] Matellán, V., Fernández, C. and Molina, J. M., "Genetic Learning of Fuzzy Reactive Controllers", *Robotics and Autonomous Systems* 25, pp. 33-41, 1998.

[75] Meeden, L., "An Incremental Approach to Developing Intelligent Neural Network Controllers for Robots", *IEEE Transactions on Systems, Man and Cybernetics* 3, 26, pp. 474-485, 1996.

[76] Menczer, F., Street, W. N. and Degeratu, M., "Evolving Heterogeneous Neural Agents by Local Selection", *Advances in the Evolutionary Synthesis of Intelligent Agents*, pp. 337-365, MIT Press, 2001.

[77] Message Passing Interface Forum, "The Message Passing Interface (MPI) Standard", *http://www.mpi-forum.org*.

[78] Meyer, J-A., Doncieux, S., Filliat, D. and Guillot, A., "Evolutionary Approaches to Neural Control of Rolling, Walking, Swimming and Flying Animats or Robots", R. J. Duro, J. Santos and M. Graña (Eds.), *Biologically Inspired Robot Behavior Engineering*, pp. 1-43, Physica-Verlag, 2002.

[79] Miglino, O., Lund, H. H. and Nolfi, S., "Evolving Mobile Robots in Simulated and Real Environments", *Artificial Life* 2, pp. 417-434, 1995.

[80] Miglino, O., Nafasi, K. and Taylor, C., "Selection for Wandering Behavior in a Small Robot", *Artificial Life* 2, pp. 101-116, 1995.

[81] Mitchel, O., "Khepera Simulator 2.0", *http://wwwi3s.unice.fr/om7khep-sim.html*.

[82] Mondada, F. and Floreano, D., "Evolution of Neural Control Structures: Some Experiments on Mobile Robots", *Robotics and Autonomous Systems* 16, pp. 183-195, 1995.

[83] Nilsson, N., "Action Networks", *Proceedings of the Rochester Planning Workshop: From Formal Systems to Practical Systems*, 1989.

[84] Nolfi, S., "Using Emergent Modularity to Develop Control Systems for Mobile Robots", *Adaptive Behaviour* 5, 3/4, pp. 343-363, 1997.

[85] Nolfi, S., "Evolving Non-Trivial Behaviours on Real Robots: A Garbage Collector Robot", *Robotics and Autonomous Systems*, 22, pp. 187-198, 1997.

[86] Nolfi, S. and Parisi, D., "Learning to Adapt to Changing Environments in Evolving Neural Networks", *Adaptive Behavior* 5, 1, pp. 75-98, 1997.

[87] Nolfi, S., Elman, J. and Parisi, D., "Learning and Evolution in Neural Networks", *Adaptive Behavior* 1, pp. 5-28, 1994.

[88] Nolfi, S., Floreano, D., Miglino, O. and Mondada, F., "How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics", *Proceedings of Fourth International Conference on Artificial Life*, pp. 190-197, 1994.

[89] Pasemann, F., Steinmetz, U., Hülse, M. and Lara, B., "Robot Control and the Evolution of Modular Neurodynamics", *Theory in Biosciences*, 120, pp. 311-326, 2001.

[90] Pfeifer, R. and Scheier, C., *Understanding Intelligence*, MIT Press, 1999.

[91] Plaut, D. S., Nowlan, S. and Hinton, G. E., "Experiments of Learning by Backpropagation", *Tech Rep, CMU-CS-86-126*, 1986.

[92] Rechenberg, I., "Cybernetic Solution Path of an Experimental Problem", *Royal Aircraft Establishment Library Translation, 1122*, 1965.

[93] Reed, R., "Pruning Algorithms - A Survey", *IEEE Transactions on Neural Networks* 4, 5, pp. 740-747, 1993.

[94] Reynolds, C. W., "Evolution of Corridor Following Behavior in a Noisy World", *From Animals to Animats* 3, pp. 402-410, 1994.

[95] Rosenschein, S. and Kaelbling, L., "The Synthesis of Digital Machines with Provable Epistemic Properties", *Proceedings of the 1st Conference on Theoretical Aspects of Reasoning about Knowledge*, pp. 83-98, 1986.

[96] Saffiotti, A., "The Uses of Fuzzy Logic in Autonomous Robot Navigation", *Soft Computing* 1, pp. 180-197, 1997.

[97] Salomon, R., "The Evolution of Different Neuronal Control Structures for Autonomous Agents", *Robotics and Autonomous Systems*, 22, pp. 199-213, 1997.

[98] Sanborn, J. and Hendler, J., "A Model of Reaction for Planning in Dynamic Environments", *International Journal for Artificial Intelligence in Engineering* 3, 2, pp. 95-102, 1988.

[99] Santos, J. and Duro, R. J., "Evolving Neural Controllers for Temporally Dependent Behaviors in Autonomous Robots", *Lecture Notes in Artificial Intelligence* 1416, pp. 319-328, 1998.

[100] Schewefel, H. P., *Evolutionsstrategie und Numerische Optimierung*, Technische Universität Berlin, 1975.

[101] Stracuzzi, D. J., "Some Methods for the Parallelization of Genetic Algorithms", *Machine Learning Lab. (Clarkson University) Report*, 1998.

[102] Takens, F., "On the Numerical Determination of the Dimension of an Attractor", *Lectures Notes in Mathematics* 898, pp. 366-381, 1981.

[103] Trolltech, "Qt: A Multiplatform C++ Application Development Framework", *http://www.trolltech.com/products/qt*.

[104] Urzelai, J., Floreano, D., Dorigo, M. and Colombetti, M., "Incremental Robot Shaping", *Connection Science Journal* 10, 384, pp. 341-360, 1998.

[105] Walter, W. G., "An Imitation of Life", *Scientific American* 5, 182, pp. 42-45, 1950.

[106] Walter, W. G., *The Living Brain*, W. W. Norton, 1953.

[107] Whitley, D., "The GENITOR Algorithm and Selection Pressure: Why Rank-based Allocation of Reproductive Trials is Best", *Proc. 3rd Int. Conf. on Genetic Algorithms*, pp. 116-121, 1989.

[108] Wiener, N., *Cibernetics, or Control and Communication in the Animal and the Machine*, Wiley, 1948.

GSA

Grupo de Sistemas Autónomos

En esta Tesis Doctoral se presenta una metodología y un conjunto asociado de herramientas y procedimientos que permiten obtener de forma estructurada, incremental y repetible sistemas de control para robots autónomos que han de realizar tareas en entornos dinámicos y no estructurados.

Este proceso de obtención de los sistemas de control, o controladores, se realiza en entornos simulados utilizando algoritmos evolutivos y reduciendo la intervención humana al mínimo, de tal forma que el diseñador sólo debe definir los entornos en los cuales se llevará a cabo el proceso evolutivo y un modelo energético de evaluación de la calidad, en el que se especifican los cambios que en el nivel de energía del robot se producen ante determinadas interacciones de éste con el entorno. Se estudian las implicaciones y posibilidades del uso de los métodos evolutivos, se usan controladores neuronales y se incorporan los mecanismos necesarios que garantizan el correcto funcionamiento en robots reales, en los cuales se comprueban los controladores obtenidos.

Se describe también una arquitectura que soporta la construcción estructurada de sistemas complejos manteniendo la minimización del papel del diseñador y maximizando la reutilización de controladores, para un mismo robot o para robots con morfologías diferentes, permitiendo su obtención de modo incremental a partir de controladores simples que se interconectan mediante mecanismos de selección o de modulación. Todo ello nos permite contemplar procesos de generación gradual y automática de arquitecturas de control para robots autónomos.